Visual Testing Procedure for PyDelt

PyDelt Research Team

July 14, 2025

Abstract

This document outlines the visual testing procedure for the PyDelt package, a Python library for calculating derivatives and integrals of time series data. The visual tests are designed to complement the traditional unit tests by providing interactive visualizations that help assess the performance of different algorithms under various conditions, including in the presence of noise.

1 Introduction

PyDelt provides several methods for calculating derivatives and integrals of time series data. While traditional unit tests verify that these methods produce results within acceptable error bounds, visual tests provide additional insights into how these methods perform under different conditions, particularly in the presence of noise.

The visual tests generate interactive HTML plots using Plotly, which allow researchers to:

- Visually compare the performance of different algorithms
- Assess the impact of noise on algorithm performance
- Evaluate the effect of parameter choices (e.g., window size)
- Examine the full pipeline from derivative calculation to signal reconstruction

2 Testing Framework

2.1 Directory Structure

The visual testing framework is organized as follows:

```
pydelt/
  local/
    tests/
      visual_test_derivatives.py
      visual_test_integrals.py
      run_visual_tests.py
    output/
      [generated HTML files]
    research/
      [documentation and research notes]
```

2.2 Test Types

The visual tests are divided into two main categories:

2.2.1 Derivative Tests

- Basic tests for each algorithm (LLA, GOLD, GLLA, FDA)
- Algorithm comparison on clean data
- Noise impact analysis across algorithms
- Window size effect analysis for LLA

2.2.2 Integral Tests

- Basic integration tests (constant, sine)
- Initial value handling
- Error estimation
- Input type handling
- Noise impact on integration
- Full pipeline: derivative calculation and signal reconstruction

3 Noise Testing Methodology

3.1 Noise Generation

Gaussian noise is added to the signals using the following function:

```python
def add_noise(signal, noise_level):
    """Add Gaussian noise to a signal."""
    np.random.seed(42)  # For reproducibility
    noise = np.random.normal(0, noise_level, size=signal.shape)
    return signal + noise
```

The noise levels used in the tests are: 0.01, 0.05, 0.1, and 0.2, representing increasing levels of signal corruption.

3.2 Derivative Algorithm Performance Under Noise

The test `visual_test_noise_comparison` evaluates how different derivative algorithms (LLA, GOLD, GLLA, FDA) handle varying levels of noise. For each noise level, the test:

1. Generates a noisy sine wave
2. Calculates derivatives using each algorithm
3. Compares the results to the expected derivative (cosine)
4. Displays both the derivative results and the original noisy signal

3.3 Window Size Effect on Noise Handling

The test `visual_test_window_size_comparison` examines how the window size parameter in the LLA algorithm affects its ability to handle noise. Larger window sizes generally provide more smoothing but may reduce responsiveness to rapid changes.

3.4 Integration with Noisy Derivatives

The test `visual_test_noise_effect_on_integration` evaluates how noise in the derivative affects integration results. This is important because noise in derivatives can accumulate during integration.

3.5 Signal Reconstruction from Noisy Data

The test `visual_test_derivative_reconstruction_with_noise` demonstrates the full pipeline:

1. Start with a noisy signal
2. Calculate its derivative
3. Reconstruct the original signal by integrating the derivative
4. Compare the reconstructed signal to both the noisy original and the clean signal

4 Running the Tests

4.1 Prerequisites

- Python 3.8 or higher
- PyDelt package and its dependencies
- Plotly for visualization

4.2 Execution

To run the visual tests:

```
# Activate the virtual environment
source venv/bin/activate

# Run all visual tests
python local/tests/run_visual_tests.py

# Run specific test files
python local/tests/visual_test_derivatives.py
python local/tests/visual_test_integrals.py
```

4.3 Output

The tests generate HTML files in the `local/output` directory. These files can be opened in any web browser to interactively explore the plots. The interactive features include:

- Zooming in/out
- Panning
- Hovering for data point values
- Toggling visibility of individual traces
- Downloading as PNG

5 Interpreting Results

5.1 Derivative Algorithm Comparison

When comparing derivative algorithms, consider:

- Accuracy: How close is the calculated derivative to the expected value?
- Noise sensitivity: How much does the algorithm's performance degrade with increasing noise?
- Boundary effects: How does the algorithm handle the edges of the data?
- Computational efficiency: How long does the algorithm take to run?

5.2 Integration Performance

When evaluating integration performance, consider:

- Accuracy: How close is the integrated result to the expected function?
- Error accumulation: How does error accumulate over the integration domain?
- Initial value sensitivity: How does the choice of initial value affect the result?
- Noise handling: How well does the integration handle noisy derivatives?

6 Conclusion

Visual testing provides valuable insights that complement traditional unit tests. By visualizing the performance of different algorithms under various conditions, researchers can make informed decisions about which methods to use for specific applications.

The PyDelt visual testing framework is designed to be extensible. New tests can be added to evaluate additional aspects of the package's performance or to compare new algorithms with existing ones.