

Algorithmic Optimization and Regression Validation of the RPD Index Implementation

Marco Righi 

11 August 2026



**Consiglio Nazionale
delle Ricerche**

National Research Council of Italy

Corresponding author:

Marco Righi
marco.righi@cnr.it

Authors, ORCID iD, email, and institutional affiliation:

Marco Righi:  <https://orcid.org/0000-0002-1448-0960>; marco.righi@cnr.it; National Research Council of Italy

Contents

Keywords	3
Abstract	3
1 Introduction	4
1.1 Purpose of the Report	4
1.2 Scope	4
2 Reference Software and Compatibility Target	4
2.1 Historical Reference Implementation	4
2.2 Implemented RPD Functions	5
2.3 Declared Compatibility Domain	5
3 Algorithmic Reformulation	5
3.1 Local Indices	5
3.2 Why the Historical Global Algorithms Are Quadratic	6
3.3 Coordinate Compression	6
3.4 Recovering Pairwise Magnitude Sums	6
3.5 Optimized RPDgep	7
3.6 Optimized RPDgea	7
3.7 Optimized RPDgepr	7
3.8 Progressive Trajectories	7
4 Computational Complexity	8
5 Regression Validation	8
5.1 Validation Objective	8
5.2 Deterministic Test Design	8
5.3 Floating-Point Criterion	9
5.4 Validation Results	9
6 Preliminary Performance Evaluation	9
6.1 Benchmark Method	9
6.2 Observed Timings	10
7 Software Repository, Release, and Licensing	10
7.1 Separate Repository Strategy	10

7.2	Repository Organization	11
7.3	Licensing	11
8	Reproducibility	11
8.1	Regression Test	11
8.2	Benchmark	11
8.3	Minimum Provenance to Record	12
9	Limitations and Interpretation	12
9.1	Regression Evidence Is Not a Universal Proof	12
9.2	Floating-Point Reordering	12
9.3	Special Numerical Values	12
9.4	Progressive Reverse Variants	13
10	Declaration of Generative AI and AI-Assisted Technologies	13
11	Conclusions	13
	Acknowledgements	14
A	Appendix: Validation Output	14
B	Appendix: Reproducibility Checklist	14

Keywords

RPD index; research software; algorithm optimization; Fenwick tree; regression testing; reproducibility; artificial intelligence

Abstract

This technical report documents the development of `RPDindexOptimized`, a separate optimized implementation of the six RPD indices previously distributed through the `RPDindex` research-software repository. The historical implementation is intentionally preserved as the archival reference associated with Zenodo DOI [10.5281/zenodo.20313771](https://doi.org/10.5281/zenodo.20313771). The new implementation preserves the intended numerical semantics for finite Python `int/float` sequences of length at least two, while reducing the asymptotic complexity of the three global indices from $O(n^2)$ to $O(n \log n)$ by using coordinate compression and Fenwick trees for cumulative counts and sums. A deterministic regression suite performs 52,248 reference-versus-optimized comparisons, and an additional 4,848 checks validate progressive-prefix trajectories. All tests pass within explicit floating-point tolerances; the largest observed absolute discrepancy is approximately 1.11×10^{-15} . A preliminary benchmark confirms the expected computational advantage of the optimized global algorithms as sequence length grows. The new software is independently archived as `RPDindexOptimized v1.0.0` with Zenodo DOI [10.5281/zenodo.21892420](https://doi.org/10.5281/zenodo.21892420). The report also documents repository organization, licensing, reproducibility procedures, compatibility limits, and the use of generative artificial intelligence as an assistive software-engineering and drafting tool under the sole author's responsibility.

1 Introduction

1.1 Purpose of the Report

The RPD index family was originally implemented in the research-software repository `RPDindex`. That implementation is archived on Zenodo and constitutes the historical software artifact associated with the original RPD work [1]. The objective of the present work was not to redefine the RPD indices or retrospectively alter that archived artifact. Instead, the objective was to create a separate, traceable software implementation with lower computational cost while preserving the numerical semantics of the original code over an explicitly declared compatibility domain.

This decision led to the creation of a distinct repository, `RPDindexOptimized`, rather than modifying the historical repository in place. The optimized release is archived independently as version 1.0.0 on Zenodo [2]. This separation provides a clear provenance chain between the reference implementation and the optimized implementation and allows independent citation of both software artifacts.

1.2 Scope

The work covered the following activities:

- preservation of the original implementation as an immutable reference baseline;
- analysis of the computational structure of the six RPD functions;
- removal of unnecessary temporary data structures in the local functions;
- replacement of explicit pair enumeration in the global functions by rank-based cumulative data structures;
- addition of a function for computing complete progressive RPD trajectories;
- construction of deterministic regression tests against the reference implementation;
- construction of a reproducible standard-library benchmark;
- creation of a separate GitHub repository, citation metadata, licensing information, and Zenodo release;
- explicit documentation of compatibility limits and floating-point behavior; and
- disclosure of the use of generative AI during software engineering and report preparation.

This report concerns implementation equivalence and computational optimization. It does not introduce a new mathematical RPD index and does not constitute a formal proof of equivalence for arbitrary numerical types or exceptional floating-point values.

2 Reference Software and Compatibility Target

2.1 Historical Reference Implementation

The authoritative historical implementation is available at:

- GitHub: <https://github.com/marcorighi/RPDindex>;
- Zenodo: [10.5281/zenodo.20313771](https://zenodo.org/record/20313771).

The Zenodo record identifies the archived software as *RPDindex: Realistic Performance Determination indices for directional changes in normalized data* [1]. A fixed snapshot of the original `computeindex.py` is also included in the optimized repository under `reference/computeindex_`

`reference.py` solely to provide a reproducible regression baseline. The original repository and DOI remain the authoritative provenance for that historical source.

2.2 Implemented RPD Functions

Both implementations expose the six principal functions

`RPDlep`, `RPDlea`, `RPDlepr`, `RPDgep`, `RPDgea`, and `RPDgepr`.

The optimized module additionally exposes `progressive_indices(elements)`, which returns the six RPD trajectories for all prefixes of length $2, \dots, n$.

Let an ordered numerical sequence be

$$X = (x_1, x_2, \dots, x_n). \quad (1)$$

For every comparison, an increase is treated as positive, a decrease as negative, and equality as neutral. This reproduces the comparison semantics of the original helper `isAworstThanB`: $A < B$ is positive, $A > B$ is negative, and $A = B$ is neutral.

The normalized RPD combination used throughout the optimized implementation can be written as

$$\text{RPD}(P, N) = \begin{cases} \frac{P - N}{P + N}, & P + N > 0, \\ 0, & P + N = 0. \end{cases} \quad (2)$$

Accordingly, the output remains bounded in $[-1, 1]$ whenever $P, N \geq 0$.

2.3 Declared Compatibility Domain

The strict regression-equivalence claim is limited to finite Python `int` and `float` values in sequences satisfying $n \geq 2$. The current validation does not establish behavior for NaN, infinities, missing-value sentinels, NumPy scalar classes, `Decimal`, `Fraction`, or arbitrary user-defined numerical classes.

For $n < 2$, the optimized implementation intentionally regularizes the interface by returning `0.0` from all six public functions. This behavior is outside the strict equivalence claim because the historical implementation does not behave uniformly for these inputs: in particular, the original `RPDlep` and `RPDgep` can divide by zero when no comparisons exist.

3 Algorithmic Reformulation

3.1 Local Indices

The three local indices operate on adjacent differences. The historical implementation already has linear asymptotic complexity, so the optimization is principally a reduction of implementation overhead. Temporary `positive` and `negative` lists and subsequent summation loops are replaced by direct accumulation of counts and magnitudes.

For `RPDlep`, let c_+ and c_- be the counts of strictly positive and strictly negative adjacent transitions, and let

$$S_+ = \sum_{i: x_{i-1} < x_i} (x_i - x_{i-1}), \quad (3)$$

$$S_- = \sum_{i: x_{i-1} > x_i} (x_{i-1} - x_i). \quad (4)$$

The reference implementation multiplies S_+ and S_- by $c_+/(n-1)$ and $c_-/(n-1)$, respectively. Since the common factor $1/(n-1)$ cancels in Eq. (2), the optimized implementation can use

$$P = c_+ S_+, \quad N = c_- S_-. \quad (5)$$

This preserves the mathematical result while avoiding temporary lists.

The weighted local variants RPD1ea and RPD1epr retain the same running comparison counts and denominators as the reference implementation. Their asymptotic complexity therefore remains $O(n)$.

3.2 Why the Historical Global Algorithms Are Quadratic

The three global functions compare each observation against multiple non-adjacent observations. In the reference implementation, these relationships are materialized by nested loops. For a single final value, the number of unordered observation pairs is

$$M = \frac{n(n-1)}{2}, \quad (6)$$

which gives $O(n^2)$ pair processing.

For RPDgep, define the set of positive global pairs as $i < j$ with $x_i < x_j$, and the negative global pairs as $i < j$ with $x_i > x_j$. Let c_+ and c_- denote their counts and

$$S_+ = \sum_{i < j, x_i < x_j} (x_j - x_i), \quad (7)$$

$$S_- = \sum_{i < j, x_i > x_j} (x_i - x_j). \quad (8)$$

As for RPD1ep, the reference factor $1/M$ is common to both weighted terms and cancels in the final normalization. Therefore RPDgep can be reconstructed from c_+ , c_- , S_+ , and S_- without explicitly visiting every pair.

3.3 Coordinate Compression

The optimized global algorithms first sort the distinct observed values and assign each value an integer rank. This coordinate compression preserves ordering while mapping arbitrary finite values to the index domain required by a Fenwick tree.

Two Fenwick trees are maintained [3]:

- a count tree storing the number of observations at each rank;
- a sum tree storing the numerical sum of observations at each rank.

Both prefix queries and point updates have $O(\log n)$ complexity.

3.4 Recovering Pairwise Magnitude Sums

Suppose the current observation is x , and the data structure contains the observations preceding it. Let ℓ be the number of preceding values strictly less than x and $s_<$ their sum. Their complete positive magnitude contribution is

$$\sum_{y < x} (x - y) = x\ell - s_<. \quad (9)$$

Similarly, if g preceding values are strictly greater than x and have sum $s_>$, their negative magnitude contribution is

$$\sum_{y>x} (y - x) = s_> - xg. \quad (10)$$

The Fenwick count and sum trees provide ℓ , g , $s_<$, and $s_>$ in logarithmic time. Equal-valued observations are excluded from the positive and negative counts, thereby retaining their neutral status.

3.5 Optimized RPDgep

While scanning from left to right, Eqs. (9) and (10) provide the count and magnitude contributions of every pair whose second endpoint is the current observation. The algorithm accumulates global c_+ , c_- , S_+ , and S_- and then applies Eq. (2) to $P = c_+S_+$ and $N = c_-S_-$. The resulting complexity is $O(n \log n)$.

3.6 Optimized RPDgea

The RPDgea definition weights comparisons by the current observation/group. For position j , let $c_{j,+}$ and $c_{j,-}$ be the positive and negative comparison counts against positions $1, \dots, j-1$, and let $S_{j,+}$ and $S_{j,-}$ be the corresponding magnitude sums. The reference grouping is retained as

$$P = \sum_{j=2}^n S_{j,+} \frac{c_{j,+}}{j-1}, \quad (11)$$

$$N = \sum_{j=2}^n S_{j,-} \frac{c_{j,-}}{j-1}. \quad (12)$$

The denominator $j-1$ is the total number of earlier comparisons, including ties. This is important: replacing it with $c_{j,+} + c_{j,-}$ would change the semantics when ties are present. Fenwick queries recover each group's counts and sums in $O(\log n)$, giving $O(n \log n)$ overall.

3.7 Optimized RPDgepr

The reverse global variant is reconstructed by scanning the sequence from right to left. At each step the Fenwick trees therefore represent observations occurring later in the original sequence. Later values greater than the current observation contribute to the positive component; later values smaller than the current observation contribute to the negative component. The reference per-group denominator is retained as the complete number of later observations, again including ties. The final complexity is $O(n \log n)$.

3.8 Progressive Trajectories

The added function `progressive_indices(elements)` evaluates all six indices for prefixes

$$X_2, X_3, \dots, X_n, \quad X_k = (x_1, \dots, x_k). \quad (13)$$

It avoids repeatedly invoking all functions from scratch wherever an incremental update is available. The current total complexities for complete trajectories are:

- RPDlep: $O(n)$;
- RPDlea: $O(n)$;

- RPDgep: $O(n \log n)$;
- RPDgea: $O(n \log n)$;
- RPDlepr: $O(n^2)$;
- RPDgepr: $O(n^2)$.

The reverse-weighted progressive forms remain quadratic because appending a new endpoint changes the weighting of earlier reverse groups. The current implementation updates or recomputes those quantities without changing their mathematical definition.

4 Computational Complexity

Table 1 summarizes the asymptotic costs for one final index value.

Table 1: Asymptotic complexity of the historical and optimized implementations for one final index value.

Function	Reference	Optimized
RPDlep	$O(n)$	$O(n)$
RPDlea	$O(n)$	$O(n)$
RPDlepr	$O(n)$	$O(n)$
RPDgep	$O(n^2)$	$O(n \log n)$
RPDgea	$O(n^2)$	$O(n \log n)$
RPDgepr	$O(n^2)$	$O(n \log n)$

The local optimizations improve constants and memory behavior but do not alter asymptotic order. The primary algorithmic contribution is the reduction of each global final-value function from quadratic to quasi-linear time.

5 Regression Validation

5.1 Validation Objective

The validation objective was to establish executable regression evidence that the optimized public functions reproduce the outputs of the archival reference implementation over the declared compatibility domain. This is deliberately described as regression evidence rather than a formal proof for all possible inputs.

5.2 Deterministic Test Design

The validation script is stored as `tests/test_equivalence.py`. It uses pseudo-random seed

$$\text{SEED} = 20260811 \quad (14)$$

and evaluates:

- 8 fixed sequences covering increasing, decreasing, constant, mixed, negative, fractional, tied, and strongly unbalanced-magnitude cases;
- 300 pseudorandom sequences for every length from 2 through 30;
- all six RPD functions on every sequence.

The pseudorandom values are generated as integers in $[-10, 10]$ divided by 3, intentionally producing negative values, fractions, and ties. The resulting number of reference-versus-optimized function comparisons is 52,248.

The same script also validates `progressive_indices()` by comparing every generated trajectory against repeated prefix evaluation of the optimized scalar functions. This produces a further 4,848 progressive consistency checks.

5.3 Floating-Point Criterion

The test compares historical and optimized values using

Listing 1: Tolerance used for regression comparison.

```
math.isclose(reference, optimized,
              rel_tol=1e-12,
              abs_tol=1e-12)
```

A tolerance is required because the optimized global algorithms accumulate mathematically equivalent quantities in a different order. Binary floating-point addition is not associative, so bitwise identity is not an appropriate compatibility requirement.

5.4 Validation Results

All 52,248 reference-versus-optimized comparisons and all 4,848 progressive checks passed. Table 2 reports the maximum observed absolute discrepancy for each scalar function.

Table 2: Maximum absolute discrepancy observed in deterministic regression validation.

Function	Maximum absolute discrepancy
RPDlep	$2.2204460492503131 \times 10^{-16}$
RPDlea	0
RPDlepr	0
RPDgep	$1.1102230246251565 \times 10^{-15}$
RPDgea	$3.8857805861880479 \times 10^{-16}$
RPDgepr	$3.8857805861880479 \times 10^{-16}$

These differences are several orders of magnitude below the explicit 10^{-12} absolute/relative tolerance and are consistent with a change in floating-point accumulation order rather than a change in the intended formulas.

6 Preliminary Performance Evaluation

6.1 Benchmark Method

A standard-library benchmark is provided in `benchmarks/benchmark.py`. For each requested sequence length, a deterministic pseudo-random sequence is generated and each reference and optimized function is timed repeatedly using `time.perf_counter()`. The reported value is the median elapsed time.

A preliminary execution performed during preparation of this report used Python 3.13.5 on Linux 6.18.35, x86_64, seed 20260811, and three repetitions per measurement. These measurements

are included to confirm the expected scaling behavior, not as hardware-independent performance guarantees.

6.2 Observed Timings

Table 3 reports the measured median execution times and reference/optimized speedup ratios.

Table 3: Preliminary reference-versus-optimized benchmark. Times are median seconds over three runs in the stated execution environment.

n	Function	Reference (s)	Optimized (s)	Speedup
100	RPDlep	0.000029391	0.000010007	2.94
100	RPDlea	0.000038213	0.000014573	2.62
100	RPDlepr	0.000031918	0.000016606	1.92
100	RPDgep	0.00137731	0.000306155	4.50
100	RPDgea	0.00120634	0.000311602	3.87
100	RPDgepr	0.00172103	0.000338047	5.09
300	RPDlep	0.000080675	0.000023598	3.42
300	RPDlea	0.000097264	0.000044730	2.17
300	RPDlepr	0.000095234	0.000048577	1.96
300	RPDgep	0.0122294	0.00121029	10.11
300	RPDgea	0.0110715	0.00115752	9.57
300	RPDgepr	0.0115906	0.00112452	10.31
1000	RPDlep	0.000290103	0.000081050	3.58
1000	RPDlea	0.000353950	0.000145347	2.44
1000	RPDlepr	0.000365531	0.000184629	1.98
1000	RPDgep	0.241338	0.00452406	53.35
1000	RPDgea	0.141396	0.00479409	29.49
1000	RPDgepr	0.147737	0.00459679	32.14

The local functions show moderate constant-factor improvements. The global functions show increasing speedup with sequence length, which is consistent with the change from $O(n^2)$ to $O(n \log n)$. Exact ratios will vary with processor, interpreter, operating system, system load, and input distribution.

7 Software Repository, Release, and Licensing

7.1 Separate Repository Strategy

The optimized implementation was intentionally published in a new repository rather than committed as a replacement for the historical implementation. This preserves the software object already associated with the original scholarly record and makes the optimization independently auditable.

The optimized repository is:

- GitHub: <https://github.com/marcorighi/RPDIndexOptimized>;
- release tag: v1.0.0;
- Zenodo DOI: [10.5281/zenodo.21892420](https://doi.org/10.5281/zenodo.21892420).

The exact Git commit corresponding to the deposited release can be recovered from the GitHub tag and should be recorded whenever the software is used in a reproducibility-critical workflow.

7.2 Repository Organization

The repository separates implementation, reference baseline, tests, benchmarks, documentation, and technical-report material:

Listing 2: Repository structure.

```
RPDindexOptimized/  
|-- README.md  
|-- CHANGELOG.md  
|-- CITATION.cff  
|-- VALIDATION.md  
|-- LICENSE  
|-- DOCUMENTATION_LICENSE.md  
|-- src/  
|   '-- computeindex_optimized.py  
|-- reference/  
|   |-- README.md  
|   '-- computeindex_reference.py  
|-- tests/  
|   '-- test_equivalence.py  
|-- benchmarks/  
|   '-- benchmark.py  
|-- docs/  
|   '-- OPTIMIZATION.md  
'-- technical_report/
```

7.3 Licensing

The optimized software source code is released under the BSD 3-Clause License. Scientific documentation, explanatory material, and this technical report are intended for release under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. The archival copy of the historical reference implementation remains attributable to its original source and archival record.

This licensing arrangement separates the conventional open-source software license used for the new implementation from the attribution-oriented license used for scientific documentation. Citation metadata are provided in `CITATION.cff`.

8 Reproducibility

8.1 Regression Test

From the repository root, the complete deterministic regression suite is executed with:

```
python3 tests/test_equivalence.py
```

A successful execution reports 52,248 passed reference-versus-optimized comparisons and 4,848 passed progressive consistency comparisons, followed by the maximum discrepancy observed for each function.

8.2 Benchmark

A baseline benchmark can be executed with:

```
python3 benchmarks/benchmark.py
```

or with explicit sequence sizes and repetitions:

```
python3 benchmarks/benchmark.py \  
--sizes 100 300 1000 \  
--repeats 3
```

Performance results should always be archived together with Python version, operating system, hardware information, seed, command line, and raw output.

8.3 Minimum Provenance to Record

For use in subsequent experiments or publications, the following provenance information should be retained:

- software name: RPDIndexOptimized;
- release/tag: v1.0.0 or later explicit release;
- Git commit hash;
- Zenodo software DOI;
- Python version;
- operating system and hardware for performance claims;
- dataset and preprocessing version;
- regression-test output; and
- benchmark raw output when computational performance is discussed.

9 Limitations and Interpretation

9.1 Regression Evidence Is Not a Universal Proof

The validation suite is extensive relative to the intended compatibility domain but remains finite. Passing the deterministic suite demonstrates agreement on the tested cases and supports the engineering equivalence claim; it does not prove equivalence for every possible floating-point bit pattern or every Python numerical class.

9.2 Floating-Point Reordering

The global optimization changes the order in which contributions are summed. Therefore tiny floating-point deviations are expected even when the real-number expressions are mathematically equivalent. Reproducibility checks should use stated numerical tolerances rather than bitwise equality unless a later implementation explicitly guarantees a fixed summation strategy.

9.3 Special Numerical Values

No compatibility claim is currently made for NaN, positive or negative infinity, or missing-value encodings. These values can affect sorting, equality, ordering, and arithmetic in ways that require an explicit policy rather than implicit acceptance.

9.4 Progressive Reverse Variants

Although all three global scalar functions are reduced to $O(n \log n)$, complete progressive trajectories for RPDlepr and RPDgepr remain $O(n^2)$ in the current implementation. This is an explicit implementation limitation and a possible target for future work if an exact incremental reformulation is identified.

10 Declaration of Generative AI and AI-Assisted Technologies

Generative artificial intelligence was used as an assistive tool during the development and documentation of RPDindexOptimized. Specifically, OpenAI ChatGPT (GPT-5.6 Sol) was used interactively to assist with analysis of the historical Python implementation, identification and formulation of the Fenwick-tree optimization strategy, drafting and refactoring of Python code, construction of regression and benchmark scripts, preparation of repository documentation and deployment material, and drafting and language revision of this technical report.

The AI system was not treated as an author and did not independently establish the scientific validity of the work. The scientific objective, compatibility requirements, release strategy, and decision to preserve the original software as a distinct archival reference were under the responsibility of the human author. Numerical equivalence claims reported in this document are based on executable comparisons between the historical and optimized implementations, not on unverified AI-generated assertions. The sole human author retains responsibility for the final source code, interpretation of the validation evidence, repository contents, software release, licensing choices, and the contents of this report.

No claim is made that generative AI outputs are intrinsically reproducible across model versions or sessions. For this reason, the software artifact, tests, benchmark scripts, versioned repository, and archived Zenodo release constitute the reproducible technical record; the AI interaction itself is not required to execute or validate the released implementation.

11 Conclusions

The work produced a new, independently archived implementation of the six RPD indices without altering the historical software artifact. The three local scalar functions retain $O(n)$ complexity with reduced implementation overhead, while the three global scalar functions are reformulated from explicit $O(n^2)$ pair enumeration to $O(n \log n)$ rank-based accumulation using Fenwick trees. Deterministic regression validation produced 52,248 successful reference-versus-optimized comparisons and 4,848 successful progressive consistency checks, with maximum numerical discrepancies on the order of 10^{-15} . Preliminary timing measurements show increasing acceleration for the optimized global functions as n grows.

The principal result is therefore an engineering optimization with explicit provenance, a bounded compatibility claim, executable regression evidence, independent GitHub/Zenodo versioning, and documented use of AI-assisted software engineering. Future work may extend the compatibility domain, add dataset-specific validation, investigate exact sub-quadratic progressive formulations for the reverse-weighted trajectories, and evaluate performance on larger controlled benchmark platforms.

Acknowledgements

No additional human co-authors are declared for this technical report. The use of generative AI is reported separately in the dedicated declaration above and does not imply authorship.

A Appendix: Validation Output

The deterministic validation run used in preparation of this report produced:

```
PASS: 52,248 reference-vs-optimized comparisons
PASS: 4,848 progressive consistency comparisons
Maximum absolute discrepancy by function:
  RPDlep : 2.2204460492503131e-16
  RPDlea : 0
  RPDlepr : 0
  RPDgep : 1.1102230246251565e-15
  RPDgea : 3.8857805861880479e-16
  RPDgepr : 3.8857805861880479e-16
```

B Appendix: Reproducibility Checklist

- Historical software preserved and cited with a persistent identifier.
- Optimized software published in a separate repository.
- Optimized release archived with Zenodo DOI [10.5281/zenodo.21892420](https://doi.org/10.5281/zenodo.21892420).
- Reference snapshot retained for deterministic regression testing.
- Regression seed and numerical tolerance documented.
- Test counts and maximum discrepancies reported.
- Benchmark script and execution parameters documented.
- Short-input behavior and unsupported numerical domains reported.
- Code and documentation licensing stated explicitly.
- Generative-AI assistance disclosed explicitly.

References

- [1] M. Righi, *RPDIndex: Realistic Performance Determination indices for directional changes in normalized data*, version v1.0.2, Historical reference implementation; repository: <https://github.com/marcorighi/RPDIndex>, 2026. DOI: [10.5281/zenodo.20313771](https://doi.org/10.5281/zenodo.20313771). [Online]. Available: <https://doi.org/10.5281/zenodo.20313771>.
- [2] M. Righi, *RPDIndexOptimized: optimized Python implementation of the RPD indices*, version v1.0.0, Repository: <https://github.com/marcorighi/RPDIndexOptimized>, 2026. DOI: [10.5281/zenodo.21892420](https://doi.org/10.5281/zenodo.21892420). [Online]. Available: <https://doi.org/10.5281/zenodo.21892420>.
- [3] P. M. Fenwick, "A new data structure for cumulative frequency tables," *Software: Practice and Experience*, vol. 24, no. 3, pp. 327–336, 1994. DOI: [10.1002/spe.4380240306](https://doi.org/10.1002/spe.4380240306).