algorithms

MDPI

Article

Wavelet-Based Optimization and Numerical Computing for Fault Detection Method—Signal Fault Localization and Classification Algorithm

Nikita Sakovich [1,2], Dmitry Aksenov [1,2], Ekaterina Pleshakova [3,*] and Sergey Gataullin [4]

[1] Financial University under the Government of the Russian Federation, Moscow 109456, Russia; 217556@edu.fa.ru (N.S.); daaksenov@fa.ru (D.A.)
[2] The Scientific Research Institute of Goznak, Mytnaya Str. 17, Moscow 115162, Russia
[3] MIREA—Russian Technological University, 78 Vernadsky Avenue, Moscow 119454, Russia
[4] Central Economics and Mathematics Institute of the Russian Academy of Sciences, Nakhimovsky Prospect, 47, Moscow 117418, Russia; sgataullin@cemi-ras.ru
* Correspondence: pleshakova@mirea.ru

Abstract: This study focuses on the development of the WONC-FD (Wavelet-Based Optimization and Numerical Computing for Fault Detection) algorithm for the accurate detection and categorization of faults in signals using wavelet analysis augmented with numerical methods. Fault detection is a key problem in areas related to seismic activity analysis, vibration assessment of industrial equipment, structural integrity control, and electrical grid reliability. In the proposed methodology, wavelet transform serves to accurately localize anomalies in the data, and optimization techniques are introduced to refine the classification based on minimizing the error function. This not only improves the accuracy of fault identification but also provides a better understanding of its nature.

Keywords: mathematical modeling; wavelet analysis; numerical methods; optimization methods; signal theory

Academic Editor: Frank Werner

Received: 27 January 2025
Revised: 28 March 2025
Accepted: 2 April 2025
Published: 10 April 2025

Citation: Sakovich, N.; Aksenov, D.; Pleshakova, E.; Gataullin, S. Wavelet-Based Optimization and Numerical Computing for Fault Detection Method—Signal Fault Localization and Classification Algorithm. *Algorithms* **2025**, *18*, 217. https://doi.org/10.3390/a18040217

1. Introduction

In this paper, we consider an algorithm for the localization and classification of faults in a signal based on wavelet analysis and numerical methods: WONC-FD (Wavelet-Based Optimization and Numerical Computing for Fault Detection). Fault detection is an important problem in seismic signal processing [1–4], vibration analysis in industrial equipment [5–9], and structural and electrical network monitoring [10–12]. This approach uses wavelet transform to localize faults in the signal for subsequent classification using optimization techniques through minimization of the target error function.

This study focuses on the development of the WONC-FD (Wavelet-Based Optimization and Numerical Computing for Fault Detection) algorithm for the accurate detection and categorization of faults in signals using wavelet analysis augmented with numerical methods. Fault detection is a key problem in areas related to seismic activity analysis, vibration assessment of industrial equipment, structural integrity control, and electrical grid reliability. In the proposed methodology, wavelet transform serves to accurately localize anomalies in the data, and optimization techniques are introduced to refine the classification based on minimizing the error function. This not only improves the accuracy of fault identification but also provides a better understanding of their nature. The development of an algorithm based on wavelet analysis and additional mathematical methods will be an important step in the development of signal processing theory and fault diagnosis methods.

This approach will increase the accuracy and sensitivity of fault detection, which are critical for ensuring the reliability and safety of electrical systems. The solution of the designated scientific problem has practical and theoretical potential. Effective fault diagnosis allows one to promptly identify and eliminate problems, minimizing the risk of accidents and equipment damage.

The developed algorithm can be adapted for use in various industries where the stable operation of electrical devices is important, such as industry, energy, transportation, and household appliances. This will minimize risks and ensure the more reliable operation of electrical systems, which is an urgent task in modern industry. Modern requirements for the quality of the maintenance and operation of electrical systems entail more accurate and reliable diagnostic methods, and the developed algorithm fully meets these requirements, contributing to their satisfaction. The current state of research on fault diagnosis in electrical circuits demonstrates significant progress in the development of signal processing and machine learning techniques. However, existing methods often face limitations in accuracy and sensitivity, especially when detecting complex and immediate faults. Traditional methods such as Fourier transform do not provide detailed time–frequency analysis of signals, making it difficult to detect minor changes in signals.

In this regard, the search for new and more effective diagnostic methods becomes one of the key tasks of modern science and engineering. One of the most promising directions in the field of signal processing is the use of wavelet transform [13–17]. Wavelet analysis allows for detailed time–frequency analysis of signals, which makes it particularly attractive for diagnosing failures in electrical circuits. Many studies on the application of wavelet transform in various fields, including medical diagnosis, audio signal processing, and data analysis, confirm its effectiveness and flexibility. In particular, wavelet transform can detect small and complex changes in signals, which is critical for fault diagnosis in electrical circuits. Examples of the use of wavelet transform in fault diagnosis are in the works [18–21], where the authors apply wavelet analysis to detect faults in electrical networks. Wavelet transform can identify anomalies in voltage and current signals, which improves the diagnostic accuracy. The studies [22–25] examine the application of wavelet transform combined with machine learning techniques to classify types of faults in electrical circuits. The authors show that the combined approach significantly improves the accuracy and sensitivity of the diagnosis.

The main research directions in world science in the field of failure diagnosis include the following:

- Developing new signal processing algorithms: Researchers are actively working on developing algorithms that combine wavelet transform with other signal processing techniques, such as filtering and machine learning. These combined approaches can improve the accuracy and sensitivity of diagnostics. For example, in [26], the authors propose a new algorithm that combines wavelet transform with an LSTM-based neural network architecture to predict faults in electrical circuits.
- Machine Learning Applications: Machine learning techniques such as neural networks [27–30] and random forests are used to classify failure types based on analyzing the time–frequency characteristics of signals. These methods are able to automatically train on large amounts of data and identify different types of failures with high accuracy. In [31], the authors use neural networks to diagnose faults in electrical circuits, showing a significant improvement in accuracy over traditional methods.
- Integration with modern technologies: Modern technologies such as the Internet of Things (IoT) and cloud computing allow large amounts of data to be collected and analyzed in real time. The integration of these technologies with fault diagnosis techniques helps to improve the efficiency and reliability of fault diagnosis systems.

In [32], the authors examine the integration of algorithms with IoT systems to monitor and diagnose failures in solar PV panels. The method suggested can be used after adaptation in agent-based modeling for complex socio-economic processes coupled with high-performance computing (HPC) [33–40].

Modern approaches to data classification tasks increasingly rely on machine learning methods [41–44], particularly neural network architectures. However, the use of neural networks is associated with a number of limitations, the main of which being the need to form extensive training samples. These datasets are usually collected within a specific subject area, which complicates the process of adapting the model to new applications and requires significant computational resources to retrain it.

Other work in the application of wavelet transform for signal fault detection has a narrower and more specialized focus, particularly for electrical instrumentation applications [45–47]. The methods then aim to work for more specific electrical signals, which improves overall reliability and accuracy but is less applicable to general time series.

The proposed method uses an alternative approach based on wavelet transform and numerical optimization techniques [48–51]. The wavelet transform performs the task of localizing meaningful features in the input data, which allows for the efficient extraction of informative features. Unlike neural network methods, which require a complex training procedure, classification in our approach is based on mathematical models using libraries of basic reference signals.

A key aspect of the proposed method is the application of the standard deviation metric (MSE) as a similarity criterion. This approach allows for comparing the input data with pre-formed libraries of failure (anomaly) patterns and performing classification without the need to pre-train the model on specialized samples. Thus, the flexibility of the method and its portability to different subject areas without significant adaptation costs are ensured.

One of the important advantages of the proposed method is its modularity. This approach can be considered as service-oriented, which makes it possible to replace individual components of the system without disturbing its overall structure. For example, the classification or localization stage can be replaced by neural network models if it turns out to be appropriate for a particular task. This hybrid approach allows the algorithm to be adapted to different scenarios, combining the advantages of traditional mathematical methods and machine learning.

Some algorithm parameters, such as thresholds and other coefficients, depend on the subject area and the specifics of the problem to be solved. Their selection is based on the characteristics of the data and may require additional calibration for optimal model performance.

In general, the proposed method combines the advantages of wavelet transform for local data analysis, numerical optimization methods for efficient classification, and the flexibility of a modular approach, allowing for the integration of neural network components as needed. This makes it a competitive alternative to purely neural network models, especially in cases where the availability of large training samples is limited and adaptation to new data must be performed with minimal computational cost.

2. Materials and Methods

2.1. Wavelet Transform

The fundamental method for the algorithm to work is wavelet transform, which decomposes the input signal into detailing coefficients that can be used for subsequent failure analysis and localization.

The wavelet decomposition algorithm, often referred to as the Mallat algorithm [52] or fast wavelet transform, is an efficient method for decomposing a signal into multiple levels

of detail. This algorithm is the basis for many signal processing applications, including noise reduction, data compression, and feature analysis. The pseudocode provided as 'WaveletDecomposition' describes the scheme of Mallat's algorithm for one-dimensional signals (Algorithm 1).

Algorithm 1 WaveletDecomposition (Mallat's Algorithm)

Require: *signal* — input one-dimensional signal
Require: *wavelet* — wavelet type (e.g., 'db20')
Require: *level* — decomposition depth
Ensure: Coefficient array $coeffs = [cA_{\text{level}}, cD_{\text{level}}, \ldots, cD_1]$
Initialize the list of coefficients $coeffs$ as empty
$current\_signal \leftarrow signal$
for $k \leftarrow 1$ to *level* **do**
    Perform convolution of *current_signal* with the scaling filter (low-pass) from the selected wavelet $\rightarrow cA_k$
    Perform convolution of *current_signal* with the high-pass filter from the selected wavelet $\rightarrow cD_k$
    Perform a downsampling operation (select every second sample) for cA_k and cD_k
    $current\_signal \leftarrow cA_k$ ▷ The signal at the next step is the approximating coefficients
    Save cD_k to the internal buffer
end for
Save *current_signal* (i.e., cA_{level}) to $coeffs$
Add all detailing coefficients cD_k to $coeffs$ in order from last to first **return** $coeffs$

2.1.1. Basic Principles and Steps of the Algorithm

The wavelet decomposition algorithm is based on the use of a set of wavelet filters, including scaling (low-pass) and detail (high-pass) filters, which are associated with the selected wavelet. The decomposition process is iterative and is applied a given number of times, determined by the decomposition level ('level').

Input data: The algorithm takes as input a one-dimensional signal ('signal'), a wavelet type ('wavelet') that specifies the set of filters to be used, and a decomposition depth ('level') indicating the number of decomposition levels.

Initialization: At the beginning of the algorithm, an empty list of 'coeffs' is initialized to store the coefficients of the wavelet decomposition. The current signal being processed 'current signal' is set as equal to the input signal.

Iterative decomposition process (FOR loop): The algorithm performs iterations, the number of which being determined by the level of decomposition ('level'). At each iteration k (from 1 to 'level'), the following steps are performed:

- Scaling filter convolution: The current signal 'current signal' is subjected to a convolution operation with the scaling filter (low-pass filter) associated with the selected wavelet. The result of this operation is the approximating coefficients of the current level, denoted as cA_k. These coefficients represent the low-frequency component of the signal, reflecting the overall structure or trend of the signal at a given resolution level.
- Wrap with detail filter: Simultaneously, the same current signal 'current signal' is convolved with a detail filter (high-pass filter) also originating from the selected wavelet. The result is the detailing coefficients of the current level, denoted as cD_k. These coefficients represent the high-frequency component of the signal, containing details and abrupt changes such as noise and signal features.
- Downsampling (decimations): Both the approximating coefficients cA_k and the detailing coefficients cD_k undergo a "downsampling" operation, which consists of selecting every second sample. This is performed to reduce the size of the data and increase the

level of resolution in the next decomposition step. Downsampling is a key part of the Mallat algorithm, ensuring its efficiency.

- Update current signal: For the next iteration of the algorithm, the current signal 'current signal' is replaced by the approximating coefficients cA_k of the current level. This means that, at the next level, only the approximated, smoother version of the previous level signal is decomposed.
- Saving detailing coefficients: The detailing coefficients cD_k obtained at the current level are temporarily stored in an internal buffer for later sequencing.

Saving approximating coefficients of the last level: At the end of the iteration cycle, when a given decomposition depth ('level') has been reached, the last obtained approximating coefficients cA_{level} (which are the 'current signal' values at the last iteration) are stored in the 'coeffs' list. These coefficients represent the crudest approximation of the original signal.

Add detail coefficients to the output array: Then, all the detailing coefficients cD_k stored in the internal buffer during the iterations are added to the 'coeffs' list. It is important to note that they are added in the order of the last level to the first, i.e., in the order $cD_{level}, cD_{level-1}, \ldots, cD_1$.

Output: The algorithm returns an array of coefficients 'coeffs', which is a list starting with the approximating coefficients of the last level cA_{level}, followed by the detailing coefficients of all levels, from cD_{level} to cD_1. Thus, the structure of the output coefficients is of the following form: $[cA_{level}, cD_{level}, cD_{level-1}, \ldots, cD_1]$.

2.1.2. Meaning and Application of the Algorithm

The Mallat wavelet decomposition algorithm is an efficient and widely used method for analyzing signals in many fields. Decomposing a signal into approximating and detailing coefficients at different levels allows the signal to be analyzed at different frequency ranges and resolution levels. This is useful for identifying various signal characteristics, such as trends, details, noise, and features. The resulting coefficients can be used for a variety of tasks, including noise reduction (by thresholding the detailing coefficients), data compression (by discarding small coefficients), and feature extraction for classification and pattern recognition.

2.2. Data Preprocessing Methods

Next, it is necessary to consider methods of data preprocessing, i.e., the received signal, for further analysis: localization and classification.

2.2.1. `find_peaks` Algorithm (Peak Search)

The `find_peaks` algorithm (Algorithm 2) is designed to detect local maxima (peaks) in a one-dimensional numeric array. The main goal of the algorithm is to select point indices that correspond to peaks that meet the given criteria, particularly the "prominence" of the peak.

Main Steps of the Algorithm

Input data: The algorithm takes as input a one-dimensional array of numbers `signal` and an optional parameter `prominence`, which is a tuple `(min_threshold, max_threshold)`. The `prominence` parameter specifies the minimum and maximum thresholds for "highlighting" the peaks to be considered significant.

Local maxima: The first step of the algorithm identifies all local maxima in the input signal. A local maximum is defined as a point with index i whose value is greater than the

values at the neighboring points $signal[i-1]$ and $signal[i+1]$ ($signal[i-1] < signal[i] > signal[i+1]$).

Algorithm 2 find_peaks

1: **Input:** *signal*—one-dimensional array of numbers
2: **Input:** $prominence = (min\_threshold,\ max\_threshold)$—minimum and maximum excretion thresholds
3: **Output:** List of *peaks* indexes (maxima corresponding to a given prominence)
4: Initialize empty list *peaks*
5: Find all local maxima in *signal* (points where $signal[i-1] < signal[i] > signal[i+1]$)
6: **for** each local maximum i **do**
7: Determine its prominence—the height of the peak relative to local troughs
8: **if** prominence $\geq$ min_threshold and $\leq$ max_threshold **then**
9: Add i to the *peaks* list
10: **end if**
11: **end for**
12: **return** *peaks*

Calculation of "prominence": For each local peak found, its "prominence" is calculated. The "prominence" of a peak is a measure of its height relative to the surrounding local troughs. The exact definition of "prominence" can vary, but, in general, it reflects the vertical distance from the top of the peak to the lowest contour point defined to the left and right of the peak until the higher peak or end of the signal is reached. (In the simplified scheme, the details of the prominence calculation are not disclosed, but it is assumed that such a calculation is made.)

Threshold filtering prominence: After computing the "prominence" for each local peak, the algorithm applies threshold filtering. A peak is considered significant and is included in the list of results `peaks` only if its "prominence" is within the specified range, i.e., greater than or equal to `min_threshold` and less than or equal to `max_threshold`. If the `prominence` parameter is not set or thresholds are not specified, filtering may not be applied or may use default values.

Output: The algorithm returns a list of `peaks` that contains the indices of local maxima that satisfy the prominence criterion (if given).

Remarks

This simplified `find_peaks` algorithm focuses on the basic steps of prominence-based peak detection. However, the presented scheme reflects the key idea of selecting significant peaks based on their prominence.

2.2.2. `detect_faults` Algorithm

The `detect_faults` algorithm (Algorithm 3) is designed to detect potential "faults" in a one-dimensional signal. It uses the `find_peaks` algorithm (see Algorithm 2) to identify peaks in the absolute value of the signal, which are interpreted as indicators of damage.

Algorithm 3 detect_faults

1: **Input:** *signal*—one-dimensional array
2: **Input:** $min\_threshold, max\_threshold$—limits for the prominence parameter
3: **Output:** Tuple $(peaks, \_)$, where peaks—an array of peak indices
4: $peaks \leftarrow$ find_peaks($|\, signal \,|,\ prominence = (min\_threshold,\ max\_threshold)$) ▷ See Algorithm 2
5: **return** *peaks*

Basic Steps of the Algorithm

Input data: The algorithm takes as input a one-dimensional array `signal` and parameters `min_threshold` and `max_threshold`, which define the bounds for the prominence parameter when searching for peaks.

Finding peaks with `find_peaks`: The algorithm calls the `find_peaks` function (Algorithm 2) to search for peaks in the absolute value of the input signal $|signal|$. Parameters `prominence=(min_threshold, max_threshold)` are passed to `find_peaks` to specify the criteria for selecting peaks based on their prominence.

Output: The algorithm `detect_faults` returns a tuple whose first element is the array `peaks`. The array `peaks` contains indices of peaks detected by the `find_peaks` algorithm, which, in this context, are interpreted as indices of potential faults.

Remarks

The `detect_faults` algorithm relies on the assumption that faults in the signal appear as peaks or abnormal outliers. By using the absolute value of the signal, both positive and negative outliers can be detected. The effectiveness of fault detection depends on the proper selection of the prominence parameters (`min_threshold`, `max_threshold`), which must be adjusted according to the signal characteristics and expected types of faults.

2.2.3. `intervals_errors` Algorithm

The `intervals_errors` algorithm (Algorithm 4) is designed to identify intervals where errors or corruptions are suspected to be in the signal. It uses wavelet decomposition, peak detection, and pair generation to determine these intervals.

Algorithm 4 intervals_errors

Input: *signal*—one-dimensional array
Input: *wavelet*—wavelet type (e.g., 'db20')
Input: $level = 3$—level of decomposition
Input: $epsilon = 2$
Input: $min\_threshold = 0.1$, $max\_threshold = 25$
Input: $h = 1e{-}6$
Output: List of intervals (pairs) where errors are suspected
$coeffs \leftarrow$ WaveletDecomposition$(signal, wavelet, level)$ ▷ See Algorithm 1
$cf\_lvl \leftarrow coeffs[level]$ ▷ Coefficients at a given level (detailing or approximating - depends on the library)
$peaks \leftarrow$ detect_faults$(cf_lvl, min_threshold, max_threshold)$
Build a list of *eps_peaks* by $peak - epsilon$ for each $peak \in peaks[0]$
$intervals \leftarrow$ combinations$(eps_peaks, 2)$
return *intervals*

Main Steps of the Algorithm

Input data: The algorithm takes as input a one-dimensional array `signal`, wavelet type `wavelet`, decomposition level `level` (default 3), parameter ϵ (epsilon, default 2), prominence threshold parameters `min_threshold` and `max_threshold`, and filter threshold `h` (default 1×10^{-6}).

Wavelet decomposition: In the first step, the algorithm applies wavelet decomposition to the input signal `signal` using the `WaveletDecomposition` algorithm (Algorithm 1), the selected wavelet `wavelet`, and the decomposition level `level`. The result is a list of coefficients `coeffs`.

Selection of level coefficients `level`: The algorithm selects coefficients at a given decomposition level from a list of `coeffs`. In pseudocode, this is the string `cf_lvl` ← `coeffs[level]`.

Detect peaks in coefficients: The algorithm applies the `detect_faults` function (Algorithm 3) to the selected wavelet coefficients `cf_lvl` with the given prominence thresholds `min_threshold` and `max_threshold`. The result is a list of peak indices of `peaks` in coefficients.

Building "extended" peaks: For each peak index `peak` of `peaks[0]`, an "extended peak" is created from `peaks[0]` `eps_peaks` by subtracting the value ϵ from the peak index (`peak-epsilon`). The purpose of this step is to possibly expand the area around the peak to determine the error interval.

Generation of intervals (pairs): The algorithm uses the `combinations` function to generate all pairs of combinations from a list of "extended" peaks `eps_peaks` of length 2. Each pair represents an interval potentially containing an error.

Output: The algorithm returns a list of `intervals` containing index pairs representing the intervals where the errors in the original signal are suspected to be located.

Remarks

The `intervals_errors` algorithm combines wavelet analysis with peak detection and combinatorics to determine error intervals. Using wavelet decomposition allows one to analyze the signal at different frequency levels and possibly better isolate features associated with faults. The ϵ parameter and pair generation can be associated with trying to identify intervals around peaks where errors are most likely to occur. Adjusting the wavelet parameters, decomposition level, prominence and filter thresholds, and the ϵ value is critical to the effectiveness of the algorithm in specific applications.

2.2.4. `clean_signal`

The `clean_signal` algorithm (Algorithm 5) is designed to "clean" a signal, represented as a structure or array with `x` and `y` fields. The algorithm removes the sinusoidal component from the signal `y` to investigate the latest signal faults.

Algorithm 5 clean_signal

1: **Input:** *signal*—structure/array with fields x and y
2: **Output:** New one-dimensional array *y_clean*
3: $(vals,\ counts) \leftarrow \text{unique}(signal[\text{“}y\text{”}],\ \text{return\_counts} = True)$
4: $Ampl \leftarrow vals[\arg\max_{i \in counts}(counts)]$ ▷ Find the most frequent value
5: $y \leftarrow signal[\text{“}y\text{”}] - |Ampl| \cdot \sin(100 \cdot \pi \cdot signal[\text{“}x\text{”}])$
6: **return** y

Main Steps of the Algorithm

Input data: The algorithm takes as input a structure or array `signal`, which is assumed to contain the fields `x` and `y`. The `x` field may represent the time axis or other independent variables, and the `y` field may represent the signal values.

Find the most frequent value. The algorithm uses the `unique` function to find unique values in the array `signal['y']` and count their frequency. The result is the arrays `vals` (unique values) and `counts` (the number of occurrences of each unique value).

Determination of the amplitude of `Ampl`:. The algorithm finds the value of `Ampl`, which is the most frequent value in `signal['y']`. This is achieved by finding the index of the maximum value in the `counts` array using the `argmax` function and then extracting the corresponding value from the `vals` array.

Sinusoidal component subtraction: The algorithm creates a new array `y` by subtracting the sine function from the original signal `signal['y']`. The sine function is of the form $|Ampl| \cdot \sin(100 \cdot \pi \cdot signal[\text{“}x\text{”}])$. The amplitude of the sinusoid is given as the absolute

value of `Ampl`, the frequency is fixed as 100π (which corresponds to a frequency of 50 Hz if `x` is in seconds, or 100 cycles per unit of `x`), and the phase is zero.

Output: The algorithm returns a new one-dimensional array `y_clean`, which is the result of subtracting the sine component from the original signal `signal['y']`.

Remarks

The `clean_signal` algorithm is designed to remove a specific type of noise: a sine wave interference with a fixed frequency (or frequency proportional to `x`). The method of determining the amplitude of the noise (by searching for the most common value) is heuristic and may not always be efficient or accurate. The choice of frequency (100π) and the shape of the sine function assumes that the type of interference is known in advance. More general noise reduction may require more sophisticated techniques, such as wavelet noise reduction or filtering in the frequency domain.

2.2.5. `pad_array` Algorithm

The `pad_array` algorithm (Algorithm 6) is designed to augment the input array `arr` with zeros up to a given length `length`. If the input array already has a length greater than or equal to `length`, the algorithm returns the original array unchanged. If the length of the array is less than the required length, the missing elements are added symmetrically on both sides of the array as zeros.

Algorithm 6 pad_array

Input: *arr*—source array, *length*—required length of the result
Output: New array of length *length*
$current\_len \leftarrow \text{length}(arr)$
if $current\_len \geq length$ **then**
    return *arr* ▷ The array is already long enough
else
    $pad\_len \leftarrow length - current\_len$
    $left\_len \leftarrow \lfloor \frac{pad\_len}{2} \rfloor$
    $right\_len \leftarrow pad\_len - left\_len$
    Augment *arr* to the left by $left\_len$ zeros and to the right by $right\_len$ zeros
    return New augmented array
end if

Main Steps of the Algorithm

Input data: The algorithm takes as input an array `arr` and the desired length `length` (integer).

Check current length: The algorithm first determines the current length of the input array `current_len`. Then, it compares `current_len` with the given length `length`.

In case the array is long enough: If `current_len` is greater than or equal to `length`, it means that the array already has the required or an excessive length. In this case, the algorithm does not perform any addition actions and simply returns the original array `arr`.

In case the array needs to be appended: If `current_len` is less than `length`, the array must be appended to the required length:

1. Calculating the length of the complement. The total length of the complement `pad_len` is calculated as the difference between the required length `length` and the current length `current_len`: $pad\_len = length - current\_len$.
2. Left and right complement distribution. The length of the complement `pad_len` is distributed approximately equally between the left and right sides of the array to ensure a symmetrical complement.

- The length of the left complement `left_len` is calculated as the integer part of dividing `pad_len` by two: $left\_len = \lfloor \frac{pad\_len}{2} \rfloor$.
- The length of the right-hand complement `right_len` is calculated as the remainder: $right\_len = pad\_len - left\_len$. This distribution ensures that the total length of the complement is equal to `pad_len`, even if `pad_len` is an odd number.

3. Zeros complementation. The array `arr` is appended with `left_len` zeros on the left and `right_len` zeros on the right. The complement operation is usually performed by creating a new array and copying the original elements of the `arr` array into it, adding zeros at the beginning and end.

Output data: The algorithm returns a new array whose length is equal to `length`. If the original array was shorter, it will be appended with zeros symmetrically on both sides. If the original array was long enough, the original array is returned.

Remarks

The `pad_array` algorithm is useful in situations where arrays need to be brought to a uniform length for further processing, such as in batch processing, where all input data must have the same dimensionality. Symmetric addition with zeros helps to minimize distortions that can be introduced by addition, especially in the context of signal processing, where the position of elements relative to the beginning and end of the array may be important.

2.2.6. `resize_vector` Algorithm

The `resize_vector` algorithm (Algorithm 7) is designed to resize a one-dimensional array (vector) `vector` to a new specified length `new_size`. The algorithm provides two main cases of resizing—length increase and length decrease—and uses different approaches for each case.

Algorithm 7 resize_vector

Input: *vector*—original one-dimensional array, *new_size*—new length
Output: New array of length *new_size*
$old\_size \leftarrow \text{length}(vector)$
if $new\_size = old\_size$ **then**
    return *vector* ▷ Size does not change
else if $new\_size > old\_size$ **then** ▷ Interpolation at increasing length
    Let *indices* be uniform values from 0 to $old\_size - 1$ with the number of points *new_size*
    Create an output array by interpolating *vector* at *indices* points
    return Resulting array
else ▷ Averaging over decreasing length
    Let *indices* be uniform values from 0 to $old\_size - 1$ with the number of points *new_size*
    Initialize *resized_vector* with zeros of length *new_size*
    for $i \leftarrow 1$ to *new_size* **do**
        $start\_index \leftarrow \lfloor indices[i] \rfloor$
        $end\_index \leftarrow start\_index + 1$ ▷ or *old_size*, if $i = new\_size$
        $resized\_vector[i] \leftarrow$ mean value of the *vector* subarray on the interval $[start\_index, end\_index)$
    end for
    return *resized_vector*
end if

Main Steps of the Algorithm

Input data: The algorithm takes as input a one-dimensional array `vector` and a new desired length `new_size` (integer).

Size comparison: The algorithm compares the current length of the array `old_size` with the new length `new_size`.

Case where the size does not change: If `new_size` is equal to `old_size`, then the array size does not need to be changed. In this case, the algorithm simply returns the original array `vector` unchanged.

Length increase case (`new_size` $>$ `old_size`): When the length of the array needs to be increased, the algorithm uses interpolation to create new values between existing points:

1. Create interpolation indexes: An array of `indices` is created containing `new_size` evenly spaced values ranging from 0 to `old_size` $-$ 1. These indices represent the positions in the original array for which values must be interpolated to produce an array of the new length.
2. Interpolation: The original array `vector` is interpolated at the points specified by the array `indices`. The interpolation method is not explicitly specified in the pseudocode, but it is implied that some one-dimensional interpolation method (e.g., linear, spline, or cubic) is used that approximates the signal values between the source points.
3. Output data (after interpolation): The algorithm returns a new array resulting from the interpolation, whose length is equal to `new_size`.

Length reduction case (`new_size` < `old_size`): When the length of the array needs to be reduced, the algorithm uses value averaging to preserve the overall information while reducing the granularity:

1. Creating indexes for averaging: Similar to the increment case, an array of `indices` is created containing `new_size` evenly spaced values ranging from 0 to `old_size` $-$ 1. These indices define the boundaries of the intervals in the original array, which will be averaged to obtain the values in the new array of reduced size.
2. Initializing the output array: A new array `resized_vector` of length `new_size` filled with zeros is created.
3. Averaging over intervals (FOR loop): For each index *i* from 1 to `new_size` (in pseudocode, the numbering starts at 1, which may be atypical for Python-like languages, usually indexes start at 0):
   - The start and end indices of the interval in the source array are specified: `start_index` and `end_index`, using the values from the `indices` array. `start_index` is taken as an integer part of `indices[i]`. `end_index` is usually equal to `start_index + 1`, but may be limited to `old_size` for the last interval to keep within the bounds of the original array.
   - The average value of the elements of the `vector` subarray on the interval $[$`start_index`, `end_index`$)$ is calculated.
   - The calculated average value is assigned to the element `resized_vector[i]`. (in the pseudocode, the indexing of `resized_vector` also starts from 1, which should be taken into account in the implementation).
4. Output (after averaging): The algorithm returns an array of `resized_vector` of length `new_size` obtained by averaging sections of the original array.

Output data: Depending on the relationship between `old_size` and `new_size`, the algorithm returns either an interpolated array (if the size increases), an averaged array (if the size decreases), or the original array (if the size does not change).

Remarks

The `resize_vector` algorithm provides two different methods of resizing an array depending on whether the size is increasing or decreasing. Using interpolation while increasing the size allows new points to be added while maintaining the overall waveform.

Averaging while decreasing the size allows one to preserve the overall energy of the signal while reducing the detail. The choice of a specific interpolation method and averaging method can be customized depending on the requirements of the task.

2.3. Algorithm

Now, we can consider the basic algorithms for signal fault localization and classification, which are based on the use of previous methods and algorithms.

Mathematically, we can describe the problem to be solved as follows:

$\arg\min_{(s,t,a)\in S\times T\times A} \mathrm{MSE}(y, \hat{y})$.

$\arg\min_{(s,t,a)\in S\times T\times A} \mathrm{MSE}(y, \mathrm{ErrorFunction}(s, t, a))$.

Here, the following applies:

1. y: the original signal represented as a time series. This signal is the reference signal and is used to evaluate the accuracy of the approximation.

2. $\hat{y}$: the approximated signal obtained by applying an error function with certain parameters. The goal is to match this signal as closely as possible to the original signal y.

3. ErrorFunction: a function that takes stretch, shift, and amplitude parameters as input and returns an approximated signal. Formally, $\mathrm{ErrorFunction}(s, t, a)$ denotes the approximated signal obtained given the parameters s, t, and a.

4. S: the set of possible values of the signal stretching parameter along the OX axis. Each element of this set represents a stretching factor that can be applied to the original signal.

5. T: the set of possible values of the signal shift parameter. Each element of this set represents a shift value that can be applied to the original signal.

6. A: a set of possible values of the signal amplitude parameter. Each element of this set represents an amplitude coefficient that can be applied to the original signal.

7. $S \times T \times A$: The Cartesian product of sets S, T, and A that is the set of all possible combinations of stretch, shift, and amplitude parameters.

The purpose of the method is to find such a combination of parameters (s, t, a) that minimizes the MSE error between the original signal y and the approximated signal $\hat{y}$.

Thus, this formula defines a procedure for finding the optimal stretch, shift, and amplitude parameters that provide the best approximation of the original signal in the sense of minimizing the MSE error.

The main steps of the algorithm are as follows (Figure 1):

1. Input signal: Obtaining a raw signal containing potential anomalies (failures).

2. Signal preprocessing: Normalization, the removal of unwanted noise components, and peak detection.

3. Wavelet analysis: Wavelet transform (decomposition of the signal into wavelet coefficients for analysis at different scales) and coefficient analysis (examination of wavelet coefficients to identify features associated with failures).

4. Error Classification: Using optimization methods (Nelder–Mead method, BFGS) to minimize the MSE between a failure in the signal interval and a possible failure from error sample libraries.

5. Results output: Error classes (defined failure types), error estimation (metrics output), temporal localization (determining when the failure occurs).

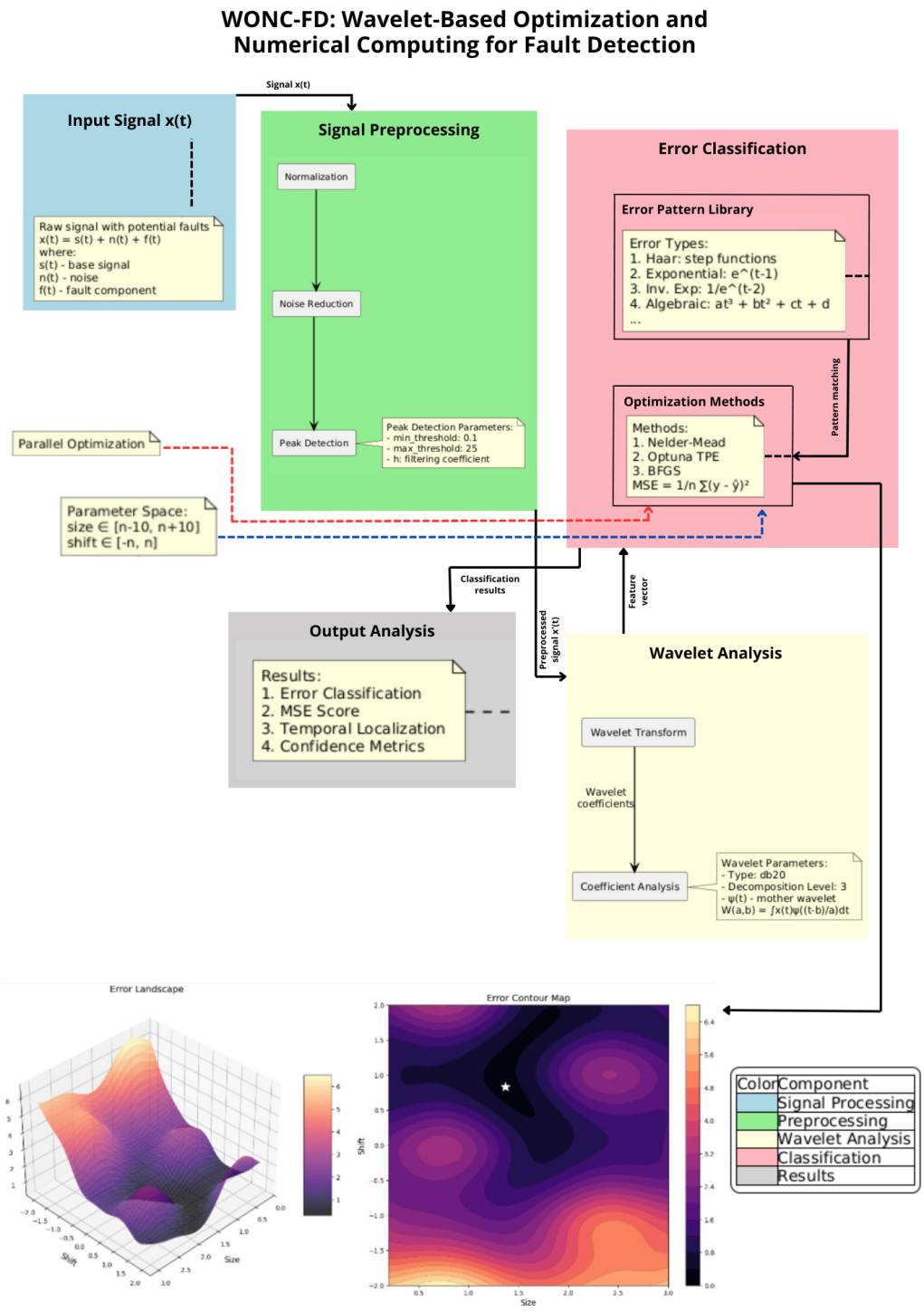

Figure 1. Algorithm diagram.

2.3.1. `WONC-FD mse_classification` Algorithm

The `mse_classification` algorithm is designed to classify the type of "error" or "defect" in a one-dimensional signal `y` by comparing it to a set of predefined "error signals" `Error_signal` of various types listed in the array `errors`. The classification is based on minimizing the mean square error (MSE) between the input signal and the approximation obtained using different error types, sizes, and shifts.

The Main Steps of the Algorithm 8.

Algorithm 8 WoncFD_mse_classification

Input: y—one-dimensional array of points,
errors—list of possible error types (e.g., ["haar", "haar1", ...]),
leng—base length of the signal (e.g., 1000),
epsilon—a small threshold to check for proximity to 0,
n—fraction of zero (modulo) points for rejection.
Output: List: [name of the best error type, MSE]
if $(\text{length}(y) \leq leng \times 0.05)$ **or** $(\sum_i (1 \text{ if } |y_i| < \epsilon \text{ else } 0) > \text{length}(y) \times n)$ **then**
    return ["Bad signal", "NaN"]
end if
Initialize empty array $result$
$x \leftarrow$ an array of uniform values from 0 to 1 of size $\text{length}(y)$
$ampl \leftarrow 1$
$size\_values \leftarrow \{\text{length}(y), \text{length}(y)+10, \text{length}(y)+20, \ldots, 2 \times \text{length}(y)\}$
$shifts \leftarrow \{-\text{length}(y), -\text{length}(y)+1, \ldots, \text{length}(y)\}$
for each err of $errors$ **do**
    $mse\_err \leftarrow$ a large number (e.g., 10^5)
    for each s of $size\_values$ **do**
        for each $shift$ of $shifts$ **do**
            Calculate the vector of error values from the dictionary $Error \leftarrow$ resize_vector(Error_signal($err, x, ampl, 1$), s)
            if $shift \geq 0$ **then**
                $arr\_error \leftarrow [0, \ldots, 0, Error][:\text{length}(x)]$ forward-shifted
            else
                $arr\_error \leftarrow [Error, 0, \ldots, 0][-\text{length}(x):]$ backward
            end if
            features_matrix $\leftarrow$ a matrix of size $(\text{length}(x), 2)$, where the first column is 1, the second column is arr_error
            Find the $coefficients$ using the least squares method:

$$\min_{\beta} \|\text{features\_matrix} \cdot \beta - y\|^2$$

            $approximated\_signal \leftarrow$ features_matrix $\cdot\, coefficients$
            $mse \leftarrow \text{MSE}(y, approximated\_signal)$
            if $mse < mse\_err$ **then**
                $mse\_err \leftarrow mse$
            end if
        end for
    end for
    Add mse_err to the $result$ array
end for
Find the index $\arg\min_{i \in result}(result)$
return $[errors[\arg\min_{i \in result}(result)], result[\arg\min_{i \in result}(result)]]$

Input data: The algorithm takes as input a one-dimensional array of points `y`, a list of error type names `errors`, a base signal length `leng`, a small threshold ϵ to check for proximity to zero, and a fraction of zero (modulo) points n to reject the signal.

Preliminary Signal Checking: The algorithm performs an initial check on the input signal `y` to weed out "bad" signals that may lead to incorrect results.

- Check signal length: The signal is checked for whether it is too short. If the length of `y` is less than or equal to 5% of the base length of `leng` (i.e., `length`(y) $\leq 0.05 \times leng$), the signal is considered "bad".
- Check for near-zero dots: The number of points in `y` whose absolute value is less than the specified small threshold ϵ ($|y_i| < \epsilon$) is counted. If the number of such points

exceeds a fraction n of the total signal length (i.e., the number of points $|y_i| < \epsilon >$ $n \times \text{length}(y)$), the signal is also considered "bad".

- Return for "bad" signals: If a signal is considered "bad" by one of the above criteria, the algorithm does not perform further classification and returns the list [‘‘`Bad signal`’’, ‘‘`Nan`’’] as the result, indicating that classification is impossible or unreliable.

Initialization: For good signals, the algorithm continues the classification process:

- An empty `result` array is initialized to store MSE values for each error type.
- An array `x` is created representing uniformly distributed values from 0 to 1, the size of the signal length `y`. This can serve as a normalized "time" axis for error generation.
- The initial amplitude of `ampl` for generated errors is set to 1.
- A set of `size_values` is defined. In pseudocode, these are sizes from `length`(y) in increments of 10 to $2 \times$ `length`(y). These dimensions will be used to resize the generated error signals.
- A set of `shifts` from $-$`length`(y) to `length`(y) in increments of 1 is defined. These shifts will be used to offset the generated error signals relative to the beginning of the `y` signal.

Cycle by error type (external FOR loop): The algorithm iterates over each error type `err` from the list `errors`. For each error type, the following process is performed:

- Initialization of minimum MSE for error type: The initial value of the minimum MSE `mse_err` for the current error type is set to a large number (e.g., 10^5). This value will be updated if smaller MSE values are found.
- Cycle by size (middle FOR loop): For each size s from the set `size_values`:
- Shift cycle (inner FOR loop): For each shift `shift` from the set of `shifts`:
  1. Generating and resizing the error signal: A "base" error signal is generated using the `Error_signal(err, x, ampl, 1)` function for the current error type `err`, "time axis" `x`, amplitude `ampl`, and scale 1. Then, using the function `resize_vector`, the size of the generated error signal is resized to length s.
  2. Applying a shift to the error signal: Depending on the sign of the shift `shift`, the error signal `Error` is shifted either forward (if `shift` ≥ 0) or backward (if `shift` < 0) relative to the signal `y`. The shift is realized by adding zeros to the beginning or end of the error signal and truncating the result to the length `length`(x).
  3. Building a feature matrix: A feature matrix `features_matrix` of size (`length`(x), 2) is created. The first column of the matrix is filled with ones and the second column is filled with the shifted error signal `arr_error`. The first column with units allows us to account for the bias (constant) in the model.
  4. Least Squares Method (LSM): A linear regression problem is solved using the least squares method to find the coefficients (β) minimizing the norm of the difference $\|\text{features\_matrix} \cdot \beta - y\|^2$. This allows us to find the optimal linear combination (bias and scaling) of the error signal to approximate the `y` signal.
  5. Calculating the approximated signal: The approximated signal `approximated_signal` is calculated as the product of the feature matrix `features_matrix` by the found coefficients `coefficients`.
  6. Calculation of MSE: The mean square error (MSE) between the original signal `y` and the approximated signal `approximated_signal` is calculated.
  7. Update Minimum MSE: If the calculated MSE value is less than the current minimum MSE `mse_err` for this error type, then `mse_err` is updated with this new, smaller MSE value.

Saving the minimum MSE for the error type: After the size and shift cycles are completed, the minimum MSE value found `mse_err` for the current error type `err` is added to the array `result`.

Determination of the best error type: After looping through all error types, the algorithm finds the index of the minimum value in the `result` array. This index corresponds to the error type that provided the lowest MSE when approximating the signal `y`.

Output: The algorithm returns a list containing two elements: the name of the best error type (from the `errors` list) and the corresponding minimum MSE value. This indicates the error type that best "explains" the `y` signal structure in terms of MSE.

Remarks

The `mse_classification` algorithm is a classification method based on matching a signal to a set of reference "error" signals. Using MSE as an approximation quality metric, it quantifies how well each error type matches the input signal. The enumeration of different sizes of and shifts in the error signals is designed to account for possible differences in scale and phase between expected errors and actual deviations in the signal. The least squares method is used to optimally "fit" the scale and offset of the error signal to the input signal. The efficiency of the algorithm depends on the representativeness of the set of `errors`, the adequacy of the `Error_signal` function to model real errors, and the proper choice of parameters, such as `size_values`, `shifts`, *epsilon*, and n. The algorithm is computationally intensive due to nested loops and multiple solutions to the least squares problem.

2.4. Parallel Algorithms

It makes sense to parallelize the previous algorithms to optimize the computations in terms of time, since these computations can take a large amount of time with a single thread.

2.4.1. `objective_function` Algorithm (Loss Function)

The `objective_function` algorithm (Algorithm 9) is a target function used in the parameter optimization process using the Optuna library (with TPESampler). This function is designed to evaluate the quality of the approximation of the input signal `y` using a model based on a given "error" type `error`. The quality of the approximation is evaluated using the mean square error (MSE). The function is used in the context of searching for optimal parameters (size and shift) to model a certain type of error in the signal.

Main Steps of the Algorithm

Input data: The algorithm takes as input a one-dimensional array `y` representing the signal points with an error, and a string `error` indicating the type of error to be modeled.

Selecting parameters to optimize: The function uses the `trial` object to suggest parameter values to be optimized:

- Error size (`size`): An integer value of `size` in the range from `length`(y) − 10 to `length`(y) + 10. This parameter specifies the length of the error signal to be generated and then reduced to the length of the `y` signal.
- Error Shift (`shift`): An integer value of `shift` in the range of $-$`length`(y) to `length`(y). This parameter defines the shift in the error signal relative to the `y` signal.

Time axis generation: An array `x` is created, representing evenly spaced values from 0 to 1, with a length equal to the length of the signal `y`. This array is used as an argument to the function `Error_signal`, which generates an error signal.

Algorithm 9 objective_function (loss function)

Input: y—one-dimensional array of signal points with error,
error—string, name of the error type
Output: MSE (Mean Square Error) for the given parameters
$size \leftarrow (\text{length}(y) - 10, \text{length}(y) + 10)$ ▷ Integer size frames
$shift \leftarrow (-\text{length}(y), \text{length}(y))$ ▷ Integer Shift Frames
$x \leftarrow$ array of uniform values from 0 to 1 of size length(y)
$ampl \leftarrow 1$
if $shift \geq 0$ **then**
    $Error \leftarrow$ resize_vector(Error_signal$(error, x, ampl, 1), size)$ ▷ Resize the error signal (See Algorithm 7).
    $Error \leftarrow \text{resize}(Error, (1, \text{length}(y)))$ ▷ Change the dimension of the Error array to (1, length(y))
    $Error \leftarrow Error[0]$ ▷ Get the first line of the Error array
    $arr\_error \leftarrow \text{concatenate}([0, \ldots, 0, Error])[: \text{length}(x)]$ ▷ Complete with zeros and trim
else
    $Error \leftarrow$ resize_vector(Error_signal$(error, x, ampl, 1), size)$ ▷ Resize the error signal (See Algorithm 7).
    $Error \leftarrow \text{resize}(Error, (1, \text{length}(y)))$ ▷ Change the dimension of the Error array to (1, length(y))
    $Error \leftarrow Error[0]$ ▷ Get the first line of the Error array
    $arr\_error \leftarrow \text{concatenate}([Error, 0, \ldots, 0])[-\text{length}(x) :]$ ▷ Complete with zeros and trim
end if
features_matrix $\leftarrow$ vstack$([1, \ldots, 1], arr_error).T$ ▷ Create feature matrix
$coefficients \leftarrow$ lstsq(features_matrix, y) ▷ Least Squares Method
$approximated\_signal \leftarrow$ dot(features_matrix, $coefficients$) ▷ Calculate approximated signal
$mse \leftarrow \text{MSE}(y, approximated\_signal)$ **return** *mse*

Generation and resizing of the error signal: A "base" error signal is generated using the `Error_signal(error, x, ampl, 1)` function for the specified error type `error`, "time axis" `x`, amplitude `ampl` (fixed at 1), and scale 1. Then, using the function `resize_vector` (Algorithm 7), the size of the generated error signal is resized to the proposed value `size`. After resizing, the array `Error` is converted to dimension $(1, \text{length}(y))$, and then the first row is extracted to obtain a one-dimensional array.

Applying a shift to the error signal: Depending on the sign of the proposed shift `shift`, the error signal `Error` is shifted relative to the beginning of the signal `y`. If `shift` ≥ 0 (forward shift), `shift` zeros are added to the error signal on the left, and the result is truncated to length `length`(x). If `shift` < 0 (backward shift), $-$`shift` zeros are added to the error signal on the right, and the result is trimmed from the beginning to `length`(x).

Building the feature matrix and approximating the least squares.

- A feature matrix `features_matrix` is created, consisting of two columns: the first column is filled with units (to account for bias) and the second column is filled with a shifted error signal `arr_error`.
- We solve the linear regression problem by the least squares method (LSM) using the function `lstsq` to find the coefficients `coefficients` that minimize the difference between the `features_matrix` $\cdot$ `coefficients` and the input signal `y`.
- The approximated signal `approximated_signal` is computed as the product of the feature matrix `features_matrix` by the found coefficients `coefficients`.

Calculation of MSE: The mean square error (MSE) between the original signal `y` and the approximated signal `approximated_signal` is calculated using the function `MSE`.

Output: The algorithm returns a computed MSE value, which is a measure of the "mismatch" between the signal `y` and its approximation using a given error type and the proposed size and shift parameters. The goal of Optuna (TPESampler) is to minimize this MSE value.

Remarks

The `objective_function` is a key component for optimizing the parameters of an error classification model using TPESampler. It evaluates the quality of the model for given values of the size and bias of the error signal, allowing TPESampler to iteratively search for parameters that minimize the MSE and hence provide the best fit between the error model and the input signal. The use of the least squares method provides an optimal linear fit of the error signal to the input signal based on the criterion of minimizing the MSE.

2.4.2. `optimize_params` Algorithm (Parameter Optimization)

The `optimize_params` algorithm (Algorithm 10) is designed to automatically optimize the parameters (size and bias) of an error classification model using the Optuna library. It uses the objective function `objective_function` (Algorithm 9) to estimate the quality of the parameters and iteratively searches for a set of parameters minimizing the MSE between the input signal and its approximation.

Algorithm 10 optimize_params (parameter optimization)

Input: y—one-dimensional array of signal points with error,
$error$—string, the name of the error type,
$iters$—number of optimization iterations.
Output: Best MSE value
$study \leftarrow$ create_study(direction='minimize', sampler $=$ TPESampler()) ▷ Create a minimization problem based on the TPESampler algorithm.
$objective\_lambda(trial) =$ objective_function$(trial, y, error)$ ▷ Wrapper for objective_function
study.optimize($objective\_lambda$, n_trials $= iters$) ▷ Start Optimization **return** study.best_value ▷ Return the best value found

Main Steps of the Algorithm

Input data: The algorithm takes as input a one-dimensional array `y` of signal points with an error, a string `error` indicating the type of error, and an integer `iters` specifying the number of optimization iterations that Optuna should perform.

Creating an optimization problem: A `study` object is created using the `create_study` function. The `direction='minimize'` parameter indicates that the optimization objective is to minimize the target function (MSE). The parameter `sampler=TPESampler()` specifies the parameter sampling algorithm; in this case, the tree-structured Parzen estimator (TPE) is used, which is an efficient algorithm for Bayesian optimization.

Define lambda function for Objective Function: A lambda function `objective_lambda` is created that "closes" the target function `objective_function` with specific inputs `y` and `error`. This is necessary so that Optuna can call the target function with the argument `trial` without passing `y` and `error` each time. In fact, `objective_lambda(trial)` is equivalent to `objective_function(trial, y, error)`.

Start optimization: The optimization process is started using the `study.optimize` method. The lambda function `objective_lambda` is passed as the first argument, and the parameter `n_trials=iters` specifies the number of optimization iterations (the number of times the target function is called to examine different sets of parameters). The optimizer

iteratively calls `objective_lambda`, offering different parameter values (size and shift) and trying to find those that minimize the return value (MSE).

Obtaining the best value: After completing a given number of optimization iterations, the algorithm retrieves the best found value of the target function (minimum MSE) achieved during the optimization using the `study.best_value` attribute.

Output: The algorithm returns the best MSE value found during the optimization process. This value represents the minimum MSE achieved for a given error type `error` and input signal `y`, with optimized size and shift parameters.

Remarks

The `optimize_params` algorithm automates the process of finding optimal parameters for modeling an error of a given type. Using the TPE method, we can efficiently explore the parameter space and find the values that provide the best fit between the error model and the input signal according to the MSE criterion. The number of `iters` iterations affects the quality of optimization and the execution time; a larger number of iterations tends to increase the probability of finding the global minimum, but also increases the computational cost.

2.4.3. `classification_parallel` Algorithm (Parallel Classification)

The `classification_parallel` algorithm (Algorithm 11) is a basic algorithm for classifying error types in a signal using parallel computing and the Optuna library for parameter optimization. It is designed to classify a `y` signal by comparing it with a set of different error types listed in the `errors` array. The classification is performed by finding the error type that provides the minimum MSE after optimizing the parameters for each error type. Parallelization is used to speed up the classification process by performing optimization for different error types simultaneously.

Algorithm 11 classification_parallel (parallel classification)

Input: y—one-dimensional array of signal points with error,
errors—list of error type names (default: [“haar”, “haar1”, . . .]),
leng—basic signal length (default: 1000),
epsilon—small threshold to check proximity to 0 (default: 0.01),
n—fraction of zero (modulo) points for rejection (default: 0.98),
iters—number of iterations for optimization (default: 100).
Output: List: [name of the best error type, MSE]
if $(\text{length}(y) \leq leng \times 0.05)$ OR $(\sum_i (1 \text{ if } |y_i| < \epsilon \text{ else } 0) > \text{length}(y) \times n)$ **then**
    return [“Bad signal”, “Nan”]
end if
$y \leftarrow \text{pad\_array}(y, leng)$ ▷ Add the signal to the base length (See Algorithm 6).
$best\_losses \leftarrow \text{Parallel}(\text{n\_jobs} = -1)(\text{delayed}(\text{optimize\_params})(y, error, iters)$ for each *error* in *errors*) ▷ Parallel optimization for each error type
$best\_loss\_idx \leftarrow \arg\min(best\_losses)$ ▷ Find the minimum MSE index
$best\_error \leftarrow errors[best\_loss\_idx]$ ▷ Name of best error type
$best\_loss \leftarrow best\_losses[best\_loss\_idx]$ ▷ Best MSE value
$result \leftarrow [best\_error, best\_loss]$ **return** *result*

Main Steps of the Algorithm

Input data: The algorithm takes as input a one-dimensional array `y` of signal points with errors, a list of strings `errors` containing the names of error types to classify (by default [“haar”, “haar1”, ‘exp’, ‘-exp’, ‘Invexp’, ‘Invexp+’, ‘algb’]), the base signal length `leng` (the default is 1000), a small threshold ϵ to check proximity to zero (the default is 0.01),

the fraction of zero (modulo) points n for signal rejection (the default is 0.98), and the number of iterations `iters` for Optuna optimization (the default is 100).

Preliminary check of the signal for "bad" quality: The algorithm performs an initial quality check on the input signal `y` to rule out cases where the signal is too short or contains an excessive number of points close to zero. These checks are similar to those in the `mse_classification` algorithm (Algorithm 8). If the signal is recognized as "bad", the algorithm does not perform classification and returns the list ["Bad signal", "Nan"].

Addition of signal to base length: If the signal passes the pre-check, it is augmented with zeros to the base length `leng` using the function `pad_array` (Algorithm 6). This is performed to standardize the length of the input signals before classification.

Parallel MSE optimization for each error type: For each error type in the `errors` list, the algorithm runs parallel optimization of the parameters (size and shift) using the `optimize_params` function (Algorithm 10). Parallel execution is provided by using the `Parallel` function from the `joblib` library with the parameter `n_jobs = -1`, which specifies the use of all available CPU cores. The `delayed` function from `joblib` is used to "delay" the call to `optimize_params` for each error type, allowing `Parallel` to distribute the computation among the cores. The result of the parallel execution is a list of `best_losses`, where each element represents the best MSE value found for the corresponding error type.

Selecting the best error type based on the minimum MSE: After the parallel optimization is complete, the algorithm finds the index of the minimum value in the list of `best_losses` using the `argmin` function. This index `best_loss_idx` indicates the error type that provided the lowest MSE under the optimal parameters. The name of the best error type `best_error` and the corresponding minimum MSE value `best_loss` are extracted from the lists `errors` and `best_losses`, respectively, using the index found.

Result Generation: A `result` list is generated containing two elements: the name of the best error type `best_error` and the corresponding minimum MSE value `best_loss`.

Output data: The algorithm returns a list of `result`, which contains the name of the error type classified as the most likely for the input signal `y` and the minimum MSE value achieved for that error type. In case the input signal was recognized as "bad", the algorithm returns ["Bad signal", "Nan"].

Remarks

The `classification_parallel` algorithm is an efficient method for classifying error types, combining automatic parameter optimization using Optuna and parallel computation to speed up the process. Parallelization is particularly important because optimizing parameters for each error type can be computationally expensive. The selection of the error type with the lowest MSE as the most likely error type is based on the assumption that the error type that can be best "fitted" to the signal `y` (in the sense of MSE) and is the most likely error type present in the signal. The efficiency of classification depends on the adequacy of the set of error types `errors` and the quality of parameter optimization.

3. Experiments and Analysis of Results

To evaluate the effectiveness of the proposed method, an experiment was conducted in which a signal containing several different failures was analyzed based on synthetic data. The input data were fed to the input of the algorithm, sequentially passing through the stages of localization and classification.

In the first stage, localization of the faults was performed based on partitioning the signal into intervals potentially containing anomalies. For this purpose, a wavelet transform was used to identify areas with characteristic changes associated with a violation of the normal behavior of the signal.

The second stage involved classification of the identified anomalies, implemented using numerical optimization methods. Classification takes place in a multithreaded mode, where each thread minimizes the MSE for a specific interval based on a library of synthetic failures. The mean square deviation (MSE) calculated between signal fragments and reference patterns of known failure types was used as a similarity criterion.

The following criteria were used to evaluate the detection success:

1. For each failure actually present in the signal, the method must produce a low MSE value with the corresponding reference failure class.

2. The method must not exhibit low MSE values for classes not present in the analyzed signal.

If both conditions were met, the detection was considered successful. Thus, the experiment allowed for not only testing the ability of the method to correctly identify failures but also evaluating its robustness to possible false positives.

A signal containing two types of anomalies, haar and algb, was chosen as the test signal for this experiment (Figure 2). These types of failures are characterized by different features: haar represents sharp jumps or dips in the data while algb reflects a failure similar to the appearance of some polynomial.

After preprocessing, including noise suppression and removal of the sinusoidal component, anomaly detection techniques were applied to the signal. The purpose of this step was to partition the signal into intervals with a high probability of containing failures (Figure 2).

Figure 3 shows the intervals with possible failures based on the grouping of intervals after performing the wavelet transform. Each interval is expected to contain an anomaly because the interval boundaries are based on the peaks that were obtained after WT.

The results of the algorithms are presented in Table 1. The analysis of the table shows that, for all intervals containing failures of haar and algb types, the algorithms demonstrated successful recognition. The quantitative quality assessment expressed through the mean square error (MSE) confirms this conclusion: for all correctly identified haar and algb failures, the MSE value does not exceed 0.5.

Table 1. Results.

Interval	Fault Type	Error (MSE)
1	Bad signal	Nan
2	Invexp	0.739
3	Invexp	0.927
4	haar	0.448
5	haar	1.644
6	haar	1.662
7	haar	1.209
8	Bad signal	Nan
9	Bad signal	Nan
10	Bad signal	Nan
11	haar	0.383
12	haar	1.505
13	Invexp	3.785
14	Bad signal	Nan
15	Bad signal	Nan
16	Invexp	0.696
17	algb	2.142

Table 1. *Cont.*

Interval	Fault Type	Error (MSE)
18	Invexp	1.922
19	Bad signal	Nan
20	Invexp	0.481
21	-exp	1.902
22	Invexp	1.696
23	Bad signal	Nan
24	algb	0.209
25	algb	3.187×10^{-13}
26	algb	0.251
27	algb	2.174×10^{-13}
28	Bad signal	Nan

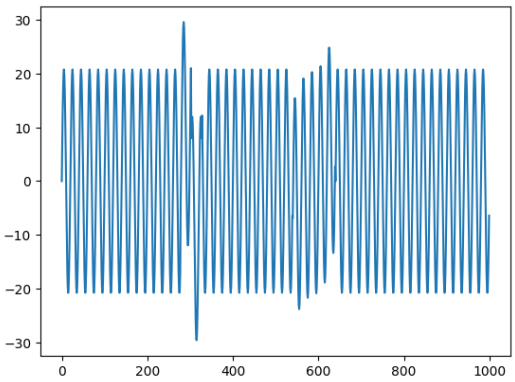

Figure 2. Signal with faults.

However, in addition to the true positives, there was one false positive case associated with the Invexp algorithm. The interval erroneously classified as containing an anomaly actually has no pronounced signs of failure.

This problem can be solved in several ways:

1. Increasing the number of iterations: When optimizing the parameters of the Invexp algorithm (or any other algorithm prone to false positives), a larger number of iterations can lead to a more accurate minimization of the target function and, consequently, a lower probability of false positives.

2. Reducing the tolerance threshold: Currently, the MSE threshold for classifying an interval as anomalous is set at 0.5. Lowering this threshold (e.g., to 0.4) would increase the rigor of the selection criteria and may reduce the number of false positives. However, this approach requires caution, as too strict a threshold may lead to missing real failures. Improving the quality of signal recognition by increasing the number of iterations, as described in (1), will allow for lowering the threshold without losing true positives.

3. Increasing the level of detail of the wavelet transform: A higher level of detail can be used for wavelet analysis. This will detect more subtle features of the signal, which can help to distinguish between normal areas and anomalies more accurately.

4. Using different wavelets: Different wavelets have different sensitivities to different types of anomalies. Experimenting with different wavelet families (e.g., Morlet, Mexican Hat, Dobeshi, etc.) can lead to better recognition performance for particular types of signals and faults.

Thus, despite the presence of false positives, the experimental results generally demonstrate the effectiveness of the proposed approach for detecting haar- and algb-type failures in a noisy signal with a sinusoidal component.

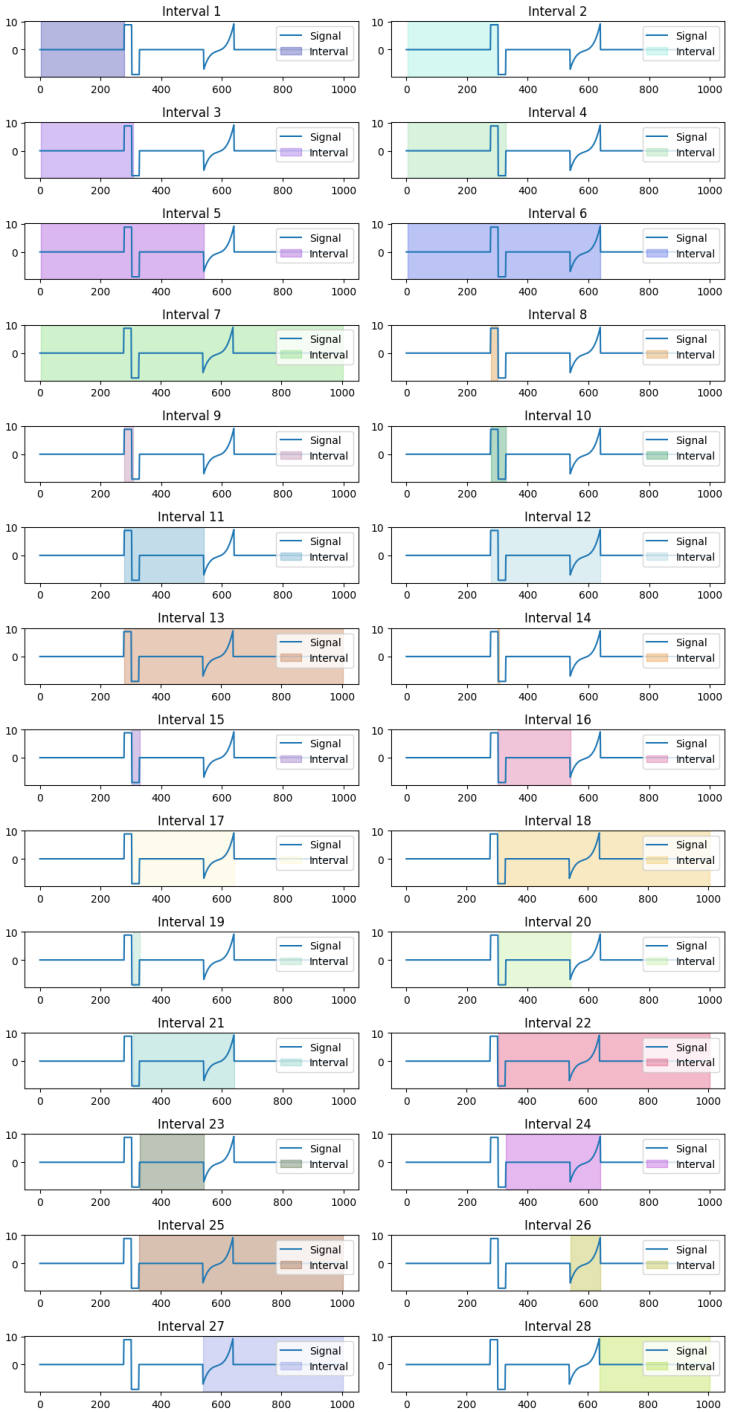

Figure 3. Intervals of signal with faults.

4. Discussion

The presented algorithms form a comprehensive system for classifying error types in one-dimensional signals. The central element of the system is the `classification_parallel` algorithm, which automates the classification process using parameter optimization and parallel computation to improve efficiency. The approach is based on the idea of comparing the input signal with a set of model "error" signals of different types and selecting the type that provides the best match in terms of mean square error (MSE).

4.1. Strengths of the Approach

One of the key advantages of the developed system is its automation. The use of optimizers to find the optimal parameters (size and shift) of the error signal eliminates the need for manual tuning and allows for efficient exploration of the parameter space. Parallelizing the optimization process with `joblib` significantly reduces computation time, making the system applicable to problems requiring fast data processing.

The use of wavelet decomposition in the `intervals_errors` algorithm, although not central to the classification of error types, demonstrates the ability to integrate different signal processing techniques to improve the analysis and detection of errors at different frequency levels. The `find_peaks` algorithm provides an effective tool for detecting characteristic features in signals that may be associated with errors or anomalies.

The modularity of the developed algorithms, including auxiliary functions such as `pad_array` and `resize_vector`, increases the flexibility and reusability of the system. The `mse_classification` function, although replaced by the more efficient `classification_parallel`, demonstrates the basic MSE-based classification approach that underlies the entire system. The `clean_signal` function shows the possibility of preprocessing the signal to remove known types of noise, which can improve the quality of classification.

4.2. Limitations and Areas for Improvement

Despite the advantages, the proposed system has some limitations and areas for further improvement.

The computational complexity of the algorithms, especially the `classification_parallel` algorithms, can be significant due to the nested optimization loops and the need to compute MSEs for a large number of combinations of error types, sizes, and shifts. Further optimization of the code, perhaps through the use of more efficient optimization algorithms or methods to speed up MSE computation, may be beneficial.

The quality of classification depends directly on the representativeness of the set of model "error" signals `Error_signal`. If the real data error types are not covered by the proposed set, the classification accuracy may decrease. Extending the `Error_signal` library by adding new, more realistic error models, as well as adapting the models to specific applications, is an important direction for future research.

Sensitivity to parameters of algorithms, such as thresholds in `find_peaks`, wavelet decomposition parameters, and the number of Optuna iterations, can affect classification results. Further work is needed to investigate the robustness of the system to parameter changes and to develop recommendations for selecting optimal parameter values for different types of signals and tasks. In particular, the automatic adaptation of the parameters, possibly based on the characteristics of the input signal, may improve the practical applicability of the system.

The use of MSE as the only quality metric may be a limitation. Depending on the classification task, other metrics such as accuracy, completeness, F1-measure, or other measures specific to the anomaly detection task may be more appropriate for evaluating classification quality. Investigating and comparing different metrics and their impact on classification performance is an interesting direction for future research.

5. Conclusions

The presented system of algorithms provides an efficient and automated approach to classifying error types in one-dimensional signals. Key advantages are the use of optimization techniques and parallel computing for speedup. The system shows potential for applications in various signal processing areas where the automatic detection and

classification of anomalies or defects are required, such as equipment diagnostics and sensor data monitoring.

Despite the results achieved, further research could focus on improving computational efficiency, expanding the model error library, investigating robustness to parameters, and exploring alternative classification quality metrics. Improvements in these areas could make the system an even more powerful and versatile tool for analyzing and interpreting complex signal data, contributing to advances in automatic diagnosis and anomaly detection tasks in various fields of science and engineering.

Author Contributions: Data curation, investigation, software—N.S.; conceptualization, methodology, validation—D.A. and N.S.; formal analysis, project administration, writing—original draft—E.P.; supervision, writing—review and editing—S.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Science and Higher Education of the Russian Federation grant no. 075-15-2024-525.

Data Availability Statement: The data that support the findings of this study are openly available at https://github.com/NekkittAY/WONC-FD-Method (accessed on 23 January 2025).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wei, X.-L.; Zhang, C.X.; Kim, S.W.; Jing, K.L.; Wang, Y.J.; Xu, S.; Xie, Z.Z. Seismic fault detection using convolutional neural networks with focal loss. *Comput. Geosci.* **2022**, *158*, 104968. [CrossRef]
2. Iqbal, N. DeepSeg: Deep segmental denoising neural network for seismic data. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *34*, 3397–3404. [CrossRef] [PubMed]
3. Mahdavi, A.; Kahoo, A.R.; Radad, M.; Monfared, M.S. Application of the local maximum synchrosqueezing transform for seismic data. *Digit. Signal Process.* **2021**, *110*, 102934. [CrossRef]
4. Bykov, A.; Grecheneva, A.; Kuzichkin, O.; Surzhik, D.; Vasilyev, G.; Yerbayev, Y. Mathematical description and laboratory study of electrophysical methods of localization of geodeformational changes during the control of the railway roadbed. *Mathematics* **2021**, *9*, 3164. [CrossRef]
5. Björck, A. *Numerical Methods for Least Squares Problems*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2024.
6. Misra, S.; Kumar, S.; Sayyad, S.; Bongale, A.; Jadhav, P.; Kotecha, K.; Abraham, A.; Gabralla, L.A. Fault detection in induction motor using time domain and spectral imaging-based transfer learning approach on vibration data. *Sensors* **2022**, *22*, 8210. [CrossRef]
7. Peng, Y.; Qiao, W.; Cheng, F.; Qu, L. Wind turbine drivetrain gearbox fault diagnosis using information fusion on vibration and current signals. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 3518011. [CrossRef]
8. Tayyab, S.M.; Chatterton, S.; Pennacchi, P. Intelligent defect diagnosis of rolling element bearings under variable operating conditions using convolutional neural network and order maps. *Sensors* **2022**, *22*, 2026. [CrossRef]
9. Osipov, A.V.; Pleshakova, E.S.; Gataullin, S.T. Production processes optimization through machine learning methods based on geophysical monitoring data. *Comput. Opt.* **2024**, *48*, 633–642. [CrossRef]
10. Dashti, R.; Daisy, M.; Mirshekali, H.; Shaker, H.R.; Aliabadi, M.H. A survey of fault prediction and location methods in electrical energy distribution networks. *Measurement* **2021**, *184*, 109947. [CrossRef]
11. Xie, H.; Jiang, M.; Zhang, D.; Goh, H.H.; Ahmad, T.; Liu, H.; Liu, T.; Wang, S.; Wu, T. IntelliSense technology in the new power systems. *Renew. Sustain. Energy Rev.* **2023**, *177*, 113229. [CrossRef]
12. Xu, W.; Wu, X.; Li, Y.; Wang, H.; Lu, L.; Ouyang, M. A comprehensive review of DC arc faults and their mechanisms, detection, early warning strategies, and protection in battery systems. *Renew. Sustain. Energy Rev.* **2023**, *186*, 113674. [CrossRef]
13. Nelder, J.A.; Mead, R. A simplex method for function minimization. *Comput. J.* **1965**, *7*, 308–313. [CrossRef]
14. Sun, G.; Wang, Y.; Luo, Q.; Li, Q. Vibration-based damage identification in composite plates using 3D-DIC and wavelet analysis. *Mech. Syst. Signal Process.* **2022**, *173*, 108890. [CrossRef]
15. Almounajjed, A.; Sahoo, A.K.; Kumar, M.K. Diagnosis of stator fault severity in induction motor based on discrete wavelet analysis. *Measurement* **2021**, *182*, 109780. [CrossRef]

16. Yan, R.; Shang, Z.; Xu, H.; Wen, J.; Zhao, Z.; Chen, X.; Gao, R.X. Wavelet transform for rotary machine fault diagnosis: 10 years revisited. *Mech. Syst. Signal Process.* **2023**, *200*, 110545. [CrossRef]
17. Martinez-Ríos, E.A.; Bustamante-Bello, R.; Navarro-Tuch, S.; Perez-Meana, H. Applications of the generalized Morse wavelets: A review. *IEEE Access* **2022**, *11*, 667–688. [CrossRef]
18. Liu, C.; Zhuo, F.; Wang, F. Fault diagnosis of commutation failure using wavelet transform and wavelet neural network in HVDC transmission system. *IEEE Trans. Instrum. Meas.* **2021**, *70*, 3525408. [CrossRef]
19. Wang, M.-H.; Lu, S.-D.; Liao, R.-M. Fault diagnosis for power cables based on convolutional neural network with chaotic system and discrete wavelet transform. *IEEE Trans. Power Deliv.* **2021**, *37*, 582–590. [CrossRef]
20. Gao, J.; Wang, X.; Wang, X.; Yang, A.; Yuan, H.; Wei, X. A high-impedance fault detection method for distribution systems based on empirical wavelet transform and differential faulty energy. *IEEE Trans. Smart Grid* **2021**, *13*, 900–912. [CrossRef]
21. Baloch, S.; Samsani, S.S.; Muhammad, M.S. Fault protection in microgrid using wavelet multiresolution analysis and data mining. *IEEE Access* **2021**, *9*, 86382–86391. [CrossRef]
22. Ma, Y.; Maqsood, A.; Oslebo, D.; Corzine, K. Wavelet transform data-driven machine learning-based real-time fault detection for naval DC pulsating loads. *IEEE Trans. Transp. Electrif.* **2021**, *8*, 1956–1965. [CrossRef]
23. Shakiba, F.M.; Azizi, S.M.; Zhou, M.; Abusorrah, A. Application of machine learning methods in fault detection and classification of power transmission lines: A survey. *Artif. Intell. Rev.* **2023**, *56*, 5799–5836. [CrossRef]
24. Cano, A.; Arévalo, P.; Benavides, D.; Jurado, F. Integrating discrete wavelet transform with neural networks and machine learning for fault detection in microgrids. *Int. J. Electr. Power Energy Syst.* **2024**, *155*, 109616. [CrossRef]
25. Nsaif, Y.M.; Hossain Lipu, M.S.; Hussain, A.; Ayob, A.; Yusof, Y.; Zainuri, M.A.A. A novel fault detection and classification strategy for photovoltaic distribution network using improved Hilbert-Huang transform and ensemble learning technique. *Sustainability* **2022**, *14*, 11749. [CrossRef]
26. Branco, N.W.; Cavalca, M.S.M.; Stefenon, S.F.; Leithardt, V.R.Q. Wavelet LSTM for fault forecasting in electrical power grids. *Sensors* **2022**, *22*, 8323. [CrossRef]
27. Andriyanov, N.; Khasanshin, I.; Utkin, D.; Gataullin, T.; Ignar, S.; Shumaev, V.; Soloviev, V. Intelligent system for estimation of the spatial position of apples based on YOLOv3 and RealSense depth camera D415. *Symmetry* **2022**, *14*, 148. [CrossRef]
28. Ivanyuk, V. Forecasting of digital financial crimes in Russia based on machine learning methods. *J. Comput. Virol. Hacking Tech.* **2024**, *20*, 349–362. [CrossRef]
29. Boltachev, E. Potential cyber threats of adversarial attacks on autonomous driving models. *J. Comput. Virol. Hacking Tech.* **2023**, *20*, 363–373. [CrossRef]
30. Efanov, D.; Aleksandrov, P.; Mironov, I. Comparison of the effectiveness of cepstral coefficients for Russian speech synthesis detection. *J. Comput. Virol. Hacking Tech.* **2024**, *20*, 375–382. [CrossRef]
31. Lu, H.; Tang, H.; Wang, Z. *Advances in Neural Networks–ISNN 2019: 16th International Symposium on Neural Networks*; LNCS 11554; Springer: Berlin/Heidelberg, Germany, 2019.
32. Mohan, V.; Senthilkumar, S. IoT based fault identification in solar photovoltaic systems using an extreme learning machine technique. *J. Intell. Fuzzy Syst.* **2022**, *43*, 3087–3100. [CrossRef]
33. Makarov, V.L.; Bakhtizin, A.R.; Sushko, E.D.; Sushko, G.B. Creation of a supercomputer simulation of a society with different types of active agents and its approbation. *Her. Russ. Acad. Sci.* **2022**, *92*, 268–275. [CrossRef]
34. Makarov, V.L.; Bakhtizin, A.R.; Hua, L.; Jie, W.; Zili, W.; Sidorenko, M.Y. Long-term demographic forecasting. *Her. Russ. Acad. Sci.* **2023**, *93*, 294–307. [CrossRef]
35. Hari, S.K.S.; Sullivan, M.B.; Tsai, T.; Keckler, S.W. Making convolutions resilient via algorithm-based error detection techniques. *IEEE Trans. Dependable Secur. Comput.* **2021**, *19*, 2546–2558. [CrossRef]
36. Petushkov, G.V.; Sigov, A.S. Analysis and selection of the structure of a multiprocessor computing system according to the performance criterion. *Russ. Technol. J.* **2024**, *12*, 20–25. [CrossRef]
37. Tulbure, A.-A.; Tulbure, A.A.; Dulf, E.H. A review on modern defect detection models using DCNNs–Deep convolutional neural networks. *J. Adv. Res.* **2022**, *35*, 33–48. [CrossRef] [PubMed]
38. Fernandes, M.; Corchado, J.M.; Marreiros, G. Machine learning techniques applied to mechanical fault diagnosis and fault prognosis in the context of real industrial manufacturing use-cases: A systematic literature review. *Appl. Intell.* **2022**, *52*, 14246–14280. [CrossRef]
39. Al-Andoli, M.N.; Tan, S.C.; Sim, K.S.; Seera, M.; Lim, C.P. A parallel ensemble learning model for fault detection and diagnosis of industrial machinery. *IEEE Access* **2023**, *11*, 39866–39878. [CrossRef]
40. Shashoa, N.A.A.; Jomah, O.S.; Abusaeeda, O.; Elmezughi, A.S. Feature selection for fault diagnosis using principal component analysis. In Proceedings of the 2023 58th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST), Nis, Serbia, 29 June–1 July 2023; pp. 39–42.
41. Jawad, R.; Abid, H. HVDC fault detection and classification with artificial neural network based on ACO-DWT method. *Energies* **2023**, *16*, 1064. [CrossRef]

42. Fu, S.; Wu, Y.; Wang, R.; Mao, M. A bearing fault diagnosis method based on wavelet denoising and machine learning. *Appl. Sci.* **2023**, *13*, 5936. [CrossRef]
43. Jamil, M.; Sharma, S.; Singh, R. Fault detection and classification in electrical power transmission system using artificial neural network. *SpringerPlus* **2015**, *4*, 334. [CrossRef]
44. Geiger, A.; Liu, D.; Alnegheimish, S.; Cuesta-Infante, A.; Veeramachaneni, K. Tadgan: Time series anomaly detection using generative adversarial networks. In Proceedings of the 2020 IEEE International Conference On Big Data (Big Data), Atlanta, GA, USA, 10–13 December 2020; pp. 33–43.
45. Sundararaman, B.; Jain, P. Fault detection and classification in electrical power transmission system using wavelet transform. *Eng. Proc.* **2023**, *59*, 71. [CrossRef]
46. Nasser Mohamed, Y.; Seker, S.; Akinci, T. Signal processing application based on a hybrid wavelet transform to fault detection and identification in power system. *Information* **2023**, *14*, 540. [CrossRef]
47. Han, D. Fault Diagnosis and Its Applications to Fault Tolerant Control of a Turbojet Engine. *Energies* **2023**, *16*, 3317. [CrossRef]
48. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for hyper-parameter optimization. *Adv. Neural Inf. Process. Syst.* **2011**, *24*, 2546–2554.
49. Tama, B.A.; Vania, M.; Lee, S.; Lim, S. Recent advances in the application of deep learning for fault diagnosis of rotating machinery using vibration signals. *Artif. Intell. Rev.* **2023**, *56*, 4667–4709. [CrossRef]
50. Nocedal, J.; Wright, S.J. *Numerical Optimization*; Springer: Berlin/Heidelberg, Germany, 1999.
51. Guo, T.; Zhang, T.; Lim, E.; Lopez-Benitez, M.; Ma, F.; Yu, L. A review of wavelet analysis and its applications: Challenges and opportunities. *IEEE Access* **2022**, *10*, 58869–58903. [CrossRef]
52. Beylkin, G.; Coifman, R.; Rokhlin, V. Fast wavelet transforms and numerical algorithms I. *Commun. Pure Appl. Math.* **1991**, *44*, 141–183. [CrossRef]