Hiya Nachnani
PES2UG22CS225
PES University

Objective

The goal of this project is to estimate three unknown parameters — **θ (theta)**, **M**, and **X** — in a given parametric curve defined by the following equations:

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

Given a set of observed data points ((x, y)), the objective is to find the parameter values that minimize the difference between the predicted and observed points.

Step 1: Data Loading

The dataset was provided in a CSV file containing columns `x` and `y`, which represent the observed coordinates of points.
These points correspond to a parametric variable `t`, which is assumed to increase linearly. Therefore, a uniformly spaced array `t_vals` was generated using NumPy's `linspace()` function.

If actual `t` values were provided, the code could easily be modified to use them directly instead of generating.

```
df = pd.read_csv("xy_data.csv")
x_vals = df['x'].values
y_vals = df['y'].values
t_vals = np.linspace(6, 60, len(x_vals))
```

Step 2: Defining the Mathematical Model

A Python function `predict(params, t)` was defined to represent the given mathematical equations.
This function takes the parameters `(θ, M, X)` and a list of `t` values, then computes the corresponding `x` and `y` predictions.

- `θ` (theta) represents a rotation angle in degrees.
- `M` controls the exponential scaling term in the oscillation.
- `X` represents a horizontal translation of the entire curve.

```
def predict(params, t):
    theta_deg, M, X = params
    theta = np.deg2rad(theta_deg)
    term = np.exp(M * np.abs(t)) * np.sin(0.3 * t)
    x_pred = t * np.cos(theta) - term * np.sin(theta) + X
    y_pred = 42 + t * np.sin(theta) + term * np.cos(theta)
    return x_pred, y_pred
```

Step 3: Loss Function Formulation

To measure how well a given set of parameters fits the data, a **loss function** was defined.
This function computes the **sum of squared differences** between the observed and predicted coordinates:

$$
L(\theta, M, X) = \sum_{i=1}^{n} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]
$$

This is equivalent to minimizing the total squared error between observed and modeled points.

```
def loss(params):
    x_pred, y_pred = predict(params, t_vals)
    return np.sum((x_vals - x_pred)**2 + (y_vals - y_pred)**2)
```

Step 4: Initial Parameter Estimation

Before optimization, reasonable initial guesses for the parameters were computed:

- **θ (theta_init_deg)** was estimated from the linear slope of `(y - 42)` vs `x`.
  Using polynomial fitting, (\tan(\theta) \approx \frac{y - 42}{x}), so θ can be estimated from the slope.

- **M_init** was set to 0 (no exponential scaling initially).

- **X_init** was estimated as the mean horizontal offset between `x` and `t * cos(θ)`.

Bounds were also set for each parameter to prevent invalid solutions:

- (0.0001 \leq \theta \leq 50^\circ)

- (-0.05 \leq M \leq 0.05)

- (0 \leq X \leq 100)

```
coeffs = np.polyfit(x_vals, y_vals - 42, 1)
theta_init_deg = np.rad2deg(np.arctan(coeffs[0]))
M_init = 0.0
X_init = np.mean(x_vals - t_vals * np.cos(np.deg2rad(theta_init_deg)))
initial = [theta_init_deg, M_init, X_init]
```

Step 5: Optimization Using L-BFGS-B

The **`scipy.optimize.minimize`** function was used with the **L-BFGS-B algorithm**, which efficiently handles bounded optimization problems.

The optimizer iteratively updates the parameter values to minimize the loss function, thereby finding the best-fit curve.

```
best_res = minimize(loss, initial, bounds=bounds, method='L-BFGS-B',
options={'maxiter':10000})
```

The result contains the optimized values of θ, M, and X, as well as the minimum loss achieved.

Step 6: Evaluation Metrics

After fitting, two evaluation metrics were computed:

- **Sum of Squared Error (SSE)**: Total squared distance between observed and predicted points.

- **L1 Distance (Sum and Mean)**: Average absolute deviation of points.

These metrics help assess the goodness of fit.
Lower values indicate a more accurate model.

```
x_pred, y_pred = predict(best_res.x, t_vals)
dists = np.sqrt((x_vals - x_pred)**2 + (y_vals - y_pred)**2)
L1_sum = np.sum(dists)
L1_mean = np.mean(dists)
```

Step 7: Result Presentation

The optimized parameters were printed clearly, along with a LaTeX-compatible equation for easy inclusion in reports.
An example result might look like:

```
Theta (degrees): 29.582815
M: -0.050000
X: 55.013607
Sum of squared error: 771686.8924
L1 distance mean: 18.8456
```

Step 8: Saving Outputs

Two files were generated:

1. **param_fit_results.csv** — containing observed, predicted, and residual values for each point.

2. **fitted_equation.txt** — containing parameter estimates, final fitted equations, and LaTeX string for documentation.

This ensures full reproducibility of the analysis.

Step 9: Visualization

The observed data points and the predicted curve were plotted using **Matplotlib**.
The red line represents the fitted model, and blue scatter points represent observed data.
This visualization provides an intuitive check of model accuracy.

```
plt.scatter(x_vals, y_vals, s=15, label="Observed", alpha=0.7)
plt.plot(x_pred, y_pred, color='red', label="Predicted Curve", linewidth=2)
plt.title("Fitted Parametric Curve")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```

Step 10: Interpretation of Results

The fitted parameters provide insights into the behavior of the system:

- **θ** determines the overall rotation of the curve.
- **M** controls how strongly the oscillation term grows or decays with |t|.
- **X** shifts the curve horizontally.

A moderate θ (around 30°) and slightly negative M (around -0.05) indicate a gentle tilt and a decaying oscillation, which aligns with many physical or geometric patterns in such data.

Conclusion

This process demonstrates a complete parameter estimation pipeline:

1. Data preparation
2. Model definition
3. Loss function design
4. Parameter initialization
5. Optimization
6. Evaluation

7. Result saving and visualization

The approach effectively combines mathematical modeling, numerical optimization, and data analysis to recover underlying parameters from observed data — achieving both interpretability and reproducibility.