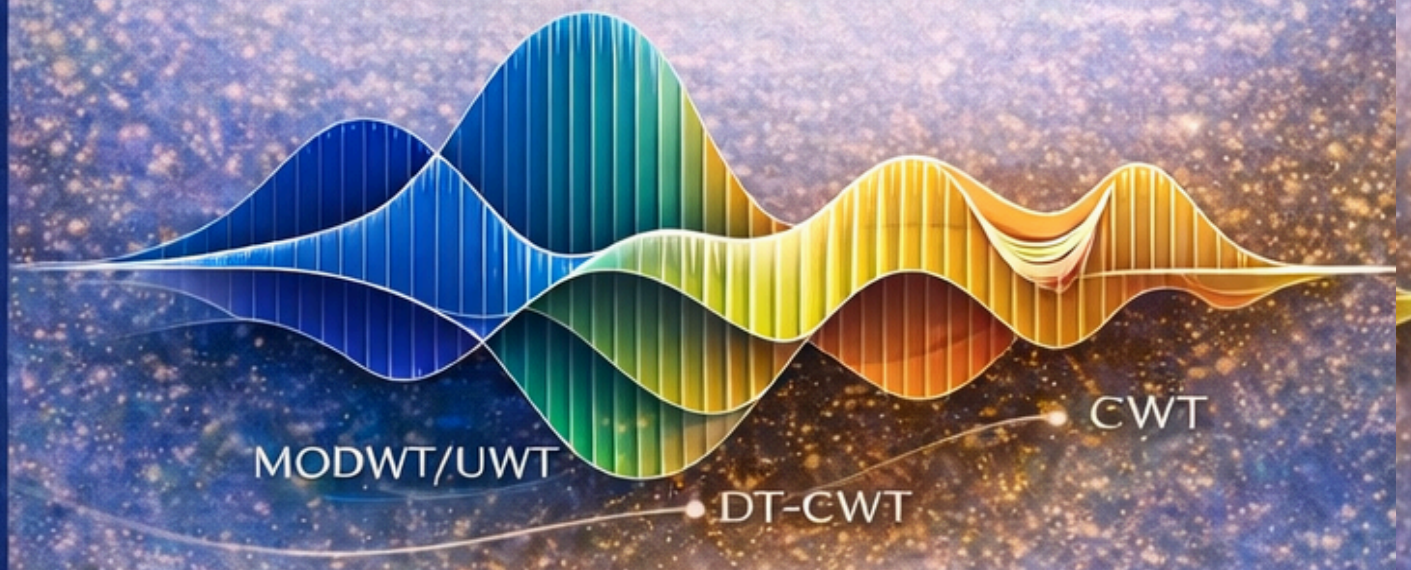
Wavelet Transform in Practice

From Theory to Production-Ready Python **Applications**

VOLUME II-B

Advanced Wavelet Methods

Multiresolution Analysis, Custom Wavelets, and Feature Engineering



Shouke Wei

Wavelet Transform in Practice

From Theory to Production-Ready Python Applications

Series

Wavelet Transform in Practice

From Theory to Production-Ready Python Applications

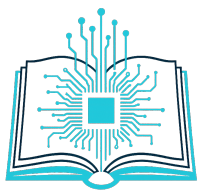
VOLUME II-B

Advanced Wavelet Methods

Multiresolution Analysis, Custom Wavelets, and Feature Engineering

---

Shouke Wei



DEEPSIM PRESS

First Edition, 2026

For permission requests, contact the publisher:
Email: `shouke.wei@deepsim.ca`

ISBN 978-1-0699284-5-0 (eBook)
DOI 10.5281/zenodo.18820664

Published by
Deepsim Press
An independent imprint of Deepsim Intelligence Technology Inc.
Abbotsford, British Columbia, Canada
https://press.deepsim.ca

This book is intended for educational and professional readers.

For resources, updates, and companion code, visit:
`https://press.deepsim.ca/wavelet-books`

About the Author

Shouke Wei, Ph.D., is a researcher, scientist, and entrepreneur specializing in intelligent IoT systems, robotics, big data analytics, modeling and forecasting, early-warning systems, and edge computing. With academic and industry experience across Europe, North America, and Asia, Dr. Wei is recognized for bridging advanced theory with real-world, production-ready systems.

Dr. Wei earned his Ph.D. in Environmental and Resource Management from the Department of Environmental Informatics at Brandenburg University of Technology Cottbus–Senftenberg (Germany). He conducted postdoctoral research at the Swiss Federal Institute of Aquatic Science and Technology (Eawag), where he also served as a doctoral supervisor, and held research positions at the University of British Columbia (Canada).

Recognized as a National High-End Talent (Class A) in China, Dr. Wei has held distinguished and adjunct professorships at multiple institutions, including Yantai University, Ludong University, and Jining University. He has served as a graduate supervisor and distinguished professor in computer science, control engineering, and applied mathematics.

Dr. Wei currently serves as CEO and Chief Scientist of Deepsim Intelligent Technology Inc. (Canada), Chief Scientist at Canadian Sincerity Enterprises Inc., and Chief Scientist of Shandong Deepsim Intelligent Technology Co., Ltd. He is also a Postdoctoral Co-Supervisor at the Shandong Postdoctoral Innovation Practice Base and currently serves as Director of Qilu Artificial Intelligence and Digital Manufacturing Innovation at Shandong Deepsim Intelligent Technology Co., Ltd., China.

Dr. Wei has led or contributed to 19 major international research projects and the development of 19 intelligent systems, including autonomous water-quality monitoring vessels, AI-based environmental early-warning platforms, medical image diagnosis models, precision agriculture robots, and autonomous service robots.

His scholarly contributions include 40+ peer-reviewed publications and 500+ tutorial online articles, 6 patents, 30 software copyrights, and 2 China national scientific and technological achievements. Dr. Wei's work focuses on making advanced computational methods—particularly wavelet-based signal processing—accessible, practical, and impactful for researchers and practitioners worldwide.

For more info, visit: `https://shouke.deepsim.ca`

Contents

Preface

This volume extends the applied wavelet workflows developed in *Volume II-A* into a set of **advanced transforms and design methodologies** that expand what wavelets can represent and how multiscale structure can be engineered for downstream tasks. While *Volume I* established the mathematical foundations and core discrete wavelet framework, and *Volume II-A* emphasized applied denoising and compression, this volume, *Advanced Wavelet Methods: Multiresolution Analysis, Custom Wavelets, and Feature Engineering* focuses on **shift-invariant analysis, complex and continuous transforms, custom wavelet construction, and feature engineering for machine learning**.

A recurring theme throughout these chapters is that practical performance often depends less on using "more wavelets" and more on choosing the **right representation** for a given signal class and objective. Undecimated methods such as the MODWT preserve temporal alignment and improve interpretability across scales. Complex wavelet constructions improve directionality and phase-aware representation. Continuous wavelet transforms provide flexible time–frequency analysis for nonstationary signals and generalize naturally to higher dimensions through translation, scaling, and rotation. Finally, custom wavelet design and wavelet-based feature extraction provide a direct path from multiscale analysis to task-driven representations in modern machine learning workflows.

The goal of this volume is not to present wavelets as isolated algorithms, but as **representation tools** that can be adapted, compared, and validated under realistic constraints. Implementation is emphasized throughout, with Python examples designed for reproducibility, experimentation, and extension.

Who This Book Is For

This volume is intended for:

- Practitioners who already use DWT-based methods and require shift-invariant, complex, or continuous representations
- Researchers working with nonstationary signals, directional textures, or multiscale phenomena that challenge standard DWT assumptions
- Data scientists and ML engineers building multiscale features for classification, regression, forecasting, or anomaly detection
- Graduate students seeking advanced wavelet methods with implementable Python workflows
- Software engineers integrating wavelet representations into analytical pipelines and production-oriented experimentation

Readers are expected to be comfortable with discrete wavelet concepts and Python workflows as introduced in *Volume I* and practiced in *Volume II-A*.

Organization of This Volume

This volume is organized into **three parts**, reflecting a progression from advanced discrete wavelet transforms to continuous formulations, and finally to representation design for data analysis and learning. Each part builds on the previous one, moving from signal-aligned decomposition toward more flexible and task-oriented multiscale representations.

Part I: Undecimated and Shift-Invariant Multiresolution Methods
This part introduces the Maximal Overlap Discrete Wavelet Transform (MODWT) and MODWT-based multiresolution analysis (MODWTMRA). Emphasis is placed on shift invariance, time alignment, and scale-wise interpretability, with detailed discussion of boundary handling, redundancy, and practical implementation considerations. Applications focus on analyzing

nonstationary signals where phase distortion and alignment artifacts of the standard DWT are problematic.

Part II: Complex and Continuous Wavelet Transforms
This part develops phase-aware and directional representations using complex wavelet methods and the Continuous Wavelet Transform (CWT). Beginning with one-dimensional continuous analysis, it extends to two- and higher-dimensional settings, addressing scale–frequency relationships, translation and rotation behavior, edge effects, and coefficient interpretation. The goal is to provide a unified view of time–frequency and time–scale representations for exploratory analysis and feature extraction.

Part III: Custom Wavelets and Feature Engineering
The final part focuses on the design of custom wavelets and the construction of task-driven multiscale features for machine learning and statistical modeling. It presents principles for wavelet selection and design, discusses evaluation criteria tied to downstream tasks, and demonstrates practical workflows for integrating wavelet-based features into classification, regression, forecasting, and anomaly detection pipelines.

Prerequisites and Computational Environment

This volume assumes:

- Proficiency in Python programming and scientific computing workflows
- Familiarity with NumPy, SciPy, Pandas, Matplotlib, scikit-learn, opencv-python,scikit-image, seaborn, and PyWavelets
- Prior understanding of DWT-based decomposition and reconstruction

Several chapters require additional libraries beyond the standard stack, including packages for MODWT-based analysis and complex wavelet transforms. A concise setup guide is provided in *Chapter 0* for this volume, and readers may also consult the setup chapters in *Volume I* and *Volume II-A* for broader environment guidance.

On Code, Evaluation, and Reproducibility

All code examples are designed to be reproducible and modular. Where possible, examples use public datasets; otherwise, synthetic signals are constructed to preserve relevant structure (e.g., chirps, transients, multiscale textures). Evaluation emphasizes both quantitative metrics and interpretability, since advanced representations often trade computational cost for improved stability or structure-aware features.

Shouke Wei, PhD
Deepsim Intelligent Technology Inc.
Deepsim Academy
Abbotsford, Canada
March 1, 2026

Acknowledgments

I am grateful to colleagues, collaborators, and practitioners whose discussions helped shape the advanced focus of this volume—particularly around shift-invariant multiresolution analysis, continuous time–frequency representations, and the challenges of interpreting multiscale structure in real data. Their perspectives reinforced that advanced transforms are most valuable when they are connected to clear objectives and validated under realistic constraints.

I thank my students and research collaborators for their careful reading, critical questions, and persistent interest in connecting wavelet theory to implementable workflows and learning-oriented representations. Their feedback continually pushed this volume toward clearer structure and stronger practical guidance.

I am deeply grateful to my family for their patience and encouragement throughout the writing of this trilogy.

This volume is built on decades of foundational work in wavelet theory and multiscale analysis by pioneers such as Ingrid Daubechies, Stéphane Mallat, and many others. I also acknowledge the open-source scientific computing community—particularly contributors to **PyWavelets**, **NumPy**, **SciPy**, **Matplotlib**, **scikit-learn**, and specialized wavelet libraries supporting MODWT and complex wavelet transforms. These tools make advanced experimentation, reproducible analysis, and rigorous comparison feasible in practice.

List of Symbols

The following symbols are used throughout this volume. (Core DWT symbols are introduced in earlier volumes; here we emphasize symbols specific to advanced transforms, custom design, and feature engineering.)

Symbol	Description
$x[n]$	Discrete-time signal
$W_x(s, \tau)$	Continuous wavelet transform of x at scale s and translation τ
s	Scale parameter (CWT)
τ	Translation parameter (CWT)
θ	Rotation/orientation parameter (multidimensional CWT)
$\psi(t)$	Mother wavelet (CWT)
$\widehat{\psi}(\omega)$	Fourier transform of the wavelet
C_ψ	Admissibility constant
ω	Angular frequency
Δt	Sampling interval
f	Frequency (Hz)
MODWT	Maximal Overlap Discrete Wavelet Transform (shift-invariant)
MODWTMRA	MODWT-based multiresolution analysis
$\tilde{W}_j[n]$	MODWT wavelet coefficients at level j (time-aligned)
$\tilde{V}_j[n]$	MODWT scaling coefficients at level j
$\tilde{D}_j[n]$	MODWT detail component at level j (MRA)

Symbol	Description
$\widetilde{S}_j[n]$	MODWT smooth component at level j (MRA)
DT-CWT	Dual-Tree Complex Wavelet Transform
$w_{j,k}^{\mathbb{C}}$	Complex wavelet coefficient at scale j, location k
$\mathfrak{R}(\cdot), \mathfrak{I}(\cdot)$	Real and imaginary parts
$\lvert w \rvert$	Magnitude of a complex coefficient
$\angle w$	Phase of a complex coefficient
h_0, h_1	Analysis lowpass/highpass filters
g_0, g_1	Dual-tree filter pair
ψ_{custom}	Custom-designed wavelet
ϕ_{custom}	Custom scaling function
$\mathbf{X}$	Feature matrix (samples $\times$ features)
$\mathbf{y}$	Target vector
$\hat{\mathbf{y}}$	Predicted outputs
μ, σ, σ^2	Mean, standard deviation, variance
MSE	Mean squared error
J	Number of decomposition levels
N	Number of samples (or pixels)

Applied Tools and Dependencies

This part of *Volume II* focuses on **applied wavelet methods, feature engineering, evaluation workflows, and real-world case studies**. Rather than repeating the general Python setup described in *Volume I* and *Volume II-A*, this section summarizes the **additional tools and libraries required to execute the applied examples in Volume II-B**.

Readers who require a complete introduction to Python environments, virtual environments, or basic wavelet usage should consult **Volume I, Chapter 0**.

Software Requirements

All executable examples in Volume II-B are implemented in **Python** and build upon the standard scientific computing stack.

In addition, several chapters introduce **specialized wavelet libraries**, **machine-learning tools**, and **data-access utilities** commonly used in applied workflows.

The following packages are used across one or more chapters in Volume II-B:

Core scientific stack

- **NumPy** — numerical computation and array operations
- **SciPy** — statistical analysis and signal-processing utilities
- **Matplotlib** — visualization and plotting
- **mpl_toolkits.mplot3d** — 3D visualization support

Wavelet and transform libraries

- **PyWavelets** — discrete wavelet transforms and multiresolution analysis
- **modwtpy** — MODWT, inverse MODWT, and MODWT-based multiresolution analysis
- **dtcwt** — Dual-Tree Complex Wavelet Transform

Machine learning and evaluation

- **scikit-learn** — feature preprocessing, model training, and evaluation metrics
  (e.g., precision, recall, F1 score, cross-validation, grid search)

Data handling and external data

- **pandas** — data manipulation and time-series handling
- **yfinance** — financial time-series data retrieval

Image and multidimensional data

- **opencv-python** — image input/output and preprocessing
- **scikit-image** — image and multidimensional signal processing
- **seaborn** — statistical visualization utilities

Installation

It is recommended to use a virtual environment to isolate dependencies:

```
python -m venv venv
source venv/bin/activate   # Linux/macOS
# or
venv\Scripts\activate    # Windows
```

All required third-party packages can then be installed using the provided `requirements.txt` file for Volume II-B:

```
pip install -r requirements.txt
```

This ensures consistent behavior across all applied examples and case studies.

Notes on Applied Transforms and Tooling

Several chapters in Volume II-B rely on wavelet transforms and analysis pipelines that extend beyond the standard discrete framework introduced earlier:

- **MODWT and MODWTMRA** examples are implemented using the `modwtpy` package
- **Dual-Tree Complex Wavelet Transform** examples are implemented using the `dtcwt` library

These libraries are not part of the core PyWavelets distribution and must be installed explicitly.

Referenced External Ecosystems

For comparison and context, Volume II-B also references wavelet tools from other ecosystems:

- **R** — `waveslim` package (full MODWT implementation)
- **MATLAB** — custom scripts and toolbox-based wavelet implementations

These tools are **not required** to execute the Python examples and are mentioned to highlight cross-platform equivalence and methodological differences.

Scope of This Setup

This setup chapter is intentionally concise. Its purpose is to:

- Identify **additional dependencies specific to applied chapters**
- Distinguish between **executable Python tools** and **referenced external ecosystems**
- Support reproducible execution of real-world examples and case studies

For foundational environment configuration and general tooling, readers are encouraged to refer back to **Volume I, Chapter 0**.

Part I

Undecimated and Shift-Invariant Multiresolution Methods

1 The Maximal Overlap Discrete Wavelet Transform: Theory and Implementation

The Maximal Overlap Discrete Wavelet Transform (MODWT) is a modified version of the standard Discrete Wavelet Transform (DWT) that offers several computational and analytical advantages. Originally developed by (Percival and Mofjeld 1997), the MODWT addresses key limitations of the traditional DWT by providing translation invariance and the ability to analyze time series of arbitrary length.

Unlike the traditional DWT, which requires signals of length 2^J and suffers from translation variance, the MODWT maintains all the essential properties of wavelet analysis while offering enhanced flexibility for practical applications (Percival and Walden 2000). The transform has found extensive applications in geophysical signal analysis, financial time series analysis, and biomedical signal processing (Paul S. Addison 2002).

The MODWT provides a redundant representation of the signal, meaning it preserves more information than the critically sampled DWT. This redundancy factor of $2^J - 1$ for J-level decomposition, while increasing computational requirements by a factor of $log_2(N)$, offers enhanced flexibility in analysis and reconstruction (Cornish, Bretherton, and Percival 2006). The translation-invariant property makes MODWT particularly suitable for feature detection, change-point analysis, and situations where the timing of signal characteristics is crucial(Anderson and Stephens 2000).

Recent developments have extended MODWT applications to multiscale variance analysis (Percival 2008), long-memory process modeling (Jensen 1999), and non-stationary time series decomposition (Nason 2008). The transform's ability to provide a multiresolution analysis (MRA) without the constraints of dyadic sampling has made it an essential tool in modern signal processing and statistical analysis.

1.1 Overview

In Chapter 7 of Volume I, the Stationary Wavelet Transform, a well-known undecimated wavelet transform, was presented. Building on that foundation, this chapter introduces another important undecimated wavelet transform: the **Maximal Overlap Discrete Wavelet Transform (MODWT)**. The chapter focuses on:

- illustration the differences between Decimated Wavelet Transform (DWT) and Undecimated Wavelet Transform (UWT),
- Stationary Wavelet Transform (SWT)
- Maximal Overlap Discrete Wavelet Transform (MODWT)

1.2 MODWT vs SWT

The **Stationary Wavelet Transform (SWT)** and the **Maximal Overlap Discrete Wavelet Transform (MODWT)**are among the most widely used forms of the **Discrete Wavelet Transform (DWT)**.

1.2.1 Maximal Overlap Discrete Wavelet Transform (MODWT)

The Maximal Overlap Discrete Wavelet Transform (MODWT) is another undecimated wavelet transform that builds on the idea of removing downsampling, like SWT. It is widely used in statistical signal processing, particularly for time series analysis (Percival and Walden 2000).

MODWT is also known in literature as:

- Undecimated DWT
- Stationary DWT
- Translation-Invariant DWT
- Redundant DWT

Unlike SWT, MODWT works with any signal length and adjusts filter coefficients through normalization (rather than simply upsampling). It retains n+1 sets of coefficients for an n-level decomposition (i.e., one approximation + n details).

Applications of MODWT:

- Multiscale decomposition of time series,
- Wavelet-based variance and correlation analysis,
- Trend and anomaly extraction in environmental and financial data.

> **Note**
>
> MODWT is not natively implemented in PyWavelets, but approximations are possible with norm=True and trim_approx=True in `pywt.swt()`. Full implementations are available in R (waveslim) and MATLAB.

1.2.2 Comparison: SWT vs. MODWT

Both the **Stationary Wavelet Transform (SWT)** and the **Maximal Overlap Discrete Wavelet Transform (MODWT)** are **undecimated**, **redundant**, and **shift-invariant** wavelet transforms designed to overcome the limitations of traditional DWT. While they share similar goals, they differ in implementation, signal length requirements, and common use cases.

1. Key Differences

A detailed comparison of the **Stationary Wavelet Transform (SWT)** and the **Maximal Overlap Discrete Wavelet Transform (MODWT)** is presented in Table 1.1, with their processing workflows illustrated in Figure 1.1.

Refer to the diagram below for a side-by-side view of how SWT and MODWT differ in structure and usage.

- **SWT** inserts zeros between filter coefficients (upsampling), preserving signal length and producing approximation and detail coefficients at every level.
- **MODWT** uses normalized filters for any-length signals, maintaining alignment and statistical properties while yielding one approximation and multiple detail levels.

Each method is optimized for different applications: SWT excels in engineering contexts like denoising and biomedical analysis, while MODWT is preferred in statistical time series and forecasting.

Table 1.1: A summary of differences between the Stationary Wavelet Transform (SWT) and the Maximal Overlap Discrete Wavelet Transform.

Aspect	SWT	MODWT
Filter handling	Upsamples filters	Normalizes filters
Signal length requirements	Typically power-of-two	Arbitrary length
Output structure	n levels: n detail + n approx	n levels: n detail + 1 approx
Energy preservation	Not exact	Yes (with correct normalization)
Statistical consistency	Moderate	Strong (used in time series stats)
PyWavelets support	`swt()`, `iswt()`	Not native (only approximations)
Common toolkits	PyWavelets, MATLAB	R (`waveslim`), MATLAB, custom Python
Typical domains	Engineering, denoising, image signals	Environmental, economic, biomedical

1.2.3 Brief Summary

- **SWT** is practical, intuitive, and widely available in tools like PyWavelets. It's well suited for applications requiring temporal alignment, such as

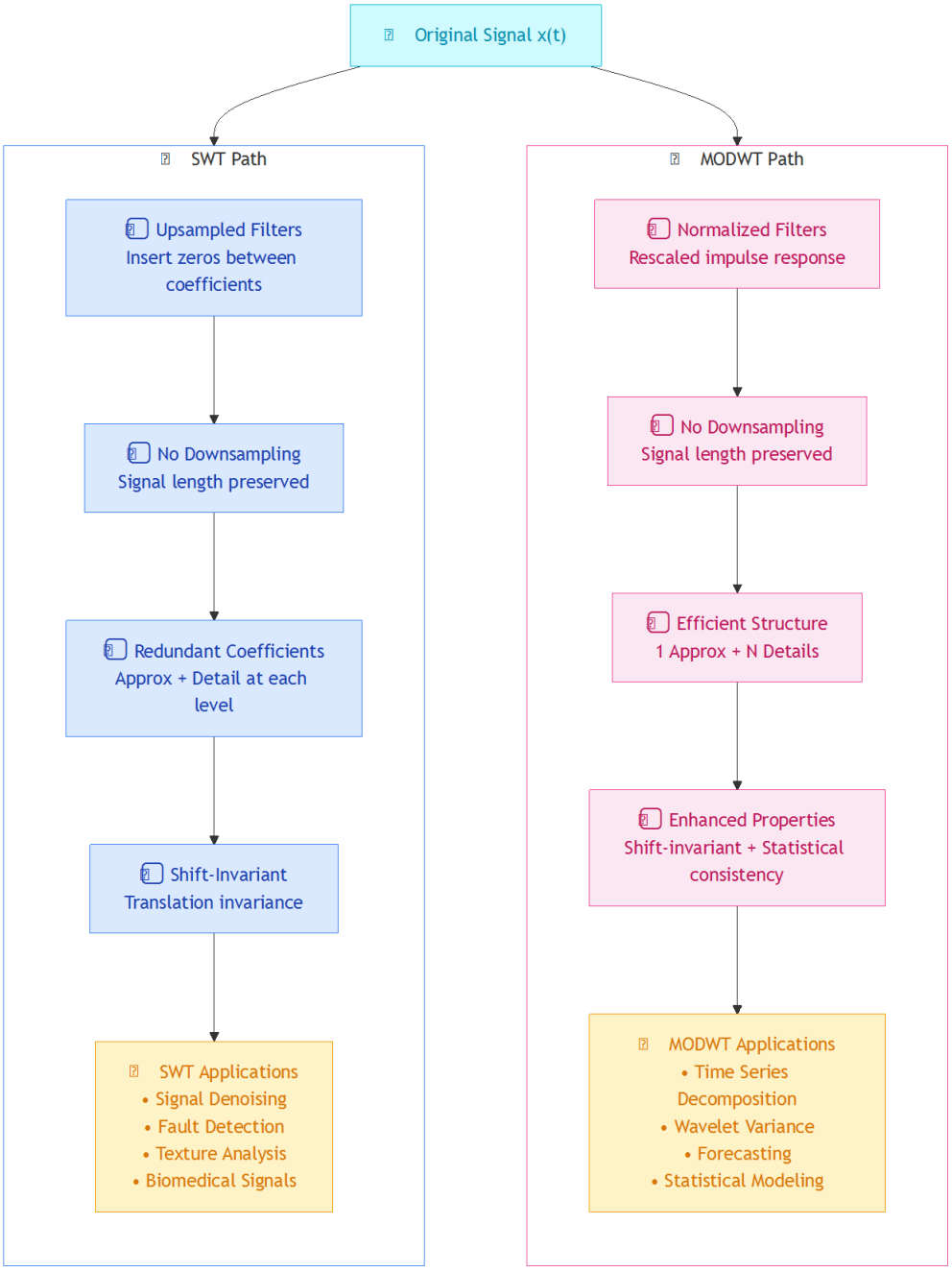


Figure 1.1: Process flow comparison of Stationary Wavelet Transform (SWT) and Maximal Overlap Discrete Wavelet Transform (MODWT).The figure illustrates how SWT and MODWT both start from the original signal and avoid downsampling, but differ in filter handling and output structure.

denoising, **fault detection**, and **texture analysis**.

- **MODWT** is rooted in statistical theory and is widely used for **time series decomposition**, **wavelet variance**, and **trend detection** in fields such as **climate science** and **finance**.

Though they differ technically, **both transforms provide stable, shift-invariant decompositions** that are especially useful in non-stationary signal analysis.

1.3 Mathematical Foundation

1.3.1 Wavelet Theory Basics

A wavelet is a mathematical function that can be used to decompose signals into different frequency components. The fundamental concept involves two functions:

- **Scaling function (father wavelet)** $\varphi(t)$: Captures the low-frequency approximation
- **Wavelet function (mother wavelet)** $\psi(t)$: Captures the high-frequency details

These functions satisfy the following properties:

Scaling equation:

$$\varphi(t) = \sqrt{2} \sum_{k} h_k \varphi(2t - k) \tag{1.1}$$

Wavelet equation:

$$\psi(t) = \sqrt{2} \sum_{k} g_k \varphi(2t - k) \tag{1.2}$$

where h_k and g_k are the scaling and wavelet filter coefficients, respectively.

1.3.2 MODWT Filter Coefficients

For the MODWT, the filter coefficients are rescaled versions of the DWT filters:

MODWT scaling filter:

$$\tilde{h}_{j,k} = \frac{h_k}{\sqrt{2^j}} \tag{1.3}$$

MODWT wavelet filter:

$$\tilde{g}_{j,k} = \frac{g_k}{\sqrt{2^j}} \tag{1.4}$$

where j represents the decomposition level.

1.3.3 MODWT Transform Equations

For a discrete signal:

$$\mathbf{x} = [x_0, x_1, \dots, x_{N-1}] \tag{1.5}$$

the MODWT coefficients at level j are computed as:

Detail coefficients (high-frequency components):

$$\tilde{W}_{j,t} = \sum_{l=0}^{L-1} \tilde{g}_l x_{t-l \bmod N} \tag{1.6}$$

Approximation coefficients (low-frequency components):

$$\tilde{V}_{j,t} = \sum_{l=0}^{L-1} \tilde{h}_l x_{t-l \bmod N} \tag{1.7}$$

where:

- L is the filter length
- N is the signal length

- $t = 0, 1, \dots, N-1$
- $\bmod N$ indicates circular boundary conditions

1.4 Key Properties of MODWT

1.4.1 Translation Invariance

Unlike the DWT, the MODWT is translation-invariant, meaning:

$$\text{MODWT}\{x(t-\tau)\} = \text{MODWT}\{x(t)\} \text{ shifted by } \tau \tag{1.8}$$

This property is crucial for applications where the timing of features matters.

1.4.2 Energy Preservation

The MODWT preserves the energy of the original signal:

$$\|\mathbf{x}\|^2 = \sum_{j=1}^{J} \|\tilde{\mathbf{W}}_j\|^2 + \|\tilde{\mathbf{V}}_J\|^2 \tag{1.9}$$

where J is the maximum decomposition level.

1.4.3 Perfect Reconstruction

The original signal can be perfectly reconstructed from its MODWT coefficients:

$$\mathbf{x} = \sum_{j=1}^{J} \tilde{\mathbf{D}}_j + \tilde{\mathbf{S}}_J \tag{1.10}$$

where:

- $\tilde{\mathbf{D}}_j$ is the reconstructed detail components
- $\tilde{\mathbf{S}}_J$ is the reconstructed smooth components.

1.5 Implementation and Practical Methods

1.5.1 Package Installation and Setup

To work with MODWT in Python, we utilize the `modwtpy` package (Pistonly 2025):

```
# Download or clone the package: https://github.com/pistonly/modwtpy
# Copy modwtpy folder to your working directory
```

1.5.2 MODWT Decomposition

The `modwt()` function performs the forward transform:

```
coeffs = modwt(data, wavelet, level)
```

Parameters: - `data`: Input signal (1D array) - `wavelet`: Wavelet type (e.g., `db1`, `db4`, `haar`) - `level`: Number of decomposition levels

Returns:

A list $[\mathbf{cD}_1, \mathbf{cD}_2, \ldots, \mathbf{cD}_n, \mathbf{cA}_n]$

where:

- $\mathbf{cD}_j$: Detail coefficients at level j.
- $\mathbf{cA}_n$: Approximation coefficients at the final level n.

1.5.3 Decomposition Example

In this section, we illustrate a practical example of performing a level-3 MODWT decomposition using the `db2` wavelet on a noisy sine wave signal. This example helps confirm the key property of MODWT: all coefficients retain the same length as the original signal.

1. Check the Coefficient Lengths

The following script performs a MODWT decomposition at level 3 using the 'db2' wavelet and verifies that all coefficient arrays (both detail and approximation) retain the same length as the original signal.

```
from modwtpy.modwt import modwt
import numpy as np

# Create a noisy sine wave signal
np.random.seed(0)
t = np.linspace(0, 1, 256)
signal = np.sin(2 * np.pi * 5 * t) + 0.5 * np.random.randn(256)

# Perform MODWT decomposition
coeffs = modwt(signal, 'db2', level=3)

# Extract coefficients
cD1, cD2, cD3, cA3 = coeffs

# Print lengths
print("Original signal length:", len(signal))
print("cD1 length:", len(cD1))
print("cD2 length:", len(cD2))
print("cD3 length:", len(cD3))
print("cA3 length:", len(cA3))
```

Output:

```
Original signal length: 256
cD1 length: 256
cD2 length: 256
cD3 length: 256
cA3 length: 256
```

The output results confirm that the approximation and detail coefficients obtained from MODWT have the same length as the original signal, highlighting the redundant and translation-invariant nature of the transform.

2. Visulize Decomisition Result

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 10))

plt.subplot(5, 1, 1)
plt.plot(t, signal, label="Noisy Signal")
plt.title("Original Noisy Signal")
plt.grid(True)

plt.subplot(5, 1, 2)
plt.plot(t, cD1, color='orange', label="Level 1 Detail (cD1)")
plt.title("Level 1 Detail Coefficients (cD1)")
plt.grid(True)

plt.subplot(5, 1, 3)
plt.plot(t, cD2, color='green', label="Level 2 Detail (cD2)")
plt.title("Level 2 Detail Coefficients (cD2)")
plt.grid(True)

plt.subplot(5, 1, 4)
plt.plot(t, cD3, color='blue', label="Level 3 Detail (cD3)")
plt.title("Level 3 Detail Coefficients (cD3)")
plt.grid(True)

plt.subplot(5, 1, 5)
plt.plot(t, cA3, color='red', label="Level 3 Approximation (cA3)")
plt.title("Level 3 Approximation Coefficients (cA3)")
plt.grid(True)

plt.tight_layout()
plt.savefig("./output/modwt_example.png")
plt.show()
```

Explanation:

- Signal: A 5 Hz sine wave sampled at 256 points with Gaussian noise.
- MODWT: 3-level decomposition using 'db2' wavelet.
- Redundancy: All coefficients (details and approximation) are length 256 (same as input).

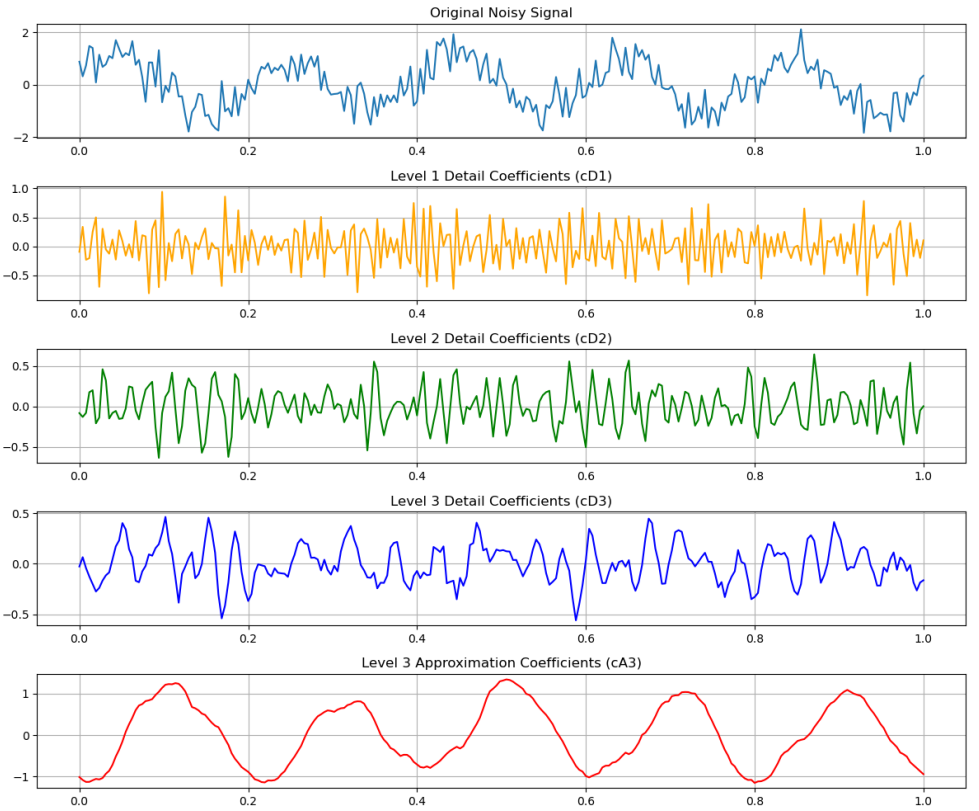


Figure 1.2: Multilevel MODWT decomposition of a noisy sine wave using the 'db2' wavelet up to level 3. The plot shows the original noisy signal, detail coefficients at levels 1–3 (cD1, cD2, cD3), and the level-3 approximation (cA3). All coefficient arrays retain the same length as the original signal, preserving alignment and enabling translation-invariant analysis.

3. Energy Distributions

```
import pandas as pd

# Compute energy = sum of squared coefficients
energies = {
    "D1": np.sum(cD1**2),
    "D2": np.sum(cD2**2),
    "D3": np.sum(cD3**2),
    "A3": np.sum(cA3**2),
}
total_energy = np.sum(signal**2)

# Build DataFrame for table
df = pd.DataFrame({
    "Energy": energies.values(),
    "Percentage (%)": [100 * e / total_energy for e in
     ↪  energies.values()]
}, index=energies.keys())

# Add total row
df.loc["Total"] = [total_energy, 100.0]

print("Energy Distribution Table:")
table_str = df.round(4).to_string()
print("-" * len(table_str.splitlines()[0]))
print(table_str)
```

Output:

```
Energy Distribution Table:
------------------------------
         Energy  Percentage (%)
D1      30.5164         15.5022
D2      14.4242          7.3274
D3       9.0630          4.6040
A3     142.8485         72.5664
Total  196.8521        100.0000
```

The energy distribution shows that the approximation coefficients at level 3 (A3) retain the majority of the signal's energy ($\approx 72.6\%$), indicating that the main signal structure is concentrated in the low-frequency components. The detail levels D1–D3 together capture only about 27.4% of the energy, mostly corresponding to high-frequency noise and finer fluctuations. This confirms that most useful information lies in the approximation, while details mainly represent noise.

1.5.4 MODWT Inverse Transform

1. Reconstruction Method

The `imodwt()` function reconstructs the original signal:

```
reconstructed_signal = imodwt(coeffs, wavelet)
```

Parameters: - `coeffs`: List of coefficient arrays from MODWT - `wavelet`: Same wavelet used in decomposition

2. Reconstruction Example

We continue from the previous decomposition example to reconstruct the original signal using the `imodwt()` function. The reconstruction is performed using the approximation and detail coefficients ($cA3, cD3, cD2, cD1$) obtained from the level-3 MODWT. This step verifies that the inverse MODWT accurately reconstructs the original signal from its multilevel components.

```
from modwtpy.modwt import imodwt
import matplotlib.pyplot as plt

# Arrange coefficients in the correct order: [cD1, cD2, cD3, cA3]
coeffs = [cD1, cD2, cD3, cA3]

# Perform inverse MODWT to reconstruct the original signal
reconstructed = imodwt(coeffs, 'db2')

# Compare original and reconstructed signals
MAE = np.mean(np.abs(signal - reconstructed))
```

```
print(f"Mean Absolute Error between original and reconstructed signal:
 ↪  {MAE}")

# Plot original and reconstructed signals
plt.figure(figsize=(10, 4))
plt.plot(t, signal, label='Original Signal', linestyle='--')
plt.plot(t, reconstructed, label='Reconstructed Signal', alpha=0.7)
plt.title("MODWT Signal Reconstruction from Level-3 Decomposition")
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig("./output/modwt_reconstruction.png")
plt.show()
```

Output:

```
Mean Absolute Error between original and reconstructed signal:
3.716373996737188e-16
```

The extremely small Mean Absolute Error (MAE) ($\sim 3.72 \times 10^{-16}$) indicates that the reconstructed signal is virtually identical to the original, confirming perfect reconstruction with MODWT/IMODWT.

The **Mean Absolute Error (MAE)** of 3.72×10^{-16} is virtually zero, confirming the accuracy of the reconstruction up to numerical precision.

1.6 Advanced Applications

1.6.1 Partial Reconstruction

1. Reconstruct A , D , D , D Separately from MODWT Coefficients

One of the powerful features of the **MODWT** is its ability to reconstruct individual frequency components of a signal. In this section, we continue the previous example to **separately reconstruct** the level-3 approximation (A_3)

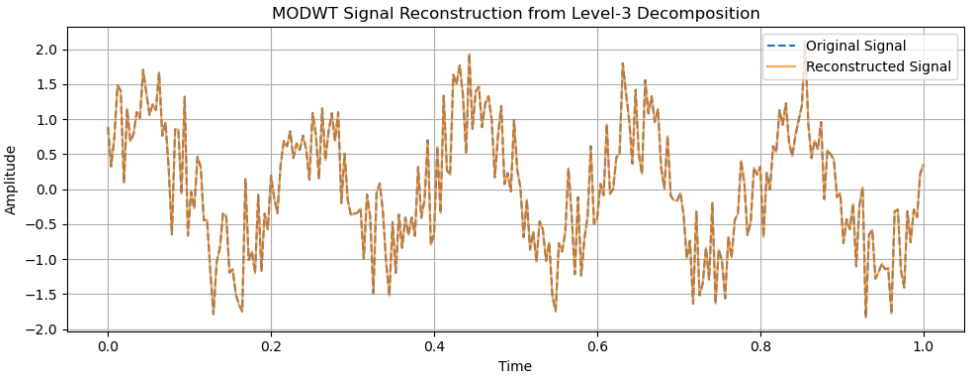


Figure 1.3: Signal reconstruction using inverse MODWT (`imodwt`) after a level-3 decomposition with the `db2` wavelet. The reconstructed signal (solid line) closely matches the original noisy signal (dashed line), confirming the perfect reconstruction property of MODWT.

and detail components at levels 3, 2, and 1 (D_3, D_2, D_1) from the MODWT coefficients. By summing these reconstructed components, we can fully recover the original signal, demonstrating MODWT's perfect reconstruction property.

```
# Perform MODWT decomposition
cD1, cD2, cD3, cA3 = modwt(signal, 'db2', level=3)

# Create zero arrays of same length
zero = np.zeros_like(cD1)

# Reconstruct components
A3 = imodwt([zero, zero, zero, cA3], 'db2')  # Approximation at level
↪  3
D3 = imodwt([zero, zero, cD3, zero], 'db2')  # Detail at level 3
D2 = imodwt([zero, cD2, zero, zero], 'db2')  # Detail at level 2
D1 = imodwt([cD1, zero, zero, zero], 'db2')  # Detail at level 1

# Optional: verify reconstruction
reconstructed = A3 + D3 + D2 + D1
mae = np.mean(np.abs(signal - reconstructed))
print("MAE:", mae)
```

Output:

```
MAE: 3.828317871046316e-16
```

The reconstruction is accurate, with a Mean Absolute Error (MAE) of 3.83×10^{-16}, confirming that the original signal is perfectly recovered (within numerical precision) by summing the partial components: $A_3 + D_3 + D_2 + D_1 \approx$ Original Signal

2. Plot Each Component

```
plt.figure(figsize=(12, 10))

plt.subplot(5, 1, 1)
plt.plot(t, signal, label="Original Signal")
plt.title("Original Signal")
plt.grid(True)

plt.subplot(5, 1, 2)
plt.plot(t, A3, label="Approximation A3", color="blue")
plt.title("Level 3 Approximation (A3)")
plt.grid(True)

plt.subplot(5, 1, 3)
plt.plot(t, D3, label="Detail D3", color="orange")
plt.title("Level 3 Detail (D3)")
plt.grid(True)

plt.subplot(5, 1, 4)
plt.plot(t, D2, label="Detail D2", color="green")
plt.title("Level 2 Detail (D2)")
plt.grid(True)

plt.subplot(5, 1, 5)
plt.plot(t, D1, label="Detail D1", color="red")
plt.title("Level 1 Detail (D1)")
plt.grid(True)

plt.tight_layout()
plt.savefig("./output/modwt_partial_reconstruction.png")
plt.show()
```

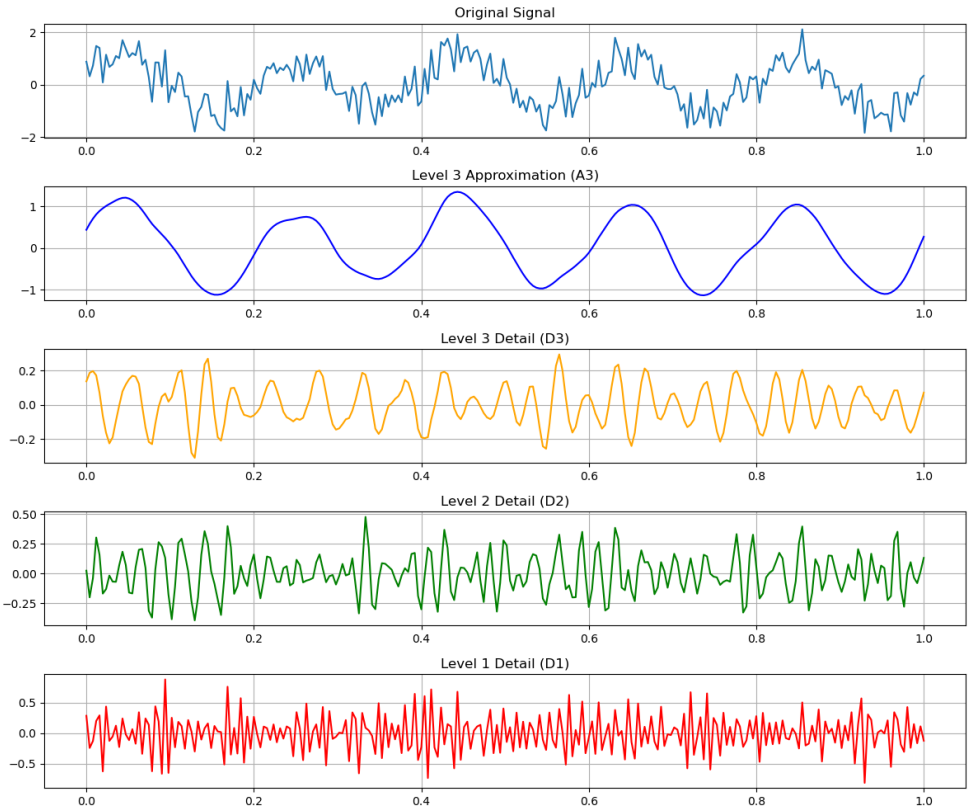


Figure 1.4: Multilevel partial reconstruction of the signal using MODWT with `db2` wavelet at level 3. The plots show the approximation component (A_3) and detail components (D_3, D_2, D_1) individually reconstructed using `imodwt()`. Summing these components yields a perfect reconstruction of the original signal with a **Mean Absolute Error (MAE)** of 3.83×10^{-16}, indicating negligible numerical error.

Note

- This method helps you analyze the contribution of each scale to the signal.
- You can recombine any subset like $A_3 + D_2 + D_1$ to filter out specific frequency bands.
- The imodwt() function supports perfect reconstruction when all components are summed.

1.6.2 Multiresolution Analysis

The MODWT enables multiresolution analysis by decomposing signals into different frequency bands:

- D_1: Captures the **highest frequency details**, representing the most rapid oscillations in the signal.
- D_2: Represents **mid-frequency variations**, corresponding to medium-scale features.
- D_3: Captures **lower-frequency details**, highlighting broader, coarser fluctuations.
- A_3: Contains the **overall trend** or **low-frequency behavior** of the signal — the smooth, large-scale structure.

1.6.3 Signal Denoising

MODWT can be used for signal denoising by:

1. Decomposing the noisy signal
2. Thresholding or filtering specific frequency components
3. Reconstructing the cleaned signal

```
def denoise_signal(signal, wavelet='db4', levels=3, threshold=0.1):
    """
    Denoise a signal using MODWT with soft thresholding on detail
↪   coefficients.
    Also computes evaluation metrics and energy distribution.
```

```
    Returns:
        denoised (array): Denoised signal
        metrics (dict): Error and quality metrics
        energy (list): Energy distribution per level
    """
    import numpy as np
    from modwtpy.modwt import modwt, imodwt

    # Perform MODWT decomposition
    coeffs = modwt(signal, wavelet, levels)

    # Compute energy before thresholding
    energy = [np.sum(c**2) for c in coeffs]

    # Apply soft thresholding to detail coefficients only
    for i in range(levels):
        coeffs[i] = np.sign(coeffs[i]) * np.maximum(np.abs(coeffs[i])
↪   - threshold, 0)

    # Reconstruct signal
    denoised = imodwt(coeffs, wavelet)

    # Compute error metrics
    mse = np.mean((signal - denoised) ** 2)
    rmse = np.sqrt(mse)
    mae = np.mean(np.abs(signal - denoised))
    snr = 10 * np.log10(np.sum(denoised**2) / np.sum((signal -
↪   denoised)**2))

    metrics = {
        "MSE": mse,
        "RMSE": rmse,
        "MAE": mae,
        "SNR (dB)": snr
    }

    return denoised, metrics, energy
```

> **Note**
>
> This function assumes that the last coefficient is the approximation (cA_n) and is left unchanged, which is typically the correct approach in wavelet denoising.
> You could also dynamically compute the threshold based on noise level or via universal thresholding (e.g., Donoho–Johnstone).

To visulize the denoised result, simply run:

```
denoised, metrics, energy = denoise_signal(signal, wavelet='db4',
        levels=3, threshold=0.2)

# Plot
fig, axs = plt.subplots(3, 1, figsize=(10, 8))

# 1. Original vs Denoised
axs[0].plot(signal, label="Noisy", alpha=0.7)
axs[0].plot(denoised, label="Denoised", linewidth=2)
axs[0].legend()
axs[0].set_title("Noisy vs Denoised Signal")

# 2. Residual
axs[1].plot(signal - denoised, color="red")
axs[1].set_title("Residual (Noisy - Denoised)")

# 3. Energy distribution
axs[2].bar(range(1, len(energy) + 1), energy, color="purple")
axs[2].set_xticks(range(1, len(energy) + 1))
axs[2].set_title("Wavelet Coefficient Energy per Level")
axs[2].set_xlabel("Level")
axs[2].set_ylabel("Energy")

plt.tight_layout()
plt.savefig("./output/modwt_denoising_eval.png")
plt.show()

# Print metrics
print("Evaluation Metrics:")
for k, v in metrics.items():
```

```
    print(f"{k}: {v:.4f}")
```

Output:

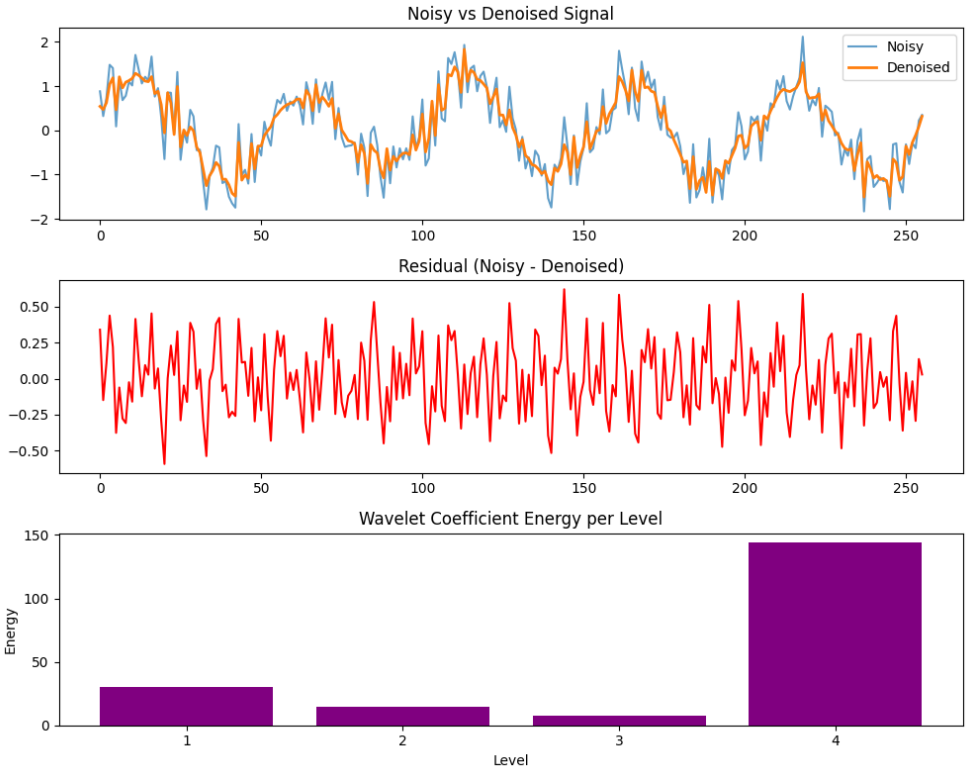


Figure 1.5: MODWT-based signal denoising evaluation. The top panel compares the noisy signal with its denoised version using MODWT and soft thresholding. The middle panel shows the residuals (difference between noisy and denoised signals), highlighting the removed noise components. The bottom panel presents the wavelet coefficient energy distribution across decomposition levels, indicating how signal energy is concentrated and how noise reduction affects different scales.

```
Evaluation Metrics:
MSE: 0.0641
RMSE: 0.2533
MAE: 0.2095
```

```
SNR (dB): 9.8049
```

The denoising process achieved a relatively low error, with an MSE of **0.0641**, RMSE of **0.2533**, and MAE of **0.2095**, indicating that the denoised signal closely approximates the original noisy input. The resulting **SNR of 9.80 dB** suggests a clear improvement in signal quality, as much of the noise has been effectively suppressed while preserving the main structure of the signal. Overall, the results demonstrate that MODWT with soft thresholding provides an efficient balance between noise reduction and signal fidelity.

1.7 Applications in Time Series Analysis

1.7.1 Trend Extraction

1. Trend Extraction Function

The approximation coefficients at the highest level provide smooth trend information:

```
def extract_trend(ts, trend_true=None, wavelet='db4', levels=4):
    """
    Extract the low-frequency trend from a time series using MODWT and
 ↪
    optionally evaluate against the true trend.

    Parameters:
        ts (array-like): Input time series.
        trend_true (array-like or None): True trend (if known, for
 ↪  evaluation).
        wavelet (str): Wavelet type (default: 'db4').
        levels (int): Number of decomposition levels.

    Returns:
        trend (array): Extracted trend component (approximation at
 ↪  final level).
        metrics (dict): Evaluation metrics (MSE, RMSE, MAE, R²) if
 ↪  trend_true given,
```

```
        else empty.
    """
    from modwtpy.modwt import modwt, imodwt
    import numpy as np

    # MODWT decomposition
    coeffs = modwt(ts, wavelet, levels)

    # Zero out all detail coefficients
    null_details = [np.zeros(len(ts)) for _ in range(levels)]

    # Keep only the approximation at the last level
    trend = imodwt(null_details + [coeffs[-1]], wavelet)

    # Compute evaluation metrics if ground truth is provided
    metrics = {}
    if trend_true is not None:
        mse = np.mean((trend_true - trend) ** 2)
        rmse = np.sqrt(mse)
        mae = np.mean(np.abs(trend_true - trend))
        r2 = (
            1 - np.sum((trend_true - trend) ** 2)
            / np.sum((trend_true - np.mean(trend_true)) ** 2)
        )

        metrics = {
            "MSE": mse,
            "RMSE": rmse,
            "MAE": mae,
            "R²": r2
        }

    return trend, metrics
```

This method:

- Keeps only the **final-level approximation** (A_n) from the MODWT decomposition.
- Zeroes out all **detail coefficients** (D_1 to D_n).

- Uses `imodwt()` to reconstruct just the trend (low-frequency content).

2. Visualization of Trending Result

To visualize the trend of a signal:

```
# ==== Example with synthetic signal ====
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
n = 256
t = np.linspace(0, 4 * np.pi, n)
trend_true = 0.5 * t  # Linear trend
noise = 0.5 * np.random.randn(n)
signal = trend_true + np.sin(t) + noise  # Trend + oscillation + noise

# Extract trend + metrics
trend_est, metrics = extract_trend(signal, trend_true=trend_true,
 ↪  wavelet='db4', levels=4)

# Plot
fig, axs = plt.subplots(2, 1, figsize=(10, 8))
axs[0].plot(signal, label="Original Signal", alpha=0.6)
axs[0].plot(trend_true, label="True Trend", linestyle="--",
 ↪  color="black")
axs[0].plot(trend_est, label="Extracted Trend (A )", linewidth=2)
axs[0].set_title("MODWT Trend Extraction (db4, 4 Levels)")
axs[0].legend()
axs[0].grid(True)

axs[1].plot(signal - trend_est, color="red")
axs[1].set_title("Residual after Removing Extracted Trend")
axs[1].grid(True)

plt.tight_layout()
plt.savefig("./output/modwt_trend_extraction_eval.png")
plt.show()

# Print metrics
print("Trend Extraction Metrics:")
```

```
for k, v in metrics.items():
    print(f"{k}: {v:.4f}")
```

Output:

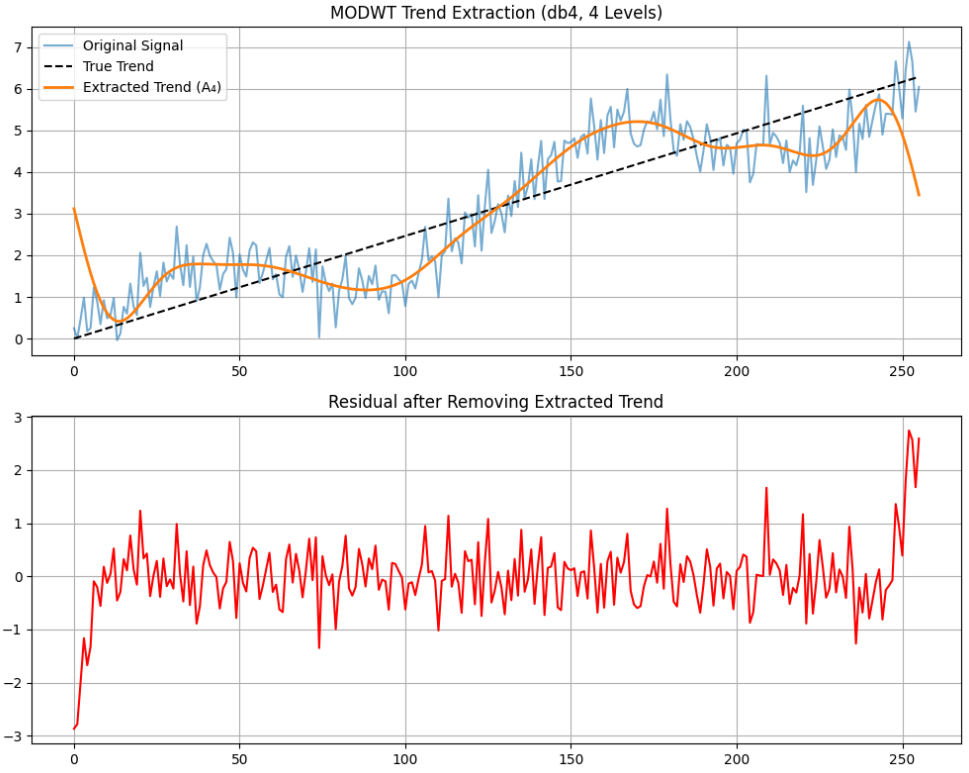


Figure 1.6: MODWT-based trend extraction of a synthetic time series. The top panel shows the original signal (trend + oscillation + noise), the true underlying linear trend, and the extracted trend using the level-4 approximation (A_4) from the MODWT. The bottom panel displays the residuals after removing the estimated trend, highlighting the remaining oscillatory and noise components.

```
Trend Extraction Metrics:
MSE: 0.7020
RMSE: 0.8379
MAE: 0.6887
R²: 0.7883
```

The extracted trend closely follows the underlying linear trend, but some of the high-frequency oscillations and noise remain in the residuals, as reflected by an MSE of **0.7020**, RMSE of **0.8379**, and MAE of **0.6887**. The $\mathbf{R^2}$ value of **0.7883** indicates that approximately **79%** of the variance in the true trend is captured by the MODWT approximation. Overall, the method effectively identifies the main low-frequency trend while leaving some finer-scale variations unremoved.

1.7.2 Anomaly Detection

Sudden changes often appear in high-frequency detail coefficients:

```
import numpy as np
import matplotlib.pyplot as plt
from modwtpy.modwt import modwt
from sklearn.metrics import precision_score, recall_score, f1_score

def detect_anomalies(ts, threshold_factor=3, wavelet='db1', level=1,
 ↪  true_anomalies=None):
    """
    Detect anomalies in a time series using MODWT detail coefficients.

    Parameters:
        ts (array-like): Input time series.
        threshold_factor (float): Multiplier for standard deviation
 ↪  threshold.
        wavelet (str): Wavelet type to use (default: 'db1').
        level (int): Decomposition level (default: 1, highest
 ↪  frequency).
        true_anomalies (array-like): Indices of true anomalies
 ↪  (optional).

    Returns:
        anomalies (np.ndarray): Indices of detected anomalies.
        details (np.ndarray): Detail coefficients used for detection.
        threshold (float): Computed threshold value.
        metrics (dict): Precision, Recall, F1 if true_anomalies
 ↪  provided, else empty.
```

```
    """
    # MODWT decomposition
    coeffs = modwt(ts, wavelet, level)
    details = coeffs[0]

    # Threshold
    threshold = threshold_factor * np.std(details)

    # Detected anomalies
    anomalies = np.where(np.abs(details) > threshold)[0]

    # Compute evaluation metrics if ground truth is provided
    metrics = {}
    if true_anomalies is not None:
        y_true = np.zeros(len(ts))
        y_true[true_anomalies] = 1
        y_pred = np.zeros(len(ts))
        y_pred[anomalies] = 1
        metrics = {
            "Precision": precision_score(y_true, y_pred),
            "Recall": recall_score(y_true, y_pred),
            "F1-score": f1_score(y_true, y_pred)
        }

    return anomalies, details, threshold, metrics
```

How It Works:

- **MODWT detail coefficients** capture **abrupt changes** in the signal (edges, spikes, outliers).
- A high magnitude in these coefficients indicates **non-typical behavior** at a particular time point.
- This method is **translation-invariant**, so anomalies aren't missed due to shift sensitivity.

An example usage:

```
# ==== Example ====
np.random.seed(42)
ts = np.sin(np.linspace(0, 10*np.pi, 500)) + 0.1*np.random.randn(500)
true_anomalies = [100, 300]
ts[true_anomalies[0]] += 3
ts[true_anomalies[1]] -= 4

# Detect anomalies + metrics
anomalies, details, threshold, metrics = detect_anomalies(
    ts, threshold_factor=3,
    true_anomalies=true_anomalies
)

# Plot
fig, axs = plt.subplots(2, 1, figsize=(10, 8))

# Signal with detected anomalies
axs[0].plot(ts, label="Signal")
axs[0].plot(anomalies, ts[anomalies], 'ro', label="Detected
 ↪  Anomalies")
axs[0].set_title("Signal with Detected Anomalies")
axs[0].legend()
axs[0].grid(True)

# Detail coefficients with threshold
axs[1].plot(details, label="Detail Coefficients (Level 1)")
axs[1].hlines([threshold, -threshold], 0, len(ts)-1, colors='r',
 ↪  linestyles='dashed', label='Threshold')
axs[1].set_title("MODWT Detail Coefficients and Detection Threshold")
axs[1].legend()
axs[1].grid(True)

plt.tight_layout()
plt.savefig("./output/modwt_anomaly_detection_eval.png")
plt.show()

# Print metrics
print("Anomaly Detection Metrics:")
for k, v in metrics.items():
    print(f"{k}: {v:.4f}")
```

Output:

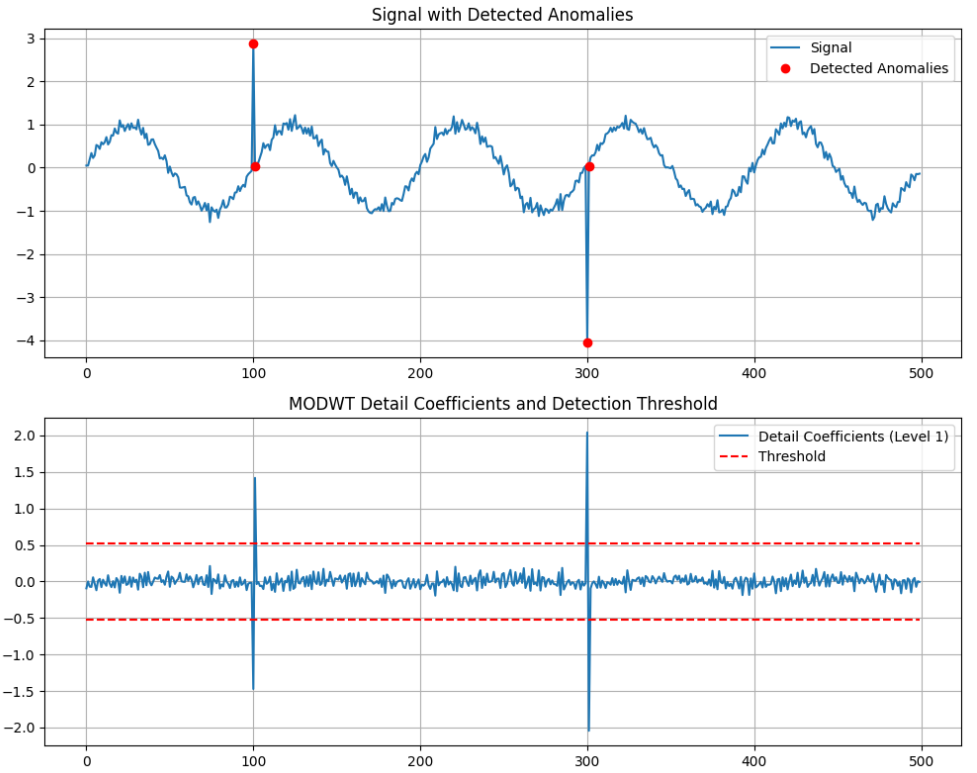


Figure 1.7: Anomaly detection in a time series using MODWT detail coefficients. The top panel shows the original signal with detected anomalies marked in red. The bottom panel displays the level-1 detail coefficients used for detection, along with dashed lines indicating the detection threshold. This visualization highlights how large deviations in high-frequency components correspond to anomalies in the signal.

```
Anomaly Detection Metrics:
Precision: 0.5000
Recall: 1.0000
F1-score: 0.6667
```

The anomaly detection successfully identified all true anomalies, as reflected by a **recall of 1.0**, indicating no missed events. However, the **precision of 0.5** shows that one false positive was also detected, resulting in an **F1-score**

of 0.6667, which balances precision and recall. Overall, the method is highly sensitive but may generate occasional false alarms.

1.8 Best Practices and Considerations

1.8.1 Wavelet Selection

- **Haar (db1)**: Simple, good for step functions and discontinuities
- **Daubechies (db4, db8)**: Balanced smoothness and localization
- **Biorthogonal**: Symmetric wavelets for image processing
- **Coiflets**: Good for numerical analysis applications

1.8.2 Level Selection

The maximum useful decomposition level is limited by:

$$J_{\max} = \lfloor \log_2(N) \rfloor \tag{1.11}$$

However, practical considerations often suggest fewer levels to avoid over-decomposition.

1.8.3 Boundary Effects

MODWT uses circular boundary conditions, which can introduce artifacts at signal boundaries. Consider:

- Zero-padding the signal
- Using symmetric extension
- Accounting for boundary coefficients in analysis

1.9 Computational Considerations

1.9.1 Memory Requirements

MODWT requires more memory than DWT due to redundancy:

- DWT: $O(N)$ memory
- MODWT: $O(JN)$ memory where J is the number of levels

1.9.2 18.9.2 Computational Efficiency

For large signals, consider:

- Limiting decomposition levels
- Using efficient implementations
- Parallel processing for multiple signals

1.10 Conclusion

The Maximal Overlap Discrete Wavelet Transform provides a powerful framework for signal analysis with several advantages over the standard DWT. Its translation invariance, ability to handle arbitrary signal lengths, and perfect reconstruction properties make it particularly valuable for:

- Time series analysis and forecasting
- Signal denoising and feature extraction
- Multiresolution analysis
- Anomaly detection
- Financial data analysis

The redundant representation, while computationally more expensive, offers enhanced flexibility and robustness in practical applications. Understanding both the theoretical foundations and practical implementation details enables effective use of MODWT in diverse signal processing tasks.

1. **MODWT Decomposition Practice**
   Generate a synthetic signal composed of multiple sinusoidal waves (e.g., 5 Hz + 20 Hz) and apply MODWT decomposition. Visualize the detail and approximation coefficients at each level.
2. **Wavelet Comparison**
   Decompose the same signal using two different wavelet families (e.g., `'db4'` vs `'sym4'`) and compare the reconstruction error and interpretability of the coefficients.
3. **Denoising via Thresholding**
   Design and implement a denoising pipeline using MODWT. Apply soft or hard thresholding to the detail coefficients and reconstruct the signal. Evaluate the effectiveness using MAE or SNR.
4. **Efficiency Analysis**
   Test the computational time and memory usage of MODWT decomposition at increasing levels. Discuss how the number of decomposition levels affects both performance and the interpretability of the results.

1.11 Exercises and Quizzes

1.11.1 Exercises

1. **MODWT Decomposition**: Generate a synthetic signal (e.g., 5 Hz + 20 Hz sinusoids) and apply `analyze_modwt` with 3 levels. Visualize coefficients and energy distribution.
2. **Wavelet Comparison**: Compare `db4` and `sym4` wavelets on the same signal, analyzing reconstruction error and coefficient interpretability.
3. **Denoising**: Implement a denoising pipeline using `analyze_modwt` with soft thresholding on detail coefficients. Evaluate using MAE or SNR.
4. **Efficiency Analysis**: Measure computational time and memory usage for increasing decomposition levels on a large signal. Discuss trade-offs.

1.11.2 Quick Quizzes

1. **True or False**: MODWT is translation-invariant while DWT is not.

2. Which of the following properties apply to MODWT?
   A. Orthogonality
   B. Redundancy
   C. Critical Sampling
   D. Periodicity

3. MODWT produces detail and approximation coefficients that are:
   A. Half the length of the original signal
   B. Double the length
   C. Same length
   D. Variable length depending on wavelet

4. What is the primary benefit of using MODWT for time series anomaly detection over DWT?

5. Briefly explain the difference in coefficient structure between MODWT and SWT.

1.11.3 Answers:

1. True
2. B
3. C
4. MODWT maintains translation invariance and preserves the temporal alignment of anomalies, making detection more consistent across time.
5. Both MODWT and SWT provide redundant representations, but:

- SWT upsamples the filters at each level, leading to aligned coefficients but requiring signal length to be divisible by 2^n

- MODWT uses circular convolution with rescaled filters (no downsampling), allowing arbitrary signal lengths and producing coefficients of the same length as the input signal.

References

Abhishek, S., and S. Veni. 2022. "Sparsity Enhancing Wavelets Design for ECG and Fetal ECG Compression." *Biomedical Signal Processing and Control* 71 (January): 103082. https://doi.org/10.1016/J.BSPC.2021.103082.

Addison, Paul S. 2002. *The Illustrated Wavelet Transform Handbook*. IOP Publishing Ltd. https://doi.org/10.1887/0750306920.

Addison, Paul S. 2017. "The Illustrated Wavelet Transform Handbook: Introductory Theory and Applications in Science, Engineering, Medicine and Finance, SECOND EDITION." *The Illustrated Wavelet Transform Handbook: Introductory Theory and Applications in Science, Engineering, Medicine and Finance, SECOND EDITION*, January, 1–446. https://doi.org/10.1201/9781315372556/ILLUSTRATED-WAVELET-TRANSFORM-HANDBOOK-PAUL-ADDISON/RIGHTS-AND-PERMISSIONS.

Amir, A., T. S. Bindiya, and Elizabeth Elias. 2018. "Design and Implementation of Reconfigurable Filter Bank Structure for Low Complexity Hearing Aids Using 2-Level Sound Wave Decomposition." *Biomedical Signal Processing and Control* 43 (May): 96–109. https://doi.org/10.1016/J.BSPC.2018.02.020.

Anderson, T. W., and M. A. Stephens. 2000. "Sign Invariance in Goodness-of-Fit Tests for Time Series." *Journal of Time Series Analysis* 21 (September): 489–96. https://doi.org/10.1111/1467-9892.00194.

Argyropoulos, Paraskevas E., and Hanoch Lev-Ari. 2015. "Wavelet Customization for Improved Fault-Location Quality in Power Networks." *IEEE Transactions on Power Delivery* 30 (October): 2215–23. https://doi.org/10.1109/TPWRD.2015.2429590.

Aslam, Aasma, Khizar Hayat, Arif Iqbal Umar, Bahman Zohuri, Payman Zarkesh-Ha, David Modissette, Sahib Zar Khan, and Babar Hussian. 2019. "Wavelet-Based Convolutional Neural Networks for Gender Classification."

Https://Doi.org/10.1117/1.JEI.28.1.013012 28 (January): 013012. https://doi.org/10.1117/1.JEI.28.1.013012.

Bayram, I., and I. W. Selesnick. 2008. "On the Dual-Tree Complex Wavelet Packet and M-Band Transforms." *IEEE Transactions on Signal Processing* 56 (6): 2298–2310. https://doi.org/10.1109/tsp.2007.916129.

Burrus, C. S.., Ramesh A.. Gopinath, and Haitao. Guo. 1998. "Introduction to Wavelets and Wavelet Transforms : A Primer," 268. https://www.mathworks.com/academia/books/introduction-to-wavelets-and-wavelet-transforms-burrus.html.

Chui, Charles K. 2006. "Adapted Wavelet Analysis from Theory to Software (Mladen Victor Wickerhauser)." *Https://Doi.org/10.1137/1038018* 38 (July): 160–60. https://doi.org/10.1137/1038018.

Cornish, Charles R., Christopher S. Bretherton, and Donald B. Percival. 2006. "Maximal Overlap Wavelet Statistical Analysis with Application to Atmospheric Turbulence." *Boundary-Layer Meteorology* 119 (May): 339–74. https://doi.org/10.1007/S10546-005-9011-Y/METRICS.

Daubechies, Ingrid. 1988. "Orthonormal Bases of Compactly Supported Wavelets." *Communications on Pure and Applied Mathematics* 41 (October): 909–96. https://doi.org/10.1002/CPA.3160410705.

———. 1992. *Ten Lectures on Wavelets*. Society for Industrial. https://doi.org/10.1137/1.9781611970104.

Dede, Albert, Henry Nunoo-Mensah, Emmanuel Kofi Akowuah, Kwame Osei Boateng, Prince Ebenezer Adjei, Francisca Adoma Acheampong, Isaac Acquah, and Jerry John Kponyo. 2025. "Wavelet-Based Feature Extraction for Efficient High-Resolution Image Classification." *Engineering Reports* 7 (February): e70027. https://doi.org/10.1002/ENG2.70027.

Gençay, Ramazan, Ramazan Gençay, Faruk Selçuk, and Brandon J. Whitcher. 2001. "An Introduction to Wavelets and Other Filtering Methods in Finance and Economics." *Elsevier Monographs*. https://ideas.repec.org/b/eee/monogr/9780122796708.html https://ideas.repec.org//b/eee/monogr/9780122796708.html.

Grossmann, A., and J. Morlet. 2006. "Decomposition of Hardy Functions into Square Integrable Wavelets of Constant Shape." *Https://Doi.org/10.1137/0515056* 15 (July): 723–36. https://doi.org/10.1137/0515056.

GuyonIsabelle, and ElisseeffAndré. 2003. "An Introduction to Variable and Feature Selection." *The Journal of Machine Learning Research* 3 (March): 1157–82. https://doi.org/10.5555/944919.944968.

Hafez, Ali G., and Tohru Kohda. 2009. "Accurate p-Wave Arrival Detection via MODWT." *International Conference on Communication and Electronics Systems*, 391–96. https://doi.org/10.1109/ICCES.2009.5383235.

Hill, P. R., N. Anantrasirichai, A. Achim, M. E. Al-Mualla, and D. R. Bull. 2015. "Undecimated Dual-Tree Complex Wavelet Transforms." *Signal Processing: Image Communication* 35 (July): 61–70. https://doi.org/10.1016/j.image.2015.04.010.

Jensen, Mark J. 1999. "Using Wavelets to Obtain a Consistent Ordinary Least Squares Estimator of the Long-Memory Parameter." *Journal of Forecasting* 18 (January): 17–32. https://doi.org/10.1002/(SICI)1099-131X(199901)18:1%3C17::AID-FOR686%3E3.0.CO;2-M.

Khalid, Ausama, Yasin Al-Aboosi, and Nor Shahida Mohd Shah. 2024. "Improvement Underwater Acoustic Signal De-Noising Based on Dual-Tree Complex Wavelet Transform." *Journal of Engineering and Sustainable Development* 28 (5): 645–55. https://doi.org/10.31272/jeasd.28.5.10.

Kingsbury, N. G. 1998. "The Dual-Tree Complex Wavelet Transform: A New Technique for Shift Invariance and Directional Filters." In *Proceedings of the 8th IEEE Digital Signal Processing Workshop*. Bryce Canyon, Utah, USA.

Kumar, Praveen, and Efi Foufoula-Georgiou. 1997. "Wavelet Analysis for Geophysical Applications." *Reviews of Geophysics* 35 (November): 385–412. https://doi.org/10.1029/97RG00427.

Laine, Andrew, and Jian Fan. 1993. "Texture Classification by Wavelet Packet Signatures." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15: 1186–91. https://doi.org/10.1109/34.244679.

Lee, Gregory, Ralf Gommers, Filip Waselewski, Kai Wohlfahrt, and Aaron O'Leary. 2019. "PyWavelets: A Python Package for Wavelet Analysis." *Journal of Open Source Software* 4 (36): 1237. https://doi.org/10.21105/joss.01237.

Mallat, S. G.. 1999. "A Wavelet Tour of Signal Processing," 637.

Martis, Roshan Joy, U. Rajendra Acharya, and Lim Choo Min. 2013. "ECG Beat Classification Using PCA, LDA, ICA and Discrete Wavelet

Transform." *Biomedical Signal Processing and Control* 8 (September): 437–48. https://doi.org/10.1016/J.BSPC.2013.01.005.

Meyer, and Yves. 1993. "Wavelets: Algorithms & Applications." *Waa*. https://ui.adsabs.harvard.edu/abs/1993waa..book.....M/abstract.

Nason, Guy P. 2008. *Wavelet Methods in Statistics with R*. Springer New York. https://doi.org/10.1007/978-0-387-75961-6.

Nünez, Jörge, Xavier Otazu, Octavi Fors, Albert Prades, Vicenç Palà, and Roman Arbiol. 1999. "Multiresolution-Based Image Fusion with Additive Wavelet Decomposition." *IEEE Transactions on Geoscience and Remote Sensing* 37: 1204–11. https://doi.org/10.1109/36.763274.

Peng, Z. K., and F. L. Chu. 2004. "Application of the Wavelet Transform in Machine Condition Monitoring and Fault Diagnostics: A Review with Bibliography." *Mechanical Systems and Signal Processing* 18 (March): 199–221. https://doi.org/10.1016/S0888-3270(03)00075-X.

Percival, Donald B. 2008. "Analysis of Geophysical Time Series Using Discrete Wavelet Transforms: An Overview." *Lecture Notes in Earth Sciences* 112: 61–79. https://doi.org/10.1007/978-3-540-78938-3_4.

Percival, Donald B., and Harold O. Mofjeld. 1997. "Analysis of Subtidal Coastal Sea Level Fluctuations Using Wavelets." *Journal of the American Statistical Association* 92 (September): 868–80. https://doi.org/10.1080/01621459.1997.10474042.

Percival, Donald B., and Andrew T. Walden. 2000. "Wavelet Methods for Time Series Analysis." *Wavelet Methods for Time SeriesAnalysis*. https://doi.org/10.1017/CBO9780511841040.

Pistonly. 2025. "Modwtpy: Modwt in Python." https://github.com/pistonly/modwtpy.

Ramsey, James B. 2002. "Wavelets in Economics and Finance: Past and Future." *Studies in Nonlinear Dynamics and Econometrics* 6 (November). https://doi.org/10.2202/1558-3708.1090/MACHINEREADABLECITATION/RIS.

Ravi, Jampani, and R. Narmadha. 2024. "Optimized Dual-Tree Complex Wavelet Transform Aided Multimodal Image Fusion with Adaptive Weighted Average Fusion Strategy." *Scientific Reports* 14 (1). https://doi.org/10.1038/s41598-024-81594-6.

Rioul, O., and M. Vetterli. 1991. "Wavelets and Signal Processing." *IEEE*

Signal Processing Magazine 8 (4): 14–38. https://doi.org/10.1109/79.91217.

Rudnicki, Marek, and Paweł Strumiłło. 2007. "A Real-Time Adaptive Wavelet Transform-Based QRS Complex Detector." *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4432 LNCS: 281–89. https://doi.org/10.1007/978-3-540-71629-7_32.

Selesnick, I. W., R. G. Baraniuk, and N. C. Kingsbury. 2005. "The Dual-Tree Complex Wavelet Transform." *IEEE Signal Processing Magazine* 22 (6): 123–51. https://doi.org/10.1109/msp.2005.1550194.

Selesnick, Ivan W. 2001. "Hilbert Transform Pairs of Wavelet Bases." *IEEE Signal Processing Letters* 8 (June): 170–73. https://doi.org/10.1109/97.923042.

Strang, Gilbert., and T.. Nguyen. 2009. "Wavelets and Filter Banks."

Talbi, Mourad, Riadh Baazaoui, and Brahim Nasraoui. 2024. "A Novel Method of Image Denoising Based on 2D Dual-Tree DWT and SWT." *International Journal of Wavelets, Multiresolution and Information Processing* 22 (04). https://doi.org/10.1142/s0219691324500097.

Telesca, Luciano, Vincenzo Lapenna, and Nikos Alexis. 2004. "Multiresolution Wavelet Analysis of Earthquakes." *Chaos, Solitons & Fractals* 22 (November): 741–48. https://doi.org/10.1016/J.CHAOS.2004.02.021.

Torrence, Christopher, and Gilbert P. Compo. 1998. "A Practical Guide to Wavelet Analysis." *Bulletin of the American Meteorological Society* 79 (January): 61–78. https://doi.org/10.1175/1520-0477(1998)079%3C0061:APGTWA%3E2.0.CO;2.

Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, et al. 2020. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." *Nature Methods* 17 (March): 261–72. https://doi.org/10.1038/S41592-019-0686-2;SUBJMETA.

Wareham, Rich, Nick Kingsbury, and Cian Shaffrey. 2017. "Dtcwt: A Python Dual-Tree Complex Wavelet Transform Library." https://dtcwt.readthedocs.io/en/0.12.0/.

Whitcher, Brandon, Peter Guttorp, and Donald B. Percival. 2000. "Wavelet Analysis of Covariance with Application to Atmospheric Time Series." *Journal of Geophysical Research: Atmospheres* 105 (June): 14941–62.

https://doi.org/10.1029/2000JD900110.

Index

Wavelet Transform in Practice, Volume II-B

Wavelet Transform in Practice, Volume II–B presents advanced wavelet methods for shift–invariant analysis, continuous time–frequency representations, and multiscale feature extraction. Emphasis is placed on advanced transforms, representation selection, and reproducible Python workflows.

This volume covers:

- MODWT and shift–invariant multiresolution analysis
- Complex wavelets and phase-aware representations
- Continuous wavelet transforms (1D–N-D)
- Custom wavelet design and construction
- Wavelet-based feature extraction for machine learning

About the Author

Shouke Wei earned his Ph.D. from Brandenburg University of Technology Cottbus–Senftenberg (Germany) and conducted postdoctoral research at Eawag (Switzerland).
He has held research positions at the University of British Columbia (Canada) and served as a distinguished and adjunct professor at multiple universities (China).

His work focuses on reproducible, deployable wavelet-based methods for analytics and signal processing.

https://press.deepsim.ca/

9 781069 928450