It looks like the file x.txt wasn't uploaded yet. Could you upload it so I can run the optimization using your data?

Once uploaded, I'll:

* Read the data into a list x
* Optimize the parameters a and k to **minimize** the objective:

**Optimization Problem**

**Objective function (cost)**:

MAE(a,k)=1N∑i=1N∣x^i−xi∣+1a\text{MAE}(a, k) = \frac{1}{N} \sum\_{i=1}^{N} |\hat{x}\_i - x\_i| + \frac{1}{a}MAE(a,k)=N1​i=1∑N​∣x^i​−xi​∣+a1​

where x^i\hat{x}\_ix^i​ is the predicted value using a sliding window of size kkk

**Decision variables**:

* + a∈(0,1)a \in (0, 1)a∈(0,1): Sensitivity factor (continuous)
  + k∈Z,k>3k \in \mathbb{Z}, k > 3k∈Z,k>3: Sliding window size (integer)

**Constraints**:

* + 0<a<10 < a < 10<a<1
  + k>3k > 3k>3

A math problem with black text

AI-generated content may be incorrect.

**How Differential Evolution Works (Intuitively)**

It’s an evolutionary algorithm — inspired by biological evolution and natural selection. Here's a breakdown of its steps:

**1. Population Initialization**

* Instead of starting with a single guess (like minimize() does), it starts with a **population** of candidate solutions.
* Each candidate is a pair: [a, k] (floating-point values within your bounds).

**2. Mutation**

* For each candidate (called a "target"), the algorithm picks **three other distinct candidates** from the population.
* It creates a **mutant vector** by combining those three, using a formula like:

mutant=x1+F⋅(x2−x3)\text{mutant} = x\_1 + F \cdot (x\_2 - x\_3)mutant=x1​+F⋅(x2​−x3​)

where FFF is a mutation factor (e.g. 0.5–1.0).

**3. Crossover**

* The mutant is mixed with the original candidate to create a **trial solution**.

**4. Selection**

* If the trial solution performs **better** than the original candidate (i.e., lower error), it replaces it.
* Otherwise, the original candidate stays.

**5. Repeat**

* This process is repeated for **many generations**, gradually improving the population until convergence.

**✅ Why It Worked Better Than minimize()**

* minimize() is **local**: it starts from one point (e.g., k=5) and looks for a local improvement — it can get stuck.
* differential\_evolution is **global**: it explores the entire search space and avoids getting stuck in local minima.
* Also, it's robust with **mixed variable types** (like a continuous and k integer-rounded).

**🔧 Important Parameters**

* mutation: controls exploration vs exploitation
* popsize: number of candidate solutions
* maxiter: how many generations to run
* strategy: how mutation is performed (e.g., 'best1bin' is commonly effective)

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a white page

AI-generated content may be incorrect.