### ****Algorithm: Z-Score Based Anomaly Detection****

#### **What is Z-Score?**

A **Z-score** measures how far a data point is from the **mean**, expressed in terms of the number of **standard deviations**.

Formula:  
Z= (X−μ)​/ σ  
Where:

* + X = New data point
  + μ = Mean of the sliding window
  + σ = Standard deviation of the sliding window

If the absolute value of the **Z-score** exceeds a specified **threshold** (in this case, 2.5), the point is considered an **anomaly**.

### ****Why Use Z-Score for Anomaly Detection?****

1. **Effective for Normally Distributed Data:**
   * Z-score works well when the data follows a **Gaussian distribution** (bell curve).
   * It helps to detect **outliers** that deviate significantly from the mean.
2. **Lightweight & Fast:**
   * Z-score calculations involve **basic statistics** (mean and standard deviation), making it suitable for **real-time streaming** with minimal computational overhead.
3. **Adaptability with Sliding Window:**
   * The use of a **sliding window** ensures that the anomaly detection adjusts dynamically to recent changes in the data, preventing outdated patterns from influencing results.

### ****Effectiveness of Z-Score Algorithm****

* **Strengths:**
  1. **Simple and easy to implement**: Uses basic statistical measures (mean, standard deviation).
  2. **Real-time capabilities**: The algorithm performs efficiently in streaming scenarios.
  3. **Adaptive behaviour**: With a sliding window, the mean and standard deviation are constantly updated to reflect the latest data.
* **Limitations:**
  1. **Sensitive to high variance**: If the data stream has high volatility, the algorithm may detect false positives.
  2. **Not suitable for non-Gaussian data**: For skewed or multi-modal distributions, other methods like Isolation Forests or Auto encoders may work better.
  3. **Edge cases**: If the sliding window is too small, anomalies might not be detected reliably.

**Error Handling in the Code**

1. **Handling Division by Zero:**
   * When calculating the **Z-score**, the code ensures that the **standard deviation** (σ) is never zero.

Python code

std\_dev = np.std(self.data\_window) or 1e-6 # Prevent division by zero

* + If std\_dev is 0 (when all values in the window are identical), it is replaced with 1e-6 to avoid dividing by zero.

1. **Managing Insufficient Data:**
   * Anomaly detection only starts once the **sliding window** has enough data points. This ensures that **statistical calculations** (mean, standard deviation) are meaningful.

Python code

if len(self.data\_window) < self.data\_window.maxlen:

return False # Not enough data yet for detection

1. **Memory Management Using Deque:**
   * The code uses a **deque** with a fixed size, which automatically **discards the oldest data** as new points are added. This ensures that the program:
     + Runs efficiently in **real-time**.
     + Does not **overflow memory** by storing too much data.

**Data Validation in the Code**

1. **Ensuring Numeric Input:**
   * Although the example uses **random floats**, in a real scenario you could easily add validation logic to ensure incoming data points are **numeric**. This would prevent errors like trying to calculate a Z-score on non-numeric values.

Example of a possible addition:

Python code

if not isinstance(new\_value, (int, float)):

raise ValueError("Data points must be numeric.")

1. **Handling Edge Cases with Random Data:**
   * In the simulation, **random values** are generated to represent both normal and anomalous data. This helps to test whether the system performs correctly when faced with **occasional spikes** or **outliers**.