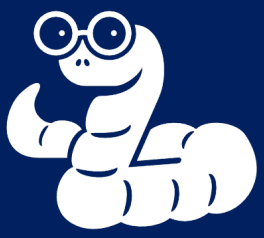

WORKing papers in Management Science

WORMS/25/02

PostForecasts.jl: A Julia package for probabilistic forecasting by postprocessing point predictions

Arkadiusz Lipiecki[1]
Rafał Weron[1]

[1] Faculty of Management, Wrocław University of Science and Technology, Poland

WORMS is a joint initiative of the Management Science departments of the Wrocław University of Science and Technology, Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland

PostForecasts.jl: A Julia package for probabilistic forecasting by postprocessing point predictions

Arkadiusz Lipiecki[a,*], Rafał Weron[a]

[a]*Faculty of Management, Wrocław University of Science and Technology, Poland*

Abstract

Postprocessing of point predictions is a relatively simple and efficient way to compute probabilistic forecasts, which are the basis of uncertainty assessment for decision support and risk management. The *PostForecasts.jl* package in Julia provides types and functions to easily convert point forecasts into probabilistic ones using Historical Simulation, Conformal Prediction, Isotonic Distributional Regression, and variants of Quantile Regression Averaging. By leveraging the developments in the point forecasting literature, it offers a set of easy-to-use, computationally undemanding, and robust tools to derive predictive distributions.

Keywords: probabilistic forecasting, postprocessing, combining forecasts, uncertainty quantification

Metadata

Nr.	Code metadata description	Metadata
C1	Current code version	v0.1.0
C2	Permanent link to code/repository used for this code version	`https://github.com/lipiecki/PostForecasts.jl/releases/tag/v0.1.0`
C3	Permanent link to Reproducible Capsule	N/A
C4	Legal Code License	MIT license (MIT)
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Julia
C7	Compilation requirements, operating environments & dependencies	Julia 1.10 (see Project.toml)
C8	If available Link to developer documentation/manual	`https://lipiecki.github.io/PostForecasts.jl/dev/`
C9	Support email for questions	arkadiusz.lipiecki@pwr.edu.pl

Table 1: Code metadata (mandatory)

*Corresponding author.

Email addresses: `arkadiusz.lipiecki@pwr.edu.pl` (Arkadiusz Lipiecki), `rafal.weron@pwr.edu.pl` (Rafał Weron)

1. Motivation and significance

Probabilistic forecasts allow decision makers to quantify risks, prepare for different scenarios, and predict future uncertainty [1, 2]. However, implementing and calibrating models that return probabilistic forecasts, e.g., a set of quantiles approximating the predictive distribution or parameters of a parametric distribution, can be a complex and computationally demanding task. A viable and efficient workaround is to postprocess point forecasts using methods that can output probabilistic forecasts [3–5].

The *PostForecasts.jl* package in Julia has been designed to do just that. It allows the user to choose between a number of postprocessing methods that differ in terms of computational complexity and assumptions about the underlying distributions: a benchmark approach based on normally distributed errors, Historical Simulation (HS) [6–8], Conformal Prediction (CP) [9, 10], four variants of an established postprocessing scheme in energy forecasting – Quantile Regression Averaging (QRA) [5, 11–14] and a novel Isotonic Distributional Regression (IDR)[15–17]. In addition, the package offers a choice of combination and conformalization schemes to further improve the probabilistic forecast, and accuracy metrics to evaluate it.

The package relies on deterministic models for reliable and repeatable results, does not require hyperparameter tuning, and leverages forecaster diversity via averaging. We believe that following these principles allowed us to develop a robust tool for computing probabilistic forecasts that combines ease of use, high accuracy, low computational costs and good interpretability of the results. This makes *PostForecasts.jl* an attractive choice for both academic and industrial applications.

The remainder of the paper is structured as follows. In Section 2 we first describe the software architecture and introduce the implemented postprocessing methods. Next, we review additional software functionalities, like forecast averaging, conformalization of quantile forecasts, assessing forecaster contributions using Shapley values, evaluation metrics, and the sample datasets shipped with *PostForecasts.jl*. In Section 3 we provide snippets illustrating the main functionality of the package. Finally, in Section 4 we elaborate on the impact and conclude.

2. Software description

The core functionality of *PostForecasts.jl* is deriving probabilistic forecasts ($\rightarrow$ *QuantForecasts* objects; see Sec. 2.1) from point predictions ($\rightarrow$ *PointForecasts* objects) with a single call of the *point2quant*() function. The latter utilizes the information contained in pairs $(\hat{\mathbf{y}}_t, y_t)$, where $\hat{\mathbf{y}}_t = \{\hat{y}_{t,1}, ..., \hat{y}_{t,m}\}$ is a vector of out-of-sample point predictions of y_t from $m \geq 1$ forecasters, and returns target

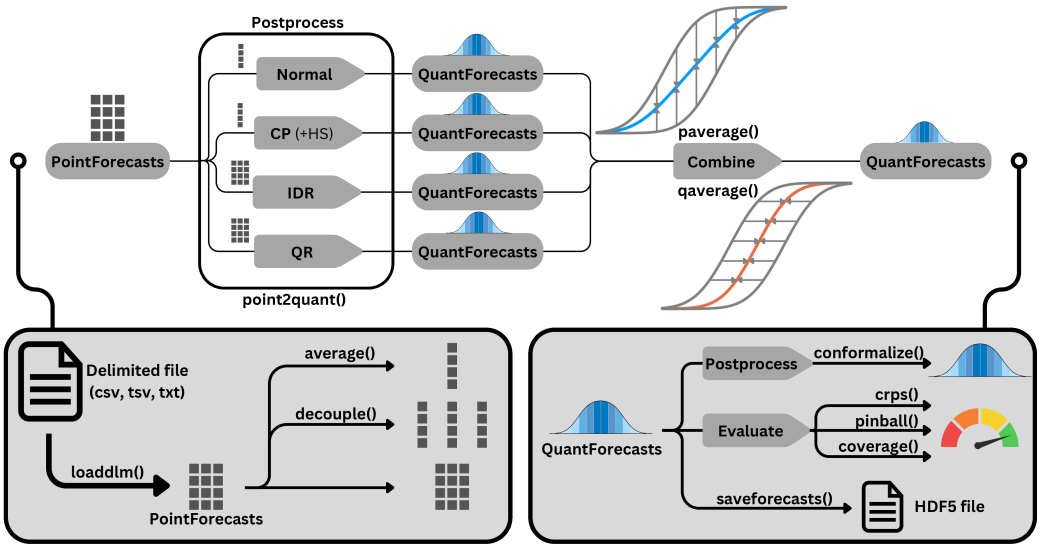

Figure 1: Structures, functions and workflow of the *PostForecasts.jl* package.

quantiles using a selected postprocessing method, training window length, and retraining frequency. The structures, functions and workflow of the package are illustrated in Figure 1.

2.1. Software architecture

To make forecast processing easy and unify the workflow, the package uses two types belonging to *Forecasts* supertype. *PointForecasts* stores the series of point forecasts along with observations and identifiers (integer labels, e.g., dates in the YYYYMMDD format). Multiple forecasts of the same target variable can be stored in a single *PointForecasts* object, allowing for multiple postprocessing, i.e., estimating predictive distributions conditional on a pool of point forecasts.

The *QuantForecasts* structure holds the series of probabilistic forecasts and the observed values. Predictive distributions are stored as a set of quantiles of specified probabilities. Analogously to *PointForecasts*, *QuantForecasts* also stores identifiers. Both types are provided with position-based and label-based indexing and slicing.

The package repository github.com/lipiecki/PostForecasts.jl is setup with a GitHub Actions workflow that automatically runs tests on every push and pull request. Testing is performed on the latest stable release of Julia on Ubuntu, Windows and MacOS platforms. At the time of submission, we can report a 100% test coverage, tracked by Codecov (https://about.codecov.io/). *PostForecasts.jl* is registered in the General Julia package registry, so it can be easily installed with:

```
using Pkg
```

```
Pkg.add("PostForecasts")
```

2.2. Implemented postprocessing methods

The *point2quant*() function converts point predictions stored in the *PointForecasts* structure into target quantiles and saves them in the *QuantForecasts* structure, see Figure 1. Postprocessing is performed with one of the implemented methods, described in Sections 2.2.1-2.2.4. Computations for each method are performed with specialized objects belonging to the *PostModel* abstract type. Methods that generate predictive distributions conditional on a single input are of the *UniPostModel* supertype, while these allowing for processing multiple inputs belong to *MultiPostModel* supertype. If a *UniPostModel* is used for postprocessing a pool of point forecasts, the pool average $\tilde{y}_t$ will be treated as input:

$$\tilde{y}_t = \frac{1}{m} \sum_{i=1}^{m} \hat{y}_{t,i}, \tag{1}$$

where $\hat{y}_{t,i}$ is the prediction for time time t from forecaster i and y_t is the observed value at time t. Apart from using implemented models via *point2quant*() function, it is also to use them directly on arrays of data with *train*() and *predict*() methods. See Section 3 for example usage of *point2quant*() and Table 2 for a cheatsheet of available methods.

Table 2: Summary of the types of methods implemented, providing their supertypes indicating whether the method supports multiple (→ *MultiPostModel*) or only single input (→ *UniPostModel*) postprocessing, and respective keyword arguments for the *point2quant*() function.

Method	**Quantile Regression**	**Isotonic Distributional Regression**	**Normal Distribution**	**Conformal Prediction**	**Historical Simulation**
Type	QR	IDR	Normal	CP	
Supertype	MultiPostModel		UniPostModel		
Keyword	:qr	:idr	:normal :zeronormal	:cp	:hs

2.2.1. The 'normal' benchmark

This benchmark model assumes that the prediction errors $\varepsilon_t = y_t - \tilde{y}_t$ are normally distributed. Training corresponds to estimating the sample mean $\hat{\mu}$ (the package allows setting a fixed mean $\hat{\mu} = 0$) and the sample standard deviation $\hat{\sigma}$ of

ε_t for $t \in \mathcal{S}$, where $\mathcal{S}$ is the training set (or window). The τ-th quantile conditional on $\tilde{y}_t$ is obtained via:

$$\hat{q}_{\tau|\tilde{y}_t} = \tilde{y}_t + \hat{\mu} + \hat{\sigma} F_N^{-1}(\tau), \tag{2}$$

where $F_N^{-1}(\tau)$ is the inverse of the standard normal cumulative distribution function. Call *point2quant*() with the keyword argument *method=:normal* to use a normal model that estimates both the standard deviation and the mean, or with *method=:zeronormal* to fix the mean at 0.

2.2.2. Historical Simulation and Conformal Prediction

Historical Simulation (HS) [7, 8] is a very simple model-independent approach that computes a prediction of the τ-quantile of variable y_t as a sum of the point forecast $\tilde{y}_t$ and a sample τ-quantile $Q_\tau(\varepsilon_t)$ of the prediction errors $\varepsilon_t = y_t - \tilde{y}_t$ for $t \in \mathcal{S}$:

$$\hat{q}_{\tau|\tilde{y}_t} = \tilde{y}_t + Q_\tau(\varepsilon_t). \tag{3}$$

Although the method itself is much older, the term 'historical simulation' can be traced back to the early 1990s and the beginnings of Value-at-Risk (VaR) in risk management practice [6]. Interestingly, by 2005, nearly 75% of the banks were still using HS instead of the more advanced VaR methods that had been developed in the meantime [7].

The related concept of Conformal Prediction (CP) originated in the machine learning literature [18]. Like HS, it computes PIs based on the prediction errors from an arbitrary point forecasting model. The implemented version corresponds to the *inductive conformal prediction* (ICP) [10, 19]. Although Zaffran et al. [20] argue that the *split conformal prediction* (SCP) corresponds to the ICP, we would like to emphasize that the approach we adopt does not use a 'split' and hence differs from what is commonly understood by SCP. Since we are only interested in postprocessing point forecasts from an already trained model, we do not need to split the data. Therefore, the calibration of conformal prediction in *PostForecasts.jl* is performed directly on the provided out-of-sample point forecasts.

In the training step, the so-called *non-conformity scores* of the form: $\lambda_t = |\varepsilon_t| = |y_t - \tilde{y}_t|$ are calculated for $t \in \mathcal{S}$; see [21] for alternative non-conformity measures and [18] for a general discussion. In the prediction step, the τ-th quantile conditional on $\tilde{y}_t$ is obtained by shifting the point forecast by an appropriate empirical quantile of the non-conformity score:

$$\hat{q}_{\tau|\tilde{y}_t} = \tilde{y}_t - \mathbb{1}_{\tau \leq 0.5} Q_{2\tau}(\lambda) + \mathbb{1}_{\tau \geq 0.5} Q_{2(1-\tau)}(\lambda), \tag{4}$$

where $Q_\tau(\lambda)$ is the τ-th sample quantile of λ_t. While $[\tilde{y}_t - Q_\tau(\lambda), \tilde{y}_t + Q_\tau(\lambda)]$ is a valid $(1-\tau)$-prediction interval without any requirements on the underlying distribution, extracting quantiles from this PI requires the assumption of symmetrically

distributed errors. This is in contrast to the HS which admits non-symmetric PIs around the point forecast.

Note that Historical Simulation can be considered as a variant of Conformal Prediction, which uses non-absolute forecast errors as the non-conformity score: $\lambda_t = \varepsilon_t = y_t - \tilde{y}_t$. For this reason, both methods use the same underlying model type *CP*. Call *point2quant*() with the keyword argument *method=:cp* to use conformal prediction or *method=:hs* for historical simulation. Moreover, *PostForecasts.jl* offers conformalization of quantile forecasts through the *conformalize*() function, see Appendix A.

2.2.3. Isotonic Distributional Regression

Isotonic Distributional Regression (IDR) has been introduced by Henzi et al. [15] as a nonparametric method for estimating distributions that are isotonic in the regressed variable. The latter means that the quantiles of such distributions are nondecreasing with respect to the regressor. Note that unlike the methods discussed in Sections 2.2.1 and 2.2.2, IDR works with individual forecasters $\hat{y}_t \in \{\hat{y}_{t,1}, ..., \hat{y}_{t,m}\}$, not their average $\tilde{y}_t$.

In the training step, all pairs $(\hat{y}_t, y_t)$ are sorted to be ascending in $\hat{y}_t$ for all $t \in \mathcal{S}$; we denote this set by $(\hat{y}_i^{\uparrow}, y_i^{\uparrow})_{i=1}^n$. Then, the conditional distributions $\hat{F}_i(z) = \hat{F}(z|\hat{y}_i^{\uparrow})$ are obtained by solving the following min-max problem via the abridged pool-adjacent-violators algorithm [16]:

$$\hat{F}_i(z) = \min_{k=1,...,i} \max_{j=k,..,n} \frac{1}{j-k+1} \sum_{l=k}^{j} \mathbb{1}\{y_l^{\uparrow} < z\}, \tag{5}$$

where $z \in (y_i^{\uparrow})_{i=1}^n$, and the conditional distribution for any $\hat{y} \in \mathbb{R}$ is obtained by interpolation. For a schematic representation of the IDR algorithm and a detailed discussion, see Figure 2 and Section 4.3 in [5].

Finally, since *QuantForecasts* stores predictive distributions in the form of quantiles, we determine IDR-implied quantiles at specified levels as:

$$\hat{q}_{\tau|\hat{y}} = \min\{z : \hat{F}(z|\hat{y}) \geq \tau\}. \tag{6}$$

The multiple IDR approach is implemented as a linear pool of independent IDR solutions estimated for each of the input point forecast series. In such a case, multiple IDR models are estimated and the resulting distribution functions $\hat{F}(z)$ are averaged. Since z is limited to the observations in $\mathcal{S}$, the distributions resulting from estimated IDRs are defined at the exact same points, which allows to efficiently compute their vertical average across probabilities by directly calculating Eq. (8). To use IDR in *point2quant*(), set *method=:idr*.

2.2.4. Quantile Regression Averaging

Quantile Regression Averaging (QRA) is a well-established postprocessing method introduced by Nowotarski and Weron [11] that has found numerous applications in energy forecasting [8, 12–14, 22–27]. It computes conditional quantiles from a linear combination of the $m \geq 1$ forecasters:

$$\hat{q}_{\tau|\hat{y}_{t,1},...,\hat{y}_{t,m}} = \beta_0 + \beta_1 \hat{y}_{t,1} + ... + \beta_m \hat{y}_{t,m}. \tag{7}$$

QRA is the most computationally demanding method of the ones implemented, since β_i's are estimated using *quantile regression* [28], which minimizes the so-called *pinball loss*, see Section 2.3.2, for each quantile τ. For this task, *PostForecasts.jl* employs *JuMP.jl* with an open source HiGHS solver. Apart from the original QRA method [11], the package allows us to compute the Quantile Regression Machine (QRM) [25, 29] and quantile regression with vertical (called F-Ave or QRF) or horizontal (called Q-Ave or QRQ) averaging [30, 31]. For an illustrative example see Appendix C.2. Call *point2quant*() with *method=:qr* to postprocess forecasts with quantile regression.

2.3. Additional software functionalities

2.3.1. Forecast averaging

Since combining forecasts is known to improve predictive accuracy [32], *PostForecasts.jl* provides forecast averaging functionality for both point and probabilistic predictions. For averaging distributions represented by sets of quantiles in the *QuantForecasts* structure, two schemes are implemented [5, 30, 33]:

- vertical averaging of probabilities → function *paverage*(),
- horizontal averaging of quantiles → function *qaverage*(),

see the graphical illustration in Figure 1. Ensembles of point forecasts can be averaged using the mean or the median → function *average*().

Averaging of probabilities. Vertical averaging of m distributionsis defined by:

$$\tilde{F}(z) = \frac{1}{m} \sum_{i=1}^{m} \hat{F}_i(z). \tag{8}$$

Since *QuantForecasts* store predictive distributions in the form of tabulated quantile functions, *paverage*() computes the vertical average of input distributions in the following steps:

1. Take m input distributions $\hat{q}^{(i)}_{\tau_i}$, where $i \in \{1, ..., m\}$, $\tau_i \in \mathcal{T}_i$, and $\mathcal{T}_i$ are quantile levels of i-th distribution.

2. Calculate $\tilde{F}(z)$ for $z \in \{\hat{q}_{\tau_i}^{(i)} : \tau_i \in \mathcal{T}_i,\ i \in \{1, ..., m\}\}$ according to

$$\tilde{F}_z = \frac{1}{m} \sum_{i=1}^{m} \max\{\tau_i : \hat{q}_{\tau_i}^{(i)} \leq z\},$$

   where we implicitly extend each distribution i with quantile predictions $\hat{q}_0^{(i)} = -\infty$ and $\hat{q}_1^{(i)} = +\infty$.
3. Quantiles of the averaged predictive distribution are then obtained as:

$$\tilde{q}_\tau = \min\{z : \tilde{F}(z) \geq \tau\}.$$

Averaging of quantiles. Horizontal averaging of m distributions is a simple average of the quantile forecasts at corresponding probabilities τ:

$$\tilde{q}_\tau = \frac{1}{m} \sum_{i=1}^{m} \hat{q}_\tau^{(i)}, \tag{9}$$

which is exactly what the *qaverage*() functions does. Note that, in contrast to *paverage*(), it requires the input distributions to be tabulated at the same quantile levels.

2.3.2. Forecast evaluation

To assess the accuracy of both point and probabilistic forecasts, the *PostForecasts.jl* package offers popular evaluation metrics for the *PointForecasts* and *QuantForecasts* objects. For point forecasts they include well-known statistical error measures [34, 35]:

- *mae*() – mean absolute error: MAE = $\frac{1}{n} \sum_{t=1}^{n} |\varepsilon_t|$,
- *mape*() – mean absolute percentage error: MAPE = $\frac{100}{n} \sum_{t=1}^{n} \frac{|\varepsilon_t|}{|y_t|}$,
- *smape*() – symmetric MAPE: sMAPE = $\frac{200}{n} \sum_{t=1}^{n} \frac{|\varepsilon_t|}{|\hat{y}_t|+|y_t|}$,
- *mse*() – mean squared error: MSE = $\frac{1}{n} \sum_{t=1}^{n} \varepsilon_t^2$,

where $\varepsilon_t = y_t - \hat{y}_t$. For probabilistic forecasts, the package offers three metrics:

- *coverage*() – unconditional coverage: UC = $\frac{1}{n} \sum_{t=1}^{n} \mathbb{1}_{\{y_t \leq \hat{q}_{\tau,t}\}}$, to evaluate the *reliability* (also called *calibration* or *unbiasedness*) [8, 36]; in the case of quantiles this corresponds to the number of observations not exceeding the quantile forecast, i.e., the closer the UC is to the nominal coverage, the more reliable are the forecasts;

- *pinball*() – pinball loss: $\mathrm{PL}(\tau) = \frac{1}{n}\sum_{t=1}^{n}\left\{(\mathbb{1}_{\hat{q}_{\tau,t}\geq y_t} - \tau)(\hat{q}_{\tau,t} - y_t)\right\}$, to assess both *reliability* and *sharpness* [5, 37]; the PL is a strictly proper scoring rule for quantiles, i.e., it is minimized when the quantile predictions are identical to the true quantiles, hence encouraging honest forecasting [38];
- *crps*() – continuous ranked probability score: $\mathrm{CRPS} \approx \frac{2}{k}\sum_{i=1}^{k}\mathrm{PL}\left(\frac{i}{k+1}\right)$, where k is the number of quantile predictions, corresponding to equidistant probability levels spanning the interval $\left[\frac{1}{k+1}, \frac{k}{k+1}\right]$, which is a single metric that evaluates the entire predictive distribution [1, 5, 39]; while the PL is a strictly proper scoring rule for individual quantiles, the CRPS is strictly proper for entire distributions.

In all of the above formulas, n corresponds to the number of predicted time steps, i.e., the length of the test period). For an illustrative example of using the CRPS to evaluate probabilistic forecasts see the snippet in Section Appendix C.1. To quantify the contributions of different forecasters to the predictions obtained by forecast averaging, *PostForecasts.jl* provides methods based on the concept of Shapley values, see Appendix B.

2.3.3. Sample datasets

The *PostForecasts.jl* package ships with two datasets. The first, called **EPEX**, is of hourly frequency and comprises five years (2019-2023) of wholesale electricity prices in Germany, as well as the corresponding day-ahead point forecasts computed by Lipiecki et al. [5] using a LASSO-Estimated AutoRegressive (LEAR) model [35, 40]. The regressors include historical prices of electricity (various lags), day-ahead predictions of the system-wide load and day-ahead predictions of wind and solar generation. The parameters are estimated separately for each of the 4 training window lengths, i.e., 56, 84, 1092 and 1456 most recent days, and employ cross-validation for selecting a regularization penalty, as discussed in [5]. For example, calling *loaddata(:epex20)*, loads the observations and forecasts of EPEX prices for the 20th trading hour of the day-ahead market (19:00).

The second dataset, called **PANGU**, contains forecasts from the PANGU weather model trained on 39 years of ERA5 reanalysis data [41]. The data spans five years (2018-2022) of forecasts of the following weather variables for the city of Wrocław, Poland: **U10** (u-component of wind speed at 10m), **V10** (v-component of wind speed at 10m), **T2M** (temperature at 2m), **T850** (temperature at 850 hPa), **Z500** (geopotential height at 500 hPa). The dataset is partitioned into 32 files, which correspond to different forecasting horizons (up to 186 hours ahead, with a 6-hour resolution, initialized each day at midnight). For example, calling *loaddata(:pangu24t850)* loads the observations and forecasts of temperature at 850 hPa with a lead time of 24 hours.

3. Illustrative examples

3.1. Loading and postprocessing point forecasts

In the first example we show how to load point forecasts from a delimited file and postprocess them using a selected model in*PostForecasts.jl*. Assume that the file named *myforecasts.csv* has the following structure:

```
time    real    predA   predB
1       110.8   118.7   116.0
2       18.0    114.1   109.2
3       71.9    82.7    75.0
...
```

To load and postprocess it we only need two function calls:

```
using PostForecasts
pf = loaddlm("myforecasts.csv", delim='\t', idcol=1, obscol
    =2, predcol = [3, 4], colnames=true)
qf = point2quant(pf, method=:qr, window=100, quantiles=[0.5,
    0.9])
```

First, we read the file with the *loaddlm*() function, which arguments specify that the file is tab delimited, the identifiers are stored in the first column, the observations in the second, and the predictions in the third and fourth. The last argument informs that the column names are present in the file, so the first row is not parsed into numeric values. Next, the *point2quant*() function postprocesses the forecasts stored in *pf*, computes quantile predictions and returns a *QuantForecasts* object, saved to *qf*. In the above snippet, the arguments of *point2quant*() specify that Quantile Regression Averaging (QRA; for other variants see Section Appendix C.2) is used for postprocessing, the length of the calibration window is 100 data points, and that we want to predict the median and the 90th percentile. By default, the postprocessing model is retrained before every prediction using a calibration window of most recent data points. For details on alternative configurations, see the documentation of the *point2quant*() function.

3.2. Probabilistic forecasts as a decision support tool

Consider an energy company that owns a battery and trades in the day-ahead market. Every morning it faces the decision about whether to submit a buy order to charge the battery and a sell order to discharge it at a later hour of the next day, or avoid trading due to adverse market conditions. In this example, we show how probabilistic forecasts can help us identify risky market conditions and prevent losses.

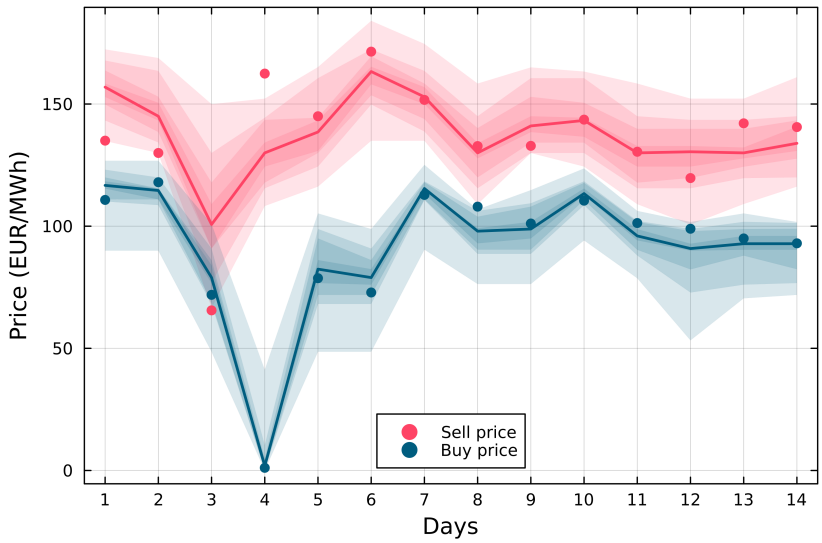

Figure 2: Median price forecasts (*solid lines*), prediction intervals (*shaded areas*) and observed prices (*dots*) for 3am (*blue*) and 7pm (*red*) during two sample weeks in April 2023. The lighter the color of the shaded area, the higher the confidence level of the corresponding prediction interval: 20% (→ *dark blue/red*), 40%, 60% and 80% (→ *light blue/red*).

For simplicity, we focus on two weeks in April 2023 and assume that buy orders are submitted for 3:00 while sell orders for 19:00. In the code snippet below we show how to postprocess point predictions from the EPEX dataset to obtain decile forecasts using the IDR (supporting code with plotting functions is presented in Appendix C.4).

```
using PostForecasts
"""
function plot_trades(qfbuy, qfsell) ...
"""
pfbuy = loaddata(Symbol(:epex, 4))  # forecasts for 3:00
pfsell = loaddata(Symbol(:epex, 20)) # forecasts for 19:00

# postprocessing point forecasts
qfbuy = point2quant(pfbuy, method=:idr, window=182, quantiles
    =9, start=20230408, stop=20230421)
qfsell = point2quant(pfsell, method=:idr, window=182,
    quantiles=9, start=20230408, stop=20230421)

plot_trades(qfbuy, qfsell) # plot intervals and observations
```

From the decile forecasts we can construct four prediction intervals (PI) centered around the median, i.e., the 5th decile, with confidence levels of 20%, 40%,

60% and 80%. For example, the 20%-PI is obtained by taking the 4th and the 6th deciles, while the 80%-PI by taking the 1st and the 9th. To visualize the results, we can plot the median price forecasts, the PIs and the observed prices for 3am and 7pm, Figure 2 presents the output of the *plot_trades*() function.

Clearly, on the third day the upper quantiles of prices for 3am significantly overlap the lower quantiles of prices for 7pm. This indicates that the buy price is quite likely to be higher than the sell price, so the trading strategy carries substantial risk. Indeed, the actual price at 7pm (→ red dot) was lower than at 3am (→ blue dot) for that day, so trading would lead to incurring a loss.

4. Impact and conclusions

Sometimes called the successor to Python, Julia stands out as a highly efficient tool for data science [42]. According to the recently released Programming Language Benchmark v2 (github.com/attractivechaos/plb2), it is up to two orders of magnitude faster than CPython (the reference implementation of Python) and about two to four times faster than PyPy (pypy.org). However, its ecosystem is much smaller and requires development before it reaches the level of Python. The *PostForecasts.jl* package fills this gap and offers a set of easy-to-use, versatile and robust computational tools to derive predictive distributions from point forecasts. To our knowledge, it is the first package in Julia to provide this functionality and such a wide range of general-purpose postprocessing methods, and probably the first open source package that allows to exploit the benefits of combining CP, IDR and QRA. For instance, *PPNN* in Python and *ensemblepp* in R are focused on postprocessing of ensemble weather forecasts. On the other hand, *ConformalPrediction.jl* in Julia and *Nixtla's* products in Python only provide CP-based postprocessing, *ProbCast* (R) and *ReModels* (Python) only QRA-based, while *EasyUQ* framework (Python) is based on IDR. However, as Lipiecki et al. [5] show, introducing diversity by combining IDR-generated predictive distributions with those of generally better performing CP and QRA significantly improves the forecast accuracy.

At the same time, the *PostForecasts.jl* package addresses another gap. By leveraging the developments in the point forecasting literature, it provides decision makers with much-needed tools to quantify risks, prepare for different scenarios, and predict future uncertainty [3, 43, 44], without the need to develop complex models that directly output predictive distributions. This is crucial since probabilistic forecasts not only are preferred by professionals – 93% participants of a wind power trading simulation opted for a probabilistic forecast, but their use also yields higher revenues – by 20% when comparing the medians of both groups in the same study [45]. Revenue increases of up to 20% were also reported for agents trading in day-ahead electricity markets that base their decisions

on probabilistic rather than point predictions [46, 47]. With probability forecasts increasingly being communicated to the public [2], e.g., ABC News FiveThirtyEight.com routinely reports them on issues related to politics, sports, health or economics, the package may turn out to be useful not only to professionals, but to all wanting to convey reliable information about future uncertainty.

On the research side, the methods implemented in *PostForecasts.jl* have already proven useful. As Lipiecki et al. [5] report, the performance of the combination of probabilistic forecasts generated by CP, IDR, and QRA (see Section 2.2) was superior to that of state-of-the-art Distributional Deep Neural Networks (DDNN) [47] over two 4.5-year test periods from the German and Spanish electricity markets. At the same time, it was twice faster to compute than the DDNN-JSU model, and ca. 100-200 times faster than a single run of the hyperparameter optimization routine for the DDNN model in Python. This shows that the developed package can be a powerful tool in various time series applications. It also demonstrates that the strength comes from the combination of different postprocessing schemes, a feature missing in other open access packages.

Acknowledgements

The study was partially supported by the National Science Center (NCN, Poland) through grant no. 2018/30/A/HS4/00444. We also thank Sebastian Lerch for preparing the PANGU dataset and Bartosz Uniejewski for generating LEAR forecasts for the EPEX dataset.

References

[1] T. Gneiting, M. Katzfuss, Probabilistic forecasting, Annual Review of Statistics and Its Application 1 (2014) 125–151.

[2] R. L. Winkler, Y. Grushka-Cockayne, K. C. Lichtendahl, V. R. R. Jose, Probability forecasts and their combination: A research perspective, Decision Analysis 16 (4) (2019) 239–260.

[3] S. Vannitsem, D. S. Wilks, J. W. Messner (Eds.), Statistical Postprocessing of Ensemble Forecasts, Elsevier, Amsterdam, NL, 2018.

[4] J. Chen, T. Janke, F. Steinke, S. Lerch, Generative machine learning methods for multivariate ensemble postprocessing, Annals of Applied Statistics 18 (1) (2024) 159–183.

[5] A. Lipiecki, B. Uniejewski, R. Weron, Postprocessing of point predictions for probabilistic forecasting of day-ahead electricity prices: The benefits of using isotonic distributional regression, Energy Economics 139 (2024) 107934.

[6] D. Hendricks, Evaluation of Value-at-Risk models using historical data, Economic Policy Review 2 (1) (1996) 39–69.

[7] C. Alexander, Market Risk Analysis IV: Value at Risk Models, Wiley, 2008.

[8] J. Nowotarski, R. Weron, Recent advances in electricity price forecasting: A review of probabilistic forecasting, Renewable and Sustainable Energy Reviews 81 (1) (2018) 1548–1568.

[9] G. Shafer, V. Vovk, A tutorial on conformal prediction, Journal of Machine Learning Research 9 (2008) 371–421.

[10] C. Kath, F. Ziel, Conformal prediction interval estimation and applications to day-ahead and intraday power markets, International Journal of Forecasting 37 (2) (2021) 777–799.

[11] J. Nowotarski, R. Weron, Computing electricity spot price prediction intervals using quantile regression and forecast averaging, Computational Statistics 30 (3) (2015) 791–803.

[12] B. Liu, J. Nowotarski, T. Hong, R. Weron, Probabilistic load forecasting via Quantile Regression Averaging on sister forecasts, IEEE Transactions on Smart Grid 8 (2) (2017) 730–737.

[13] K. Maciejowska, T. Serafin, B. Uniejewski, Probabilistic forecasting with a hybrid Factor-QRA approach: Application to electricity trading, Electric Power Systems Research 234 (2024) 110541.

[14] G. Zakrzewski, K. Skonieczka, M. Małkiński, J. Mańdziuk, ReModels: Quantile Regression Averaging models, SoftwareX 28 (2024) 101905.

[15] A. Henzi, J. F. Ziegel, T. Gneiting, Isotonic distributional regression, Journal of the Royal Statistical Society. Series B: Statistical Methodology 83 (5) (2021) 963–993.

[16] A. Henzi, A. Mösching, L. Dümbgen, Accelerating the pool-adjacent-violators algorithm for isotonic distributional regression, Methodology and Computing in Applied Probability 24 (4) (2022) 2633–2645.

[17] E.-M. Walz, A. Henzi, J. Ziegel, T. Gneiting, Easy uncertainty quantification (EasyUQ): Generating predictive distributions from single-valued model output, SIAM Review 66 (1) (2024) 91–122.

[18] V. Vovk, A. Gammerman, G. Shafer, Algorithmic learning in a random world, Springer Science & Business Media, 2005.

[19] H. Papadopoulos, K. Proedrou, V. Vovk, A. Gammerman, Inductive confidence machines for regression, Lecture Notes in Computer Science 2430 (2002) 345–356.

[20] M. Zaffran, O. Féron, Y. Goude, J. Josse, A. Dieuleveut, Adaptive conformal predictions for time series, Proceedings of Machine Learning Research 162 (2022) 25834–25866.

[21] Y. Kato, D. M. Tax, M. Loog, A review of nonconformity measures for conformal prediction in regression, Proceedings of Machine Learning Research 204 (2023) 369–383.

[22] R. Weron, Electricity price forecasting: A review of the state-of-the-art with a look into the future, International Journal of Forecasting 30 (4) (2014) 1030–1081.

[23] K. Maciejowska, J. Nowotarski, A hybrid model for GEFCom2014 probabilistic electricity price forecasting, International Journal of Forecasting 32 (3) (2016) 1051–1056.

[24] Y. Wang, N. Zhang, Y. Tan, T. Hong, D. Kirschen, C. Kang, Combining probabilistic load forecasts, IEEE Transactions on Smart Grid 10 (4) (2019) 3664–3674.

[25] B. Uniejewski, Enhancing accuracy of probabilistic electricity price forecasting: A comparative study of novel quantile regression averaging generalization, in: 19th International Conference on the European Energy Market (EEM), 2023, doi: 10.1109/EEM58374.2023.10161748.

[26] D. Yang, G. Yang, B. Liu, Combining quantiles of calibrated solar forecasts from ensemble numerical weather prediction, Renewable Energy 215 (2023) 118993.

[27] C. Cornell, N. T. Dinh, S. A. Pourmousavi, A probabilistic forecast methodology for volatile electricity prices in the Australian National Electricity Market, International Journal of Forecasting 40 (4) (2024) 1421–1437.

[28] R. Koenker, Quantile regression: 40 years on, Annual Review of Economics 9 (2017) 155–176.

[29] G. Marcjasz, B. Uniejewski, R. Weron, Probabilistic electricity price forecasting with NARX networks: Combine point or probabilistic forecasts?, International Journal of Forecasting 36 (2) (2020) 466–479.

[30] B. Uniejewski, G. Marcjasz, R. Weron, On the importance of the long-term seasonal component in day-ahead electricity price forecasting: Part II – Probabilistic forecasting, Energy Economics 79 (2019) 171–182.

[31] B. Uniejewski, Smoothing quantile regression averaging: A new approach to probabilistic forecasting of electricity prices (2023). arXiv:2302.00411.

[32] X. Wang, R. J. Hyndman, F. Li, Y. Kang, Forecast combinations: An over 50-year review, International Journal of Forecasting 39 (4) (2023) 1518–1547.

[33] K. C. Lichtendahl, Y. Grushka-Cockayne, R. L. Winkler, Is it better to average probabilities or quantiles?, Management Science 59 (7) (2013) 1594–1611.

[34] R. Hyndman, A. Koehler, Another look at measures of forecast accuracy, International Journal of Forecasting 22 (4) (2006) 679–688.

[35] J. Lago, G. Marcjasz, B. De Schutter, R. Weron, Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark, Applied Energy 293 (2021) 116983.

[36] P. H. Kupiec, Techniques for verifying the accuracy of risk measurement models, The Journal of Derivatives 3 (2) (1995) 73–84.

[37] T. Gneiting, D. Wolffram, J. Resin, K. Kraus, J. Bracher, T. Dimitriadis, V. Hagenmeyer, A. I. Jordan, S. Lerch, K. Phipps, M. Schienle, Model diagnostics and forecast evaluation for quantiles, Annual Review of Statistics and Its Application 10 (2023) 597–621.

[38] T. Gneiting, A. Raftery, Strictly proper scoring rules, prediction, and estimation, Journal of the American Statistical Association 102 (477) (2007) 359–378.

[39] J. Berrisch, F. Ziel, CRPS learning, Journal of Econometrics 237 (2) (2023) 105221.

[40] B. Uniejewski, Regularization for electricity price forecasting, Operations Research and Decisions 34 (3) (2024) 267–286.

[41] C. Bülte, N. Horat, J. Quinting, S. Lerch, Uncertainty quantification for data-driven weather models, Artificial Intelligence for the Earth Systems (2025).

[42] B. Kamiński, Julia for Data Analysis, Manning Publications, 2022.

[43] K. Maciejowska, Portfolio management of a small RES utility with a structural vector autoregressive model of electricity markets in Germany, Operations Research and Decisions 32 (4) (2022) 75–90.

[44] T. Gneiting, S. Lerch, B. Schulz, Probabilistic solar forecasting: Benchmarks, post-processing, verification, Solar Energy 252 (2023) 72–80.

[45] C. Möhrlen, R. Bessa, N. Fleischhut, A decision-making experiment under wind power forecast uncertainty, Meteorological Applications 29 (3) (2022) e2077.

[46] B. Uniejewski, R. Weron, Regularized quantile regression averaging for probabilistic electricity price forecasting, Energy Economics 95 (2021) 105121.

[47] G. Marcjasz, M. Narajewski, R. Weron, F. Ziel, Distributional neural networks for electricity price forecasting, Energy Economics 125 (2023) 106843.

[48] Y. Romano, E. Patterson, E. J. Candès, Conformalized quantile regression, in: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA, 2019.

[49] L. S. Shapley, Notes on the n-person game – II: The value of an n-person game, RAND Research Memorandum (1951).

[50] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, S.-I. Lee, From local explanations to global understanding with explainable AI for trees, Nature Machine Intelligence 2 (1) (2020) 56–67.

[51] I. C. Covert, S. Lundberg, S.-I. Lee, Understanding global feature contributions with additive importance measures, in: Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20, Curran Associates Inc., Red Hook, NY, USA, 2020.

[52] S. M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 4768–4777.

[53] C. Wan, Z. Xu, J. Østergaard, Z. Y. Dong, K. P. Wong, Discussion of "combined nonparametric prediction intervals for wind power generation", IEEE Transactions on Sustainable Energy 5 (3) (2014) 1021–1021.

Appendix A. Conformalizing quantile forecasts

The package also offers postprocessing of predictive distributions by conformalizing quantile forecasts, a method initially introduced to improve the coverage of prediction intervals derived from quantile regression[48]. The same idea can be employed to conformalize individual quantile forecasts instead of prediction intervals. Given the forecasts $\hat{q}_{\tau,t}$ from a prediction model for a τ-quantile, calculate the non-conformity scores defined as

$$\lambda_t = y_t - \hat{q}_{\tau,t}. \tag{A.1}$$

Then, the conformalized quantile prediction is given by:

$$\tilde{q}_{\tau,t} = \hat{q}_{\tau,t} + Q_{1-\tau}(\lambda), \tag{A.2}$$

where $Q_{1-\tau}(\lambda)$ is the $(1-\tau)$-th quantile of nonformity score λ_t for $t \in \mathcal{S}$.

Call *conformalize*() or its in-place counterpart *conformalize!*() to use this functionality on a *QuantForecasts* object. See Section Appendix C.3 for an example usage.

Appendix B. Shapley values and forecaster contributions

When averaging multiple predictions, the question of what each forecaster brings to the table arises. To answer it, we can use the concept of Shapley values [49]. They were originally developed to fairly distribute total wins ($\rightarrow$ predictive power) among players ($\rightarrow$ ensemble components) in a cooperative game based on their individual contributions. In our approach, we consider a coalition game $v_x(\mathcal{M})$, defined as

$$v_x(\mathcal{M}) = -L\Big(\mathrm{Ave}_x(\mathcal{M}), x\Big), \tag{B.1}$$

where $\mathcal{M}$ is a non-empty subset of players ($\rightarrow$ $m \geq 1$ forecasters), L is a loss function and $\mathrm{Ave}_x(\mathcal{M})$ is the prediction of x obtained by averaging forecasts from $\mathcal{M}$. Shapley values for $v_x(\mathcal{M})$ are analogous to the Loss SHapley Additive exPlanations (LossSHAP) [50] of model $\mathrm{Ave}_x(\mathcal{M})$, while their mean over the test period is a counterpart of the Shapley Additive Global importancE (SAGE) [51]. Note that we consider simple averaging methods for which marginal contributions can be calculated directly, without resorting to approximation algorithms required by the popular in the machine learning literature SHapley Additive exPlanations (SHAP) [52], as well as by LossSHAP and SAGE.

For the game $v_x(\mathcal{M})$ and a set of $\mathcal{N}$ players ($\rightarrow$ forecasters), Shapley value ϕ_i of player i is usually defined as [49]:

$$\phi_i = \frac{1}{|\mathcal{N}|} \sum_{\mathcal{M} \in \mathcal{P}(\mathcal{N}\setminus\{i\})} \binom{|\mathcal{N}|-1}{|\mathcal{M}|}^{-1} \left[v_x(\mathcal{M}\cup\{i\}) - v_x(\mathcal{M})\right]. \tag{B.2}$$

The sum above extends over the entire power set $\mathcal{P}(\mathcal{N}\setminus\{i\})$, i.e., the set of all subsets of $\mathcal{N}$ less forecaster i, but including the empty set $\varnothing$. Since when averaging forecasters there is no reasonable interpretation for the empty set, we omit the empty coalition and define *Shapley contributions* as:

$$\Phi_i = \frac{1}{|\mathcal{N}|} \sum_{\mathcal{M} \in \mathcal{P}(\mathcal{N}\setminus\{i\})\setminus\varnothing} \binom{|\mathcal{N}|-1}{|\mathcal{M}|}^{-1} \left[v_x(\mathcal{M}\cup\{i\}) - v_x(\mathcal{M})\right], \tag{B.3}$$

which are related to Shapley values through the relation:

$$\Phi_i = \phi_i - v_x(\{i\}) + v_x(\varnothing). \tag{B.4}$$

While the standard Shapley values sum up to value of the grand coalition $v_x(\mathcal{N})$ ($\rightarrow$ the accuracy of the ensemble average), Shapley contributions sum up to the accuracy gained from averaging:

$$\sum_{i\in\mathcal{N}} \Phi_i = v_x(\mathcal{N}) - \frac{1}{|\mathcal{N}|} \sum_{i\in\mathcal{N}} v_x(\{i\}), \tag{B.5}$$

i.e., the difference between the accuracy of the ensemble average and the average accuracy of individual ensemble components. The

Although the Φ_i's defined by Eq. (B.3) differ from standard Shapley values and discount the standalone value of players, they remain fair allocations, in the sense that $\Phi_i > \Phi_j$ if and only if $v_x(\{i\}) > v_x(\{j\})$ for $\mathcal{N} = \{i, j\}$. Furthermore, the properties of symmetry, linearity and null-player also hold for the Φ_i's.

The *PostForecasts.jl* package provides the *shapley*() function that calculates Shapley values for point or probabilistic forecasts using an arbitrary averaging method and payoff function. The following snippet calculates MAE-based ensemble contributions for a pool of four point forecasts:

```
using PostForecasts
pf = loaddata(:epex12)
players = decouple(pf)
coalition(X) = average(X)
payoff(x) = -mae(x)[1]
contributions = shapley(players, coalition, payoff)
```

Note that the payoff used in this example is the negative of the MAE (as in Eq. (B.1)), which means that we reward the players for minimizing the loss function.

Appendix C. Additional examples

Appendix C.1. Probabilistic forecasting of day-ahead electricity prices

In this example we show how to compute probabilistic forecasts of day-ahead electricity prices from point forecasts stored in the EPEX dataset (see Section 2.3.3) for all hours of the year 2023, using three different postprocessing schemes – IDR, CP and QRA. See Lipiecki et al. [5] for more details on this forecasting task.

The code snippet below first creates a dictionary *qf* that for each key (corresponding to the postprocessing method) will store a vector of 24 *QuantForecasts* objects. Then it iterates over the 24 hours of the day, loads the point forecasts and, using each method, generates the probabilistic forecasts of 9 deciles (i.e. 10%, 20%, ..., 90% percentiles) for 2023:

```
using PostForecasts
methods = [:idr, :cp, :qr]
qf = Dict((m => Vector{QuantForecasts}(undef, 24) for m in
    methods)...)
for h in 1:24
    pf = loaddata("epex$(h)")
    for m in methods
        qf[m][h] = point2quant(pf, method=m, window=56,
    quantiles=9, start=20230101, stop=20231231)
```

```
    end
end
```

Probabilistic forecasts can then be combined, e.g., using vertical distribution averaging → function *paverage*(), see Eq. (8), and the CRPS of the individual and the combined predictive distributions can be compared:

```
qf[:ave] = Vector{QuantForecasts}(undef, 24)
for h in 1:24
    qf[:ave][h] = paverage([qf[m][h] for m in methods], quantiles=9)
end
println("Method \t| CRPS ")
println("-"^20)
for m in [methods..., :ave]
    println(uppercase(string(m)), " \t| CRPS: ", round(sum(crps.(qf[m]))/24, digits=3))
end
"""
Method   | CRPS
--------------------
IDR      | CRPS: 9.752
CP       | CRPS: 9.822
QR       | CRPS: 9.986
AVE      | CRPS: 9.248
"""
```

Looking at the output results, we can observe that averaging probabilistic forecasts obtained from three different methods leads to a significantly smaller CRPS. Combining forecasts is known to improve the accuracy, and often outperforms the single best model. For general considerations of forecast averaging, we refer the reader to Wang et al. [53], and for a more extensive analysis of combining IDR, CP and QRA in the context of electricity price forecasting to Lipiecki et al. [5]. Note that the computations can take some time. While IDR and CP are almost instantaneous, QRA is more time-consuming and can take up to a few minutes.

Appendix C.2. Variants of quantile regression

There are multiple approaches to applying quantile regression to a pool of point forecasts, here we compare four, which can be readily computed with the *PostForecasts.jl* package:

- Quantile Regression Averaging (QRA), where each point forecast is treated as a separate regressor in a multiple quantile regression [11, 22, 23].

- Quantile Regression Machine (QRM), where point forecasts are averaged and treated as a single regressor in a simple quantile regression [25, 29].
- Quantile Regression with averaging over probabilities (QRF), where each point forecast is treated as a regressor in a simple quantile regression and the output distributions of $m \geq 1$ quantile regressions are vertically (see Figure 1) averaged over probabilities [30, 31].
- Quantile Regression with averaging over quantiles (QRQ), where each point forecast is treated as a regressor in a simple quantile regression and the output distributions of $m \geq 1$ quantile regressions are horizontally (see Figure 1) averaged over quantiles [30].

The code below shows how to compute probabilistic forecasts of day-ahead electricity prices at 19:00 for the entire 2021 from point forecasts stored in the EPEX dataset, using the above four variants of quantile regression and a one-year training window. Once computed, the predictive distributions are compared using the CRPS:

```
using PostForecasts
pf = loaddata(:epex20)(20200101, 20211231)
qf = Dict()

qf["QRA"] = point2quant(pf, method=:qr, window=365, quantiles=9)
qf["QRM"] = point2quant(average(pf), method=:qr, window=365, quantiles=9)
qf["QRF"] = paverage(point2quant.(decouple(pf), method=:qr, window=365, quantiles=9))
qf["QRQ"] = qaverage(point2quant.(decouple(pf), method=:qr, window=365, quantiles=9))

println("Method \t| CRPS ")
println("-"^20)
for method in ["QRA", "QRM", "QRF", "QRQ"]
    println(method, "\t| ", round(crps(qf[method]), digits=3))
end
"""
Method   | CRPS
--------------------
QRA      | 10.464
QRM      | 10.229
QRF      | 10.308
QRQ      | 10.285
"""
```

Note how the input forecasts change when passed on to the *point2quant*() function. For the default QRA method the input is simply the *pf* object, for the QRM it is the averaged point forecast → *average*(*pf*), while for the QRF and QRQ the inputs are vectors of point forecasts → *decouple*(*pf*). The latter are later averaged vertically → *paverage*() or horizontally → *qaverage*().

While all of these approaches use the same information set and aim to forecast the same target variable, the differences in processing information lead to different forecasting accuracy. The performance of each method may differ based on the accuracy of input point forecasts, the size of the forecast pool, the length of the training window and the dataset.

Appendix C.3. Conformalizing weather predictions

In this example we show how to conformalize quantile forecasts to improve the coverage of predictive distributions of weather variables from the PANGU dataset postprocessed using IDR, and visualize the results using the *Plots* package.

The script presented below first selects the variable to forecast and the lead time of point predictions. Then using the function *point2quant*() computes the quantile forecasts *qf* for 9 deciles using the IDR with a training window of 364 days and calculates the miscoverage, i.e., the difference between nominal and empirical coverage. Then, it conformalizes *qf* using a 182-day training window and computes the miscoverage of conformalized quantiles. See Section Appendix C.3 for details.

```
using PostForecasts, Plots

variable = :u10 # u10, c10, t2m, t850 or z500
leadtime = 24 # between 0 and 186, divisible by 6

pf = loaddata(Symbol(:pangu, leadtime, variable))

qf = point2quant(pf, method=:idr, window=364, quantiles=9)
miscoverage_idr = (coverage(qf) - getprob(qf)).*100
conformalize!(qf, window=182)
miscoverage_cidr = (coverage(qf) - getprob(qf)).*100
```

Note that the in-place function *conformalize!*() will leave the first 182 unmodified predictions in *qf*, ensuring that the we compare the results on the same time period. Figure C.3 presents the comparison of *miscoverage_idr* and *miscoverag_cidr*.

Appendix C.4. Plotting function for the 'Probabilistic forecasts as a decision support tool' example

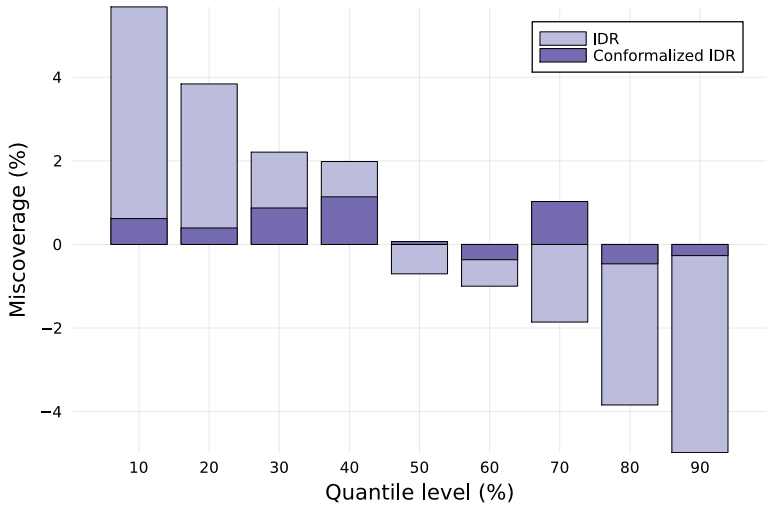

Figure C.3: A comparison of the miscoverage, i.e., the difference between nominal and empirical coverage, for IDR-implied quantile forecasts before and after conformalization. Conformalizing the forecasts shrinks the miscoverage towards zero, which corresponds to improving the reliability – the empirical coverage of the quantile is closer to the nominal one.

```julia
using PostForecasts, Plots
theme(:default, palette = theme_palette(:dark))

function plot_obs!(plt, fs::Forecasts; kwargs...)
    kwargs=Dict{Symbol, Any}(kwargs)
    kwargs[:st] = haskey(kwargs, :st) ? kwargs[:st] : :scatter
    kwargs[:msw] = haskey(kwargs, :msw) ? kwargs[:msw] : 0
    kwargs[:label] = haskey(kwargs, :label) ? kwargs[:label] : nothing
    plot!(plt, viewobs(fs); kwargs...)
end

function plot_quantile!(plt, qf::QuantForecasts, quantile::Integer; kwargs...)
    kwargs=Dict{Symbol, Any}(kwargs)
    kwargs[:lw] = haskey(kwargs, :lw) ? kwargs[:lw] : 2
    kwargs[:label] = haskey(kwargs, :label) ? kwargs[:label] : nothing
    plot!(plt, viewpred(qf, eachindex(qf), quantile); kwargs...)
end

function plot_intervals!(plt, qf::QuantForecasts; kwargs...)
    kwargs=Dict{Symbol, Any}(kwargs)
    kwargs[:lw] = haskey(kwargs, :lw) ? kwargs[:lw] : 0.0
```

```
    kwargs[:fa] = haskey(kwargs, :fa) ? kwargs[:fa] : 0.15
    kwargs[:label] = haskey(kwargs, :label) ? kwargs[:label] : nothing
    if npred(qf) % 2 == 0
        for i in 1:Int(npred(qf)/2)
            plot!(plt, viewpred(qf, eachindex(qf), i), fillrange=viewpred(qf, eachindex(qf), npred(qf)-i+1); kwargs...)
        end
    else
        central_quantile = Int((npred(qf)-1)/2) + 1
        for i in 1:(central_quantile-1)
            plot!(plt, viewpred(qf, eachindex(qf), central_quantile-i), fillrange=viewpred(qf, eachindex(qf), central_quantile+i); lw=0, kwargs...)
        end
    end
end

function plot_trades(qfbuy, qfsell)
    plt = plot(legend=:bottom, xlabel="Days", ylabel="Price (EUR/MWh)", xticks=1:14, framestyle=:box)
    plot_intervals!(plt, qfsell, color=1)
    plot_intervals!(plt, qfbuy, color=4)
    plot_quantile!(plt, qfsell, 5, color=1)
    plot_quantile!(plt, qfbuy, 5, color=4)
    plot_obs!(plt, qfsell, color=1, label="Sell price")
    plot_obs!(plt, qfbuy, color=4, label="Buy price")
    return plt
end
```