**Time Series Structural Break Challenge Methodology**

Sections:

1. **Introduction to Challenge**
2. **Summary of Wavelet Based Detection, Khudhayr et al. 2022 Paper**
3. **Implementation of Wavelet Based Detection feature set for supervised learning.**
4. **Summary and Implementation of Nonparametric Structural-Break Detection (Roy et al., 2024) feature set for supervised learning.**

**Section 1: Time Series Data Science Structural Break Challenge**

🔗 Competition link: [ADIA Lab Structural Break Challenge](https://hub.crunchdao.com/competitions/structural-break)

**Objective:**  
Detect whether a structural break occurred at a given boundary point in a univariate time series.

**Definition of Structural Break (Summary):**

* A structural break is a **change in the underlying data-generating process (DGP)** at a specific time.
* Can manifest as changes in parameters (e.g., mean, variance, volatility), functional form, or regime transitions.
* Breaks may be abrupt or gradual.
* No break means the process parameters remain stable across the boundary.

**Setup:**

* Each time series is split into two candidate periods:
  + *Period 0*: before the boundary
  + *Period 1*: after the boundary
* Not every boundary represents a true break — some are false splits.
* Task: assign a score ∈ [0,1] indicating likelihood of a break.
* Evaluation: ROC AUC.

**Training Data:**

* X\_train: MultiIndex DataFrame (id, time) with columns:
  + value (float): observed time series value
  + period (0/1): before vs. after the boundary
* y\_train: Series indexed by id, Boolean values:
  + True = genuine structural break
  + False = no break
* Distribution: ~30% True, ~70% False.

**Test Data:**

* Same structure as X\_train, but labels are hidden.
* Submission: Python code with train() and infer() functions, returning one prediction per id.

**Dataset Description (Summary):**

* Tens of thousands of synthetic univariate time series, each ~1,000–5,000 values.
* Each series has a designated boundary point and label (True/False) indicating break or no break.
* Series simulate scenarios from **finance, climate, industry, biomedical signals**, with varying detection difficulty.
* Goal: algorithms must **generalize across diverse scenarios** and detect breaks in unseen data.

**Methodology Suggestions (Summary):**

* Approaches include: change-point detection, distribution comparison tests, anomaly detection, supervised ML, or deep learning.
* Useful strategies: statistical tests before/after the boundary, feature extraction from both parts, time series modeling, automated deep feature learning.
* Careful preprocessing is key for robustness.
* Aim: build reliable, generalizable detection models.

**📊 Dataset Schema Diagram**

**X\_train (DataFrame with MultiIndex)**

Index: [id, time]

Columns:

- value (float)

- period (int: {0,1})

Example:

| **id** | **time** | **value** | **period** |
| --- | --- | --- | --- |
| 0 | 0 | 0.001858 | 0 |
| 0 | 1 | -0.00166 | 0 |
| 0 | 1890 | -0.00590 | 1 |
| 0 | 1891 | 0.007295 | 1 |

**y\_train (Series)**

Index: [id]

Values: bool (True/False)

Example:

| **id** | **y** |
| --- | --- |
| 0 | True |
| 1 | False |
| 2 | False |

**Methodology for Structural Break Detection**

**Core approach:**  
Use supervised learning (e.g., gradient boosting trees, random forests, or neural nets) trained on features, tests, models extracted from periods 0 and 1.

# Section 2. Summary of Boundary Break Test via MODWT-Calibrated Residual Analysis

## Boundary Break Test Via Modwt-Calibrated Residual Analysis

(Adapted from: Rashedi, Ismail, Serroukh, Al Wadi, “Wavelet Based Detection of Outliers in Volatility Time Series Models,” CMC, 2022)

## Goal & Data Setup

* Goal: Test whether the data-generating process (DGP) in Period 1 differs materially from Period 0 at a known boundary.
* Data: Univariate time series with a known split into Period 0 (before boundary) and Period 1 (after boundary).
* Output: A decision (break vs no break), and optionally a p-value or confidence measure.

## Assumptions & Notation

* Let B be the boundary index.
* If prices are strictly positive, work with returns: R\_t = 100 \* (ln X\_t − ln X\_{t−1}); else use first differences.
* Under H0 (no break), one stationary univariate model explains the whole series across the boundary.

## Step-By-Step Procedure

0) PREPROCESSING (promote stationarity)

* If X\_t > 0, convert to returns R\_t. Otherwise difference the series.
* Optionally mean-center R\_t.

1) FIT A UNIVARIATE VOLATILITY MODEL UNDER H0 (NO BREAK)

* Fit on the entire concatenated sequence (Period 0 + Period 1).
* Recommended models: GARCH(1,1), EGARCH(1,1), or GJR-GARCH(1,1).

Example: R\_t = μ\_t + a\_t, a\_t = σ\_t ε\_t, σ\_t^2 = ω + α a\_{t−1}^2 + β σ\_{t−1}^2.

* Choose error law (Normal or Student-t). Student-t is often more robust for fat tails.
* Select model by AIC/BIC if desired.

## 2) Compute Standardized Residuals

* Obtain fitted residuals â\_t and conditional volatilities σ̂\_t.
* Standardize: ε̂\_t = â\_t / σ̂\_t. Under a well-specified H0, ε̂\_t should behave i.i.d.

## 3) Apply Modwt To Residuals

* Use MODWT with a moderate-length wavelet filter (Least Asymmetric LA(8) recommended; Haar is allowed but may cause artifacts).
* Compute wavelet coefficients W\_{j,t} for scales j = 1,…,J (typically J ≤ 3).

4) THRESHOLD CALIBRATION BY MONTE CARLO (PAPER’S CALIBRATION REGIME)

* Calibration target: max statistic per scale, W\_{j,max} = max\_t |W\_{j,t}|.
* Simulate ε\_t i.i.d. under the chosen error law and series length n.
* For each simulation, apply MODWT and record W\_{j,max}.
* Obtain quantiles q\_{j,α}(n) for desired α (e.g., 0.05 or 0.02). These are the critical values used by the paper.
* For localization near the boundary (see Step 5), also simulate window-length quantiles q\_{j,α}(m) for a local window of size m.

## 5) Localize The Test At The Known Boundary

* Define a symmetric window 𝒲 around B: 𝒲 = { t : |t − B| ≤ w }, with window size m.
* Local max statistic per scale: S\_j(B) = max\_{t∈𝒲} |W\_{j,t}|.
* Decision per scale j: Reject H0 at scale j if S\_j(B) > q\_{j,α}(m).
* Combine across scales j = 1,…,J using a multiplicity correction (e.g., Bonferroni). If any adjusted test rejects, declare a break.

Robust alternatives (optional):

* Count test: C\_j(B) = number of t ∈ 𝒲 such that |W\_{j,t}| exceeds a pointwise threshold with nominal exceedance rate α.

Under H0, C\_j(B) ≈ Binomial(m, α). Large C\_j(B) supports a break.

* Energy contrast: Compare E\_{j,0} = ∑\_{t∈Period 0} W\_{j,t}^2 and E\_{j,1} = ∑\_{t∈Period 1} W\_{j,t}^2 (normalized for segment length).

Large deviations imply a scale-specific change (variance/roughness shift). Calibrate via Monte Carlo.

6) OPTIONAL ITERATIVE CLEANING (AS IN THE PAPER)

* If large exceedances are found (global or local), zero-out the largest offending coefficient(s), inverse-MODWT to reconstruct “cleaned” residuals, and refit the volatility model.
* Iterate until no exceedances remain (stabilized residual kurtosis near 3). This step improves model adequacy but is not required for the boundary test itself.

## 7) Final Decision

* Break if any scale-specific boundary-localized test rejects after multiplicity control.
* Otherwise, fail to reject (no break detected).

## Implementation Notes

* Filter choice: LA(8) preferred to avoid blocky artifacts; Haar is acceptable but may be less smooth.
* Number of scales: small J (≤3) is sufficient and consistent with the paper; higher levels produce smaller thresholds and capture coarser changes.
* Error law for calibration: match the fitted residual law (Normal or t). If in doubt, use t with ν≈7–10.
* Window size m: choose so that m ≪ n but large enough to capture boundary-localized energy (e.g., 0.5–2% of n on each side).
* Multiple testing: across scales (and/or across statistics), adjust α (Bonferroni/Simes).
* Power vs specificity: the paper notes a “lower common hard threshold” trick (use a higher-level, smaller quantile across lower levels) to gain sensitivity when tails are heavier than assumed.

PSEUDOCODE (HIGH-LEVEL)

Input: series X\_t with boundary B, significance α, window half-width w, scales J, wavelet filter F.

1. Transform X\_t → R\_t (returns or differences).

2. Fit GARCH-family model on full R\_t; compute standardized residuals ε̂\_t.

3. MODWT\_F(ε̂\_t) → { W\_{j,t} for j=1..J }.

4. Calibrate q\_{j,α}(m) via Monte Carlo (error law = fitted; length = window size m).

5. For each scale j: S\_j(B) = max\_{t∈𝒲} |W\_{j,t}|; reject\_j = [ S\_j(B) > q\_{j,α}(m) ].

6. Apply multiplicity correction across j. If any reject\_j survives correction → declare BREAK; else NO BREAK.

(Optional) Clean-and-refit loop to improve residual diagnostics, following the paper’s procedure.

**Section 3: Implementation towards a rich feature set**

**Synopsis of Khudhayr et al. (2022)**

The paper introduces a procedure to **detect and remove outliers** in financial time series, particularly those modeled with GARCH-family models. The core idea is that if a volatility model (like GARCH) correctly captures the data's structure, its standardized residuals should be random noise (i.i.d.). However, outliers in the original data will manifest as large, sharp spikes in these residuals4.

The authors use the **Maximal-Overlap Discrete Wavelet Transform (MODWT)** to decompose the residual series5. MODWT is chosen for its flexibility over the standard DWT, as it works with any sample size and avoids certain artifacts. Outliers are identified by comparing the maximum absolute value of the wavelet coefficients at different scales against pre-calculated statistical thresholds. These thresholds are determined via Monte Carlo simulations of pure noise. The paper's main application is an iterative "cleaning" process: detect an outlier, remove its effect, and refit the model until the residuals are well-behaved.

**Outline of the Main Methodology**

**Intuitive Outline**

Imagine you're trying to tune an old radio to a clear station.

1. **Tune the Radio (Model the Data):** First, you fit a GARCH model to your financial data. This is like tuning the radio to the main signal (the predictable volatility patterns).
2. **Listen to the Static (Get Residuals):** The leftover noise or "static" that the model can't explain is the series of residuals. If the model is good, this should just be random, quiet static.
3. **Find the "Pops" (Wavelet Analysis):** An outlier in the original data creates a loud, sudden "pop" in the static. The wavelet transform (MODWT) is a special audio analyzer that is extremely good at detecting these sharp, short-lived pops at different frequencies (or "scales").
4. **Set a Volume Threshold (Statistical Test):** To decide if a pop is significant or just random noise, you need a threshold. The authors figure this out by listening to thousands of recordings of pure static to see how loud it gets naturally. Any pop in your data that's louder than this "natural maximum" is flagged as a real outlier.
5. **Clean the Signal (Iterative Removal):** Once a pop is found, the method digitally removes it from the static, tunes the radio again, and listens for more pops, repeating until the static is clean.

**⚙️ Technical Outline**

The paper's procedure for outlier detection is as follows:

1. **Model Fitting:** Fit a GARCH-family model (e.g., GARCH(1,1), EGARCH) to the financial returns series, assuming a specific error distribution (e.g., Normal or Student's t)10101010.
2. **Residual Extraction:** Compute the standardized residuals, ϵ^t​=a^t​/σ^t​. Under the null hypothesis of a correctly specified model with no outliers, these residuals should be i.i.d.
3. **Wavelet Decomposition:** Apply the MODWT with a suitable filter (LA(8) is recommended) to the standardized residuals

ϵ^t​ to obtain wavelet coefficients Wj,t​ for scales j=1,...,J.

1. **Thresholding:** For each scale j, compare the observed maximum absolute wavelet coefficient, Wj,max(obs)​=maxt​∣Wj,t​∣, to a critical value qj,α​.
2. **Calibration:** These critical values qj,α​ are (1-α) quantiles pre-computed via Monte Carlo simulation. The simulation involves generating many series of i.i.d. noise of length

n from the assumed error distribution, applying MODWT, and recording the distribution of Wj,max​.

1. **Iterative Cleaning:** If Wj,max(obs)​>qj,α​ at any scale, an outlier is detected. The procedure then sets the largest offending wavelet coefficient to zero, reconstructs the residuals via an inverse MODWT, and refits the GARCH model on the "cleaned" data. This process is repeated until no more outliers are detected.

**Revised Implementation for the Structural Break Challenge**

Proposed implementation towards a robust, feature-based machine learning solution tailored for the challenge.

**Overall Strategy Shift**

Instead of a binary hypothesis test, we'll **extract multiple statistical features** from the wavelet decomposition of the residuals. These features are designed to quantify different aspects of the change around the boundary. A supervised classifier will then learn to map these features to the probability of a break.

**Proposed Feature Engineering**

For each time series id, perform the following steps:

1. **Preprocessing & Modeling:** As you outlined, convert the full series (Period 0 + Period 1) to returns or differences and fit a single GARCH-family model under the null hypothesis (no break)16. Obtain the full series of standardized residuals

ϵ^t​.

1. **MODWT:** Apply MODWT to the entire ϵ^t​ series to get wavelet coefficients Wj,t​ for scales j=1,2,3.
2. **Feature Extraction:** Now, compute the following features for each scale j.
   * **Feature 1: Localized Maximum Coefficient**
     + This is your statistic

Sj​(B). Define a window

W of size m around the boundary B. Calculate the maximum absolute coefficient within this window. This detects sharp, spike-like changes at the boundary.

* + - Fj,max\_local​=maxt∈W​∣Wj,t​∣
  + **Feature 2: Energy Contrast**
    - This quantifies changes in variance or volatility at different scales. It compares the energy (sum of squared coefficients) in Period 1 to Period 0, normalized by length.
    - Ej,0​=len0​1​∑t∈Period 0​Wj,t2​
    - Ej,1​=len1​1​∑t∈Period 1​Wj,t2​
    - Fj,energy\_ratio​=Ej,0​Ej,1​​ (or log(Ej,1​/Ej,0​) for stability)
  + **Feature 3: Localized Exceedance Count**
    - This captures a "cluster" of unusual activity near the boundary. First, find a global threshold qj,α​(n) based on the full series length n. Then, count how many coefficients inside the window

W exceed this threshold.

* + - Fj,count\_local​=∑t∈W​I(∣Wj,t​∣>qj,α​(n))
    - We can make this feature more sensitive by using the "lower common hard threshold" trick from the paper (e.g., using

q3,α​(n) for all scales j=1,2,3).

* + **Feature 4: Distributional Change of Coefficients**
    - Compare the statistical properties of the wavelet coefficients between the two periods.
    - Fj,mean\_diff​=mean(Wj,t​ in Period 1)−mean(Wj,t​ in Period 0)
    - Fj,std\_ratio​=std(Wj,t​ in Period 1)/std(Wj,t​ in Period 0)

By calculating these 4 types of features for 3 wavelet scales, you generate a 12-dimensional feature vector for each time series id.

**Revised Step-by-Step Algorithm**

**train(X\_train, y\_train) function:**

1. Initialize an empty list for features and corresponding labels.
2. For each id in X\_train:

a. Extract the full time series value.

b. Preprocess, fit GARCH, get standardized residuals.

c. Apply MODWT to the residuals.

d. Calculate the 12-dimensional feature vector as described above.

e. Append the feature vector to the feature list and the corresponding y\_train[id] to the label list.

1. Train a gradient boosting classifier (e.g., LightGBM, XGBoost) on the complete feature matrix and labels.
2. Return the trained model.

**infer(model, X\_test) function:**

1. Initialize an empty list for features.
2. For each id in X\_test:

a. Repeat steps 2a-2d from the training function to generate the same 12-dimensional feature vector for the test series.

b. Append the feature vector to the feature list.

1. Use the trained model.predict\_proba() method on the test feature matrix to get the probability scores for the positive class (break = True).
2. Return these scores as the final predictions.

## Reference

Rashedi, K. A., Ismail, M. T., Serroukh, A., & Al Wadi, S. (2022). Wavelet Based Detection of Outliers in Volatility Time Series Models. Computers, Materials & Continua, 72(2), 3835–3847.

**Section 4. Summary of Nonparametric Structural-Break Detection (Roy et al., 2024)**

**Synopsis.** Roy, Podder & Deb (2024) propose a *nonparametric* change-detection framework for time series in the location–scale model

Y\_t = μ(X\_t) + σ(X\_t)·ε\_t,

where ε\_t are i.i.d. with unit variance. The idea is to compare the *entire shapes* of the conditional mean μ(·) and conditional variance σ²(·) **before vs. after** a split, using kernel estimators. The test statistic is the **standardized sup-norm** of the difference between segment-wise curves, evaluated on a fine grid. Under H₀ (no break), this supremum obeys a **Gumbel** extreme-value limit, giving analytic critical values. They also give **CPFind**, a two-stage binary-segmentation algorithm to locate unknown breaks.

**4.1 Intuitive outline**

* **Smooth each side:** Nonparametrically smooth the data in Period 0 and Period 1 to estimate μ₀(x), μ₁(x) (and similarly σ²₀(x), σ²₁(x)).
* **Compare shapes:** Look for the **largest absolute gap** between the two smoothed curves across x. If the biggest gap is unusually large relative to sampling noise → a break.
* **Mean and variance:** Do this once for the conditional mean and once for the conditional variance. Either can signal a structural break.

**4.2 Technical outline (known-boundary version)**

**Model.** Y\_t = μ(X\_t) + σ(X\_t)ε\_t; X\_t may be the lagged response (X\_t = Y\_{t-1}) or a short lag-vector. In our challenge we use a univariate regressor by default.

**Segments.** Split at the *given* boundary B into T₋ (Period 0) and T₊ (Period 1). Let g(·) denote μ(·) or σ²(·). Define g\_diff(x) = g₊(x) − g₋(x).

**Estimators.** Use Nadaraya–Watson (NW) kernel estimators with bandwidth b\_n and kernel K(·):

* \hat μ\_±(x) = Σ\_{t∈T\_±} K((x − X\_t)/b\_n)·Y\_t / Σ\_{t∈T\_±} K((x − X\_t)/b\_n).
* For variance, estimate \hat σ²\_±(x) from squared residuals using a bias-reduced mean (μ̂\*), per Roy24.

**Grid.** Evaluate on Π\_n = {x\_j : x\_j = Λ₁ + 2j·b\_n, j=0,…,m\_n−1}, where [Λ₁,Λ₂] is the observed support of X and m\_n ≈ (Λ₂−Λ₁)/(2b\_n).

**Standardized sup statistics.**

* Mean-break statistic:

T\_μ = c\_μ · max\_{x∈Π\_n} [ √{f̂\_X(x)} · | μ̂*\_+(x) − μ̂*\_−(x) | / √{σ̂²(x)} ],

where f̂\_X is the kernel density of X and c\_μ is the standard Roy24 normalizer ∝ √{m\_n/(n b\_n)} with a kernel-specific constant.

* Variance-break statistic:

T\_σ = c\_σ · max\_{x∈Π\_n} [ √{f̂\_X(x)} · | σ̂*²\_+(x) − σ̂*²\_−(x) | / S\_b(x) ],

where S\_b(x) and ν̂\_ε follow Roy24 to stabilize the variance of the variance-estimator.

**Null calibration.** Under H₀ and regularity conditions, T\_μ and T\_σ converge to **Gumbel**; so the critical value is B\_{m\_n}(z\_α) with z\_α = −log{−log(1−α)}. A rejection in either channel (mean or variance) implies a break.

**Unknown-break (reference).** Roy24’s **CPFind** recursively tests midpoints, splits on rejections, and runs confirmatory tests on candidate cutpoints to avoid over-segmentation. We only borrow elements of CPFind as *features* (see §5) since our boundary is known.

**4.3 Practical choices**

* **Regressor X\_t.** Default X\_t = Y\_{t−1}. (Optionally augment with short-lag vector and reduce via first PC.)
* **Kernel & bandwidth.** Symmetric bounded kernel (Epanechnikov or Gaussian). Bandwidth via CV on the NW MSE or weighted-MSE criterion described in Roy24.
* **Trimming.** For stability, restrict Π\_n to the central 98% of the empirical X-range.
* **Minimal segment length.** Enforce n\_± ≥ n\_min (e.g., 300) before computing statistics.

**Section 5. Implementation toward a Roy24 feature set (for supervised learning)**

We transform Roy24’s test apparatus into a **feature generator** evaluated at the known boundary B. Features are designed to be stable across series lengths and informative under class imbalance.

**5.1 Pointwise and sup-norm signals (per channel)**

Compute on Π\_n:

1. **Sup gap (raw):** sup\_x | μ̂*\_+(x) − μ̂−(x) |, and sup\_x | σ̂\*²*+(x) − σ̂\*²\_−(x) |.
2. **Sup gap (standardized):** T\_μ and T\_σ (the Roy24-scaled statistics).
3. **L₁/L₂ gap integrals:** mean\_x |Δμ̂\*(x)|, mean\_x Δμ̂\*(x)²; same for variance.

**5.2 Exceedance geometry on the grid**

On each channel, form standardized pointwise scores S\_μ(x) and S\_σ(x) (the bracketed terms in T\_μ/T\_σ):  
4) **Exceedance count:** #{x∈Π\_n : S\_μ(x) > c\_α}, #{x∈Π\_n : S\_σ(x) > c\_α}, where c\_α is the Roy24 Gumbel-based pointwise threshold.  
5) **Exceedance mass:** mean of S\_μ(x) over exceedance set; likewise for S\_σ(x).  
6) **Cluster width near boundary:** length (in x) of the largest contiguous exceedance cluster intersecting the boundary neighborhood.

**5.3 Boundary-localization features**

1. **Argmax position:** x*\_{μ} = argmax\_x S\_μ(x), x*\_{σ} similarly.
2. **Offset from boundary:** |x\* − x\_B| after mapping the time-boundary B to the X-scale (for X\_t = Y\_{t−1}, use contemporaneous X at B).
3. **Two-stage confirmation (CPFind-lite):** run the Roy24 confirmatory check at the nearest gridpoint to B; include its binary outcome as a feature.

**5.4 Bandwidth/complexity descriptors (regularizers)**

1. **Selected bandwidths:** b̂\_μ, b̂\_σ from CV; **grid size** m\_n; **effective sample sizes** n\_−, n\_+.

These 10–20 features (per series) integrate smoothly with the existing Wavelet21 feature block.

**5.5 Defaults and hyperparameters**

* Kernel = Epanechnikov; α = 0.05; trim = 1% per tail; grid step = 2·b̂; n\_min = 300.
* For σ²(·) estimation use the mean-bias correction (μ̂\*), S\_b(·) and ν̂\_ε per Roy24’s variance theory.
* Scale/clip S\_μ, S\_σ to [0, p99] for robustness before computing L₂ features.

**Section 6. Repository mapping and integration**

**Code location.** methods/roy24/ (mirrors the structure of methods/wavelet21/).

**Proposed modules (sketch):**

* nw.py — fast NW regression, CV bandwidth, density f̂\_X.
* roy24\_stats.py — build grid Π\_n; compute S\_μ(x), S\_σ(x); return T\_μ, T\_σ and pointwise arrays.
* roy24\_features.py — assemble the features in §5.
* \_\_init\_\_.py — user-facing extract\_roy24\_features(series\_df, cfg).

**Pipeline hooks.**

* Training: feature union [wavelet21 | roy24] → model → calibration.
* Inference: identical extraction path; ensure same kernel/bandwidth policy.

**Config keys.**

roy24:

x\_type: "lag1" # or "pc-lags:k"

kernel: "epa" # or "gauss"

alpha: 0.05

trim: 0.01

min\_seg\_len: 300

cv\_objective: "MSE" # or "WMSE"

grid\_step\_factor: 2.0

store\_pointwise: false

**Section 7. Risks, checks, and unit tests**

* **Edge support:** if Π\_n has < 10 points after trimming, fall back to larger b\_n or no-trim.
* **Small segments:** if min(n\_±) < n\_min, skip Roy24 features for that series (impute NaNs and flag).
* **Bandwidth CV:** cap b\_n ∈ [c₁·n^{-1/5}, c₂·n^{-1/5}] and monitor over-smoothing.
* **Numerical:** clip f̂\_X(x) away from 0; add ε to denominators; handle ties in argmax.

**Unit tests (minimal):**

1. No-break synthetic AR(1): T\_μ, T\_σ below B\_{m\_n}(z\_0.05) ≥ 95% of runs.
2. Mean-shift only: T\_μ power ≫ T\_σ.
3. Volatility-shift only: T\_σ power ≫ T\_μ.
4. Heavy tails (t\_ν): robustness vs. Gaussian; confirm Gumbel calibration within tolerance.