1 Quantile Spread Adjustment

The general idea behind Quantile Spread Adjustment (QSA) is to adjust the spreads of each forecasted quantile by some factor. Quantile spreads are defined as the distance between the respective quantile and some basis. Three different points in the forecasting spectrum, intuitively make sensible basis: the median, the next inner neighbor and the symmetric interval quantile. The quantile spread for these different bases are illustrated in Figure 1.

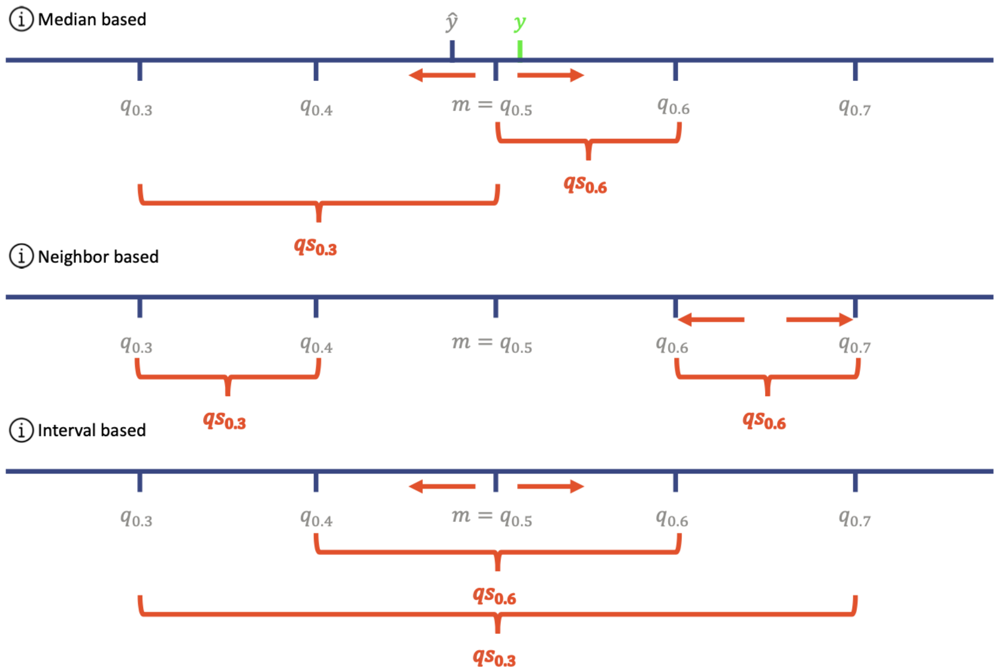

Figure 1: Quantile Spreads for different Basis

We choose the median based definition of the quantile spreads for two main reasons. First, in contrast to the neighborhood based definition, the median basis has the advantage that different quantile spreads are independent of each other. This property makes finding the optimal quantile spread adjustments for a large set of quantiles much simpler. It comes at the cost, however, that adjustments might lead to *quantile crossing*[1], which would not be the case for neighborhood based adjustments. Our second reason to use the median basis is that, unlike the interval based approach, it does not restrict adjustments for quantile pairs to be symmetric.

1.1 Theory

Using the median based definition, the next step is to determine how to optimally adjust the quantile spreads. As target function, QSA uses the Weighted Interval Score from **??**. Equation 1 shows how the QSA weights $\mathbf{w}$ influence the WIS, where p denotes the number of confidence intervals, α_p the certainty level of a confidence interval and n the number of observations.

[1] As discussed in **??** `update_predictions()` automatically takes care of quantile crossing by reordering the output predictions in increasing quantile order

$$
\begin{aligned}
\mathbf{w}^* &= \underset{\mathbf{w}\in\mathbb{R}^p}{\arg\min}\, WIS(\mathbf{y}) \\
&= \underset{\mathbf{w}\in\mathbb{R}^p}{\arg\min} \sum_{i=1}^{p} \frac{\alpha_i}{2} \sum_{j=1}^{n} (u^*_{i,j} - l^*_{i,j}) + \frac{2}{\alpha_i} \cdot (l^*_{i,j} - y_j) \cdot \mathbf{1}(y_j \le l^*_{i,j}) + \frac{2}{\alpha_i} \cdot (y_j - u^*_{i,j}) \cdot \mathbf{1}(y_j \ge u^*_{i,j}) \quad (1) \\
\text{s.t.} \quad & l^*_{i,j} = l_{i,j} + (l_{i,j} - m) \cdot (w^l_i - 1) \quad \text{and} \quad u^*_{i,j} = u_{i,j} + (u_{i,j} - m) \cdot (w^u_i - 1)
\end{aligned}
$$

By varying the QSA factor w^l_i for the lower and w^u_i for the upper bound of a given prediction interval level α_i, QSA moves the quantiles from their original $l_{i,j}$ and $u_{i,j}$ to their updated values $l^*_{i,j}$ and $u^*_{i,j}$. QSA factors larger than 1 lead to an *increased* prediction interval, thus $w^l_i > 1$ reduces the value of $l^*_{i,j}$ and $w^u_i > 1$ increases the value of $u^*_{i,j}$.

These changes have two effects: On the one side an increase in w^l_i and w^u_i reduces the sharpness and increases the WIS, on the other side the increased interval may capture more observation which reduces the under- and overprediction penalties in the WIS. Thus depending on the positions of the observed values and predicted quantiles, QSA will either increase or decrease the interval size in order to minimize the WIS.

The `postforecasts` package implements the QSA optimization in three flavors that differ in the restriction of the weight vector $\mathbf{w}$: `qsa_uniform`, `qsa_flexible_symmetric` and `qsa_flexible`. These are listed in Equation 2.

$$
\begin{aligned}
\text{uniform}: w_i &= c \quad i \in [0, 1, \ldots, p-1, p], \quad c \in \mathbb{R} \\
\text{flexibel\_symmetric}: w_i &= w_{p-i} \quad i \in [0, 1, \ldots, \frac{p}{2} - 1], \quad w_i \in \mathbb{R} \qquad (2) \\
\text{flexible}: \mathbf{w} &\in \mathbb{R}^{'}
\end{aligned}
$$

`qsa_uniform` restricts all weight vector values to be identical. `qsa_flexible_symmetric` only restricts pairwise adjustments to be identical. It essentially represents unrestricted QSA with interval based adjustments. Finally, `qsa_flexible` is completely unrestricted as each quantile is adjusted separately.

In addition to different flavors, the `postforecasts` package also provides the option to regularize the optimization. Equation 3 displays the penalization term that is added to the WIS. It is designed to penalize differences between weight vector values by adding a factor proportional to the sum of squared deviations of the weight vector values from there mean. It therefore regularizes towards the `qsa_uniform` method and only has an effect for the `qsa_flexible_symmetric` and `qsa_flexible` flavors.

$$
\begin{aligned}
\mathbf{w}^* &= \underset{\mathbf{w}\in\mathbb{R}^p}{\arg\min}\ WIS_\alpha(\mathbf{y}) + r \cdot Pen(\mathbf{w}), \quad Pen(\mathbf{w}) = \sum_{i=1}^{p} (w_i - \bar{w})^2 \\
\text{s.t.} \quad \bar{w} &= \frac{1}{p} \sum_{i=1}^{p} w_i
\end{aligned} \qquad (3)
$$

1.2 Optimization

Underneath the hood, `postforecasts` accesses the `optim()` function from the stats[2] package. Out of the available optimization methods, `BFGS` (Fletcher 2013) and `L-BFGS-B` (Byrd et al. 1995) turned out to be the most reliable for QSA.

`BFGS` is named after Broyden, Fletcher, Goldfarb and Shanno and a quasi-Newton method. `L-BFGS-B` is a limited memory version of `BFGS` and additionally supports box constraints. As default value we set

[2] https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/optim

the optimization method to L-BFGS-B as it converges faster than BFGS in our data set, due to its limited memory property. The time gain is especially important for the qsa_flexible_symmetric and qsa_flexible methods which take considerably longer than qsa_uniform due to the large number of quantiles. Furthermore L-BFGS-B also has the advantage that we can lower bound the quantile spread factor to zero and thereby prevent quantile crossing with the median.

The optimization method can be accessed in the update_predictions() function by means of the optim_method argument. For L-BFGS-B, the lower and upper bound box constraints can be set with the arguments lower_bound_optim and upper_bound_optim.

Besides the use of optim(), postforecasts also provides a line search optimization which is used by setting the optim_method to line_search. As the run time increases exponentially with the parameter space, this method is currently restricted to the qsa_uniform method. It runs QSA for all values of the QSA factor within a sequence. This sequence is defined by its upper and lower values set with the arguments lower_bound_optim and upper_bound_optim as well as its step size set by steps_optim.

Regarding the QSA optimization functions shape, there is a potential issue: Due to the trade-off between sharpness and coverage defining the WIS, it might happen that an interval of QSA factors result in the same score. In other words, the WIS loss function has *plateaus*. This becomes less likely the more observations and quantiles are available, nevertheless it still has to be kept in mind.

The line_search optimization handles multiple optima by choosing the value closest to 1, hence the smallest possible adjustment of the quantiles. In essence, this is a regularization. For the BFGS and L-BFGS-B algorithms plateaus mean that both methods can converge to different optima while attaining the same WIS. In a future version of the package we aim to tackle this by adding a line search after the use of BFGS and L-BFGS-B in order to find the optima closest to 1 and thereby regularize the results.

Furthermore, due to the long computation times, particularly for large data sets and small initial training periods, the postforecasts package provides the option to run QSA *in parallel*. The parallelization is implemented with the foreach[3] package (Revolution Analytics and Weston, n.d.). It can be activated by defining a parallel processing environment and then specifiying parallel = TRUE within the update_predictions() function. A parallel processing environment is defined by setting the number of cores available with the registerDoParallel() function from the doParallel[4] package (Corporation and Weston 2022).

The following commands are valid for a device with two available cores:

```
library(postforecasts)
library(doParallel)
registerDoParallel(cores = 2)

df_updated_parallel <- update_predictions(
  df,
  methods = "qsa_flexible",
  optim_method = "L-BFGS-B",
  lower_bound_optim = 0,
  upper_bound_optim = 5,
  parallel = TRUE,
  verbose = TRUE
)
```

To observe the progress of the computations users can set the verbose = TRUE in order to print the logging output that the foreach() function provides.

Byrd, Richard H, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. 1995. "A Limited Memory Algorithm for Bound Constrained Optimization." *SIAM Journal on Scientific Computing* 16 (5): 1190–1208.

Corporation, Microsoft, and Steve Weston. 2022. *doParallel: Foreach Parallel Adaptor for the Parallel Package*. https://github.com/RevolutionAnalytics/doparallel.

[3] https://www.rdocumentation.org/packages/foreach/versions/1.5.2
[4] https://www.rdocumentation.org/packages/doParallel/versions/1.0.16

Fletcher, Roger. 2013. *Practical Methods of Optimization.* John Wiley & Sons.
Revolution Analytics, and Steve Weston. n.d. *Foreach: Provides Foreach Looping Construct.*