Post-Shock Forecasting with Donor Balancing Weights in R

Qiyang Wang

2025-12-08

Contents

1 Abstract

We develop an R implementation for post-shock forecasting of time series using donor balancing weights. The framework is generic and can be applied in any setting where a target series and auxiliary donor series are available; in the empirical section we illustrate the approach with financial data. The implementation consists of four core components: (i) a donor balancing weights procedure (`dbw`) that constructs convex combinations of donor series on a pre-shock window; (ii) a synthetic prediction function (`SynthPrediction`) for post-shock mean forecasting; (iii) an automatic GARCH-X model selection routine (`auto-garchx`) for volatility modeling; and (iv) a volatility-oriented synthetic forecasting routine (`SynthVolForecast`). This technical report documents the data-processing pipeline, model specifications, and code used to obtain the empirical results. All results can be reproduced by running this RMarkdown file together with the accompanying R scripts in the `postshock` package.

2 Introduction

Many time series in economics, finance, environmental science, and other applied domains exhibit structural shocks. Examples include policy announcements that change tax rates, supply disruptions that affect commodity prices, regulatory interventions in financial markets, or abrupt changes in demand following major events. When such shocks occur, models that are estimated on pre-shock data—such as standard ARIMA or GARCH specifications—may no longer provide reliable forecasts for the post-shock period.

In many applications, however, the analyst has access not only to a single target series but also to additional time series that can serve as **donors**. These donor series may have experienced similar shocks in the past, or may be driven by related underlying forces. Donor-based methods, including synthetic control approaches, exploit this additional information by constructing a weighted combination of donors that mimics the pre-shock behaviour of the target. The resulting synthetic series can then be used as a proxy for the counterfactual path of the target in the absence of the current shock.

Existing forecasting practice often treats the target series in isolation, or uses donors only as simple covariates in regression or ARIMAX models. Such approaches typically do not explicitly balance the donors against the target on a pre-shock window and therefore may fail to extract the most informative donor combinations. Moreover, implementations are frequently problem-specific and scattered across ad hoc scripts, making it difficult to reproduce results or to apply the same methodology to new data sets.

The `postshock` package aims to address these limitations by providing a modular R implementation of donor-based post-shock forecasting. The framework is deliberately generic: the target and donor series can be revenue time series, macroeconomic indicators, volatility measures, or any other

variables observed on a common time grid. The current version of the package contains four core components: - `dbw`, a donor balancing weights procedure that constructs convex combinations of donor series on a pre-shock window; - `SynthPrediction`, a synthetic prediction routine for post-shock mean forecasts based on ARIMA/ARIMAX models; - `auto-garchx`, an automatic GARCH-X model-selection routine for conditional volatility; and - `SynthVolForecast`, a volatility-oriented synthetic forecasting procedure that combines donor-based weighting with GARCH-X modeling.

Our methodology is inspired by donor-pool ideas such as those of Lin and Eck (2021), in which shocks from a pool of related series are aggregated to form shock estimators. In contrast to problem-specific implementations, we focus on building a reusable software framework. The weighting layer, implemented in `dbw()`, is separated from the modeling layer, which currently includes ARIMA/ARIMAX and GARCH-X components. This separation makes it straightforward to plug in alternative time-series models in future work while keeping the donor-balancing machinery unchanged.

The purpose of this technical report is not to introduce new theoretical estimators, but rather to document the implementation and to provide a transparent, reproducible account of how the methods are used in practice. We describe the problem setup and notation, the algorithms underlying the four core functions, and the empirical pipeline that produces our results from a single RMarkdown file. As an empirical benchmark, we build on Experiment 1 in Lin and Eck (2021) and analyze post-shock forecasting for ConocoPhillips stock using the same underlying data. While Lin and Eck focus on an AR(1) specification and a problem-specific implementation, our framework provides a more general donor-based architecture with both mean and volatility components, implemented in reusable R functions. Using the ConocoPhillips data as a case study, we demonstrate that the proposed methods can achieve more accurate post-shock forecasts and are readily applicable to other time-series settings beyond this particular example.

The remainder of the report is organized as follows. Section 3 introduces the notation and describes the methodology and implementation of the four core components of the postshock package. Section 4 presents the data and empirical setup. Section 5 reports a simulation study for both the postshock mean and volatility equations. Section 6 contains an empirical application to ConocoPhillips under large shocks and illustrates how the proposed framework can be used in practice.

3 Methodology and Implementation

3.1 Problem setup and notation

We consider a single target time series $Y_{0,t}$, observed at equally spaced times $t = 1, \dots, T$. In addition, we observe J donor series $Y_{j,t}$, $j = 1, \dots, J$, that are potentially informative about the behaviour of the target after a structural shock. Let T_0 denote the last observation before the shock for the target. We write

$$\mathcal{T}_0 = \{1, \dots, T_0\}$$

for the pre-shock window, and we are primarily interested in forecasting the post-shock values

$$Y_{0,T_0+1}, \dots, Y_{0,T_0+k}$$

for a given forecast horizon $k \geq 1$.

For each series we may also observe a vector of contemporaneous covariates $X_{j,t} \in \mathbb{R}^p$, collected into matrices

$$X_j = (X_{j,t}^\top)_{t \in \mathcal{T}_0}, \qquad j = 0, \dots, J.$$

In the `postshock` package these objects are represented as lists,

$$\mathcal{Y} = \{Y_0, Y_1, \dots, Y_J\}, \qquad \mathcal{X} = \{X_0, X_1, \dots, X_J\},$$

which correspond to the R arguments `Y_series_list` and `covariates_series_list`, respectively. The user also supplies a vector `shock_time_vec` that records the last pre-shock time T_0 for each series, and a vector `shock_length_vec` that records the length of the shock window.

Our implementation proceeds in two layers. In the **weighting layer**, pre-shock information is used to construct donor weights that balance the target and donors on $\mathcal{T}_0$; this is implemented in the function `dbw()`. In the **modeling layer**, time-series models are fitted to each series and combined using the donor weights to obtain post-shock forecasts. The modeling layer currently includes ARIMA/ARIMAX components for mean forecasting and GARCH-X components for volatility forecasting, accessed through the functions `SynthPrediction`, `auto-garchx`, and `SynthVolForecast`.

3.2 Donor balancing weights (`dbw`)

The first component of the framework is a donor balancing weights procedure, implemented in the function `dbw()`. The goal is to construct a weight vector

$$\omega = (\omega_1, \dots, \omega_J)^\top$$

on the donor series such that a weighted combination of donors closely matches the target on a pre-shock window. Intuitively, donors that look more similar to the target before the shock should receive larger weights, while dissimilar donors should receive smaller (or zero) weights.

For exposition, suppose that each series is represented by a feature vector $z_i \in \mathbb{R}^K$ summarizing its pre-shock history, where $i = 0$ denotes the target and $i = 1, \dots, J$ denote the donors. These features can include raw covariates at the last pre-shock time, multiple lookback values, and transformed auxiliary information, as described below. Let

$$z_0 \in \mathbb{R}^K \quad \text{and} \quad Z_0 = \begin{pmatrix} z_1^\top \\ \vdots \\ z_J^\top \end{pmatrix} \in \mathbb{R}^{J \times K}$$

collect the target feature vector and the donor feature matrix. A basic formulation of the donor balancing problem is then

$$\min_{\omega} \left\| z_0 - Z_0^\top \omega \right\|_p + \lambda \cdot \mathrm{pen}(\omega) \quad \text{subject to} \quad \sum_{j=1}^{J} \omega_j = 1, \qquad L \le \omega_j \le U \text{ for all } j,$$

where $\|\cdot\|_p$ is either an ℓ_1 or ℓ_2 norm, $\lambda \ge 0$ is a regularization parameter, $\mathrm{pen}(\omega)$ is an ℓ_1 (LASSO-type) or ℓ_2 (ridge-type) penalty on the weights, and L, U are lower and upper bounds. The simplex constraint $\sum_j \omega_j = 1$ together with nonnegative bounds ensures that ω can be interpreted as convex combination weights on the donors.

For clarity, we summarize the main notation used in this section. The integer J denotes the number of donor series, and K is the dimension of the feature vectors used for balancing. For each series i,

the vector $z_i \in \mathbb{R}^K$ collects its pre-shock features, with z_0 corresponding to the target and $z_1, \ldots, z_J$ to the donors. The matrix $Z_0 \in \mathbb{R}^{J \times K}$ stacks the donor features row-wise, so that $Z_0^\top \omega$ is the weighted donor feature vector under weights $\omega = (\omega_1, \ldots, \omega_J)^\top \in \mathbb{R}^J$. The parameter $p \in \{1, 2\}$ determines whether an ℓ_1 or ℓ_2 norm is used in the matching loss, and $\lambda \geq 0$ controls the strength of the penalty $\text{pen}(\omega)$, which can be either ℓ_1-type (LASSO) or ℓ_2-type (ridge) depending on the argument `penalty_normchoice`. The constants L and U are lower and upper bounds on each weight, typically $L = 0$ and $U = 1$, while the equality constraint $\sum_{j=1}^{J} \omega_j = 1$ (corresponding to `sum_to_1 = TRUE`) enforces that ω defines a convex combination of the donor series.

In the `postshock` implementation, the input `X` is a list of data frames or matrices. The first element `X[[1L]]` contains the target series, and the remaining elements `X[[2L]]`, …, `X[[J+1L]]` contain the donors. Rows correspond to time and columns to features. The integer vector `dbw_indices` selects which feature columns are used in the matching objective. The vector `shock_time_vec` has the same length as `X` and records, for each series, the last pre-shock row; each series is truncated to rows `1:shock_time_vec[i]` before any further processing.

Several options control how the feature vectors z_i are constructed:

- The optional list `Y` can contain auxiliary series for each element of `X`, for example returns or transformed variables.
- The argument `Y_lookback_indices` specifies which backwards-looking rows of `Y[[i]]` are to be used (for example, `c(1, 2, 5)` corresponds to the last, second last, and fifth last pre-shock rows), and the function `inputted_transformation` is applied to each `Y[[i]]` before these lookback values are extracted.
- Similarly, `X_lookback_indices` can be used to extract multiple lookback rows from the truncated `X[[i]]`; if `X_lookback_indices` is `NULL`, only the final pre-shock row is used.

The selected rows are vectorized and concatenated to form a single feature vector for each series. Constant columns are removed automatically, and optional centering and scaling are applied via the logical arguments `center` and `scale`. If `princ_comp_count` is not `NULL`, a principal component analysis is performed and only the specified number of leading components are retained, which can help mitigate collinearity in high-dimensional settings.

Distance and regularization choices are controlled by the arguments `normchoice`, `penalty_normchoice`, and `penalty_lambda`. The argument `normchoice` is either `"l1"` or `"l2"`, corresponding to ℓ_1 or ℓ_2 distance in the matching loss. The argument `penalty_normchoice` selects an ℓ_1 (`"l1"`) or ℓ_2 (`"l2"`) penalty on the weights, and `penalty_lambda` is constrained to be nonnegative and specifies the penalty strength. When `sum_to_1 = TRUE` and the bounds `bounded_below_by` and `bounded_above_by` enforce nonnegative weights, the ℓ_1 norm of ω is (up to numerical tolerance) fixed at one; in this regime an ℓ_1 penalty only adds a constant to the objective and does not change the minimizer, so an ℓ_2 penalty is typically more effective under simplex constraints.

Formally, the optimization problem is solved using the `solnp` routine from the `Rsolnp` package, subject to the box constraints `bounded_below_by` and `bounded_above_by` and the optional equality constraint `sum_to_1`. The function returns a list with three main components: `opt_params`, the estimated donor weight vector $\widehat{\omega}$; `loss`, the final matching loss; and `convergence`, a flag indicating whether the numerical optimizer reports successful convergence. These weights are then used inside

`SynthPrediction()` and `SynthVolForecast()` to combine donor-specific forecasts for the target series.

In the empirical work of this report we adopt a convex-combination specification with ℓ_2-regularized weights. Concretely, all examples in Sections 3–4 use the following default options:

- `sum_to_1 = TRUE`
- `bounded_below_by = 0`
- `bounded_above_by = 1`
- `normchoice = "l2"`
- `penalty_normchoice = "l2"`
- `penalty_lambda = 1e-2`

Under this configuration the weights can be interpreted as nonnegative donor shares that sum to one, while the ℓ_2 penalty stabilizes the solution when donors are highly correlated.

```r
set.seed(1)

T  <- 50
K  <- 3

# Construct a target and two donors with K=3 features
target <- data.frame(
  x1 = rnorm(T),
  x2 = rnorm(T,  0.2),
  x3 = rnorm(T, -0.1)
)
donor1 <- data.frame(
  x1 = rnorm(T,  0.1),
  x2 = rnorm(T, -0.1),
  x3 = rnorm(T,  0.3)
)
donor2 <- data.frame(
  x1 = rnorm(T, -0.2),
  x2 = rnorm(T,  0.4),
  x3 = rnorm(T,  0.0)
)

X_list <- list(target, donor1, donor2)
shock  <- c(40, 40, 40)   # last pre-shock row for each series

dbw_out <- dbw(
  X                = X_list,
  dbw_indices      = 1:K,
  shock_time_vec   = shock,
  center           = TRUE,
  scale            = TRUE,
  sum_to_1         = TRUE,
```

```
  bounded_below_by  = 0,
  bounded_above_by  = 1,
  normchoice        = "l2",
  penalty_normchoice = "l2",
  penalty_lambda    = 1e-2
)
```

```
##
## Iter: 1 fn: 2.8119    Pars:  0.17033 0.82967
## Iter: 2 fn: 2.8119    Pars:  0.17032 0.82968
## solnp--> Completed in 2 iterations
```

```
cat("Estimated donor weights:\n")
```

```
## Estimated donor weights:
```

```
print(round(dbw_out$opt_params, 3))
```

```
## [1] 0.17 0.83
```

```
cat("\nFinal matching loss:\n")
```

```
##
## Final matching loss:
```

```
print(dbw_out$loss)
```

```
## [1] 2.804706
```

```
cat("\nConvergence flag:\n")
```

```
##
## Convergence flag:
```

```
print(dbw_out$convergence)
```

```
## [1] "convergence"
```

3.3 Synthetic mean forecasting (`SynthPrediction`)

The second component of the framework is a synthetic mean–forecasting procedure, implemented in the function `SynthPrediction()`. The aim is to produce k-step-ahead forecasts for a target series by combining information from a pool of donor series that have experienced structural shocks in the past. The procedure operates in four layers: (i) estimation of donor-specific shock effects via (S)ARIMA models with a post-shock indicator; (ii) construction of donor weights (either equal or DBW-based); (iii) fitting an ARIMA model to the pre-shock portion of the target; and (iv) adjusting the target forecast by the weighted donor shock effect.

For notation, let $Y_{i,t}$ denote the value of series $i = 0, 1, \dots, J$ at time t, where $i = 0$ is the target and $i = 1, \dots, J$ are donors. For each series we observe a shock start time T_i and a post-shock window length L_i. In the implementation these are supplied via the vectors `shock_time_vec` and `shock_length_vec`, and are first converted to integer indices that match the time index of `Y_series_list`.

3.3.1 Donor-level (S)ARIMA with post-shock indicators

For each donor $i = 1, \dots, J$, we fit a time-series model with a post-shock indicator. Let

$$D_{i,t} = \begin{cases} 0, & t \le T_i, \\ 1, & T_i < t \le T_i + L_i, \end{cases}$$

and write $X_{i,t}$ for a vector of contemporaneous covariates (if any). The donor model can be written schematically as

$$Y_{i,t} = \mathrm{ARIMA}_i(Y_{i,\cdot}, X_{i,\cdot}) + \gamma_i D_{i,t} + \varepsilon_{i,t},$$

where γ_i is a donor-specific shock effect and $\varepsilon_{i,t}$ denotes the innovation. In the code, `SynthPrediction()` implements this by calling `forecast::auto.arima()` (or a user-specified `arima_order`) with `xreg` equal to a design matrix whose last column is the post-shock indicator $D_{i,t}$. The function `decide_seasonal()` is used to determine for each series whether a seasonal component should be included, based on `forecast::findfrequency()` or the user-specified `seasonal_period`.

If the argument `covariate_indices` is non-`NULL`, the donor design matrix contains both the post-shock indicator and the selected columns of `covariates_series_list[[i]]` as contemporaneous regressors. If `covariate_indices` is `NULL`, the only regressor is the post-shock indicator. After fitting the model, the donor shock effect $\hat{\gamma}_i$ is extracted as the coefficient on the last regressor (i.e., the post-shock indicator) using `lmtest::coeftest()`. Collecting these into a vector

$$\hat{\gamma} = (\hat{\gamma}_1, \dots, \hat{\gamma}_J)^\top,$$

we obtain donor-level estimates of how each past shock affected its corresponding series over the specified window.

3.3.2 Donor weighting

In the second layer, `SynthPrediction()` constructs donor weights $w = (w_1, \dots, w_J)^\top$. If `use_dbw = FALSE`, equal weights $w_j = 1/J$ are used. If `use_dbw = TRUE`, the function calls `dbw()` to compute data-driven weights based on pre-shock covariates.

For this purpose, the object `X_for_dbw` is formed by taking `covariates_series_list` (if available) or falling back to `Y_series_list`. Each element is converted to a numeric matrix. If `dbw_indices` is `NULL`, informative columns are selected automatically by examining cross-series variance at the last pre-shock row and discarding nearly constant columns. The resulting indices, together with the integer shock times, are passed to `dbw()` along with options such as `dbw_scale`, `dbw_center`, `penalty_lambda`, and `penalty_normchoice`. When `dbw()` succeeds, the optimal weights $\widehat{w}$ are taken from `dbw_output$opt_params`; otherwise the procedure falls back to equal weights.

The combined donor shock effect used for adjustment is then

$$\hat{\theta} = \sum_{j=1}^{J} \widehat{w}_j \hat{\gamma}_j,$$

which aggregates individual donor effects according to their DBW weights.

3.3.3 Target (S)ARIMA and shock adjustment

In the third layer, we fit an ARIMA model to the target series using only pre-shock data. Let T_0 denote the last pre-shock time for the target. `SynthPrediction()` restricts the target series to observations $t = 1, \dots, T_0$ and fits an ARIMA or seasonal ARIMA model in the same manner as for the donors, using `auto.arima()` or a user-specified `arima_order`. If `covariate_indices` is non-`NULL`, the selected columns of `covariates_series_list[[1]]` are lagged by one period (via `lag.xts`) and included as `xreg` for model fitting. For forecasting, the last pre-shock covariate row is replicated k times to form the `newxreg` matrix, so that the k-step-ahead forecasts correspond to holding the covariates fixed at their last observed values.

Let $\widehat{Y}^{\text{raw}}_{0,T_0+h}$ denote the resulting raw forecast for the target at horizon $h = 1, \dots, k$. The adjusted "DBW version" forecast is defined by

$$\widehat{Y}^{\text{adj}}_{0,T_0+h} = \widehat{Y}^{\text{raw}}_{0,T_0+h} + \hat{\theta},$$

which adds the weighted donor shock effect $\hat{\theta}$ to the raw ARIMA forecast. For comparison, the function also computes an "arithmetic mean" version that uses the simple average of donor effects,

$$\widehat{Y}^{\text{arith}}_{0,T_0+h} = \widehat{Y}^{\text{raw}}_{0,T_0+h} + \frac{1}{J} \sum_{j=1}^{J} \hat{\gamma}_j.$$

The outputs of `SynthPrediction()` are collected in a list with three main components. The element `linear_combinations` contains the donor weight vector $\widehat{w}$; `predictions` contains three forecast vectors labelled `unadjusted`, `adjusted`, and `arithmetic_mean`; and `meta` stores additional information such as the donor-specific $\hat{\gamma}_j$, the combined effect $\hat{\theta}$, the ARIMA orders, and the information criterion used. These objects are used in Section 4 to construct the empirical results.

3.3.3.1 Illustrative simulation example To illustrate how `SynthPrediction()` behaves in a simple setting, we simulate one target series and two donor series observed on a daily grid with $T = 200$. The target Y_0 is a random walk around level 100. Donor 1 is a very close copy of the target (small extra noise), while donor 2 is much noisier. As covariates we construct a second random walk ("base signal") and create two highly informative covariates X_0, X_1 plus one largely

uninformative covariate X_2. We treat time index $T_0 = 150$ as the shock time for all series and estimate a one–step–ahead forecast ($k = 1$).

```r
library(xts)
set.seed(123)

## 1) Target and donors ---------------------------------
T     <- 200
dates <- seq.Date(as.Date("2021-01-01"), by = "day", length.out = T)

# target: random walk around level 100
y0 <- cumsum(rnorm(T, mean = 0, sd = 1)) + 100
Y0 <- xts(y0, order.by = dates)

# donor 1: very similar to target (small extra noise)
y1 <- y0 + rnorm(T, mean = 0, sd = 0.1)
Y1 <- xts(y1, order.by = dates)

# donor 2: noisier copy of target
y2 <- y0 + rnorm(T, mean = 0, sd = 2)
Y2 <- xts(y2, order.by = dates)

Y_list <- list(Y0, Y1, Y2)

## 2) Covariates ----------------------------------------
base_signal <- cumsum(rnorm(T, mean = 0, sd = 1))   # common latent factor

X0 <- xts(base_signal + rnorm(T, sd = 0.1), order.by = dates)
X1 <- xts(base_signal + rnorm(T, sd = 0.1), order.by = dates)
X2 <- xts(cumsum(rnorm(T, sd = 2)),         order.by = dates)  # less informative

X_list <- list(X0, X1, X2)

## 3) Shock specification ------------------------------
shock_time_vec   <- c(150, 150, 150)  # shock happens after time 150
shock_length_vec <- c(1L,  1L,  1L)   # one-step post-shock window

## 4) Run SynthPrediction ------------------------------
res <- SynthPrediction(
  Y_series_list          = Y_list,
  covariates_series_list = X_list,
  shock_time_vec         = shock_time_vec,
  shock_length_vec       = shock_length_vec,
  k                      = 1,
  covariate_indices      = 1,
  seasonal               = FALSE,
  plots                  = FALSE
)
```

```
##
## Iter: 1 fn: 0.000000001299    Pars:  0.98802 0.01198
## Iter: 2 fn: 0.000000001299    Pars:  0.98802 0.01198
## solnp--> Completed in 2 iterations
```

```
res$linear_combinations      # DBW donor weights
```

```
## [1] 0.9880241 0.0119759
```

```
res$predictions              # raw / adjusted forecasts
```

```
## $unadjusted
## [1] 96.35314
##
## $adjusted
## [1] 97.25629
##
## $arithmetic_mean
## [1] 97.46149
```

```
# True value of Y0 at T0+1 (time 151)
Y_list[[1]][150:151]
```

```
##                [,1]
## 2021-05-30 96.34557
## 2021-05-31 97.13331
```

```
truth <- as.numeric(Y_list[[1]][151])

c(
  truth      = truth,
  raw        = res$predictions$unadjusted[1],
  dbw_adj    = res$predictions$adjusted[1],
  arith_mean = res$predictions$arithmetic_mean[1]
)
```

```
##      truth        raw    dbw_adj arith_mean
##   97.13331   96.35314   97.25629   97.46149
```

3.4 Volatility model selection (`auto_garchx`)

The next component of the framework is a volatility–forecasting module based on GARCH–type models with exogenous regressors. It is implemented in the function `auto_garchx()`, which selects

a GARCHX specification by BIC and optionally refits the selected model using a more stable backcast for the conditional variance.

Let $\{y_t\}_{t=1}^n$ denote a (demeaned) return series and let x_t collect exogenous regressors (if any). Following the parametrisation of the `garchx` package, we consider models of the form

$$y_t = \sigma_t \varepsilon_t, \qquad \varepsilon_t \sim \text{i.i.d. } (0,1),$$

with conditional variance

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i y_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2 + \sum_{\ell=1}^{q} \gamma_\ell y_{t-\ell}^2 \mathbf{1}\{y_{t-\ell} < 0\} + \delta^\top x_t.$$

Here q is the ARCH order, p is the GARCH order, and the coefficients γ_ℓ capture GJR–type asymmetry. In the `garchx` implementation these orders are supplied as an integer vector `order` $= (q, p, o)$, where $o \in \{0, 1\}$ indicates whether the asymmetric GJR term is included ($o = 1$) or not ($o = 0$).

The main arguments of `auto_garchx()` are:

- `y`: numeric vector of returns or residuals (no `NA`/`Inf`/`NaN`);
- `xreg`: optional matrix of exogenous regressors aligned with `y`;
- `max.p`, `max.q`: upper bounds for the GARCH (`p`) and ARCH (`q`) orders;
- `search_o`: subset of $\{0, 1\}$ indicating whether symmetric (`o = 0`) and/or asymmetric (`o = 1`) specifications are considered;
- `final_refit`: logical flag controlling whether a second "backcast refit" step is applied to the selected model;
- `clamp_factor`: two–element numeric vector controlling the range used to clamp the unconditional variance relative to the sample variance;
- `vcov.type`: type of standard errors requested from `garchx` (`"ordinary"` or `"robust"`).

3.4.1 Grid search over (q, p, o) by BIC

The function `auto_garchx()` performs a grid search over the order triple (q, p, o). Given user–supplied maxima `max.q`, `max.p` and the set `search_o {0,1}`, the function loops over

$$q = 0, \ldots, \texttt{max.q}, \quad p = 0, \ldots, \texttt{max.p}, \quad o \in \texttt{search\_o},$$

and fits a `garchx` model

3.4.2 Illustrative simulation example

```
library(postshock)

## Simulate a GARCH(1,1) series
set.seed(123)
n      <- 2000
omega  <- 0.05
```

```r
alpha  <- 0.07
beta   <- 0.90

eps    <- rnorm(n)
sigma2 <- numeric(n)
y      <- numeric(n)

sigma2[1] <- omega / (1 - alpha - beta)
y[1]      <- sqrt(sigma2[1]) * eps[1]

for (t in 2:n) {
  sigma2[t] <- omega + alpha * y[t-1]^2 + beta * sigma2[t-1]
  y[t]      <- sqrt(sigma2[t]) * eps[t]
}

## Run auto_garchx
best <- auto_garchx(
  y,
  max.p       = 3,
  max.q       = 3,
  search_o    = c(0L, 1L),
  final_refit = TRUE,
  vcov.type   = "robust",
  verbose     = FALSE
)

## Show selected orders and BIC
c(p = best$p, q = best$q, o = best$o)
```

```
## p q o
## 1 1 0
```

```r
best$bic
```

```
## [1] 6571.432
```

```r
coef(best$fit)
```

```
##  intercept      arch1     garch1
## 0.04597810 0.06060302 0.91151343
```

3.5 Synthetic volatility forecasting (`SynthVolForecast`)

The last component of the framework is a *synthetic volatility–forecasting* module, implemented in the function `SynthVolForecast()`.

The goal is to produce k-step–ahead **variance** forecasts for a target series by combining a standard pre–shock GARCH model with donor–based post–shock fixed effects.

For one target series and J donors we observe demeaned returns $\{y_{i,t}\}_{t=1}^{n}$ and, for donors, a set of contemporaneous covariates $x_{i,t}$. For each series we also observe a shock time T_i and a post–shock window length L_i, supplied via the vectors `shock_time_vec` and `shock_length_vec`, and collected into the list `Y_series_list` (target in position 1, donors in positions 2,…,$J+1$).

3.5.1 Donor–level GARCH–X with post–shock indicators

For each donor $i = 1, \dots, J$ we fit a GARCH–type model with a 0/1 post–shock indicator. Following the parametrisation of the `garchx` package, we consider

$$y_{i,t} = \sigma_{i,t}\varepsilon_{i,t}, \qquad \varepsilon_{i,t} \sim \text{i.i.d. } (0,1),$$

with conditional variance

$$\sigma_{i,t}^2 = \omega_i + \sum_{\ell=1}^{q_i} \alpha_{i,\ell} y_{i,t-\ell}^2 + \sum_{\ell=1}^{p_i} \beta_{i,\ell}\sigma_{i,t-\ell}^2 + \sum_{\ell=1}^{q_i} \gamma_{i,\ell} y_{i,t-\ell}^2 \mathbf{1}\{y_{i,t-\ell} < 0\} + \delta_i^\top x_{i,t} + \eta_i D_{i,t},$$

where q_i and p_i are the ARCH and GARCH orders, and $D_{i,t}$ is a post–shock indicator defined by

$$D_{i,t} = \begin{cases} 0, & t \le T_i, \\ 1, & T_i < t \le T_i + L_i, \\ 0, & t > T_i + L_i. \end{cases}$$

In the implementation we build the design matrix for donor i by stacking the selected donor covariates (columns chosen via `covariate_indices`) together with `post_shock_indicator`, whose coefficient corresponds to η_i. The function then fits either

- a user–supplied GARCH order `garch_order = c(q,p,o)`, or
- if `garch_order = NULL`, a data–driven specification chosen by `auto_garchx()` via BIC over a grid in (q, p, o).

After fitting, the donor–specific post–shock effect $\widehat{\omega}_i^* := \widehat{\eta}_i$ is extracted using `lmtest::coeftest()` if available, and stored in the vector

$$\widehat{\omega}^* = (\widehat{\omega}_1^*, \dots, \widehat{\omega}_J^*)^\top.$$

3.5.2 Donor weighting via DBW

To transfer information from donors to the target we form a covariate collection

$$X_{\text{for dbw}} = (X_{\text{target}}, X_1, \dots, X_J),$$

where X_{target} is the covariate matrix for the target (`target_covariates`) and X_j is the donor–j covariate matrix (`covariates_series_list[[j]]`). The function `dbw()` is then used to compute donor weights $w = (w_1, \dots, w_J)^\top$ by solving a regularised balancing problem on the pre–shock covariates. The main tuning parameters are

- `dbw_indices` : which covariate columns enter the balancing problem;
- `penalty_lambda` and `penalty_normchoice` : regularisation weight and norm;
- `dbw_scale` and `dbw_center` : whether to standardise covariates.

The resulting combined donor shock effect is

$$\hat{\theta} = \sum_{j=1}^{J} w_j \hat{\omega}_j^*,$$

which aggregates donor–level post–shock effects according to their DBW weights.

3.5.3 Target GARCH and variance adjustment

For the target series we fit a **pre–shock** GARCH model *without* exogenous regressors. Let T_0 denote the last pre–shock time for the target. The function `SynthVolForecast()` restricts the target data to $t = 1, \dots, T_0$ and fits either a user–specified `garch_order` or a BIC–selected specification via `auto_garchx()`. Because no `xreg` is used for the target, forecasting does not require a future `newxreg` matrix.

Let $\hat{v}^{\text{raw}}_{0,T_0+h}$ denote the resulting k-step–ahead **variance** forecast at horizon $h = 1, \dots, k$. The DBW–adjusted volatility forecast is defined by

$$\hat{v}^{\text{adj}}_{0,T_0+h} = \hat{v}^{\text{raw}}_{0,T_0+h} + \hat{\theta},$$

which adds the combined donor post–shock effect $\hat{\theta}$ to the raw GARCH variance forecast. For comparison, the function also computes an "arithmetic–mean" adjustment that ignores DBW and uses the simple average of donor effects,

$$\hat{v}^{\text{arith}}_{0,T_0+h} = \hat{v}^{\text{raw}}_{0,T_0+h} + \frac{1}{J} \sum_{j=1}^{J} \hat{\omega}_j^*.$$

The outputs of `SynthVolForecast()` are collected in a list with three main components. The element `linear_combinations` contains the donor weight vector $\hat{w}$; `predictions` contains three forecast vectors labelled `unadjusted`, `adjusted`, and `arithmetic_mean`; and `meta` stores additional information such as the donor–specific $\hat{\omega}_j^*$, the combined effect $\hat{\theta}$, the selected GARCH orders, and the DBW convergence status. If a vector of realised variances `ground_truth_vec` is supplied, the function also returns QLIKE losses for each forecast version.

3.5.4 Illustrative simulation example

```
library(postshock)

set.seed(123)
```

```r
## 1) Simulate one target and two donor return series ---------------------
n    <- 500
eps0 <- rnorm(n)
eps1 <- rnorm(n)
eps2 <- rnorm(n)

# simple homoskedastic DGP for illustration
y0 <- eps0                      # target
y1 <- 0.8 * eps0 + 0.2 * eps1   # donor 1, highly correlated with target
y2 <- 0.5 * eps0 + 0.5 * eps2   # donor 2, less correlated

Y_list <- list(y0, y1, y2)

## 2) Simple covariates for target and donors -----------------------------

X1 <- cbind(x1 = rnorm(n), x2 = rnorm(n))   # donor 1 covariates
X2 <- cbind(x1 = rnorm(n), x2 = rnorm(n))   # donor 2 covariates
Xt <- cbind(x1 = rnorm(n), x2 = rnorm(n))   # target covariates

## 3) Shock times and window lengths --------------------------------------
shock_time_vec   <- c(350L, 350L, 350L)
shock_length_vec <- c(20L,  20L,  20L)

## 4) Call SynthVolForecast -----------------------------------------------
out_vol <- SynthVolForecast(
  Y_series_list          = Y_list,
  covariates_series_list = list(X1, X2),
  target_covariates      = Xt,
  shock_time_vec         = shock_time_vec,
  shock_length_vec       = shock_length_vec,
  k                      = 5,
  covariate_indices      = 1:2,
  dbw_indices            = 1:2,
  plots                  = FALSE,
  return_fits            = FALSE
)
```

```
##
## Iter: 1 fn: 1.5017    Pars:  0.0000000118 0.9999999882
## Iter: 2 fn: 1.5017    Pars:  0.000000005126 0.999999994874
## solnp--> Completed in 2 iterations
```

```r
## 5) Show number of donors, weights, and forecasts -----------------------
out_vol$meta$n_donors
```

```
## [1] 2
```

```
out_vol$linear_combinations
```

```
## [1] 5.125629e-09 1.000000e+00
```

```
out_vol$predictions
```

```
## $unadjusted
## [1] 0.8920638 0.9152028 0.9166576 0.9157751 0.9147486
##
## $adjusted
## [1] 0.9887246 1.0118637 1.0133185 1.0124360 1.0114094
##
## $arithmetic_mean
## [1] 0.9864278 1.0095668 1.0110216 1.0101392 1.0091126
```

4 Data and Empirical Setup

4.1 Data sources

Our empirical analysis focuses on ConocoPhillips (COP) as the target series. We construct a daily panel that combines COP with a set of macro-financial variables that are plausibly related to its behaviour. Specifically, we collect:

- the daily closing price of ConocoPhillips (ticker: COP);
- the daily closing level of the S&P 500 index (ticker: ^GSPC);
- the daily closing price of WTI crude oil (downloaded from the DataHub "wti-daily" dataset);
- the daily U.S. Dollar Index (ticker: DX-Y.NYB);
- the daily 3-month U.S. Treasury bill rate (ticker: ^IRX);
- the daily closing level of the VIX volatility index (ticker: ^VIX).

All series are obtained at the daily frequency starting in January 2000, so that the sample covers the 2008 financial crisis, the 2014–2015 oil price decline, and the COVID-19 period. For each series we retain trading days for which all variables are observed and merge them into a common panel using the calendar date as the key.

4.2 Inflation adjustment

To make different historical episodes comparable in magnitude, we convert all nominal price and index series into 2020 U.S. dollars. We obtain the monthly consumer price index series CPIAUCSL from the Federal Reserve Economic Data (FRED) and download it in CSV format. For each calendar year y we compute the average CPI level $\overline{\text{CPI}}_y$. Let $\overline{\text{CPI}}_{2020}$ denote the average CPI level in 2020. We then construct an inflation adjustment factor

$$\text{adj}_y = \frac{\overline{\text{CPI}}_{2020}}{\overline{\text{CPI}}_y},$$

and multiply all observations in year y by adj_y. This procedure rescales COP, the S&P 500, WTI, the Dollar Index, the T-bill rate, and the VIX so that they are all expressed in "2020-dollar" units, removing long-run inflation and allowing us to compare shocks that occur in different years on a common real scale.

4.3 Construction of the analysis panel

After computing the inflation adjustment factors described above, we construct the panel that is used as input to our post-shock forecasting procedures. The CPI-based scaling is applied within each shock window in the empirical examples. We first align COP and the five covariates by date and keep only dates for which all six series are observed. Let COP_t denote the inflation-adjusted closing price of COP on day t. We define the response series Y_t as the one-day-ahead COP value,

$$Y_t = \text{COP}_{t+1},$$

and stack the contemporaneous covariates at day t into a vector

$$X_t = (\text{COP}_t,\ \text{GSPC}_t,\ \text{WTI}_t,\ \text{USD}_t,\ \text{TB}_t,\ \text{VIX}_t)^\top.$$

In the implementation, we create a data frame `COP_close` that contains the date, the inflation-adjusted COP price, the inflation-adjusted S&P 500, WTI, Dollar Index, T-bill rate, VIX, and the response variable Y_t. Each row of this data frame corresponds to one trading day and provides the basic building block from which we later extract shock windows for the target episode and the donor episodes used in the empirical examples.

```
## [1] "COP"
```

```
## [1] "GSPC"
```

```
## [1] "GC=F"
```

```
## [1] "DX-Y.NYB"
```

```
## [1] "VIX"
```

5 Simulation study

In this section we investigate the finite–sample behaviour of the proposed postshock forecasting framework in controlled simulation settings. We run two sets of Monte Carlo experiments: one for the postshock mean equation and one for the postshock volatility equation. In each design we generate a target series together with several donor series from a known data–generating process (DGP) that mimics the empirical setting, introduce a one–time structural shock, and compare different forecasting procedures.

All simulations were implemented in R. The R Markdown files `sim-test.Rmd` (for the mean–equation simulations) and `sim_vol_tets.Rmd` (for the volatility simulations) are stored in the same `technical_report` folder as this report and can be used to reproduce all figures and tables in this section.

5.1 Simulation for the postshock mean equation

5.1.1 Design

For the mean equation we use a simplified ARIMAX–type DGP that is aligned with the postshock mean specification in Section~2. For each series $j = 0, 1, \dots, J$ we generate

$$Y_t^{(j)} = \mu_t^{(j)} + \varepsilon_t^{(j)}, \qquad \varepsilon_t^{(j)} \sim \mathcal{N}(0, \sigma^2),$$

and specify the conditional mean as

$$\mu_t^{(j)} = \phi^{(j)} Y_{t-1}^{(j)} + \beta^{(j)\top} X_t^{(j)} + \omega^{(j)} \mathbf{1}\{t = T_0^{(j)} + 1\},$$

where $X_t^{(j)}$ collects the exogenous covariates and $T_0^{(j)}$ denotes the last pre–shock time point for series j. The index $j = 0$ corresponds to the target series and $j \geq 1$ to the donors. The constants $\omega^{(j)}$ represent the true one–period postshock effects in the DGP.

The AR and regression parameters $\phi^{(j)}$ and $\beta^{(j)}$ are chosen so that the target and donors have similar, but not identical, pre–shock dynamics. In particular, some donors are constructed to be close to the target in terms of their pre–shock covariate means and serial dependence, while others are deliberately more distant. This mirrors the empirical setting where some historical episodes are more informative donors than others.

For each series we define a local window of fixed length around the shock. Let $T_0^{(j)}$ denote the last pre–shock time point for series j. We take $T_{\text{pre}} = 30$ pre–shock observations and $T_{\text{post}} = 30$ postshock observations, so that each window consists of

$$t = T_0^{(j)} - T_{\text{pre}} + 1, \dots, T_0^{(j)} + T_{\text{post}},$$

with the structural break occurring between $T_0^{(j)}$ and $T_0^{(j)} + 1$. The shock date $T_0^{(j)} + 1$ and the window lengths are treated as known to the analyst.

The simulation designs implemented in `sim-test.Rmd` cover a range of parameter configurations, including different magnitudes of the target shock $\omega^{(0)}$, different degrees of donor heterogeneity in $\phi^{(j)}$ and $\beta^{(j)}$, and different noise levels σ^2.

5.1.2 Estimators and evaluation criteria

For each Monte Carlo replication we construct the target and donor windows, stack the corresponding outcome and covariate series into `Y_series_list` and `covariates_series_list`, and call the `SynthPrediction()` function exactly as in the empirical application to ConocoPhillips. We focus on one–step–ahead postshock forecasting ($k = 1$) and specify a one–period shock length for all series.

We compare three estimators of the postshock mean for the target series:

1. **Unadjusted ARIMAX forecast.**
   The ARIMAX model is fitted only on the pre–shock segment of the target series, and the one–step–ahead postshock forecast is obtained by direct model extrapolation without adding any shock effect. This corresponds to the `out$predictions$unadjusted` component in the output of `SynthPrediction()` and serves as a "no–shock" counterfactual benchmark.

2. **Arithmetic–mean adjusted forecast.**
   The unadjusted ARIMAX forecast is augmented by the simple arithmetic mean of the estimated donor shock effects,

$$\widehat{y}_{\mathrm{AM}} = \widehat{y}_{\mathrm{base}} + \bar{\omega}, \qquad \bar{\omega} = \frac{1}{J}\sum_{j=1}^{J} \widehat{\omega}^{(j)},$$

   where $\widehat{y}_{\mathrm{base}}$ is the unadjusted forecast and $\widehat{\omega}^{(j)}$ is the estimated shock effect for donor j. No covariate balancing is used in this adjustment.

3. **DBW–adjusted forecast.**
   The ARIMAX baseline is augmented by a donor–balancing–weights (DBW) combination of estimated donor shock effects,

$$\widehat{y}_{\mathrm{DBW}} = \widehat{y}_{\mathrm{base}} + \sum_{j=1}^{J} \widehat{w}_j \widehat{\omega}^{(j)},$$

   where the weights $\widehat{w}_j$ are constructed by matching pre–shock covariate means of the target and donors using the DBW procedure described in Section~3. This corresponds to `out$predictions$adjusted` in the implementation.

Performance is evaluated along two dimensions:

- **Shock–effect estimation.**
  We compute the bias and root mean squared error (RMSE) of the implied adjustment term (arithmetic–mean or DBW–weighted) relative to the true target shock $\omega^{(0)}$.
- **Postshock forecast accuracy.**
  We measure the RMSE of the one–step–ahead postshock forecast for the target, i.e. the RMSE of the predicted $Y^{(0)}_{T_0^{(0)}+1}$, across Monte Carlo replications.

All metrics are averaged over a large number of replications (e.g. $R = 1000$) and summarised across parameter settings.

5.1.3 Results

Across all designs considered in `sim-test.Rmd`, the unadjusted ARIMAX forecast is biased towards zero and systematically underestimates the magnitude of the true shock effect. This behaviour is expected: because the baseline model is fitted only on the pre–shock segment, it has no information about the structural break at the postshock date and therefore extrapolates the pre–shock dynamics too smoothly.

The arithmetic–mean adjustment reduces the bias by incorporating information from donor episodes. However, when donors are heterogeneous in terms of their pre–shock covariates or shock sizes, simple averaging can lead to inflated RMSE. Donors that are poorly aligned with the target in their pre–shock environment can exert undue influence on the average and increase forecast dispersion.

The DBW–adjusted forecast exhibits the most favourable performance in the scenarios we considered. By reweighting donors to match pre–shock covariate means, the DBW procedure effectively downweights donors that are dissimilar to the target and places more mass on those episodes that share similar pre–shock environments. Consequently, the DBW–adjusted forecasts attain smaller bias and RMSE for the postshock mean than both the unadjusted and arithmetic–mean estimators, especially in settings with pronounced donor heterogeneity.

5.2 Simulation for the volatility equation

5.2.1 Design

We next investigate the volatility component of the postshock framework using a GARCHX–type DGP aligned with the volatility specification in Section~3. For each series j we generate returns

$$y_t^{(j)} = \sigma_t^{(j)} \varepsilon_t^{(j)}, \qquad \varepsilon_t^{(j)} \sim \mathcal{N}(0,1),$$

with conditional variance

$$\sigma_t^{2(j)} = \alpha_0^{(j)} + \alpha_1^{(j)} y_{t-1}^{2(j)} + \beta_1^{(j)} \sigma_{t-1}^{2(j)} + \gamma^{(j)\top} z_t^{(j)} + \delta^{(j)} \mathbf{1}\{t = T_0^{(j)} + 1\},$$

where $z_t^{(j)}$ collects exogenous volatility regressors (such as lagged realised volatility or macro–financial indicators), and $\delta^{(j)}$ represents the size of a one–time volatility shock at $T_0^{(j)} + 1$.

The parameter vectors $(\alpha_0^{(j)}, \alpha_1^{(j)}, \beta_1^{(j)}, \gamma^{(j)}, \delta^{(j)})$ are chosen so that the series exhibit persistent volatility, leverage from the covariates, and heterogeneous shock magnitudes across donors. As in the mean–equation simulations, we construct local pre– and postshock windows around the shock, with the same pre– and postshock lengths as in the empirical study, and treat both the shock date and the window structure as known.

The volatility simulation designs implemented in `sim_vol_tets.Rmd` explore different levels of volatility persistence, different magnitudes of the volatility shock $\delta^{(0)}$ for the target, and different degrees of donor heterogeneity in the variance parameters.

5.2.2 Estimators and evaluation criteria

For each simulated data set we apply the volatility component of the postshock framework. Specifically, we use `auto_garchx()` to select and estimate a GARCHX specification for each series and then call `SynthVolForecast()` (or its analogue) to construct postshock volatility forecasts for the target, combining information from the donor episodes.

We again compare three approaches:

1. **Unadjusted GARCHX forecast.**
   The GARCHX model is estimated on the pre–shock segment of the target and its conditional variance forecast is extrapolated beyond the shock without any donor–based adjustment. This corresponds to the standard GARCHX variance forecast.

2. **Arithmetic–mean adjusted volatility.**
   The unadjusted GARCHX forecast is shifted by the arithmetic mean of donor–specific volatility shocks, defined analogously to the mean–equation case, by averaging the estimated donor shock effects and adding the resulting adjustment term to the baseline forecast.

3. **DBW–adjusted volatility.**
   Donor–specific volatility shocks are combined using donor balancing weights constructed from pre–shock volatility features and covariates. The resulting weighted adjustment term is added to the baseline GARCHX forecast, yielding a DBW–adjusted postshock variance prediction.

We focus on the accuracy of the one–step–ahead variance forecast at the postshock date and over a short horizon after the shock. Performance is measured by the mean squared error (MSE) of the variance forecast relative to the true conditional variance $\sigma_t^{2(0)}$, averaged across replications and across the time points of interest.

5.3 Results

The simulation results for the volatility component mirror the findings from the mean equation. The unadjusted GARCHX forecasts tend to underreact to sudden increases in volatility at the shock date, especially when the shock magnitude is large relative to typical pre–shock fluctuations. This leads to systematic underprediction of the conditional variance around the shock.

Incorporating donor information through a simple arithmetic average of donor–specific volatility shocks improves the forecasts in many cases, but can be unstable when donors differ markedly in their volatility regimes. Donors with unusually high or low shock responses can distort the average and produce either overreaction or underreaction in the adjusted variance predictions.

The DBW–adjusted volatility forecasts exhibit more stable and accurate behaviour. By matching pre–shock volatility and covariate patterns between the target and the donors, the DBW procedure assigns larger weights to donors that experience similar volatility environments and smaller weights to those that do not. Across the designs considered in `sim_vol_tets.Rmd`, the DBW–adjusted forecasts consistently reduce the MSE relative to both the unadjusted and arithmetic–mean approaches at the postshock date and over the subsequent few periods.

5.4 Discussion

The simulation experiments for both the postshock mean and volatility equations provide a controlled environment in which the true shock effects and conditional moments are known. In this setting we find that unadjusted ARIMAX and GARCHX forecasts systematically underestimate the impact of structural shocks, as they are calibrated solely on pre–shock dynamics. Simple donor averaging partially corrects this bias but can suffer from high variance and instability when donors are heterogeneous.

By contrast, the donor balancing weights used in the proposed framework exploit information in pre–shock covariates to construct synthetic donors that are more closely aligned with the target. In our Monte Carlo designs this leads to lower bias and RMSE for the postshock mean and lower MSE for the postshock variance forecasts. Taken together, the simulation results support the use of DBW in both the mean and volatility components and motivate the empirical application to ConocoPhillips in the next section.

6 Empirical Application: ConocoPhillips under Large Shocks

In this section we apply the postshock forecasting framework developed in Sections 2–3 to the ConocoPhillips (COP) data set described in Section 4. The goal is to forecast the behaviour of COP around large negative shocks by borrowing information from earlier shock episodes that occurred under different market conditions. We treat the COVID-19 oil price collapse in March 2020 as the target episode and use several earlier crisis episodes as donors.

6.1 Target and donor shock episodes

We focus on the large negative shock to ConocoPhillips on 9 March 2020 as the target episode. In the implementation we follow the convention of Section 3 and treat the last pre-shock trading day as the index T_0. For the March 2020 event this corresponds to 6 March 2020, so the shock itself occurs between T_0 and $T_0 + 1$.

As donor episodes we select four earlier crisis windows for COP:

- 17 March 2008 (Bear Stearns collapse, financial crisis episode), with last pre-shock trading day 14 March 2008;
- 8, 12 and 26 September 2008 (Lehman Brothers and bailout turmoil), with 5, 11 and 25 September 2008 as the corresponding last pre-shock days;
- 27 November 2014 (OPEC-related oil price collapse), with last pre-shock trading day 26 November 2014.

For each episode we construct a local window of 30 trading days before T_0 and 30 trading days after the shock, and we work with inflation-adjusted prices expressed in 2020 U.S. dollars, as described in Section 4.

6.2 Construction of shock windows

For each shock episode we work with a local window that includes 30 trading days before the shock date and 30 trading days after it. Let the calendar day of the shock be denoted by $t^\star$. In our implementation we index $t^\star$ as the "centre" of the window and include the range from $t^\star - 30$ to $t^\star + 29$, yielding 60 trading days in total.

Within each window we extract the COP response series Y_t and the contemporaneous covariates X_t from the panel `COP_close` constructed in Section 4. For comparability across time, all prices and covariates in the window are rescaled to 2020 U.S. dollars using the CPI-based inflation factors stored in `inflation_adj`.

```
# Helper to construct an inflation-adjusted shock window
# - center_date: calendar date of the shock (e.g. "2008-03-17")
# - pre:         number of days before the shock to include in the window
# - post:        number of days after the shock to include in the window
#                (including the shock day as the first post-shock day)
# - year_adj:    calendar year used for CPI-based inflation adjustment
make_window <- function(center_date, pre = 30, post = 30, year_adj) {
```

```r
  # Locate the index of the shock date in COP_close
  idx_c <- which(COP_close$Date == center_date)
  if (length(idx_c) != 1L) {
    stop("Date not found or found more than once: ", center_date)
  }

  # Compute start and end row indices of the window
  # Resulting window has 'pre + post' rows:
  #   t = center_date - pre, ..., center_date + post - 1
  idx_start <- idx_c - pre
  idx_end   <- idx_c + post - 1

  # Subset COP_close to obtain the local window
  TS <- COP_close[idx_start:idx_end, ]

  # Apply CPI-based inflation adjustment to COP and covariates
  # (here we assume columns 2:7 are COP + the five covariates)
  factor <- inflation_adj$dollars_2020[inflation_adj$year == year_adj]
  TS[, 2:7] <- TS[, 2:7] * factor

  return(TS)
}

## ================== Build target and donor windows ==================

# Target window around 2020-03-09 (COVID + oil price war episode)
# pre = 30, post = 30 → 30 days before the shock + shock day + 29 days after
TS_target <- make_window("2020-03-09", pre = 30, post = 30, year_adj = 2020)

# Donor 1: 2008-03-17 (March 2008 financial crisis shock)
TS_2008_03_17 <- make_window("2008-03-17", pre = 30, post = 30, year_adj = 2008)

# Donor 2: 2014-11-28 (OPEC supply shock episode)
TS_2014_11_28 <- make_window("2014-11-28", pre = 30, post = 30, year_adj = 2014)

# Donor 3: 2008-09-09 (first September 2008 shock around the Lehman crisis)
TS_2008_09_09 <- make_window("2008-09-09", pre = 30, post = 30, year_adj = 2008)

# Donor 4: 2008-09-15 (Lehman Brothers bankruptcy filing date)
TS_2008_09_15 <- make_window("2008-09-15", pre = 30, post = 30, year_adj = 2008)

# Donor 5: 2008-09-29 (first rejection of the U.S. bailout plan; another large drop)
TS_2008_09_29 <- make_window("2008-09-29", pre = 30, post = 30, year_adj = 2008)

## ================== Quick sanity check: dimensions ==================
lapply(
  list(TS_target,
```

```
       TS_2008_03_17,
       TS_2014_11_28,
       TS_2008_09_09,
       TS_2008_09_15,
       TS_2008_09_29),
  dim
)
```

```
## [[1]]
## [1] 60  8
##
## [[2]]
## [1] 60  8
##
## [[3]]
## [1] 60  8
##
## [[4]]
## [1] 60  8
##
## [[5]]
## [1] 60  8
##
## [[6]]
## [1] 60  8
```

```
# Check that rows 30 and 31 in each window correspond to
# "last pre-shock day" and "shock day" in terms of COP_Close
lapply(
  list(TS_target,
       TS_2008_03_17,
       TS_2014_11_28,
       TS_2008_09_09,
       TS_2008_09_15,
       TS_2008_09_29),
  function(ts) ts[c(30, 31), c("Date", "COP_Close")]
)
```

```
## [[1]]
##            Date COP_Close
## 5054 2020-03-06     45.33
## 5055 2020-03-09     34.07
##
## [[2]]
##            Date COP_Close
## 2047 2008-03-14  71.03741
```

```
## 2048 2008-03-17  69.56147
##
## [[3]]
##            Date COP_Close
## 3736 2014-11-26  77.45497
## 3737 2014-11-28  72.24975
##
## [[4]]
##            Date COP_Close
## 2169 2008-09-08  68.47056
## 2170 2008-09-09  62.62183
##
## [[5]]
##            Date COP_Close
## 2173 2008-09-12  67.31548
## 2174 2008-09-15  62.99769
##
## [[6]]
##            Date COP_Close
## 2183 2008-09-26  69.89149
## 2184 2008-09-29  63.53856
```

6.3 Preparing the series for postshock forecasting

For each window we take the COP closing price as the response series and use five macro–financial variables as covariates: the S&P 500 index, WTI crude oil, the U.S. Dollar Index, the 3–month Treasury bill rate, and the VIX. These are stacked into the lists `Y_series_list` and `covariates_series_list`. Each window has 30 pre–shock observations, so we set the shock time to 30 for all series and focus on a one–day shock effect.

```r
## ========== 1) Choose covariates to use ==========
covar_names <- c("GSPC_Close", "WTI_Close", "USD_Close", "TB_Close", "VIX_Close")

## ========== 2) Build Y_series_list (target + 5 donors) ==========
# Each element is a univariate time series of COP closing prices within a window
Y_series_list <- list(
  Y_target      = TS_target$COP_Close,      # target series: 2020-03-09 window
  Y_2008_03_17  = TS_2008_03_17$COP_Close,  # donor: 2008-03-17 shock
  Y_2014_11_28  = TS_2014_11_28$COP_Close,  # donor: 2014-11-28 shock
  Y_2008_09_09  = TS_2008_09_09$COP_Close,  # donor: 2008-09-09 shock
  Y_2008_09_15  = TS_2008_09_15$COP_Close,  # donor: 2008-09-15 shock
  Y_2008_09_29  = TS_2008_09_29$COP_Close   # donor: 2008-09-29 shock
)

## ========== 3) Build covariates_series_list ==========
# Each element is a matrix of covariates aligned with the corresponding Y series
covariates_series_list <- list(
```

```
  X_target      = as.matrix(TS_target[,      covar_names]),
  X_2008_03_17  = as.matrix(TS_2008_03_17[,  covar_names]),
  X_2014_11_28  = as.matrix(TS_2014_11_28[,  covar_names]),
  X_2008_09_09  = as.matrix(TS_2008_09_09[,  covar_names]),
  X_2008_09_15  = as.matrix(TS_2008_09_15[,  covar_names]),
  X_2008_09_29  = as.matrix(TS_2008_09_29[,  covar_names])
)

## ========== 4) Specify pre- and post-shock lengths ==========
# Each window has 30 pre-shock days. We focus on a one-day shock effect.
pre_len  <- 30L
post_len <- 1L

# shock_time_vec: index of the last pre-shock observation in each series
shock_time_vec   <- rep(pre_len,  length(Y_series_list))   # all 30

# shock_length_vec: number of post-shock points used to estimate the shock effect
shock_length_vec <- rep(post_len, length(Y_series_list))   # all 1

## ========== 5) Quick sanity checks ==========
sapply(Y_series_list, length)  # should all be 60 (30 pre + 30 post)
```

```
##     Y_target Y_2008_03_17 Y_2014_11_28 Y_2008_09_09 Y_2008_09_15 Y_2008_09_29
##           60           60           60           60           60           60
```

```
shock_time_vec                  # all equal to 30
```

```
## [1] 30 30 30 30 30 30
```

```
shock_length_vec                # all equal to 1
```

```
## [1] 1 1 1 1 1 1
```

6.4 Postshock forecasting with donor balancing weights

We now implement the postshock forecasting procedure using the series lists constructed above. The target series is the COP window around the 9 March 2020 shock, and the donors are the five historical shock windows from 2008 and 2014. For each series we specify a one–day shock effect and a one–day–ahead forecast horizon ($k = 1$).

On the covariate side we work with five macro–financial variables: the S&P 500 index, WTI crude oil, the U.S. Dollar Index, the 3–month Treasury bill rate, and the VIX. In the call to `SynthPrediction()` we set `covariate_indices = NULL`, so all available covariates are used both in the ARIMAX specification and in constructing donor balancing weights (DBW). We allow for a seasonal ARIMA structure and switch on DBW via `use_dbw = TRUE`.

The function returns (i) the synthetic–control weights on the donor episodes for the target window, (ii) the estimated donor–specific shock effects, and (iii) several versions of the postshock forecast for the target series. In particular, the unadjusted forecast corresponds to the ARIMAX baseline without adding any shock effect, while the DBW–adjusted forecast adds a DBW–weighted combination of donor shock effects to this baseline.

```
## ================== Postshock forecast with DBW ==================

# GSPC_Close is the 1st column in covar_names, WTI_Close is the 2nd
# (kept here for reference if we later want to restrict to a subset)
covariate_indices <- c(1, 2)

out <- SynthPrediction(
  Y_series_list          = Y_series_list,           # list of outcome series (target + donors)
  covariates_series_list = covariates_series_list,  # list of corresponding covariate matrices
  shock_time_vec         = shock_time_vec,          # index of the last pre-shock day in each
  shock_length_vec       = shock_length_vec,        # length of the post-shock segment used fo
  k                      = 1,                       # forecast horizon: 1 day after the shock
  covariate_indices      = NULL,                    # use all available covariates (no sub-sel
  use_dbw                = TRUE,                    # turn on donor balancing weights (DBW)
  seasonal               = TRUE,                    # allow for a seasonal ARIMA structure
  plots                  = FALSE                    # do not produce plots in this call
)
```

```
##
## Iter: 1 fn: 6.3015    Pars:  0.000000009242 0.636603889381 0.000000001636 0.000000013083 0.3
## Iter: 2 fn: 6.3015    Pars:  0.0000000082101 0.6366038896691 0.0000000009315 0.000000011887
## solnp--> Completed in 2 iterations
```

```
## ================== Inspect key components of the output ==================

# Synthetic-control donor weights for the target series
out$linear_combinations
```

```
## [1] 8.210053e-09 6.366039e-01 9.314648e-10 1.188764e-08 3.633961e-01
```

```
# Estimated donor-specific shock effects (omega_j for each donor j)
out$meta$omega_vec
```

```
## [1] -1.771793 -5.205221 -7.269179 -4.317799 -6.352937
```

```
# Postshock forecasts for the target:
#   $unadjusted      = ARIMAX baseline (no shock effect added)
#   $adjusted        = baseline + DBW-weighted shock effect
#   $arithmetic_mean = baseline + simple average shock effect
out$predictions
```

```
## $unadjusted
## [1] 44.7338
##
## $adjusted
## [1] 39.1115
##
## $arithmetic_mean
## [1] 39.75041
```

```
## ================== Compare forecasts to the realised price ==================

# Extract the last pre-shock day (2020-03-06) and the shock day (2020-03-09)
COP_close[COP_close$Date %in% as.Date(c("2020-03-06", "2020-03-09")),
          c("Date", "COP_Close")]
```

```
##            Date COP_Close
## 5054 2020-03-06     45.33
## 5055 2020-03-09     34.07
```

```
# Compare the realised price on 2020-03-09 with the baseline
# and the DBW-adjusted postshock forecasts
c(
  actual     = COP_close$COP_Close[COP_close$Date == as.Date("2020-03-09")],
  unadjusted = out$predictions$unadjusted,
  adjusted   = out$predictions$adjusted
)
```

```
##     actual unadjusted   adjusted
##    34.0700    44.7338    39.1115
```

6.5 Summary and discussion

In this empirical section we implemented the postshock forecasting framework of Section 3 on the ConocoPhillips data set. The COVID-19 shock on 9 March 2020 was treated as the target episode, while five earlier crisis episodes from 2008 and 2014 served as donor windows. For each episode we constructed a 60-day window (30 pre-shock and 30 post-shock trading days), applied CPI-based inflation adjustment, and extracted the COP closing price together with five macro-financial covariates as inputs to the procedure.

Using these windows, we formed the lists `Y_series_list` and `covariates_series_list` and specified a one-day shock effect with a one-step-ahead forecast horizon. The function `SynthPrediction()` was then applied with donor balancing weights turned on. The optimisation routine placed most of the synthetic-control weight on a subset of the donor episodes, reflecting their similarity to the March 2020 event in terms of pre-shock covariates. The estimated donor-specific shock effects were all negative, corresponding to large downward jumps in COP around each crisis date.

The postshock forecasts illustrate the contribution of donor information relative to the ARIMAX baseline. The unadjusted forecast, which is obtained from the pre-shock ARIMAX fit without adding any shock effect, lies substantially above the realised COP price on 9 March 2020. Once we add the DBW-weighted shock effect, the adjusted forecast moves closer to the realised value, reducing the forecast error relative to the baseline. The arithmetic-mean adjustment, which simply averages donor shock effects without balancing, also improves upon the unadjusted forecast but performs slightly worse than the DBW-adjusted version. This comparison highlights the benefit of reweighting donors so that their pre-shock covariate profiles match that of the target episode.

Overall, this case study shows that the proposed postshock framework can effectively transfer information from historical shock episodes to a new target event. The implementation is modular: by changing only the input windows and the choice of covariates, the same code can be applied to other firms, macroeconomic indicators, or volatility series. Extending the empirical analysis to the volatility component using `auto_garchx()` and `SynthVolForecast()` is a natural next step, but is left for future work.

7 References

Lin, Jilei, and Daniel J. Eck (2021). "Minimizing post-shock forecasting error through aggregation of outside information." *International Journal of Forecasting*, 37(4), 1710–1727. https://doi.org/10.1016/j.ijforecast.2021.03.010

Sucarrat, Genaro (2021). "garchx: Flexible and Robust GARCH-X Modelling." *The R Journal*, 13(1), 276–291.