Forecasting with flowz

Giancarlo Vercellino

1 Inputs and a complete first forecast

The package models a joint distribution of future levels of a regularly observed univariate series. Supply a finite numeric vector, ordered from oldest to newest observation. A univariate `ts` is accepted, but its calendar attributes are not carried into the output. One forecast horizon means one observation interval. Resolve missing data before fitting; dropping rows can change the meaning of that interval.

The following deterministic series needs no external data. These small computational settings make the vignette fast to rebuild; they are not accuracy recommendations for a production application.

```
> t <- seq_len(260)
> x <- cumsum(sin(t / 4) + cos(t / 11))
```

```
> fit <- flowz(
+   x, horizon = 3, transform = "none",
+   baseline = "ewma", baseline_control = list(), autotune = TRUE,
+   drift_lambda = 0.90, vol_lambda = 0.94,
+   min_train = NULL, innovation_lags = 8,
+   n_hidden = 10, lambda = 1e-3, flow_aug = 1,
+   n_steps = 3, n_sims = 30,
+   cal_fraction = 0.20, test_fraction = 0.20,
+   cal_sims = 30, test_sims = 30,
+   spectral_modes = NULL, spectral_spar = 0.65,
+   spectral_shrink_to_one = 0.10,
+   spectral_tau_bounds = c(0.15, 2.50),
+   distribution_bw_factor = 0.50,
+   seed = 123, n_origins = NULL,
+   parallel = FALSE, n_cores = NULL,
+   plot = FALSE, verbose = FALSE
+ )
> fit$forecast$calibrated$summary
```

```
  horizon      mean    median        sd       q05       q25       q75
1       1 -4.027398 -4.093338 0.5496961 -4.737544 -4.457808 -3.707470
2       2 -3.672775 -3.445313 1.0804119 -5.468959 -3.919701 -3.010462
3       3 -2.495481 -2.562268 1.7855710 -5.452585 -3.297145 -1.546502
        q95
1 -3.095579
2 -2.366875
3  0.326547
```

```
> dim(fit$forecast$calibrated$samples)
```

```
[1] 30  3
```

Each row of the sample matrix is one simulated future path; columns represent horizons. The primary model is `calibrated`, after spectral calibration. The `uncalibrated` model contains paths before calibration, and `baseline` contains the selected baseline forecast. Summary quantiles describe the empirical simulated paths. They can differ slightly from quantiles returned by the smooth marginal distribution functions.

2 A consistent output structure

The fitted object has five visible sections: `forecast`, `evaluation`, `calibration`, `model`, and `metadata`. The forecast contains three model components: `calibrated`, `baseline`, and `uncalibrated`. Every model component uses the same structure:

summary One data-frame row per horizon; means, medians, standard deviations, and quantiles are on the original data scale.

samples A path matrix whose columns are named `h1`, `h2`, and so on.

pred_fun Marginal functions grouped as `dfun`, `pfun`, `qfun`, and `rfun` horizon lists.

```
> fc <- fit$forecast
> names(fc)

[1] "calibrated"   "baseline"     "uncalibrated"

> fc$calibrated$pred_fun$qfun$h3(c(.05, .5, .95))

[1] -6.1049265 -2.5769483  0.6144173

> fc$baseline$pred_fun$qfun$h3(c(.05, .5, .95))

[1] -7.834113 -5.012720 -2.191326

> fc$uncalibrated$pred_fun$qfun$h3(c(.05, .5, .95))

[1] -6.8913905 -2.6295340  0.9589341

> fit$evaluation$metadata$scale

[1] "original"
```

The `calibrated` group is the final calibrated forecast. The `baseline` group is the selected EWMA, ARIMA, ARFIMA, or structural model, and `uncalibrated` contains Flow marginals before spectral correction. Each model owns all four functions: `dfun`, `pfun`, `qfun`, and `rfun`. Evaluation belongs to the fitted model, not to an updated forecast whose future observations are unknown. Use `fit$evaluation` for scores, `fit$calibration` for spectral estimates, and `fit$metadata$diagnostics` for fitting details. There are no duplicated forecast aliases: select a model first, then its summary, paths, or marginal functions. See `?flowz_forecast` for the exact contract.

3 Choosing a baseline engine

The baseline supplies the causal mean trajectory and horizon-specific uncertainty used to define Flow residuals. Four engines are available:

ewma Local exponentially weighted drift and volatility; the default.

arima Nonseasonal ARIMA. Autotuning selects the order on the initial training window.

arfima Fractionally integrated ARMA for persistent, long-memory series.

structural A structural local-level or local-trend state-space model.

The selected specification is refitted using only observations available at each rolling origin. This prevents future observations from entering baseline estimation. EWMA states remain conditioning features for the Flow layer across all four choices, so only the baseline trajectory and scale change.

The following calls illustrate the interface; they are not run during the vignette build because rolling ARFIMA estimation is more costly.

```
> fit_arima <- flowz(x, baseline = "arima", autotune = TRUE)
> fit_arfima <- flowz(x, baseline = "arfima", autotune = TRUE)
> fit_structural <- flowz(x, baseline = "structural",
+   baseline_control = list(type = "trend"), autotune = FALSE)
> fit_arima$metadata$baseline_label
```

For a fixed ARIMA specification, use for example `baseline_control = list(order = c(1,1,1))` with `autotune = FALSE`. ARFIMA accepts `drange` and `estim`; the structural engine accepts `type = "level"` or `"trend"`. An isolated fitting failure is recorded in `fit$metadata$diagnostics$baseline` and falls back to Drift/EWMA.

4 Method and interpretation

Write $y_t = x_t$ for an untransformed fit and $y_t = \log(x_t)$ for a log fit. For the default EWMA engine, let $\Delta_t = y_t - y_{t-1}$. It tracks local drift and innovation variance using

$$\mu_t = \lambda_\mu \mu_{t-1} + (1 - \lambda_\mu)\Delta_t, \qquad e_t = \Delta_t - \mu_{t-1},$$
$$\sigma_t^2 = \lambda_\sigma \sigma_{t-1}^2 + (1 - \lambda_\sigma) e_t^2.$$

At origin i, its horizon-h baseline is $y_i + h\mu_i$, with scale $\sigma_i\sqrt{h}$. The other engines replace this trajectory and scale with their causal forecasts. Horizon corrections estimated from training origins standardize the multi-step level errors for every engine.

The model learns the joint error vector directly. Conditioning features include lagged standardized innovations, drift, volatility, and momentum. Random Gaussian starting points and training error vectors are interpolated linearly at random times. An extreme learning machine (ELM), with random tanh hidden features and ridge-fitted output weights, approximates the conditional velocity field. Fourth-order Runge–Kutta integration generates future residual paths, which are added to the baseline.

Calibration uses a discrete cosine transform (DCT) of path deviations around the Flow sample mean. Calibration-origin errors and simulated variation determine spectral dispersion multipliers. Smoothing, shrinkage towards one, and bounds regularize those multipliers. Unselected higher-frequency modes are left unchanged before the final bounds are applied. Calibration preserves the sample mean on the fitting scale; nonlinear back-transformation need not preserve the mean on the price scale.

There is a single fitted velocity field. The simulated uncertainty represents the learned residual distribution, without a bootstrap ensemble for parameter uncertainty. Coverage is empirical, not an exact finite-sample guarantee.

5 Evaluation and its limits

Eligible historical origins begin at `min_train` and end at `length(x) - horizon`. Automatic origin selection uses a heuristic with a cap of 500; `n_origins = Inf` requests all eligible origins. Selected origins are split chronologically into training, calibration, and test blocks. Calibration and test blocks each contain at least ten origins; at least twenty must remain for training. A small manually specified origin count can therefore fail even when the input series is long.

The current implementation does not enforce an embargo between forecast windows. Consecutive origins, including origins on opposite sides of a split, may have overlapping future observations. Chronological ordering alone does not make these scores independent holdout estimates. Inspect the spacing and reserve a separate future period when assessing generalization.

```
> o <- fit$metadata$diagnostics$origin_selection
> c(available = o$available_origins,
+   selected = o$selected_origins,
+   median_spacing = o$median_spacing)

     available       selected median_spacing
           180            180              1
```

```
> sc <- fit$calibration
> c(training = sc$train_origins,
+   calibration = sc$calibration_origins,
+   test = sc$test_origins)

   training calibration        test
        108          36          36

> fit$evaluation$metrics$aggregate

                 metric     value
1         RMSE_BASELINE 1.9511665
2         RMSE_FLOW_RAW 0.4769100
3         RMSE_FLOW_CAL 0.4769100
4          MAE_BASELINE 1.4830177
5          MAE_FLOW_RAW 0.3271070
6          MAE_FLOW_CAL 0.3271070
7  COVERAGE_90_BASELINE 0.6851852
8  COVERAGE_90_FLOW_RAW 0.9814815
9  COVERAGE_90_FLOW_CAL 0.9537037
10        CRPS_BASELINE 1.1099306
11        CRPS_FLOW_RAW 0.2976556
12        CRPS_FLOW_CAL 0.2819351
```

RMSE and MAE measure errors of point forecasts. CRPS assesses marginal distribution accuracy. Smaller values are preferable for all three. Empirical 90 percent coverage measures the fraction of observations inside the 5th–95th percentile interval; coverage alone does not measure how useful or narrow that interval is. Compare Flow against the selected baseline across horizons, not only by an aggregate score.

```
> head(fit$evaluation$metrics$per_horizon)

  horizon rmse_baseline rmse_flow_raw rmse_flow_cal mae_baseline
1       1     0.8364261     0.1721778     0.1721778    0.6952698
2       2     1.7602695     0.4634451     0.4634451    1.4625354
3       3     2.7609772     0.6617422     0.6617422    2.2912479
  mae_flow_raw mae_flow_cal coverage90_baseline coverage90_flow_raw
1    0.1286646    0.1286646           0.8888889           1.0000000
2    0.3416272    0.3416272           0.6111111           0.9444444
3    0.5110291    0.5110291           0.5555556           1.0000000
  coverage90_flow_cal crps_baseline crps_flow_raw crps_flow_cal
1           1.0000000     0.4847343     0.1421603     0.1320266
2           0.8888889     1.0761536     0.2725334     0.2690784
3           0.9722222     1.7689037     0.4782730     0.4447003
```

Changing settings repeatedly based on these test scores makes this block a model-selection set. Use a further untouched future period for final assessment. Serial dependence and overlapping horizons also affect the interpretation of pooled calibration diagnostics.

6 Marginal probabilities and joint events

The named horizon lists make marginal queries explicit:

```
> fun <- fit$forecast$calibrated$pred_fun
> fun$qfun$h3(c(0.05, 0.50, 0.95))

[1] -6.1049265 -2.5769483  0.6144173
```

```
> threshold <- fun$qfun$h3(0.5)
> fun$pfun$h3(threshold)

[1] 0.5

> fun$dfun$h3(threshold)

[1] 0.2280317

> set.seed(42)
> head(fun$rfun$h3(10))

[1] -2.6960699 -4.2907081 -3.2582917 -0.5680976 -3.4305698 -3.2762642
```

For horizon h, Flow uses a Gaussian-kernel mixture centered on simulated values. The bandwidth multiplier controls smoothing of these marginal functions, without altering stored paths or spectral calibration. The `dfun`, `pfun`, and `qfun` functions accept the usual `log`, `lower.tail`, and `log.p` arguments where applicable.

Use complete sample rows for joint events. Independent calls to `rfun$h1` and `rfun$h2` do not retain cross-horizon dependence. For example, estimate the probability that all three future levels exceed the last observed level:

```
> paths <- fit$forecast$calibrated$samples
> mean(rowSums(paths > tail(x, 1)) == ncol(paths))

[1] 0.8333333
```

With only 30 simulated paths this is a coarse Monte Carlo estimate. Increase the simulation budget for decisions that depend on tail probabilities.

7 Updating the forecast

Pass the full updated history. Prediction recomputes the local state and context, keeping the fitted horizon, velocity field, and calibration fixed. It does not retrain the model.

```
> t_new <- seq_len(265)
> x_new <- cumsum(sin(t_new / 4) + cos(t_new / 11))
> pr <- predict(fit, x_new, n_sims = 50, seed = 42)
> pr$forecast$calibrated$summary

  horizon      mean    median        sd       q05       q25        q75
1       1 -2.001089 -1.944356 0.5012420 -2.970937 -2.345762 -1.6910098
2       2 -1.826189 -1.719212 0.7277178 -2.865878 -2.268030 -1.3735854
3       3 -1.572723 -1.555238 1.4107058 -3.846127 -2.262575 -0.8416496
         q95
1 -1.2739637
2 -0.8946848
3  0.3567202

> pr$forecast$calibrated$pred_fun$qfun$h3(c(0.05, 0.95))

[1] -3.9613037  0.4580592
```

The generic `predict()` method defaults to 1000 paths. Specify `n_sims` explicitly for comparable runs. Refit when the learned residual dynamics or calibration may have changed.

8 Positive prices and log transformation

Use log transformation for strictly positive data when additive errors on the log scale are appropriate. This produces positive forecast paths and marginal quantiles under ordinary floating-point ranges.

```
> prices <- 100 * exp(x / 20)
> price_fit <- flowz(prices, horizon = 3, transform = "log",
+   n_hidden = 10, flow_aug = 1, n_steps = 3,
+   n_sims = 30, cal_sims = 30, test_sims = 30, parallel = FALSE)
> price_fit$forecast$calibrated$summary[, c("horizon", "mean", "q05", "q95")]
```

```
  horizon     mean      q05       q95
1       1 81.79089 78.90882  85.66055
2       2 83.33845 76.08314  88.84653
3       3 88.60766 76.18719 101.66385
```

```
> price_fit$forecast$calibrated$pred_fun$qfun$h3(c(0.05, 0.50, 0.95))
```

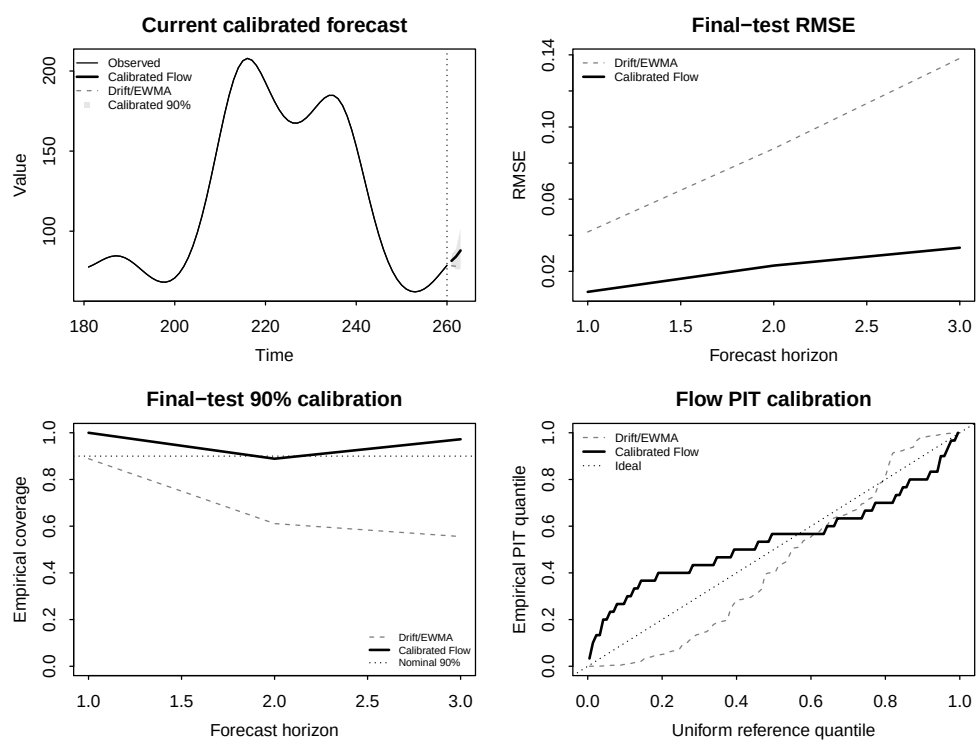
```
[1]  73.69418  87.91081 103.11976
```

```
> stopifnot(all(price_fit$forecast$calibrated$samples > 0))
```

Forecast summary means and standard deviations are calculated from the back-transformed samples; exponentiating a log mean does not give an arithmetic price mean. The baseline trajectory is exponentiated directly. Kernel bandwidths, fitted parameters, and evaluation diagnostics remain on the log scale. In particular, the test RMSE panel reports log-scale errors even though the current forecast panel displays price levels.

9 Reading the four-panel plot

```
> plot(price_fit)
```



Calibrated Flow is thick solid black; the selected baseline is dashed gray. The upper-left panel shows the observed history, current

median forecast, baseline trajectory, and empirical 90 percent band. The other panels compare test RMSE, empirical coverage against the dotted 90 percent reference, and PIT quantiles against the dotted uniform reference. The PIT panel pools horizons and test origins and may hide differences by horizon. A line near the diagonal is useful evidence, not proof of calibration.

Set `plot = TRUE` in `flowz()` to draw this display immediately. `plot(fit)` always uses the original fitted forecast and diagnostics. It does not display the result of a later `predict()` call. Currently prediction-time plotting is available for untransformed fits; for a log fit, draw updated predictions from the returned original-scale summary or samples.

10 Choosing computational settings

For a reproducible first run, keep the smoothing parameters fixed and use automatic origin selection. Increase one computational control at a time, holding the seed and evaluation design constant. Inspect both per-horizon scores and forecast bands after each change. For the EWMA engine, the two persistence parameters are fixed inputs, with defaults 0.90 and 0.94; larger values adapt more slowly. For ARIMA, ARFIMA, and structural engines, `autotune = TRUE` selects the specification on the initial training window. It does not tune the Flow hyperparameters.

Increase `n_hidden` to expand capacity, `flow_aug` to supply more random training interpolations, and `n_steps` to improve integration resolution. These increase computation and do not guarantee better forecasts. The ridge penalty regularizes the fitted output weights. Increase `cal_sims`, `test_sims`, and `n_sims` to reduce simulation noise in their respective stages. More spectral shrinkage gives a weaker dispersion correction; narrow bounds limit extreme corrections. Validate these choices using an appropriate external chronological design.

Parallel calibration and testing are enabled by default. Automatic worker selection uses at most two cores; an explicit `n_cores` can request more. Windows uses PSOCK workers and other platforms use forked processes. These vignette examples use serial execution to keep build overhead small. For small models, process startup and communication can cost more than they save. Use `fit$metadata$diagnostics$timing` to inspect elapsed stage times.

A fixed seed supports repeatable runs for the same data, settings, and software environment. Fitting and prediction advance R's random-number generator. When prediction uses `seed = NULL`, it continues from the current generator state; provide an explicit seed to compare repeated forecasts. Store the complete object using `saveRDS`; `readRDS` restores the fitted object and its internal prediction state. Consult `?flowz` and `?predict.flowz` for the full return structure and method signatures.

11 Common questions and troubleshooting

Too few origins. The split needs at least twenty training origins and at least ten in each later block. Start with `n_origins = NULL`. Longer horizons may require a longer input history.

Different quantiles. Summary quantiles describe stored paths. The marginal `qfun` functions use a smooth kernel mixture. Small differences are expected, especially with a small simulation budget.

Changing prediction results. Specify the same prediction seed and simulation count for a direct comparison.

Unexpected score units. With a log fit, evaluation scores remain on the log scale. Evaluate original-scale predictions separately when the decision requires price-unit errors.

Parallel overhead. Serial execution can be faster for small models. Compare stage timings with the same data and settings before increasing the worker count.