WaveletsExt.jl: Extending the boundaries of wavelets in Julia

Zeng Fung Liew\textsuperscript{1}, Shozen Dan\textsuperscript{1,2}, and Naoki Saito\textsuperscript{3}

\textsuperscript{1}Department of Statistics, University of California, Davis, United States
\textsuperscript{2}Department of Mathematics, Imperial College London, United Kingdom
\textsuperscript{3}Department of Mathematics, University of California, Davis, United States

DOI: 10.21105/joss.03937

Software
- Review
- Repository
- Archive

Editor: Brian McFee

Reviewers:
- @lostanlen
- @malmaud

Submitted: 11 November 2021
Published: 23 January 2022

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

Whether it is seismic surveys, ECG signals, stock market trends, or sensor data, the wavelet and wavelet packet transforms are powerful tools for signal analysis and classification with many advantages over the conventional Fourier methods. Primary among them is the ability to extract information localized in both time and frequency domains, enabling multiresolution analysis (Daubechies, 1992; Mallat, 2009). As such, wavelets and wavelet packets have become popular tools for computational harmonic analysis. WaveletsExt.jl was developed to augment Wavelets.jl (the existing wavelet toolbox for Julia) by providing routines for wavelet analysis, wavelet packet analysis, and associated utilities.

Statement of Need

Julia’s principal package for wavelets is Wavelets.jl (JuliaDSP/Wavelets.jl, 2021), which provides the essential building blocks for data analysis using wavelets. These include 1-D, 2-D, and 3-D wavelet transforms via filter banks or lifting, a range of thresholding functions, and other utilities. However, as a general-purpose package for wavelets, Wavelets.jl does not include many targeted and sophisticated methods present in the literature.

WaveletsExt.jl (Wavelets Extension) enlarges the wavelet toolbox for Julia by providing a host of useful wavelet-based functions such as Stationary Wavelet Transform (Nason & Silverman, 1995), Autocorrelation Wavelet Transform (Saito & Beylkin, 1993), Local Discriminant Basis (Saito & Coifman, 1995), and Shift-invariant Wavelet Packet Decomposition (Cohen et al., 1995). The package also contains denoising utilities such as SureShrink (Donoho & Johnstone, 1995) and Relative Error Shrink (Irion & Saito, 2017) as well as several data visualization utilities.

One of the most distinguishing features of WaveletsExt.jl is the presence of algorithms for handling an ensemble of input signals. Currently, Wavelets.jl implements best basis selection utilities for wavelet packets for a single input. However, it does not include methods for selecting a single best basis for a set of inputs with similar properties (e.g., signals or images belonging to the same class), which is valuable for feature extraction and data compression. To address this, WaveletsExt.jl implements the Joint Best Basis (JBB) (Wickerhauser, 1996) and the Least Statistically Dependent Basis (LSDB) (Saito, 2001), which provide approximations of the Principal Component Analysis (PCA) and Independent Component Analysis (ICA), respectively, in a computationally fast manner through a dictionary of orthonormal bases.

Examples

1. Redundant Wavelet Transforms

WaveletsExt.jl implements several redundant wavelet transforms including Autocorrelation Wavelet Transform (Saito & Beylkin, 1993) and Stationary Wavelet Transform (SWT) (Nason & Silverman, 1995). These transformations can be performed using the acdwt and sdwt functions, and the resulting decomposition can be visualized with the wiggle function included in WaveletsExt.jl.

```julia
using Plots, Wavelets, WaveletsExt

x = zeros(1<<8) # Generate a unit impulse (dirac delta) signal
x[128] = 1
wt = wavelet(WT.db4) # Construct Daubechies 4-tap wavelet filter

# ----- Autocorrelation Wavelet Transforms -----
y = acdwt(x, wt)
p1 = wiggle(y) |> p -> plot!(p, yticks=1:9, title="Autocorrelation WT")

# ----- Stationary Wavelet Transforms -----
y = sdwt(x, wt)
p2 = wiggle(y) |> p -> plot!(p, yticks=1:9, title="Stationary WT")

# Combine and save plot
p = plot(p1, p2, layout=(1,2), size=(600,300))
savefig(p, "transforms.png")
```

Figure 1: “Wiggle” plots displaying the value of coefficients at each level of the autocorrelation and stationary wavelet transform for a unit impulse signal.

2. Best Basis Algorithms

WaveletsExt.jl can select a best basis for a multiple signal input (i.e., an array of signals) through the Joint Best Basis (JBB) (Wickerhauser, 1996) or Least Statistically Dependent Basis (LSDB) (Saito, 2001) algorithms. The resulting best basis tree can be visualized using plot_tfbdry also included in WaveletsExt.jl.

```julia
using Plots, Wavelets, WaveletsExt

# Generate 100 noisy heavysine signals of length 2
x = generesignals(:heavysine, 8) |> x -> duplicatesignals(x, 100, 2, true, 0.5)

# Wavelet packet decomposition of all signals
xw = wpdall(x, wt, 6)

# ----- Joint Best Basis (JBB)
tree = bestbasistree(xw, JBB())
p1 = plot_tfbdry(tree,
               node_color=:green,
               line_color=:black,
               background_color=:white) |> p -> plot!(p, title="JBB")

# ----- Least Statistically Dependent Basis (LSDB)
tree = bestbasistree(xw, LSDB())
p2 = plot_tfbdry(tree,
               node_color=:green,
               line_color=:black,
               background_color=:white) |> p -> plot!(p, title="LSDB")

# Combine and save plot
p = plot(p1, p2, layout=(1,2), size=(600,300))
savefig(p, "bestbasis.png")
```

3. Denoising Algorithms

WaveletsExt.jl contains two functions for denoising: denoise and denoiseall. The former denoises a single signal input whereas the latter denoises multiple signal input. For more examples of denoising algorithms in WaveletsExt.jl, see (Liew et al., 2021).

using Plots, Wavelets, WaveletsExt

# Generate 6 circularly shifted HeaviSine signals
x = generatesignals(:heavisine, 8) |> x -> duplicatesignals(x, 6, 2, false)

# Generate 6 noisy versions of the original signals
x = generatesignals(:heavisine, 8) |> x -> duplicatesignals(x, 6, 2, true, 0.8)

# Decompose each noisy signal
xw = wpdall(x, wt)

# Get best basis tree from the decomposition of signals
bt = bestbasistree(xw, JBB())

# Get best basis coefficients based on best basis tree
y = bestbasiscoef(xw, bt)

# Denoise all signals based on computed best basis tree

Figure 2: The best basis trees of 100 HeaviSine signals (A sinusoid + two Heaviside step functions) (Buckheit & Donoho, 1995; Donoho & Johnstone, 1995) selected by the JBB and LSDB algorithms. Each row represents a decomposition level, where level 0 is the original input signal, and each cell represents a frequency subband (low to high frequency from left to right). The colored cells indicate those subbands selected by the JBB (left) and the LSDB (right) algorithms.
\hat{x} = \text{denoiseall}(y, \text{:wpt, wt, tree}=bt)

# Plot results
p1 = plot(title="Noisy Signals")
wiggle!(x, sc=0.7, FaceColor=:white, ZDir=:reverse)
wiggle!(x, sc=0.7, FaceColor=:white, ZDir=:reverse)

p2 = plot(title="Denoised Signals")
wiggle!(\hat{x}, sc=0.7, FaceColor=:white, ZDir=:reverse)
wiggle!(\hat{x}, sc=0.7, FaceColor=:white, ZDir=:reverse)

# Combine and save plot
p = plot(p1, p2, layout=(1,2), size=(600,300))
savefig(p, "denoising.png")

Figure 3: Left: HeaviSine signals with Gaussian noise. Black lines represent the original (non-noisy) signal. Right: Simultaneously denoised signals using the JBB algorithm with a universal thresholding constant determined by the VisuShrink method (Donoho & Johnstone, 1994).

4. Feature Extraction

For signal classification problems, users can extract distinguishing features localized in both the time and frequency domains using the Local Discriminant Basis (LDB) algorithm. Further details can be found in the original papers by Saito and his collaborators (Saito et al., 2002; Saito & Coifman, 1995) as well as the interactive tutorial (Dan et al., 2021).

using Plots, Wavelets, WaveletsExt

# Generate 100 signals for each class of cylinder-bell-funnel
X, y = generateclassdata(ClassData(:cbf, 100, 100, 100))
# View sample signals and how each class differs from one another
cylinder = wiggle(X[:,1:5], sc=0.3, EdgeColor='white', FaceColor='white')
plot!(cylinder, yticks=1:5, label="Cylinder")
bell = wiggle(X[:,101:105], sc=0.3, EdgeColor='white', FaceColor='white')
plot!(bell, yticks=1:5, label="Bell")
funnel = wiggle(X[:,201:205], sc=0.3, EdgeColor='white', FaceColor='white')
plot!(funnel, yticks=1:5, label="Funnel")
p1 = plot(cylinder, bell, funnel, layout=(3,1))

# Instantiate the LDB object
wt = wavelet(WT.coif4)
ldb = LocalDiscriminantBasis(
    wt=wt,
    max_dec_level=6,
    dm=SymmetricRelativeEntropy(),
    en=TimeFrequency(),
    dp=BasisDiscriminantMeasure(),
    top_k=10,
    n_features=10 # Number of features to extract
)

# Extract features using LDB
X̂ = fit_transform(ldb, X, y)

# Plot the best basis for feature extraction
p2 = plot_tfbdry(ldb.tree,
    node_color=:green,
    line_color=:black,
    background_color=:white)
plot!(p2, title="Basis Selection using LDB")

# Combine and save plot
p = plot(p1, p2, size=(600,300))
savefig(p, "ldb.png")

---

Reproducible Research

WaveletsExt.jl was partially inspired by the WaveLab library in MATLAB, which was developed to enable reproducible wavelet research (Buckheit & Donoho, 1995). In this spirit, we wrote a series of tutorials, examples, and experiments using Pluto.jl (Dan et al., 2021; Liew et al., 2021), a platform with which Julia users can create and share reactive documents (Van der Plas, 2021). By downloading and running these so-called Pluto notebooks, researchers and students alike can reproduce the results of our research and interactively adjust parameters to see the changes in experiment outcomes.

Acknowledgements

This project was partially supported by the following grants from the US National Science Foundation: DMS-1148643; DMS-1418779; DMS-1912747; and CCF-1934568.

References


Saito, N. (2001). Image approximation and modeling via least statistically dependent bases. *Pattern Recognition, 34*(9), 1765–1784. [https://doi.org/10.1016/S0031-3203(00)00116-3](https://doi.org/10.1016/S0031-3203(00)00116-3)

Saito, N., & Beylkin, G. (1993). Multiresolution representations using the autocorrelation functions of compactly supported wavelets. *IEEE Transactions on Signal Processing, 41*(12), 3584–3590. [https://doi.org/10.1109/78.258102](https://doi.org/10.1109/78.258102)


