Autoregressive Strategies

FRE7241, Spring 2023

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

May 12, 2023

NYU | TANDON SCHOOL OF ENGINEERING

Daily Mean Reverting Strategy

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

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

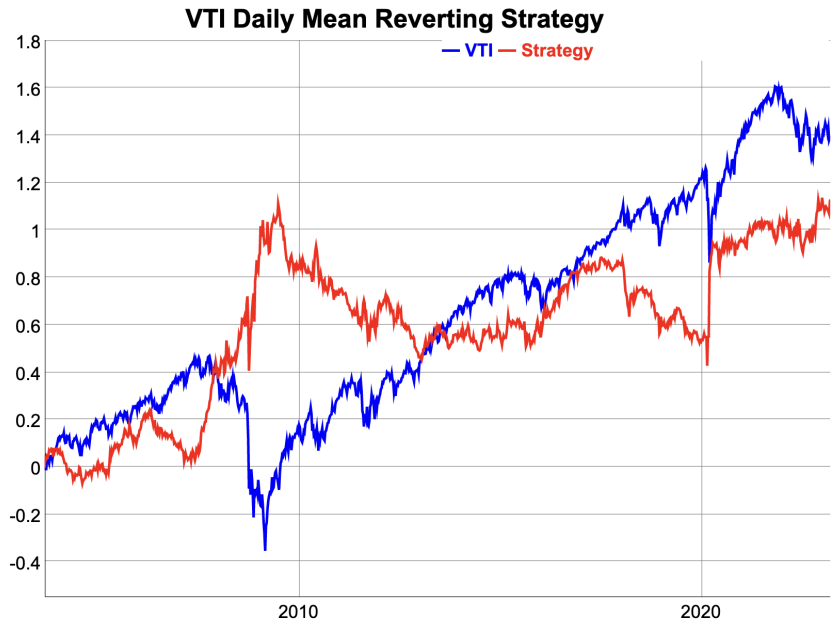
The function `rutils::plot_acf()` calculates and plots the autocorrelations of a time series.

Daily stock returns often exhibit some negative autocorrelations.

The daily mean reverting 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.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Calculate the autocorrelations of VTI daily returns
> rutils::plot_acf(retp)
> # Simulate mean reverting strategy
> posv <- rutils::lagit(sign(retp), lagg=1)
> pnls <- (-retp*posv)
```



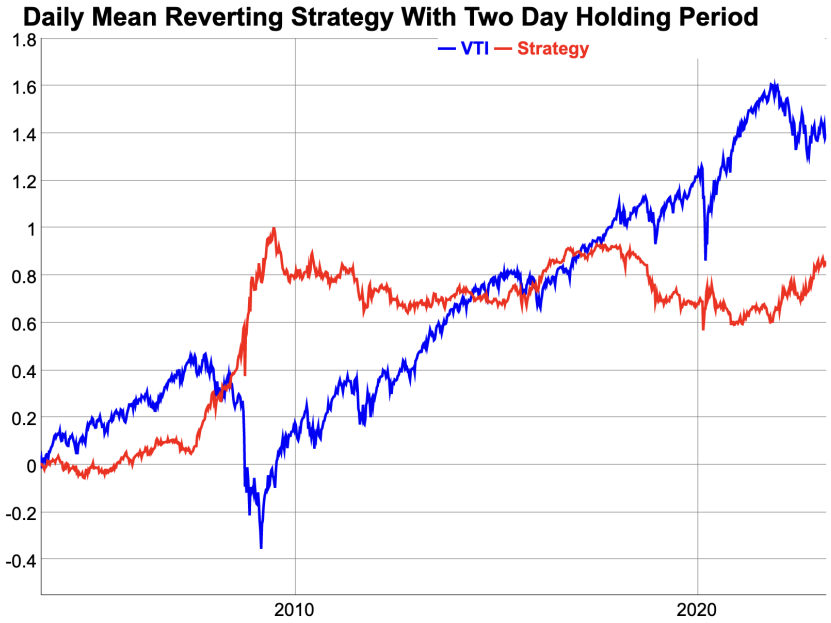

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv,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

Daily Mean Reverting Strategy With a Holding Period

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

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



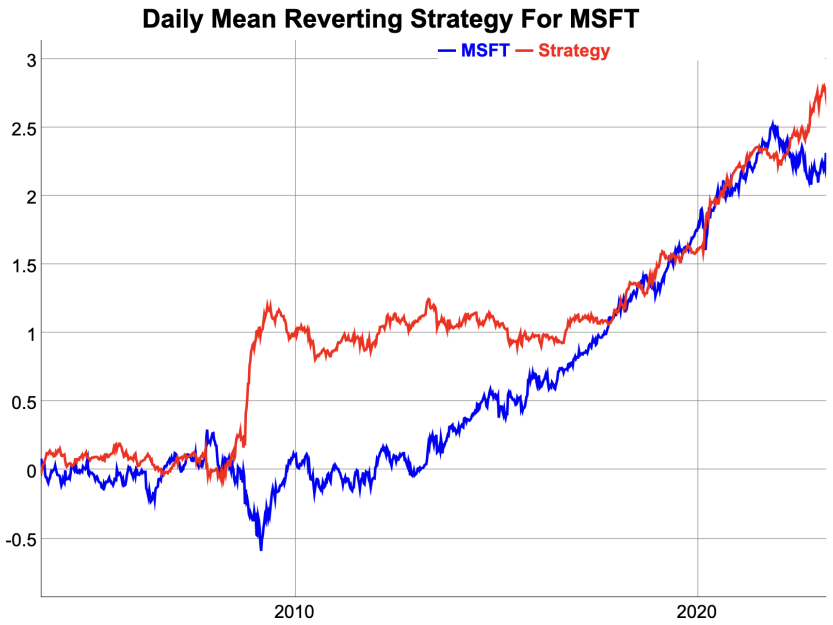

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv,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

Daily Mean Reverting Strategy For Stocks

Some daily stock returns exhibit stronger negative autocorrelations than ETFs.

But the daily mean reverting 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(returns$MSFT)
> rutils::plot_acf(retp)
> # Simulate mean reverting strategy with two day holding period
> posv <- rutils::lagit(rutils::roll_sum(sign(retp), look_back=2))/2
> pnls <- (-retp*posv)
```



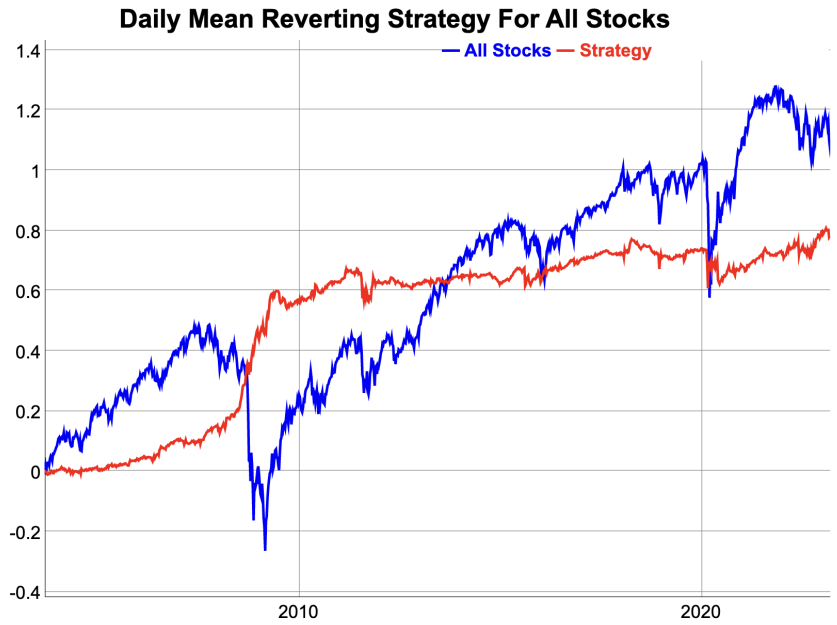

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv,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

Daily Mean Reverting Strategy For All Stocks

The combined daily mean reverting 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 mean reverting strategy for all S&P500 stocks
> pnls <- lapply(returns, function(retp) {
+   retp <- na.omit(retp)
+   posv <- rutils::roll_sum(sign(retp), look_back=2)/2
+   posv <- rutils::lagit(posv)
+   pnls <- (-retp*posv)
+   pnls
+ }) # end lapply
> pnls <- do.call(cbind, pnls)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> # Calculate the average returns of all S&P500 stocks
> datev <- zoo::index(returns)
> indeks <- rowMeans(returns, na.rm=TRUE)
```




```
> # 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 mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Mean Reverting Strategy For All Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

The EWMA Mean-Reversion Strategy

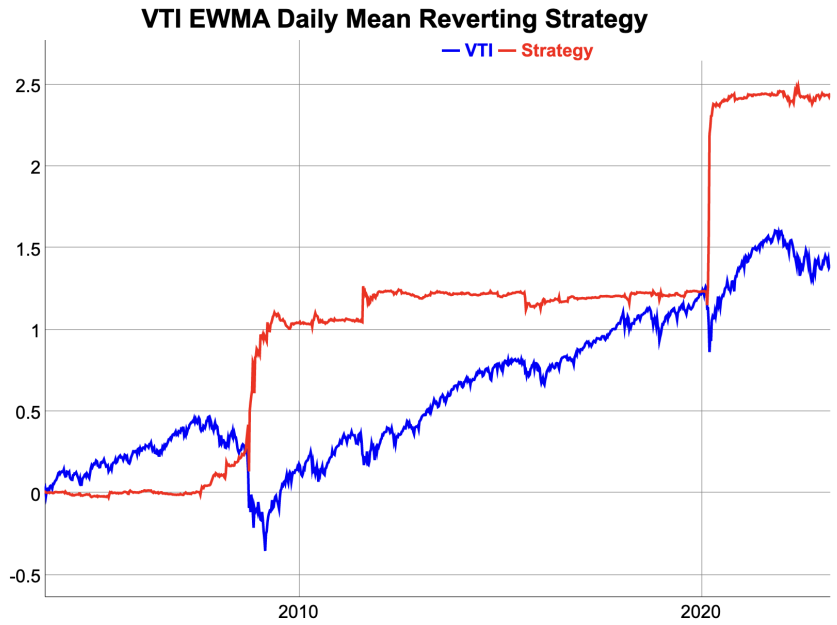
The *EWMA* mean-reversion strategy holds both long stock positions and short positions proportional to the trailing *EWMA* of past returns.

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 problem is that the strategy makes profits mostly in periods of high volatility, but otherwise it's not profitable.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Calculate the EWMA returns recursively using C++ code
> retma <- HighFreq::run_mean(retp, lambda=0.1)
> # Calculate the positions and pnls
> posv <- retma
> posv <- rutils::lagit(posv, lagg=1)
> pnls <- (-retp*posv)
> pnls <- pnls*sd(retp)/sd(pnls)
```



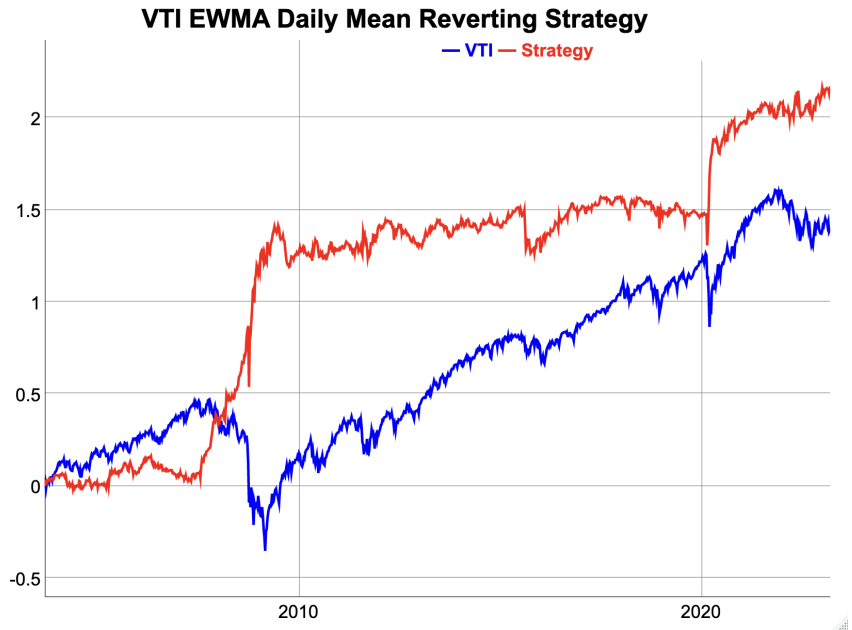

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv,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

The EWMA 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.

```
> # Calculate the trailing volatility
> volat <- HighFreq::run_var(retp, lambda=0.5)
> volat <- sqrt(volat)
> # Scale the returns by their trailing volatility
> retsc <- ifelse(volat > 0, retp/volat, 0)
> # Calculate the EWMA returns
> retma <- HighFreq::run_mean(retsc, lambda=0.1)
> # Calculate the positions and pnls
> posv <- retma
> posv <- rutils::lagit(posv, lagg=1)
> pnls <- (-retp*posv)
> pnls <- pnls*sd(retp)/sd(pnls)
```




```
> # 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 mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI EWMA Daily Mean Reverting 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)
> 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* **r** multiplied by the inverse of the *predictor matrix* $\mathbb{P}$:

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

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

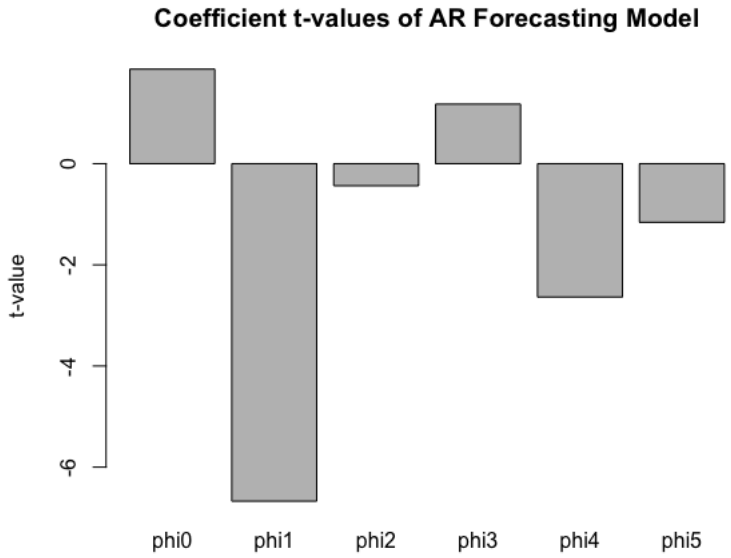
The *in-sample* forecasts of the *AR(n)* autoregressive model are calculated by multiplying the predictor matrix by the fitted AR coefficients:

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

The variance of the autoregressive coefficients φ is given by:

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

Where σ^2_{ε} is the variance of the residuals.




```
> # 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 forecast
> round(cor(resids, fcasts), 6)
> round(sapply(predm[, -1], function(x) cor(resids, x)), 6)
```

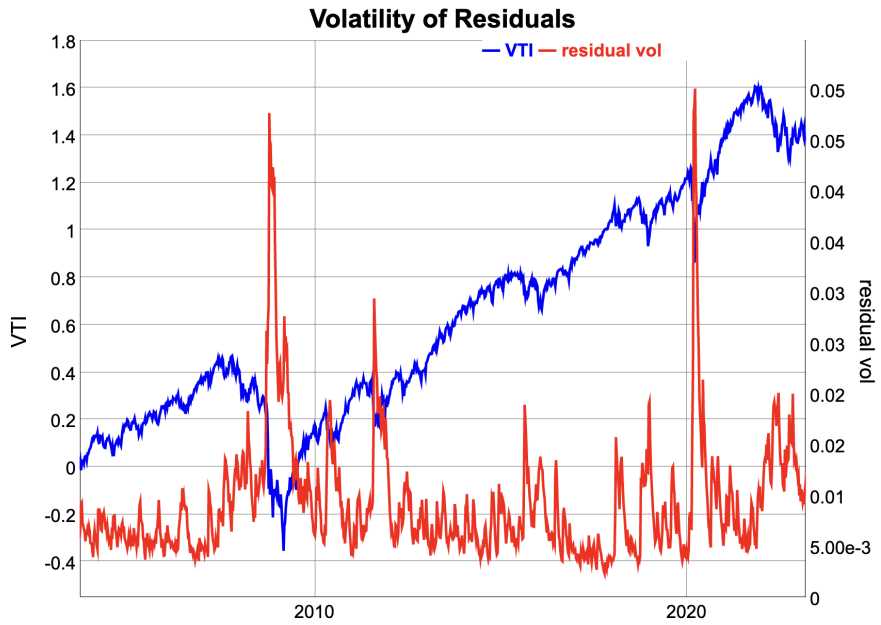
```
> # Calculate the variance of the residuals
> vares <- sum(resids^2)/(nrows-NROW(coeff))
> # Calculate the predictor matrix squared
> predm2 <- crossprod(predm)
> # Calculate the covariance matrix of the AR coefficients
> covar <- vares*MASS::ginv(predm2)
> coeffsd <- sqrt(diag(covar))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coeffsd)
> # Plot the t-values of the AR coefficients
> lagv <- paste0("phi", 0:(NROW(coefft)-1))
> barplot(coefft ~ lagv, xlab="", ylab="t-value",
+   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.

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




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

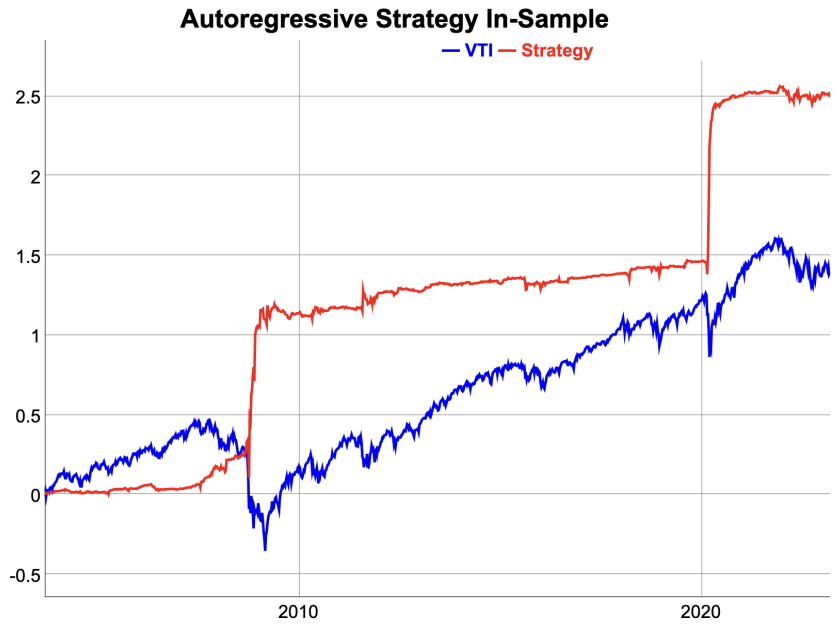
Autoregressive Strategy In-Sample

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 very high risk.




```
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp)/sd(pnls)
```

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

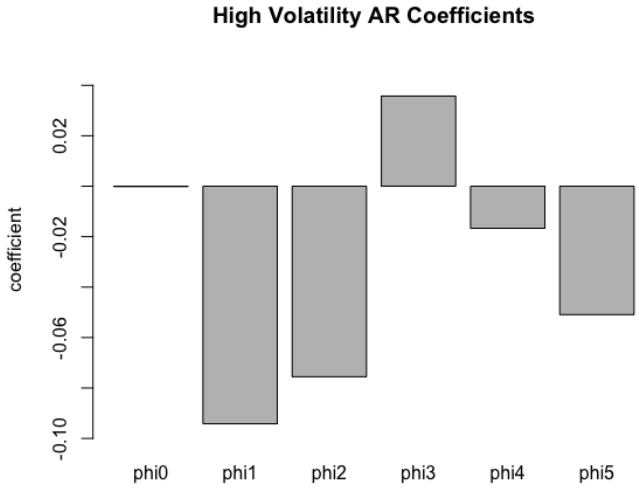
Autoregressive Coefficients in Periods of High and Low Volatility

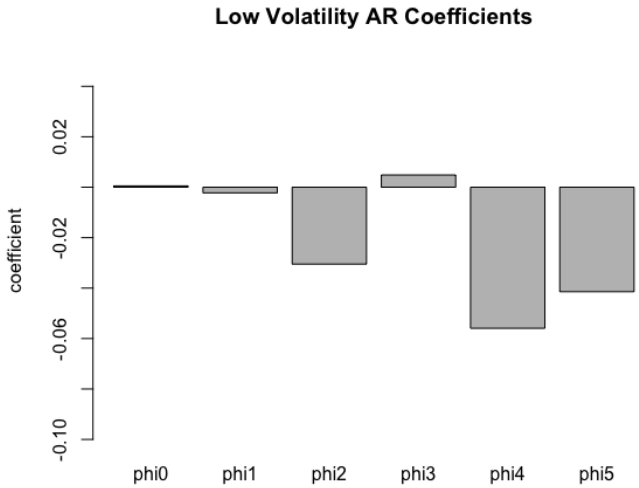
The autoregressive model assumes stationary returns and residuals, with similar volatility over time. In reality stock volatility is highly time dependent.

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)
> lagv <- paste0("phi", 0:(NROW(coeffh)-1))
> barplot(coeffh ~ lagv, main="High Volatility AR Coefficients",
+   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)
> barplot(coeffl ~ lagv, main="Low Volatility AR Coefficients",
+   xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
```

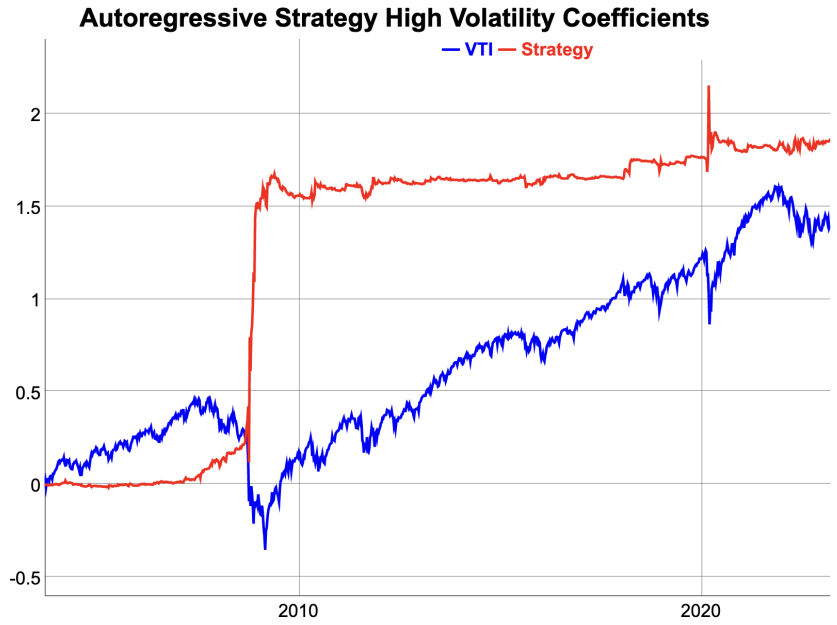


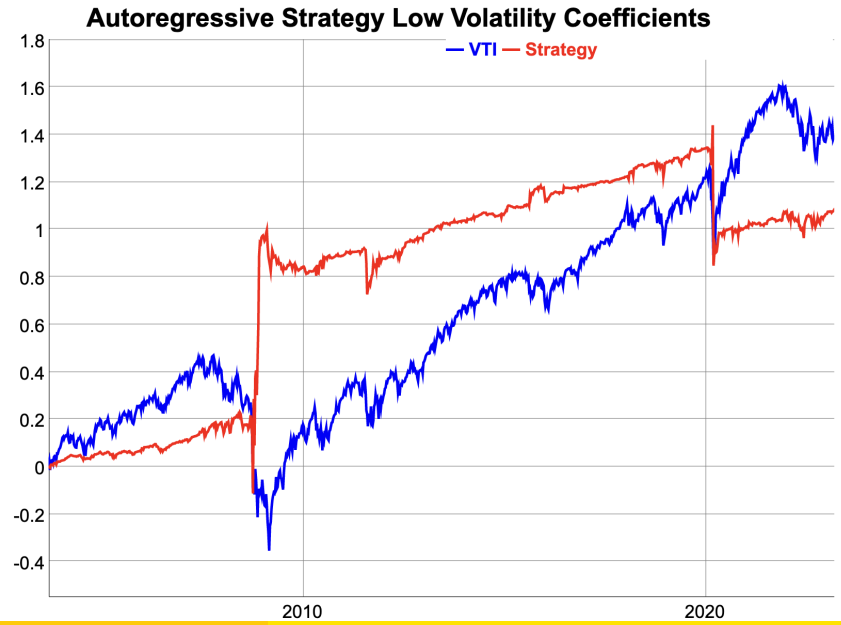


Performance of High and Low Volatility Autoregressive Coefficients

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
> pnls <- retp*fcasts
> pnls <- pnls*sd(retp)/sd(pnls)
> # 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 autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   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
> pnls <- retp*fcasts
> pnls <- pnls*sd(retp)/sd(pnls)
> # 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 autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   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 high and low volatility regimes, with high and low volatility coefficients.

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

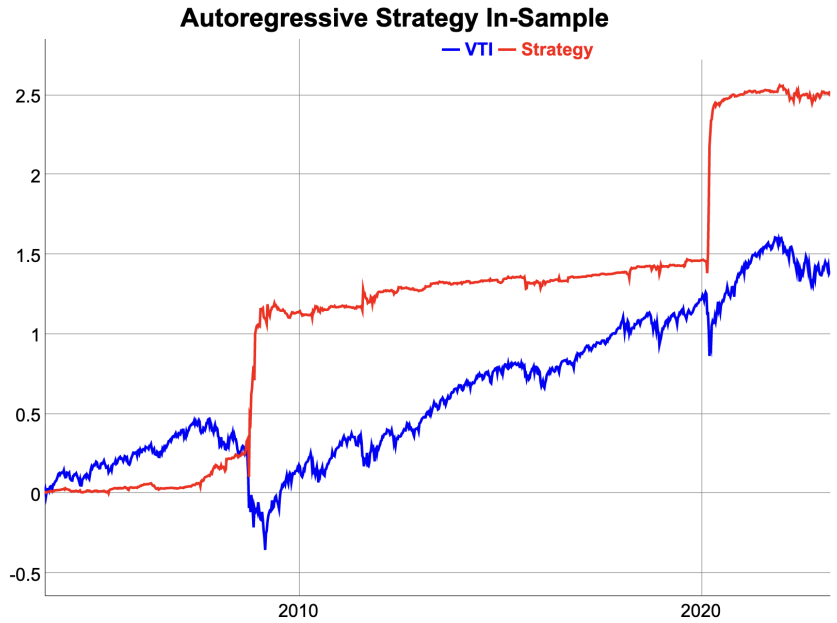
Doesn't work because the square forecast errors are similar.

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 very high 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 autoregressive strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   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:

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

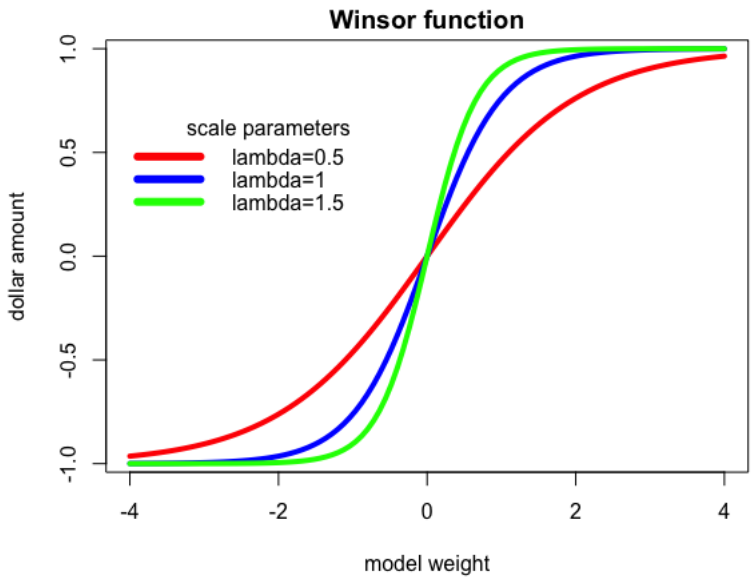
Where λ is the scale parameter.

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.



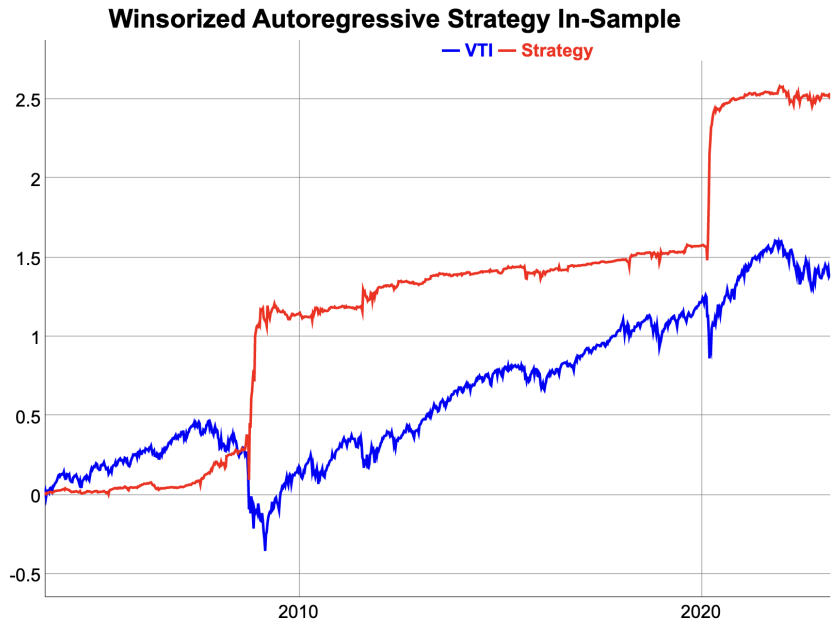

```
> lambdav <- c(0.5, 1, 1.5)
> colorv <- c("red", "blue", "green")
> # Define the winsor function
> winsorfun <- function(retp, lambda) tanh(lambda*retp)
> # Plot three curves in loop
> for (indeks in 1:3) {
+   curve(expr=winsorfun(x, lambda=lambdav[indeks]),
+ xlim=c(-4, 4), type="l", lwd=4,
+ xlab="model 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("lambda", 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 in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> # Winsorize the forecasts
> # fcasts <- winsorfun(fcasts/mad(fcasts), lambda=1.5)
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp)/sd(pnls)
```



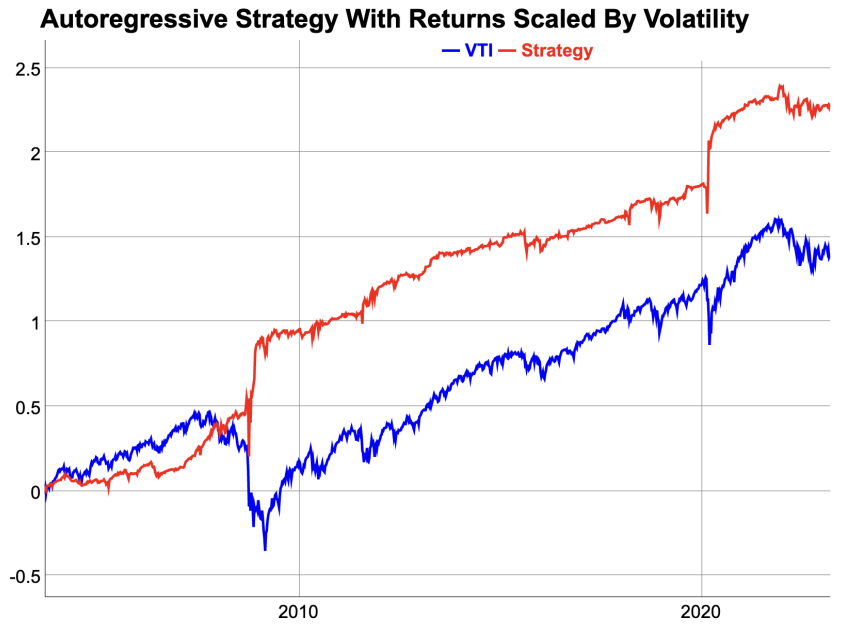

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of autoregressive strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   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.

```
> # Scale the returns by their trailing volatility
> varv <- HighFreq::run_var(retp, lambda=0.99)
> retsc <- ifelse(varv > 0, retp/sqrt(varv), 0)
> # 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 in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp)/sd(pnls)
```



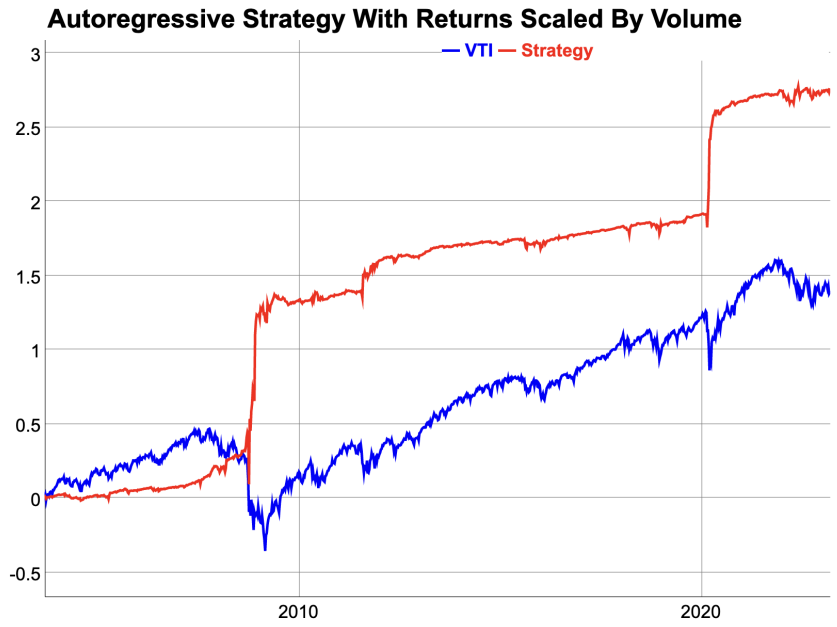

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   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.

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns in *trading time*, to account for time-dependent volatility.

```
> # Calculate VTI returns and trading volumes
> ohlc <- rutils::etfenv$VTI
> datev <- zoo::index(ohlc)
> nrows <- NROW(ohlc)
> closep <- quantmod::Cl(ohlc)
> retp <- rutils::diffit(log(closep))
> volumv <- quantmod::Vo(ohlc)
> # Calculate trailing average volume
> volumr <- HighFreq::run_mean(volumv, lambda=0.7)
> # Scale the returns using volume clock to trading time
> retsc <- ifelse(volumv > 0, volumr*retp/volumv, 0)
> # Calculate the AR coefficients
> respv <- retsc
> 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 <- predinv %*% respv
> # Calculate the in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcasts
> pnls <- pnls*sd(retp)/sd(pnls)
```




```
> # 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 autoregressive strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy With Returns Scaled By Volume") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

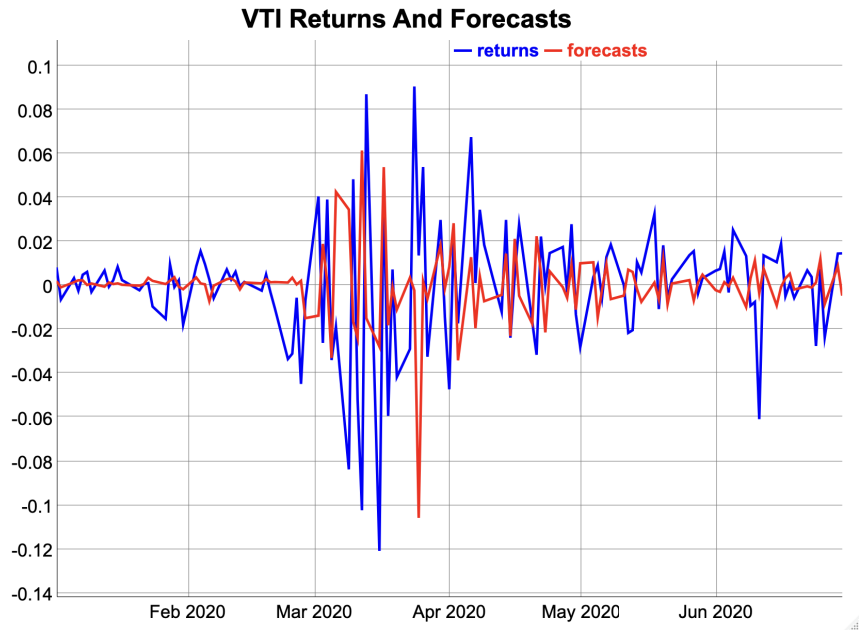
Mean Squared Error of the Autoregressive Forecasting Model

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: $\varepsilon_i = f_t - r_t$:

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

```
> # Calculate the correlation between forecasts and returns
> cor(fcasts, retp)
> # Calculate the forecasting errors
> errorf <- (fcasts - retp)
> # Mean squared error
> mean(errorf^2)
```




```
> # Plot the forecasts
> datav <- cbind(retp, 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)
```

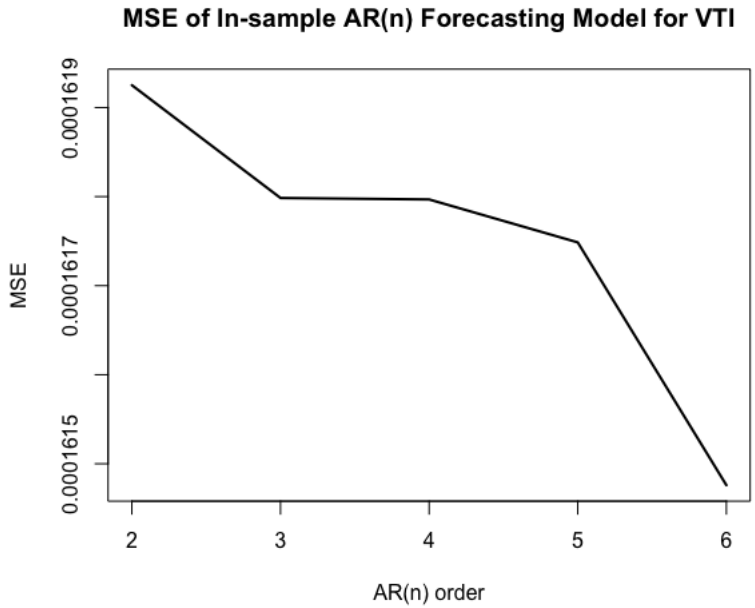
In-sample Order Selection of Autoregressive Forecasting

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 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
+   # 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")
```

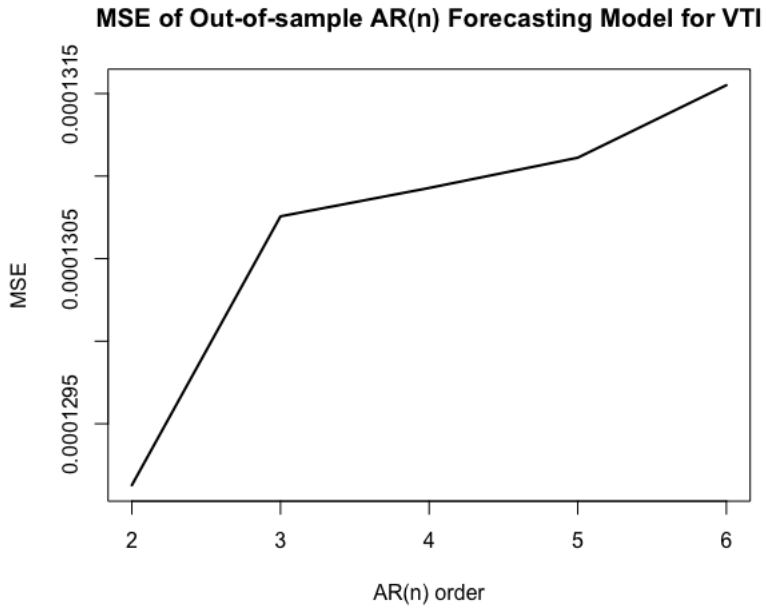
Out-of-Sample Forecasting Using Autoregressive Models

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 function of the AR order
> fcasts <- lapply(2:NCOL(predm), function(ordern) {
+   # Calculate the fitted AR coefficients
+   predinv <- MASS::ginv(predm[insample, 1:ordern])
+   coeff <- predinv %*% respv[insample]
+   # Calculate the out-of-sample forecasts of VTI
+   drop(predm[outsample, 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[outsample] - x)^2), cor=cor(respv[outsample],
+ })  # 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 Out-of-sample AR(n) Forecasting Model for VTI")
```

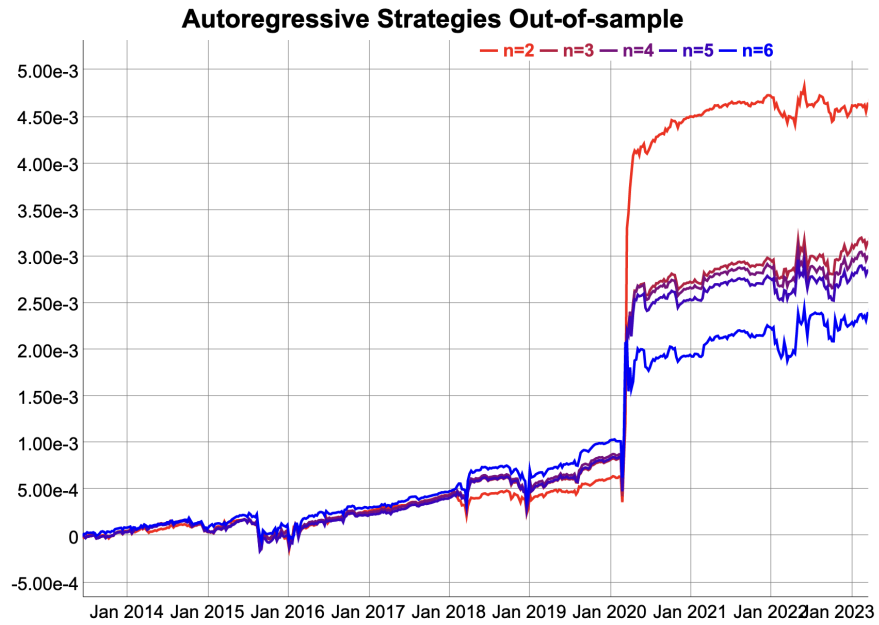
Out-of-Sample Autoregressive Strategy

The autoregressive strategy invests a single dollar amount of *VTI* equal to the sign of the forecasts.

The performance of the autoregressive strategy is better with a smaller order parameter *n* of the *AR(n)* model.

The optimal order parameter is equal to 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 optimal AR coefficients
> predinv <- MASS::ginv(predm[insample, 1:2])
> coeff <- drop(predinv %*% respv[insample])
> # Calculate the out-of-sample PnLs
> pnls <- lapply(fcasts, function(fcasts) {
+   cumsum(fcasts*retp[outsample])
+ })  # end lapply
> pnls <- rutils::do_call(cbind, pnls)
> colnames(pnls) <- names(fcasts)
```

```
> # Plot dygraph of out-of-sample PnLs
> colorv <- colorRampPalette(c("red", "blue"))(NCOL(pnls))
> colnamev <- colnames(pnls)
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(pnls[endd],
+   main="Autoregressive Strategies Out-of-sample") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(width=300)
```

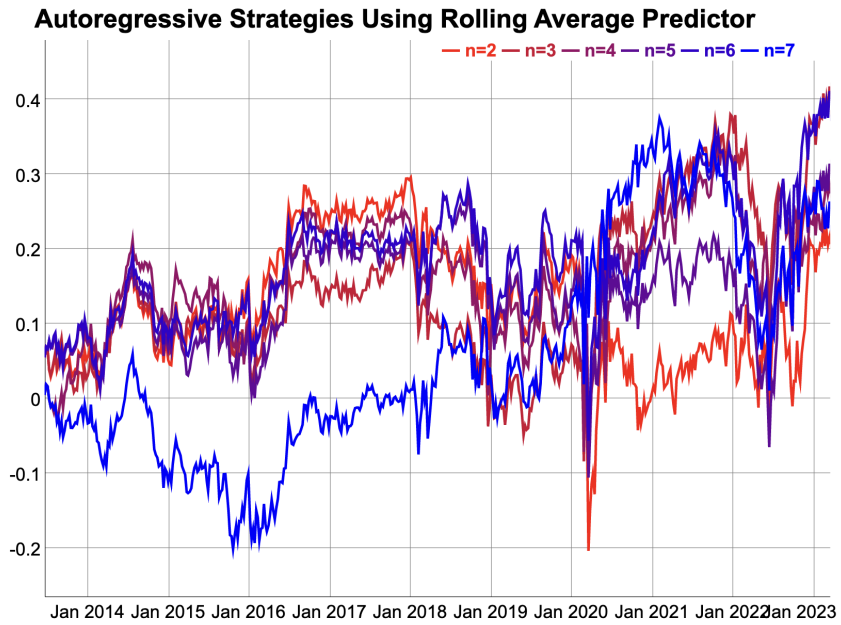
depr: Autoregressive Strategy Using Average Past Returns

The *out-of-sample* forecasts can be improved by using the rolling average of the returns as a predictor.

This is because the average of returns has a lower *variance*.

But the average also has a higher *bias* because it includes past returns that may be unrelated to the present.

Using the rolling average of returns as a predictor reduces the forecast variance at the expense of increasing its bias (known as the *bias-variance tradeoff*).




```
> # Define predictor as a rolling mean
> nagg <- 5
> predm <- HighFreq::roll_mean(matrix(retp), nagg)
> # Define predictor matrix for forecasting
> predm <- sapply(1+nagg*(0:ordmax), rutils::lagit, input=predm)
> predm <- cbind(rep(1, nrows), predm)
> # Calculate the forecasts as function of the AR order
> fcasts <- lapply(2:NCOL(predm), function(ordern) {
+   predinv <- MASS::ginv(predm[insample, 1:ordern])
+   coeff <- drop(predinv %*% respv[insample])
+   drop(predm[outsample, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcasts) <- paste0("n=", 2:NCOL(predm))
```

```
> # Calculate the out-of-sample PnLs
> pnls <- sapply(fcasts, function(x) {
+   cumsum(sign(x)*retp[outsample])
+ })  # end sapply
> colnames(pnls) <- names(fcasts)
> pnls <- xts::xts(pnls, datev[outsample])
> # Plot dygraph of out-of-sample PnLs
> colorv <- colorRampPalette(c("red", "blue"))(NCOL(pnls))
> dygraphs::dygraph(pnls[endd],
+   main="Autoregressive Strategies Using Rolling Average Predictor"
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(width=300)
```

depr: Autoregressive Strategy Using Average of Past Forecasts

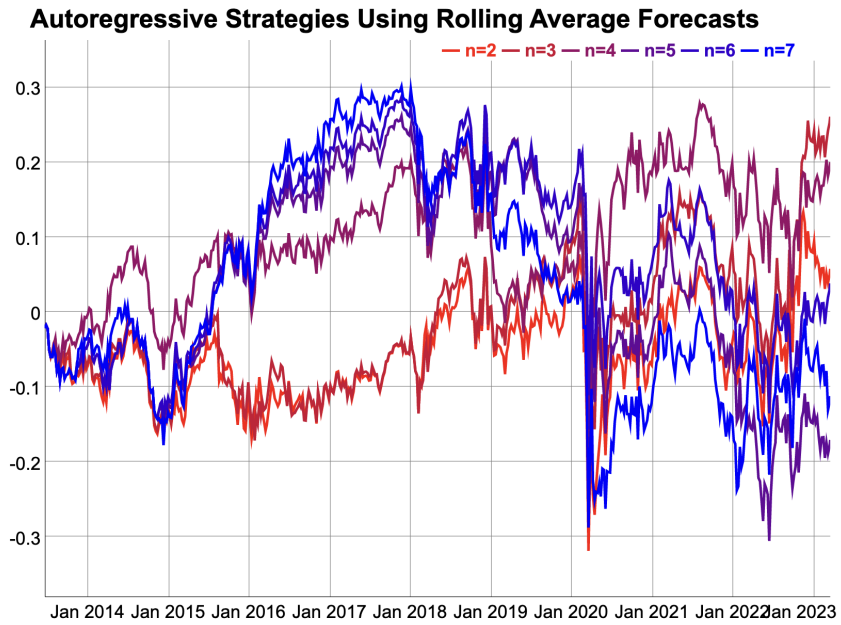
The *out-of-sample* forecasts can be further improved by using the average of past forecasts.

This is because the average of forecasts has a lower *variance*.

But the average also has a higher *bias* because it includes past forecasts that may be unrelated to the present.

Using the rolling average of past forecasts reduces the forecast variance at the expense of increasing its bias (known as the *bias-variance tradeoff*).

```
> # Calculate the PnLs using the average of past forecasts
> nagg <- 5
> pnls <- sapply(fcasts, function(x) {
+   x <- HighFreq::roll_mean(matrix(x), nagg)
+   cumsum(sign(x)*retp[outsample])
+ })  # end sapply
> colnames(pnls) <- names(fcasts)
> pnls <- xts::xts(pnls, datev[outsample])
```




```
> # Plot dygraph of out-of-sample PnLs
> dygraphs::dygraph(pnls[endd],
+   main="Autoregressive Strategies Using Rolling Average Forecasts"
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(width=300)
```

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 (`look_back`).

```
> # Perform rolling forecasting
> look_back <- 100
> fcasts <- sapply((look_back+1):nrows, function(tday) {
+   # Define rolling look-back range
+   startp <- max(1, tday-look_back)
+   # Or expanding look-back range
+   # startp <- 1
+   rangev <- startp:(tday-1) # In-sample range
+   # Invert the predictor matrix
+   predinv <- MASS::ginv(predm[rangev, ])
+   # Calculate the fitted AR coefficients
+   coeff <- predinv %*% respv[rangev]
+   # Calculate the out-of-sample forecast
+   predm[tday, ] %*% coeff
+ })  # end sapply
> # Add warmup period
> fcasts <- c(rep(0, look_back), fcasts)
```

draft: Forecasting Returns Using Rolling Autoregressive Models

Adopt the text

Forecasting using an autoregressive model is performed by first fitting an *AR(n)* model to past data, and calculating its coefficients.

The fitted AR coefficients are then applied to calculating the *out-of-sample* forecasts.

The forecasting model depends on two unknown *meta-parameters*: the order n of the *AR(n)* model and the length of the look-back interval (`look_back`).

```
> library(rutils)
> # Calculate the a vector of daily VTI log returns
> pricev <- log(quantmod::Cl(rutils::etfenv$VTI))
> retp <- rutils::diffit(pricev)
> retp <- as.numeric(retp)
> nrows <- NROW(retp)
> # Define predictor matrix for forecasting
> ordmax <- 5
> desv <- sapply(1:ordmax, rutils::lagit, input=respv)
> colnames(desv) <- paste0("pred", 1:NCOL(desv))
> # Add response equal to VTI
> desv <- cbind(retp, desv)
> colnames(desv)[1] <- "response"
> # Specify length of look-back interval
> look_back <- 100
> # Invert the predictor matrix
> rangev <- (nrows-look_back):(nrows-1)
> predinv <- MASS::ginv(desv[rangev, -1])
> # Calculate the fitted AR coefficients
> coeff <- predinv %*% desv[rangev, 1]
> # Calculate the forecast of VTI for nrows
> drop(desv[nrows, -1] %*% coeff)
> # Compare with actual value
> desv[nrows, 1]
```

Backtesting Function for the Forecasting 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 (`look_back`).

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 (`look_back`).

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

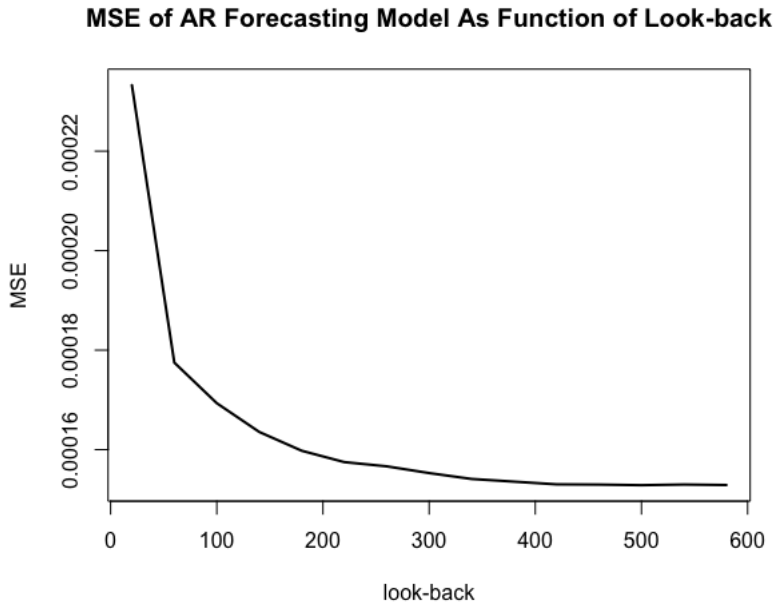
Forecasting Dependence On the Look-back Interval

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 (`look_back`).

The accuracy of the forecasting model increases with longer look-back intervals (`look_back`), 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
> cluster <- makeCluster(ncores)
> # Perform parallel loop under Windows
> look_backs <- seq(20, 600, 40)
> fcasts <- parLapply(cluster, look_backs, sim_fcasts, ordern=6)
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(look_backs, sim_fcasts, ordern=6, mc.cores=nc
```

```
> # 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) <- look_backs
> # Select optimal look_back interval
> look_back <- look_backs[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=look_backs, y=mse[, 1],
+   xlab="look-back", ylab="MSE", type="l", lwd=2,
+   main="MSE of AR Forecasting Model As Function of Look-back")
```

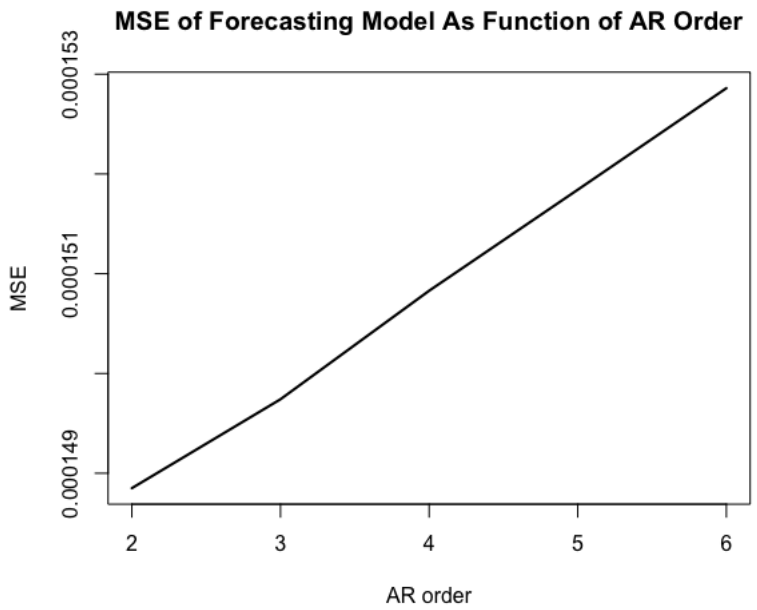
Order Dependence With Fixed Look-back

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 (`look_back`).

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
> cluster <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(cluster, orderv, sim_fcasts,
+   look_back=look_back)
> stopCluster(cluster)  # Stop R processes over cluster under Windo
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   look_back=look_back, 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")
```

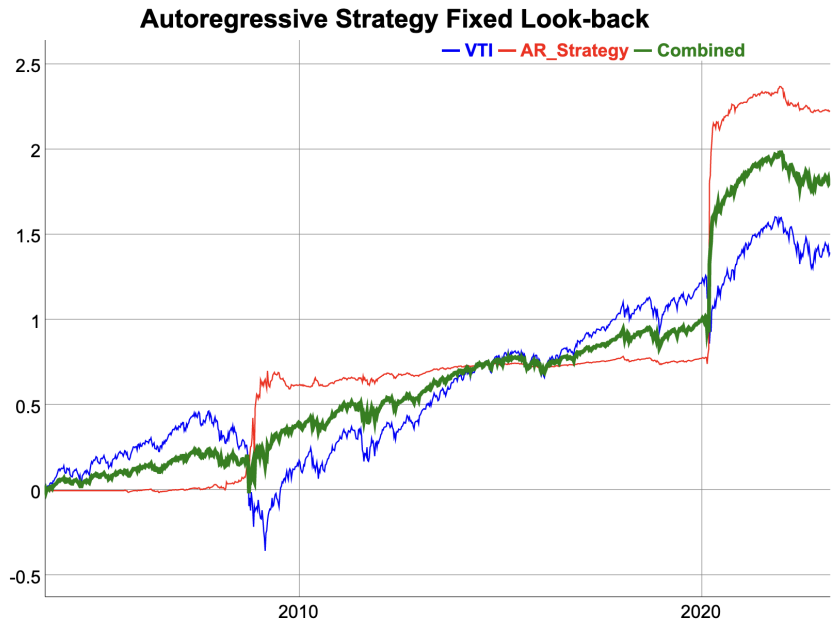
Autoregressive Strategy With Fixed Look-back

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(look_back=look_back, ordern=ordern)
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp)/sd(pnls)
> 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)[endd],
+   main="Autoregressive Strategy Fixed Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

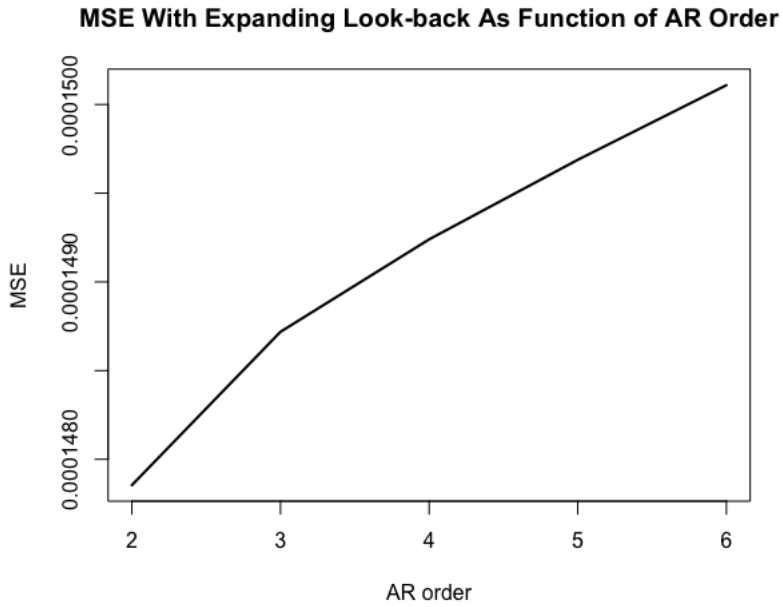
Order Dependence With Expanding Look-back

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 (`look_back`) 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
> cluster <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(cluster, orderv, sim_fcasts,
+   look_back=look_back, fixedlb=FALSE)
> stopCluster(cluster)  # Stop R processes over cluster under Windo
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   look_back=look_back, fixedlb=FALSE, mc.cores=ncores)
```

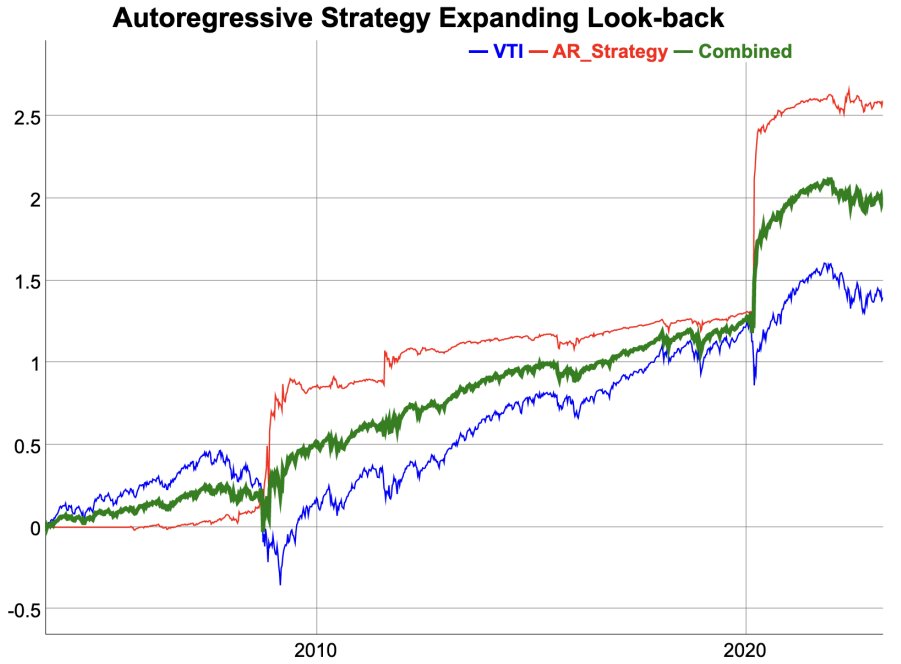
```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, 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(look_back=look_back, ordern=ordern, fixedlb=F
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> pnls <- pnls*sd(retp)/sd(pnls)
> 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)[endd],
+   main="Autoregressive Strategy Expanding Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

draft: Backtesting of Strategies

Backtesting is the simulation of a rolling strategy's performance on historical data.

A *rolling strategy* can be *backtested* by specifying the parameter updating frequency, the formation interval, and the holding period:

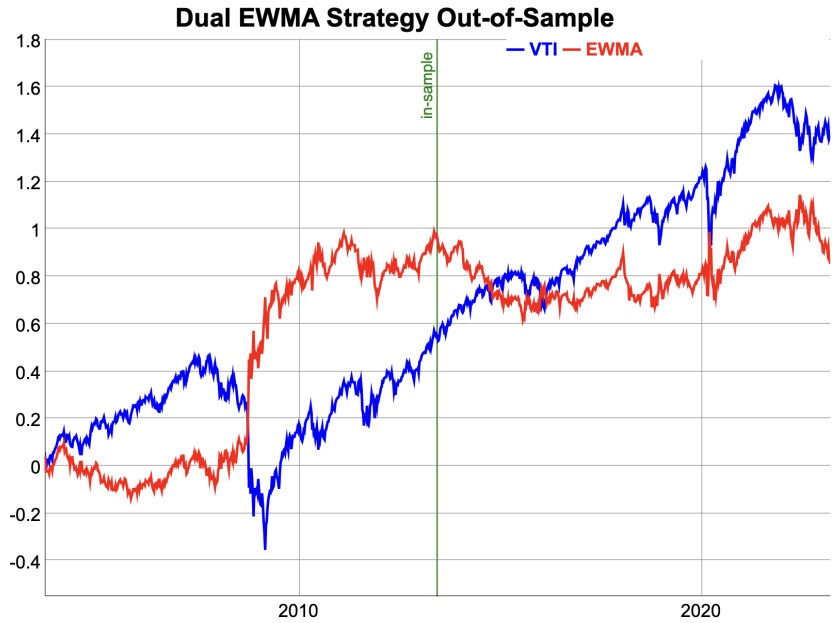
- Calculate the *end points* for parameter updating,
- Define an objective function for parameter optimization,
- Calculate the optimal parameters in the in-sample formation interval,
- Calculate the out-of-sample strategy returns,
- Calculate the transaction costs and subtract them from the strategy returns.

But using a different updating frequency in the *backtest* can produce different values for the optimal trading strategy parameters.

The *backtesting* redefines the problem of finding (tuning) the optimal trading strategy parameters, into the problem of finding the optimal *backtest* (meta-model) parameters.

But the advantage of using the *backtest* meta-model is that it can reduce the number of parameters that need to be optimized.

In-sample, the best *Dual EWMA* strategy performs




```
> # Dygraphs plot with custom line colors
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Dual Crossover Stra
+   dyEvent(zoo::index(wealthv[last(insample)]), label="in-sample",
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
```

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:

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

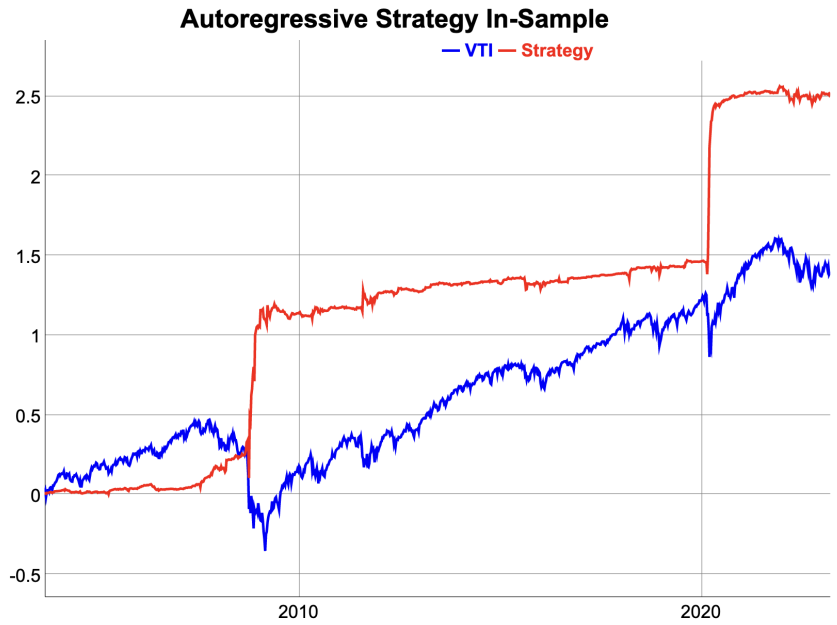
The *residuals* ε_t are assumed to be normally distributed, independent, and stationary.

```
> # 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.

```
> # Simulate autoregressive strategy in-sample
> 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 autoregressive strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Interest Rate Yield Curve and Stock Returns

Daily stock returns have insignificant correlations with the daily changes in interest rates, with the possible exception of the 10-year bond yield.

And these correlations change significantly over time.

```
> # Load constant maturity Treasury rates
> load(file="/Users/jerzy/Develop/lecture_slides/data/rates_data.RI
> # Combine rates into single xts series
> rates <- do.call(cbind, as.list(ratesenv))
> # Sort the columns of rates according bond maturity
> namesv <- colnames(rates)
> namesv <- substr(namesv, start=4, stop=10)
> namesv <- as.numeric(names)
> indeks <- order(names)
> rates <- rates[, indeks]
> # Align rates dates with VTI prices
> closep <- log(quantmod::Cl(rutils::etfenv$VTI))
> colnames(closep) <- "VTI"
> nrows <- NROW(closep)
> datev <- zoo::index(closep)
> rates <- na.omit(rates[datev])
> closep <- closep[zoo::index(rates)]
> datev <- zoo::index(closep)
```

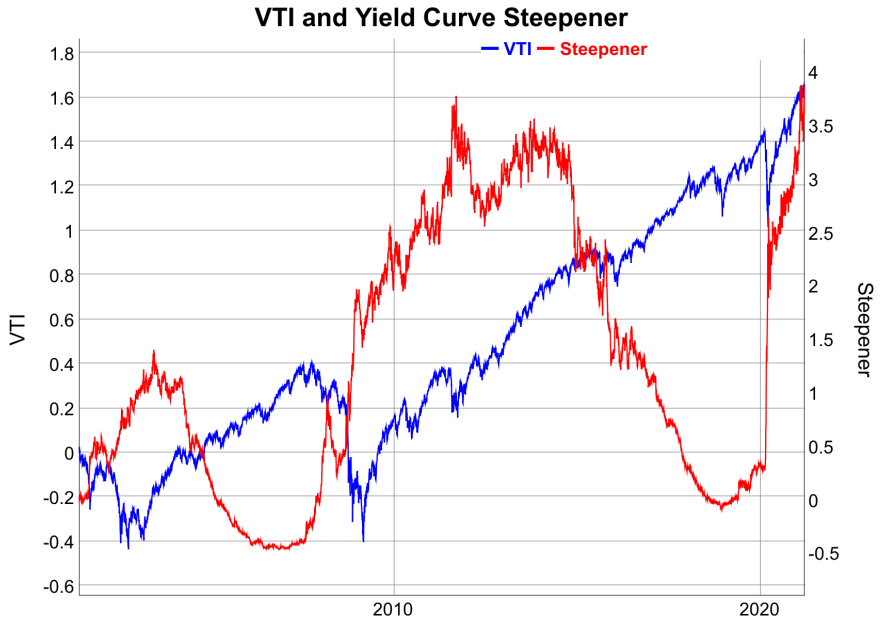
```
> # Calculate VTI returns and IR changes
> retp <- rutils::diffit(log(closep))
> retr <- rutils::diffit(log(rates))
> # Regress VTI returns versus the lagged rate differences
> predm <- rutils::lagit(retr)
> regmod <- lm(retp ~ predm)
> summary(regmod)
> # Regress VTI returns before and after 2012
> summary(lm(retp["/2012"] ~ predm["/2012"]))
> summary(lm(retp["2012/"] ~ predm["2012/"]))
```

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 correlation matrix
> eigend <- eigen(cor(retr))
> pcar <- -(retr %*% eigend$vectors)
> colnames(pcar) <- paste0("PC", 1:6)
> # Define predictor as the YC PCAs
> predm <- rutils::lagit(pcar)
> regmod <- lm(retp ~ predm)
> summary(regmod)
```




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

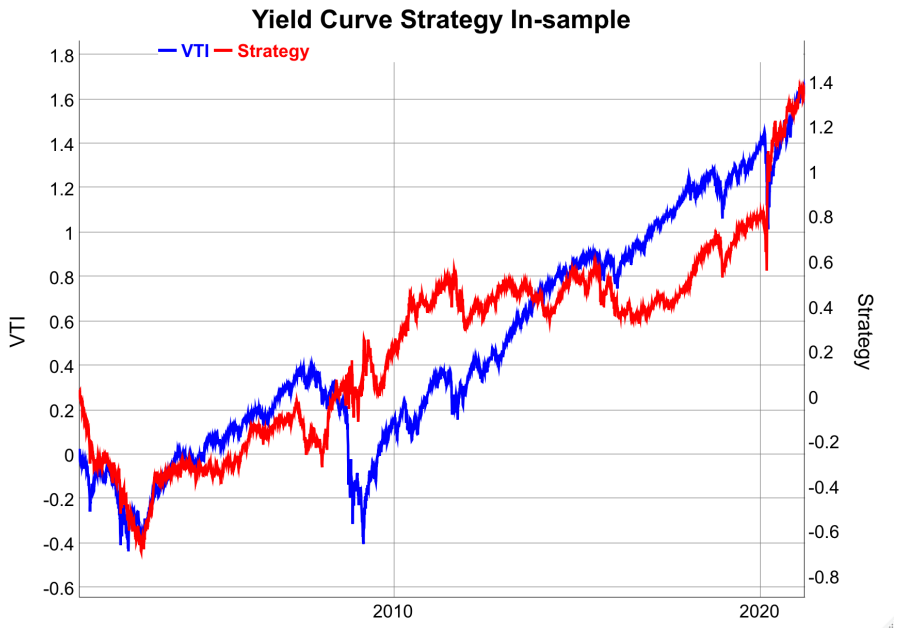
Yield Curve Strategy In-Sample

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 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)
> pnls <- sign(fcasts)*response
> # Calculate in-sample factors
> factors <- (predm*coeff)
> apply(factors, 2, sd)
```

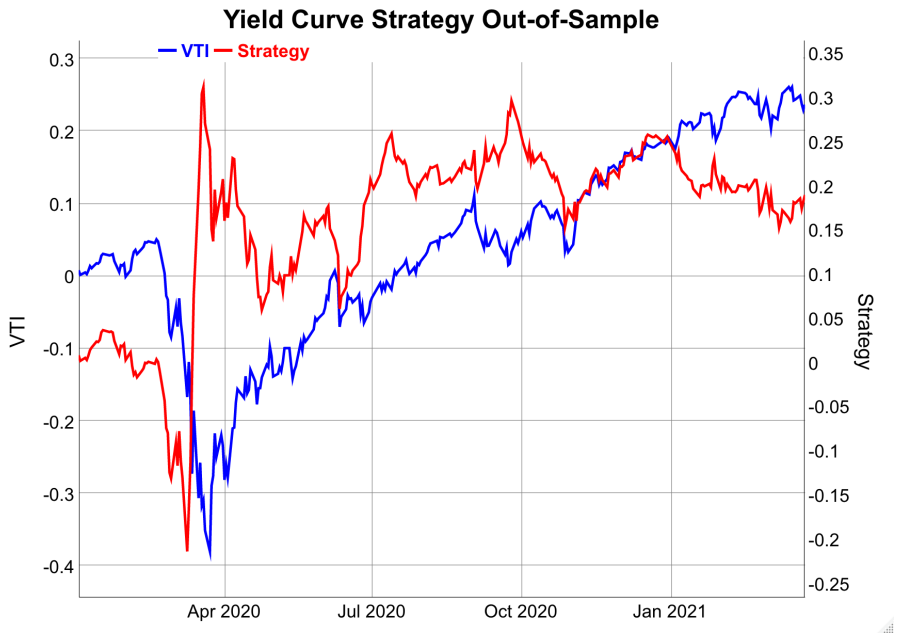
```
> # Plot dygraph of in-sample IR strategy
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Yield Curve Strategy In-sample") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="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 data, and forecasts are calculated using the test data.

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 inverse of predictor in-sample
> invreg <- MASS::ginv(predm[insample, ])
> # Calculate coefficients in-sample
> coeff <- drop(invreg %*% respv[insample, ])
> # Calculate forecasts and pnls out-of-sample
> fcasts <- (predm[outsample, ] %*% coeff)
> pnls <- sign(fcasts)*respv[outsample, ]
```

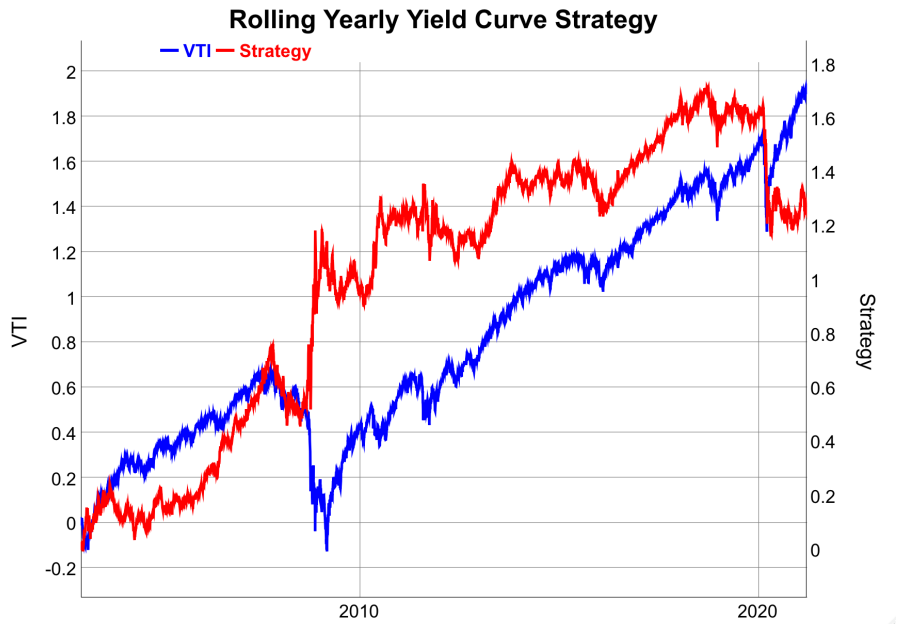



```
> # Plot dygraph of out-of-sample IR PCA strategy
> wealthv <- cbind(retp[outsample, ], pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Yield Curve Strategy Out-of-Sample") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="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 year. The coefficients are applied to calculate out-of-sample forecasts in the following year.

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




```
> # Define yearly dates
> format(datev[1], "%Y")
> years <- paste0(seq(2001, 2022, 1), "-01-01")
> years <- as.Date(years)
> # Perform loop over yearly dates
> pnls <- lapply(3:(NROW(years)-1), function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > years[tday-1]) & (datev < years[tday])
+   outsample <- (datev >= years[tday]) & (datev < years[tday+1])
+   # Calculate coefficients in-sample
+   invreg <- MASS::ginv(predm[insample, ])
+   coeff <- drop(invreg %*% respv[insample, ])
+   # Calculate forecasts and pnls out-of-sample
+   fcasts <- (predm[outsample, ] %*% coeff)
+   sign(fcasts)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```

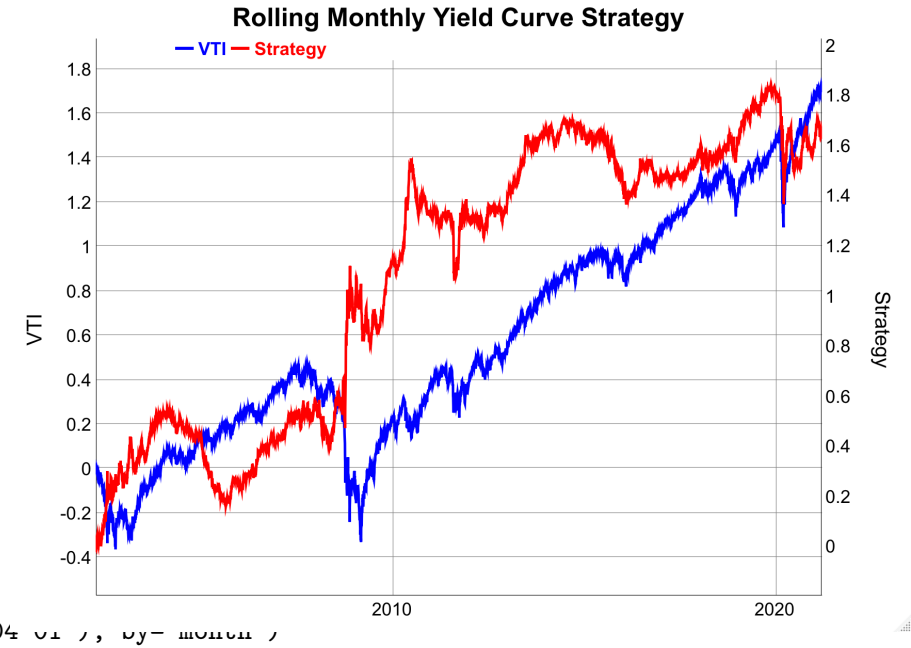
```
> # Plot dygraph of rolling yearly IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Yearly Yield Curve Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="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 11 months. 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.



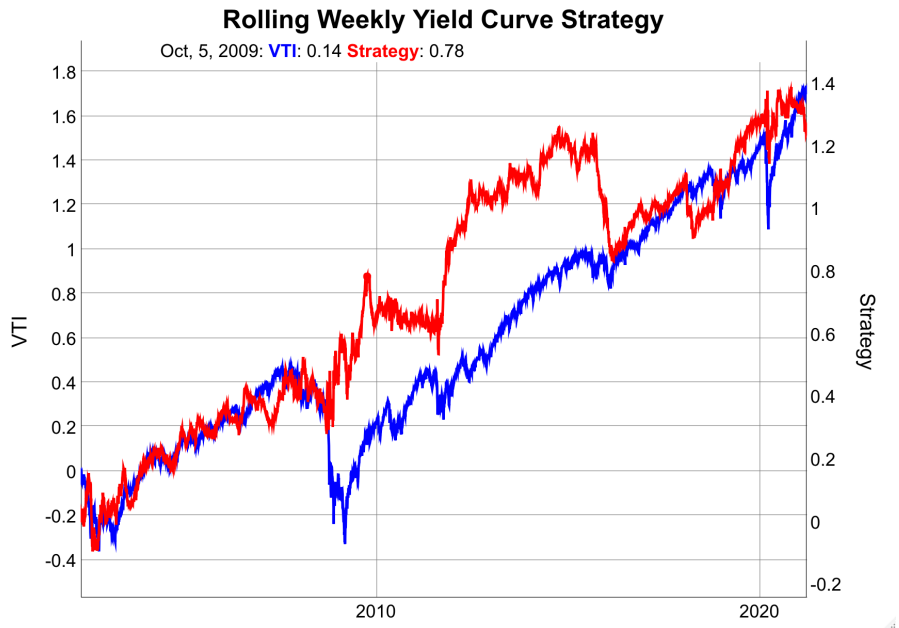

```
> # Define monthly dates
> format(datev[1], "%m-%Y")
> format(datev[NROW(datev)], "%m-%Y")
> months <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-01"), by="month")
> # Perform loop over monthly dates
> pnls <- lapply(12:(NROW(months)-1), function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > months[tday-11]) & (datev < months[tday])
+   outsample <- (datev > months[tday]) & (datev < months[tday+1])
+   # Calculate forecasts and pnls out-of-sample
+   invreg <- MASS::ginv(predm[insample, ])
+   coeff <- drop(invreg %*% respv[insample, ])
+   fcasts <- (predm[outsample, ] %*% coeff)
+   sign(fcasts)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```

```
> # Plot dygraph of rolling monthly IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Monthly Yield Curve Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="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 10 weeks. The coefficients are applied to perform out-of-sample forecasts in the following week.

```
> # Define weekly dates
> weeks <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-
> # Perform loop over weekly dates
> pnls <- lapply(51:(NROW(weeks)-1), function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > weeks[tday-10]) & (datev < weeks[tday])
+   outsample <- (datev > weeks[tday]) & (datev < weeks[tday+1])
+   # Calculate forecasts and pnls out-of-sample
+   invreg <- MASS::ginv(predm[insample, ])
+   coeff <- drop(invreg %*% respv[insample, ])
+   fcasts <- (predm[outsample, ] %*% coeff)
+   sign(fcasts)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```




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

Regularization of the Inverse Predictor Matrix

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

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

Where $\mathbb{U}$ and $\mathbb{V}$ are the *singular matrices*, and $\mathbf{\Sigma}$ 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:

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

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

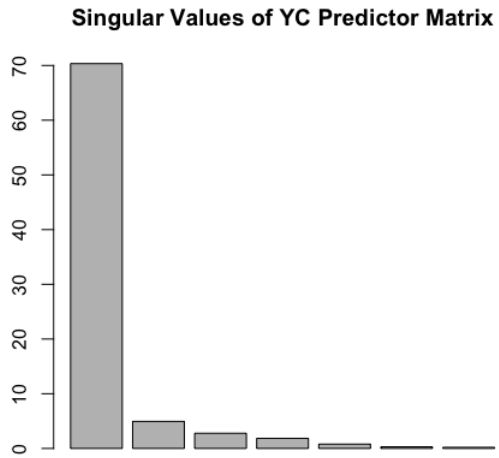
Regularization is the removal of zero singular values, to make calculating the inverse matrix possible.

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

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

Where $\mathbb{U}_n$, $\mathbb{V}_n$ and $\mathbf{\Sigma}_n$ 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$d, main="Singular Values of YC Predictor Matrix")
> # Calculate generalized inverse from SVD
> invsvd <- svdec$v %*% (t(svdec$u) / svdec$d)
> # Verify inverse property of inverse
> all.equal(zoo::coredata(predm), predm %*% invsvd %*% predm)
> # Calculate 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.

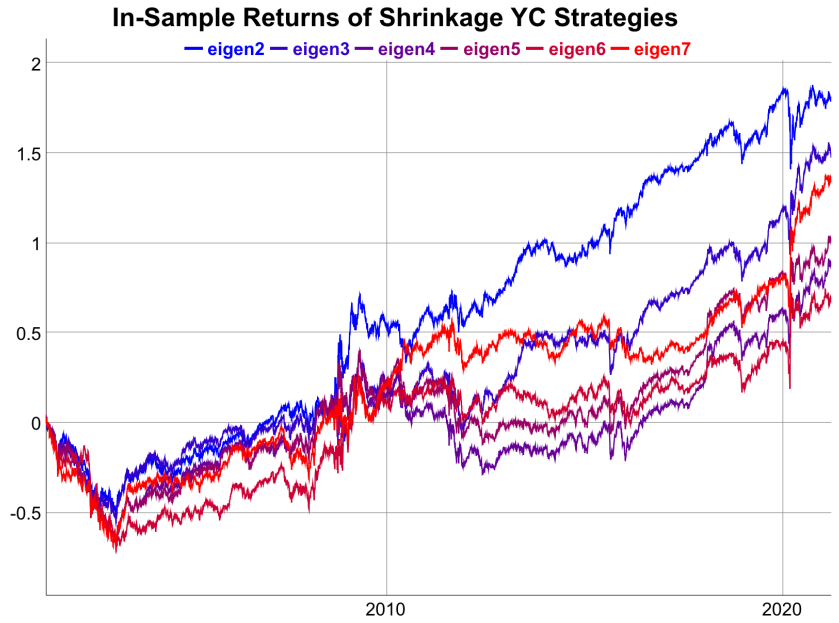
```
> # Calculate reduced inverse from SVD
> dimax <- 3
> invred <- svdec$v[, 1:dimax] %*%
+   (t(svdec$u[, 1:dimax]) / svdec$d[1:dimax])
> # Inverse property fails for invred
> all.equal(zoo::coredata(predm), predm %*% invred %*% predm)
> # Calculate reduced inverse using RcppArmadillo
> invrcpp <-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

Yield Curve 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
> dimaxs <- 2:7
> pnls <- lapply(dimaxs, function(dimax) {
+   invred <- HighFreq::calc_inv(predm, dimax=dimax)
+   coeff <- drop(invred %*% respv)
+   fcasts <- (predm %*% coeff)
+   sign(fcasts)*response
+ })
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("eigen", dimaxs)
```

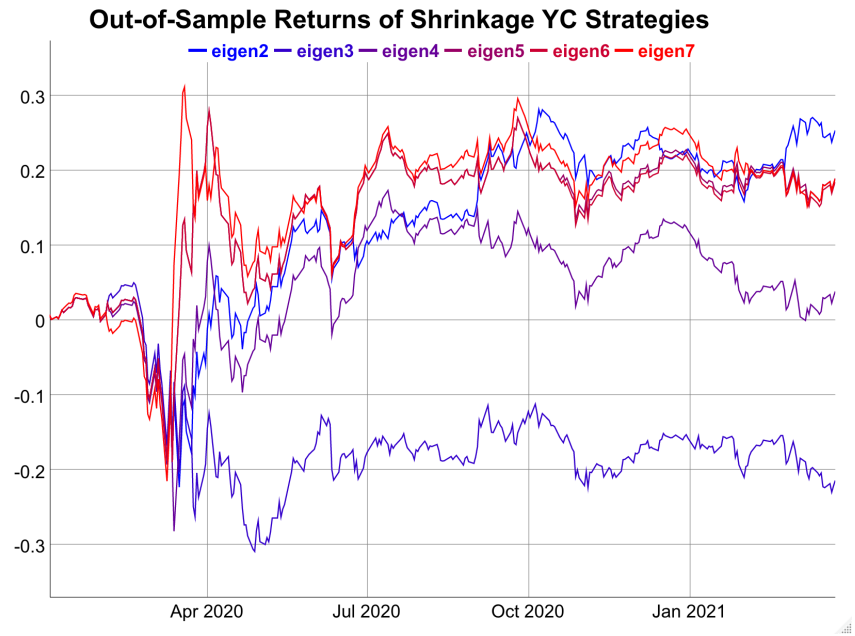
```
> # Plot dygraph of in-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="In-Sample Returns of Shrinka
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Yield Curve 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 data, and forecasts are calculated using the test data.

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
> dimaxs <- 2:7
> pnls <- lapply(dimaxs, function(dimax) {
+   invred <- HighFreq::calc_inv(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("eigen", dimaxs)
```



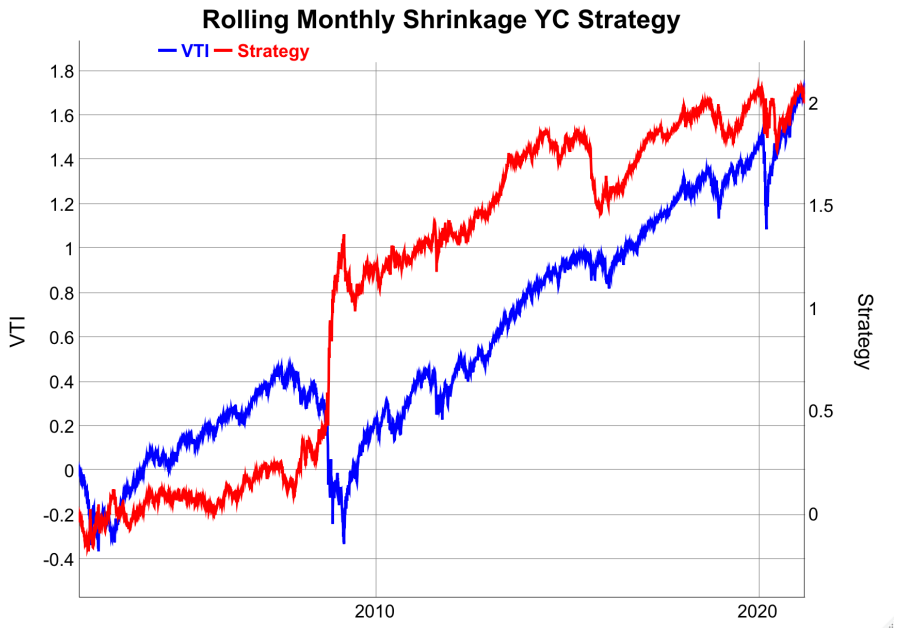

```
> # Plot dygraph of out-of-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="Out-of-Sample Returns of Shr
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Monthly Yield Curve Strategy With Dimension Reduction

The shrinkage rolling monthly strategy performs better than the standard strategy because regularization allows using shorter `look_back` 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")
> months <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04
> # Perform loop over monthly dates
> look_back <- 6
> dimax <- 3
> pnls <- lapply((look_back+1):(NROW(months)-1), function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > months[tday-look_back]) & (datev < months
+   outsample <- (datev > months[tday]) & (datev < months[tday+1])
+   # Calculate forecasts and pnls out-of-sample
+   invred <- HighFreq::calc_inv(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 IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Monthly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="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 4 weeks. The coefficients are applied to perform out-of-sample forecasts in the following week.

```
> # Define weekly dates
> weeks <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-
> # Perform loop over weekly dates
> look_back <- 4
> dimax <- 4
> pnls <- lapply((look_back+1):(NROW(weeks)-1), function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > weeks[tday-look_back]) & (datev < weeks[tda
+   outsample <- (datev > weeks[tday]) & (datev < weeks[tday+1])
+   # Calculate forecasts and pnls out-of-sample
+   invred <- HighFreq::calc_inv(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 IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Weekly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="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.

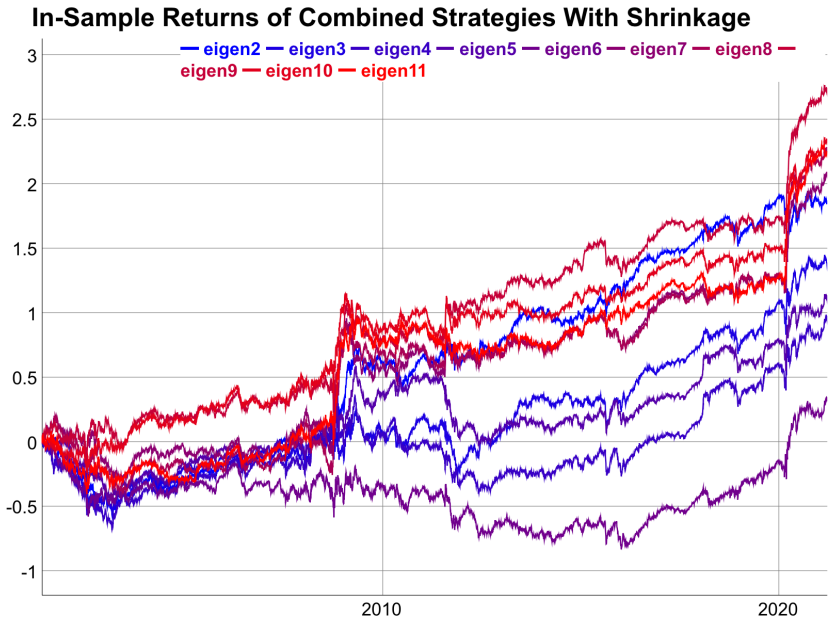
```
> # Load the yield curve data
> load(file="/Users/jerzy/Develop/lecture_slides/data/rates_data.RDa
> rates <- do.call(cbind, as.list(ratesenv))
> namesv <- colnames(rates)
> namesv <- substr(namesv, start=4, stop=10)
> namesv <- as.numeric(names)
> indeks <- order(names)
> rates <- rates[, indeks]
> closep <- log(quantmod::Cl(rutils::etfenv$VTI))
> colnames(closep) <- "VTI"
> nrows <- NROW(closep)
> datev <- zoo::index(closep)
> rates <- na.omit(rates[datev])
> closep <- closep[zoo::index(rates)]
> datev <- zoo::index(closep)
> retp <- rutils::diffit(log(closep))
> retr <- rutils::diffit(log(rates))
> # Create a combined predictor matrix
> ordmax <- 5
> predm <- sapply(1:ordmax, 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
> dimaxs <- 2:11
> pnls <- lapply(dimaxs, function(dimax) {
+   invred <- HighFreq::calc_inv(predm, dimax=dimax)
+   coeff <- drop(invred %*% respv)
+   fcasts <- (predm %*% coeff)
+   sign(fcasts)*response
+ })
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("eigen", dimaxs)
```

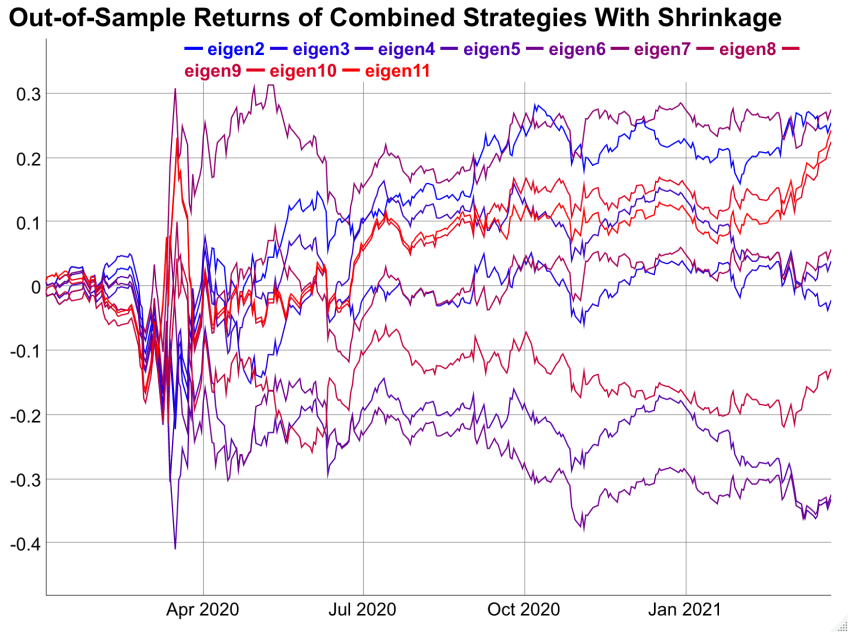
```
> # Plot dygraph of in-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="In-Sample Returns of Combine
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

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 data, and forecasts are calculated using the test data.

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
> dimaxs <- 2:11
> pnls <- lapply(dimaxs, function(dimax) {
+   invred <- HighFreq::calc_inv(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("eigen", dimaxs)
```

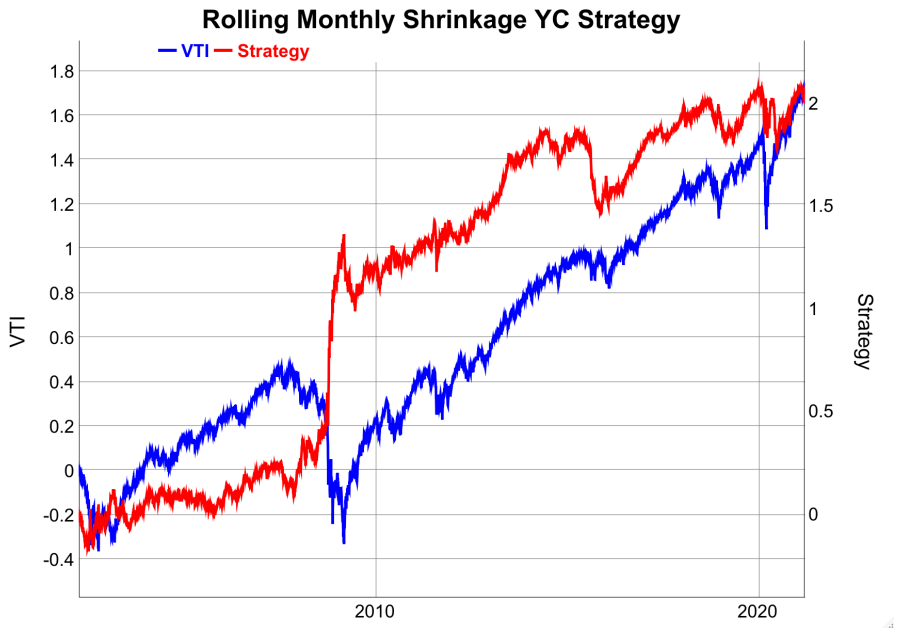



```
> # Plot dygraph of out-of-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="Out-of-Sample Returns of Co
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

draft: Rolling Monthly Combined Strategy With Dimension Reduction

The shrinkage rolling monthly strategy performs better than the standard strategy because regularization allows using shorter `look_back` 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")
> months <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04
> # Perform loop over monthly dates
> look_back <- 6
> dimax <- 3
> pnls <- lapply((look_back+1):(NROW(months)-1), function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > months[tday-look_back]) & (datev < months
+   outsample <- (datev > months[tday]) & (datev < months[tday+1])
+   # Calculate forecasts and pnls out-of-sample
+   invred <- HighFreq::calc_inv(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 IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Monthly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[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
> weeks <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-
> # Perform loop over weekly dates
> look_back <- 8
> dimax <- 4
> pnls <- lapply((look_back+1):(NROW(weeks)-1), function(tday) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > weeks[tday-look_back]) & (datev < weeks[tda
+   outsample <- (datev > weeks[tday]) & (datev < weeks[tday+1])
+   # Calculate forecasts and pnls out-of-sample
+   invred <- HighFreq::calc_inv(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 IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Weekly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

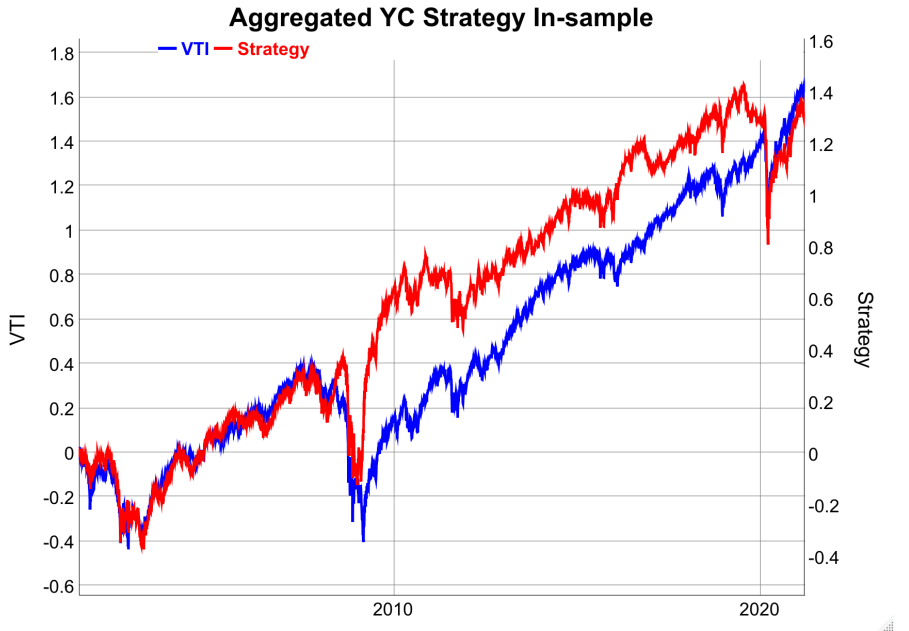
draft: Forecasts Using Aggregated Predictor

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, look_back=nagg)
+   predm <- cbind(rep(1, NROW(predm)), predm)
+   predm <- rutils::lagit(predm)
+   regmod <- lm(respv ~ predm - 1)
+   modsum <- summary(regmod)
+   max(abs(modsum$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, look_back=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)*response
> # Plot dygraph of in-sample IR strategy
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Aggregated YC Strategy In-sample") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

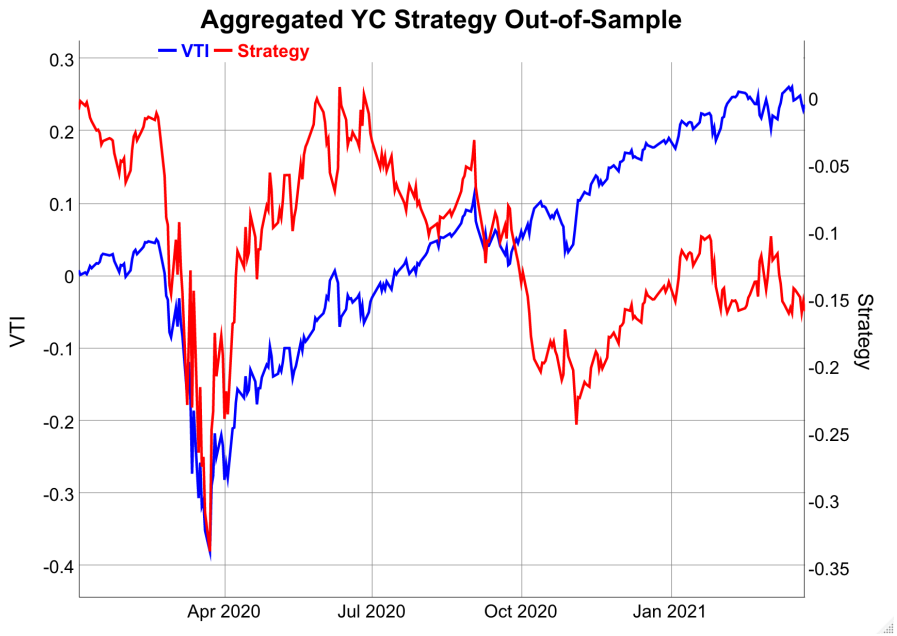
draft: Aggregated Forecasts Out-of-Sample

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 data, and forecasts are calculated using the test data.

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")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Aggregated YC Strategy Out-of-Sample") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```