Autoregressive Strategies

FRE7241, Spring 2025

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

May 12, 2025

The Bid-Ask Spread

The *bid-ask spread* is the difference between the best ask (offer) price minus the best bid price in the market.

The *bid-ask spread* can be estimated from the differences between the execution prices of consecutive buy and sell market orders (roundtrip trades).

Market orders are orders to buy or sell a stock immediately at the best available price in the market. Market orders guarantee that the trade will be executed, but they do not guarantee the execution price. Market orders are subject to the *bid-ask spread*.

Limit orders are orders to buy or sell a stock at the limit price or better (the investor sets the limit price). Limit orders do not guarantee that the trade will be executed, but they guarantee the execution price. Limit orders are placed only for a certain time when they are "live".

Market orders are executed by matching them with live limit orders through a matching engine at an exchange.

The *bid-ask spread* for many liquid ETFs is about 1 basis point. For example the *XLK ETF*

The most liquid *SPY ETF* usually trades at a *bid-ask spread* of only one tick (cent=$0.01, or about 0.2 basis points).

In reality the *bid-ask spread* is not static and depends on many factors, such as market liquidity (trading volume), volatility, and the time of day.

```
> # Load the roundtrip trades
> dtable <- data.table::fread("/Users/jerzy/Develop/lecture_slides/c
> nrows <- NROW(dtable)
> class(dtable$timefill)
> # Sort the trades according to the execution time
> dtable <- dtable[order(dtable$timefill)]
> # Calculate the dollar bid-ask spread
> pricebuy <- dtable$price[dtable$side == "buy"]
> pricesell <- dtable$price[dtable$side == "sell"]
> bidask <- mean(pricebuy-pricesell)
> # Calculate the percentage bid-ask spread
> bidask/mean(pricesell)
```

Autocorrelations of Stock Returns

Daily stock returns often exhibit negative autocorrelations at one-day lags.

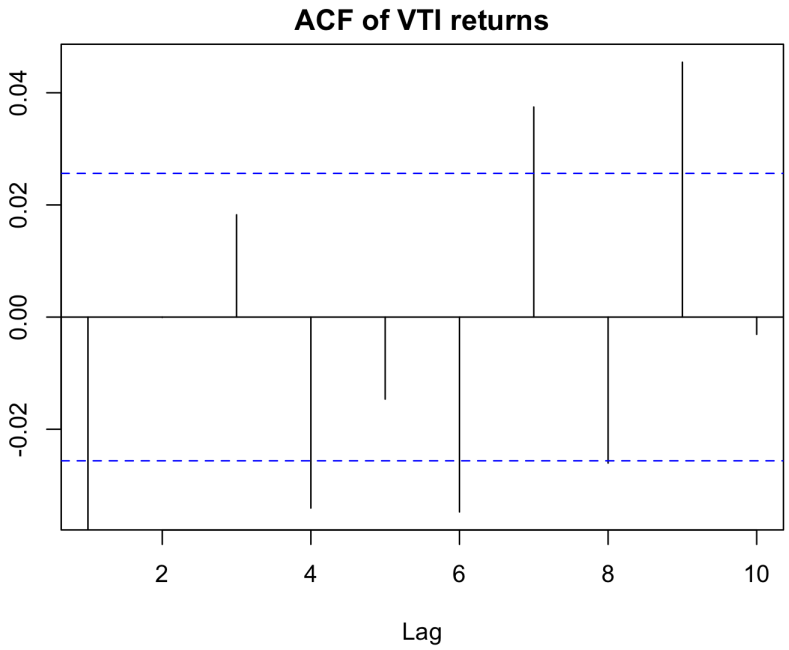
The *VTI* returns appear to have some small, yet significant negative autocorrelations at `lag=1`, and some positive autocorrelations at larger lags.

The *autocorrelation* of lag k of a time series of returns r_t is equal to:

$$\rho_k = \frac{\sum_{t=k+1}^{n}(r_t - \bar{r})(r_{t-k} - \bar{r})}{(n-k)\,\sigma^2}$$

The function `rutils::plot_acf()` calculates and plots the autocorrelations of a time series.

But the visual inspection of the *ACF* plot alone is not enough to test whether autocorrelations are statistically significant or not.




```
> # Calculate the daily VTI percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Calculate the autocorrelations of daily VTI returns
> rutils::plot_acf(retp, lag=10, main="ACF of VTI returns")
```

Daily Contrarian Strategy

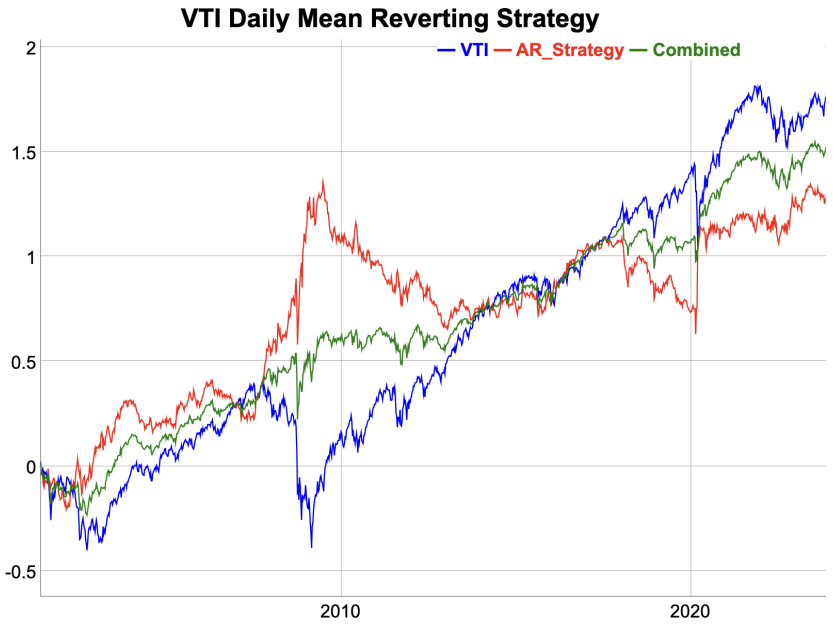
The daily contrarian strategy buys or sells short \$1 of stock at the end of each day (depending on the sign of the previous daily return), and holds the position until the next day.

If the previous daily return was positive, it sells short \$1 of stock. If the previous daily return was negative, it buys \$1 of stock.

The contrarian strategy has lower returns than *VTI*, but it has a very low correlation to *VTI*, so it has a positive *alpha*.

Thanks to its low correlation to *VTI*, the contrarian strategy provides diversification of risk, and combined with *VTI*, a higher *Sharpe* ratio than *VTI* alone (it has a positive marginal *alpha*).

Strategies which have low or negative correlations to stocks, can contribute a significant marginal *alpha*, even if they have low returns.




```
> # Simulate the contrarian strategy
> posv <- -rutils::lagit(sign(retp), lagg=1)
> pnls <- retp*posv
> # Subtract transaction costs from the pnls
> bidask <- 0.0001 # The bid-ask spread is equal to 1 basis point
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
> # Calculate the strategy beta and alpha
> betac <- cov(pnls, retp)/var(retp)
> alphac <- mean(pnls) - betac*mean(retp)
```

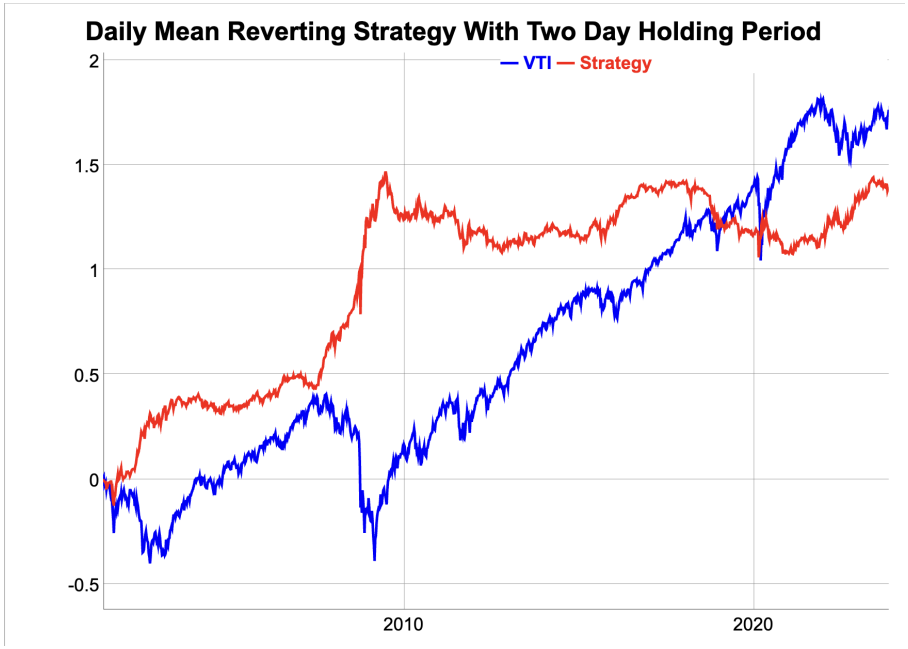
```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of contrarian strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="VTI Daily Contrarian Strategy") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dyLegend(show="always", width=300)
```

Daily Contrarian Strategy With a Holding Period

The daily contrarian strategy can be improved by combining the daily returns from the previous two days. This is equivalent to holding the position for two days, instead of rolling it daily.

The daily contrarian strategy with a holding period performs better than the simple daily strategy because of risk diversification.

```
> # Simulate contrarian strategy with two day holding period
> posv <- -rutils::roll_sum(sign(retp), lookb=2)/2
> pnls <- retp*rutils::lagit(posv)
```



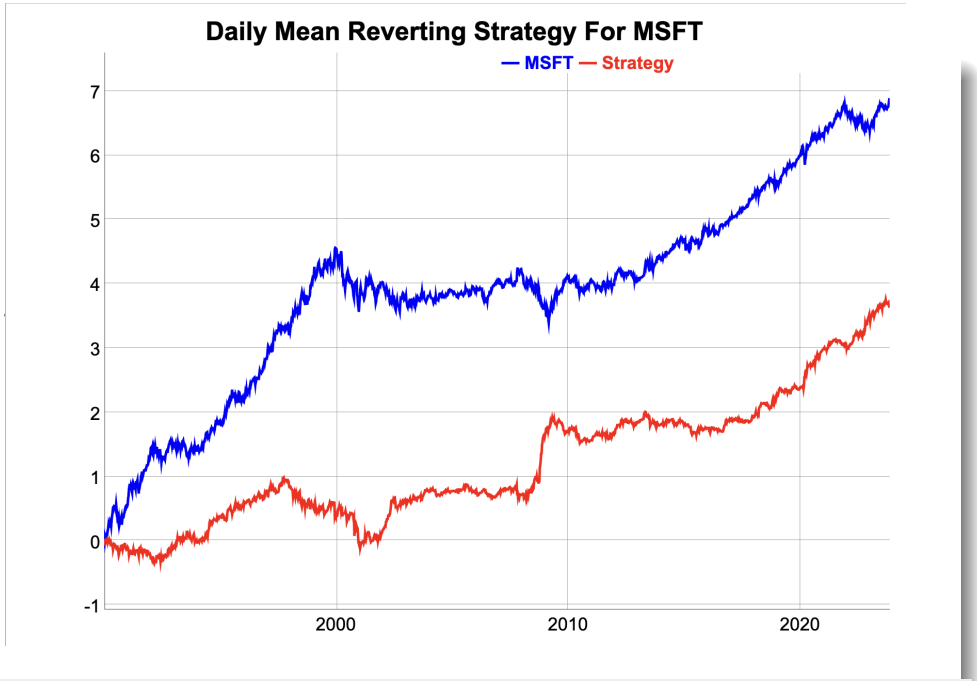

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of contrarian strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Daily Contrarian Strategy With Two Day Holding Period") %>
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Daily Contrarian Strategy For Stocks

Some daily stock returns exhibit stronger negative autocorrelations than ETFs.

But the daily contrarian strategy doesn't perform well for many stocks.

```
> # Load daily S&P500 stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> retp <- na.omit(retstock$MSFT)
> rutils::plot_acf(retp)
> # Simulate contrarian strategy with two day holding period
> posv <- -rutils::roll_sum(sign(retp), lookb=2)/2
> pnls <- retp*rutils::lagit(posv)
```



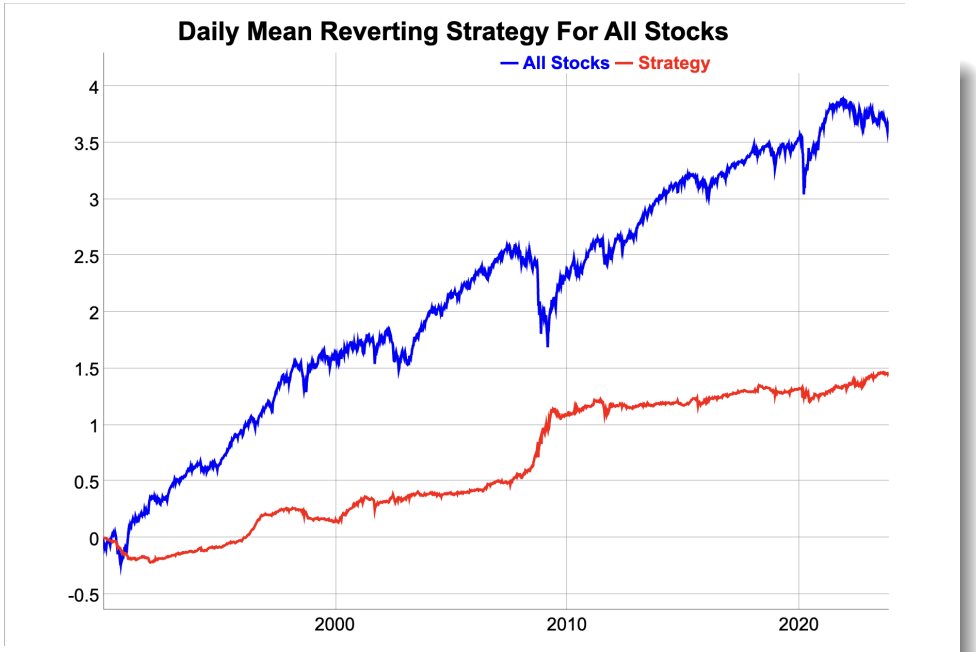

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of contrarian strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Daily Contrarian Strategy For MSFT") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Daily Contrarian Strategy For All Stocks

The combined daily contrarian strategy for all *S&P500* stocks performed well prior to and during the 2008 financial crisis, but was flat afterwards.

Averaging the stock returns using the function `rowMeans()` with `na.rm=TRUE` is equivalent to rebalancing the portfolio so that stocks with `NA` returns have zero weight.

```
> # Simulate contrarian strategy for all S&P500 stocks
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnll <- mclapply(retstock, function(retp) {
+   retp <- na.omit(retp)
+   posv <- -rutils::roll_sum(sign(retp), lookb=2)/2
+   retp*rutils::lagit(posv)
+ }, mc.cores=ncores)  # end mclapply
> pnls <- do.call(cbind, pnll)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> # Calculate the average returns of all S&P500 stocks
> datev <- zoo::index(retstock)
> datev <- datev[-1]
> indeks <- rowMeans(retstock, na.rm=TRUE)
> indeks <- indeks[-1]
```

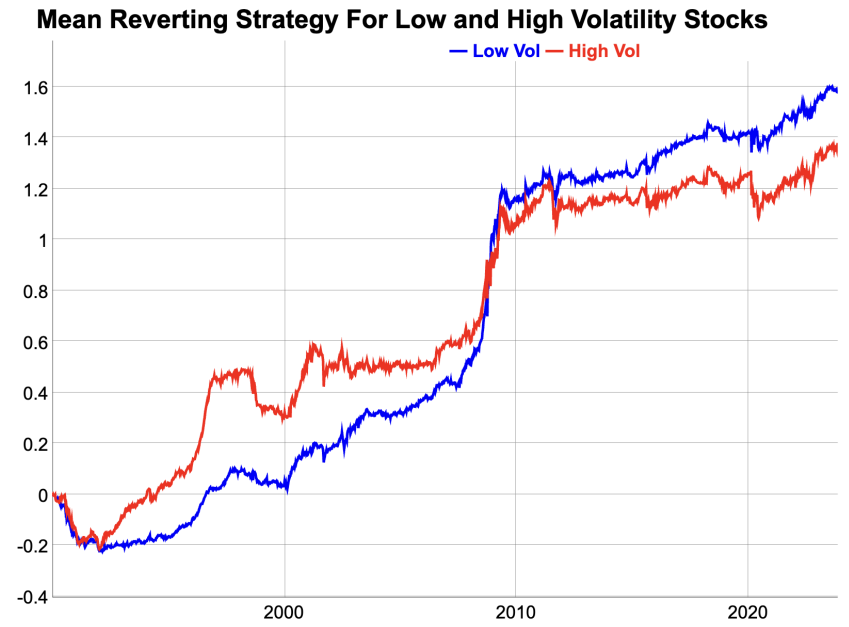



```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("All Stocks", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of contrarian strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Daily Contrarian Strategy For All Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Contrarian Strategy For Low and High Volatility Stocks

The daily contrarian strategy performs better for low volatility stocks than for high volatility stocks.

```
> # Calculate the stock volatilities
> volv <- mclapply(retstock, function(retp) {
+   sd(na.omit(retp))
+ }, mc.cores=ncores)  # end mclapply
> volv <- do.call(c, volv)
> # Calculate the median volatility
> medianv <- median(volv)
> # Calculate the pnls for low volatility stocks
> pnlovol <- do.call(cbind, pnll[volv < medianv])
> pnlovol <- rowMeans(pnlovol, na.rm=TRUE)
> # Calculate the pnls for high volatility stocks
> pnlhivol <- do.call(cbind, pnll[volv >= medianv])
> pnlhivol <- rowMeans(pnlhivol, na.rm=TRUE)
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(pnlovol, pnlhivol)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Low Vol", "High Vol")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of contrarian strategy
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Contrarian Strategy For Low and High Volatility Stocks") %
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

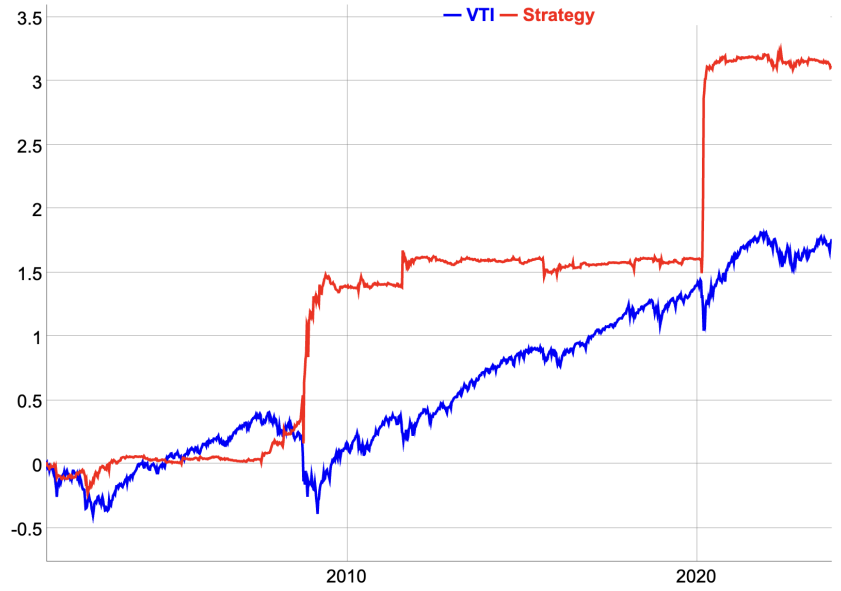
The EMA Mean-Reversion Strategy

The *EMA* mean-reversion strategy holds either long stock positions or short positions proportional to minus the trailing *EMA* of past returns.

The strategy adjusts its stock position at the end of each day, just before the close of the market.

The strategy takes very large positions in periods of high volatility, when returns are large and highly anti-correlated.

The strategy makes profits mostly in periods of high volatility, but otherwise it's not very profitable.




```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Calculate the EMA returns recursively using C++ code
> retma <- HighFreq::run_mean(retp, lambda=0.1)
> # Calculate the positions and PnLs
> posv <- -rutils::lagit(retma, lagg=1)
> pnls <- retp*posv
> # Subtract transaction costs from the pnls
> bidask <- 0.0001 # The bid-ask spread is equal to 1 basis point
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of contrarian strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="VTI EMA Daily Contrarian Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

The EMA Mean-Reversion Strategy Scaled By Volatility

Dividing the returns by their trailing volatility reduces the effect of time-dependent volatility.

Scaling the returns by their trailing volatilities reduces the profits in periods of high volatility, but doesn't improve profits in periods of low volatility.

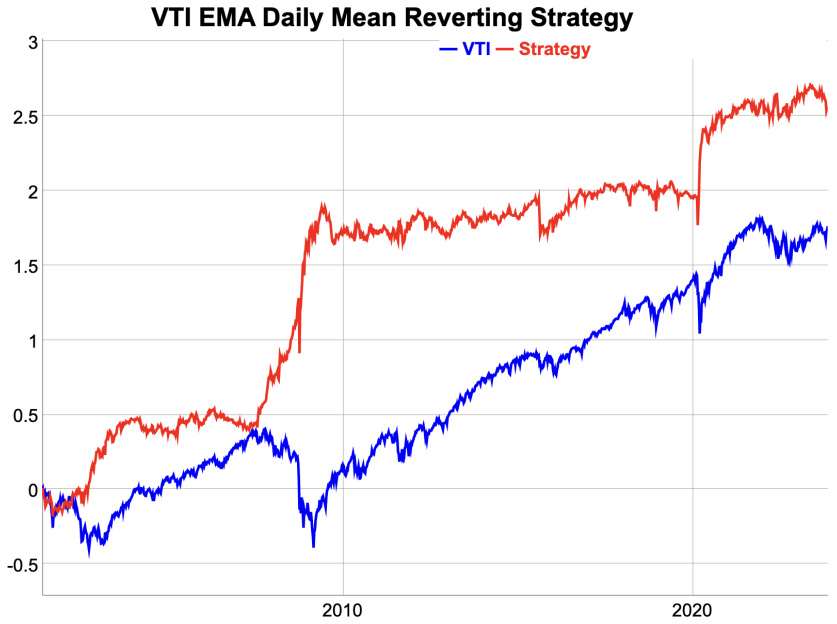
The function `HighFreq::run_var()` calculates the trailing mean and variance of the returns r_t, by recursively weighting the past variance estimates σ^2_{t-1}, with the squared differences of the returns minus their trailing means $(r_t - \bar{r}_t)^2$, using the decay factor λ:

$$\bar{r}_t = \lambda \bar{r}_{t-1} + (1 - \lambda) r_t$$

$$\sigma^2_t = \lambda^2 \sigma^2_{t-1} + (1 - \lambda^2)(r_t - \bar{r}_t)^2$$

Where $\bar{r}_t$ and σ^2_t are the trailing mean and variance at time t.

The decay factor λ determines how quickly the mean and variance estimates are updated, with smaller values of λ producing faster updating, giving more weight to recent prices, and vice versa.




```
> # Calculate the EMA returns and volatilities
> volv <- HighFreq::run_var(retp, lambda=0.5)
> retma <- volv[, 1]
> volv <- sqrt(volv[, 2])
> # Scale the returns by their trailing volatility
> retsc <- retp/volv
> # Calculate the positions and PnLs
> posv <- -rutils::lagit(retsc, lagg=1)
> pnls <- retp*posv
```

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of contrarian strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="VTI EMA Daily Contrarian Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Autoregressive Model of Stock Returns

The stock returns r_t can be fitted into an *autoregressive* model *AR(n)* with a constant intercept term φ_0:

$$r_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n} + \varepsilon_t$$

The *residuals* ε_t are assumed to be normally distributed, independent, and stationary.

The autoregressive model can be written in matrix form as:

$$\mathbf{r} = \varphi \, \mathbb{P} + \varepsilon$$

Where $\varphi = \{\varphi_0, \varphi_1, \varphi_2, \ldots \varphi_n\}$ is the vector of autoregressive coefficients.

The *autoregressive* model is equivalent to *multivariate* linear regression, with the *response* equal to the returns $\mathbf{r}$, and the columns of the *predictor matrix* $\mathbb{P}$ equal to the lags of the returns.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> nrows <- NROW(retp)
> # Define the response and predictor matrices
> respv <- retp
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> # Add constant column for intercept coefficient phi0
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
```

Forecasting Stock Returns Using Autoregressive Models

The fitted autoregressive coefficients φ are equal to the *response* $\mathbf{r}$ multiplied by the inverse of the *predictor matrix* $\mathbb{P}$:

$$\varphi = \mathbb{P}^{-1} \mathbf{r}$$

The *in-sample* autoregressive forecasts of the returns are calculated by multiplying the predictor matrix by the fitted *AR* coefficients:

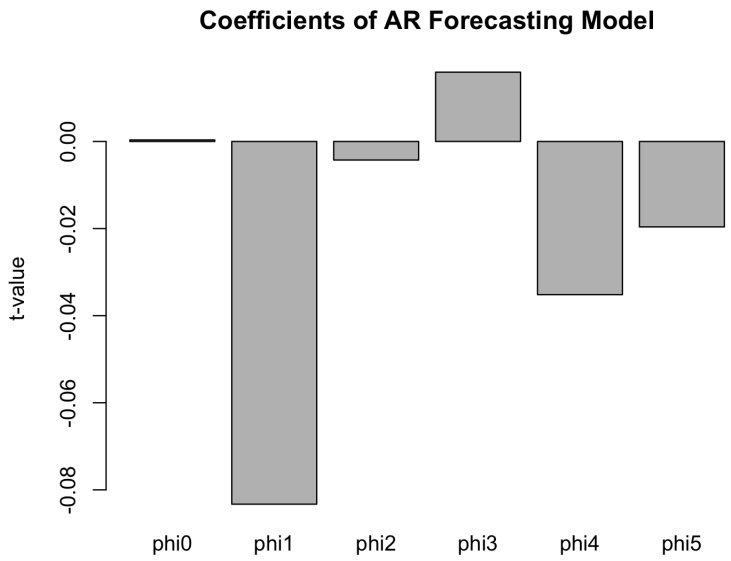
$$f_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n}$$

For *VTI* returns, the intercept coefficient φ_0 has a small positive value, while the first autoregressive coefficient φ_1 has a small negative value.

This means that the autoregressive forecasting model is a combination of a static long stock position, plus a mean-reverting model which switches its stock position to the reverse of the previous day's return.

The function `MASS::ginv()` calculates the generalized inverse of a matrix.

```
> # Calculate the fitted autoregressive coefficients
> predinv <- MASS::ginv(predm)
> coeff <- drop(predinv %*% respv)
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
```




```
> # Plot the AR coefficients
> names(coeff) <- colnames(predm)
> barplot(coeff, xlab="", ylab="coeff", col="grey",
+   main="Coefficients of AR Forecasting Model")
```

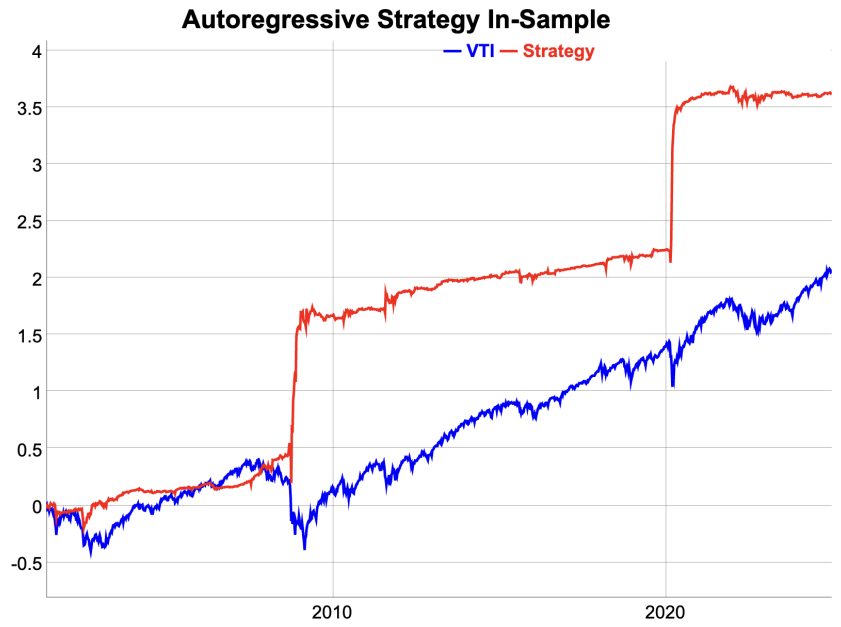
Autoregressive Strategy In-Sample

The first step in strategy development is optimizing it in-sample, even though in practice it can't be implemented. Because a strategy can't perform well out-of-sample if it doesn't perform well in-sample.

The autoregressive strategy invests dollar amounts of *VTI* stock proportional to the in-sample forecasts.

The in-sample autoregressive strategy performs well during periods of high volatility, but not as well in low volatility periods.

The dollar allocations of *VTI* stock are too large in periods of high volatility, which causes over-leverage and higher risk.




```
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> # costv <- 0.5*bidask*abs(rutils::diffit(posv))
> # pnls <- (pnls - costv)
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

The t-values of the Autoregressive Coefficients

The forecast residuals are equal to the differences between the return forecasts minus the actual returns: $\varepsilon = f_t - r_t$

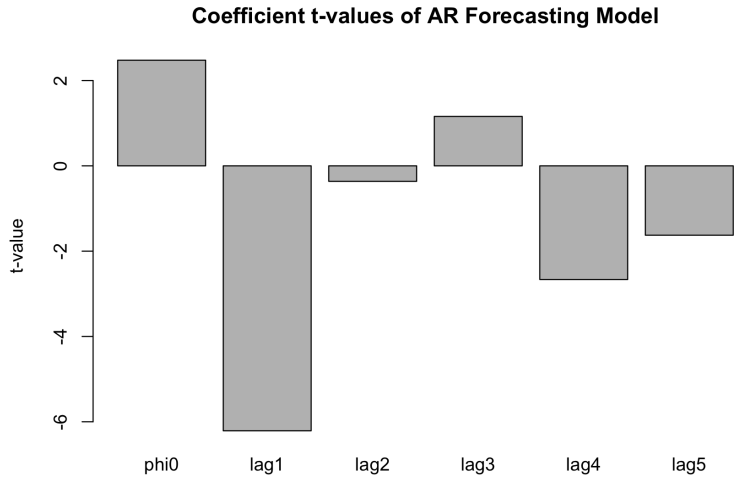
The variance of the autoregressive coefficients σ^2_{φ} is equal to the variance of the forecast residuals σ^2_{ε} divided by the squared *predictor matrix* $\mathbb{P}$:

$$\sigma^2_{\varphi} = \sigma^2_{\varepsilon}(\mathbb{P}^T\mathbb{P})^{-1}$$

The t-values of the autoregressive coefficients are equal to the coefficient values divided by their volatilities:

$$\varphi_{tval} = \frac{\varphi}{\sigma_{\varphi}}$$

The intercept coefficient φ_0 and the first autoregressive coefficient φ_1 have statistically significant t-values.




```
> # Calculate the residuals (forecast errors)
> resids <- (fcasts - respv)
> # The residuals are orthogonal to the predictors and the forecast
> round(cor(resids, fcasts), 6)
> round(sapply(predm[, -1], function(x) cor(resids, x)), 6)
> # Calculate the variance of the residuals
> varv <- sum(resids^2)/(nrows-NROW(coeff))
```

```
> # Calculate the predictor matrix squared
> pred2 <- crossprod(predm)
> # Calculate the covariance matrix of the AR coefficients
> covmat <- varv*MASS::ginv(pred2)
> coefsd <- sqrt(diag(covmat))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coefsd)
> names(coefft) <- colnames(predm)
> # Plot the t-values of the AR coefficients
> barplot(coefft, xlab="", ylab="t-value", col="grey",
+   main="Coefficient t-values of AR Forecasting Model")
```

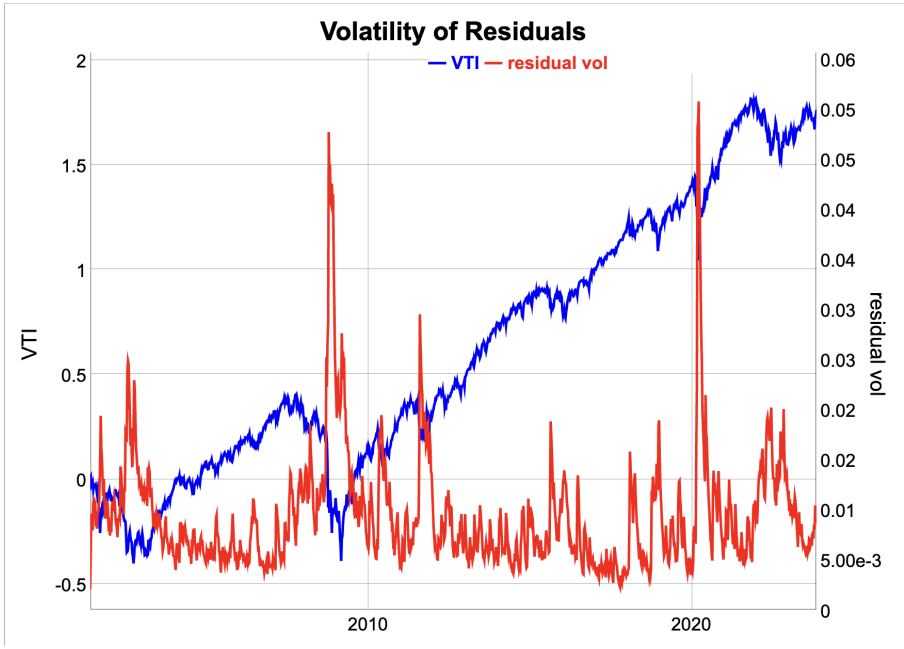
Residuals of Autoregressive Forecasting Model

The autoregressive model assumes stationary returns and residuals, with similar volatility over time.

In reality stock volatility is highly time dependent, so the volatility of the residuals is also time dependent.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Calculate the trailing volatility of the residuals
> residv <- sqrt(HighFreq::run_var(resids, lambda=0.9)[, 2])
```




```
> # Plot dygraph of volatility of residuals
> datav <- cbind(cumsum(retp), residv)
> colnames(datav) <- c("VTI", "residual vol")
> endw <- rutils::calc_endpoints(datav, interval="weeks")
> dygraphs::dygraph(datav[endw], main="Volatility of Residuals") %>%
+   dyAxis("y", label="VTI", independentTicks=TRUE) %>%
+   dyAxis("y2", label="residual vol", independentTicks=TRUE) %>%
+   dySeries(name="VTI", axis="y", strokeWidth=2, col="blue") %>%
+   dySeries(name="residual vol", axis="y2", strokeWidth=2, col="red
+   dyLegend(show="always", width=300)
```

AR Strategy With Forecasts Scaled By Volatility

The in-sample autoregressive strategy performs well during periods of high volatility, but not as well in low volatility periods.

The dollar allocations of *VTI* stock are too large in periods of high volatility, which causes over-leverage and higher risk.

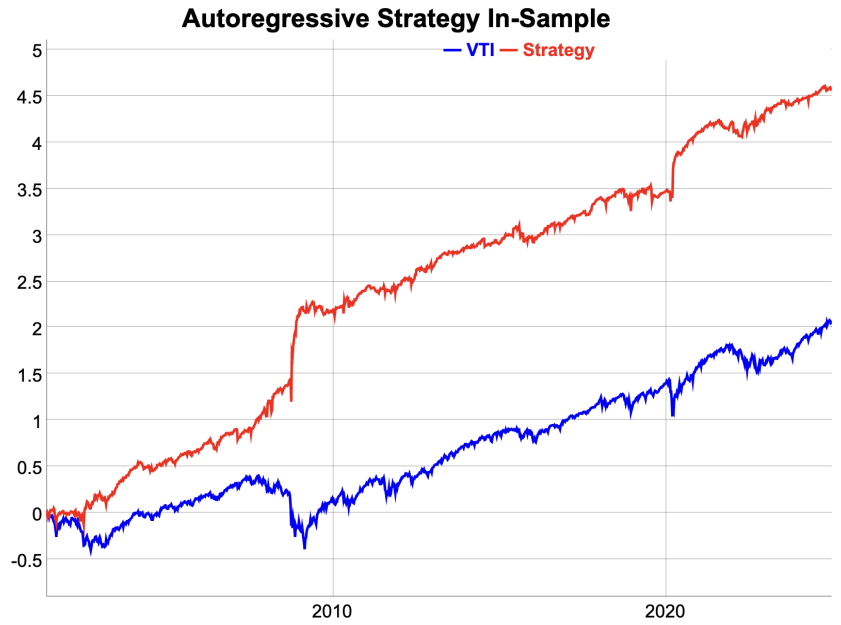
The leverage can be reduced by scaling (dividing) the forecasts by their trailing volatility.

This makes the performance more uniform, at the expense of the performance during periods of high volatility.

The tradeoff is that the strategy will not perform as well in periods of high volatility, but it will be less risky.

Whether to scale the forecasts or not depends on the investment objectives and risk tolerance of the investor.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.




```
> # Scale the forecasts by their volatility
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> # costv <- 0.5*bidask*abs(rutils::diffit(posv))
> # pnls <- (pnls - costv)
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

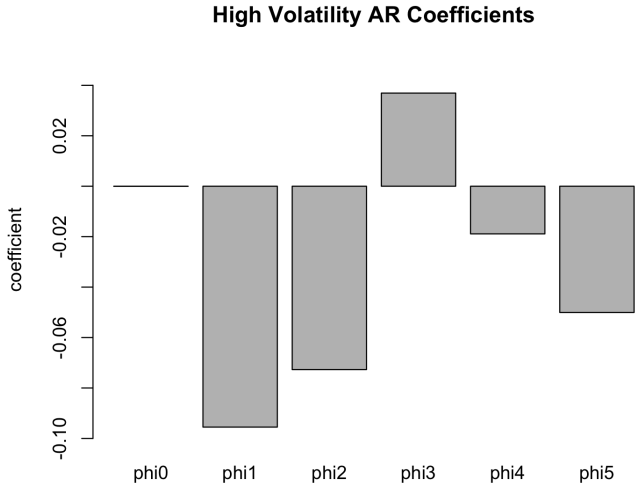
Autoregressive Coefficients in Periods of Low and High Volatility

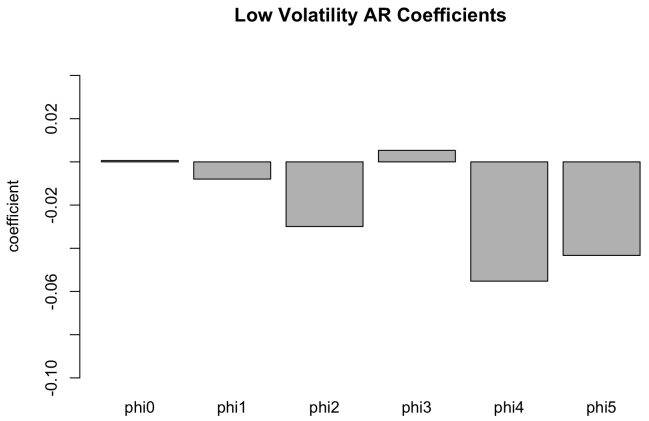
The autoregressive model assumes stationary returns and residuals, with similar volatility over time. In reality stock volatility is highly time dependent.

The autoregressive coefficients in periods of high volatility are very different from those under low volatility.

In periods of high volatility, there are larger negative autocorrelations than in low volatility.

```
> # Calculate the high volatility AR coefficients
> respv <- retp["2008/2011"]
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> predinv <- MASS::ginv(predm)
> coeffh <- drop(predinv %*% respv)
> names(coeffh) <- colnames(predm)
> barplot(coeffh, main="High Volatility AR Coefficients",
+   col="grey", xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
> # Calculate the low volatility AR coefficients
> respv <- retp["2012/2019"]
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> predinv <- MASS::ginv(predm)
> coeffl <- drop(predinv %*% respv)
> names(coeffl) <- colnames(predm)
> barplot(coeffl, main="Low Volatility AR Coefficients",
+   xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
```

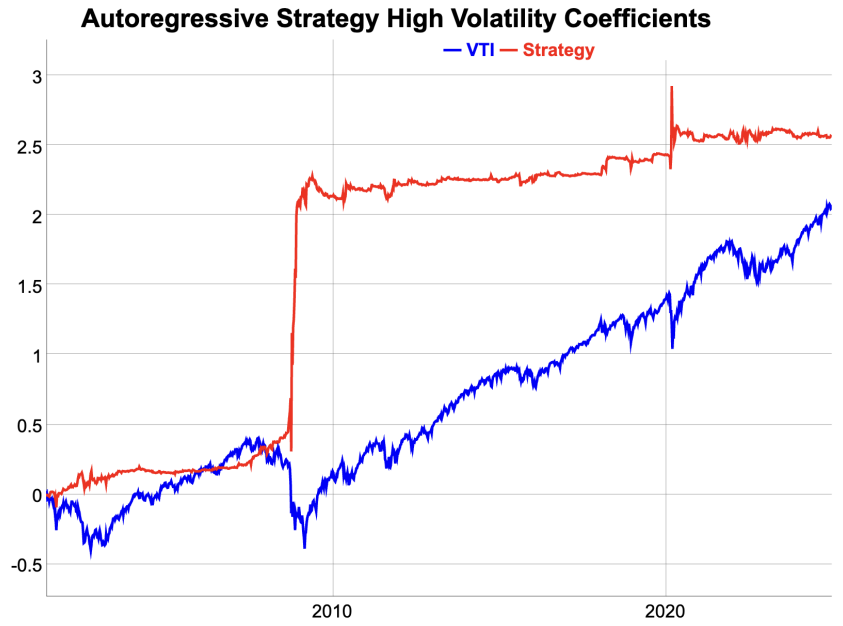


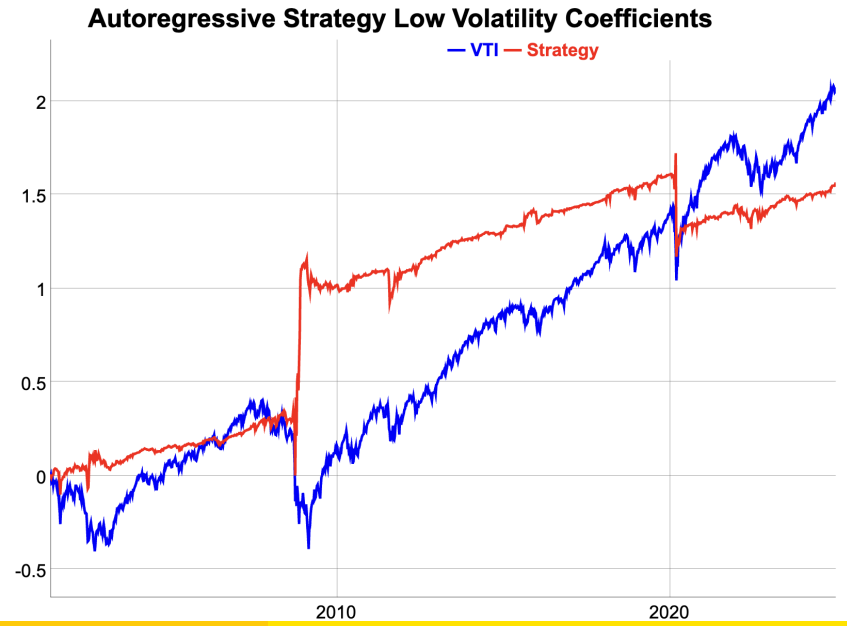


Performance of Low and High Volatility Autoregressive Coefficients

The autoregressive coefficients obtained from periods of high volatility are overfitted and only perform well in periods of high volatility. Similarly the low volatility coefficients.

```
> # Calculate the pnls for the high volatility AR coefficients
> predm <- lapply(1:orderp, rutils::lagit, input=retp)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> fcasts <- predm %*% coeffh
> pnlh <- retp*fcasts
> pnlh <- pnlh*sd(retp[retp<0])/sd(pnlh[pnlh<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnlh)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy High Volatility Coefficients") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Calculate the pnls for the low volatility AR coefficients
> fcasts <- predm %*% coeffl
> pnll <- retp*fcasts
> pnll <- pnll*sd(retp[retp<0])/sd(pnll[pnll<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnll)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy Low Volatility Coefficients") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```





draft: Regime Switching Autoregressive Strategy

Run two competing autoregressive models for the low and high volatility regimes, with low and high volatility coefficients.

Calculate the Kalman gains from the trailing square forecast errors, and apply the Kalman gains to the forecasts.

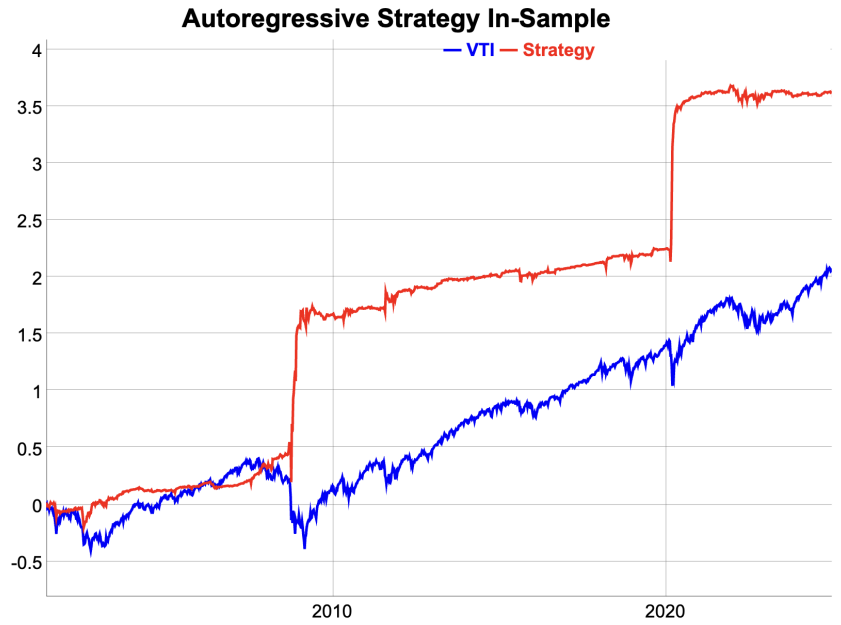
Doesn't work because the square forecast errors are similar.

The autoregressive strategy invests dollar amounts of *VTI* stock proportional to the in-sample forecasts.

The in-sample autoregressive strategy performs well during periods of high volatility, but not in low volatility periods.

The dollar allocations of *VTI* stock are too large in periods of high volatility, which causes over-leverage and higher risk.

The autoregressive model assumes stationary returns and residuals, with similar volatility over time. In reality stock volatility is highly time dependent.




```
> # Define the response and predictor matrices
> respv <- retp
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predinv <- MASS::ginv(predm)
> # Simulate strategy with high volatility AR coefficients
> fcasts <- drop(predm %*% coeffh)
>
```

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

The *Winsor* Function

Some models produce very large dollar allocations, leading to large portfolio leverage (dollars invested divided by the capital).

The *winsor function* maps the *model weight* w into the dollar amount for investment. The hyperbolic tangent function can serve as a winsor function:

$$W(x) = \frac{\exp(\lambda w) - \exp(-\lambda w)}{\exp(\lambda w) + \exp(-\lambda w)}$$

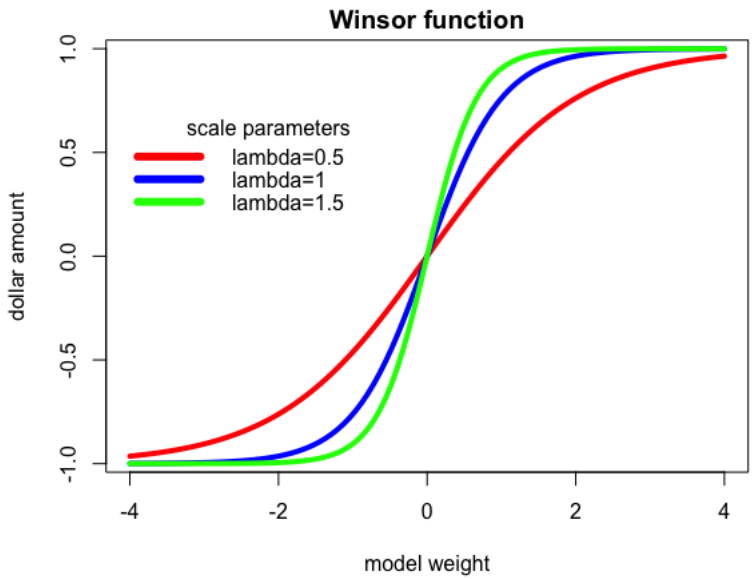
Where λ is the scale parameter.

The hyperbolic tangent is close to linear for small values of the *model weight* w, and saturates to $+1\$/-1\$$ for very large positive and negative values of the *model weight*.

The saturation effect limits (caps) the leverage in the strategy to $+1\$/-1\$$.

For very small values of the scale parameter λ, the invested dollar amount is linear for a wide range of *model weights*. So the strategy is mostly invested in dollar amounts proportional to the *model weights*.

For very large values of the scale parameter λ, the invested dollar amount jumps from $-1\$$ for negative *model weights* to $+1\$$ for positive *model weight* values. So the strategy is invested in either $-1\$$ or $+1\$$ dollar amounts.



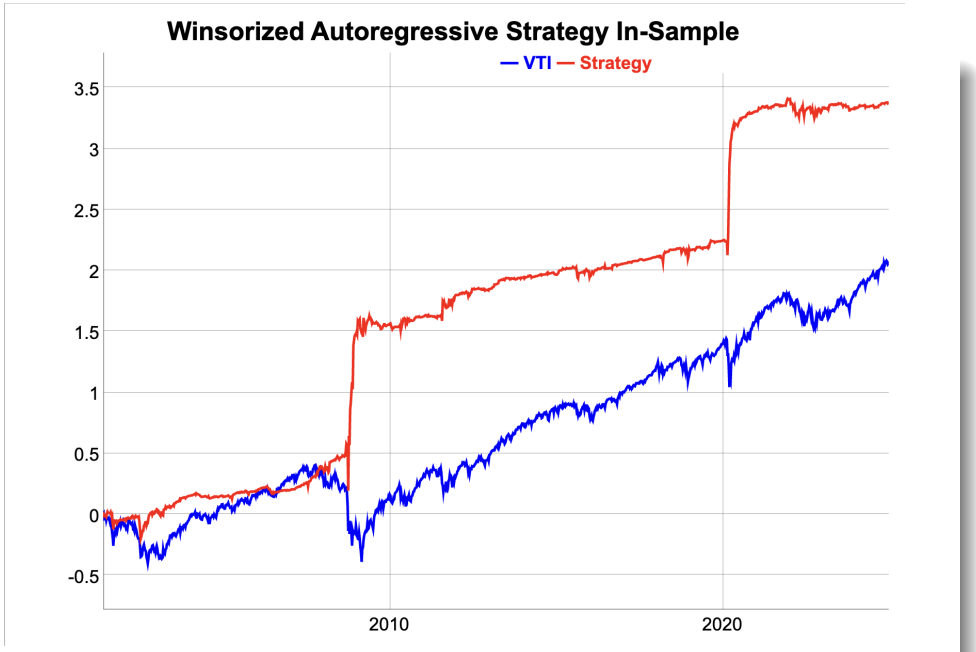

```
> lambdav <- c(0.5, 1, 1.5)
> colorv <- c("red", "blue", "green")
> # Define the winsor function
> winsorfun <- function(retp, lambdaf) tanh(lambdaf*retp)
> # Plot three curves in loop
> for (indeks in 1:3) {
+   curve(expr=winsorfun(x, lambda=lambdav[indeks]),
+ xlim=c(-4, 4), type="l", lwd=4,
+ xlab="model weight", ylab="dollar amount",
+ col=colorv[indeks], add=(indeks>1))
+ }  # end for
> # Add title and legend
> title(main="Winsor function", line=0.5)
> legend("topleft", title="scale parameters\n",
+    paste("lambdaf", lambdav, sep="="), inset=0.0, cex=1.0,
+    lwd=6, bty="n", y.intersp=0.3, lty=1, col=colorv)
```

Winsorized Autoregressive Strategy

The performance of the autoregressive strategy can be improved by fitting its coefficients using the *winsorized returns*, to reduce the effect of time-dependent volatility.

The performance can also be improved by *winsorizing* the forecasts, by reducing the leverage due to very large forecasts.

```
> # Winsorize the VTI returns
> retw <- winsorfun(retp/0.01, lambda=0.1)
> # Define the response and predictor matrices
> predm <- lapply(1:orderp, rutils::lagit, input=retw)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% retw
> # Calculate the scaled in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> # Winsorize the forecasts
> # fcasts <- winsorfun(fcasts/mad(fcasts), lambda=1.5)
```




```
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Winsorized Autoregressive Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

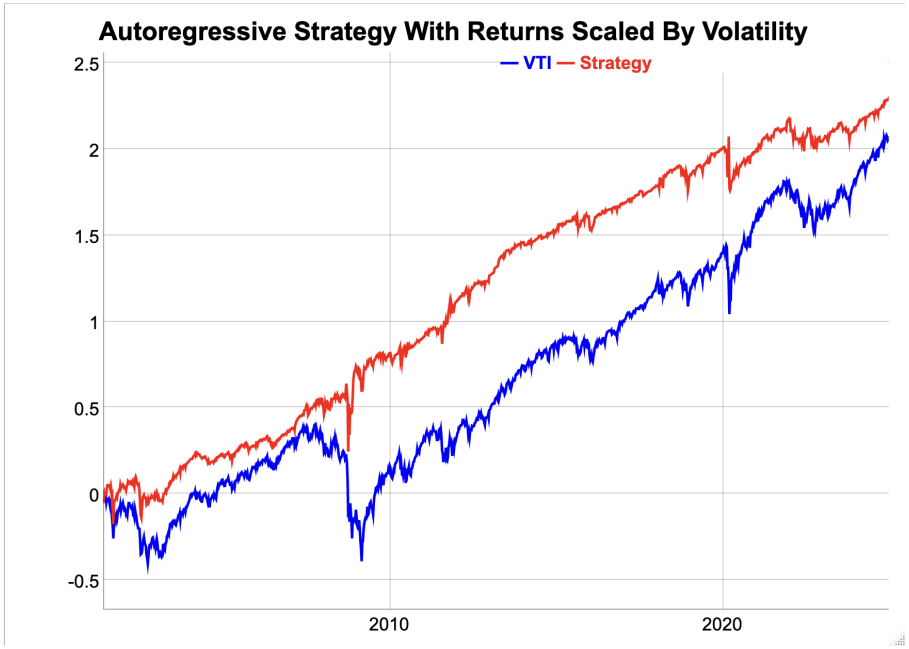
Autoregressive Strategy With Returns Scaled By Volatility

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns divided by their trailing volatility.

Dividing the returns by their trailing volatility reduces the effect of time-dependent volatility.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Scale the returns by their trailing volatility
> varv <- HighFreq::run_var(retp, lambda=0.99)[, 2]
> retsc <- retp/sqrt(varv)
> # Calculate the AR coefficients
> predm <- lapply(1:orderp, rutils::lagit, input=retsc)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% retsc
> # Calculate the scaled in-sample forecasts of VTI
> fcasts <- predm %*% coeff
```



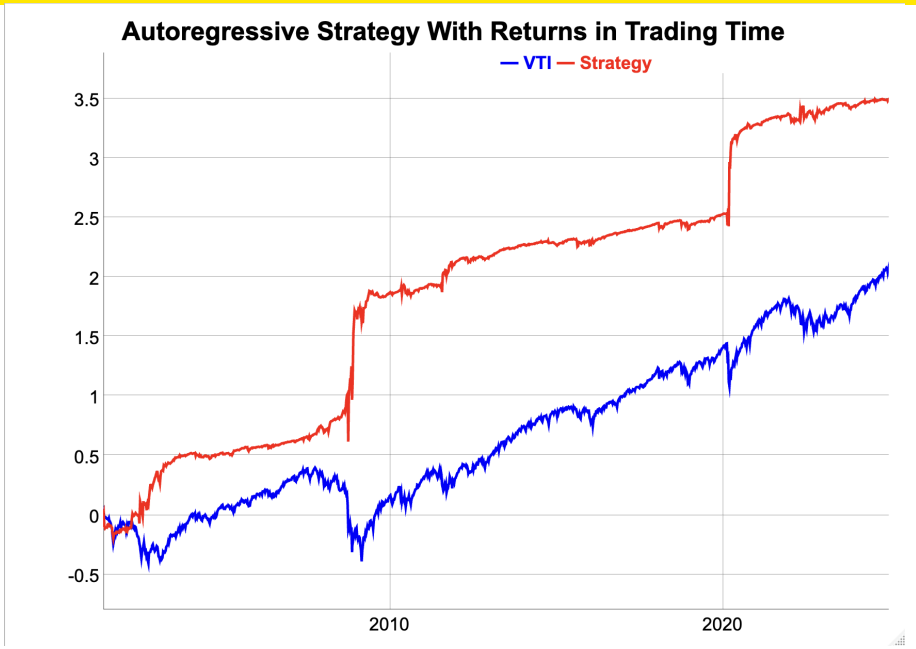

```
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy With Returns Scaled By Volatility"
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Autoregressive Strategy in Trading Time

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns divided by the trading volumes (returns in *trading time*).

Dividing the returns by the trading volumes reduces the effect of time-dependent volatility.

```
> # Calculate VTI returns and trading volumes
> ohlc <- rutils::etfenv$VTI
> datev <- zoo::index(ohlc)
> nrows <- NROW(ohlc)
> closep <- quantmod::Cl(ohlc)
> colnames(closep) <- "VTI"
> retp <- rutils::diffit(log(closep))
> volumv <- quantmod::Vo(ohlc)
> # Scale the returns using volume clock to trading time
> volumr <- HighFreq::run_mean(volumv, lambda=0.8)
> respv <- retp*volumr/volumv
> # Calculate the AR coefficients
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> predinv <- MASS::ginv(predm)
> coeff <- drop(predinv %*% respv)
> # Calculate the scaled in-sample forecasts of VTI
> fcasts <- predm %*% coeff
```




```
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv["2010/"], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy With Returns in Trading Time") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Mean Squared Error of the Autoregressive Forecasting Model

The accuracy of a forecasting model can be measured using the *mean squared error* and the *correlation*.

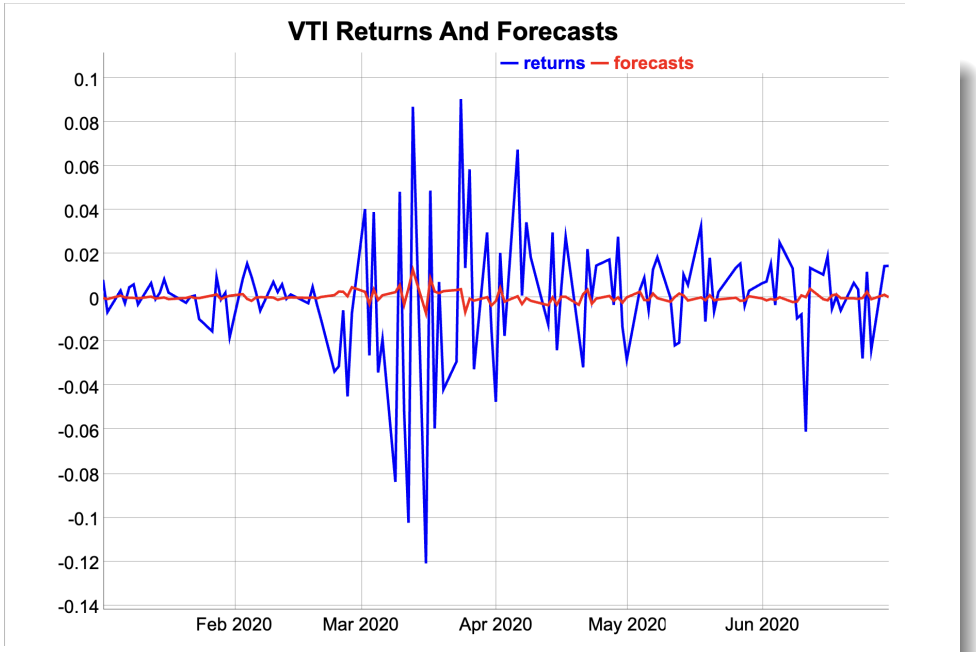
The mean squared error (*MSE*) of a forecasting model is the average of the squared forecasting errors ε_i, equal to the differences between the *forecasts* f_t minus the actual values r_t: $\varepsilon_i = f_t - r_t$:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(r_t - f_t)^2$$

The *MSE* is equivalent to the *residual sum of squares* (*RSS*) of the regression model.

The volatility of the forecasts is much lower than the returns, because of the low correlation between the forecasts and the returns, so the smaller the forecasts the smaller the *RSS*.

```
> # Define the response and predictor matrices
> respv <- retp
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> # Calculate the AR coefficients and the in-sample forecasts
> predinv <- MASS::ginv(predm)
> coeff <- drop(predinv %*% respv)
> names(coeff) <- colnames(predm)
> fcasts <- predm %*% coeff
```




```
> # Calculate the correlation between forecasts and returns
> cor(fcasts, respv)
> # Calculate the volatilities of the returns and forecasts
> sd(respv); sd(fcasts)
> # Calculate the mean squared error of the forecasts
> mean((fcasts - respv)^2)
> # Plot the forecasts
> datav <- cbind(respv, fcasts)["2020-01/2020-06"]
> colnames(datav) <- c("returns", "forecasts")
> dygraphs::dygraph(datav, main="VTI Returns And Forecasts") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

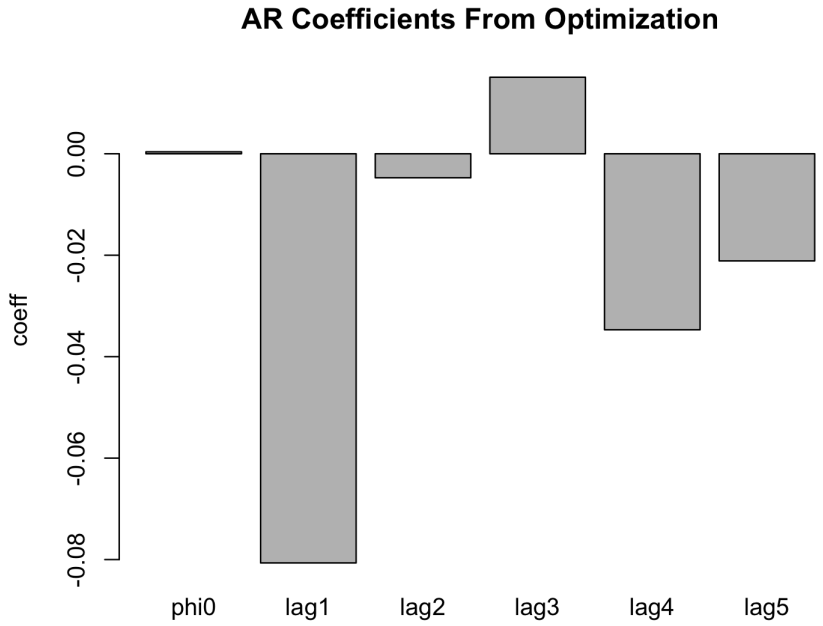
Optimization of the Autoregressive Coefficients

The *AR* coefficients can also be calculated by minimizing the *MSE* of the in-sample forecasts.

The objective function is equal to the *MSE*:

$$\text{ObjFunc} = \sum_{i=1}^{n} (r_t - f_t)^2$$

```
> # Objective function for the in-sample AR coefficients
> objfun <- function(coeff) {
+   fcasts <- predm %*% coeff
+   sum((respv - fcasts)^2)
+ }  # end objfun
> # Perform optimization using the quasi-Newton method
> optiml <- optim(par=numeric(orderp+1),
+           fn=objfun,
+           method="L-BFGS-B",
+           upper=rep(10, orderp+1),
+           lower=rep(-10, orderp+1))
```




```
> # Extract the AR coefficients
> coeffopt <- optiml$par
> names(coeffopt) <- colnames(predm)
> all.equal(coeffopt, coeff)
> barplot(coeffopt, xlab="", ylab="coeff", col="grey",
+   main="AR Coefficients From Optimization")
```

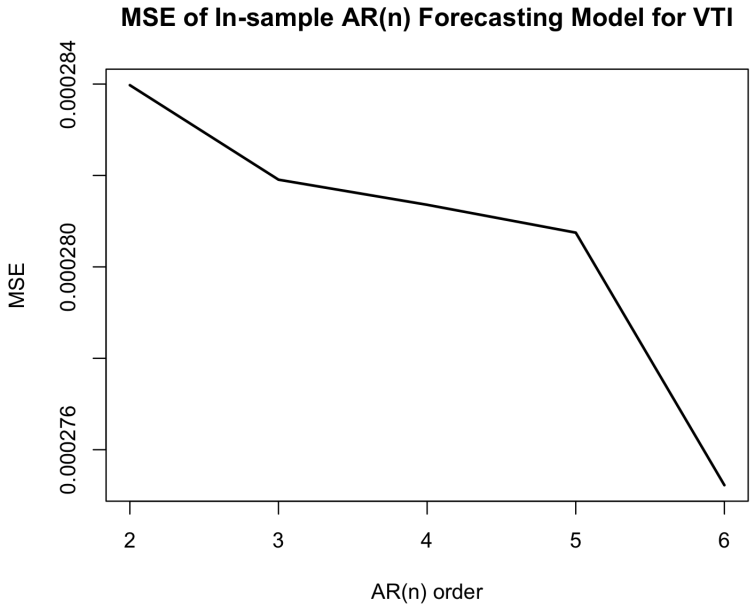
In-sample Order Selection of Autoregressive Forecasting

The mean squared errors (*MSE*) of the *in-sample* forecasts decrease steadily with the increasing order parameter *n* of the *AR(n)* forecasting model.

In-sample forecasting consists of first fitting an *AR(n)* model to the data, and calculating its coefficients.

The *in-sample* forecasts are calculated by multiplying the predictor matrix by the fitted *AR* coefficients.

```
> # Calculate the forecasts as a function of the AR order
> fcasts <- lapply(2:NCOL(predm), function(ordern) {
+   # Calculate the fitted AR coefficients
+   predinv <- MASS::ginv(predm[, 1:ordern])
+   coeff <- predinv %*% respv
+   coeff <- coeff/sqrt(sum(coeff^2))
+   # Calculate the in-sample forecasts of VTI
+   drop(predm[, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcasts) <- paste0("n=", 2:NCOL(predm))
```




```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((respv - x)^2), cor=cor(respv, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- names(fcasts)
> # Plot forecasting MSE
> plot(x=2:NCOL(predm), y=mse[, 1],
+   xlab="AR(n) order", ylab="MSE", type="l", lwd=2,
+   main="MSE of In-sample AR(n) Forecasting Model for VTI")
```

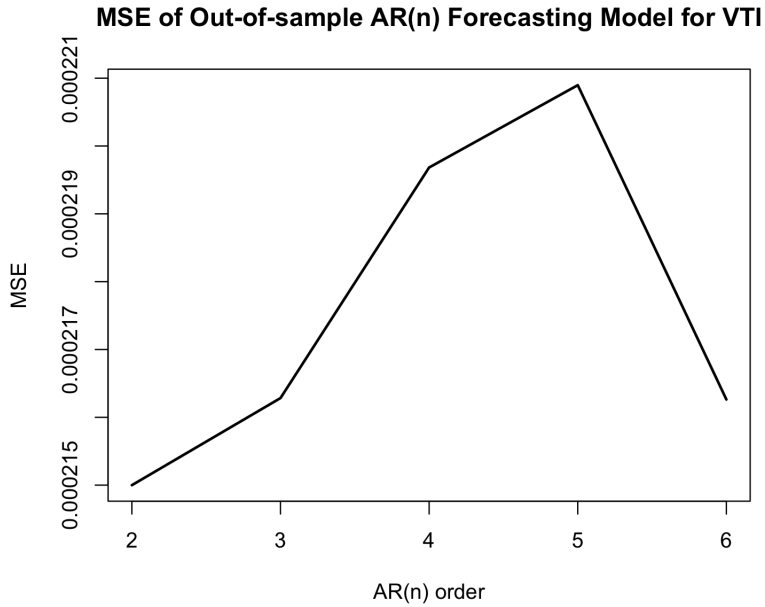
Out-of-Sample Forecasting Using Autoregressive Models

The mean squared errors (*MSE*) of the *out-of-sample* forecasts increase with the increasing order parameter *n* of the *AR(n)* model.

The reason for the increasing out-of-sample MSE is the *overfitting* of the coefficients to the training data for larger order parameters.

Out-of-sample forecasting consists of first fitting an *AR(n)* model to the training data, and calculating its coefficients.

The *out-of-sample* forecasts are calculated by multiplying the *out-of-sample* predictor matrix by the fitted *AR* coefficients.




```
> # Define in-sample and out-of-sample intervals
> nrows <- NROW(retp)
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> # Calculate the forecasts as a function of the AR order
> fcastl <- lapply(2:NCOL(predm), function(ordern) {
+   # Calculate the in-sample AR coefficients
+   predinv <- MASS::ginv(predm[insample, 1:ordern])
+   coeff <- predinv %*% respv[insample]
+   coeff <- coeff/sqrt(sum(coeff^2))
+   # Calculate the out-of-sample forecasts of VTI
+   predm[outsample, 1:ordern] %*% coeff
+ })  # end lapply
> names(fcastl) <- paste0("n=", 2:NCOL(predm))
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcastl, function(x) {
+   c(mse=mean((respv[outsample] - x)^2), cor=cor(respv[outsample],
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- names(fcastl)
> # Plot forecasting MSE
> plot(x=2:NCOL(predm), y=mse[, 1],
+   xlab="AR(n) order", ylab="MSE", type="l", lwd=2,
+   main="MSE of Out-of-sample AR(n) Forecasting Model for VTI")
```

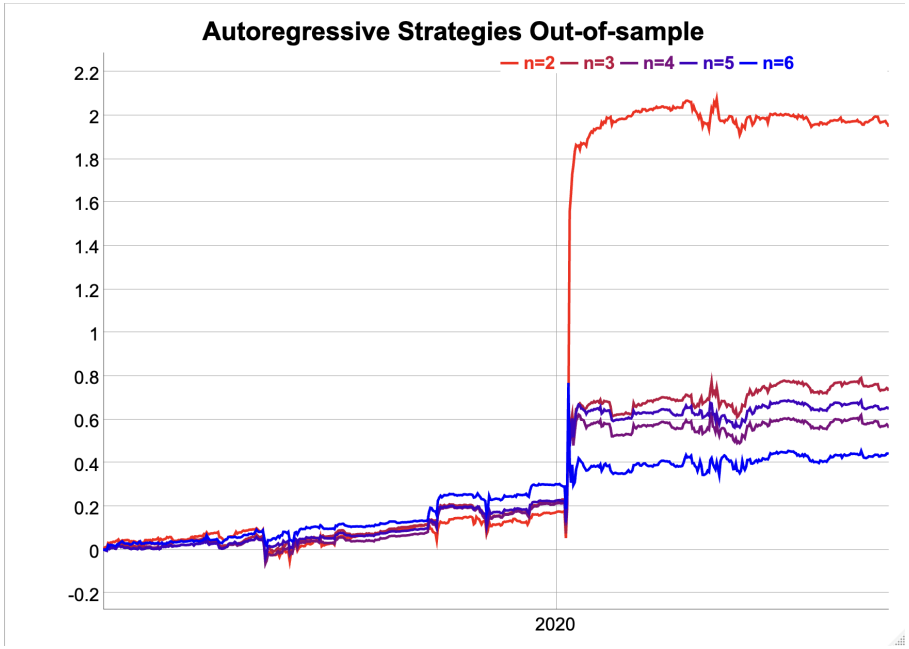
Out-of-Sample Autoregressive Strategy

The autoregressive strategy invests dollar amounts of *VTI* proportional to the AR forecasts.

The out-of-sample, risk-adjusted performance of the autoregressive strategy is better for a smaller order parameter n of the *AR(n)* model.

The optimal order parameter is `n = 2`, with a positive intercept coefficient φ_0 (since the average *VTI* returns were positive), and a negative coefficient φ_1 (because of strong negative autocorrelations in periods of high volatility).

Decreasing the order parameter of the autoregressive model is a form of *shrinkage* because it reduces the number of predictive variables.




```
> # Calculate the out-of-sample PnLs
> pnls <- lapply(fcastl, function(fcasts) {
+   pnls <- fcasts*retp[outsample]
+   pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
+ })  # end lapply
> pnls <- rutils::do_call(cbind, pnls)
> colnames(pnls) <- names(fcastl)
```

```
> # Plot dygraph of out-of-sample PnLs
> colorv <- colorRampPalette(c("red", "blue"))(NCOL(pnls))
> colv <- colnames(pnls)
> sqrt(252)*sapply(pnls, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> endw <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endw],
+   main="Autoregressive Strategies Out-of-sample") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(width=300)
```

Out-of-Sample Autoregressive Forecasts

The autoregressive coefficients φ are equal to the in-sample *response* $\mathbf{r}$ times the inverse of the in-sample *predictor matrix* $\mathbb{P}$:

$$\varphi = \mathbb{P}^{-1} \mathbf{r}$$

The variance of the autoregressive coefficients σ_{φ}^2 is equal to the variance of the in-sample forecast residuals σ_{ε}^2 divided by the squared *predictor matrix* $\mathbb{P}$:

$$\sigma_{\varphi}^2 = \sigma_{\varepsilon}^2 (\mathbb{P}^T \mathbb{P})^{-1}$$

The t-values of the autoregressive coefficients are equal to the coefficient values divided by their volatilities:

$$\varphi_{tval} = \frac{\varphi}{\sigma_{\varphi}}$$

The *out-of-sample* autoregressive forecast f_t is equal to the single row of the predictor $\mathbb{P}_t$ times the fitted *AR* coefficients φ:

$$f_t = \varphi \mathbb{P}_t$$

The variance σ_f^2 of the *forecast value* is equal to the inner product of the predictor $\mathbb{P}_t$ times the coefficient covariance matrix σ_{φ}^2:

$$\sigma_f^2 = \mathbb{P}_t \, \sigma_{\varphi}^2 \, \mathbb{P}_t^T$$

```
> # Define the look-back range
> lookb <- 100
> tday <- nrows
> startp <- max(1, tday-lookb)
> rangev <- startp:(tday-1)
> # Subset the response and predictors
> resps <- respv[rangev]
> preds <- predm[rangev]
> # Invert the predictor matrix
> predinv <- MASS::ginv(preds)
> # Calculate the fitted AR coefficients
> coeff <- predinv %*% resps
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- preds %*% coeff
> # Calculate the residuals (forecast errors)
> resids <- (fcasts - resps)
> # Calculate the variance of the residuals
> varv <- sum(resids^2)/(NROW(preds)-NROW(coeff))
> # Calculate the predictor matrix squared
> pred2 <- crossprod(preds)
> # Calculate the covariance matrix of the AR coefficients
> covmat <- varv*MASS::ginv(pred2)
> coefsd <- sqrt(diag(covmat))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coefsd)
> # Calculate the out-of-sample forecast
> predn <- predm[tday, ]
> fcast <- drop(predn %*% coeff)
> # Calculate the variance of the forecast
> varf <- drop(predn %*% covmat %*% t(predn))
> # Calculate the t-value of the out-of-sample forecast
> fcast/sqrt(varf)
```

Rolling Autoregressive Forecasting Model

The autoregressive coefficients can be calibrated dynamically over a *rolling* look-back interval, and applied to calculating the *out-of-sample* forecasts.

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

The *backtesting* procedure allows determining the optimal *meta-parameters* of the forecasting model: the order *n* of the *AR(n)* model and the length of look-back interval (`lookb`).

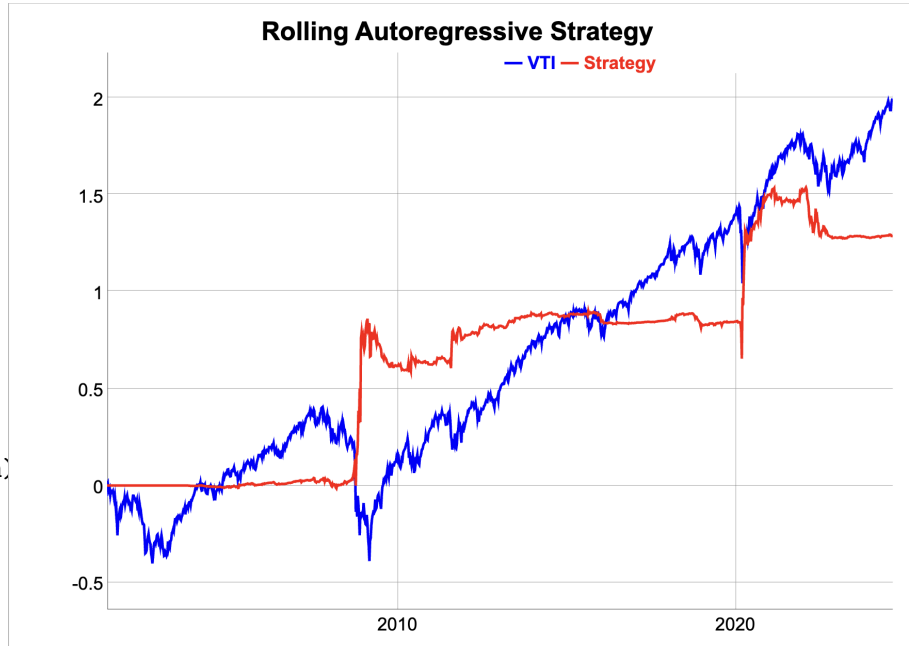
```
> # Perform rolling forecasting
> lookb <- 500
> fcasts <- parallel::mclapply(1:nrows, function(tday) {
+   if (tday > lookb) {
+     # Define the rolling look-back range
+     startp <- max(1, tday-lookb)
+     # startp <- 1 # Expanding look-back range
+     rangev <- startp:(tday-1) # In-sample range
+     # Subset the response and predictors
+     resps <- respv[rangev]
+     preds <- predm[rangev]
+     # Calculate the fitted AR coefficients
+     predinv <- MASS::ginv(preds)
+     coeff <- predinv %*% resps
+     # Calculate the in-sample forecasts of VTI (fitted values)
+     fcasts <- preds %*% coeff
+     # Calculate the residuals (forecast errors)
+     resids <- (fcasts - resps)
+     # Calculate the variance of the residuals
+     varv <- sum(resids^2)/(NROW(preds)-NROW(coeff))
+     # Calculate the covariance matrix of the AR coefficients
+     pred2 <- crossprod(preds)
+     covmat <- varv*MASS::ginv(pred2)
+     coefsd <- sqrt(diag(covmat))
+     coefft <- drop(coeff/coefsd) # t-values of the AR coefficients
+     # Calculate the out-of-sample forecast
+     predn <- predm[tday, ]
+     fcast <- drop(predn %*% coeff)
+     # Calculate the variance of the forecast
+     varf <- drop(predn %*% covmat %*% t(predn))
+     return(c(sd(resps), fcast=fcast, fstderr=sqrt(varf), coefft=co
+   } else {
+     return(c(volv=0, fcast=0, fstderr=0, coefft=rep(0, NCOL(predm)
+   } # end if
+ })  # end sapply
```

Rolling Autoregressive Strategy Performance

In the rolling autoregressive strategy, the autoregressive coefficients are calibrated on past data from a *rolling* look-back interval, and applied to calculating the *out-of-sample* forecasts.

The rolling autoregressive strategy performance depends on the length of the look-back interval.

```
> # Coerce fcasts to a time series
> fcasts <- do.call(rbind, fcasts)
> ncols <- NCOL(fcasts)
> colnames(fcasts) <- c("volv", "fcasts", "fstderr", colnames(predm)
> fcasts <- xts::xts(fcasts[, "fcasts"], zoo::index(retp))
> # Calculate the strategy PnLs
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot dygraph of the autoregressive strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Rolling Autoregressive Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Backtesting the Autoregressive Model

The *meta-parameters* of the *backtesting* function are the order *n* of the *AR(n)* model and the length of the look-back interval (`lookb`).

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

The *backtesting* procedure allows determining the optimal *meta-parameters* of the forecasting model: the order *n* of the *AR(n)* model and the length of look-back interval (`lookb`).

```
> # Define backtesting function
> sim_fcasts <- function(lookb=100, ordern=5, fixedlb=TRUE) {
+   # Perform rolling forecasting
+   fcasts <- sapply((lookb+1):nrows, function(tday) {
+     # Rolling look-back range
+     startp <- max(1, tday-lookb)
+     # Expanding look-back range
+     if (!fixedlb) {startp <- 1}
+     startp <- max(1, tday-lookb)
+     rangev <- startp:(tday-1) # In-sample range
+     # Subset the response and predictors
+     resps <- respv[rangev]
+     preds <- predm[rangev, 1:ordern]
+     # Invert the predictor matrix
+     predinv <- MASS::ginv(preds)
+     # Calculate the fitted AR coefficients
+     coeff <- predinv %*% resps
+     # Calculate the out-of-sample forecast
+     drop(predm[tday, 1:ordern] %*% coeff)
+   })  # end sapply
+   # Add warmup period
+   fcasts <- c(rep(0, lookb), fcasts)
+ }  # end sim_fcasts
> # Simulate the rolling autoregressive forecasts
> fcasts <- sim_fcasts(lookb=100, ordern=5)
> c(mse=mean((fcasts - retp)^2), cor=cor(retp, fcasts))
```

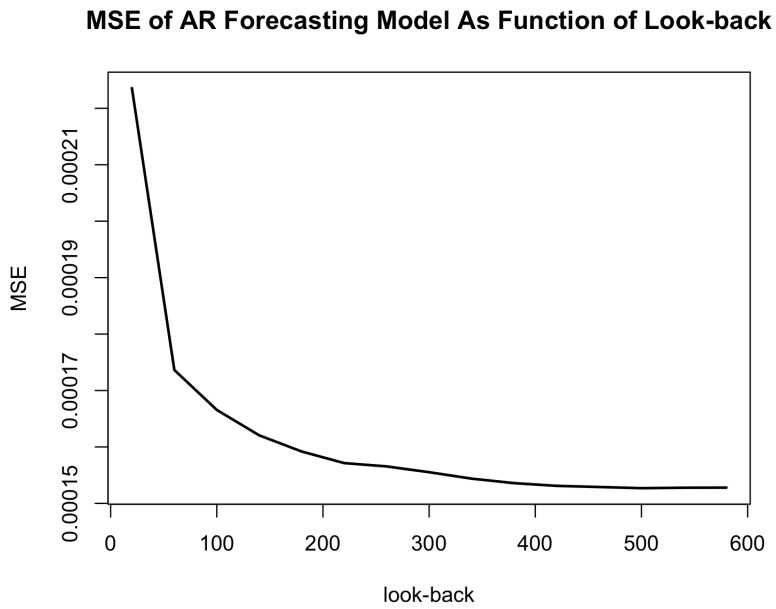
Forecasting Dependence On the Look-back Interval

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`lookb`).

The accuracy of the forecasting model increases with longer look-back intervals (`lookb`), because more data improves the estimates of the autoregressive coefficients.




```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> compclust <- makeCluster(ncores)
> # Perform parallel loop under Windows
> lookbv <- seq(20, 600, 40)
> fcasts <- parLapply(compclust, lookbv, sim_fcasts, ordern=6)
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(lookbv, sim_fcasts, ordern=6, mc.cores=ncores)
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- lookbv
> # Select optimal lookb interval
> lookb <- lookbv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=lookbv, y=mse[, 1],
+   xlab="look-back", ylab="MSE", type="l", lwd=2,
+   main="MSE of AR Forecasting Model As Function of Look-back")
```

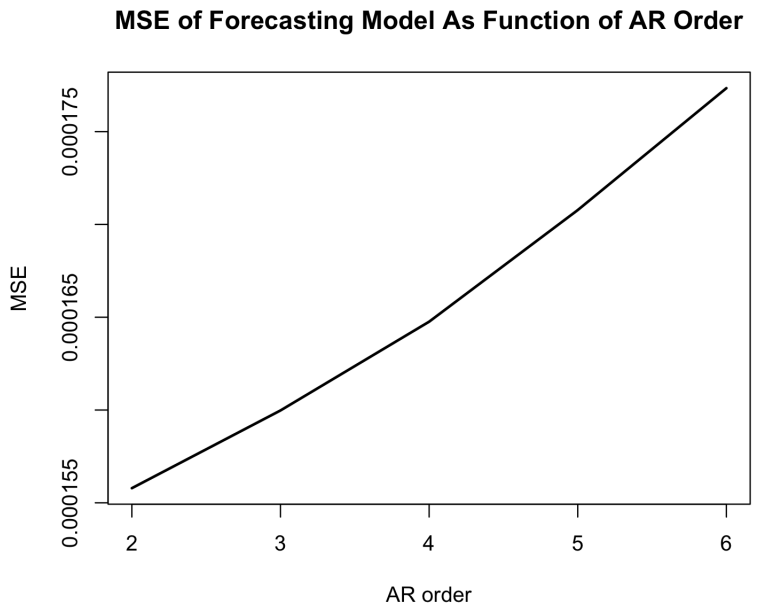
Order Dependence With Fixed Look-back

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`lookb`).

The accuracy of the forecasting model decreases for larger AR order parameters, because of overfitting in-sample.




```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> compclust <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(compclust, orderv, sim_fcasts, lookb=lookb)
> stopCluster(compclust)  # Stop R processes over cluster under Wi
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   lookb=lookb, mc.cores=ncores)
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, y=mse[, 1],
+   xlab="AR order", ylab="MSE", type="l", lwd=2,
+   main="MSE of Forecasting Model As Function of AR Order")
```

Autoregressive Strategy With Fixed Look-back

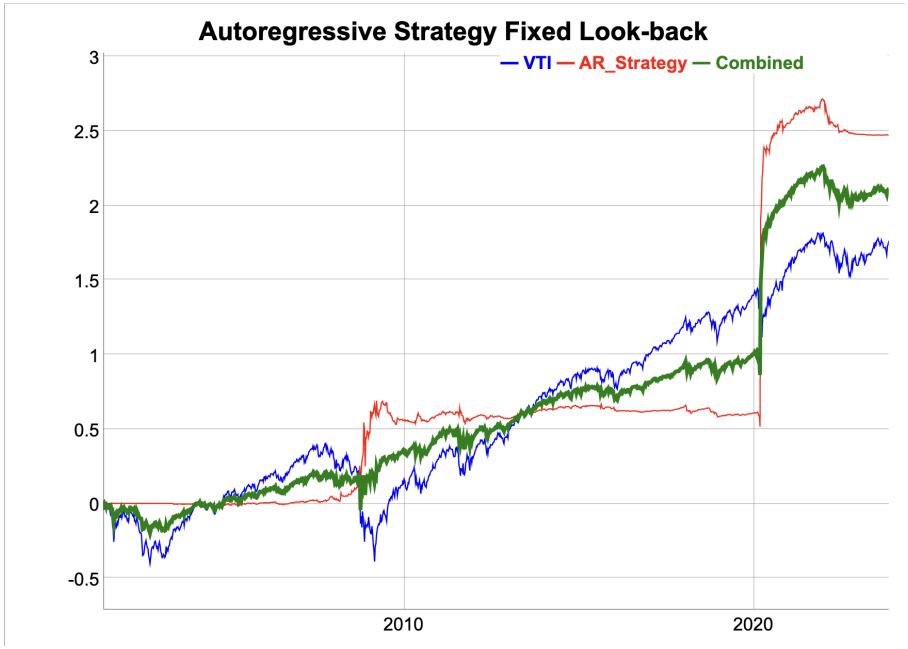
The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

Using the return forecasts as portfolio weights produces very large weights in periods of high volatility, and creates excessive risk.

To reduce excessive risk, a binary strategy can be used, with portfolio weights equal to the sign of the forecasts.

```
> # Simulate the rolling autoregressive forecasts
> fcasts <- sim_fcasts(lookb=lookb, ordern=ordern)
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```




```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy Fixed Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Order Dependence With Expanding Look-back

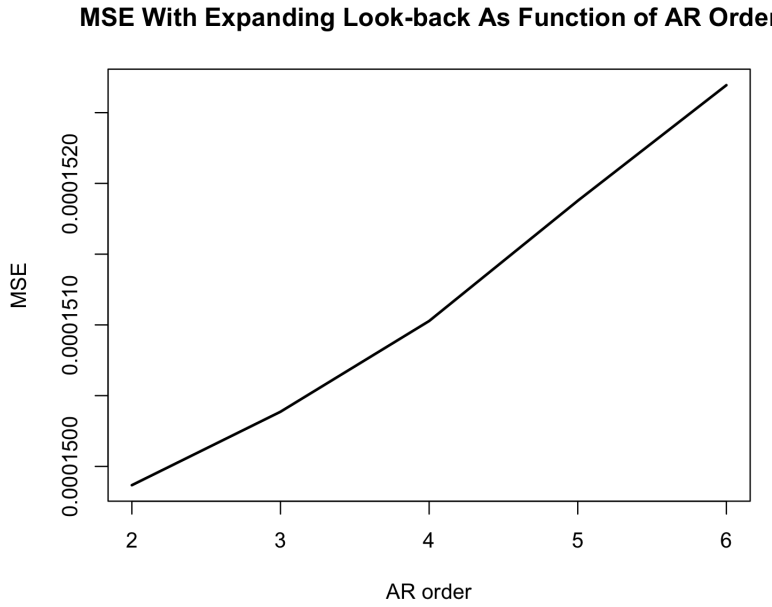
The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model.

Longer look-back intervals (`lookb`) are usually better for the autoregressive forecasting model.

The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> compclust <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(compclust, orderv, sim_fcasts,
+   lookb=lookb, fixedlb=FALSE)
> stopCluster(compclust)  # Stop R processes over cluster under Wi
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   lookb=lookb, fixedlb=FALSE, mc.cores=ncores)
```



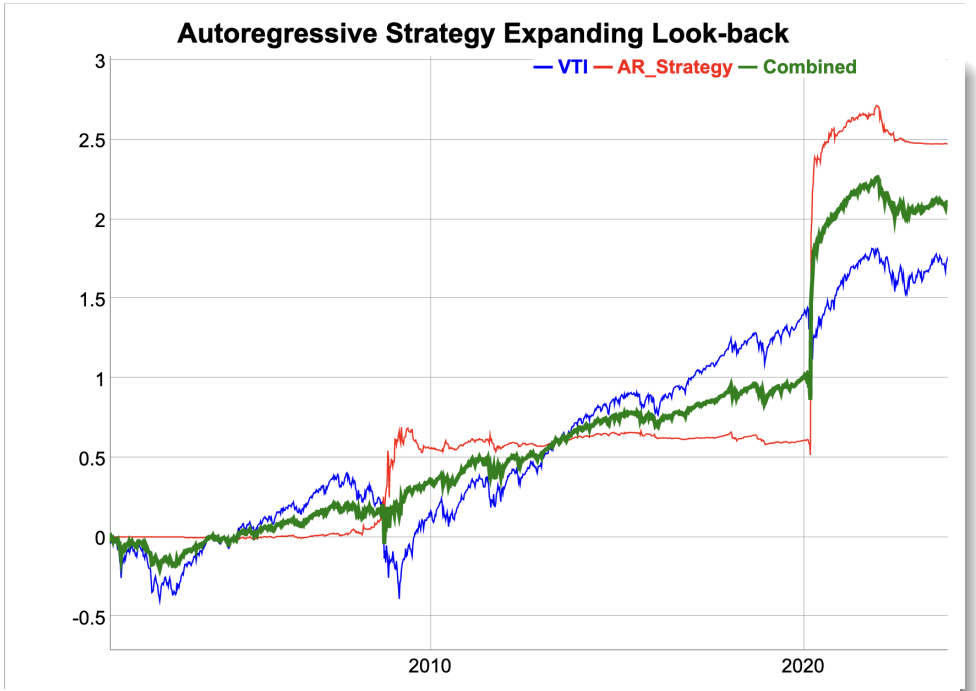

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, y=mse[, 1],
+   xlab="AR order", ylab="MSE", type="l", lwd=2,
+   main="MSE With Expanding Look-back As Function of AR Order")
```

Autoregressive Strategy With Expanding Look-back

The model with an *expanding* look-back interval has better performance compared to the *fixed* look-back interval.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

```
> # Simulate the autoregressive forecasts with expanding look-back
> fcasts <- sim_fcasts(lookb=lookb, ordern=ordern, fixedlb=FALSE)
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```



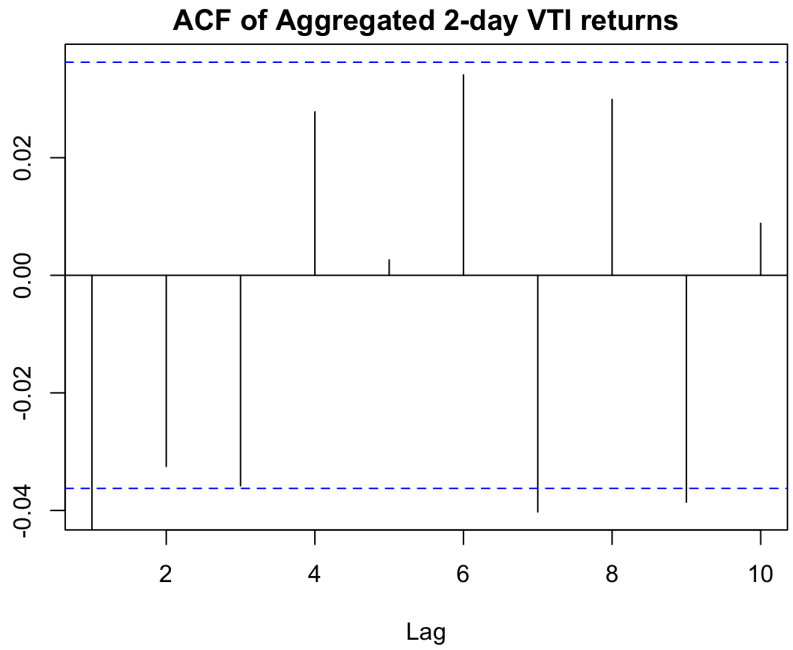

```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy Expanding Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Autocorrelations of Aggregated Stock Returns

Stock returns aggregated over several days have much smaller autocorrelations.

But aggregating stock returns can reduce their noise and improve the performance of autoregressive strategies.

```
> # Calculate VTI returns over non-overlapping 2-day intervals
> pricev <- na.omit(rutils::etfenv$prices$VTI)
> reta <- rutils::diffit(log(pricev), lag=2)
> reta <- reta[2*(1:(NROW(pricev) %/% 2))]
> # Calculate the autocorrelations of daily VTI returns
> rutils::plot_acf(reta, lag=10, main="ACF of Aggregated 2-day VTI r
```



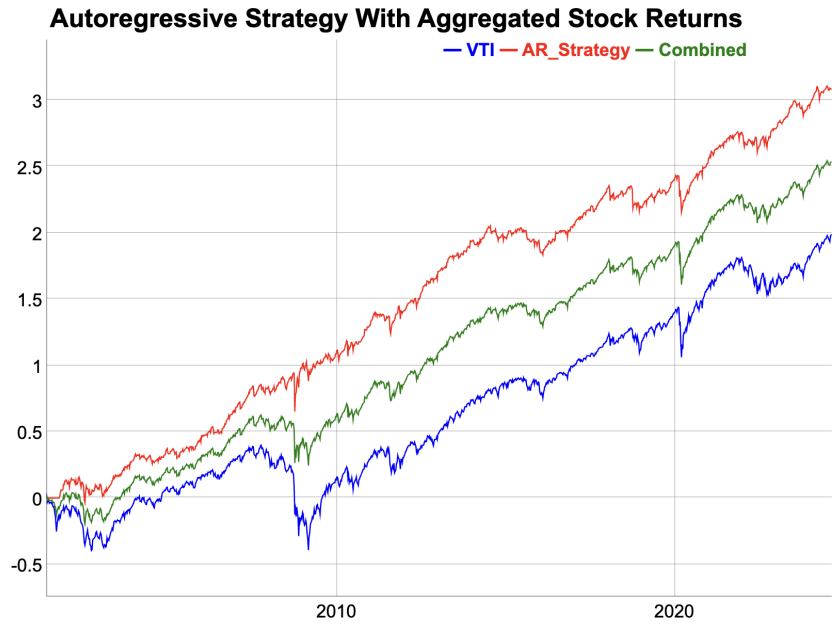
Autoregressive Strategy With Aggregated Stock Returns

The autoregressive strategy with aggregated stock returns calculates the *AR* coefficients and the forecasts using stock returns aggregated over several days.

The strategy holds the stock position for several days, and rebalances the position at the end of the aggregation interval.

In this example, the returns are calculated over 2-day intervals, and the stock is held for 2 days. Effectively, there are two strategies, rebalancing every other day.

The aggregated autoregressive strategy has good performance both in periods of high volatility and in low volatility.




```
> # Define the response and predictor matrices
> reta <- rutils::diffit(log(pricev), lag=2)/2
> orderp <- 5
> predm <- lapply(2*(1:orderp), rutils::lagit, input=reta)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(reta)), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> # Calculate the AR coefficients
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% reta
> coeffn <- paste0("phi", 0:(NROW(coeff)-1))
> barplot(coeff ~ coeffn, xlab="", ylab="t-value", col="grey",
+   main="Coefficients of AR Forecasting Model")
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
```

```
> # Calculate the autoregressive strategy PnLs
> pnls <- reta*fcasts
> pnls <- pnls*sd(reta[reta<0])/sd(pnls[pnls<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(reta, pnls, (reta+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv["2010/"], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy With Aggregated Stock Returns") %>
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dyLegend(show="always", width=300)
```

Daytime and Overnight Stock Strategies

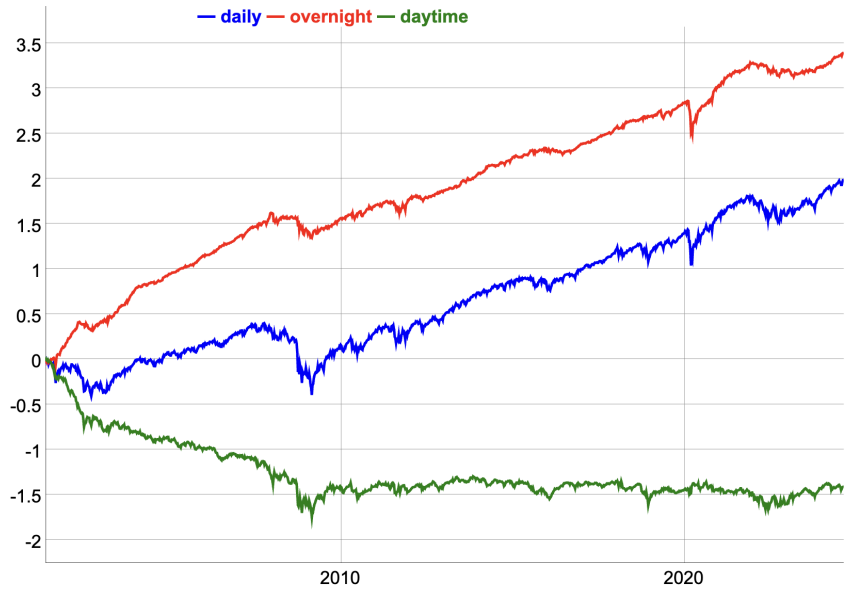
The overnight stock strategy consists of holding a long position only overnight (buying at the market close and selling at the open the next day).

The daytime stock strategy consists of holding a long position only during the daytime (buying at the market open and selling at the close the same day).

The *Overnight Market Anomaly* is the consistent outperformance of overnight returns relative to the daytime returns.

The *Overnight Market Anomaly* has been observed for many decades for most stock market indices, but not always for all stock sectors.

The *Overnight Market Anomaly* is not as pronounced after the 2008-2009 financial crisis.




```
> # Calculate the log of OHLC VTI prices
> ohlc <- log(rutils::etfenv$VTI)
> nrows <- NROW(ohlc)
> openp <- quantmod::Op(ohlc)
> highp <- quantmod::Hi(ohlc)
> lowp <- quantmod::Lo(ohlc)
> closep <- quantmod::Cl(ohlc)
> # Calculate the close-to-close log returns,
> # the daytime open-to-close returns
> # and the overnight close-to-open returns.
> retp <- rutils::diffit(closep)
> colnames(retp) <- "daily"
> retd <- (closep - openp)
> colnames(retd) <- "daytime"
> reton <- (openp - rutils::lagit(closep, lagg=1, pad_zeros=FALSE))
> colnames(reton) <- "overnight"
```

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, reton, retd)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the Daytime and Overnight strategies
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Wealth of Close-to-Close, Overnight, and Daytime Strategie
+   dySeries(name="daily", strokeWidth=2, col="blue") %>%
+   dySeries(name="overnight", strokeWidth=2, col="red") %>%
+   dySeries(name="daytime", strokeWidth=2, col="green") %>%
+   dyLegend(width=600)
```

EMA Mean-Reversion Strategy For Daytime Returns

After the 2008–2009 financial crisis, the cumulative daytime stock index returns have been range-bound. So the daytime returns have more significant negative autocorrelations than overnight returns.

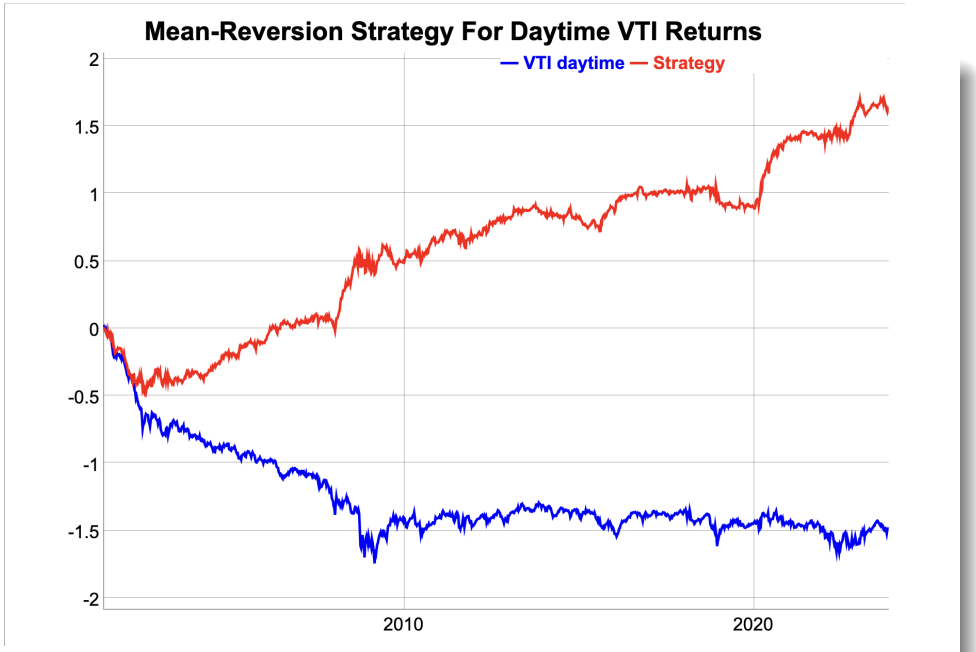
The *EMA* mean-reversion strategy holds stock positions equal to the *sign* of the trailing *EMA* of past returns.

The strategy adjusts its stock position at the end of each day, just before the close of the market.

An alternative strategy holds positions proportional to minus of the sign of the trailing *EMA* of past returns.

The strategy takes very large positions in periods of high volatility, when returns are large and highly anti-correlated.

The limitation is that this strategy makes most of its profits in periods of high volatility, but otherwise it's profits are small.




```
> # Calculate the autocorrelations of daytime and overnight returns
> pacfl <- pacf(retd, lag.max=10, plot=FALSE)
> sum(pacfl$acf)
> pacfl <- pacf(reton, lag.max=10, plot=FALSE)
> sum(pacfl$acf)
> # Calculate the EMA returns recursively using C++ code
> retma <- HighFreq::run_mean(retd, lambda=0.4)
> # Calculate the positions and PnLs
> posv <- -rutils::lagit(sign(retma), lagg=1)
> pnls <- retd*posv
```

```
> # Calculate the pnls and the transaction costs
> bidask <- 0.0001 # The bid-ask spread is equal to 1 basis point
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI daytime", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+ + c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of crossover strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Mean-Reversion Strategy For Daytime VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Bollinger Strategy For Daytime Returns

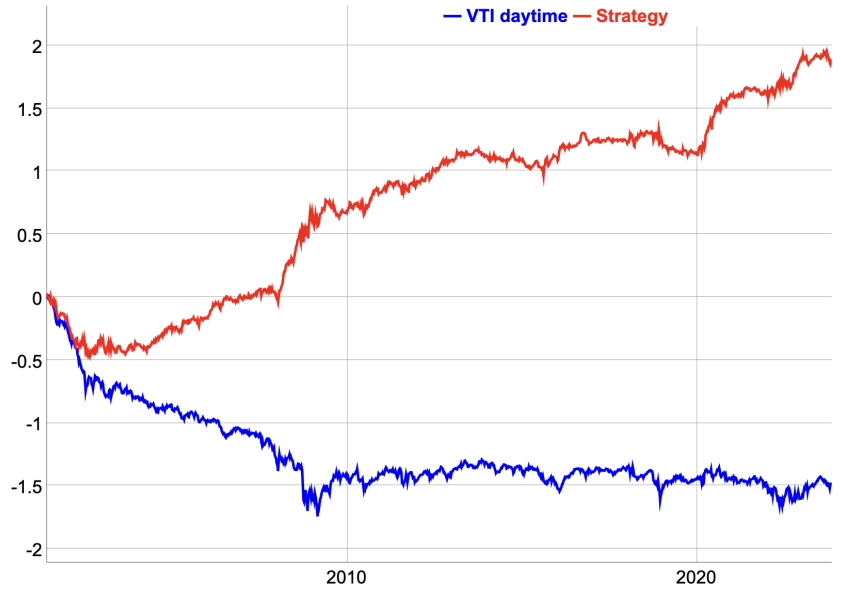
The Bollinger *z-score* for daytime returns z_t, is equal to the difference between the cumulative returns $p_t = \sum r_t$ minus their trailing mean $\bar{p}_t$, divided by their volatility σ_t:

$$z_t = \frac{p_t - \bar{p}_t}{\sigma_t}$$

The Bollinger strategy switches to \$1 dollar long stock if the *z-score* drops below the threshold of -1 (indicating the prices are cheap), and switches to $-\$1$ dollar short if the *z-score* exceeds the threshold of 1 (indicating the prices are rich - expensive).

The Bollinger strategy is a *mean reverting* (contrarian) strategy because it bets on the cumulative returns reverting to their mean value.

The Bollinger strategy has performed well for daytime *VTI* returns because they exhibit significant mean-reversion.




```
> # Calculate the z-scores of the cumulative daytime returns
> retc <- cumsum(retd)
> lambdaf <- 0.24
> retm <- rutils::lagit(HighFreq::run_mean(retc, lambda=lambdaf))
> retv <- sqrt(rutils::lagit(HighFreq::run_var(retc, lambda=lambda
> zscores <- ifelse(retv > 0, (retc - retm)/retv, 0)
> # Calculate the positions from the Bollinger z-scores
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(zscores > 1, -1, posv)
> posv <- ifelse(zscores < -1, 1, posv)
> posv <- zoo::na.locf(posv)
```

```
> # Calculate the pnls and the transaction costs
> pnls <- retd*posv
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
> # Calculate the Sharpe ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI daytime", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of daytime Bollinger strategy
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Bollinger strategy For Daytime VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

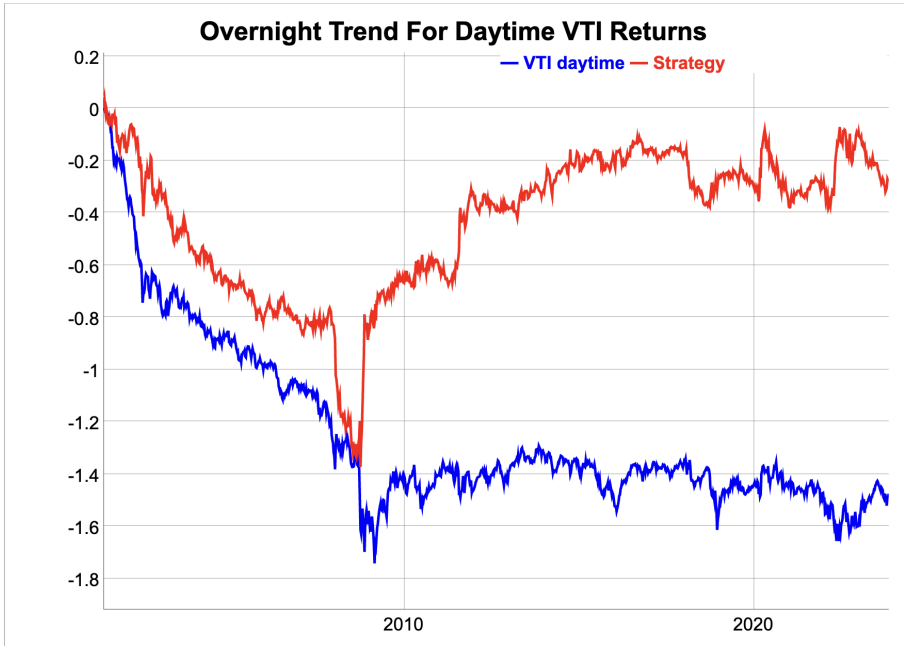
Overnight Trend Strategy

Analysts at *JPMorgan* and at the *Federal Reserve* have observed that there is a trend in the overnight returns.

Positive overnight returns are often followed by positive daytime returns, and vice versa.

If the overnight returns were positive, then the strategy buys \$1 dollar of stock at the market open and sells it at the market close, or if the overnight returns were negative then it shorts -\$1 dollar of stock.

The strategy has performed well immediately after the 2008-2009 financial crisis, but it has waned in recent years.




```
> # Calculate the pnls and the transaction costs
> posv <- sign(reton)
> pnls <- posv*retd
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
```

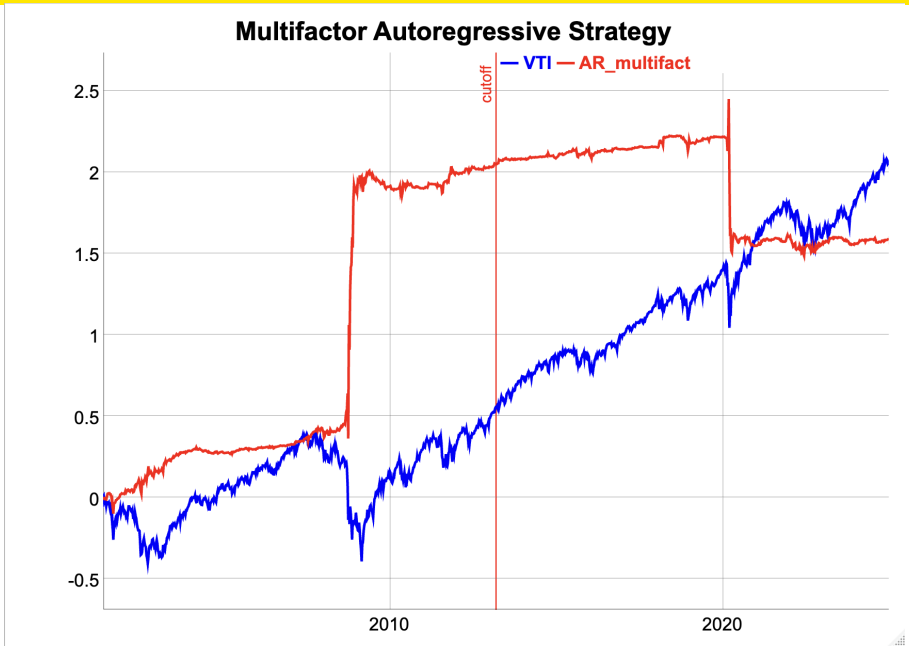
```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI daytime", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of crossover strategy
> dygraphs::dygraph(cumsum(wealthv)[endw],
+ main="Overnight Trend For Daytime VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Multifactor Autoregressive Strategy

Multifactor autoregressive strategies can have a large number of predictors, and are often called *kitchen sink* strategies.

Multifactor strategies are overfit to the in-sample data because they have a large number of parameters.

The out-of-sample performance of the multifactor strategy is much worse than its in-sample performance, because the strategy is overfit to the in-sample data.




```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> datev <- index(retp)
> nrows <- NROW(retp)
> # Define in-sample and out-of-sample intervals
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> cutoff <- nrows %/% 2
> # Define the response and predictor matrices
> respv <- retp
> orderp <- 8 # 9 predictors!!!
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> # Calculate the in-sample fitted autoregressive coefficients
> predinv <- MASS::ginv(predm[insample, ])
> coeff <- drop(predinv %*% respv[insample, ])
> names(coeff) <- colnames(predm)
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
```

```
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> # Calculate the in-sample and out-of-sample Sharpe and Sortino rat
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "ARMultiFact")
> sqrt(252)*sapply(wealthv[insample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv[outsample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategies
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(wealthv))
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Multifactor Autoregressive Strategy") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(show="always", width=300)
```

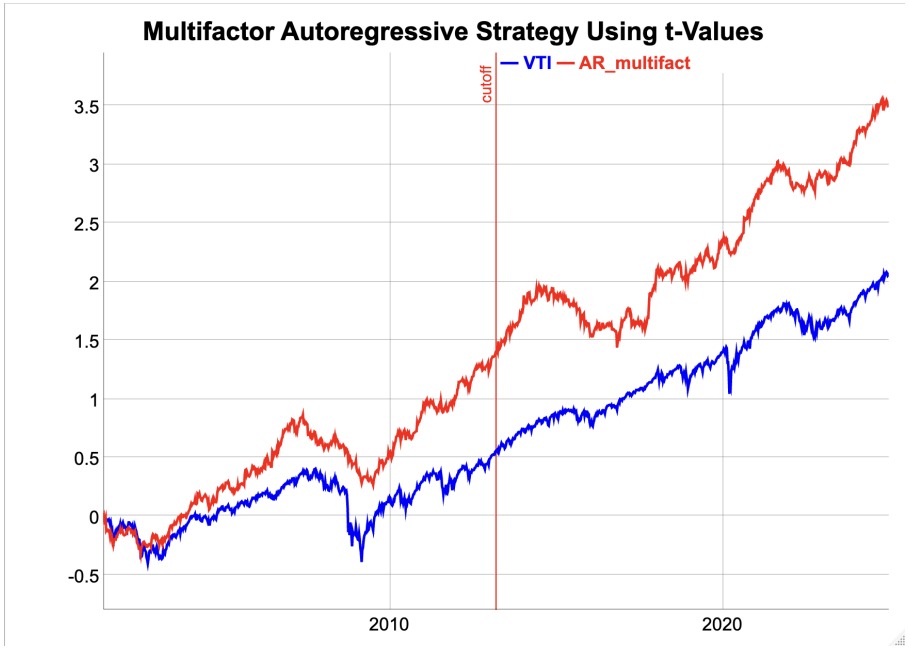
Multifactor Autoregressive Strategy Using t-Values

The out-of-sample performance of the multifactor AR strategy can be improved by using the *t-values* of the *AR* coefficients.

The out-of-sample performance of the multifactor strategy using the *t-values* of the *AR* coefficients is much better than before.

The forecasts calculated from the *t-values* have a positive bias, because the lowest order *t-value* is large.

```
> # Calculate the t-values of the AR coefficients
> resids <- (fcasts[insample, ] - respv[insample, ])
> varv <- sum(resids^2)/(nrows-NROW(coeff))
> pred2 <- crossprod(predm[insample, ])
> covmat <- varv*MASS::ginv(pred2)
> coefsd <- sqrt(diag(covmat))
> coefft <- drop(coeff/coefsd)
> names(coefft) <- colnames(predm)
> # Plot the t-values of the AR coefficients
> barplot(coefft, xlab="", ylab="t-value", col="grey",
+   main="Coefficient t-values of AR Forecasting Model")
> # Calculate the autoregressive strategy PnLs
> fcasts <- predm %*% coefft
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> pnls <- retp*fcasts
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




```
> # Calculate the in-sample and out-of-sample Sharpe and Sortino rat
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "ARMultiFact")
> sqrt(252)*sapply(wealthv[insample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv[outsample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategies
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Multifactor Autoregressive Strategy Using t-Values") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(show="always", width=300)
```

Regularization of the Inverse Predictor Matrix

The *SVD* of a rectangular matrix $\mathbb{A}$ is defined as the factorization:

$$\mathbb{A} = \mathbb{U}\mathbb{\Sigma}\mathbb{V}^T$$

Where $\mathbb{U}$ and $\mathbb{V}$ are the *singular matrices*, and $\mathbb{\Sigma}$ is a diagonal matrix of *singular values*.

The *generalized inverse* matrix $\mathbb{A}^{-1}$ satisfies the inverse equation: $\mathbb{A}\mathbb{A}^{-1}\mathbb{A} = \mathbb{A}$, and it can be expressed as a product of the *SVD* matrices as follows:

$$\mathbb{A}^{-1} = \mathbb{V}\,\mathbb{\Sigma}^{-1}\,\mathbb{U}^T$$

If any of the *singular values* are zero then the *generalized inverse* does not exist.

Regularization is the removal of zero singular values, to make calculating the inverse matrix possible.

The *generalized inverse* is obtained by removing the zero *singular values*:

$$\mathbb{A}^{-1} = \mathbb{V}_n\,\mathbb{\Sigma}_n^{-1}\,\mathbb{U}_n^T$$

Where $\mathbb{U}_n$, $\mathbb{V}_n$ and $\mathbb{\Sigma}_n$ are the *SVD* matrices without the zero *singular values*.

The generalized inverse satisfies the inverse matrix equation: $\mathbb{A}\mathbb{A}^{-1}\mathbb{A} = \mathbb{A}$.

```
> # Calculate singular value decomposition of the predictor matrix
> svdec <- svd(predm)
> barplot(svdec$d, main="Singular Values of Predictor Matrix")
> # Calculate generalized inverse from SVD
> invsvd <- svdec$v %*% (t(svdec$u) / svdec$d)
> # Verify inverse property of the inverse
> all.equal(zoo::coredata(predm), predm %*% invsvd %*% predm)
> # Compare with the generalized inverse using MASS::ginv()
> invreg <- MASS::ginv(predm)
> all.equal(invreg, invsvd)
> # Set tolerance for determining zero singular values
> precv <- sqrt(.Machine$double.eps)
> # Check for zero singular values
> round(svdec$d, 12)
> notzero <- (svdec$d > (precv*svdec$d[1]))
> # Calculate generalized inverse from SVD
> invsvd <- svdec$v[, notzero] %*%
+   (t(svdec$u[, notzero]) / svdec$d[notzero])
> # Verify inverse property of invsvd
> all.equal(zoo::coredata(predm), predm %*% invsvd %*% predm)
> all.equal(invsvd, invreg)
```

Reduced Inverse of the Predictor Matrix

Regularization is the removal of zero singular values, to make calculating the inverse matrix possible.

If the higher order singular values are very small then the inverse matrix will amplify the noise in the response matrix.

Dimension reduction is achieved by the removal of small singular values, to improve the out-of-sample performance of the inverse matrix.

The *reduced inverse* is obtained by removing the very small *singular values*.

$$\mathbb{A}^{-1} = \mathbb{V}_n \, \mathbf{\Sigma}_n^{-1} \, \mathbb{U}_n^T$$

This effectively reduces the number of parameters in the model.

The *reduced inverse* satisfies the inverse equation only approximately (it is *biased*), but it's often used in machine learning because it produces a lower *variance* of the forecasts than the exact inverse.

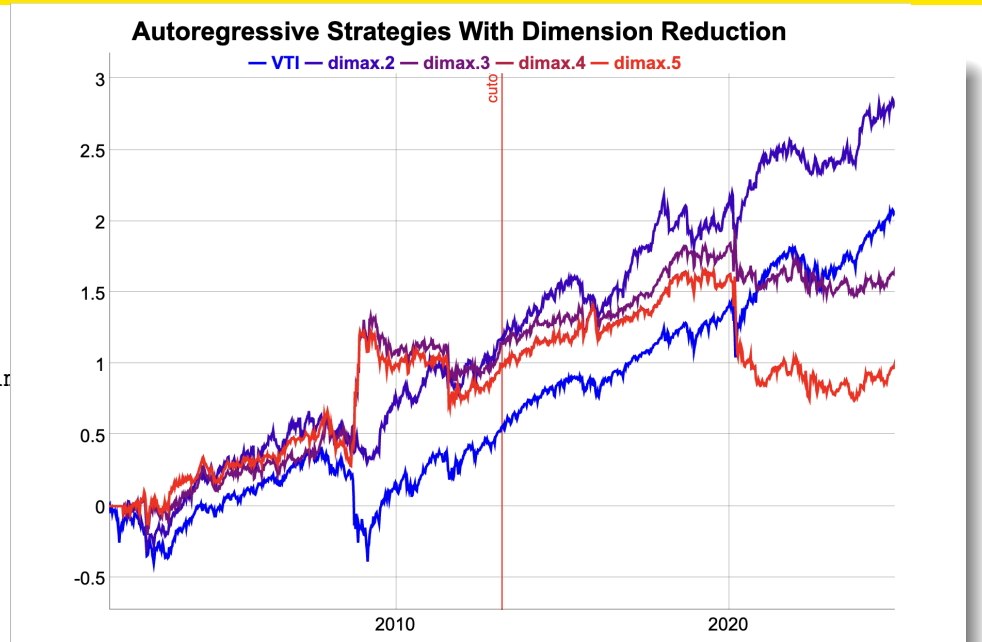
```
> # Calculate reduced inverse from SVD
> dimax <- 3 # Number of dimensions to keep
> invred <- svdec$v[, 1:dimax] %*%
+   (t(svdec$u[, 1:dimax]) / svdec$d[1:dimax])
> # Inverse property fails for invred
> all.equal(zoo::coredata(predm), predm %*% invred %*% predm)
> # Calculate reduced inverse using RcppArmadillo
> invrcpp <- HighFreq::calc_invsvd(predm, dimax=dimax)
> all.equal(invred, invrcpp, check.attributes=FALSE)
```

Autoregressive Strategy With Dimension Reduction

Dimension reduction improves the out-of-sample performance of the multifactor AR strategy.

The best performance is obtained using the smallest order parameter (strongest dimension reduction) *dimax* $= 2$.

```
> # Calculate the in-sample SVD
> svdec <- svd(predm[insample, ])
> # Calculate the in-sample fitted AR coefficients for different dir
> dimv <- 2:5
> # dimv <- c(2, 5, 10, NCOL(predm))
> coeffm <- sapply(dimv, function(dimax) {
+   predinv <- svdec$v[, 1:dimax] %*%
+     (t(svdec$u[, 1:dimax]) / svdec$d[1:dimax])
+   predinv %*% respv[insample]
+ })  # end lapply
> colnames(coeffm) <- paste0("dim=", dimv)
> colorv <- colorRampPalette(c("red", "blue"))(NCOL(coeffm))
> matplot(y=coeffm, type="l", lty="solid", lwd=1, col=colorv,
+   xlab="predictor", ylab="coeff",
+   main="AR Coefficients For Different Dimensions")
> # Calculate the forecasts of VTI
> fcasts <- predm %*% coeffm
> fcasts <- apply(fcasts, 2, function(x) {
+   fcastv <- sqrt(HighFreq::run_var(matrix(x), lambda=0.8)[, 2])
+   fcastv[1:10] <- 1 # Warmup
+   x/fcastv
+ }) # end apply
> # Simulate the autoregressive strategies
> retn <- coredata(retp)
> pnls <- apply(fcasts, 2, function(x) (x*retn))
> pnls <- xts(pnls, datev)
> # Scale the PnL volatility to that of VTI
> pnls <- lapply(pnls, function(x) x/sd(x))
> pnls <- sd(retp)*do.call(cbind, pnls)
```




```
> # Calculate the in-sample and out-of-sample Sharpe and Sortino rat
> wealthv <- cbind(retp, pnls)
> sqrt(252)*sapply(wealthv[insample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv[outsample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategies
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(wealthv))
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategies With Dimension Reduction") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(show="always", width=500)
```

Autoregressive Coefficients With Ridge Shrinkage

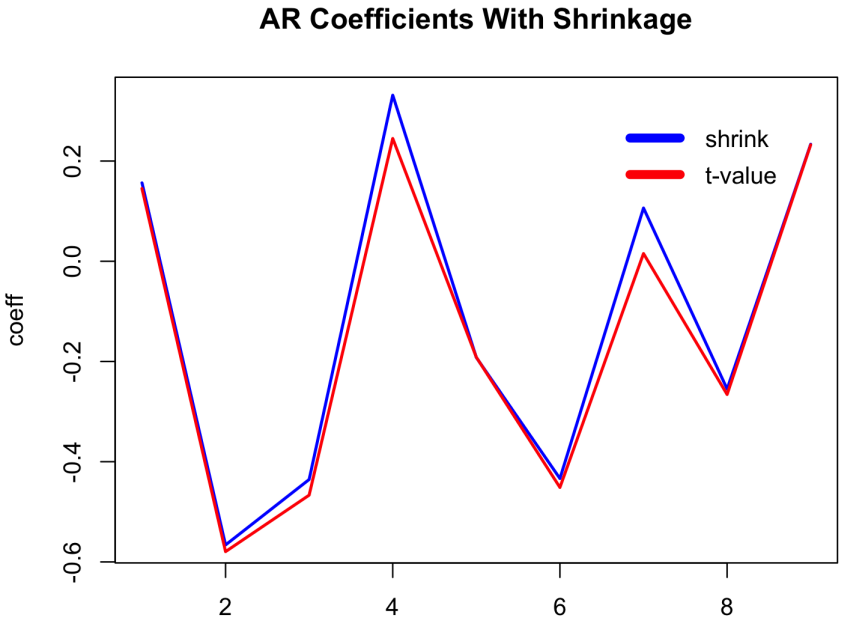
The *AR* coefficients can be found by minimizing the *MSE* of the in-sample forecasts.

The objective function is the sum of the *MSE* plus a *ridge penalty* term proportional to the square of the *AR* coefficients:

$$\text{ObjFunc} = \sum_{i=1}^{n}(r_t - f_t)^2 + \lambda \sum_{i=1}^{k} \varphi_i^2$$

For larger values of the *shrinkage factor* λ, the *AR* coefficients are shrunk closer to zero, and they become proportional to the *t-values* of the coefficients.

The shrinkage and dimension reduction both shrink the coefficients closer to zero.




```
> # Objective function for the in-sample AR coefficients
> objfun <- function(coeff, lambdaf) {
+   fcasts <- predm[insample, ] %*% coeff
+   sum((respv[insample, ] - fcasts)^2) + lambdaf*sum(coeff^2)
+ }  # end objfun
> # Perform optimization using the quasi-Newton method
> optiml <- optim(par=numeric(orderp+1),
+           fn=objfun, lambdaf=50.0,
+           method="L-BFGS-B",
+           upper=rep(10, orderp+1),
+           lower=rep(-10, orderp+1))
> # Extract the shrinkage coefficients
> coeff <- optiml$par
> names(coeff) <- colnames(predm)
```

```
> # Plot the AR coefficients
> barplot(coeff, xlab="", ylab="coeff", col="grey",
+   main="AR Coefficients With Shrinkage")
> # Plot the shrinkage coefficients and the t-values of the coeffic
> plot(coeff/sqrt(sum(coeff^2)), xlab="", ylab="coeff",
+      col="blue", lwd=2, t="l",
+      main="AR Coefficients With Shrinkage")
> lines(coefft/sqrt(sum(coefft^2)), xlab="", ylab="coeff",
+      col="red", lwd=2)
> legend("topright", c("shrink", "t-value"),
+  inset=0.05, col=c("blue", "red"), lwd=6, bty="n")
```

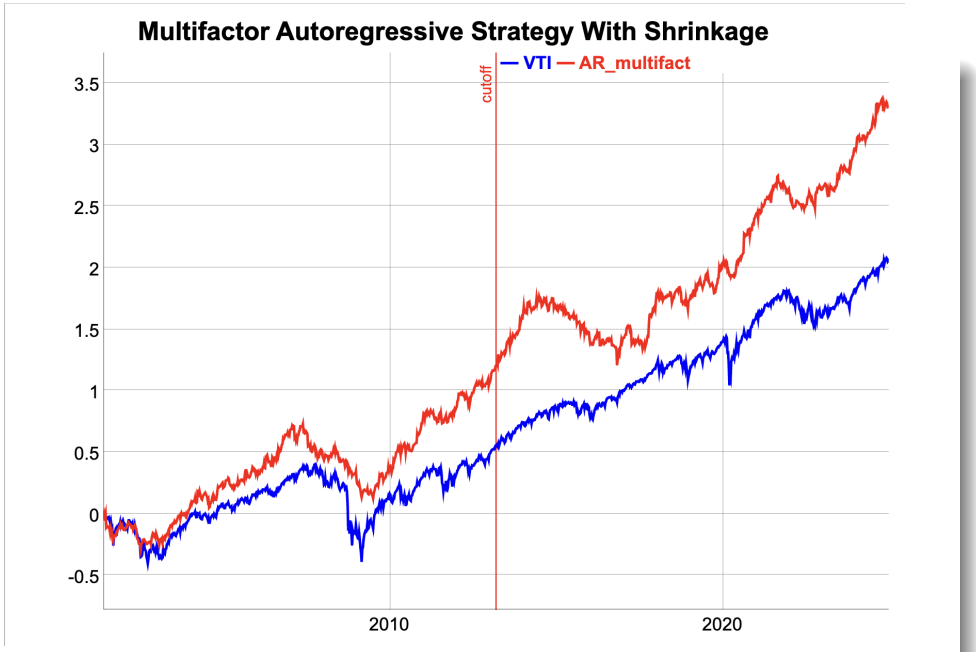
Multifactor AR Strategy With Ridge Shrinkage

The out-of-sample performance of the multifactor autoregressive strategy can be improved by applying *shrinkage* to the *AR* coefficients.

The value of the *shrinkage factor* λ can be chosen to maximize the out-of-sample performance.

Scaling the *AR* forecasts can improve the performance even more.

```
> # Calculate the forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




```
> # Calculate the in-sample and out-of-sample Sharpe and Sortino rat
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "ARMultiFact")
> sqrt(252)*sapply(wealthv[insample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv[outsample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategies
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(wealthv))
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Multifactor Autoregressive Strategy With Shrinkage") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(show="always", width=300)
```

Kitchen Sink Model of Stock Returns

The "kitchen sink" model uses many possible predictor variables, so the predictor matrix has a very large number of columns.

The predictor matrix includes the lagged and scaled daily returns and the squared returns.

stock returns r_t can be fitted into an *autoregressive* model *AR(n)* with a constant intercept term φ_0:

$$r_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n} + \varepsilon_t$$

The *residuals* ε_t are assumed to be normally distributed, independent, and stationary.

The autoregressive model can be written in matrix form as:

$$\mathbf{r} = \varphi \, \mathbb{P} + \varepsilon$$

Where $\varphi = \{\varphi_0, \varphi_1, \varphi_2, \ldots \varphi_n\}$ is the vector of autoregressive coefficients.

The *autoregressive* model is equivalent to *multivariate* linear regression, with the *response* equal to the returns $\mathbf{r}$, and the columns of the *predictor matrix* $\mathbb{P}$ equal to the lags of the returns.

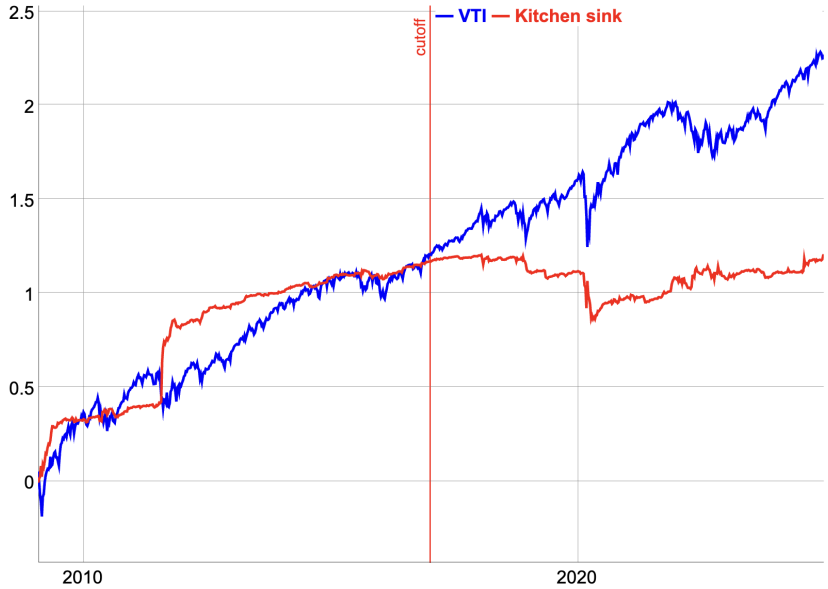
```
> # Calculate the returns of VTI, TLT, and VXX
> retp <- na.omit(rutils::etfenv$returns[, c("VTI", "TLT", "VXX")])
> datev <- zoo::index(retp)
> nrows <- NROW(retp)
> # Define the response and the VTI predictor matrix
> respv <- retp$VTI
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("VTI", 1:orderp))
> # Add the TLT predictor matrix
> predx <- lapply(1:orderp, rutils::lagit, input=retp$TLT)
> predx <- rutils::do_call(cbind, predx)
> colnames(predx) <- paste0("TLT", 1:orderp)
> predm <- cbind(predm, predx)
> # Add the VXX predictor matrix
> predx <- lapply(1:orderp, rutils::lagit, input=retp$VXX)
> predx <- rutils::do_call(cbind, predx)
> colnames(predx) <- paste0("VXX", 1:orderp)
> predm <- cbind(predm, predx)
> # Perform the multivariate linear regression
> regmod <- lm(respv ~ predm - 1)
> summary(regmod)
```

Kitchen Sink Autoregressive Strategy

Multifactor autoregressive strategies can have a large number of predictors, and are often called *kitchen sink* strategies.

Multifactor strategies are overfit to the in-sample data because they have a large number of parameters.

The out-of-sample performance of the multifactor strategy is much worse than its in-sample performance, because the strategy is overfit to the in-sample data.




```
> # Define in-sample and out-of-sample intervals
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> cutoff <- nrows %/% 2
> # Calculate the in-sample fitted autoregressive coefficients
> predinv <- MASS::ginv(predm[insample, ])
> coeff <- drop(predinv %*% respv[insample, ])
> names(coeff) <- colnames(predm)
> barplot(coeff, xlab="", ylab="coeff", col="grey",
+   main="Coefficients of Kitchen Sink Autoregressive Model")
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
> # fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> # fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
```

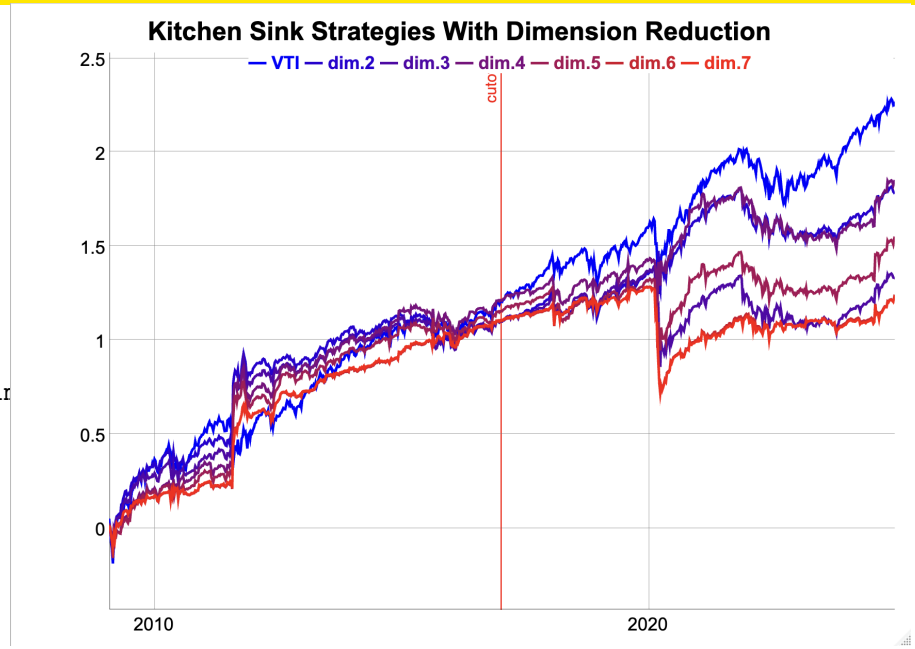
```
> # Calculate the autoregressive strategy PnLs
> pnls <- respv*fcasts
> pnls <- pnls*sd(respv[respv<0])/sd(pnls[pnls<0])
> # Calculate the in-sample and out-of-sample Sharpe and Sortino rat
> wealthv <- cbind(respv, pnls)
> colnames(wealthv) <- c("VTI", "Kitchen sink")
> sqrt(252)*sapply(wealthv[insample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv[outsample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategies
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(wealthv))
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Kitchen Sink Autoregressive Strategy") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(show="always", width=300)
```

Kitchen Sink Strategy With Dimension Reduction

Dimension reduction improves the out-of-sample, risk-adjusted performance of the multifactor autoregressive strategy.

The best performance is achieved with the intermediate value of the order parameter equal to $dimax = 5$.

```
> # Calculate the in-sample SVD
> svdec <- svd(predm[insample, ])
> # Calculate the in-sample fitted AR coefficients for different dim
> dimv <- 2:7
> # dimv <- c(2, 5, 10, NCOL(predm))
> coeffm <- sapply(dimv, function(dimax) {
+   predinv <- svdec$v[, 1:dimax] %*%
+     (t(svdec$u[, 1:dimax]) / svdec$d[1:dimax])
+   predinv %*% respv[insample]
+ })  # end lapply
> colnames(coeffm) <- paste0("dim=", dimv)
> colorv <- colorRampPalette(c("red", "blue"))(NCOL(coeffm))
> matplot(y=coeffm, type="l", lty="solid", lwd=1, col=colorv,
+   xlab="predictor", ylab="coeff",
+   main="AR Coefficients For Different Dimensions")
> # Calculate the forecasts of VTI
> fcasts <- predm %*% coeffm
> fcasts <- apply(fcasts, 2, function(x) {
+   fcastv <- sqrt(HighFreq::run_var(matrix(x), lambda=0.8)[, 2])
+   fcastv[1:10] <- 1 # Warmup
+   x/fcastv
+ }) # end apply
> # Simulate the autoregressive strategies
> retn <- coredata(respv)
> pnls <- apply(fcasts, 2, function(x) (x*retn))
> pnls <- xts(pnls, datev)
> # Scale the PnL volatility to that of VTI
> pnls <- lapply(pnls, function(x) x/sd(x))
> pnls <- sd(respv)*do.call(cbind, pnls)
```




```
> # Calculate the in-sample and out-of-sample Sharpe and Sortino rat
> wealthv <- cbind(respv, pnls)
> sqrt(252)*sapply(wealthv[insample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv[outsample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategies
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(wealthv))
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Kitchen Sink Strategies With Dimension Reduction") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(show="always", width=500)
```

Kitchen Sink Coefficients With Ridge Shrinkage

The *AR* coefficients can be found by minimizing the *MSE* of the in-sample forecasts.

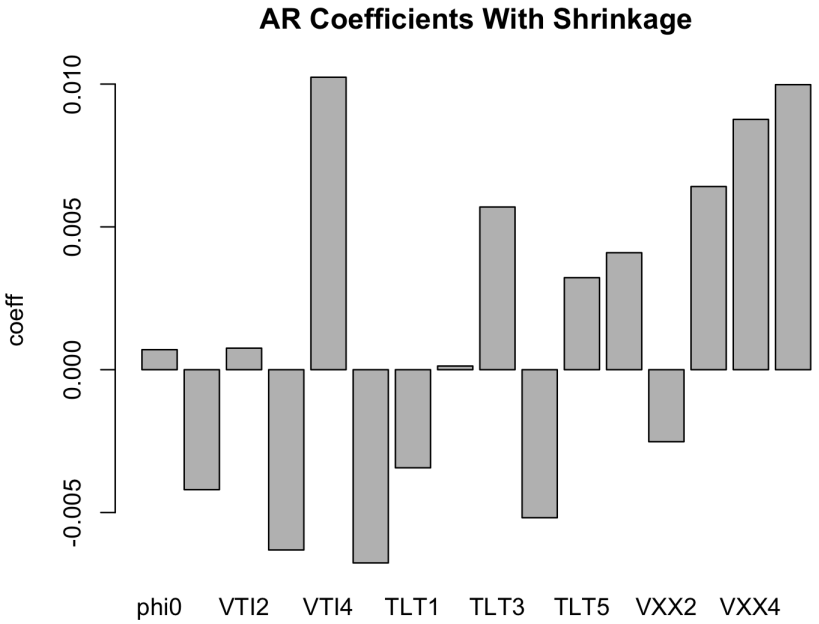
The objective function is the sum of the *MSE* plus a *ridge penalty* term proportional to the square of the *AR* coefficients:

$$\text{ObjFunc} = \sum_{i=1}^{n}(r_t - f_t)^2 + \lambda \sum_{i=1}^{k} \varphi_i^2$$

As the objective function is minimized, the *shrinkage* penalty shrinks the *AR* coefficients closer to zero.

A larger *shrinkage factor* λ applies more *shrinkage* to the *AR* coefficients, to shrink them closer to zero.

The coefficient shrinkage also produces a dimension reduction effect, because the higher order coefficients are shrunk closer to zero.




```
> # Objective function for the in-sample AR coefficients
> objfun <- function(coeff, respv, predm, lambdaf) {
+   fcasts <- predm %*% coeff
+   sum((respv - fcasts)^2) + lambdaf*sum(coeff^2)
+ }  # end objfun
```

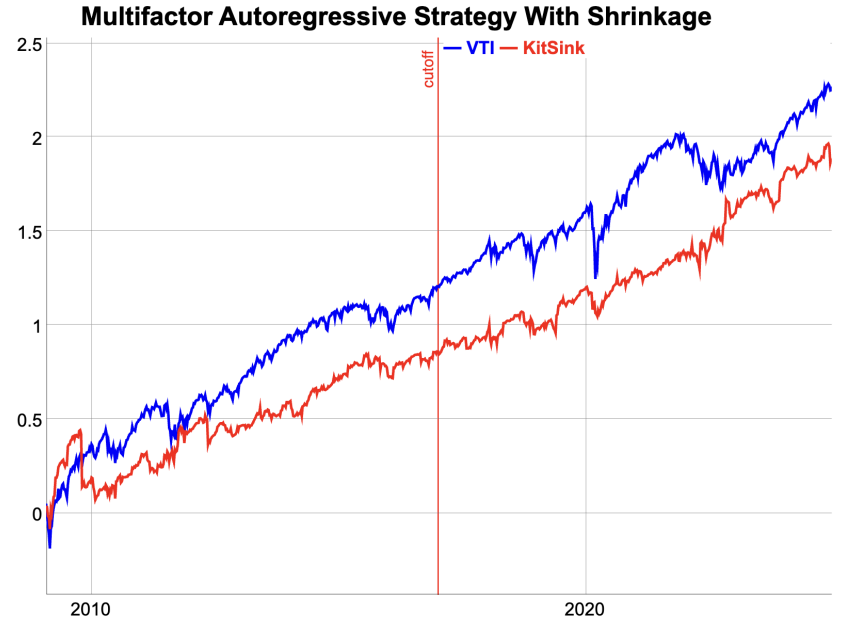
```
> # Perform optimization using the quasi-Newton method
> ncoeff <- NROW(coeff)
> optiml <- optim(par=numeric(ncoeff),
+           fn=objfun,
+           respv=respv[insample, ], predm=predm[insample, ],
+           lambdaf=1.0,
+           method="L-BFGS-B",
+           upper=rep(10, ncoeff),
+           lower=rep(-10, ncoeff))
> # Extract the AR coefficients
> coeff <- optiml$par
> names(coeff) <- colnames(predm)
> barplot(coeff, xlab="", ylab="coeff", col="grey",
+   main="AR Coefficients With Shrinkage")
```

Kitchen Sink Strategy With Shrinkage

The out-of-sample performance of the multifactor autoregressive strategy can be improved by applying *shrinkage* to the *AR* coefficients.

The value of the *shrinkage factor* λ can be chosen to maximize the out-of-sample performance.

```
> # Calculate the forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> # Calculate the autoregressive strategy PnLs
> pnls <- respv*fcasts
> pnls <- pnls*sd(respv[respv<0])/sd(pnls[pnls<0])
```




```
> # Calculate the in-sample and out-of-sample Sharpe and Sortino rat
> wealthv <- cbind(respv, pnls)
> colnames(wealthv) <- c("VTI", "KitSink")
> sqrt(252)*sapply(wealthv[insample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv[outsample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategies
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(wealthv))
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Multifactor Autoregressive Strategy With Shrinkage") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(show="always", width=300)
```

draft: Multifactor Autoregressive Strategy With Features

When *VXX* and *SVXY* returns are used as predictors of *VTI* returns, then the coefficients are overfitted to profit from single events, like the 2018 flash crash and the 2020 pandemic crash. But the model doesn't perform well outside of those single events.

The stock returns r_t can be fitted into an *autoregressive* model *AR(n)* with a constant intercept term φ_0:

$$r_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n} + \varepsilon_t$$

The *residuals* ε_t are assumed to be normally distributed, independent, and stationary.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns[, c("VTI", "VXX", "SVXY")])
> nrows <- NROW(retp)
> # Define the response and predictor matrices
> respv <- retp["/2019", "VTI"]
> orderp <- 3
> predm <- lapply(1:orderp, rutils::lagit, input=retp["/2019", c("VX
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> colnames(predm) <- c("phi0", paste0(c("VXX", "SVXY"), rep(1:orderp
> # Calculate the fitted autoregressive coefficients
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
> # Calculate the residuals (forecast errors)
> resids <- (fcasts - respv)
> # The residuals are orthogonal to the predictors and the forecasts
> round(cor(resids, fcasts), 6)
> round(sapply(predm[, -1], function(x) cor(resids, x)), 6)
```

draft: Forecasting Stock Returns Using Autoregressive Models

Explain why the sum of the *AR* coefficients is negative, even though the stock returns are positive on average.

The fitted autoregressive coefficients φ are equal to the *response* $\mathbf{r}$ multiplied by the inverse of the *predictor matrix* $\mathbb{P}$:

$$\varphi = \mathbb{P}^{-1} \mathbf{r}$$

The *in-sample* autoregressive forecasts of the returns are calculated by multiplying the predictor matrix by the fitted *AR* coefficients:

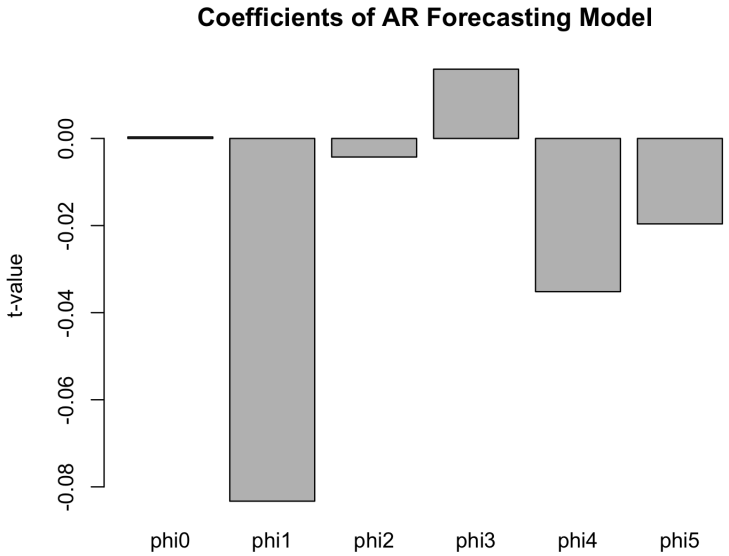
$$f_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n}$$

For *VTI* returns, the intercept coefficient φ_0 has a small positive value, while the first autoregressive coefficient φ_1 has a small negative value.

This means that the autoregressive forecasting model is a combination of a static long stock position, plus a mean-reverting model which switches its stock position to the reverse of the previous day's return.

The function `MASS::ginv()` calculates the generalized inverse of a matrix.

```
> # Calculate the fitted autoregressive coefficients
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> sum(coeff[1:orderp])
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
```




```
> # Plot the AR coefficients
> coeffn <- paste0("phi", 0:(NROW(coeff)-1))
> barplot(coeff ~ coeffn, xlab="", ylab="coeff", col="grey",
+   main="Coefficients of AR Forecasting Model")
```

draft: The t-values of the Autoregressive Coefficients

The forecast residuals are equal to the differences between the return forecasts minus the actual returns: $\varepsilon = f_t - r_t$

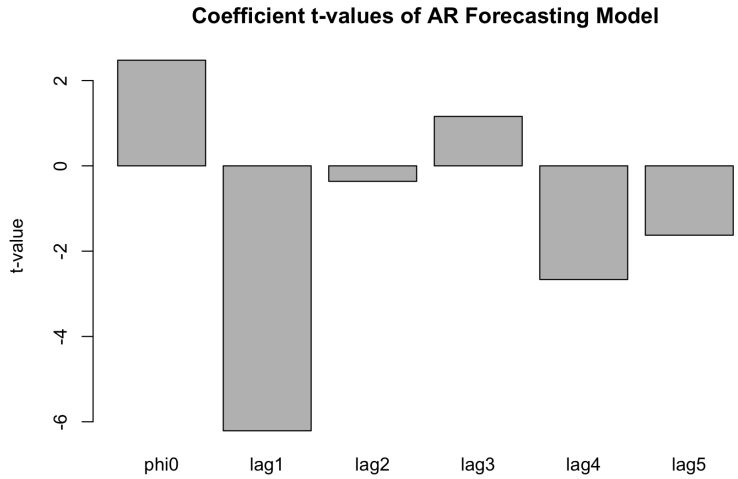
The variance of the autoregressive coefficients σ^2_{φ} is equal to the variance of the forecast residuals σ^2_{ε} divided by the squared *predictor matrix* $\mathbb{P}$:

$$\sigma^2_{\varphi} = \sigma^2_{\varepsilon}(\mathbb{P}^T\mathbb{P})^{-1}$$

The t-values of the autoregressive coefficients are equal to the coefficient values divided by their volatilities:

$$\varphi_{tval} = \frac{\varphi}{\sigma_{\varphi}}$$

The intercept coefficient φ_0 and the first autoregressive coefficient φ_1 have statistically significant t-values.




```
> # Calculate the residuals (forecast errors)
> resids <- (fcasts - respv)
> # The residuals are orthogonal to the predictors and the forecast
> round(cor(resids, fcasts), 6)
> round(sapply(predm[, -1], function(x) cor(resids, x)), 6)
> # Calculate the variance of the residuals
> varv <- sum(resids^2)/(nrows-NROW(coeff))
```

```
> # Calculate the predictor matrix squared
> pred2 <- crossprod(predm)
> # Calculate the covariance matrix of the AR coefficients
> covmat <- varv*MASS::ginv(pred2)
> coefsd <- sqrt(diag(covmat))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coefsd)
> coeffn <- paste0("phi", 0:(NROW(coeff)-1))
> # Plot the t-values of the AR coefficients
> barplot(coefft ~ coeffn, xlab="", ylab="t-value", col="grey",
+   main="Coefficient t-values of AR Forecasting Model")
```

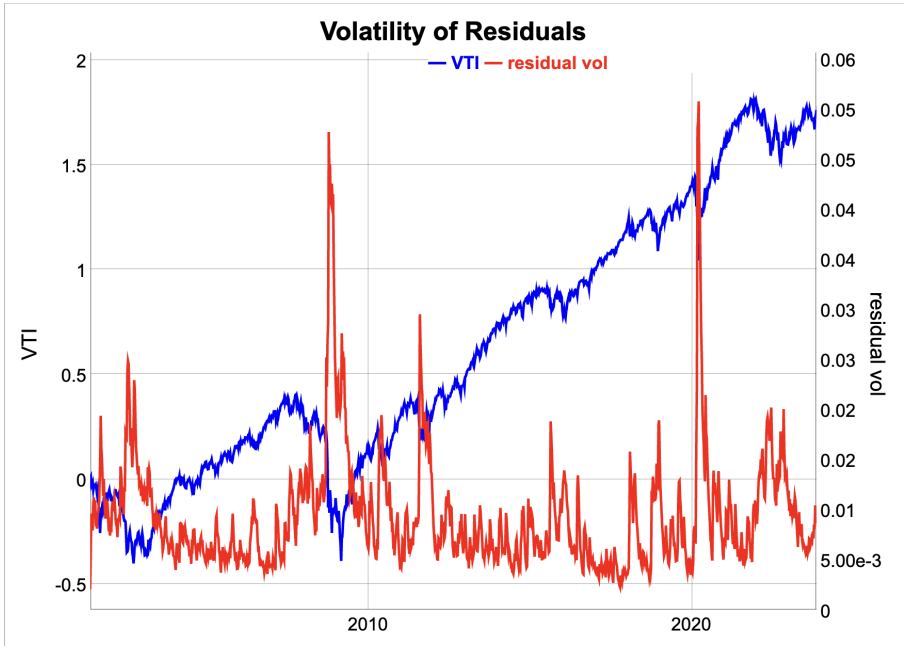
draft: Residuals of Autoregressive Forecasting Model

The autoregressive model assumes stationary returns and residuals, with similar volatility over time.

In reality stock volatility is highly time dependent, so the volatility of the residuals is also time dependent.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Calculate the trailing volatility of the residuals
> residv <- sqrt(HighFreq::run_var(resids, lambda=0.9)[, 2])
```




```
> # Plot dygraph of volatility of residuals
> datav <- cbind(cumsum(retp), residv)
> colnames(datav) <- c("VTI", "residual vol")
> endw <- rutils::calc_endpoints(datav, interval="weeks")
> dygraphs::dygraph(datav[endw], main="Volatility of Residuals") %>%
+   dyAxis("y", label="VTI", independentTicks=TRUE) %>%
+   dyAxis("y2", label="residual vol", independentTicks=TRUE) %>%
+   dySeries(name="VTI", axis="y", strokeWidth=2, col="blue") %>%
+   dySeries(name="residual vol", axis="y2", strokeWidth=2, col="red
+   dyLegend(show="always", width=300)
```

draft: Autoregressive Strategy In-Sample

The kitchen sink strategy has more predictor variables (and more degrees of freedom), so in-sample it can forecast the returns more closely.

The first step in strategy development is optimizing it in-sample, even though in practice it can't be implemented. Because a strategy can't perform well out-of-sample if it doesn't perform well in-sample.

The autoregressive strategy invests dollar amounts of *VTI* stock proportional to the in-sample forecasts.

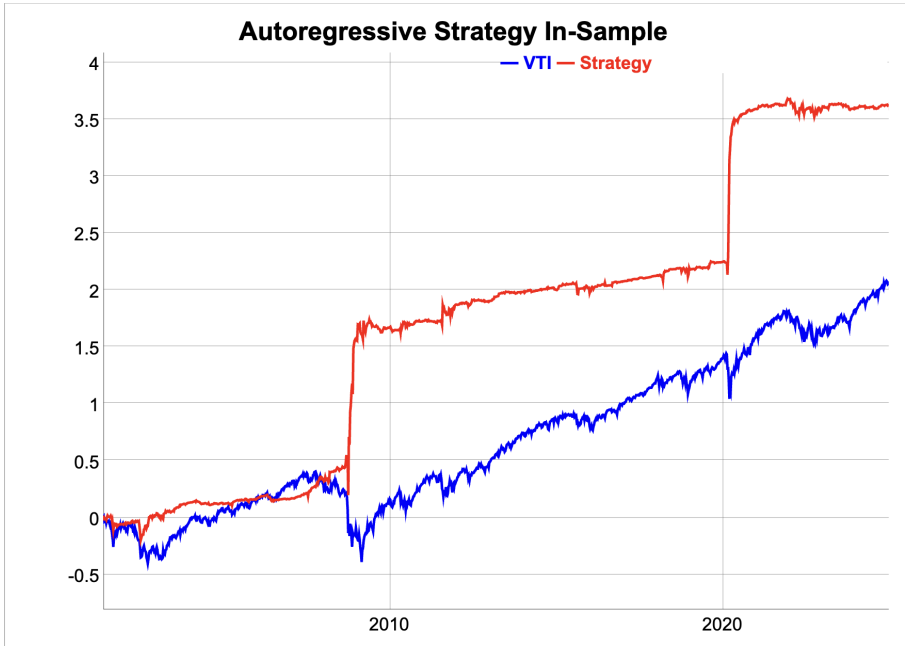
The in-sample autoregressive strategy performs well during periods of high volatility, but not as well in low volatility periods.

The dollar allocations of *VTI* stock are too large in periods of high volatility, which causes over-leverage and higher risk.

The leverage can be reduced by scaling (dividing) the forecasts by their trailing volatility.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Calculate the kitchen sink strategy in-sample
> pnls <- retp*fcasts
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

draft: Out-of-Sample Performance of the Kitchen Sink Strategy

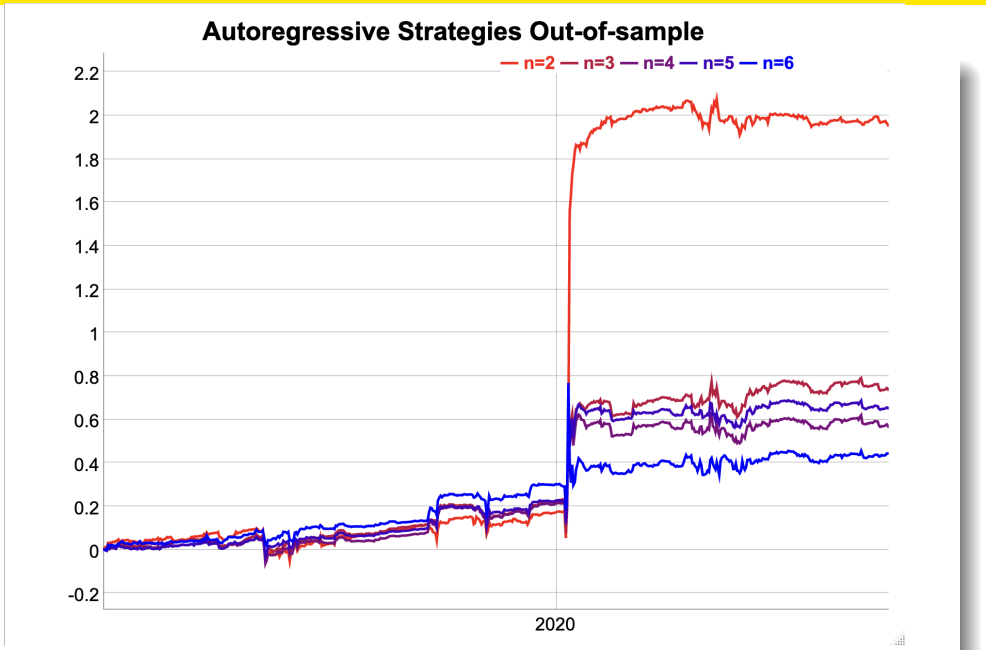
The out-of-sample performance of the kitchen sink strategy is much worse than its in-sample performance, because the strategy is overfit to the in-sample data.

The autoregressive strategy invests dollar amounts of *VTI* proportional to the AR forecasts.

The out-of-sample, risk-adjusted performance of the autoregressive strategy is better for a smaller order parameter n of the *AR(n)* model.

The optimal order parameter is `n = 2`, with a positive intercept coefficient φ_0 (since the average *VTI* returns were positive), and a negative coefficient φ_1 (because of strong negative autocorrelations in periods of high volatility).

Decreasing the order parameter of the autoregressive model is a form of *shrinkage* because it reduces the number of predictive variables.




```
> # Define in-sample and out-of-sample intervals
> nrows <- NROW(retp)
> cutoff <- nrows %/% 2
> datev[cutoff]
> insample <- 1:cutoff
> outsample <- (cutoff + 1):nrows
> # Calculate the optimal AR coefficients
> predinv <- MASS::ginv(predm[insample, ])
> coeff <- drop(predinv %*% respv[insample])
> # Calculate the strategy PnLs
> fcasts <- predm %*% coeff
> pnls <- sign(fcasts)*retp
> wealthv <- cbind(retp, pnls)
```

```
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[insample, ],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[outsample, ],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Kitchen Sink Strategy Out-of-sample") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(show="always", width=300)
```

draft: Out-of-Sample Performance With Dimension Reduction

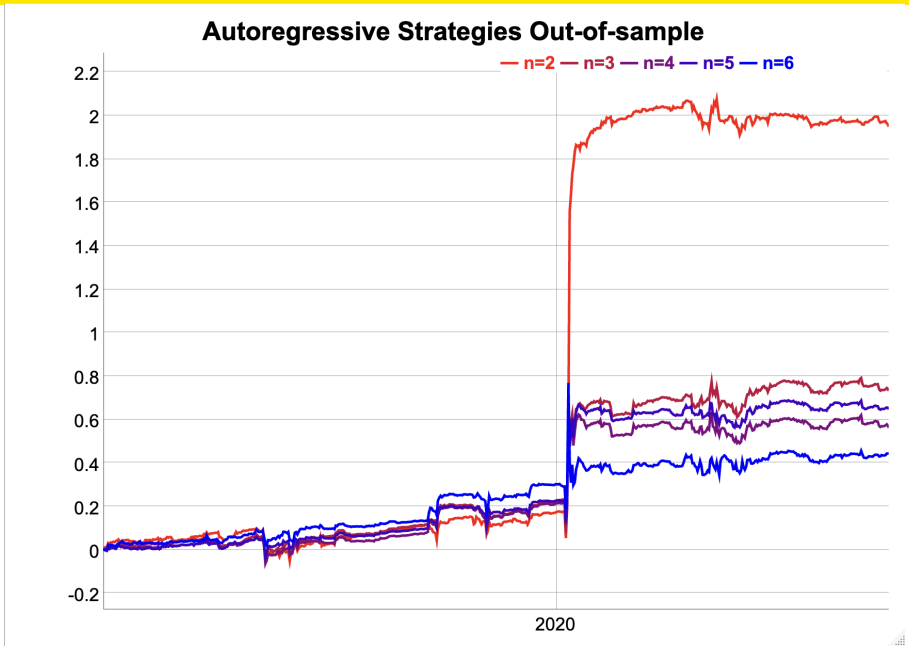
The out-of-sample performance of the kitchen sink strategy is much worse than its in-sample performance, because the strategy is overfit to the in-sample data.

The autoregressive strategy invests dollar amounts of *VTI* proportional to the AR forecasts.

The out-of-sample, risk-adjusted performance of the autoregressive strategy is better for a smaller order parameter n of the *AR(n)* model.

The optimal order parameter is `n = 2`, with a positive intercept coefficient φ_0 (since the average *VTI* returns were positive), and a negative coefficient φ_1 (because of strong negative autocorrelations in periods of high volatility).

Decreasing the order parameter of the autoregressive model is a form of *shrinkage* because it reduces the number of predictive variables.




```
> # Define in-sample and out-of-sample intervals
> nrows <- NROW(retp)
> cutoff <- nrows %/% 2
> datev[cutoff]
> insample <- 1:cutoff
> outsample <- (cutoff + 1):nrows
> # Calculate reduced inverse of the predictor matrix from SVD
> svdec <- svd(predm[insample, ])
> dimax <- 2
> predinv <- svdec$v[, 1:dimax] %*%
+   (t(svdec$u[, 1:dimax]) / svdec$d[1:dimax])
> coeff <- drop(predinv %*% respv[insample])
> # Calculate the strategy PnLs
```

```
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[insample, ],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[outsample, ],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Kitchen Sink Strategy Out-of-sample") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(show="always", width=300)
```

draft: Out-of-Sample Autoregressive Forecasts

The autoregressive coefficients φ are equal to the in-sample *response* $\mathbf{r}$ times the inverse of the in-sample *predictor matrix* $\mathbb{P}$:

$$\varphi = \mathbb{P}^{-1} \mathbf{r}$$

The variance of the autoregressive coefficients σ_{φ}^2 is equal to the variance of the in-sample forecast residuals σ_{ε}^2 divided by the squared *predictor matrix* $\mathbb{P}$:

$$\sigma_{\varphi}^2 = \sigma_{\varepsilon}^2 (\mathbb{P}^T \mathbb{P})^{-1}$$

The t-values of the autoregressive coefficients are equal to the coefficient values divided by their volatilities:

$$\varphi_{tval} = \frac{\varphi}{\sigma_{\varphi}}$$

The *out-of-sample* autoregressive forecast f_t is equal to the single row of the predictor $\mathbb{P}_t$ times the fitted *AR* coefficients φ:

$$f_t = \varphi \mathbb{P}_t$$

The variance σ_f^2 of the *forecast value* is equal to the inner product of the predictor $\mathbb{P}_t$ times the coefficient covariance matrix σ_{φ}^2:

$$\sigma_f^2 = \mathbb{P}_t \, \sigma_{\varphi}^2 \, \mathbb{P}_t^T$$

```
> # Define the look-back range
> lookb <- 100
> tday <- nrows
> startp <- max(1, tday-lookb)
> rangev <- startp:(tday-1)
> # Subset the response and predictors
> resps <- respv[rangev]
> preds <- predm[rangev]
> # Invert the predictor matrix
> predinv <- MASS::ginv(preds)
> # Calculate the fitted AR coefficients
> coeff <- predinv %*% resps
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- preds %*% coeff
> # Calculate the residuals (forecast errors)
> resids <- (fcasts - resps)
> # Calculate the variance of the residuals
> varv <- sum(resids^2)/(NROW(preds)-NROW(coeff))
> # Calculate the predictor matrix squared
> pred2 <- crossprod(preds)
> # Calculate the covariance matrix of the AR coefficients
> covmat <- varv*MASS::ginv(pred2)
> coefsd <- sqrt(diag(covmat))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coefsd)
> # Calculate the out-of-sample forecast
> predn <- predm[tday, ]
> fcast <- drop(predn %*% coeff)
> # Calculate the variance of the forecast
> varf <- drop(predn %*% covmat %*% t(predn))
> # Calculate the t-value of the out-of-sample forecast
> fcast/sqrt(varf)
```

draft: Rolling Autoregressive Forecasting Model

The autoregressive coefficients can be calibrated dynamically over a *rolling* look-back interval, and applied to calculating the *out-of-sample* forecasts.

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

The *backtesting* procedure allows determining the optimal *meta-parameters* of the forecasting model: the order *n* of the *AR(n)* model and the length of look-back interval (`lookb`).

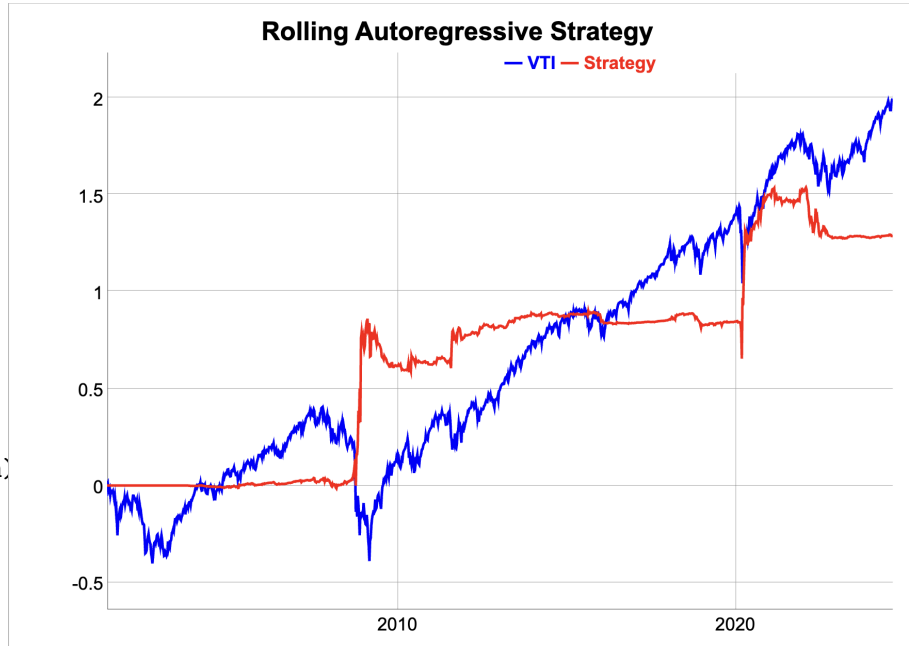
```
> # Perform rolling forecasting
> lookb <- 100
> fcasts <- sapply(1:nrows, function(tday) {
+   if (tday > lookb) {
+     # Define the rolling look-back range
+     startp <- max(1, tday-lookb)
+     # startp <- 1 # Expanding look-back range
+     rangev <- startp:(tday-1) # In-sample range
+     # Subset the response and predictors
+     resps <- respv[rangev]
+     preds <- predm[rangev]
+     # Calculate the fitted AR coefficients
+     predinv <- MASS::ginv(preds)
+     coeff <- predinv %*% resps
+     # Calculate the in-sample forecasts of VTI (fitted values)
+     fcasts <- preds %*% coeff
+     # Calculate the residuals (forecast errors)
+     resids <- (fcasts - resps)
+     # Calculate the variance of the residuals
+     varv <- sum(resids^2)/(NROW(preds)-NROW(coeff))
+     # Calculate the covariance matrix of the AR coefficients
+     pred2 <- crossprod(preds)
+     covmat <- varv*MASS::ginv(pred2)
+     coefsd <- sqrt(diag(covmat))
+     coefft <- drop(coeff/coefsd) # t-values of the AR coefficients
+     # Calculate the out-of-sample forecast
+     predn <- predm[tday, ]
+     fcast <- drop(predn %*% coeff)
+     # Calculate the variance of the forecast
+     varf <- drop(predn %*% covmat %*% t(predn))
+     return(c(sd(resps), fcast=fcast, fstderr=sqrt(varf), coefft=co
+   } else {
+     return(c(volv=0, fcast=0, fstderr=0, coefft=rep(0, NCOL(predm)
+   } # end if
+ })  # end sapply
```

draft: Rolling Autoregressive Strategy Performance

In the rolling autoregressive strategy, the autoregressive coefficients are calibrated on past data from a *rolling* look-back interval, and applied to calculating the *out-of-sample* forecasts.

The rolling autoregressive strategy performance depends on the length of the look-back interval.

```
> # Coerce fcasts to a time series
> fcasts <- t(fcasts)
> ncols <- NCOL(fcasts)
> colnames(fcasts) <- c("volv", "fcasts", "fstderr", colnames(predm)
> fcasts <- xts::xts(fcasts, zoo::index(retp))
> # Calculate the strategy PnLs
> pnls <- retp*fcasts$fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+ c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot dygraph of the autoregressive strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Rolling Autoregressive Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+ dyLegend(show="always", width=300)
```

draft: Backtesting the Autoregressive Model

The *meta-parameters* of the *backtesting* function are the order *n* of the *AR(n)* model and the length of the look-back interval (`lookb`).

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

The *backtesting* procedure allows determining the optimal *meta-parameters* of the forecasting model: the order *n* of the *AR(n)* model and the length of look-back interval (`lookb`).

```
> # Define backtesting function
> sim_fcasts <- function(lookb=100, ordern=5, fixedlb=TRUE) {
+   # Perform rolling forecasting
+   fcasts <- sapply((lookb+1):nrows, function(tday) {
+     # Rolling look-back range
+     startp <- max(1, tday-lookb)
+     # Expanding look-back range
+     if (!fixedlb) {startp <- 1}
+     startp <- max(1, tday-lookb)
+     rangev <- startp:(tday-1) # In-sample range
+     # Subset the response and predictors
+     resps <- respv[rangev]
+     preds <- predm[rangev, 1:ordern]
+     # Invert the predictor matrix
+     predinv <- MASS::ginv(preds)
+     # Calculate the fitted AR coefficients
+     coeff <- predinv %*% resps
+     # Calculate the out-of-sample forecast
+     drop(predm[tday, 1:ordern] %*% coeff)
+   })  # end sapply
+   # Add warmup period
+   fcasts <- c(rep(0, lookb), fcasts)
+ }  # end sim_fcasts
> # Simulate the rolling autoregressive forecasts
> fcasts <- sim_fcasts(lookb=100, ordern=5)
> c(mse=mean((fcasts - retp)^2), cor=cor(retp, fcasts))
```

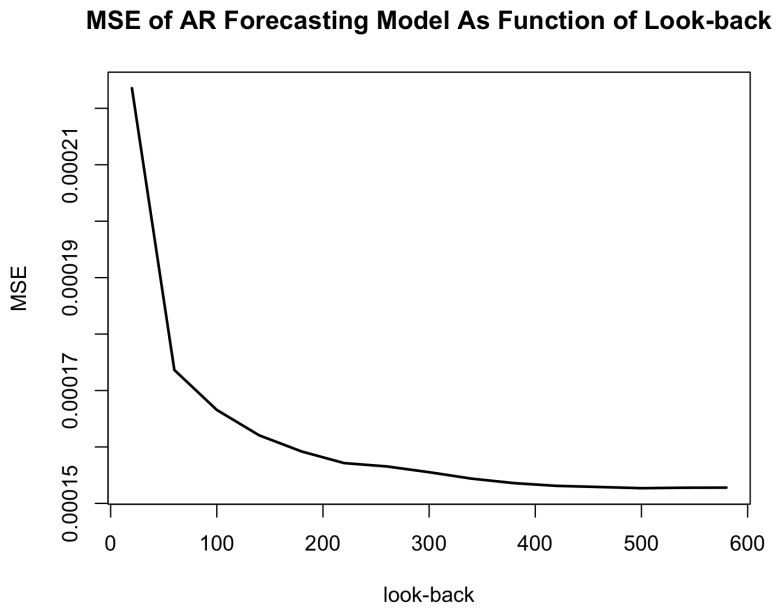
draft: Forecasting Dependence On the Look-back Interval

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`lookb`).

The accuracy of the forecasting model increases with longer look-back intervals (`lookb`), because more data improves the estimates of the autoregressive coefficients.




```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> compclust <- makeCluster(ncores)
> # Perform parallel loop under Windows
> lookbv <- seq(20, 600, 40)
> fcasts <- parLapply(compclust, lookbv, sim_fcasts, ordern=6)
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(lookbv, sim_fcasts, ordern=6, mc.cores=ncores)
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- lookbv
> # Select optimal lookb interval
> lookb <- lookbv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=lookbv, y=mse[, 1],
+   xlab="look-back", ylab="MSE", type="l", lwd=2,
+   main="MSE of AR Forecasting Model As Function of Look-back")
```

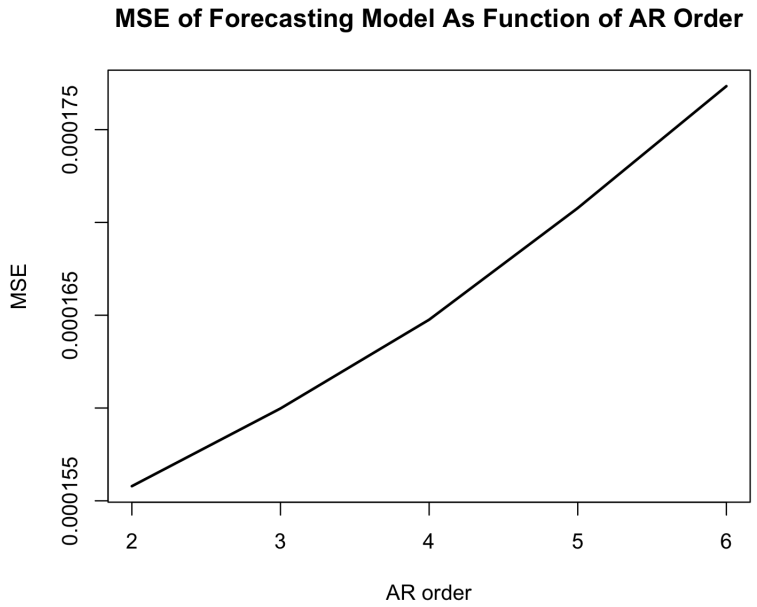
draft: Order Dependence With Fixed Look-back

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`lookb`).

The accuracy of the forecasting model decreases for larger AR order parameters, because of overfitting in-sample.




```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> compclust <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(compclust, orderv, sim_fcasts, lookb=lookb)
> stopCluster(compclust)  # Stop R processes over cluster under Wi
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   lookb=lookb, mc.cores=ncores)
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, y=mse[, 1],
+   xlab="AR order", ylab="MSE", type="l", lwd=2,
+   main="MSE of Forecasting Model As Function of AR Order")
```

draft: Autoregressive Strategy With Fixed Look-back

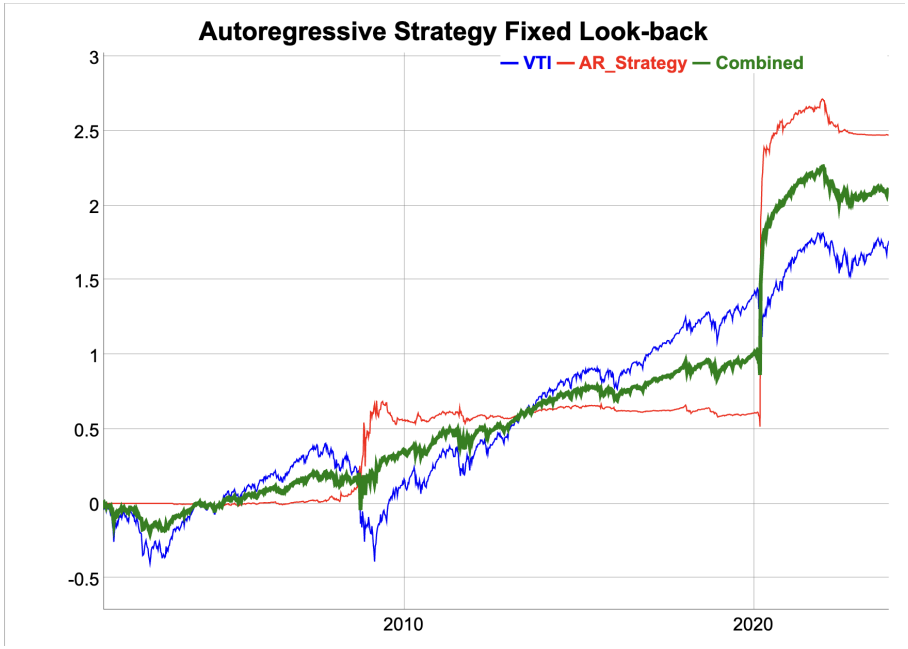
The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

Using the return forecasts as portfolio weights produces very large weights in periods of high volatility, and creates excessive risk.

To reduce excessive risk, a binary strategy can be used, with portfolio weights equal to the sign of the forecasts.

```
> # Simulate the rolling autoregressive forecasts
> fcasts <- sim_fcasts(lookb=lookb, ordern=ordern)
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```




```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy Fixed Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

draft: Order Dependence With Expanding Look-back

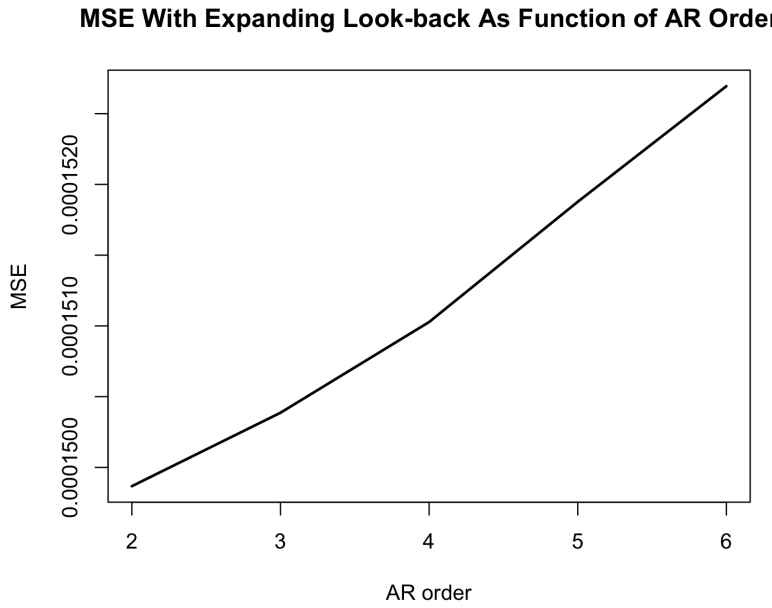
The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model.

Longer look-back intervals (`lookb`) are usually better for the autoregressive forecasting model.

The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> compclust <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(compclust, orderv, sim_fcasts,
+   lookb=lookb, fixedlb=FALSE)
> stopCluster(compclust)  # Stop R processes over cluster under Wi
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   lookb=lookb, fixedlb=FALSE, mc.cores=ncores)
```



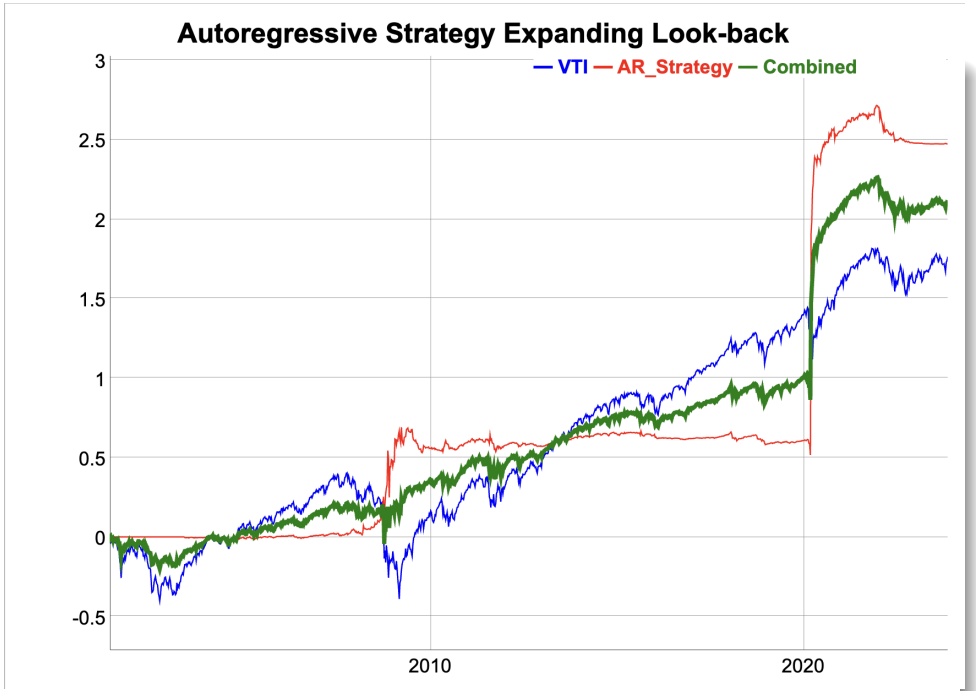

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, y=mse[, 1],
+   xlab="AR order", ylab="MSE", type="l", lwd=2,
+   main="MSE With Expanding Look-back As Function of AR Order")
```

draft: Autoregressive Strategy With Expanding Look-back

The model with an *expanding* look-back interval has better performance compared to the *fixed* look-back interval.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

```
> # Simulate the autoregressive forecasts with expanding look-back
> fcasts <- sim_fcasts(lookb=lookb, ordern=ordern, fixedlb=FALSE)
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```




```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy Expanding Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

draft: Multifactor Autoregressive Strategy of Stock Returns

When *VXX* and *SVXY* returns are used as predictors of *VTI* returns, then the coefficients are overfitted to profit from single events, like the 2018 flash crash and the 2020 pandemic crash. But the model doesn't perform well outside of those single events.

The stock returns r_t can be fitted into an *autoregressive* model *AR(n)* with a constant intercept term φ_0:

$$r_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n} + \varepsilon_t$$

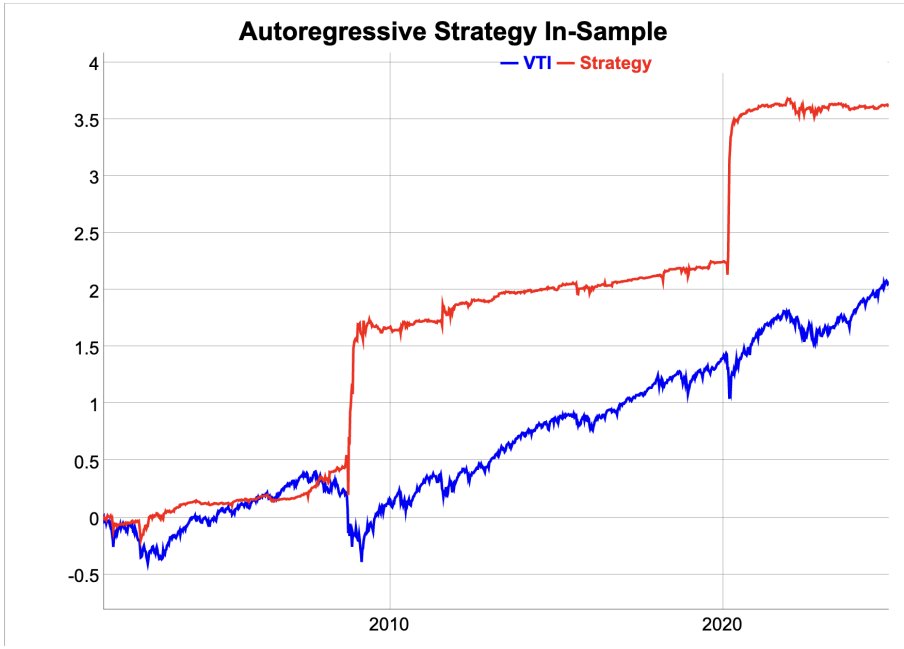
The *residuals* ε_t are assumed to be normally distributed, independent, and stationary.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns[, c("VTI", "VXX", "SVXY")])
> nrows <- NROW(retp)
> # Define the response and predictor matrices
> respv <- retp["/2019", "VTI"]
> orderp <- 3
> predm <- lapply(1:orderp, rutils::lagit, input=retp["/2019", c("VX
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> colnames(predm) <- c("phi0", paste0(c("VXX", "SVXY"), rep(1:orderp
> # Calculate the fitted autoregressive coefficients
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
> # Calculate the residuals (forecast errors)
> resids <- (fcasts - respv)
> # The residuals are orthogonal to the predictors and the forecasts
> round(cor(resids, fcasts), 6)
> round(sapply(predm[, -1], function(x) cor(resids, x)), 6)
```

draft: Multifactor Autoregressive Strategy In-Sample

The coefficients are calibrated with returns before 2020, but the model still makes profit during the 2020 stock crash.

```
> # Calculate the autoregressive strategy PnLs
> pnls <- respv*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(respv)/sd(pnls)
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(respv, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Autoregressive Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Interest Rate Yield Curve and Stock Returns

The level of interest rates and the shape of the yield curve can be used to forecast stock returns.

The *FRED* database provides historical constant maturity Treasury yields:
https://fred.stlouisfed.org/series/DGS5

```
> # Download time series of bond yields
> # symbolv <- c("DGS1", "DGS2", "DGS5", "DGS10", "DGS20", "DGS30")
> # ratesenv <- new.env()
> # quantmod::getSymbols(symbolv, env=ratesenv, src="FRED")
> # Load constant maturity Treasury rates
> load(file="/Users/jerzy/Develop/lecture_slides/data/rates_data.RData")
> # Combine rates into single xts series
> ratev <- do.call(cbind, as.list(ratesenv))
> # Sort the columns of rates according bond maturity
> namev <- colnames(ratev)
> namev <- substr(namev, start=4, stop=10)
> namev <- as.numeric(namev)
> indeks <- order(namev)
> ratev <- ratev[, indeks]
> # Align rates dates with VTI prices
> closep <- log(quantmod::Cl(rutils::etfenv$VTI))
> colnames(closep) <- "VTI"
> datev <- zoo::index(closep)
> ratev <- na.omit(ratev[datev])
> closep <- closep[zoo::index(ratev)]
> datev <- zoo::index(closep)
> nrows <- NROW(closep)
```

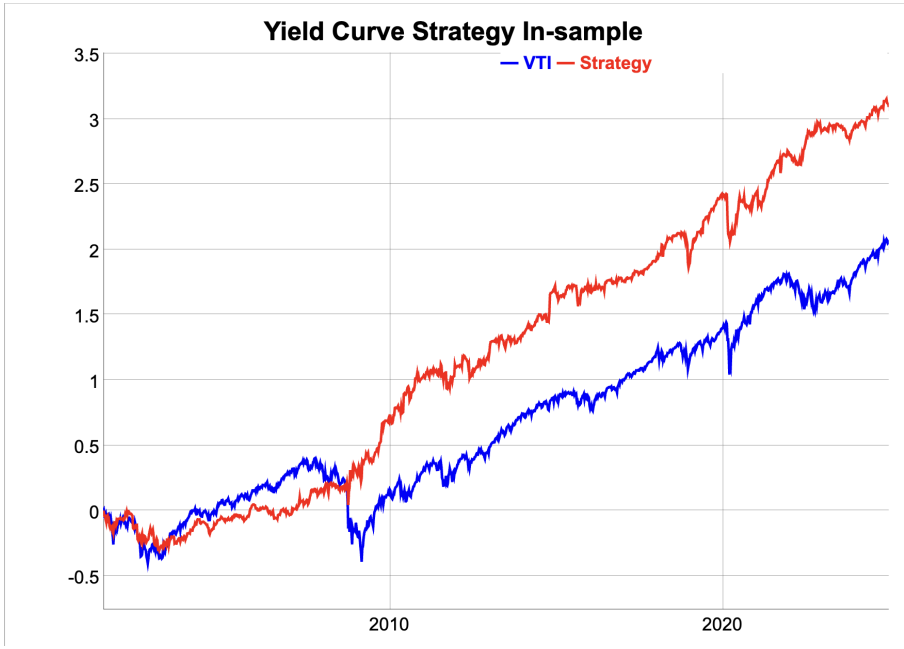
```
> # Calculate VTI returns and IR changes
> retp <- rutils::diffit(closep)
> retr <- rutils::diffit(ratev)
> # Regress VTI returns versus the lagged rate differences
> predm <- rutils::lagit(retr)
> regmod <- lm(retp ~ predm)
> summary(regmod)
> # Regress VTI returns before and after 2010
> summary(lm(retp["/2010"] ~ predm["/2010"]))
> summary(lm(retp["2010/"] ~ predm["2010/"]))
```

Yield Curve Strategy In-Sample

For in-sample forecasts, the training set and the test set are the same. The model is calibrated on the data that is used for forecasting.

Although it's not realistic to achieve the in-sample performance, it's useful because it provides insights into how the model works.

The in-sample strategy performs well in periods of high volatility, but not as well in periods of low volatility.



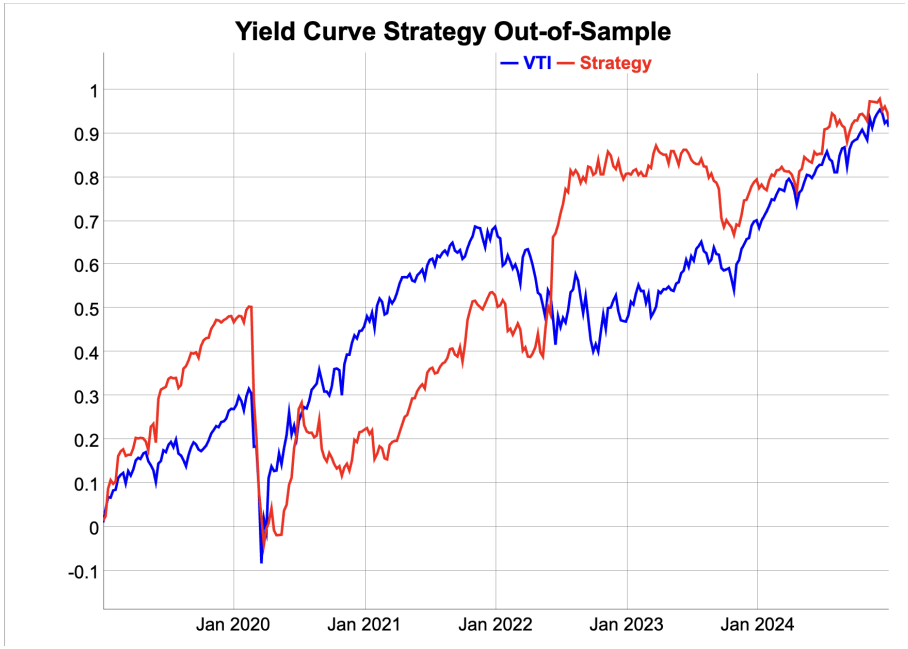

```
> # Define predictor with intercept term
> predm <- rutils::lagit(retr)
> predm <- cbind(rep(1, NROW(predm)), predm)
> colnames(predm)[1] <- "intercept"
> # Calculate inverse of predictor
> invreg <- MASS::ginv(predm)
> # Calculate coefficients from response and inverse of predictor
> respv <- retp
> coeff <- drop(invreg %*% respv)
> # Calculate forecasts and PnLs in-sample
> fcasts <- (predm %*% coeff)
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> pnls <- fcasts*respv
> # Calculate in-sample factors
> factv <- (predm*coeff)
> apply(factv, 2, sd)
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(respv)/sd(pnls)
```

```
> # Plot dygraph of in-sample YC strategy
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> colv <- colnames(wealthv)
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Yield Curve Strategy In-sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Yield Curve Strategy Out-of-Sample

For out-of-sample forecasts, the training set and the test set are separate. The model is calibrated on the training set, and forecasts are calculated using the test set.

```
> # Calculate inverse of predictor in-sample
> invreg <- MASS::ginv(predm["/2019"])
> # Calculate coefficients in-sample
> coeff <- drop(invreg %*% respv["/2019"])
> # Calculate forecasts and PnLs
> fcasts <- (predm %*% coeff)
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> pnls <- fcasts*respv
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(respv)/sd(pnls)
```

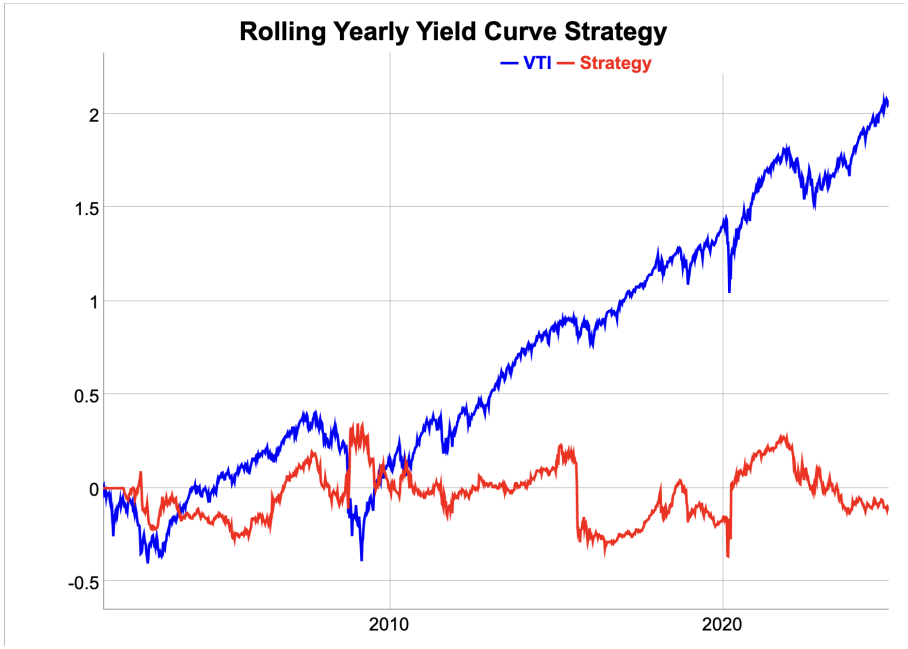



```
> # Plot dygraph of out-of-sample YC strategy
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colv <- colnames(wealthv)
> endw <- rutils::calc_endpoints(wealthv["2019/"], interval="weeks")
> dygraphs::dygraph(cumsum(wealthv["2019/"])[endw],
+   main="Yield Curve Strategy Out-of-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Yearly Yield Curve Strategy

In the rolling yearly yield curve strategy, the model is recalibrated at the end of every year using a training set of data from the past `lookb` days. The coefficients are applied to calculate out-of-sample forecasts in the following year.

```
> # Define yearly dates
> endd <- rutils::calc_endpoints(respv, interval="years")
> # endd <- index(closep)[endd]
> # Perform loop over yearly dates
> lookb <- 500
> fcasts <- lapply(seq_along(endd)[-1], function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (max(1, endd[tday-1]-lookb):endd[tday-1])
+   # insample <- (1:endd[tday-1]) # Expanding look-back
+   outsample <- (endd[tday-1]+1):endd[tday]
+   # Calculate coefficients in-sample
+   invreg <- MASS::ginv(predm[insample, ])
+   coeff <- drop(invreg %*% respv[insample, ])
+   # Calculate forecasts out-of-sample
+   fcasts <- (predm[outsample, ] %*% coeff)
+ })  # end lapply
> fcasts <- do.call(rbind, fcasts)
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> pnls <- fcasts*respv
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(respv)/sd(pnls)
```




```
> # Plot dygraph of rolling yearly YC strategy
> wealthv <- cbind(respv, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colv <- colnames(wealthv)
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Rolling Yearly Yield Curve Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

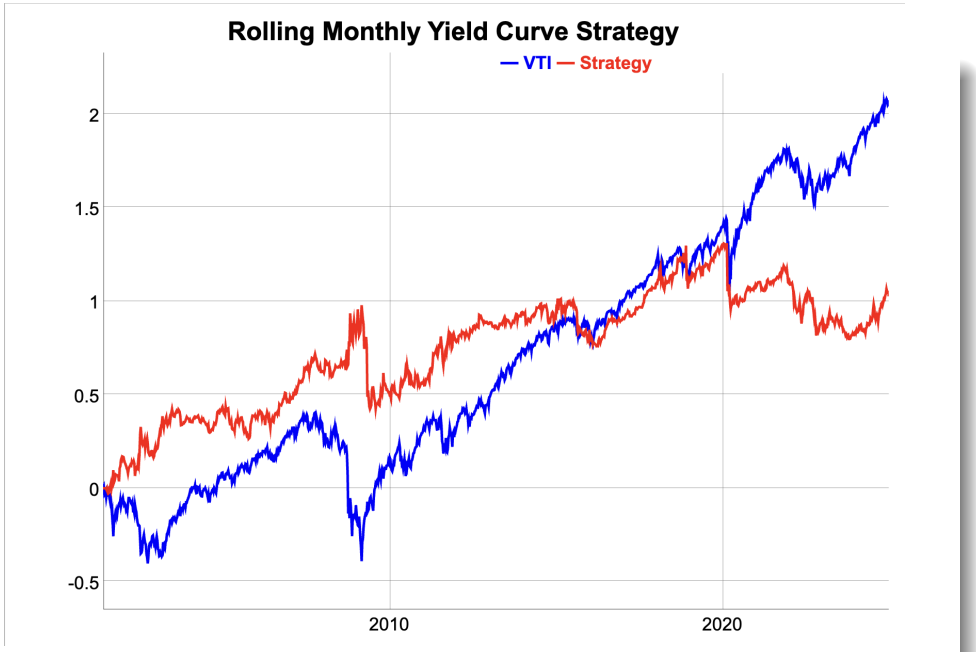
Rolling Monthly Yield Curve Strategy

In the rolling monthly yield curve strategy, the model is recalibrated at the end of every month using a training set of the past `lookb` days. The coefficients are applied to perform out-of-sample forecasts in the following month.

Research shows that looking back roughly a year provides the best out-of-sample forecasts.

The rolling monthly strategy performs better than the yearly strategy, but mostly in periods of high volatility, and otherwise it's flat.

```
> # Define monthly dates
> endd <- rutils::calc_endpoints(respv, interval="month")
> fcasts <- lapply(seq_along(endd)[-1], function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (max(1, endd[tday-1]-lookb):endd[tday-1])
+   # insample <- (1:endd[tday-1]) # Expanding look-back
+   outsample <- (endd[tday-1]+1):endd[tday]
+   # Calculate coefficients in-sample
+   invreg <- MASS::ginv(predm[insample, ])
+   coeff <- drop(invreg %*% respv[insample, ])
+   # Calculate forecasts out-of-sample
+   fcasts <- (predm[outsample, ] %*% coeff)
+ })  # end lapply
> fcasts <- do.call(rbind, fcasts)
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> pnls <- fcasts*respv
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(respv)/sd(pnls)
```

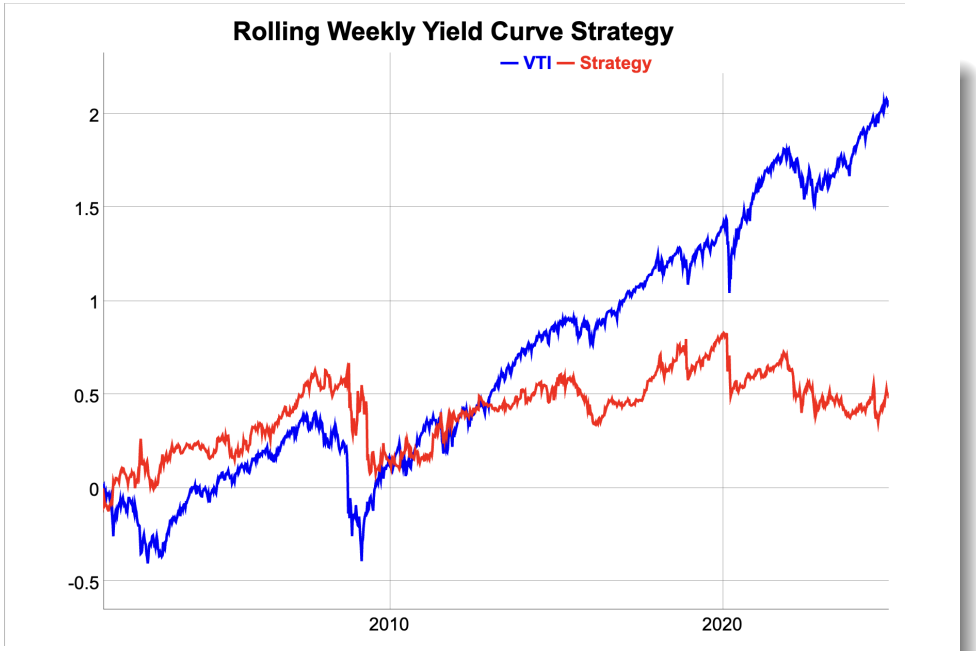



```
> # Plot dygraph of rolling monthly YC strategy
> wealthv <- cbind(respv, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colv <- colnames(wealthv)
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Rolling Monthly Yield Curve Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Weekly Yield Curve Strategy

In the rolling weekly yield curve strategy, the model is recalibrated at the end of every week using a training set of the past `lookb` days. The coefficients are applied to perform out-of-sample forecasts in the following week.

```
> # Define weekly dates
> endd <- rutils::calc_endpoints(respv, interval="weeks")
> fcasts <- lapply(seq_along(endd)[-1], function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (max(1, endd[tday-1]-lookb):endd[tday-1])
+   # insample <- (1:endd[tday-1]) # Expanding look-back
+   outsample <- (endd[tday-1]+1):endd[tday]
+   # Calculate coefficients in-sample
+   invreg <- MASS::ginv(predm[insample, ])
+   coeff <- drop(invreg %*% respv[insample, ])
+   # Calculate forecasts out-of-sample
+   fcasts <- (predm[outsample, ] %*% coeff)
+ })  # end lapply
> fcasts <- do.call(rbind, fcasts)
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> pnls <- fcasts*respv
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(respv)/sd(pnls)
```



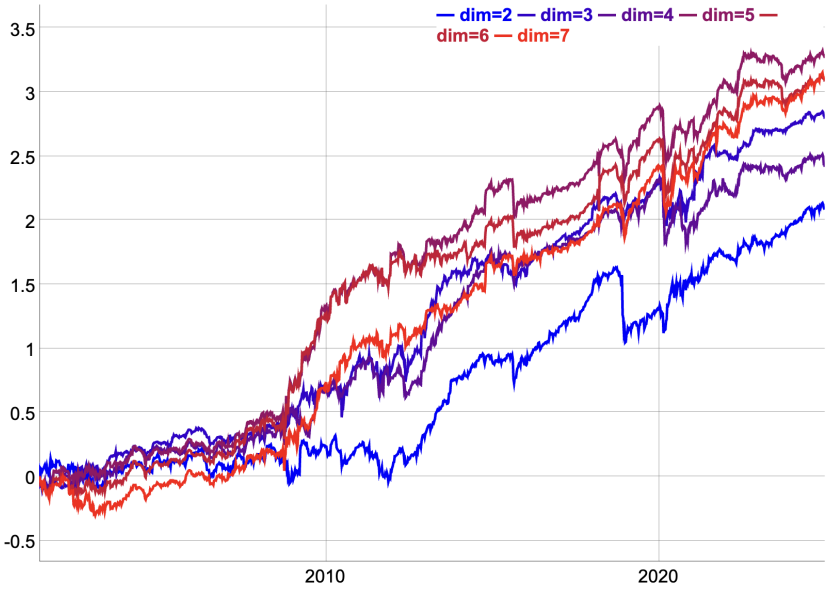

```
> # Plot dygraph of rolling weekly YC strategy
> wealthv <- cbind(respv, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colv <- colnames(wealthv)
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Rolling Weekly Yield Curve Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Yield Curve Strategy With Dimension Reduction In-Sample

The technique of *dimension reduction* is used to reduce the number of predictors in a model, for example in portfolio optimization.

Regularization of the inverse predictor matrix improves the in-sample performance of the yield curve strategy.

Although it's not realistic to achieve the in-sample performance, it's useful because it provides insights into how the model can be improved.




```
> # Calculate in-sample pnls for different dimax values
> dimv <- 2:7
> pnls <- lapply(dimv, function(dimax) {
+   invred <- HighFreq::calc_invsvd(predm, dimax=dimax)
+   coeff <- drop(invred %*% respv)
+   fcasts <- (predm %*% coeff)
+   fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
+   fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
+   pnls <- fcasts*respv
+   pnls/sd(pnls)
+ })
> pnls <- sd(respv)*do.call(cbind, pnls)
> colnames(pnls) <- paste0("dim=", dimv)
```

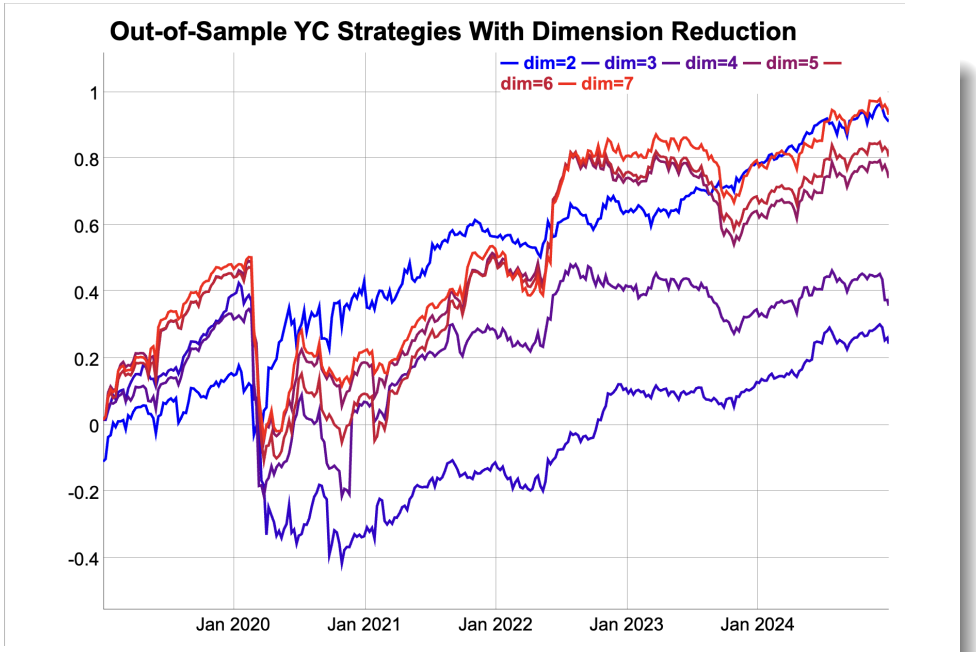
```
> # Plot dygraph of in-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> endw <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endw], main="In-Sample YC Strategie
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Yield Curve Strategy With Dimension Reduction Out-of-Sample

For out-of-sample forecasts, the training set and the test set are separate. The model is calibrated on the training set, and forecasts are calculated using the test set.

The best performing out-of-sample YC strategy is with the smallest order parameter (strongest dimension reduction) $dimax = 2$.

```
> # Calculate in-sample pnls for different dimax values
> pnls <- lapply(dimv, function(dimax) {
+   invred <- HighFreq::calc_invsvd(predm["/2019"], dimax=dimax)
+   coeff <- drop(invred %*% respv["/2019"])
+   fcasts <- (predm %*% coeff)
+   fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
+   fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
+   pnls <- fcasts*respv
+   pnls/sd(pnls)
+ })
> pnls <- sd(respv)*do.call(cbind, pnls)
> colnames(pnls) <- paste0("dim=", dimv)
```



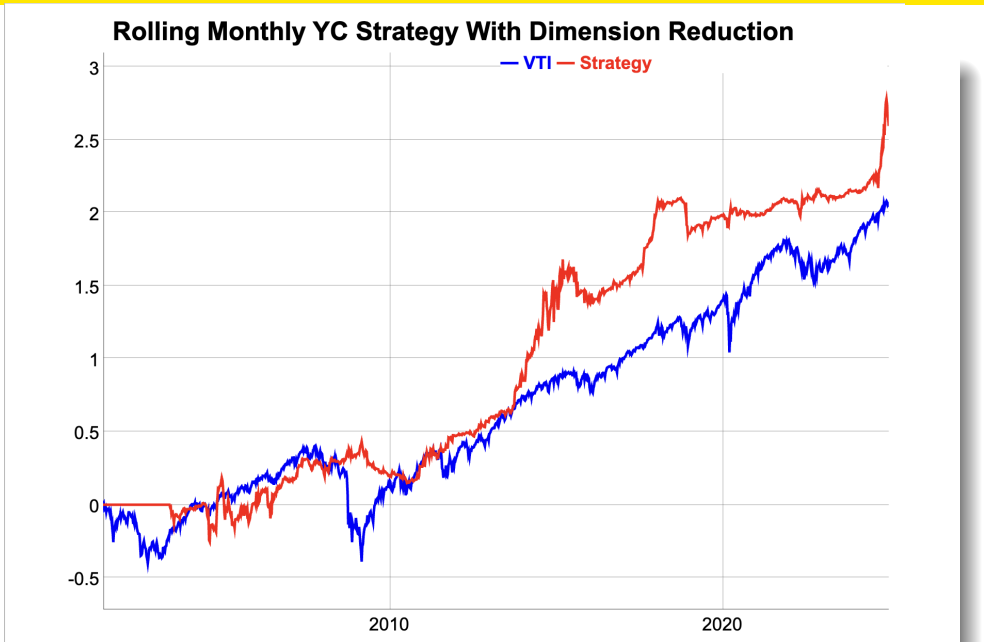

```
> # Plot dygraph of out-of-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> endw <- rutils::calc_endpoints(pnls["2019/"], interval="weeks")
> dygraphs::dygraph(cumsum(pnls["2019/"])[endw], main="Out-of-Sample
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Monthly Yield Curve Strategy With Dimension Reduction

The rolling monthly strategy with dimension reduction performs better than the standard strategy because dimension reduction allows using shorter `lookb` intervals since it suppresses the response noise.

In the rolling monthly yield curve strategy, the model is recalibrated at the end of every month using a training set of the past `lookb` days. The coefficients are applied to perform out-of-sample forecasts in the following month.

```
> # Define monthly dates
> endd <- rutils::calc_endpoints(respv, interval="month")
> enddd <- seq_along(endd)[endd > lookb]
> # Perform loop over monthly dates
> lookb <- 500
> dimax <- 2
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> fcasts <- mclapply(enddd, function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (max(1, endd[tday-1]-lookb):endd[tday-1])
+   outsample <- (endd[tday-1]+1):endd[tday]
+   # Calculate coefficients in-sample
+   invreg <- HighFreq::calc_invsvd(predm[insample, ], dimax=dimax)
+   coeff <- drop(invreg %*% respv[insample, ])
+   # Calculate forecasts out-of-sample
+   fcasts <- (predm[outsample, ] %*% coeff)
+ }, mc.cores=ncores)  # end mclapply
> fcasts <- do.call(rbind, fcasts)
> fcasts <- rbind(matrix(rep(0, nrows-NROW(fcasts)), nc=1), fcasts)
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> pnls <- fcasts*respv
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(respv)/sd(pnls)
```

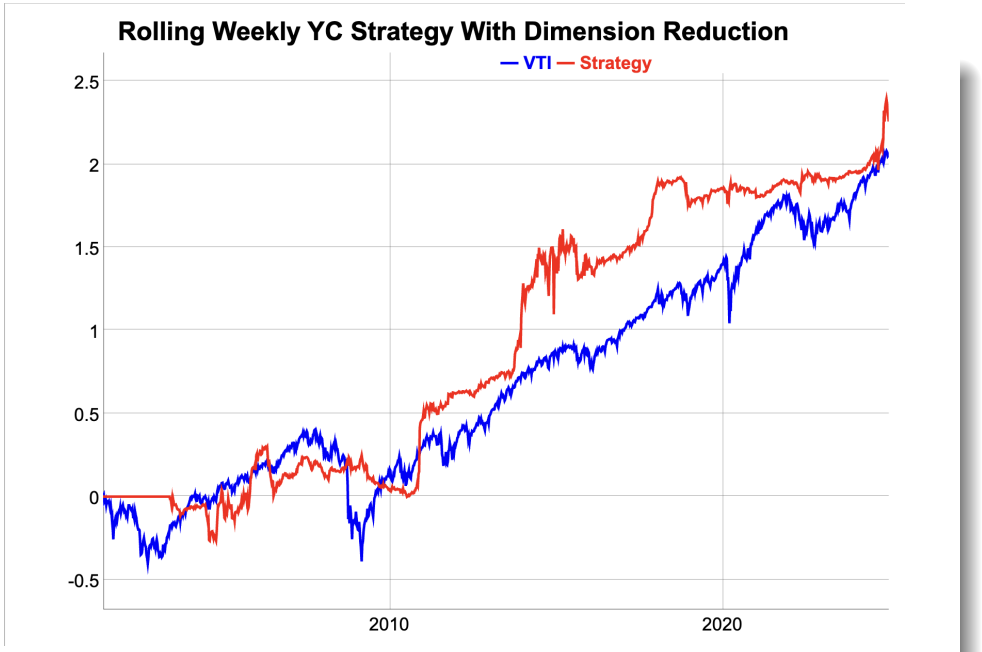



```
> # Plot dygraph of rolling monthly YC strategy
> wealthv <- cbind(respv, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> colv <- colnames(wealthv)
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Rolling Monthly YC Strategy With Dimension Reduction") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Weekly Yield Curve Strategy With Shrinkage

In the rolling weekly yield curve strategy, the model is recalibrated at the end of every week using a training set of the past `lookb` days. The coefficients are applied to perform out-of-sample forecasts in the following week.

```
> # Define weekly dates
> endd <- rutils::calc_endpoints(closep, interval="weeks")
> enddd <- seq_along(endd)[endd > lookb]
> fcasts <- mclapply(enddd, function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (max(1, endd[tday-1]-lookb):endd[tday-1])
+   outsample <- (endd[tday-1]+1):endd[tday]
+   # Calculate coefficients in-sample
+   invreg <- HighFreq::calc_invsvd(predm[insample, ], dimax=dimax)
+   coeff <- drop(invreg %*% respv[insample, ])
+   # Calculate forecasts out-of-sample
+   fcasts <- (predm[outsample, ] %*% coeff)
+ }, mc.cores=ncores)  # end mclapply
> fcasts <- do.call(rbind, fcasts)
> fcasts <- rbind(matrix(rep(0, nrows-NROW(fcasts)), nc=1), fcasts)
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> pnls <- fcasts*respv
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(respv)/sd(pnls)
```



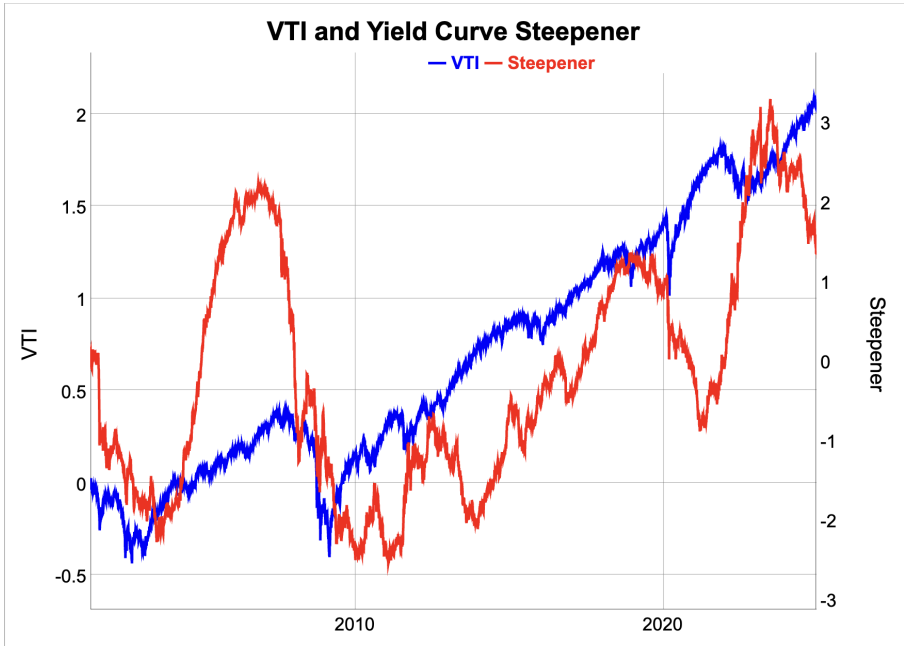

```
> # Plot dygraph of rolling weekly YC strategy
> wealthv <- cbind(respv, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> colv <- colnames(wealthv)
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Rolling Weekly YC Strategy With Dimension Reduction") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Yield Curve Principal Components and Stock Returns

The principal components of the interest rate yield curve can also be used as predictors of stock indices.

The second principal component describes the steepening and flattening of the yield curve, and it's an indicator of investor risk appetite. So it's also related to bullish and bearish market periods.

```
> # Calculate PCA of rates from correlation matrix
> eigend <- eigen(cor(retr))
> pcar <- (retr %*% eigend$vectors)
> colnames(pcar) <- paste0("PC", 1:6)
> pcar <- xts::xts(pcar, datev)
> # Define predictor as the YC PCAs
> predm <- rutils::lagit(pcar)
> regmod <- lm(retp ~ predm)
> summary(regmod)
> # After 2010, the PCAs are not good predictors
> regmod <- lm(retp["2010/"] ~ predm["2010/"])
> summary(regmod)
```




```
> # Plot YC steepener principal component with VTI
> datav <- cbind(respv, pcar[, 2])
> colnames(datav) <- c("VTI", "Steepener")
> colv <- colnames(datav)
> dygraphs::dygraph(cumsum(datav),
+   main="VTI and Yield Curve Steepener") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", strokeWidth=2, col="blue") %>%
+   dySeries(name=colv[2], axis="y2", strokeWidth=2, col="red") %>%
+   dyLegend(show="always", width=300)
```

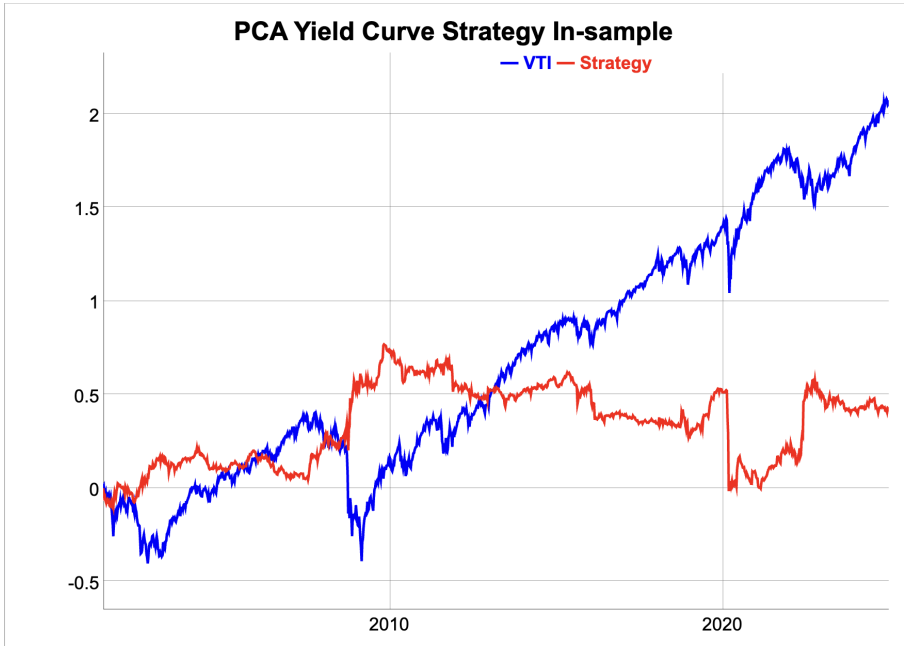
PCA Yield Curve Strategy In-Sample

For in-sample forecasts, the training set and the test set are the same. The model is calibrated on the data that is used for forecasting.

Yield Curve Strategy

Although it's not realistic to achieve the in-sample performance, it's useful because it provides insights into how the model works.

The in-sample strategy performs well in periods of high volatility, but otherwise it's flat.




```
> # Define predictor without intercept term
> predm <- rutils::lagit(pcar[, 1:3])
> # Calculate inverse of predictor
> invreg <- MASS::ginv(predm)
> # Calculate coefficients from response and inverse of predictor
> coeff <- drop(invreg %*% respv)
> # Calculate forecasts and PnLs in-sample
> fcasts <- (predm %*% coeff)
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.4)[, 2])
> fcasts <- ifelse(fcastv > mad(fcastv)/20, fcasts/fcastv, 0)
> pnls <- fcasts*respv
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(respv[respv<0])/sd(pnls[pnls<0])
> # Calculate in-sample factors
> factv <- (predm*coeff)
> apply(factv, 2, sd)
```

```
> # Plot dygraph of in-sample YC strategy
> wealthv <- cbind(respv, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> colv <- colnames(wealthv)
> endw <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="PCA Yield Curve Strategy In-sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

draft: Combined Predictor Matrix

A "kitchen sink" strategy combines many different predictors into a large predictor matrix with many columns.

For example by combining the yield curve predictors with the lagged returns.

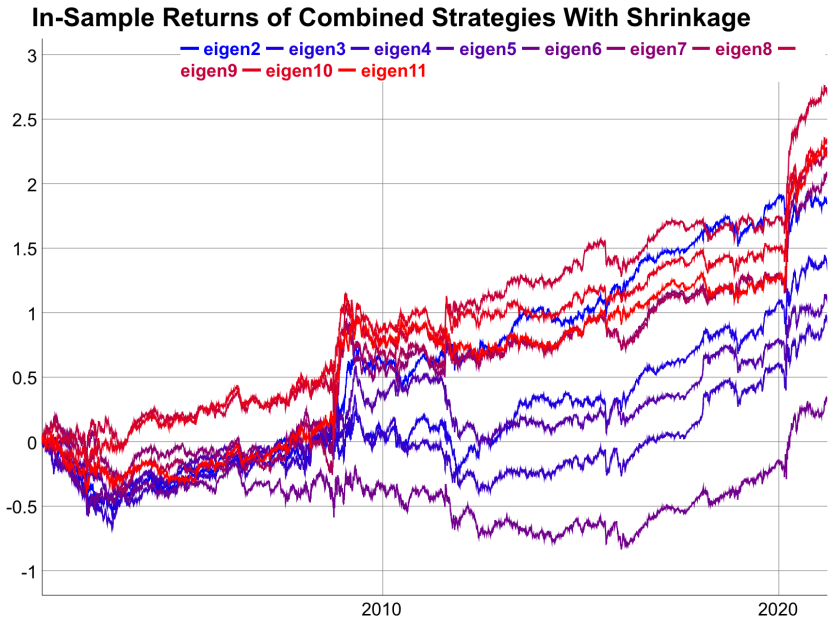
```
> # Load the yield curve data
> load(file="/Users/jerzy/Develop/lecture_slides/data/rates_data.RDa
> ratev <- do.call(cbind, as.list(ratesenv))
> namev <- colnames(ratev)
> namev <- substr(namev, start=4, stop=10)
> namev <- as.numeric(namev)
> indeks <- order(namev)
> ratev <- ratev[, indeks]
> closep <- log(quantmod::Cl(rutils::etfenv$VTI))
> colnames(closep) <- "VTI"
> nrows <- NROW(closep)
> datev <- zoo::index(closep)
> ratev <- na.omit(ratev[datev])
> closep <- closep[zoo::index(ratev)]
> datev <- zoo::index(closep)
> retp <- rutils::diffit(closep)
> retr <- rutils::diffit(ratev)
> # Create a combined predictor matrix
> dimax <- 5
> predm <- sapply(1:dimax, rutils::lagit, input=as.numeric(retp))
> colnames(predm) <- paste0("retslag", 1:NCOL(predm))
> predm <- cbind(predm, rutils::lagit(retr))
> predm <- cbind(rep(1, NROW(predm)), predm)
> colnames(predm)[1] <- "intercept"
> respv <- retp
```

draft: Combined Strategy With Shrinkage In-Sample

The technique of *regularization* is designed to reduce the number of parameters in a model, for example in portfolio optimization.

Regularization of the inverse predictor matrix improves the in-sample performance of the yield curve strategy.

Although it's not realistic to achieve the in-sample performance, it's useful because it provides insights into how the model can be improved.




```
> # Calculate in-sample pnls for different dimax values
> dimv <- 2:11
> pnls <- lapply(dimv, function(dimax) {
+   invred <- HighFreq::calc_invsvd(predm, dimax=dimax)
+   coeff <- drop(invred %*% respv)
+   fcasts <- (predm %*% coeff)
+   sign(fcasts)*respv
+ })
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("dim=", dimv)
```

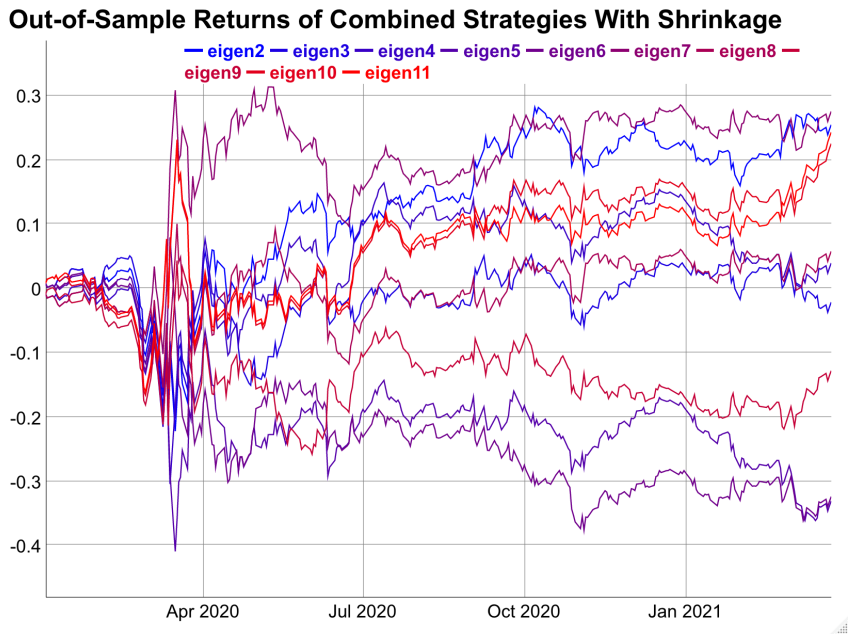
```
> # Plot dygraph of in-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="In-Sample Returns of Combine
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

draft: Combined Strategy With Shrinkage Out-of-Sample

For out-of-sample forecasts, the training set and the test set are separate. The model is calibrated on the training set, and forecasts are calculated using the test set.

The out-of-sample strategy performs well in periods of high volatility, but otherwise it's flat.

```
> # Define in-sample and out-of-sample intervals
> insample <- (datev < as.Date("2020-01-01"))
> outsample <- (datev >= as.Date("2020-01-01"))
> # Calculate in-sample pnls for different dimax values
> dimv <- 2:11
> pnls <- lapply(dimv, function(dimax) {
+   invred <- HighFreq::calc_invsvd(predm[insample, ], dimax=dimax)
+   coeff <- drop(invred %*% respv[insample, ])
+   fcasts <- (predm[outsample, ] %*% coeff)
+   sign(fcasts)*respv[outsample, ]
+ })
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("dim=", dimv)
```



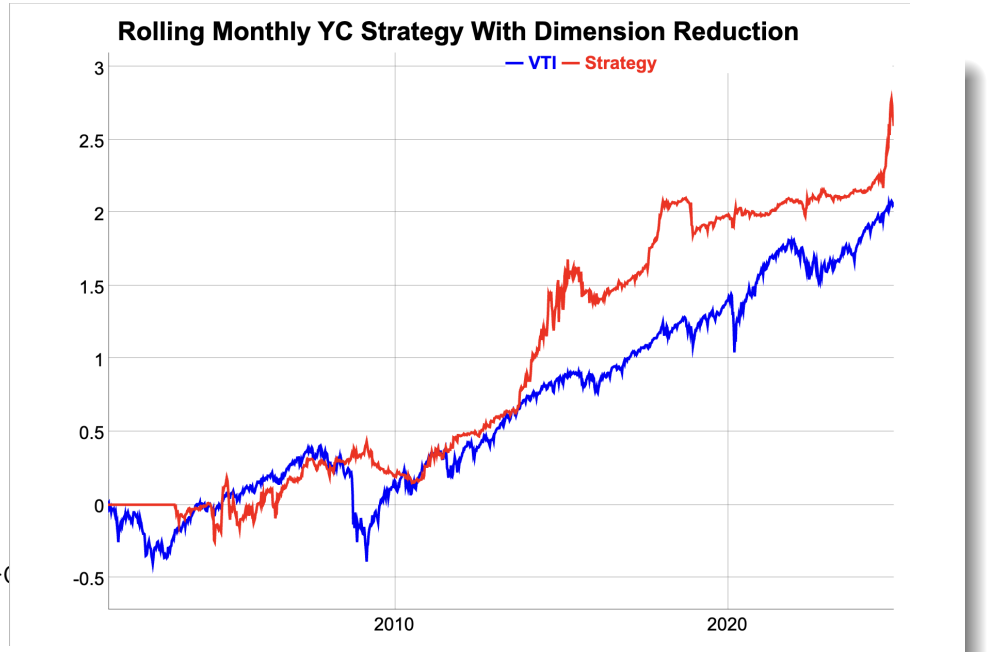

```
> # Plot dygraph of out-of-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="Out-of-Sample Returns of Co
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

draft: Rolling Monthly Combined Strategy With Dimension Reduction

The shrinkage rolling monthly strategy performs better than the standard strategy because regularization allows using shorter `lookb` intervals since it suppresses the response noise.

In the rolling monthly yield curve strategy, the model is recalibrated at the end of every month using a training set of the past 6 months. The coefficients are applied to perform out-of-sample forecasts in the following month.

```
> # Define monthly dates
> format(datev[1], "%m-%Y")
> format(datev[NROW(datev)], "%m-%Y")
> endd <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-(
> # Perform loop over monthly dates
> lookb <- 6
> dimax <- 3
> pnls <- lapply((lookb+1):(NROW(endd)-1), function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > endd[tday-lookb]) & (datev < endd[tday])
+   outsample <- (datev > endd[tday]) & (datev < endd[tday+1])
+   # Calculate forecasts and PnLs out-of-sample
+   invred <- HighFreq::calc_invsd(predm[insample, ], dimax=dimax)
+   coeff <- drop(invred %*% respv[insample, ])
+   fcasts <- (predm[outsample, ] %*% coeff)
+   sign(fcasts)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```

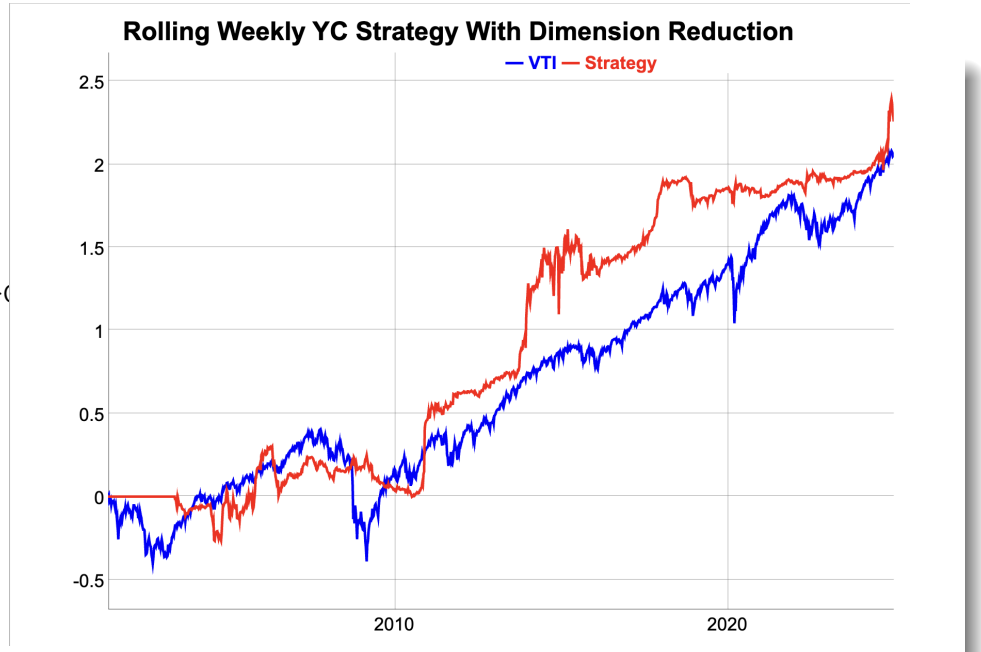



```
> # Plot dygraph of rolling monthly YC strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colv <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Rolling Monthly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

draft: Rolling Weekly Combined Strategy With Shrinkage

In the rolling weekly yield curve strategy, the model is recalibrated at the end of every week using a training set of the past 4 weeks. The coefficients are applied to perform out-of-sample forecasts in the following week.

```
> # Define weekly dates
> endd <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-(
> # Perform loop over weekly dates
> lookb <- 8
> dimax <- 4
> pnls <- lapply((lookb+1):(NROW(endd)-1), function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > endd[tday-lookb]) & (datev < endd[tday])
+   outsample <- (datev > endd[tday]) & (datev < endd[tday+1])
+   # Calculate forecasts and PnLs out-of-sample
+   invred <- HighFreq::calc_invsd(predm[insample, ], dimax=dimax)
+   coeff <- drop(invred %*% respv[insample, ])
+   fcasts <- (predm[outsample, ] %*% coeff)
+   sign(fcasts)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```




```
> # Plot dygraph of rolling weekly YC strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colv <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Rolling Weekly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

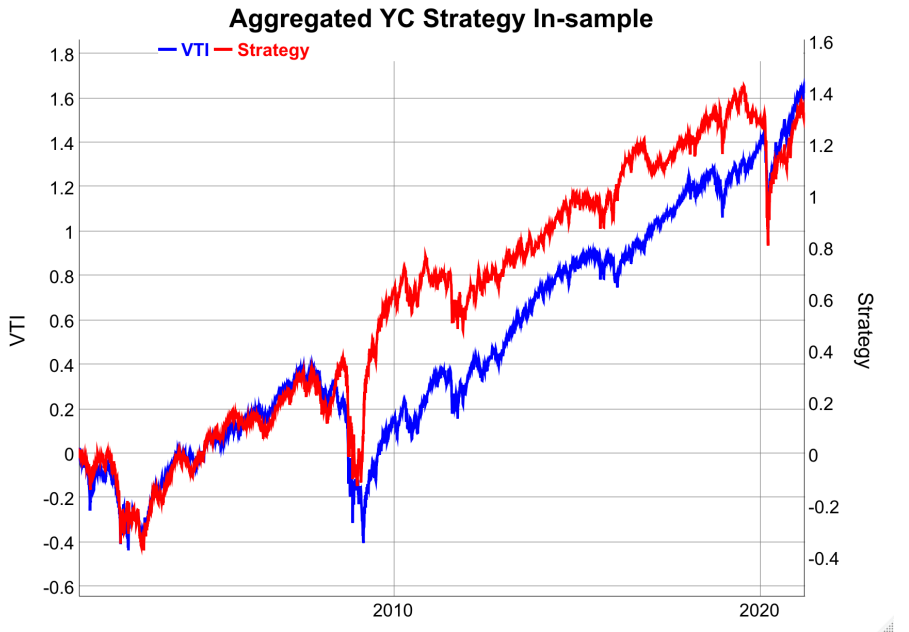
draft: Forecasts Using Aggregated Predictor

Needs more work to improve performance

Aggregating the predictor reduces its noise and increases the significance of correlations.

The optimal aggregation number can be found by maximizing the regression t-values.

```
> # Find optimal nagg for predictor
> naggs <- 5:100
> tvalues <- sapply(naggs, function(nagg) {
+   predm <- HighFreq::roll_mean(retr, lookb=nagg)
+   predm <- cbind(rep(1, NROW(predm)), predm)
+   predm <- rutils::lagit(predm)
+   regmod <- lm(respv ~ predm - 1)
+   regsum <- summary(regmod)
+   max(abs(regsum$coefficients[, 3][-1]))
+ })  # end sapply
> naggs[which.max(tvalues)]
> plot(naggs, tvalues, t="l", col="blue", lwd=2)
> # Calculate aggregated predictor
> nagg <- 53
> predm <- HighFreq::roll_mean(retr, lookb=nagg)
> predm <- rutils::lagit(predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> regmod <- lm(respv ~ predm - 1)
> summary(regmod)
```




```
> # Calculate forecasts and PnLs in-sample
> invreg <- MASS::ginv(predm)
> coeff <- drop(invreg %*% respv)
> fcasts <- (predm %*% coeff)
> pnls <- sign(fcasts)*respv
> # Plot dygraph of in-sample YC strategy
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colv <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Aggregated YC Strategy In-sample") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

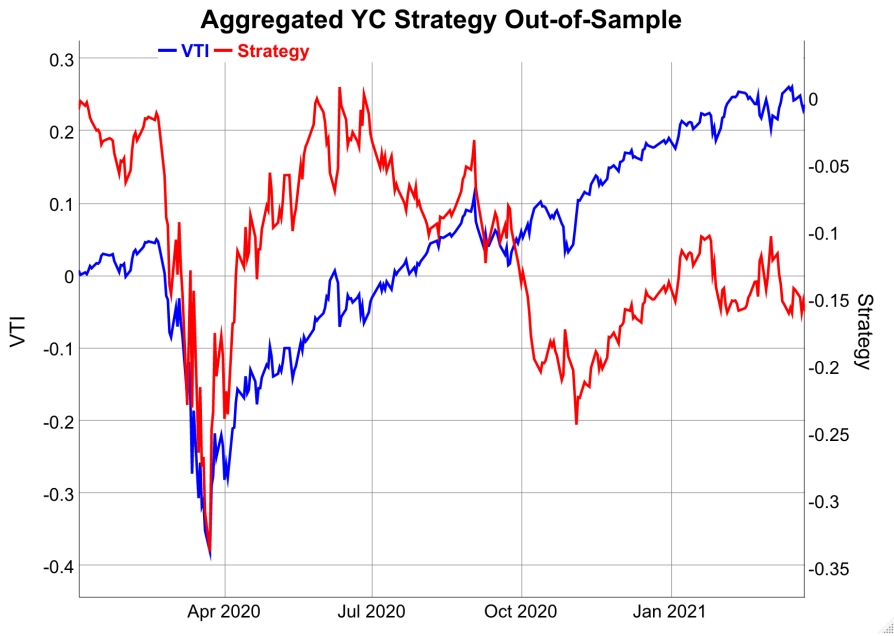
draft: Aggregated Forecasts Out-of-Sample

Needs more work to improve performance

For out-of-sample forecasts, the training set and the test set are separate. The model is calibrated on the training set, and forecasts are calculated using the test set.

The out-of-sample strategy performs well in periods of high volatility, but otherwise it's flat.

```
> # Define in-sample and out-of-sample intervals
> insample <- (datev < as.Date("2020-01-01"))
> outsample <- (datev >= as.Date("2020-01-01"))
> # Calculate forecasts and PnLs out-of-sample
> invreg <- MASS::ginv(predm[insample, ])
> coeff <- drop(invreg %*% respv[insample, ])
> fcasts <- (predm[outsample, ] %*% coeff)
> pnls <- sign(fcasts)*respv[outsample, ]
```




```
> # Plot dygraph of out-of-sample YC strategy
> wealthv <- cbind(retp[outsample, ], pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colv <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endw],
+   main="Aggregated YC Strategy Out-of-Sample") %>%
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```