PyELW: Exact Local Whittle Estimation for Long Memory Time Series in **Python**

Jason R. Blevins
The Ohio State University
August 29, 2025

Abstract

Fractionally integrated time series, characterized by slowly decaying autocorrelations and spectral densities exhibiting power-law behavior at low frequencies, require accurate estimation of the memory parameter d to distinguish between stationary long memory ($0 < d < 0.5$), nonstationary processes ($0.5 \leq d < 1$), and unit root behavior ($d \geq 1$). This paper introduces **PyELW**, a Python package for local Whittle estimation of the memory parameter d including the foundational estimator of Robinson (1995), tapered variants by Velasco (1999) and Hurvich and Chen (2000), the exact local Whittle estimator of Shimotsu and Phillips (2005), and the two-step estimator of Shimotsu (2010). While a package exists for Stata and implementations are available for R and MATLAB, these are either limited-scope or no longer maintained and there was previously no Python implementation of these methods. **PyELW** provides a wide array of local Whittle estimators in a single package, featuring fast $O(n \log n)$ fractional differencing, a consistent, object-oriented API with theoretically motivated defaults, and extensive validation through exact replication of previously-published results and rigorous cross-platform verification. We demonstrate the package's usage through simulations and applications to macroeconomic time series.

Keywords: time series, fractional integration, fractional differencing, long memory, local Whittle estimation, Python.

1. Introduction

Long memory time series are characterized by slowly decaying autocorrelations and spectral densities exhibiting power-law behavior at low frequencies. These processes, often modeled as fractionally integrated processes, arise naturally in many fields including economics, finance, and climate science. Accurate semiparametric estimation of the long memory parameter, without relying on parametric distributional assumptions, is important for robust understanding of the persistence properties of such series.

The presence of long memory has been documented extensively in empirical applications. For example, daily realized volatilities of foreign exchange rates, computed from 5-minute returns, exhibit fractional integration with $d \approx 0.4$, indicating persistent but stationary long-memory dynamics (Andersen, Bollerslev, Diebold, and Labys 2001). Climate series such as average annual temperatures display moderate long memory persistence with estimates of d from 0.2 to 0.4, indicating stationary but highly persistent behavior that has important implications for climate modeling and forecasting (Baillie and Chung 2002). Economic time series such as inflation rates exhibit intermediate memory, with d estimates between 0.4 and 0.6, challenging the traditional $I(0)/I(1)$ dichotomy (Hassler and Wolters 1995). Local Whittle estimation provides a robust semiparametric framework for distinguishing between stationary long memory ($0 < d < 0.5$), nonstationary but mean-reverting processes ($0.5 \leq d < 1$), and unit root or explosive behavior ($d \geq 1$) across applications such as these.

While the statistical theory for semiparametric estimation of the fractional integration parameter has advanced significantly over the past three decades, beginning with Geweke and Porter-Hudak (1983) and Robinson (1995), practical implementation remains more limited and varies across software platforms. The computational challenges are non-trivial but manageable. Efficient fractional differencing algorithms with $O(n \log n)$ complexity are available, and while no closed-form solutions exist for d, the scalar nature of the optimization problem makes reliable estimation feasible. However, the various tapered estimators involve specialized window functions, potentially complex-valued, and frequency subsampling techniques that require careful implementation. Additionally, there are important subtleties regarding the treatment of unknown means and trends.

1.1. Existing packages

		Methods					
Package	Language	LW	V	HC	ELW	2ELW	Notes
whittle	Stata	✓			✓		Proprietary, LW and ELW only
elwcode10	MATLAB	✓	✓		✓	✓	Shimotsu's reference implementation
LongMemoryTS	R	✓	✓	✓	✓	✓	Removed from CRAN; HC and 2ELW contained errors
PyELW	Python	✓	✓	✓	✓	✓	Comprehensive, packaged implementation

Table 1: Software packages for local Whittle and related estimation methods. Note: LW = local Whittle, V = Velasco, HC = Hurvich-Chen, ELW = exact local Whittle, 2ELW = two-step exact local Whittle.

Table 1.1 reveals a limited software landscape with a critical gap in the Python ecosystem. Stata's **whittle** command, introduced by Baum, Hurn, and Lindsay (2020), provides reliable implementations of LW and ELW, but requires proprietary software and does not implement many other methods we consider. We also found that it is unable to estimate parameters for unit root processes and beyond, $d \geq 1$, due to limitations in the built-in fractional differencing routine. The MATLAB implementation of Shimotsu (2010) implements several methods, but is aimed at academic users without formal packaging, and also requires proprietary software.[1] The most comprehensive package, **LongMemoryTS**, was available for R but has been removed from CRAN. Our analysis revealed implementation errors in its HC taper and two-step ELW

[1]However, we were able to use several of the functions in GNU Octave with little difficulty.

estimators, which we describe in Appendix A. Other related packages with a different focus include the R packages **longmemo** and **fracdiff** as well as the Python package **whittlehurst**. These packages include methods for classical parametric Whittle estimation of fGn and/or ARFIMA models, but do not include semiparametric local Whittle methods which are our focus.

PyELW addresses this implementation gap by providing a comprehensive, open-source Python implementation with several key contributions. First, it implements all major local Whittle variants—untapered, tapered variants (Velasco, Hurvich-Chen), exact local Whittle, and two-step ELW—with algorithms verified against published empirical and Monte Carlo results. Second, it provides efficient $O(n \log n)$ fractional differencing following Jensen and Nielsen (2014), enabling fast preprocessing, nested fractional differentiation, and simulation of ARFIMA processes. Third, the package offers a consistent object-oriented API with sensible defaults for bandwidth selection, parameter bounds, and standard error calculation, making it accessible to both practitioners and researchers. Fourth, **PyELW** includes comprehensive documentation, unit tests, and replication scripts for key papers in the literature.

1.2. Organization of paper

The remainder of this paper is organized as follows. Section 2 provides a methodological overview of semiparametric estimation for fractionally integrated processes, covering the evolution from the basic local Whittle estimator through tapered variants to exact local Whittle methods. Section 3 describes the **PyELW** package, including the API, implementation details, and examples for each estimator. Section 4 details our comprehensive validation approach, including replication of Monte Carlo and empirical results from the literature, comparison with existing implementations across software platforms, and extensive unit testing with both simulated and real data. Section 5 concludes.

2. Semiparametric estimation of fractional integration

Formally, consider the discrete-time fractionally integrated process:

$$(1 - L)^{d_0} X_t = u_t 1\{t \geq 1\}, \quad t = 0, \pm 1, \pm 2, \ldots$$

where L denotes the lag operator, $d_0 \in \mathbb{R}$ represents the true fractional differencing parameter, $\mathbf{1}\{\cdot\}$ is the indicator function (with $X_t = 0$ for $t \leq 0$ as the initial condition), and $\{u_t\}$ is a stationary process with spectral density $f_u(\lambda)$ that captures short-run dynamics. The parameter d_0 governs the memory properties: $d_0 = 0$ yields standard stationary ARMA behavior, $d_0 \in (0, 0.5)$ produces stationary long memory, $d_0 = 0.5$ represents the boundary between stationarity and nonstationarity, $d_0 \in (0.5, 1)$ generates nonstationary mean-reverting behavior, and $d_0 = 1$ corresponds to the familiar unit root case. Negative values $d_0 \in (-0.5, 0)$ represent antipersistent or intermediate memory processes.

For observations $\{X_t\}_{t=1}^n$, the discrete Fourier transform at frequencies $\lambda_j = 2\pi j/n$ for $j = 1, \ldots, n-1$ is defined as:[2]

$$w_X(\lambda_j) = \frac{1}{\sqrt{2\pi n}} \sum_{t=1}^{n} X_t e^{-i\lambda_j t}.$$

[2]Following Robinson (1995), we use the normalization factor $(2\pi n)^{-1/2}$ throughout.

The periodogram $I_X(\lambda_j) = |w_X(\lambda_j)|^2$ provides a natural estimator of the spectral density $f_X(\lambda_j)$.

The Whittle likelihood (Whittle 1951) approximates the Gaussian likelihood in the frequency domain by exploiting the fact that the Fourier coefficients $w_X(\lambda_j)$ are asymptotically independent complex Gaussian random variables with variance $f_X(\lambda_j)$. For a parametric model with spectral density $f_X(\lambda_j;\theta)$ where $\theta = (d,\psi)$ consists of the memory parameter d and nuisance parameters ψ governing short-run dynamics, the Whittle log-likelihood is:

$$L_{\mathrm{W}}(\theta) = -\sum_{j=1}^{\lfloor n/2 \rfloor} \left[\log f_X(\lambda_j;\theta) + \frac{I_X(\lambda_j)}{f_X(\lambda_j;\theta)} \right].$$

This frequency-domain approach reduces computational complexity from $O(n^3)$ for time-domain maximum likelihood to $O(n \log n)$, while maintaining consistency and asymptotic efficiency under regularity conditions. However, it requires fully specifying the parametric form of the spectral density. The semiparametric approaches implemented in **PyELW** exploit the special spectral structure of fractionally integrated processes to estimate d without requiring a full parametric model.

2.1. The local Whittle estimator

A fundamental insight of Granger and Joyeux (1980) and Hosking (1981) is that fractionally integrated processes have a distinctive spectral shape at low frequencies. Specifically, the spectral density of X_t satisfies

$$f_X(\lambda) \sim G\lambda^{-2d_0} \quad \text{as} \quad \lambda \to 0^+ \tag{1}$$

where $G = f_u(0) > 0$, meaning that u_t is not overdifferenced. This power law behavior characterizes the long memory property: the spectral density diverges as λ^{-2d_0} for $d_0 > 0$, capturing the slow decay of autocorrelations.

The challenge in using this relationship to estimate d_0 is that the power law behavior in (1) only holds at frequencies near zero. At higher frequencies, the spectrum is dominated by short-run dynamics that we wish to avoid specifying parametrically. Robinson (1995) developed the local Whittle (LW) estimator by restricting attention to only the first $m = m(n) \ll n$ Fourier frequencies where (1) holds most accurately. Substituting the approximation $f_X(\lambda_j) \approx G\lambda_j^{-2d}$ into the Whittle likelihood and maximizing with respect to G yields $\hat{G}(d) = \frac{1}{m}\sum_{j=1}^{m} \lambda_j^{2d} I_X(\lambda_j)$. Substituting this back gives the concentrated local Whittle objective function:

$$R(d) = \log\left(\frac{1}{m}\sum_{j=1}^{m} \lambda_j^{2d} I_X(\lambda_j)\right) - \frac{2d}{m}\sum_{j=1}^{m} \log \lambda_j.$$

The objective function $R(d)$ is convex in d (Baum *et al.* 2020), ensuring that any local minimum is a global minimum, making optimization straightforward and reliable. Furthermore, the periodogram is independent of d and can be pre-computed, meaning that the objective function requires only $O(m)$ operations per evaluation.

The local Whittle estimator is defined as:

$$\hat{d}_{\mathrm{LW}} = \arg \min_{d \in [\Delta_1, \Delta_2]} R(d)$$

where $[\Delta_1, \Delta_2]$ denotes the admissible parameter space. Robinson (1995) established regularity conditions such that for $d_0 \in (-0.5, 0.5)$, the local Whittle estimator is consistent, converges at the optimal $\sqrt{m}$ rate, and achieves the semiparametric efficiency bound:

$$\sqrt{m}(\hat{d}_{LW} - d_0) \xrightarrow{\mathrm{d}} \mathrm{N}(0, 1/4)$$

The bandwidth parameter m controls the bias-variance tradeoff: larger m reduces variance but increases bias as higher frequencies contaminated by short-run dynamics enter the estimation. The result above requires $m \to \infty$ and $m/n \to 0$ as $n \to \infty$ to balance these concerns. Typical choices follow $m = \lfloor n^\alpha \rfloor$ with $\alpha \in (0.6, 0.8)$. See Henry and Robinson (1996) for details on optimal bandwidth selection. Following the Stata implementation, **PyELW** uses $m = \lfloor n^{0.65} \rfloor$ as the default bandwidth, but allows the user to customize this.

2.2. Tapered local Whittle estimators

Although the LW estimator is efficient for $d_0 \in (-0.5, 0.5)$, this range excludes important processes beyond the nonstationary boundary. Subsequent developments following Robinson (1995) aimed to relax this limitation while preserving the desirable asymptotic properties. Tapering offers a solution: by down-weighting observations near the sample boundaries through multiplication by a smooth function that decays to zero at the edges, we can reduce the spectral distortion that occurs from polynomial time trends. Formally, tapering involves scaling the original series X_t by a weight sequence h_t that is symmetric around $\lfloor n/2 \rfloor$, satisfies $\max h_t = 1$, and decays smoothly to zero at the sample boundaries. The tapered periodogram $I_h(\lambda_j) = |w_h(\lambda_j)|^2$ where $w_h(\lambda_j) = (2\pi n)^{-1/2} \sum_{t=1}^n h_t X_t e^{-i\lambda_j t}$ then replaces the standard periodogram in the objective function. That is, the tapered local Whittle estimator minimizes:

$$R_h(d) = \log\left(\frac{1}{m}\sum_{j=1}^m \lambda_j^{2d} I_h(\lambda_j)\right) - \frac{2d}{m}\sum_{j=1}^m \log \lambda_j.$$

Velasco (1999) demonstrated that tapering extends consistency to $d_0 \in (-0.5, 1)$ and asymptotic normality to $d_0 < 0.75$, though at the cost of increased variance (at least 2.1 times the LW variance). Using tapers of higher orders p can extend the feasible range of d_0 values and provide robustness to trends of order $p-1$. Velasco (1999) discusses the triangular Bartlett window taper ($p = 2$), which is valid for $d_0 \in (-0.5, 2.5)$ and robust to linear trends, as well as the cosine bell and Zhurbenko-Kolmogorov tapers ($p = 3$), which are valid for $d_0 \in (-0.5, 3.5)$ and robust to linear and quadratic time trends. For Velasco tapers, the periodogram is subsampled at frequencies $\lambda_{jp} = 2\pi jp/n$ for $j = 1, \ldots, m$, where p is the taper order.

Hurvich and Chen (2000) developed a tapered local Whittle estimator that achieves asymptotic variance of only $1.5/4m$ for $d_0 \in (-0.5, 1.5)$. The Hurvich-Chen (HC) approach applies a complex-valued taper to first-differenced data $\Delta X_t = X_t - X_{t-1}$, then adds back one degree of integration in the estimation. This differencing step ensures validity for d_0 up to 1.5, while the shifted frequencies $\lambda_{\tilde{j}} = 2\pi(j + 0.5)/n$ avoid the singularity at zero frequency that would otherwise cause numerical problems for the overdifferenced series when $d_0 < 1$.

2.3. The exact local Whittle estimator

The exact local Whittle (ELW) estimator of Shimotsu and Phillips (2005) addresses a fundamental limitation of the LW estimator. The LW approach relies on the approximation $I_X(\lambda_j) \approx G\lambda_j^{-2d_0} I_u(\lambda_j)$, which is only valid for stationary processes ($d_0 < 0.5$). The ELW estimator addresses this by working directly with the fractionally differenced series. Since $(1-L)^{d_0} X_t = u_t$, it follows that $(1-L)^d X_t = (1-L)^{d-d_0} u_t$ and for $d = d_0$, we recover u_t exactly. By computing $\Delta^d X_t$ for each value of d and using its periodogram $I_{\Delta^d X}(\lambda_j)$, we work with the exact relationship.

The fractional differencing operator is defined through the generalized binomial expansion:

$$\Delta^d X_t = (1-L)^d X_t = \sum_{k=0}^{t-1} \pi_k(d) X_{t-k}$$

where the coefficients can be computed recursively via $\pi_0(d) = 1$ and

$$\pi_k(d) = \pi_{k-1}(d) \cdot \frac{k-1-d}{k} \quad \text{for } k \geq 1.$$

For positive d, this operation “differences” the series by a fractional amount, while for negative d, it performs fractional “summation” or integration.

The ELW objective function has the same functional form as that of LW, but uses the periodogram of the fractionally differenced series:

$$R_{\text{ELW}}(d) = \log\left(\frac{1}{m}\sum_{j=1}^{m} I_{\Delta^d X}(\lambda_j)\right) - \frac{2d}{m}\sum_{j=1}^{m} \log \lambda_j$$

where $I_{\Delta^d X}(\lambda_j)$ denotes the periodogram of $\Delta^d X_t$. The ELW estimator is then:

$$\hat{d}_{\text{ELW}} = \arg \min_{d \in [\Delta_1, \Delta_2]} R_{\text{ELW}}(d)$$

where $[\Delta_1, \Delta_2]$ denotes any interval of length at most $9/2$. Therefore, the advantage of the ELW estimator is that we can consistently estimate d_0 whether it lies in the stationary region, the nonstationary region, or even exactly at the boundary $d_0 = 0.5$, provided that the optimization interval contains the true value. Furthermore, the ELW estimator retains the efficient limiting distribution of the LW estimator despite this expanded range:

$$\sqrt{m}(\hat{d}_{\text{ELW}} - d_0) \xrightarrow{\text{d}} \text{N}(0, 1/4).$$

The computational cost of ELW is higher than for the LW estimator, which only required one precomputed periodogram calculation and no fractional differencing operations. For the ELW estimator, for each candidate value of d during optimization we must recompute the fractional difference $\Delta^d X_t$ and the periodogram $I_{\Delta^d X}$. While computing $\Delta^d X_t$ naively requires $O(n^2)$ operations, **PyELW** implements the fast fractional differencing algorithm of Jensen and Nielsen (2014) which reduces this to $O(n \log n)$ using the fast Fourier transform. For this additional computational cost, the benefit is consistent estimation for both stationary and nonstationary processes with the same optimal asymptotic variance.

Although Shimotsu and Phillips (2005) assumed the mean of the process was known and equal to zero, Shimotsu (2010) considered the behavior of the ELW estimator with two estimators for an unknown mean. When using the sample mean $\widehat{\mu} = \bar{X}$, Shimotsu (2010) demonstrates consistency for $d_0 \in (-1/2, 1)$ and asymptotic normality for $d_0 \in (-1/2, 3/4)$. In contrast, when using the initial observation $\widehat{\mu} = X_1$, both consistency and asymptotic normality hold for $d_0 > 0$, providing broader coverage in the nonstationary region but requiring positive memory parameters.

2.4. Two-step exact local Whittle estimator

The previous estimators assume that the stochastic process $\{X_t\}$ has zero mean and does not contain a trend. However, real economic time series typically have unknown means or trends that can bias fractional integration estimates. Shimotsu (2010) developed the two-step exact local Whittle (2ELW) estimator to address these issues while preserving the optimal $\mathrm{N}(0, 1/4)$ limiting distribution.

Consider the model with unknown mean:

$$X_t = \mu_0 + X_t^0, \quad X_t^0 = (1 - L)^{-d_0} u_t 1\{t \geq 1\}$$

where μ_0 is an unknown constant. The problem is subtle, because the best way to handle the unknown mean depends on the unknown value of d_0 itself. For stationary processes ($d_0 < 0.5$), the sample average $\bar{X}$ provides the best mean estimate, but for highly persistent processes ($d_0 > 0.75$) the first observation X_1 is better, as the sample mean becomes biased. To address this, Shimotsu (2010) proposed an adaptive mean estimation step inside the objective function that varies smoothly with the value of d:

$$\tilde{\mu}(d) = w(d)\bar{X} + (1 - w(d))X_1,$$

where $w(d)$ is the weight function:

$$w(d) = \begin{cases} 1 & \text{if } d \leq 0.5, \\ \frac{1}{2}[1 + \cos(4\pi d)] & \text{if } 0.5 < d < 0.75, \\ 0 & \text{if } d \geq 0.75. \end{cases}$$

This creates a modified ELW objective function that estimates the mean jointly with d_0:

$$R_F(d) = \log\left(\frac{1}{m}\sum_{j=1}^{m} I_{\Delta^d(X - \tilde{\mu}(d))}(\lambda_j)\right) - \frac{2d}{m}\sum_{j=1}^{m}\log\lambda_j$$

Estimation of the mean in this way rendered direct asymptotic theory difficult, so Shimotsu (2010) instead proposed a two-step approach based on a $\sqrt{m}$-consistent first-step estimator—one of the tapered local Whittle estimators discussed above—to obtain an initial estimate $\hat{d}_T$. This estimate need not be efficient, only consistent. The second step applies a single Newton-Raphson step to the modified ELW objective function starting from $\hat{d}_T$:

$$\hat{d}_{2\mathrm{ELW}} = \hat{d}_T - \frac{R_F'(\hat{d}_T)}{R_F''(\hat{d}_T)}$$

where R_F' and R_F'' denote the first and second derivatives of R_F. While one Newton-Raphson step is sufficient for the asymptotic theory, practitioners often iterate until convergence. This procedure achieves the optimal $\mathrm{N}(0, 1/4)$ limiting distribution for $d_0 \in (-1/2, 2)$, dramatically expanding the feasible range while maintaining efficiency and robustness to the unknown mean.

In practice, the Hessian $R''_F(\hat{d}_T)$ can become very small or negative in finite samples, leading to unstable Newton-Raphson updates, as documented by Shimotsu (2010) and Baum *et al.* (2020). For robustness, **PyELW** instead uses golden section search to minimize $R_F(d)$ within a 99% confidence interval around the first-step tapered LW estimate, which avoids numerical instabilities while preserving the asymptotic properties.

For data containing polynomial trends of order p, the procedure first removes the trend via OLS regression on $(1, t, t^2, \ldots, t^p)$, then applies the two-step estimator to the residuals. This detrending preserves consistency but restricts the feasible range to $d_0 \in (-0.5, 2 - p/2)$, yielding $d_0 \in (-0.5, 1.5)$ for linear detrending.

2.5. Computational considerations

With the landscape of local Whittle estimators established, practical implementation requires decisions about computational matters that can impact accuracy and performance. In terms of the local Whittle estimators we consider, these relate to choices about fractional differencing, optimization of scalar objective functions, and parameter search boundaries.

The fractional differencing operation $\Delta^d X_t = (1 - L)^d X_t$ involves an infinite binomial expansion that must be truncated in practice. The naive approach using direct binomial coefficients has $O(n^2)$ complexity, so we use the fast fractional differencing algorithm developed by Jensen and Nielsen (2014), which is an $O(n \log n)$ method based on FFT convolution.

For minimizing the various local Whittle objective functions introduced above, **PyELW** uses golden section search for univariate optimization. This provides guaranteed convergence without requiring derivatives. For each estimator, we provide default search bounds that balance empirical relevance and theory, but we allow the user to override the defaults.

3. Package description and examples

The **PyELW** package provides a comprehensive and efficient implementation of local Whittle and exact local Whittle estimation methods for fractionally integrated time series. The package leverages NumPy for efficient, vectorized array operations. The package is available under the BSD licence and can be obtained at `https://github.com/jrblevin/pyelw`. To install **PyELW**:

```
pip install pyelw
```

This section first discusses each of the three main estimator classes, provides an overview of their initialization and usage in estimation, introduces two auxiliary functions for fractional differencing and ARFIMA simulation, and then proceeds through three detailed usage examples.

3.1. Main estimator classes

PyELW provides three main estimator classes that implement different local Whittle estimator variants: `LW`, `ELW`, and `TwoStepELW`. These classes, summarized in Table 2, have a consistent

Class	**Method**	**Citations**
`LW`	Local Whittle	Untapered: Robinson (1995) Tapered: Velasco (1999), Hurvich and Chen (2000)
`ELW`	Exact local Whittle	Shimotsu and Phillips (2005)
`TwoStepELW`	Two-step ELW	Shimotsu (2010)

Table 2: Main estimator classes in **PyELW**

API: the main `estimate` method accepts as inputs the time series data `X` and, optionally the bandwidth `m` and parameter `bounds`. It returns a result dictionary containing the parameter estimate, standard error, diagnostic information, and any method-specific return values.

The time series should be provided as a **NumPy** `ndarray`. Only the data `X` is required. The default bandwidth is `m = int(n**0.65)` and the default bounds are chosen to cover most common processes encountered without being overly wide.

Arguments of the `estimate` method:

- `X`: Input time series data as `numpy.ndarray`
- `m`: Number of frequencies to use (default: $\lfloor n \rfloor^{0.65}$)
- `bounds`: Tuple (d_{min}, d_{max}) for optimization bounds (default: $(-1.0, 2.2)$)

Return value of the `estimate` method: a `dict` with keys

- `'d_hat'`: Estimate of memory parameter d
- `'se'`: Standard error based on Fisher information
- `'ase'`: Standard error based on asymptotic theory
- `'objective'`: Final objective function value
- `'method'`: String identifier for estimation method used
- `'nfev'`: Number of objective function evaluations
- `'n'`: Number of time series observations
- `'m'`: Number of frequencies used
- Additional method-specific keys as described below.

LW Class

The '`LW`' class implements untapered and tapered local Whittle estimation. The class can be initialized with a default taper type (which is `'none'` by default). The `estimate` method accepts the following method-specific parameters (in addition to the common `X`, `m`, and `bounds` arguments):

- `taper`: Taper type from one of the following options:
  - `'none'` (default): The standard (untapered) local Whittle estimator of (Robinson 1995) for stationary and invertible processes with $d_0 \in (-0.5, 0.5)$.
  - `'kolmogorov'`: The Zhurbenko-Kolmogorov taper introduced in Velasco (1999) with order $p = 3$. The weights can be constructed by solving the generating equation stated on page 97 of Velasco (1999). **PyELW** implements the taper following the simple convolution used in Shimotsu's MATLAB implementation.
  - `'cosine'`: The cosine bell taper discussed in Velasco (1999) with $p = 3$ and weights
$$h_t = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi t}{n}\right)\right].$$
  - `'bartlett'`: The triangular Bartlett window taper from Velasco (1999) with $p = 2$. The taper weights are
$$h_t = 1 - \frac{|t - \lceil n/2 \rceil|}{\lceil n/2 \rceil}, \quad t = 1, \dots, n.$$
  - `'hc'`: The complex-valued taper of Hurvich and Chen (2000). This approach differences the data before estimation a number of times specified by the `diff` argument, applies the taper for estimation, then adds back `diff` after estimation to determine $\hat{d}$ As described in Hurvich and Chen (2000), the estimator uses modified frequencies $\tilde{\lambda}_j = 2\pi(j + 0.5)/n$ to reduce bias.
- `diff`: Number of times the series is differenced before estimation (for HC taper only, default: 1)

The return value of `estimate` is a dictionary with the following keys (in addition to the common return values mentioned above):

- `'taper'`: Which taper was used (`'none'` for standard LW)
- `'diff'`: Number of differencing operations performed (for HC taper only)

For the Velasco tapers, the implementation follows the original paper, including the periodogram subsampling factor p and the variance inflation factor Φ for each corresponding value of p.

ELW Class

The `ELW` class implements the exact local Whittle estimator of Shimotsu and Phillips (2005). This estimator extends the range of consistency to all values of the true parameter d_0, provided that the optimization interval has length less than $9/2$, by using fractional differencing to transform the data. This approach eliminates the need for tapering while extending the valid parameter range, however, it does not handle processes with unknown mean or trends. The `estimate` method accepts one additional optional argument to control whether the data are demeaned and if so, which estimator for the mean to use:

- `mean_est`: Type of demeaning, if any, before estimation.

  - 'none' (default): Process the time series as is, without demeaning.
  - 'mean' Process the time series after subtracting the sample mean from each observation.
  - 'init' Process the time series after subtracting the initial value from each observation.

The dictionary returned does not include any additional keys beyond the standard ones described above.

TwoStepELW Class

The TwoStepELW class implements the two-step estimator of Shimotsu (2010), which extends the ELW methodology to handle time series with potentially unknown mean and polynomial trends of known order. The estimator uses a two-step procedure: first applying a tapered local Whittle estimator, then using this initial estimate to construct an adaptive mean-adjusted ELW estimator. The estimate method for this class accepts the following additional arguments:

- taper: Step 1 taper type (default: 'hc')
- detrend_order: Polynomial detrending order (default: 0 for demeaning only)

The result dictionary returned includes the following additional method-specific keys:

- 'taper': Step 1 taper used
- 'detrend_order': Polynomial detrending order
- 'd_step1': Estimated first-step $\hat{d}$ via tapered LW
- 'nfev_step1': Number of objective function evaluations in step 1
- 'objective_step1': Final objective function value in step 1

3.2. Basic initialization and estimation

Here we provide a simple overview of the available estimators. First, with the LW class, there are five tapering options:

```
from pyelw import LW
import numpy as np

x = np.random.randn(500)                      # Sample data

lw = LW()                                     # Standard LW (Robinson, 1995)
result = lw.estimate(x)                       # Defaults
result = lw.estimate(x, m=42)                 # Number of frequencies
result = lw.estimate(x, bounds=(-1.0, 3.0))   # Parameter bounds
```

In addition to the standard (untapered) LW estimator, there are four tapering options:

```
lw_kol  = LW(taper='kolmogorov')     # Kolmogorov taper (Velasco, 1999)
lw_cos  = LW(taper='cosine')         # Cosine bell taper (Velasco, 1999)
lw_bart = LW(taper='bartlett')       # Triangular Bartlett taper (Velasco, 1999)
lw_hc   = LW(taper='hc')             # Complex taper (Hurvich and Chen, 2000)
```

You can also override the taper on a per-call basis:

```
lw = LW(taper='hc')                      # LW instance with HC taper
result = lw.estimate(x)                  # Use the HC taper
result = lw.estimate(x, taper='none')    # Override taper for this call
```

Second, with the `ELW` and `TwoStepELW` classes, no initialization is required, but various estimation parameters are available:

```
# ELW instance and estimation
elw = ELW()
result = elw.estimate(x)                          # Defaults
result = elw.estimate(x, m=42)                    # Number of frequencies
result = elw.estimate(x, bounds=(-1.0, 3.0))      # Parameter bounds

# TwoStepELW instance and estimation
elw2 = TwoStepELW()
result = elw2.estimate(x)                         # Defaults
result = elw2.estimate(x, m=42)                   # Number of frequencies
result = elw2.estimate(x, bounds=(-1.0, 3.0))     # Parameter bounds
result = elw2.estimate(x, taper='kolmogorov')     # First step taper
result = elw2.estimate(x, detrend_order=1)        # Remove linear trend
```

3.3. Auxiliary functions

In addition to the main estimator classes, **PyELW** provides several auxiliary functions for working with fractionally integrated time series. The `fracdiff` function implements the fast $O(n \log n)$ algorithm of Jensen and Nielsen (2014) for applying the fractional differencing operator $(1 - L)^d$ to a time series.

```
>>> from pyelw.fracdiff import fracdiff
>>> x = [1, 1, 1, 1]
>>> fracdiff(x, d=0.4)

array([1.   , 0.6  , 0.48 , 0.416])
```

The `arfima` function generates realizations of ARFIMA$(0, d, 0)$ and ARFIMA$(1, d, 0)$ processes using a two-step algorithm: first generating an AR(1) process with given autocorrelation parameter ϕ and then applying the fractional filter $(1 - L)^(- d)$ via the `fracdiff` function.

```
>>> from pyelw.simulate import arfima
>>> n = 10
>>> x = arfima(n, d=0.4, sigma=1.0, phi=0.1, seed=42)
>>> print(x)
```

```
[-0.46575231 -0.37114045  0.42485813  1.68154856  0.60491577  0.19986061
  1.81105932  1.77565976  0.6346111   1.14893899]
```

3.4. Examples

Example 1: Simulated ARFIMA data

In this first example, we simulate 500 observations from an ARFIMA$(0, d, 0)$ model with true parameter $d_0 = 0.4$ and use the simulated data to estimate d_0 via ELW. We obtain a reasonably close estimate $\hat{d} = 0.3786$ with estimation error $|\hat{d} - d_0| = 0.0214$.

```
>>> from pyelw import ELW
>>> from pyelw.simulate import arfima
>>>
>>> # Simulation parameters
>>> d_true = 0.4      # True memory parameter
>>> n = 500           # Sample size
>>> m = int(n**0.65)  # Number of frequencies
>>>
>>> # Simulate ARFIMA(0,d,0) process
>>> print(f"Simulating ARFIMA(0,{d_true},0) with n={n} observations...")
>>> x = arfima(n, d_true, sigma=1.0, seed=42)
>>>
>>> # Estimate the memory parameter via ELW
>>> elw = ELW()
>>> result = elw.estimate(x, m=m)
>>>
>>> # Display results
>>> print(f"True d:           {d_true}")
>>> print(f"Estimated d:      {result['d_hat']:.4f}")
>>> print(f"Standard error:   {result['se']:.4f}")
>>> print(f"Estimation error: {abs(result['d_hat'] - d_true):.4f}")
```

```
Simulating ARFIMA(0,0.4,0) with n=500 observations...
True d:           0.4
Estimated d:      0.3786
Standard error:   0.0784
Estimation error: 0.0214
```

Example 2: Nile River Level Data

The following example uses **Pandas** to load a CSV dataset containing yearly observations on the minimum level of the Nile river and estimates d via LW and ELW. The `nile.csv` dataset is available in the **PyELW** repository.

```
>>> import pandas as pd
>>> from pyelw import LW, ELW
>>>
>>> # Load time series from 'nile' column of data/nile.csv
>>> df = pd.read_csv('data/nile.csv')
>>> nile = pd.to_numeric(df['nile']).values
>>> print(f"Loaded {len(nile)} observations from nile.csv")
>>>
>>> # Estimate d using local Whittle estimator
>>> lw = LW()
>>> result = lw.estimate(nile)
>>> print(f"LW estimate:  {result['d_hat']:8.3f} ({result['se']:.3f})")
>>>
>>> # Estimate d using exact local Whittle estimator
>>> elw = ELW()
>>> result = elw.estimate(nile)
>>> print(f"ELW estimate: {result['d_hat']:8.3f} ({result['se']:.3f})")
```

```
Loaded 663 observations from nile.csv
LW estimate:     0.409 (0.062)
ELW estimate:    0.886 (0.066)
```

Example 3: U.S. real GDP data from FRED

To illustrate the real-world usage of **PyELW**, below is a complete example in which we analyze U.S. real GDP data obtained from U.S. Federal Reserve Economic Data (FRED) service to estimate the memory parameter d using the two-step ELW estimator. The estimate $\hat{d} = 1.0096$ suggests that the logarithm of real GDP exhibits unit root behavior.

```
>>> import numpy as np
>>> import pandas_datareader as pdr
>>> from pyelw import TwoStepELW
>>>
>>> # Download real GDP from FRED into a Pandas dataframe
>>> df = pdr.get_data_fred('GDPC1', start='1950-01-01', end='2024-12-31')
>>> gdp = df.values.flatten()
>>> n = len(gdp)  # Number of observations
>>> log_gdp = np.log(gdp)  # Natural logarithm
>>> print(f"Downloaded {n} observations for U.S. real GDP")
>>>
>>> # Two-Step ELW estimation with linear detrending
>>> estimator = TwoStepELW()
>>> m = int(n**0.65)  # Choose bandwidth/number of frequencies
>>> result = estimator.estimate(log_gdp, m=m, detrend_order=1)
>>> ci_lower = result['d_hat'] - 1.96 * result['se']
>>> ci_upper = result['d_hat'] + 1.96 * result['se']
>>>
>>> # Display results
>>> print("\nTwo-Step ELW Results:")
>>> print(f"Sample size:           {n}")
>>> print(f"Number of frequencies: {m}")
>>> print(f"Estimated d:           {result['d_hat']:.4f}")
>>> print(f"Standard error:        {result['se']:.4f}")
>>> print(f"95% CI:                [{ci_lower:.4f}, {ci_upper:.4f}]")
```

```
Downloaded 300 observations for U.S. real GDP

Two-Step ELW Results:
Sample size:           300
Number of frequencies: 40
Estimated d:           1.0096
Standard error:        0.0791
95% CI:                [0.8547, 1.1646]
```

4. Validation and replication

We validate **PyELW**'s implementations through comprehensive testing that includes replicating Monte Carlo experiments and empirical results from the literature, implementing cross-platform comparisons with existing R, Stata, and MATLAB implementations surveyed in Table 1.1, and maintaining an extensive **pytest** unit test suite with over 2,400 parametrized test cases. Table 3 provides an overview of the various replications and tests implemented, grouped by estimator.

The source code for each empirical and Monte Carlo replication is included in the `examples` directory of the **PyELW** repository, exhibiting close agreement with original published results. Agreement with certain empirical results from Baum *et al.* (2020) is also enforced in the unit tests in the `tests` directory alongside simulation and cross-platform validation tests to ensure accuracy and robustness.

Simulation-based tests

The simulation-based test suite validates the estimators using ARFIMA processes with known memory parameters across estimator-specific parameter ranges (as detailed in Table 3), with sample sizes ranging from $n = 500$ to $n = 20,000$. The test framework verifies important theoretical properties: consistency at rate $O(m^{-1/2})$ for stationary processes, proper standard error scaling according to $\sqrt{m_2/m_1}$ for bandwidth sequences $m_1 < m_2$, and edge case handling across parameter boundaries including near-unit root processes, antipersistent series with negative d values, and highly nonstationary cases.

Estimator	**Implementation Details**
Local Whittle (LW)	Robinson (1995)
Simulation tests	ARFIMA$(0, d, 0)$ for $d \in [-0.3, 0.4]$
Empirical replications	Shimotsu (2010) Table 8
	Baum *et al.* (2020)
Monte Carlo replications	Shimotsu and Phillips (2005) Table 1
Cross-platform validation	Stata **whittle**, R **LongMemoryTS**
Tapered LW (Velasco)	Velasco (1999)
Simulation tests	ARFIMA$(0, d, 0)$ for $d \in [-0.4, 2.0]$
Monte Carlo replications	Shimotsu and Phillips (2005) Table 2
Cross-platform validation	MATLAB/Octave **veltaper.m**
Tapered LW (HC)	Hurvich and Chen (2000)
Simulation tests	ARFIMA$(0, d, 0)$ for $d \in [-0.4, 2.0]$
Empirical replications	Hurvich and Chen (2000) Table III
Monte Carlo replications	Hurvich and Chen (2000) Table I
	Shimotsu and Phillips (2005) Table 2
Cross-platform validation	R **LongMemoryTS** (corrected version)
Exact LW (ELW)	Shimotsu and Phillips (2005)
Simulation tests	ARFIMA$(0, d, 0)$ for $d \in [-2.5, 4.0]$
Empirical replications	Baum *et al.* (2020)
Monte Carlo replications	Shimotsu and Phillips (2005) Table 1
Cross-platform validation	Stata **whittle**, R **LongMemoryTS**
Two-Step ELW (2ELW)	Shimotsu (2010)
Simulation tests	ARFIMA$(0, d, 0)$ for $d \in [-2.5, 3.0]$
Empirical replications	Shimotsu (2010) Table 8
Monte Carlo replications	Shimotsu (2010) Table 2
Cross-platform validation	R **LongMemoryTS**

Table 3: Summary of replications and tests performed by estimator. Notes: Replication scripts are provided in the `examples` directory of the **PyELW** repository. Simulation tests and cross-platform validations are performed in the included **pytest** unit tests. in the `tests` directory.

Empirical validation tests

The empirical validation tests in the suite employ two well-known time series datasets used in Baum *et al.* (2020) to ensure estimator accuracy:

1. `nile.csv`: Annual minimum water levels of the Nile river (622–1284 CE, $n = 663$).

2. `sealevel.csv`: Monthly global sea level measurements (1880–1985, $n = 1,272$).

For each dataset, the test suite validates estimates across multiple bandwidth selections ($m = n^\alpha$ for various values of α) and compares results against the published results of Baum *et al.* (2020) using Stata's **whittle** package.

Cross-platform validation tests

Perhaps the most comprehensive validation in the **PyELW** test suite involves systematic comparison against implementations in multiple statistical computing platforms to verify numerical agreement and reproducibility of point estimates and standard errors.

- R **LongMemoryTS**: We validate against the `local.W` function for standard and tapered LW estimation, the `ELW` function for ELW estimation, and the `ELW2S` function for two-step ELW estimation with polynomial detrending.

- Stata **whittle**: The test suite compares results with Baum *et al.* (2020)'s implementation for both the standard LW and ELW estimators using the `nile.csv` and `sealevel.csv` datasets.

- MATLAB **elwcode10**: We validate tapered estimation against Shimotsu's `veltaper.m` implementation of the Velasco (1999) Zhurbenko-Kolmogorov taper with $p = 3$ polynomial order, ensuring numerical equivalence for this specialized estimation approach.

Each test employs JSON-serialized reference results for automated regression testing, with tolerance thresholds reflecting comparison characteristics. Cross-platform comparisons accommodate numerical and implementation differences with point estimates agreeing within 10^{-9} to 10^{-2} absolute tolerance and standard errors within 10^{-8} to 10^{-2} absolute tolerance. Specific tolerance values for individual test cases can be found in the test suite files `tests/test_lw.py`, `tests/test_elw.py`, and `tests/test_twostep.py`.

Other Tests

Beyond the core validation tests, the unit test suite also includes tests for the included auxiliary methods.

In the `test_fracdiff.py` module, our implementation of the Jensen and Nielsen (2014) fast fractional differencing algorithm undergoes verification against known coefficient recursions, comparison with R's **LongMemoryTS** implementation, cross-validation with a reference based on direct convolution, and inversion property testing across various parameter ranges and edge cases, including boundary conditions for integer differencing ($d \in \mathbb{Z}$) and unit impulse coefficient validation.

The `test_simulate.py` module contains comprehensive tests of the ARFIMA simulation function, validating theoretical properties including variance scaling with σ^2, spectral density behavior with slopes $-2d$ at zero frequency, autocorrelation structure for both stationary ($d < 0.5$) and nonstationary ($d \geq 0.5$) processes, variance growth properties for unit root and explosive cases, AR(1) initialization correctness, asymptotic autocorrelation decay patterns, stationarity boundary conditions, and antipersistence properties for negative memory parameters. The tests cover extreme parameter values ($d \in [-0.49, 2.0]$) and verify boundary conditions such as white noise recovery when $d = 0$.

In the `test_optimization.py` module, the golden section search optimization routine is tested for convergence on various function types including quadratic, quartic, absolute value, and exponential functions, with validation of custom tolerances, iteration limits, boundary conditions, and Local Whittle-like objective functions.

5. Summary

PyELW provides a comprehensive and thoroughly validated implementation of local Whittle and exact local Whittle estimation methods for long memory time series analysis in Python. The package implements the original local Whittle estimator of Robinson (1995), the tapered variants of Velasco (1999) and Hurvich and Chen (2000), and the exact local Whittle methods of Shimotsu and Phillips (2005) and Shimotsu (2010). Validation includes accurate replications of major published results, cross-platform verification against R, Stata, and MATLAB implementations, and comprehensive Monte Carlo testing across a wide range of memory parameter values. The object-oriented design promotes extensibility while providing a consistent interface, and the package includes several practical examples with real data as well as a comprehensive **pytest** test suite.

Computational details

The results in this paper were obtained using Python 3.13 with **NumPy** 2.3.2. The **PyELW** package is available from the Python Package Index (PyPI) at `https://pypi.org/` and the source code is hosted on GitHub at `https://github.com/jrblevin/pyelw`.

The estimators involve optimization of scalar objective functions. We could use `minimize_scalar` from **scipy.optimize**, however, to keep dependencies to a minimum (primarily **NumPy**), we implemented a golden section search method internally.

To create test cases for the LW, tapered LW, ELW and two-step ELW estimators, we used an archived version of the **LongMemoryTS** package (version 0.1.0) for R (version 4.5.1) as well as the **whittle** package (version 1.0.4) for Stata (version 16.1).

Acknowledgments

The author is grateful to the developers of the original implementations that served as benchmarks for this work, particularly Katsumi Shimotsu for sharing the MATLAB implementations of several estimators, Christian Leschinski, Michelle Voges, and Kai Wenger, the authors of the R **LongMemoryTS** package, and Christopher Baum, Stan Hurn, and Kenneth Lindsay,

authors of the Stata **whittle** package.

References

Andersen TG, Bollerslev T, Diebold FX, Labys P (2001). "The Distribution of Realized Exchange Rate Volatility." *Journal of the American Statistical Association*, **96**(453), 42–55.

Baillie RT, Chung SK (2002). "Modeling and forecasting from trend-stationary long memory models with applications to climatology." *International Journal of Forecasting*, **18**(2), 215–226.

Baum CF, Hurn S, Lindsay K (2020). "Local Whittle estimation of the long-memory parameter." *Stata Journal*, **20**(3), 565–583.

Geweke J, Porter-Hudak S (1983). "The Estimation and Application of Long Memory Time Series Models." *Journal of Time Series Analysis*, **4**(4), 221–238.

Granger CW, Joyeux R (1980). "An Introduction to Long-Memory Time Series Models and Fractional Differencing." *Journal of Time Series Analysis*, **1**(1), 15–29. doi:10.1111/j.1467-9892.1980.tb00297.x.

Hassler U, Wolters J (1995). "Long Memory in Inflation Rates: International Evidence." *Journal of Business & Economic Statistics*, **13**(1), 37–45.

Henry M, Robinson PM (1996). "Bandwidth Choice in Gaussian Semiparametric Estimation of Long Range Dependence." In *Athens Conference on Applied Probability and Time Series*, volume 115 of *Lecture Notes in Statistics*, pp. 220–232. Springer, New York.

Hosking JRM (1981). "Fractional Differencing." *Biometrika*, **68**(1), 165–176.

Hurvich CM, Chen WW (2000). "An Efficient Taper for Potentially Overdifferenced Long-Memory Time Series." *Journal of Time Series Analysis*, **21**(2), 155–180. doi:10.1111/1467-9892.00179.

Jensen AN, Nielsen MO (2014). "A Fast Fractional Difference Algorithm." *Journal of Time Series Analysis*, **35**(5), 428–436. doi:10.1111/jtsa.12074.

Robinson PM (1995). "Gaussian Semiparametric Estimation of Long Range Dependence." *Annals of Statistics*, **23**(5), 1630–1661.

Shimotsu K (2010). "Exact Local Whittle Estimation of Fractional Integration with Unknown Mean and Time Trend." *Econometric Theory*, **26**(2), 501–540.

Shimotsu K, Phillips PCB (2005). "Exact local Whittle estimation of fractional integration." *Annals of Statistics*, **33**, 1890–1933.

Velasco C (1999). "Gaussian Semiparametric Estimation for Non-Stationary Time Series." *Journal of Time Series Analysis*, **20**(1), 87–126.

Whittle P (1951). *Hypothesis Testing in Time Series Analysis*. Uppsala: Almqvist & Wiksells.

A. Corrections to R LongMemoryTS package

During cross-platform validation against the **LongMemoryTS** package for R, we identified several implementation errors in the original code that prevented replication of published results. These errors affected both the Hurvich and Chen (2000) tapered LW estimator and the two-step ELW estimator of Shimotsu (2010). Because the **LongMemoryTS** package has been removed from CRAN and is no longer in active development, we are unable to have our changes incorporated into the package. Instead, we document these findings here and provide corrected implementations of the specific affected functions in the **PyELW** repository.

First, the original **LongMemoryTS** implementation contained three errors in the HC complex taper specification:

1. The complex-valued taper function was incorrectly implemented using `cos` instead of the exponential function specified in Hurvich and Chen (2000):

   ```
   # Original
   cos_bell_cmplx <- function(u) { 1/2 * (1 - cos(1i * 2 * pi * u)) }

   # Corrected
   cos_bell_cmplx <- function(u) { 1/2 * (1 - exp(1i * 2 * pi * u)) }
   ```

2. The original implementation used the standard periodogram for all tapers, but Hurvich and Chen (2000) used a specialized tapered periodogram. We added the missing `hc_per` function:

   ```
   hc_per <- function(data, m) {
     T <- length(data)
     norm_factor <- sqrt(2 * pi * T) * sqrt(2)
     lambda <- 2 * pi * seq_len(m) %o% seq_len(T) / T
     W <- rowSums(exp(1i * lambda) * rep(data, each = m)) / norm_factor
     return(abs(W)^2)
   }
   ```

3. The taper was applied incorrectly and the differencing parameter was hard-coded rather than using the actual value:

   ```
   # Original
   ht <- cos_bell_cmplx((1:T)/T)
   peri <- per(data)[-1]
   d.hat <- optimize(...)$minimum + 1  # Always adds 1

   # Corrected
   ht <- cos_bell_cmplx((1:T-0.5)/T)  # Proper t-0.5 shift
   peri <- hc_per(data, m)            # Use HC periodogram
   d.hat <- optimize(...)$minimum + diff_param  # Use actual parameter
   ```

Second, the two-step estimator implementation contained fundamental errors in the weight function and data handling:

1. The adaptive weight function `wd.elw` did not implement the piecewise specification from Shimotsu (2010):

   ```
   # Original
   wd.elw <- function(d) { 1/2 * (1 + cos(4 * pi * d)) }

   # Corrected
   wd.elw <- function(d) {
     if(d <= 0.5) {
       return(1.0)                          # Sample mean for stationary
     } else if(d < 0.75) {
       return(0.5 * (1 + cos(4 * pi * d)))  # Smooth transition
     } else {
       return(0.0)                          # First observation for persistent
     }
   }
   ```

2. The most critical issue was in the `ELW2S` function, which computed detrended data `Xt` but did not use it in either estimation step:

   ```
   # Original
   Xt <- residuals(lm(data ~ poly(1:length(data), trend_order)))
   aux_est <- local.W(data=data, m=m, taper=taper, ...)  # Uses raw data
   d.hat <- optim(..., data=data, ...)                   # Uses raw data

   # Corrected
   Xt <- residuals(lm(data ~ poly(1:length(data), trend_order)))
   aux_est <- local.W(data=Xt, m=m, taper=taper, ...)   # Uses Xt
   d.hat <- optim(..., data=Xt, ...)                    # Uses Xt
   ```

3. Finally, a more minor change, the original weighted objective function removed the first observation, but we now keep all observations.

   ```
   # Original
   data <- (data - wd.elw(d)*mean(data) - (1-wd.elw(d))*data[1])[-1]

   # Corrected
   weight <- wd.elw(d)
   myu <- weight * mean(data) + (1 - weight) * data[1]
   data_corrected <- data - myu  # No observation removal
   ```

These corrections proved to be necessary for accurate implementation of the HC tapered LW and two-step ELW estimators. We verified our corrections by replicating the Monte Carlo experiments in Hurvich and Chen (2000) Table 1 and Shimotsu (2010) Table 8. The corrected implementations agree closely with the published results from both papers and our **PyELW** implementation, while the original implementations yield biased results. Our replication code along with our corrected `R` implementations is available in the `R/` directory of the **PyELW** repository.

Affiliation:

Jason R. Blevins
Department of Economics
The Ohio State University
1945 N. High Street
Columbus, OH 43210
E-mail: blevins.141@osu.edu
URL: https://jblevins.org/