Estimating Unknown Parameters for a Parametric Curve

1. The Problem: What We Had to Do

The assignment was a **parameter estimation** problem. We were given:

1. A set of parametric equations for $x(t)$ and $y(t)$ that define a curve.
2. These equations had three **unknown variables**: θ, M, and X.
3. A CSV file (`xy_data.csv`) containing 1,500 x and y points that lie on the *actual* curve.
4. The ranges (or "bounds") for the unknown variables (e.g., $0 < \theta < 50$).

Our goal was to find the precise numerical values for θ, M, and X that make the parametric equations generate a curve that **best fits** the provided data points.

2. The Strategy: How We Decided to Solve It

We can't guess the values manually. The strategy was to use a computational approach to find the best-fitting parameters. This process involves two key parts:

1. **Define a "Loss Function":** We needed a way to measure how "wrong" any given guess for (θ, M, X) is. The assignment specified using the **L1 distance** (also known as Mean Absolute Error). Our loss function would take a guess, generate the 1,500 x,y points from that guess, and compare them to the 1,500 *real* x,y points from the CSV. It then returns a single "error score." A lower score is better.
2. **Use an "Optimizer":** We would use an optimization algorithm from a scientific library. This is like a "smart robot" that we give our Loss Function to. The robot's job is to automatically and rapidly test thousands of combinations of θ, M, and X (within their allowed ranges) until it finds the single combination that produces the **lowest possible error score**.

3. The Implementation: Step-by-Step Process

We used the **Python** programming language, along with the `Pandas`, `NumPy`, and `SciPy` libraries to execute this strategy.

Step 1: Data Loading and Preparation

- We used `pandas` to load the `xy_data.csv` file.
- We immediately noticed the file only contained 'x' and 'y' columns. The 't' parameter was missing.
- Based on the assignment ("parameter 't' has range: $6 < t < 60$" and "uniformly sampled"), we deduced that we had to generate the 't' values ourselves.
- Using `numpy.linspace(6, 60, 1500)`, we created a `t_data` array of 1,500 points, perfectly spaced between 6 and 60, to match the 1,500 x,y points.

Step 2: Defining the Model (The `predict_curve` function)

- We created a Python function that acts as our "prediction machine."
- This function takes a set of parameters (θ, M, X) and our `t_data` array as input.
- It implements the exact math equations from the assignment to calculate the predicted x and y values.
- **Key Detail:** The θ range was in **degrees**, but Python's `np.sin` and `np.cos` functions require **radians**. Our function correctly used `np.deg2rad()` to convert θ before any calculations, ensuring mathematical accuracy.

Step 3: Defining the Loss Function (The `calculate_l1_loss` function)

- This function is the "scorekeeper" that our optimizer tries to minimize.
- It takes a *guess* for the parameters (e.g., `[25, 0.01, 50]`) as input.
- Inside, it does the following:
  - Calls `predict_curve` with this guess to get the *predicted* x and y points.
  - Calculates the L1 distance: `abs(x_actual - x_predicted)` and `abs(y_actual - y_predicted)`.
  - Calculates the **mean** of all these errors and returns the total error score.

Step 4: Running the Optimization

- We used the `scipy.optimize.differential_evolution` algorithm. This is a powerful global optimizer that is very effective at finding the best solution, even in complex problems, and respects the `bounds` we provide.
- We "told" the optimizer:
  - Minimize our `calculate_l1_loss` function.
  - Use the `bounds` we specified (e.g., $0 < \theta < 50$).

    - Pass in our `t_data`, `x_data`, and `y_data` as the "ground truth" to compare against.
- The optimizer then ran for about a minute, testing many combinations until it "converged" on the set of parameters that produced the lowest possible L1 error.

4. The Results

The link to Desmos: https://www.desmos.com/calculator/rutt8njcln

The optimization was successful. The script printed the final, optimal values for the three unknown variables: