tempdisagg: A Modular Python Framework for Temporal Disaggregation of Time Series

Jaime Vera-Jaramillo*

*E-mail: jaimevera1107@gmail.com

Abstract

This paper introduces `tempdisagg`, a Python framework designed for temporal disaggregation of time series data, systematically converting low-frequency observations into consistent and statistically robust high-frequency estimates. The package implements classical and advanced methods such as Chow-Lin, Denton, Fernández, and Litterman, enhanced by automatic optimization of critical parameters and robust ensemble modeling techniques. It features rigorous validation, dynamic construction of aggregation matrices, and meticulous post-estimation adjustments ensuring non-negativity and aggregation consistency. `tempdisagg` offers a modular, production-ready solution suitable for economics, statistical research, forecasting applications, and integration into reproducible data science workflows.

Keywords: temporal disaggregation; econometric modeling; statistical interpolation; open-source software; ensemble learning; data harmonization; frequency conversion; quantitative methods

1 Introduction

The `tempdisagg` Python package provides a robust, modular, and production-ready framework designed for temporal disaggregation of time series data, effectively bridging the gap between low-frequency official statistics and high-frequency decision-making needs. Developed independently with inspiration from the established R package Sax and Steiner (2013), this Python implementation not only replicates classical econometric and statistical disaggregation methods but significantly extends their capabilities. The library implements a variety of well-established approaches, including regression-based models such as Chow-Lin Chow and Lin (1971) and Litterman Litterman (1983), smoothing and differencing techniques like Denton Denton (1971) and Fernández Fernández (1981), and even simple uniform distributions. Moreover, it introduces optimized variants of these methods by automatically estimating critical parameters, notably the autocorrelation coefficient (ρ), either by the maximum log-likelihood or by minimizing the residual sum of squares, improving the accuracy and reliability of the estimation Quilis (2018); Sax and Steiner (2013).

Temporal disaggregation techniques are crucial for converting low-frequency economic and statistical data into higher-frequency estimates. High-frequency data provide timely insights, essential for economic analysis, policy-making, and business decisions. The `tempdisagg` package fills a notable gap by providing these methods within a robust Python implementation, inspired by and extending the original R package by Sax and Steiner Sax and Steiner (2013). The software can be used by economists, data scientists, statisticians, and public institutions to disaggregate low-frequency data into high-frequency estimates. It is especially relevant for national statistics, macroeconomic forecasting, and business analytics. It can also be extended for teaching, validation, or integration with machine learning pipelines.

A standout feature of the library is its ensemble modeling capability, which allows practitioners to combine multiple disaggregation methods into a single, robust prediction. This is achieved through a nonnegative least squares optimization process, where individual model predictions are weighted to minimize aggregated prediction errors under strict aggregation constraints. Bootstrap-based confidence intervals and comprehensive performance metrics (e.g. RMSE, MAE, MAPE, and R^2) further enhance interpretability and trust in the results. In terms of architecture, the `tempdisagg` library follows a highly modular and extensible design. It incorporates rigorous validation at every stage, from input preprocessing and robust interpolation of missing values to the dynamic construction of aggregation matrices and meticulous post-estimation adjustments. These post-estimation steps explicitly prevent negative predicted values while ensuring strict consistency with low-frequency aggregates, a critical requirement in practical applications.

The software adheres to a clear and intuitive API structure inspired by `Scikit-learn`, featuring standardized methods such as `fit()`, `predict()`, `summary()`, and `plot()`. This design choice simplifies integration into existing data analysis and machine learning pipelines, making it accessible for researchers, economists, statisticians, and data scientists alike. Thanks to its flexibility and rigorous documentation, `tempdisagg` is highly suitable for diverse applications, ranging from economic analysis and macroeconomic forecasting to business intelligence and public statistics production. Furthermore, the detailed online documentation, accessible via GitHub and PyPI, includes illustrative examples and Jupyter notebooks that facilitate reproducibility and ease of adoption.

In summary, `tempdisagg` offers an advanced, open-source solution for temporal disaggregation, pushing beyond traditional methods to embrace ensemble modeling, extensive validation, and seamless integration into modern data science workflows. Its open-source MIT license encourages wide reuse and collaborative advancement by the global scientific and economic community.

2 Methodology

The `tempdisagg` framework provides a structured, modular methodology for temporal disaggregation. This methodology systematically transforms low-frequency time series into high-frequency estimates, maintaining consistency and statistical coherence through a series of clearly defined steps:

2.1 Validation of Input Data Structure

The framework initially performs a rigorous validation of the input data. It ensures the dataset contains all necessary columns: `Index` for low-frequency groupings (such as year or quarter), `Grain` for sub-period identifiers (such as months within a year), `y` as the target variable at low frequency, and `X` as the high-frequency indicator variable. Any inconsistencies or structural issues are flagged at this stage to avoid downstream errors. To effectively use the `TempDisModel`, the input time series must be organized in a long-format DataFrame with one row per high-frequency observation. The DataFrame should explicitly contain the following columns:

Table 1: Required Input Columns for `TempDisModel`

Column	Description
`Index`	Identifier for the low-frequency group (e.g., year or quarter). This column groups the observations to be disaggregated.
`Grain`	Identifier for high-frequency subdivisions within each `Index` (e.g., month within a year, quarter number within a year).
`y`	The low-frequency target variable. This variable is constant across all high-frequency observations within each low-frequency group and represents the aggregate value to be disaggregated.
`X`	The high-frequency indicator variable. This variable should be available for each high-frequency sub-period and is used to guide the temporal disaggregation.

Table 2 illustrates an example of how the DataFrame should be structured for use in `tempdisagg`.

Table 2: Example Input DataFrame Structure

Index	Grain	y	X
2000	1	1000.00	80.21
2000	2	1000.00	91.13
2000	3	1000.00	85.44
2000	4	1000.00	92.32
2001	1	1200.00	88.71
2001	2	1200.00	93.55
...	...	...	...

2.2 Completion of Missing Sub-Periods Using Robust Interpolation

Once validated, the dataset undergoes a robust interpolation procedure to fill in any missing sub-periods. This interpolation employs reliable numerical methods such as linear, nearest-neighbor, or spline interpolation, ensuring that all combinations of `Index` and `Grain` are represented. This step guarantees a continuous and consistent time series necessary for subsequent modeling.

2.3 Construction of Aggregation Matrix

Next, an aggregation matrix C is constructed dynamically according to the specified aggregation rule. Supported rules include *sum*, *average*, *first*, and *last*. The aggregation matrix precisely maps the relationship between high-frequency observations and their corresponding low-frequency aggregates, ensuring internal coherence in the modeling process.

2.4 Construction of Aggregation Matrix

The aggregation matrix C is constructed dynamically based on the specified aggregation rule, mapping high-frequency data to low-frequency aggregates. For each low-frequency group, the corresponding rows of C are defined as follows, where y_l are low-frequency aggregates and y are the corresponding high-frequency observations:

- *Sum*: Each high-frequency observation within a group has a weight of 1.
- *Average*: Each observation within a group has equal weight $1/m$, where m is the number of high-frequency periods per low-frequency group.
- *First*: Only the first high-frequency observation in each group has a weight of 1; all others have a weight of 0.
- *Last*: Only the last high-frequency observation in each group has a weight of 1; all others have a weight of 0.

2.5 Application of Classical and Advanced Disaggregation Models

The `tempdisagg` framework implements a variety of classical and advanced temporal disaggregation methods. Each of these methods currently uses only one high-frequency indicator, meaning the framework is strictly univariate at this stage and does not yet support multivariate disaggregation. The available methods, including Chow-Lin, Denton, Fernández, and Litterman, provide distinct statistical approaches such as regression-based autoregressive adjustments, smoothing techniques, and differencing methods. The choice among these methods should be guided by the specific statistical properties of the dataset and the analytical objectives at hand. Table 3 presents a concise overview of these implemented disaggregation methods and their key characteristics, highlighting their suitability to different data contexts.

2.6 Optional Optimization of ρ and Ensemble Modeling

An advanced feature of `tempdisagg` is the optional optimization of the autocorrelation parameter ρ, which enhances the accuracy of regression-based models. Optimization methods include maximum log-likelihood and residual sum of squares minimization. Furthermore, an ensemble modeling feature enables the combination of multiple individual disaggregation models, employing a non-negative least squares optimization to find optimal weights that minimize aggregate prediction errors. The ensemble modeling feature enables the combination of multiple individual disaggregation models. Mathematically, the optimal weights w_i for combining M individual model predictions $\hat{y}_i$ are determined by solving the following non-negative least squares optimization problem:

Table 3: Methods Implemented in the `tempdisagg` Package

Method(s)	**Description**
`ols`	Ordinary Least Squares regression; serves as the baseline method.
`denton`, `denton-opt`	Denton's interpolation, employing optional higher-order differencing to smooth estimates and maintain proportionality to indicator series Denton (1971).
`denton-cholette`	Denton-Cholette modified smoother proposed by Dagum and Cholette, designed to minimize discrepancies and preserve trend smoothness Dagum and Cholette (2006).
`chow-lin`, `chow-lin-opt`, `chow-lin-ecotrim`, `chow-lin-quilis`	Chow-Lin family methods, regression-based approaches incorporating autoregressive adjustments, including optimized and adapted variants such as Ecotrim and Quilis Chow and Lin (1971); Quilis (2018).
`litterman`, `litterman-opt`	Litterman's method applying a Bayesian random walk or AR(1) prior for smoothing, including optimized variants for autocorrelation estimation Litterman (1983).
`fernandez`	Fernández method; employs second-order differencing, equivalent to a special case of the Litterman method with autocorrelation parameter $\rho = 0$ Fernández (1981).
`fast`	Fast approximation of Denton-Cholette method, optimized for computational efficiency without significantly compromising accuracy Dagum and Cholette (2006).
`uniform`	Uniform distribution method evenly distributing low-frequency aggregates across high-frequency sub-periods, serving as a non-informative benchmark.

$$\min_{w \geq 0} \left\| C \left(\sum_{i=1}^{M} w_i \hat{y}_i \right) - y_l \right\|^2 \quad \text{subject to} \quad \sum_{i=1}^{M} w_i = 1$$

Here, C is the aggregation matrix, y_l represents the observed low-frequency aggregates, and $\hat{y}_i$ denotes the high-frequency predictions from each individual model. The resulting optimal weights ensure minimal discrepancy between aggregated ensemble predictions and the actual low-frequency data.

2.7 Post-Estimation Adjustments for Non-negativity and Consistency

Finally, the estimated high-frequency series undergo post-estimation adjustments designed to eliminate negative values and strictly enforce consistency with the original low-frequency aggregates. This adjustment step redistributes negative values within each low-frequency period, ensuring realistic and statistically sound high-frequency predictions suitable for practical applications.

Negative predictions in the high-frequency series $\hat{y}$ are corrected by solving an optimization problem that minimizes the squared deviation from original predictions while enforcing non-negativity. Mathematically, this can be expressed as:

$$\min_{y \geq 0} \sum_{i} (y_i - \hat{y}_i)^2 \quad \text{subject to} \quad \sum_{i} y_i = \sum_{i} \hat{y}_i$$

Where y_i are the adjusted non-negative predictions and $\hat{y}_i$ are the original predictions. This constraint ensures that the aggregation matrix C accurately reconstructs the original low-frequency observations from the adjusted high-frequency estimates. The final adjustment simultaneously satisfies both constraints by redistributing any residual discrepancy proportionally or uniformly across the high-frequency predictions within each low-frequency group.

3 Quality Control

The software implements rigorous quality control checks at each analytical stage. Examples include matrix dimensionality validation, fallback mechanisms for interpolation, and detailed logging of the number of missing periods imputed. Additional validations verify aggregation consistency by ensuring that aggregated high-frequency estimates match exactly the original low-frequency values provided.

4 Illustrative Examples

To demonstrate the functionality of the `tempdisagg` package in practice, we present two real-world disaggregation examples using U.S. macroeconomic data. Both examples use annual GDP as the low-frequency target variable and quarterly real consumption as the high-frequency indicator. These examples are designed to showcase the flexibility and robustness of the implemented methods.

The first example performs disaggregation using the optimized Chow-Lin method. The results are visualized in Figure **??**, where the original annual GDP values (red diamonds), the disaggregated

quarterly estimates (blue line), and the indicator series (gray dashed line) are displayed. The Python implementation for this example is shown in Listing 1.

The second example leverages the ensemble modeling capability of the library, combining the predictions of three methods: Chow-Lin, Litterman, and Fernández. In addition, the output is post-processed to enforce non-negativity and exact aggregation consistency. The resulting prediction is shown in Figure **??**. This example illustrates how ensemble learning can improve prediction robustness while ensuring practical constraints are respected.

All code examples, Jupyter notebooks, and visual outputs are available in the official GitHub repository: https://github.com/jaimevera1107/tempdisagg.

4.1 Example 1:

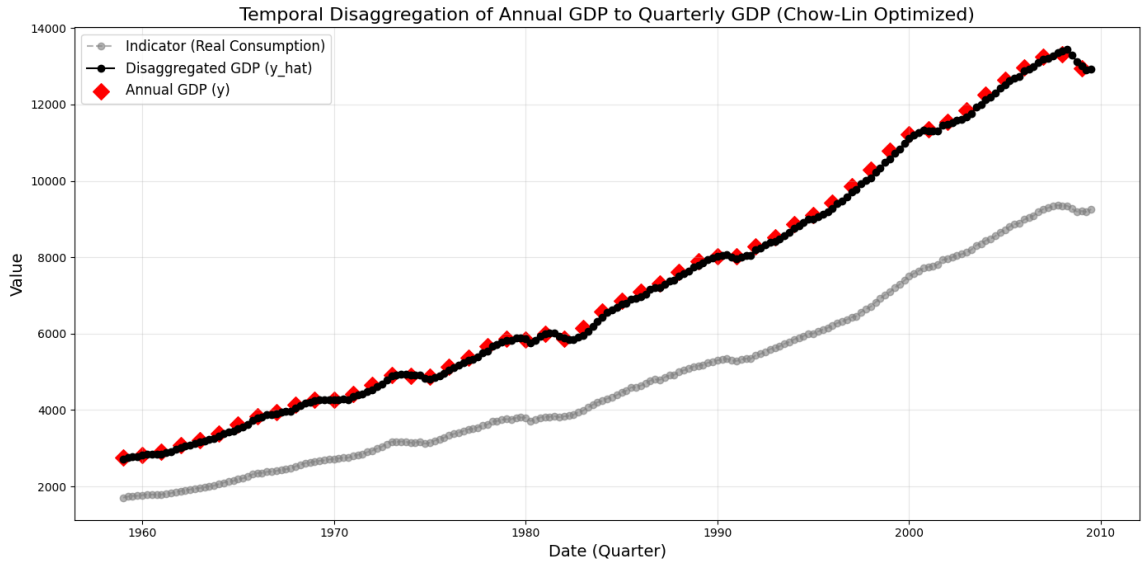

Figure 1: Temporal disaggregation of annual GDP into quarterly estimates using Chow-Lin optimized method. The red diamonds represent the original annual values, the blue line shows the disaggregated estimates, and the gray dashed line represents the high-frequency indicator (real consumption).

```
# First, install dependencies if needed:
# pip install tempdisagg statsmodels matplotlib
import pandas as pd
import matplotlib.pyplot as plt
from tempdisagg import TempDisaggModel
import statsmodels.api as sm

data = sm.datasets.macrodata.load_pandas().data

data['year'] = data['year'].astype(int)
data['quarter'] = data['quarter'].astype(int)
```

```
data['realgdp'] = data['realgdp']
data['realcons'] = data['realcons']
quarters = pd.period_range(start=f"{data['year'].min()}Q{data['quarter'].min()
    }", periods=len(data), freq='Q')
data['date'] = quarters.to_timestamp()

data_annual = data.groupby('year')['realgdp'].mean().reset_index()
data_annual.columns = ['Index', 'y']
df = data.merge(data_annual, left_on='year', right_on='Index', how='left')
df.rename(columns={'realcons': 'X', 'quarter': 'Grain'}, inplace=True)
df = df[['date', 'Index', 'Grain', 'y', 'X']]

model = TempDisaggModel(method="chow-lin-opt", conversion="average")
model.fit(df.rename(columns={'date': 'Time'}))
df['y_hat'] = model.predict()

plt.figure(figsize=(14, 7))
plt.plot(df['date'], df['X'], linestyle='--', marker='o', color='gray', alpha
    =0.6)
plt.plot(df['date'], df['y_hat'], linestyle='-', marker='o', color='blue',
    linewidth=2.5)
plt.scatter(df.drop_duplicates('Index')['date'], df.drop_duplicates('Index')['
    y'], color='red', marker='D', s=100)
plt.title('Temporal Disaggregation of Annual GDP to Quarterly GDP')
plt.xlabel('Date (Quarter)')
plt.ylabel('Value')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.savefig("example.png", dpi=300, bbox_inches='tight')
plt.close()
```

Listing 1: Disaggregation using real GDP and consumption data

4.2 Example 2:

```
# First, install dependencies if needed:
# pip install tempdisagg statsmodels matplotlib
import pandas as pd
import matplotlib.pyplot as plt
from tempdisagg import TempDisaggModel
import statsmodels.api as sm
```

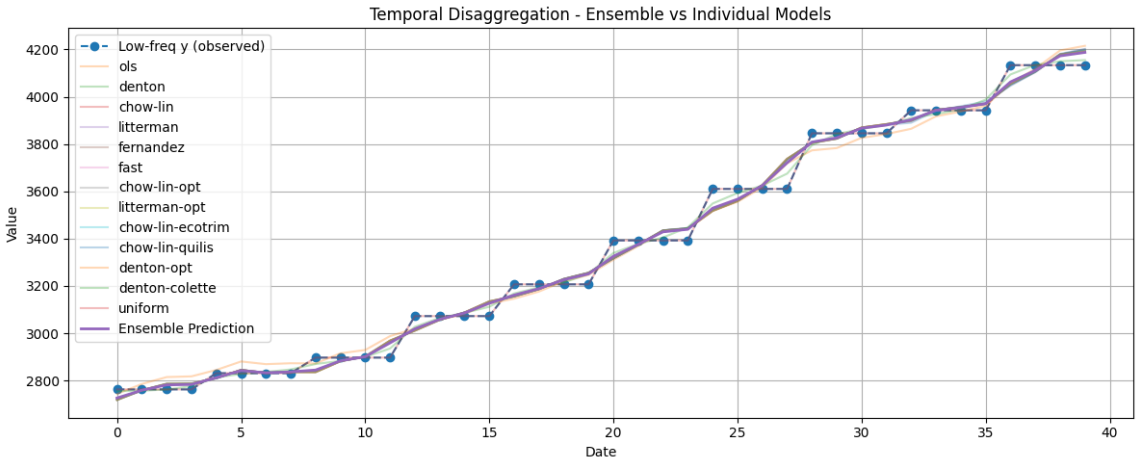

Figure 2: Temporal disaggregation of annual GDP into quarterly estimates using Chow-Lin optimized method. The red diamonds represent the original annual values, the blue line shows the disaggregated estimates, and the gray dashed line represents the high-frequency indicator (real consumption).

```
data = sm.datasets.macrodata.load_pandas().data

data['year'] = data['year'].astype(int)
data['quarter'] = data['quarter'].astype(int)
data['realgdp'] = data['realgdp']
data['realcons'] = data['realcons']

quarters = pd.period_range(start=f"{data['year'].min()}Q{data['quarter'].min()
    }", periods=len(data), freq='Q')
data['date'] = quarters.to_timestamp()

data_annual = data.groupby('year')['realgdp'].mean().reset_index()
data_annual.columns = ['Index', 'y']

df = data.merge(data_annual, left_on='year', right_on='Index', how='left')
df.rename(columns={'realcons': 'X', 'quarter': 'Grain'}, inplace=True)
df = df[['Index', 'Grain', 'y', 'X']]
df = df.head(40)

model = TempDisaggModel(conversion="average")
model.ensemble(df)
model.plot(df)
```

5 Limitations and Future Work

Currently, `tempdisagg` supports only univariate temporal disaggregation methods, relying on a single high-frequency indicator series. Multivariate disaggregation and incorporation of multiple indicators simultaneously are planned as future enhancements. Further developments will also include Bayesian and state-space approaches, expanding methodological flexibility and robustness.

6 Project Description

- **Repository location:** The code and documentation are hosted on GitHub: https://github.com/jaimevera1107/tempdisagg. The PyPI package is published at: https://pypi.org/project/tempdisagg/
- **Repository name:** `tempdisagg`
- **Language:** Python, English
- **License:** MIT
- **Publication date:** 2025-03-23

7 Conclusions

The `tempdisagg` package provides a comprehensive and versatile solution for temporal disaggregation in Python, delivering an accessible, production-ready toolkit to researchers, statisticians, economists, and data analysts. Its modular design, rigorous validation framework, and user-friendly API ensure high usability, transparency, and reproducibility, effectively bridging the gap between traditional econometric practices and modern data science workflows.

Beyond replicating classical methods like Chow-Lin, Denton, Fernández, and Litterman, the package introduces ensemble modeling capabilities and post-estimation adjustments, allowing users to improve robustness and respect real-world constraints such as non-negativity and aggregation consistency. These features make `tempdisagg` particularly valuable for applications in national statistics, policy evaluation, and economic forecasting.

By combining theoretical rigor with software engineering best practices, `tempdisagg` serves as both a practical tool and an extensible research platform. Future developments may include support for multivariate disaggregation, integration with state-space models, and Bayesian estimation approaches, further extending its applicability in time series analysis.

Acknowledgments

Thanks to the R package `tempdisagg` for their methodological foundation. The appreciation also goes to the open-source community for tools such as `NumPy`, `SciPy`, and Pandas.

Funding Statement

This project did not receive external funding. The author has no competing interests to declare.

References

Chow, G. C., & Lin, A.-l. (1971). Best linear unbiased interpolation, distribution, and extrapolation of time series by related series. *The Review of Economics and Statistics*, *53*(4), 372–375.

Dagum, E. B., & Cholette, P. A. (2006). *Benchmarking, temporal distribution, and reconciliation methods for time series* (Tech. Rep.). Statistics Canada.

Denton, F. T. (1971). Adjustment of monthly or quarterly series to annual totals: An approach based on quadratic minimization. *Journal of the American Statistical Association*, *66*(333), 99–102.

Fernández, R. B. (1981). A methodological note on the estimation of time series. *The Review of Economics and Statistics*, *63*(3), 471–476.

Litterman, R. B. (1983). A random walk, markov model for the distribution of time series. *Journal of Business & Economic Statistics*, *1*(2), 169–173.

Quilis, E. M. (2018). A generalized version of the chow–lin procedure for temporal disaggregation. *Journal of Official Statistics*, *34*(2), 523–542.

Sax, C., & Steiner, P. (2013). tempdisagg: Methods for temporal disaggregation and interpolation of time series [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=tempdisagg (R package version 1.0)