INTERNATIONAL MONETARY FUND

A Python Package to Assist Macroframework Forecasting

Concepts and Examples

Sakai Ando, Shuvam Das, Sultan Orazbayev

WP/25/172

2025
AUG

WORKING PAPER

WP/25/172

IMF Working Paper
Research Department

A Python Package to Assist Macroframework Forecasting: Concepts and Examples

Prepared by Sakai Ando, Shuvam Das, and Sultan Orazbayev *

Authorized for distribution by Emine Boz
August 2025

ABSTRACT: In forecasting economic time series, statistical models often need to be complemented with a process to impose various constraints in a smooth manner. Systematically imposing constraints and retaining smoothness are important but challenging. Ando (2024) proposes a systematic approach, but a user-friendly package to implement it has not been developed. This paper addresses this gap by introducing a Python package, `macroframe-forecast`, that allows users to generate forecasts that are both smooth over time and consistent with user-specified constraints. We demonstrate the package's functionality with two examples about forecasting US GDP and fiscal variables.

RECOMMENDED CITATION: Ando, Sakai, Shuvam Das, and Sultan Orazbayev (2025), "A Python Package to Assist Macroframework Forecasting: Concepts and Examples," IMF Working Paper.

JEL Classification Numbers:	C53, E17
Keywords:	Forecast Reconciliation; Python Package; Macroframework
Author's E-Mail Address:	sando@imf.org; sdas7@imf.org; sorazbayev@imf.org

* We thank Mingmei Xiao, Yuki Sato, and Doga Bilgin for their earlier contributions.

WORKING PAPERS

A Python Package to Assist Macroframework Forecasting

Concepts and Examples

Prepared by Sakai Ando, Shuvam Das, Sultan Orazbayev

1. Introduction

In forecasting economic time series, statistical models often need to be supplemented with procedures that impose constraints while preserving smoothness over time. For example, GDP forecasts generated using models such as autoregressions or decision trees may not align with the long-term growth rates anticipated by forecasters. In such cases, forecasters aim to adjust the time series so that it converges smoothly to the desired long-term growth path. However, ad hoc constraint imposition, such as manually altering only the terminal value in a long time series, can introduce undesirable discontinuities between the penultimate and terminal values. Similar challenges arise when forecasting aggregate variables and their subcomponents, such as fiscal balance, revenue, and expenditure. Relying solely on statistical models may fail to ensure that forecasts satisfy accounting identity constraints, and imposing these constraints in an ad hoc manner, such as treating one variable as a residual, can result in forecasts that lack the desired smoothness since the residual variable absorbs the forecast errors of the rest. In general, adjusting the forecasts to satisfy constraints often breaks the smoothness, and vice versa.

Systematically imposing constraints while retaining smoothness is important but challenging. Constraints often stem from accounting identities and expert judgment, making their incorporation essential for internal consistency. Smoothness is equally critical, as optimal forecasts typically exhibit less volatility than historical data. For instance, in a random walk, historical data are volatile, but the optimal forecast is constant over time, equal to the last observed value. Achieving both objectives manually is resource-intensive, especially when dealing with numerous variables and constraints, raising the question of how to systematically impose constraints and smoothness.

Ando (2024) proposes a systematic approach to impose constraints and maintain smoothness, but a user-friendly package to implement it has not been developed. Building on the forecast reconciliation literature, notably reviewed by Athanasopoulos et al. (2024) and the smoothing method of Hodrick and Prescott (1997), Ando (2024) defines a quadratic programming problem that can impose both the constraints and temporal smoothness in a close form, applicable to a large system of time series. Ando (2024) then provides three examples to illustrate how to combine statistical models with the proposed smooth reconciliation method, a la Ando and Kim (2023). Although Ando (2024) provides the replication code for the examples used in the paper, a user-friendly package to implement the method remains a gap.

Existing packages in R and Python assist forecast reconciliation and smoothing separately but not jointly. For instance, the `hts` (Hyndman et al., 2021) and `FoReco` (Girolimetto and Di Fonzo, 2023) packages in R support reconciliation, but the reconciled forecast may not be smooth over time. This is also the case for `hierarchicalforecast` (Olivares et al., 2024) package in Python. On the other hand, packages, such as `smooth` (Svetunkov, 2024) and `forecast` (Hyndman et al., 2024) for R and `statsmodels` (Seabold and Perktold, 2010) for Python, provide methods to generate smooth forecasts but do not have the functionality to impose constraints. To our knowledge, no package supports the simultaneous application of both reconciliation and smoothing.

This paper introduces `macroframe-forecast`, a Python package that enables users to generate forecasts that are temporally smooth and meet user-defined constraints. The package produces forecasts in two steps a la Ando and Kim (2023) and Ando (2024). The framework is model agnostic in its first step, allowing users to produce unconstrained forecasts using any preferred forecasting model in the `sktime` package (Loning et al., 2019), including machine learning pipelines or traditional econometric methods. Compatibility with the rich machine learning packages is one of the benefits of writing a package in Python. In the second step, these forecasts undergo a reconciliation process a la Ando (2024) that enforces equality or inequality constraints while smoothing the forecast trajectory over time. The two-step approach contrasts with Chan et al. (2025), who jointly conduct forecasting and reconciliation but require all variables to follow a Gaussian distribution.

The package features plug-and-play simplicity and flexibility to fine-tune the details. The interface is designed so that forecasters can use the package's main class `MFF` without explicitly providing various mathematical inputs for the reconciliation problem. For example, users can specify constraints using strings, rather than inputting matrices as required by other packages. By understanding the conceptual framework and internal structure, however, forecasters can fine-tune the first and second steps independently. For example, forecasters can either use sophisticated machine learning models in the first step or provide the first step forecasts exogenously. Forecasters can then fine-tune the reconciliation step by experimenting with different constraints or smoothness. Such flexibility could be useful for practitioners who want to experiment with different target values, inequality constraints, or smoothness before settling on the final forecasts.

We demonstrate the package's functionality with two examples. The first example focuses on a simple case in which a single variable, U.S. GDP, is forecasted subject to the constraint that the growth rate at the end of the forecast horizon matches a predefined value. The second example illustrates a multivariable scenario, forecasting fiscal variable, namely revenue, expenditure, and interest expense, subject to the accounting identity, with the path of the primary balance given exogenously.

The rest of the paper is structured as follows. Section 2 outlines the conceptual framework of the methods implemented in the package. Section 3 provides instructions for installing and using the package, with illustrative examples. Section 4 concludes the paper.

2. Conceptual framework

This section presents the conceptual framework behind the Python package. The framework consists of two steps, where the first step provides users with a flexible choice of forecasting models, and the second step allows users to adjust the first-step forecasts so that the forecasts are smooth over time and satisfy various constraints, such as accounting identities and prespecified targets. For the theoretical properties and examples of the reconciliation process in the second step, see Ando (2024).

2.1. First-step forecast

The first step of the forecasting process is to generate unconstrained forecasts for all unknown variables across all time horizons. Formally, for each variable $i = 1, \dots, M$, the first-step forecast using information up to time t for $h \geq 1$ periods ahead can be written as

$$\hat{y}_{i,t+h|t} = \mathbb{E}\left[y^*_{i,t+h} \middle| \hat{\theta}_{i,h}\right], \tag{1}$$

where $\hat{\theta}_{i,h}$ refers to the estimated parameters for variable $y^*_{i,t+h} \in \mathbb{R}$. The model used for each variable can be different across i and h and can be parametric or non-parametric, such as Ordinary Least Squares (OLS) or other machine-learning models. In the Python package, users can specify their own models or use the default model.

It is important to note that the first-step forecast is unconstrained. While users have the flexibility to apply any forecasting model, the resulting forecasts may not satisfy essential requirements, such as accounting identities or inequality constraints. Depending on the chosen model, the unconstrained forecast might also exhibit sharp fluctuations or patterns inconsistent with the imposed constraints. The purpose of the second step is to reconcile the first-step forecast, ensuring that the final forecasts are both smoother over the forecast horizon and fully compliant with the specified constraints.

2.2. Second-step reconciliation

Suppose that the first step generates the forecast of M variables, where each variable $i = 1, \dots, M$ has forecast horizon of $h = 1, \dots, T_i$. Let $\hat{y}$ denote the column vector that stacks M variables and N denote the size of the vector

$$\hat{y} \coloneqq \left[\hat{y}_{1,t+1}, \dots, \hat{y}_{1,t+T_1}, \dots, \hat{y}_{M,t+1}, \dots, \hat{y}_{M,T+T_M}\right]' \in \mathbb{R}^N, \tag{2}$$

The second step $\tilde{y} \in \mathbb{R}^N$ adjusts the first-step forecast by (1) smoothing it over time and (2) ensuring that it satisfies the predefined constraints. Let the constraints be denoted by

$$C_{eq}y - d_{eq} = 0, \qquad C_{ineq}y - d_{ineq} \leq 0, \tag{3}$$

where C_{eq}, d_{eq}, C_{ineq}, d_{ineq} are of size $K_{eq} \times N$, $K_{eq} \times 1$, $K_{ineq} \times N$, and $K_{ineq} \times 1$. The second-step forecast $\tilde{y}$ solves

$$\tilde{y} \coloneqq \arg\min_{y \in \mathbb{R}^N} (y - \hat{y})' W^{-1} (y - \hat{y}) + y'\Phi y \;\; s.t. \;\; C_{eq}y = d_{eq}, \qquad C_{ineq}y \leq d_{ineq}\,, \tag{4}$$

where W is an estimator of the forecast error covariance $V(y^* - \hat{y})$. Since the weight W reflects forecast error, accurately forecasted variables tend to be changed less in the second step than the less accurately forecasted variables. Φ is the smoothness matrix

$$\Phi = \begin{bmatrix} \lambda_1 F_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_M F_M \end{bmatrix}, \qquad F_i = \begin{bmatrix} 1 & -2 & 1 & 0 & & & & & \\ -2 & 5 & -4 & 1 & & & & & \\ 1 & -4 & 6 & -4 & & & & & \\ 0 & 1 & -4 & 6 & & & & & \\ & & & & \ddots & & & & \\ & & & & & 6 & -4 & 1 & 0 \\ & & & & & -4 & 6 & -4 & 1 \\ & & & & & 1 & -4 & 5 & -2 \\ & & & & & 0 & 1 & -2 & 1 \end{bmatrix}, \tag{5}$$

where the matrix F_i is the degenerate penta-diagonal matrix used in the calculation of the Hodrick- Prescott filter (1997). The matrix F_i suggests that each λ_i controls the smoothness, defined by how small the difference in difference $\left\{\left(y_{i,t+1}-y_{i,t}\right)-\left(y_{i,t}-y_{i,t-1}\right)\right\}^2$ is, as in the HP filter.

Intuitively, the second-step forecast $\tilde{y}$ tries to be as close as possible to the first-step forecast $\hat{y}$ subject to the constraints. The distance is measured by the inverse of the weight W, and jumps over time are punished by the smoothness matrix Φ.The problem is quadratic programing, and it has a closed-form solution if inequality constraints are slack.

2.2.1. Choice of weight matrix W

Theoretically, the optimal weight is the forecast error covariance $W = V(y^* - \hat{y})$. (Wickramasuriya et al., 2019; Ando and Narita, 2024) In practice, when the matrix is large relative to the sample size, the sample covariance matrix $\hat{\Sigma}$ may not be invertible. Thus, by default, the weight matrix W uses the Oracle Shrinkage Approximating estimator (OAS) à la Chen et al. (2010), which shrinks the sample covariance matrix of the forecast error $\hat{\Sigma}$ towards the diagonal matrix where all diagonal elements are the average of diagonal elements $tr(\hat{\Sigma})/N$. Specifically, suppose the sample size to estimate the matrix is s based on time series cross-validation of historical data. The weight matrix W based on OAS is

$$W_{OAS} = (1-\rho_{OAS})\hat{\Sigma} + \rho_{OAS}\frac{tr(\hat{\Sigma})}{N}I_{N\times N}, \qquad \rho_{OAS} = \min\left\{\frac{\left(1-\frac{2}{N}\right)tr(\hat{\Sigma}^2)+tr(\hat{\Sigma})^2}{\left(s+1-\frac{2}{N}\right)\left[tr(\hat{\Sigma}^2)-\frac{tr(\hat{\Sigma})^2}{N}\right]}, 1\right\}. \tag{6}$$

Alternatively, the weight matrix can use the Oracle Shrinkage Approximating estimator with Diagonal target (OASD) as in Ando and Xiao (2023), which provides robust estimates when different timeseries have different units.

$$W_{OASD} = \rho_{OASD}\hat{\Sigma} + (1-\rho_{OASD})diag\,(\hat{\Sigma}), \qquad \rho_{OASD} = \min\left\{\frac{1}{s\phi}, 1\right\}, \qquad \phi = \frac{tr(\hat{\Sigma}^2) - tr\left(diag(\hat{\Sigma})^2\right)}{tr(\hat{\Sigma}^2) + tr(\hat{\Sigma})^2 - 2tr\left(diag(\hat{\Sigma})^2\right)}. \tag{7}$$

Users can also use the identity matrix as the weight matrix $W = I_{N\times N}$ or provide custom weights.

2.2.2. Choice of smoothness parameters λ

By default, the M smoothness parameters $\{\lambda_i\}_{i=1}^M$ are calculated by

$$\lambda_i = \frac{\lambda_i^*}{\sigma_i^2}, \qquad \sigma_i^2 = \min_t (W_i)_{tt}, \qquad i = 1, \dots, M, \tag{8}$$

where λ_i^* is the standard smoothness parameter of HP filter for the corresponding frequency, like 1600 for the quarterly frequency, and W_i is a $T_i \times T_i$ submatrix of the weight matrix W that corresponds to variable i.

Intuitively, the standard HP filter parameter λ_i^* is used to smooth each time series, but the parameter is normalized so that the objective function (4) is independent of each time series' unit.

When the frequency of the time series is not available, the package chooses λ_i for each time series i by minimizing the mean squared forecast error from time-series cross validation. Users can also provide custom values.

2.3. Discussion

To highlight the benefits and limitations of the conceptual framework, this section makes three observations.

First, the model selection in the first step is flexible, and it is important to choose the right one. Any model can be used as the first-step forecast. It is important to note, however, that any shortcomings in the initial model specification may be carried forward into the reconciliation process through the first step forecast $\hat{y}$ and the forecast error covariance matrix W. Thus, it is important to select a model with high forecast accuracy in the first step.

Second, the choice of constraints and smoothness in the second step is flexible, and it is important to choose the right ones. The constraints can reflect systematic relationships, like accounting identities and non-negativity of certain variables, or come from ad-hoc sources, such as expert judgement. Imposing the right constraints tends to improve the first-step forecast by providing additional information that the statistical models couldn't infer in the first step, but imposing wrong constraints could worsen it. Similarly, imposing the right smoothness can remove noises in the first step forecast, and vice versa.

Third, the `macroframe-forecast` package currently supports only point forecasts. Forecasters can, however, construct confidence intervals by using the forecast errors generated during the first-step forecasting process. One can draw from these forecast errors across all horizons and variables, add them to the point forecasts, and then carry out the reconciliation process. Repeating this procedure generates a distribution of second-step reconciled forecasts, from which confidence or prediction intervals can be derived. This simulation-based approach to generating reconciled forecast distributions aligns with methods used in the reconciliation literature (e.g., Girolimetto et al., 2024).

3. Python package

This section outlines how to use the Python package `macroframe-forecast`. The documentation and Github repository are available online: https://sakaiando.github.io/macroframe-forecast/index.html and https://github.com/sakaiando/macroframe-forecast. They are both live documents and could evolve from this paper.

The primary class within the package is `MFF`, which produces forecasts based on input data and specified constraints. The basic syntax for invoking this class is:

```
>>> df2 = MFF(df, equality_constraints)
```

where `df` is a dataframe with missing values to be forecasted and `constraints` is a list of strings that represents the linear constraints to be satisfied by the variables. In the subsequent sections, we will explain the functionalities of the `MFF` class in detail, covering essential inputs, optional parameters, and advanced features with simple examples.[1]

3.1. Installation

This package is available through `pip` and via GitHub. The following command on the command line installs the `macroframe-forecast` package:

```
pip install macroframe-forecast
```

This will install the package and all dependency packages. In case a previous or incompatible version of a dependency is already installed, this will update these packages to the required version as well. We recommend using Python version 3.11.9 or later to ensure compatibility and optimal performance.

3.2. Inputs

The `MFF` class requires one essential input to generate forecasts, followed by several optional parameters.

3.2.1 Essential inputs

- `df`: A `pandas` dataframe. The dataframe must be in a long format, with each row being a time period, and each column being a variable. The observations to be forecasted are NaN in the dataframe. Island values are known values for variables which are preceded by unknown values. For example, if a forecaster expects the GDP growth rate for a country to be 4 percent at the end of a period of, say 5 years from the current time period, but has no view on the growth rate during the interim years, the 4 percent is an island value following four NaN values as the table below.

Table 1. Example of island value

period	GDP growth
0	3.00
1	NaN
2	NaN
3	NaN
4	NaN
5	4.00

[1] This exploration will include discussions on handling data with a single frequency, although the `macroframe-forecast` package includes an experimental `MFF_mixed_frequency` class, which accommodates multiple frequencies pending the resolution of a bug in the `pandas` library (GitHub issue 59775).

3.2.2 Optional inputs

- `forecaster`: Users can provide a descendant of `BaseForecaster` class in the `sktime` package, including the pipeline with pre-processing steps and models for generating the first-step forecast. When no forecasting pipeline is defined by the user, the function `DefaultForecaster()` is called. `DefaultForecaster()` is a pipeline that uses a time-series cross validation algorithm with five folds to select the best performing model among the following standard forecasting models:
  1. Naïve forecaster;
  2. Elastic Net;
  3. Ordinary Least Squares (OLS) with features created by Primary Component Analysis (PCA);
  4. Ordinary Least Squares (OLS) using the first feature.

  In the extreme cases where the number of non-NaN observations is too small for Grid Search algorithm, `DefaultForecaster()` is `NaiveForecaster()` from `sktime`.

- `equality_constraints`: A list of equality constraints expressed as strings. These constraints can either be specified to bind at a certain period or can be defined with wildcards that extend the constraint to all forecast horizons. The constraints are specified as a list of strings using the column names of `df` and time periods. If no input is provided, no constraints will be imposed, and the second-step forecast will be as the smoothed version of the first-step forecast. If an island value is provided in `df` as in Table 1, users do not need to specify it in `equality_constraints`.

- `inequality_constraints`: A list of inequality constraints, analogous to the `equality_constraints`.

- `parallelize`: A Boolean indicator of whether `dask`'s parallel computing is utilized for generating forecasts.

- `n_forecast_error`: The number of time series cross validationsplits to generate the first-step forecast error covariance matrix $\hat{\Sigma}$. By default, this value is set to be 5.

- `shrinkage_method`: The method used for shrinking the sample covariance matrix estimated from first-step forecasts. By default, the Oracle Approximating Shrinkage (OAS) Estimator (Chen et al., 2010) is used for the shrinkage step. Other options are Oracle Approximating Estimator with Diagonal target (OASD) as laid out in Ando and Xiao (2023), identity matrix, or the monotone diagonal method.

- `default_lam`: The value of lambda λ_i in to be used for smoothing forecasts for the second step. By default, this value is set to be -1, which means that the value of lambda is to be selected by minimizing the mean-squared error from time-series cross validation. When it is 0, no smoothing will be applied.

- `max_lam`: The maximum value that the lambda parameter λ_i can take if it is being estimated optimally (`default_lam` is set to -1). By default, this is set to 129600.

3.3. Outputs

Since the macroframework forecast is produced in two steps, and the second step generates the smoothed forecasts that satisfy constraints, the main output is denoted by `df2`.

- `df2`: A dataframe with the same size as `df`, with NaN replaced by the forecasts that satisfy specified constraints.

Apart from this, the following attributes store information on the intermediate steps of the forecasting process:

- `df0`: The original input dataframe, with all island values set to NaN.
- `df1`: A dataframe with first-step forecasts filled in the NaN cells of dataframe `df0`.
- `df1_model`: A dataframe where each cell corresponds to the model used for generating the corresponding forecasted value in `df1`.
- `pred`: A dataframe with predicted values from in-sample predictions generated using pseudo-historical datasets. This is used for generating the first-step forecast errors.
- `true`: A dataframe with actual values of variables corresponding to the predicted values in `pred`. The first step forecast errors are produced by subtracting `true` from `pred`.
- `model`: A dataframe containing the models used for generating in-sample forecasts stored in `pred`.
- `C_eq`: The left side of the equality constraint equation $C_{eq}y - d_{eq} = 0$ in matrix form from equation (3).
- `d_eq`: The right side of the equality constraint equation $C_{eq}y - d_{eq} = 0$ in matrix form from equation (3).
- `C_ineq`: The left side of the inequality constraint equation $C_{ineq}y - d_{ineq} \leq 0$ in matrix form from equation (3).
- `d_ineq`: The right side of the equality constraint equation $C_{ineq}y - d_{ineq} \leq 0$ in matrix form from equation (3).
- `W`: The weight matrix implied by the first step forecast error.
- `Phi`: Smoothing matrix used for generating smooth second-step forecasts.
- `shrinkage`: The shrinkage parameter ρ associated with the second-step adjustment, corresponding to ρ_{OAS} when OAS is used and ρ_{OASD} when OASD is used. `np.nan` when identity option is used.
- `smoothness`: A series containing the smoothing variable λ_i^* used for each variable i.

3.4. Examples

In this section, we illustrate how to use the package using two examples. The first example is kept simple, illustrating the plug-and-play aspect of the package. The second example illustrates how the package can handle more complicated situations with multiple variables, as well as equality and inequality constraints.

3.4.1. Single variable example

In this section, we use US nominal GDP data in trillion US dollars from the World Economic Outlook database. We use the historical data from 1950-2024 and generate forecasts for 2025-2030. The dataframe has only one column, with data up to 2024. Rows 2025-2030 are all NaN to be forecasted.

```
>>> df0
```

year	GDP
1950	0.30
1951	0.35
...	...
2024	28.18
2025	NaN
2026	NaN
2027	NaN
2028	NaN
2029	NaN
2030	NaN

Suppose we want to constrain the growth rate for GDP for the year 2030 around 4 percent. This can be written as

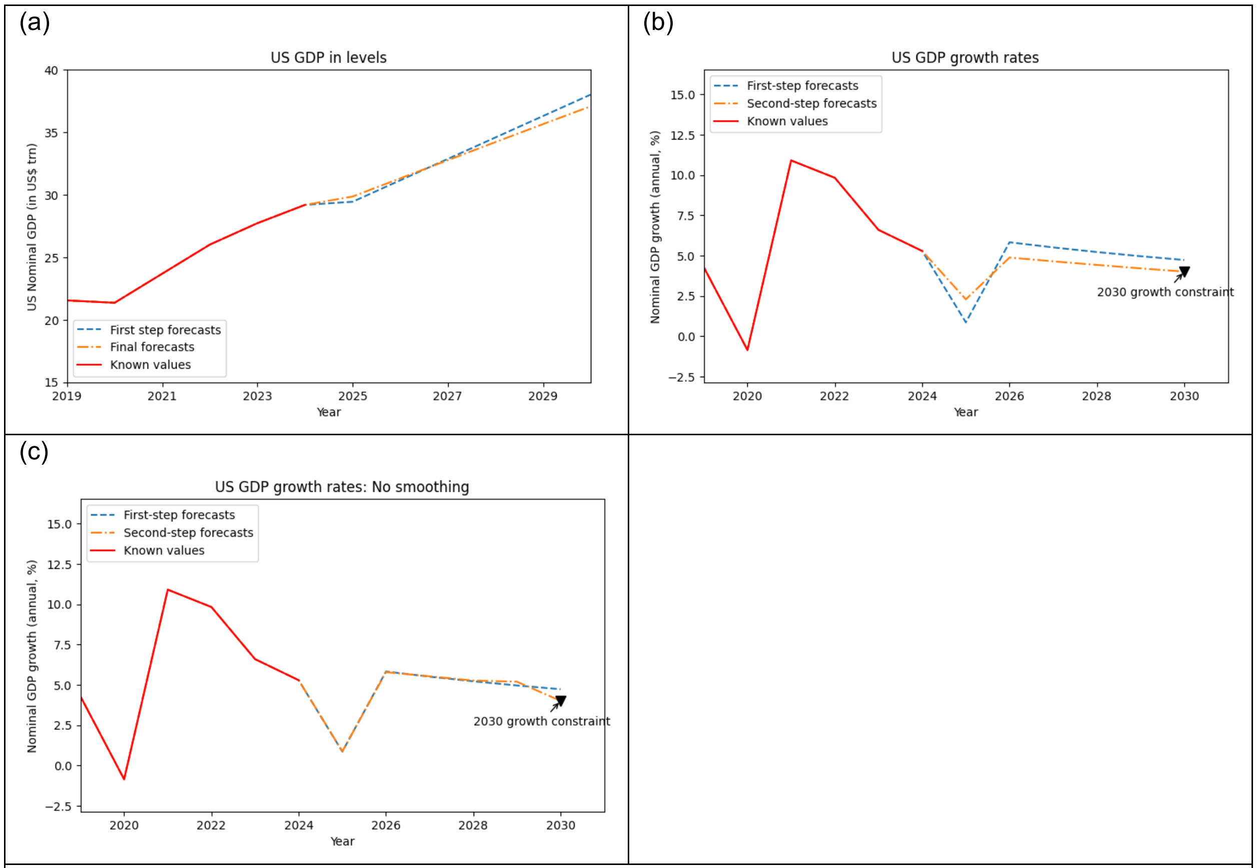
$$\frac{GDP_{2030} - GDP_{2030}}{GDP_{2030}} \times 100 = 4 \Leftrightarrow GDP_{2030} - 1.04 \times GDP_{2029} = 0. \tag{9}$$

The forecast with this constraint on long-term growth can be made by

```
>>> m = MFF(df =df0,
>>>         equality_constraints = ['GDP_2030 - 1.04 * GDP_2029'])
>>> m.fit()
```

The output from executing the `fit` command is stored inside the object `m`. The first- and second-step forecasts are stored as `df1` and `df2` respectively. The first-step forecast may not be smooth, and the annual GDP growth may not be 4 percent at the end of the period. The second step forecast, however, satisfies the imposed growth rate constraint, as is evident in Figure 1(b). If the forecaster wants to only reconcile the first step forecasts to the constraint but does not want to smooth the forecast, they can do so by setting the parameter `default_lam` to 0 in the `MFF` call. Figure 1 (c) shows that, in this case, the growth rates in 2029 and 2030 are not smooth.

Figure 1. Forecast of US Annual GDP



Source: IMF April 2025 World Economic Outlook database and authors' calculations.
Note: Panel (a) shows the forecast of annual US GDP in trillion USD. Panel (b) shows the growth rates computed from the level forecasts. Panel (c) shows the output when no smoothing is applied.

As discussed, it is not necessary that all horizons in the first step were forecasted using the same model. The models that are used for each first-step forecast can be accessed by using the following command:

```
>>> m.df1_model
```

For example, we can see the model used for forecasting the 2028 GDP value as

```
>>> m.df1_model.loc['2028', 'GDP']
```

3.4.1.1. Customizing second step reconciliation

What if users want to fine tune the second step reconciliation after running the example in the previous section 3.4.1? For example, users may want to provide an exogenous first-step forecast, weight matrix, or the smoothness parameters. Such experiments can be conducted by using the `Reconciliation` function. It requires knowledge about the conceptual framework and internal structure of the `MFF` class, but it provides additional flexibility in the forecasting exercise, so we illustrate a potential use in this section.

Suppose the user wants to replace the first step forecast by

```
>>> GDP_forecasts_external = pd.DataFrame({"GDP": [29.0, 31.5, 33, 30.2, 36.8, 39]},
                                  index = [2025, 2026, 2027, 2028, 2029, 2030])
```

After running the example in the previous section 3.4.1, the user can extract and replace the corresponding object with this alternative first step forecast.

```
>>> y1_exo = m.y1
>>> y1_exo[:] = GDP_forecasts_external['GDP']
```

Similarly, the weight matrix can be changed to an identity matrix as below.

```
>>> W_exo = m.W
>>> W_exo.iloc[:,:] = np.eye(len(m.y1))
```

If the user wants to halve the smoothness parameter λ^*, `GenSmoothingMatrix()` can be used to update the corresponding smoothing matrix Φ.

```
>>> from macroframe_forecast.utils import GenSmoothingMatrix
>>> smoothness_exo = m.smoothness * .5
>>> Phi_exo = GenSmoothingMatrix(m.W, smoothness_exo)

>>> from macroframe_forecast.utils import Reconciliation
>>> alternative_y2 = Reconciliation(y1= y1_exo,
>>>                                   W = W_exo,
>>>                                   Phi = Phi_exo,
>>>                                   C = m.C, d = m.d,
>>>                                   C_ineq = m.C_ineq, d_ineq = m.d_ineq)
```

3.4.2. Multivariable example

In this section, we analyze fiscal data for the United States, focusing on revenue, expenditure, interest payments, and primary balance from the April 2025 WEO. They satisfy:

Primary balance = revenue – expenditure + interest payments.

These fiscal variables are expressed as ratios to nominal GDP and are used going forward for the forecasting exercise. The April 2025 WEO provides historical data up to 2024 and projections for 2025-30. We assume that the forecaster takes the WEO forecast on primary balance to GDP ratio as given over the entire forecast horizon, but expenditure, revenue, and interest payments to GDP are unknown, so we want to forecast these three variables up to 2030. The input dataframe for the `MFF` is structured as follows:

```
>>> fiscal_data
```

year	exp	rev	int_payments	pb
2001	32.80	32.26	3.26	2.71

year	exp	rev	int_payments	pb
2002	33.70	29.88	2.89	-0.93
2003	34.04	29.27	2.66	-2.11
2004	33.72	29.48	2.56	-1.68
2005	33.93	30.85	2.70	-0.37
...	...	...	...	...
2024	37.59	30.33	4.20	-3.06
2025	NaN	NaN	NaN	-2.16
2026	NaN	NaN	NaN	-1.06
2027	NaN	NaN	NaN	-0.90
2028	NaN	NaN	NaN	-1.25
2029	NaN	NaN	NaN	-1.14
2030	NaN	NaN	NaN	-1.32

One constraint we impose is that the fiscal identity holds for the entire forecast horizon. In addition, we assume that the forecaster wants the expenditure to GDP ratio to be 37 percent in 2030. Instead of specifying the fiscal identity constraint separately for each horizon, we can use the wildcard functionality to write the constraint as follows:

```
>>> fiscal_constraint = ['pb? - rev? + exp? - int_payments?',
                         'exp_2030 - 37']
```

The question mark “?“ is a wildcard that expands the constraint across all forecast horizons in one go. The constraint on expenditure in 2030, `exp_2030 = 37`, is written explicitly, but forecasters can also set the corresponding cell of the dataframe to be 37. Such an island value will be automatically treated as an equality constraint.

To illustrate how to impose inequality constraints, suppose the forecaster also believes that the interest payments to GDP ratio always stay between 2.5 and 3.5 percent. Such inequality constraints can be expressed as following.

```
>>> int_constraint = ['int_payments? - 3.5', '2.5 - int_payments?']
```

We describe how to use a user-defined model to generate the first step forecast instead of relying on the default forecasting pipeline. Suppose that the forecaster wants to use a linear regression model for making predictions. We can use the `LinearRegression` class in `sklearn` package and wrap it in `DirectReductionForecaster` from `sktime` to make the model compatible with the `sktime` forecasting pipeline.

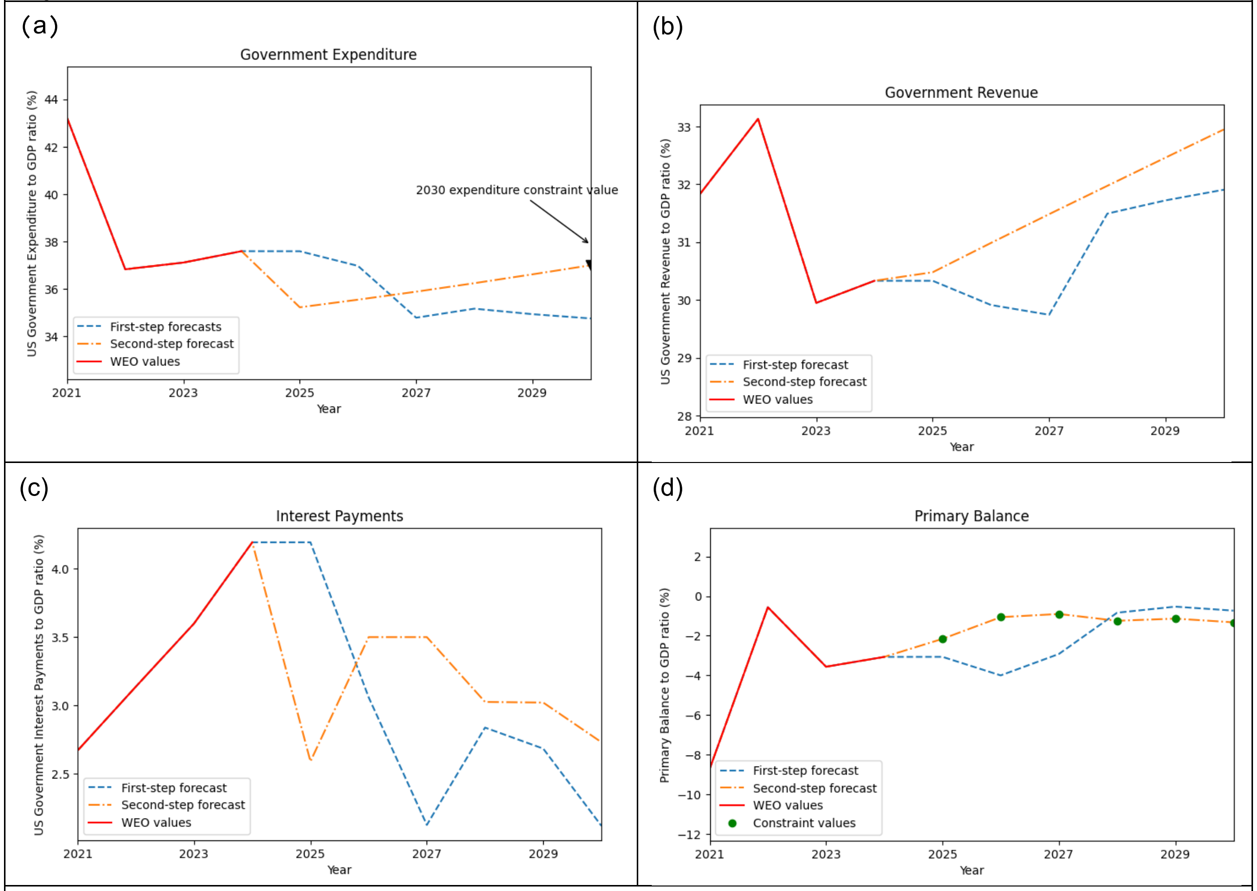
```
>>> from sktime.forecasting.compose import ForecastingPipeline
>>> from sktime.forecasting.compose import DirectReductionForecaster
>>> ols_model = ForecastingPipeline(steps=[
>>>    (‘ols’,DirectReductionForecaster(LinearRegression()))])
```

The forecaster can add additional steps in the pipeline before the model is applied, such as taking logs and first differences. Once this is defined, we can call `MFF` in the following manner:

```
>>> m = MFF(df = fiscal_data,
>>>          equality_constraints = fiscal_constraint,
>>>          inequality_constraints = int_constraint,
>>>          forecaster = ols_model)
>>> m.fit()
```

From the first- and second-step forecasts of expenditure, revenues, and interest payments in Figures 2 (a), (b) and (c), we see that the second-step forecasts are smoother than the first step. We also see that the expenditure forecast in the second step is equal to the target in the equality constraint, and interest payments always remain within the predefined bounds. Figure 2 (d) compares the primary balance derived from the first- and second-step forecasts. We see that, while the first-step forecast does not satisfy the accounting identity, the second-step forecast does.

Figure 2. Forecasts of Annual US fiscal variables



Source: IMF April 2025 World Economic Outlook database and authors’ calculations.

Note: Panel (a), (b), and (c) show the forecasts for the US annual expenditure, revenue, and interest expense to GDP ratio, respectively. Panel (d) shows the implied primary balance from the first- and second-step forecasts.

4. Conclusion

This paper introduces a forecasting framework and accompanying Python package that enables users to generate forecasts exhibiting temporal smoothness while adhering to user-specified constraints. The package distinctly separates model estimation from the imposition of constraints and smoothness adjustments, allowing users to employ any statistical or machine learning models to produce the initial forecasts. This structure provides forecasters with flexibility in their modeling choices, while ensuring that the final forecasts remain internally consistent and smooth.

The `macroframe-forecast` package, however, is a beta version, and the internal structure and the user interface will continue to be improved. Extending support to mixed-frequency time series, speeding up parallel processing, and Excel add-in would be interesting directions for future development.

References

Ando, Sakai (2024), “Smooth Forecast Reconciliation,” IMF Working Paper 24/66.

Ando, Sakai and Taehoon Kim (2023), “Systematizing Macroframework Forecasting: High-Dimensional Conditional Forecasting with Accounting Identities,” IMF Economic Review, Vol. 72, pp 1386-1410.

Ando, Sakai and Mingmei Xiao (2023), “High-Dimensional Covariance Matrix Estimation: Shrinkage Toward a Diagonal Target,” IMF Working Paper 23/257.

Ando, Sakai and Futoshi Narita (2024), “An Alternative Proof of Minimum Trace Reconciliation,” Forecasting, Vol. 6(2), pp 456-461.

Athanasopoulos, George, Rob J Hyndman, Nikolas Kourentzes, Anastasios Panagiotelis (2024), “Forecast reconciliation: A review,” International Journal of Forecasting, Vol. 40(2), pp 430-456.

Chan, Joshua, Davide Pettenuzzo, Aubrey Poon, Dan Zhu (2025), “Conditional forecasts in large Bayesian VARs with multiple equality and inequality constraints,” Journal of Economic Dynamics and Control, Vol. 173, 105061.

Chen, Yilun, Ami Wiesel, Yonina Eldar, Alfred Hero (2010), “Shrinkage Algorithms for MMSE Covariance Estimation,” IEEE Transactions on Signal Processing, Vol. 58, pp 5016-5029.

Girolimetto, Daniele and Tommaso Di Fonzo (2023), “FoReco: Point Forecast Reconciliation,” R package version 2.6.

Girolimetto, Daniele and Tommaso Di Fonzo (2023), “Point and Probabilistic Forecast Reconciliation for General Linearly Constrained Multiple Time Series,” Statistical Methods & Applications, Vol. 39(1), pp 39-57.

Girolimetto, Daniele, George Athanasopoulos, Tommaso Di Fonzo, and Rob Hyndman (2024), “Cross-temporal probabilistic forecast reconciliation: Methodological and practical issues,” International Journal of Forecasting, Vol. 40(3), pp 1134-1151.

Hyndman, Rob, Roman A. Ahmed, George Athanasopoulos, Han Lin Shang (2011), “Optimal combination forecasts for hierarchical time series,” Computational Statistics & Data Analysis, Vol. 55(9), pp 2579-2589.

Hyndman, Rob, Alan Lee, Earo Wang, Shanika Wickramasuriya (2021), “hts: Hierarchical and Grouped Time Series,” R package version 6.0.2.

Hyndman, Rob, George Athanasopoulos, Christoph Bergmeir, Gabriel Caceres, Leanne Chhay, Mitchell O'Hara-Wild, Fotios Petropoulos, Slava Razbash, Earo Wang, Farah Yasmeen (2024), “forecast: Forecasting functions for time series and linear model,” R package version 8.22.0.

Hodrick, Robert and Edward Prescott (1997), “Postwar U.S. Business Cycles: An Empirical Investigation,” Journal of Money, Credit and Banking, Vol. 29(1), pp 1-16.

Olivares, Kin G., Azul Garza, David Luo, Cristian Challú, Max Mergenthaler, Souhaib Ben Taieb, Shanika L. Wickramasuriya, Artur Dubrawski (2024), “HierarchicalForecast: A Reference Framework for Hierarchical Forecasting in Python,” arXiv:2207.03517.

Loning, Markus, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz Kiraly (2019), “sktime: A Unified Interface for Machine Learning with Time Series.”

Seabold, Skipper, and Josef Perktold (2010), “statsmodels: Econometric and statistical modeling with python,” Proceedings of the 9th Python in Science Conference.

Svetunkov, Ivan (2024), “smooth: Forecasting Using State Space Models,” R package version 4.0.1.

Wickramasuriya, Shanika, George Athanasopoulos, and Rob Hyndman (2019), “Optimal Forecast Reconciliation for Hierarchical and Grouped Time Series Through Trace Minimization,” Journal of the American Statistical Association, Vol. 114(526), pp 804-819.