FLAM ASSIGNMENT-EXPLANATION

Assignment for Research and Development / AI

VELCHURI SARVAN

velchurisarvan6@gmail.com

Problem Statement

Here a dataset containing several (x,y) coordinate points were given. These points are known to lie on a curve, but the exact mathematical parameters of the curve are unknown. The curve itself is defined by the following parametric equations:

$$x = (t * \cos(\theta) - e^{M|t|} \cdot \sin(0.3t)\sin(\theta) + X)$$

$$y = (42 + t * \sin(\theta) + e^{M|t|} \cdot \sin(0.3t)\cos(\theta))$$

In these equations, the parameter t behaves like a time variable and is given to lie within the range:

$$6 < t < 60$$

The three variables θ, M, and X are unknown and must be determined. Their possible ranges were also provided:

$$0° < \theta < 50°$$
$$-0.05 < M < 0.05$$
$$0 < X < 100$$

The objective was to determine the numerical values of θ, M, and Xsuch that the resulting curve best matches (fits) the given dataset of (x, y) points.

APPROACH:

A structured approach has been followed in order to determine the unknown parameters θ, M, and X. Since the dataset contained only the (x, y)points and not the corresponding $t$values, the first step was to generate $t$values uniformly in the range specified in the problem statement. It was stated that $t$varies between 6 and 60, therefore we need to assume that the points in the dataset are uniformly sampled along that interval and generated an equally spaced sequence of $t$values using numpy.linspace().

Once t was established, a function was created that represents the given mathematical model. This function takes θ, M, and Xas inputs and returns the $x(t)$and $y(t)$values generated from the equation. I implemented the function directly from the equation provided:

$$x = (t * \cos(\theta) - e^{M|t|} \cdot \sin(0.3t)\sin(\theta) + X)$$
$$y = (42 + t * \sin(\theta) + e^{M|t|} \cdot \sin(0.3t)\cos(\theta))$$

The next step was to compare the curve generated by the model with the actual data points from the CSV file. For this comparison, **L1 distance** has been used, which is the sum of distances between each actual point and its corresponding predicted point:

$$L1 = \sum \sqrt{(x_{pred} - x_{obs})^2 + (y_{pred} - y_{obs})^2}$$

The task was to minimize this L1 distance. In simple terms, we need to adjust θ, M, and Xrepeatedly until the curve produced by the equation matched the dataset as closely as possible.

To automate this parameter search, **numerical optimization** has been used. Instead of manually testing values, I used the minimize() function from the SciPy library passing three important things to the optimizer:

1. **The objective function** (the L1 distance equation),
2. **Initial guesses** for the parameters, and

3. **Bounds** (the allowed ranges provided in the question).

The optimizer then explored different values of θ, M, and Xwithin the permitted ranges and updated them iteratively until it reached a combination that resulted in the lowest error.

Finally, after the optimizer converged, It extracted the best-fitting parameter values and substituted them back into the parametric equation. And then plotted both the actual datapoints (from the CSV) and the generated curve on the same graph to visually verify that the fitted curve aligns well with the given dataset.

The output of the optimization returned the following final parameters:

$$\theta = 0.5241 \text{ radians } (30.029348^\circ)$$
$$M = -0.0045502$$
$$X = 55.329186$$

When these parameters were substituted back into the original equation, the curve generated by the function matched the dataset with a very low L1 distance, meaning the fit was successful.

Code Explanation:

- I started by importing the required libraries:
  numpy for mathematical operations,
  pandas for handling the CSV file,
  math for angle conversions and basic math operations like (sqrt),
  minimize from scipy.optimize for optimization,
  and matplotlib.pyplot for visualization.
- Next, I loaded the dataset using Pandas: df = pd.read_csv(...)
- This allowed me to extract the x and y values from the CSV into variables x_observed and y_observed. And the length of the dataset was stored in N.
- Since the problem statement mentions that the parameter **t lies between 6 and 60**, I generated a sequence of t values uniformly using: t = np.linspace(6, 60, N)

- This assumes that the data points are measured uniformly along the curve.
- I then created a function called compute_curve() that directly implements the given curve equation. Inside this function, I extracted the three unknown parameters (θ, M, X) from the input argument.
- To keep the curve equation easy and readable, I computed intermediate values:
    - exp(M * |t|) using NumPy's exponential function
    - sin(0.3 * t) using NumPy's sine function
- Using these, I reconstructed the equation for x(t) and y(t):
  [x = t\cos(\theta) - e^{M|t|} \cdot \sin(0.3t)\sin(\theta) + X]
  [y = 42 + t\sin(\theta) + e^{M|t|} \cdot \sin(0.3t)\cos(\theta)]
- After building the model, I created another function called distance_objective().
  This function calculates the **L1 distance (error)** between:
    - the predicted x,y values (generated by model)
    - the actual x,y values from the dataset
- The optimizer's job is to **minimize this error**.
  Lower error means the curve fits the data better.
- Then, I called minimize() from SciPy: result = minimize(...)
- Then i passed:
    - our objective function,
    - an initial guess for the parameters [0.5, 0, 50]
    - the method "Powell" (does not need derivative information)
    - the **bounds** for each parameter (as given in the assignment)
- The optimizer ran multiple iterations internally, trying different values of θ, M, and X until it found the combination of parameters that produced the least error.
- Once optimization was completed, I extracted the final estimated values:
- theta_optimized_value, M_optimized_value, X_optimized_value = result.x
- Using these optimized parameters, I generated the final predicted points using the model again.

- To visually verify the result, I plotted:
  - The actual dataset (scatter plot)
  - The fitted curve (line plot)
- If the curve passes smoothly through or very close to the observed data points, that confirms our optimization worked correctly.
- Finally, I printed the optimized values and the total L1 error. These values represent the best fit for the given dataset within the allowed parameter constraints.

Final Output:

Parameter	Final estimated value
θ	0.5241 radians (≈ 30.03°)
M	−0.0045502
X	55.329186

Mathematical Formulas Used:

1. Given Parametric Curve Equation

The shape of the curve is defined using a parametric form, where both x and y depend on a parameter t:

$$x = (t * \cos(\theta) - e^{M|t|} \cdot \sin(0.3t)\sin(\theta) + X)$$
$$y = (42 + t * \sin(\theta) + e^{M|t|} \cdot \sin(0.3t)\cos(\theta))$$

Where:

- $t$is the varying parameter (similar to time)
- θ, M, and $X$are unknown constants to be determined
- $e^{M|t|}$introduces a growing/decaying effect depending on the sign of M

- sin $(0.3t)$introduces a wave-like oscillation in the curve

2. Conversion from radians to degrees (when needed)

The angle $\theta$is found in **radians**, but to convert it to degrees:

$$\theta_{degrees} = \theta_{radians} \times \frac{180}{\pi}$$

3. L1 Distance (Error Function Used for Optimization)

To determine the best values of θ, M, X, we need to compare:

- Predicted points (x_{pred}, y_{pred})
- Actual observed points (x_{obs}, y_{obs})

Distance formula between two points:

$$d_i = \sqrt{(x_{pred_i} - x_{obs_i})^2 + (y_{pred_i} - y_{obs_i})^2}$$

Sum of distances over the entire dataset:

$$L1 = \sum_{i=1}^{N} \quad d_i$$

This value is minimized using numerical optimization.

4. Final Solved Equation (after optimizing parameters)

After optimization, the best-fit values obtained were:

$$\theta = 0.5241 \text{ radians} (\approx 30.03^\circ)$$
$$M = -0.0045502$$
$$X = 55.329186$$

Substituting the optimized values back into the model:

$$x = (t * \cos(0.5241) - e^{-0.0045502|t|} \cdot \sin(0.3t)\sin(0.5241) + 55.329186)$$
$$y = (42 + t * \sin(0.5241) + e^{-0.0045502|t|} \cdot \sin(0.3t)\cos(0.5241))$$

RESULT:

x = \left(t*\cos(0.5241) - e^{-0.0045502\left|t\right|} \cdot \sin(0.3t)\sin(0.5241) + 55.329186 \right)

y = \left(42 + t*\sin(0.5241) + e^{-0.0045502\left|t\right|} \cdot \sin(0.3t)\cos(0.5241) \right)