FRE7241 Algorithmic Portfolio Management

Lecture#4, Spring 2024

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

April 9, 2024

NYU | TANDON SCHOOL OF ENGINEERING

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)
```

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.

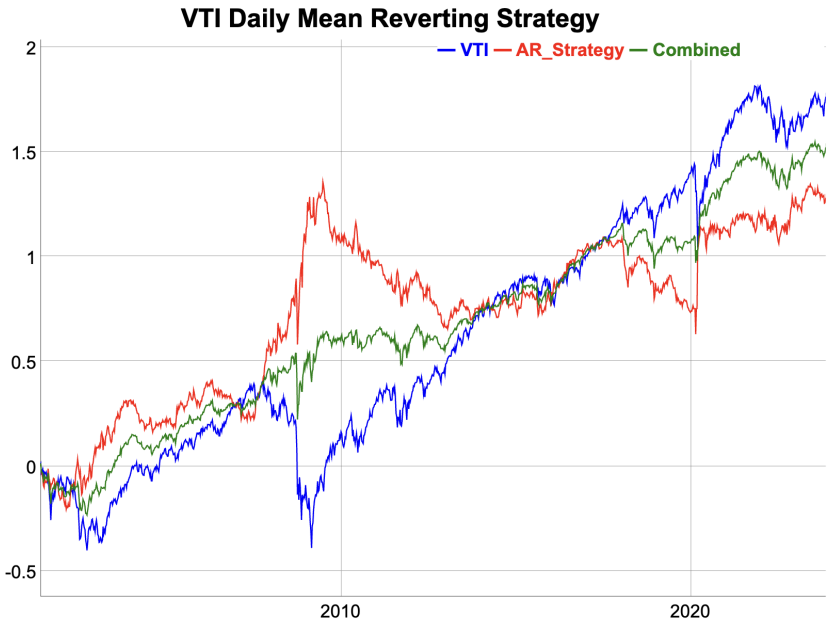
Daily stock returns often exhibit some negative autocorrelations.

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.

Combining the mean reverting strategy with the stock produces the best risk-adjusted returns.

```
> # 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 <- -sign(retp)
> pnls <- retp*rutils::lagit(posv, lagg=1)
> # Subtract transaction costs from the pnls
> bidask <- 0.0001 # Bid-ask spread equal to 1 basis point
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
```



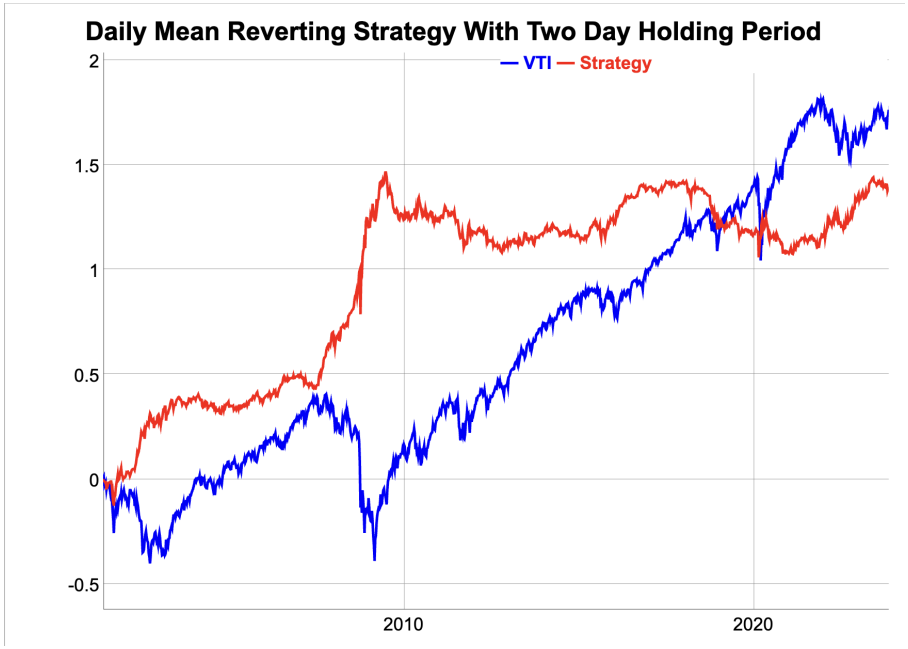

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI Daily Mean Reverting Strategy") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   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::roll_sum(sign(retp), look_back=2)/2
> pnls <- retp*rutils::lagit(posv)
```



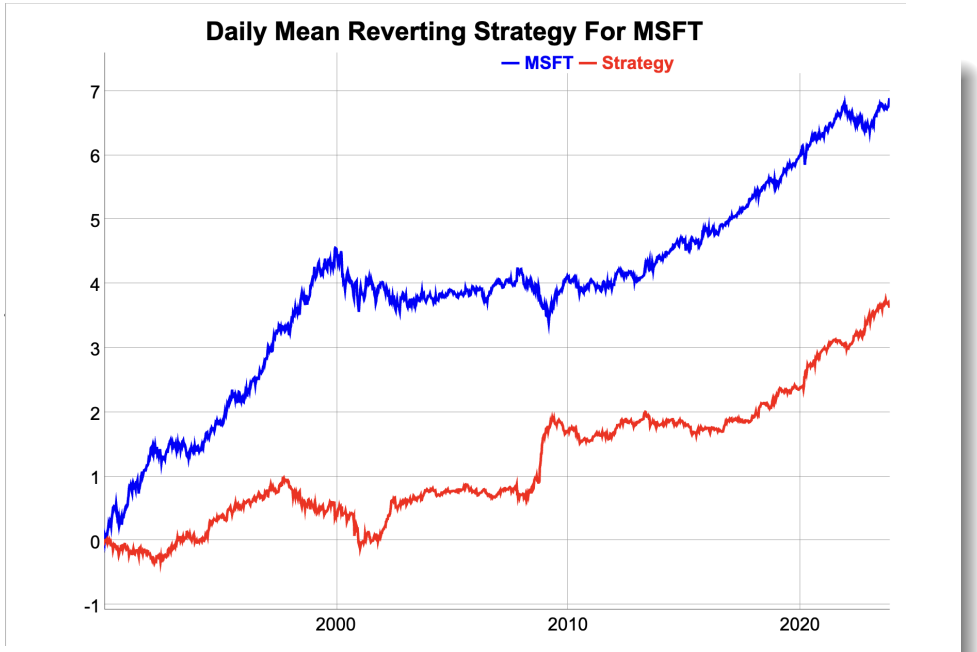

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, 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(retstock$MSFT)
> rutils::plot_acf(retp)
> # Simulate mean reverting strategy with two day holding period
> posv <- -rutils::roll_sum(sign(retp), look_back=2)/2
> pnls <- retp*rutils::lagit(posv)
```



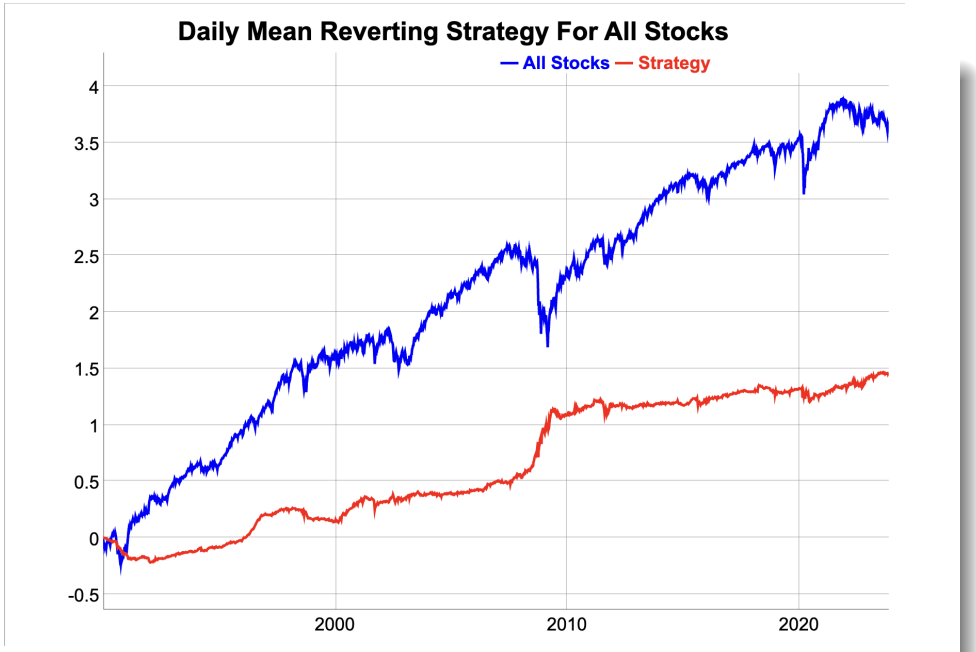

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, 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
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnll <- mclapply(retstock, function(retp) {
+   retp <- na.omit(retp)
+   posv <- -rutils::roll_sum(sign(retp), look_back=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]
```

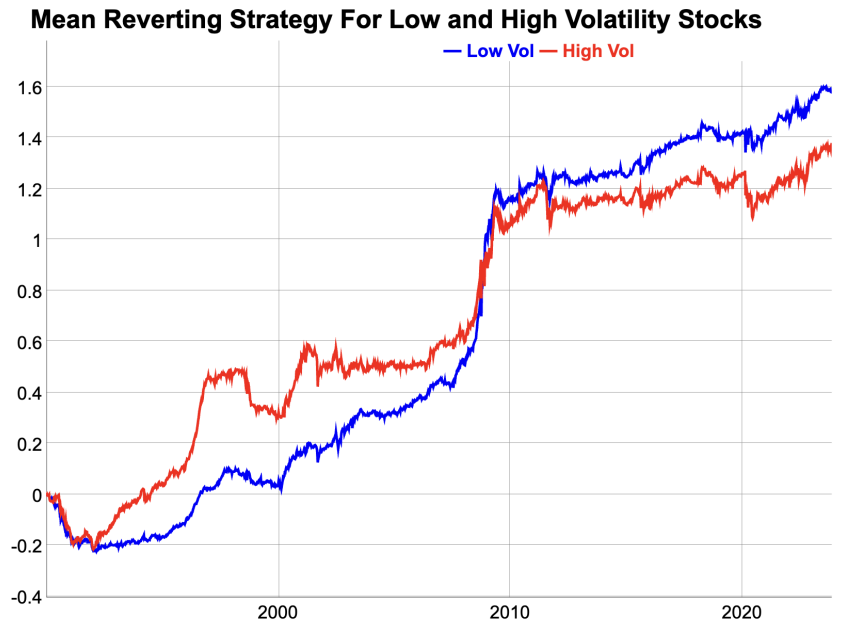



```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("All Stocks", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting 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)
```

Mean Reverting Strategy For Low and High Volatility Stocks

The daily mean reverting 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 mean reverting strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Mean Reverting Strategy For Low and High Volatility Stocks
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

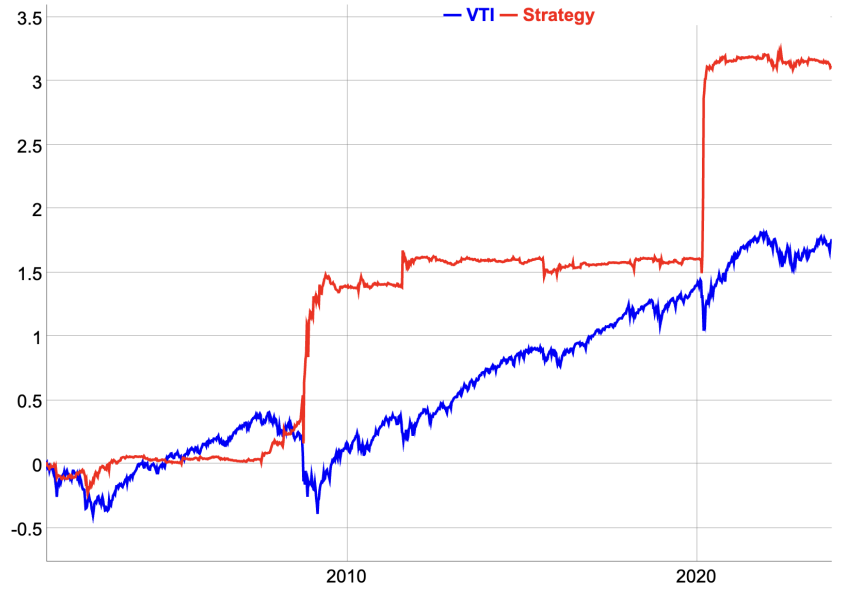
The EMA Mean-Reversion Strategy

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 # Bid-ask spread equal to 1 basis point
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
> # 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 mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI EMA Daily Mean Reverting Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

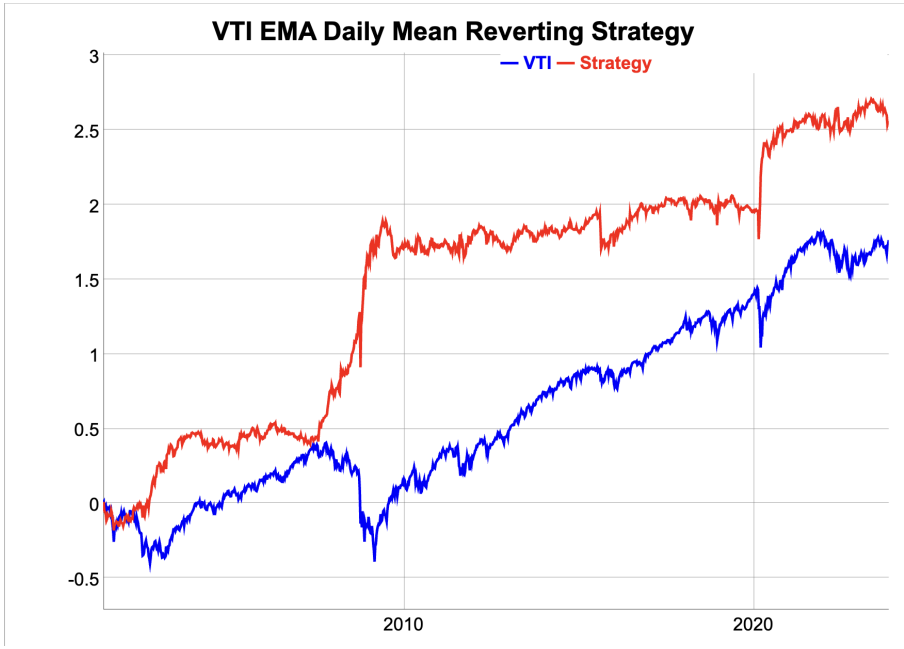
The EMA Mean-Reversion Strategy Scaled By Volatility

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 variance of a time series using exponential weights.

```
> # Calculate the trailing volatility
> volv <- HighFreq::run_var(retp, lambda=0.5)
> volv <- sqrt(volv)
> # Scale the returns by their trailing volatility
> retsc <- ifelse(volv > 0, retp/volv, 0)
> # Calculate the EMA returns
> retma <- HighFreq::run_mean(retsc, lambda=0.1)
> # Calculate the positions and PnLs
> posv <- -rutils::lagit(retma, lagg=1)
> pnls <- retp*posv
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
> # 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 mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI EMA 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)
> # 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:

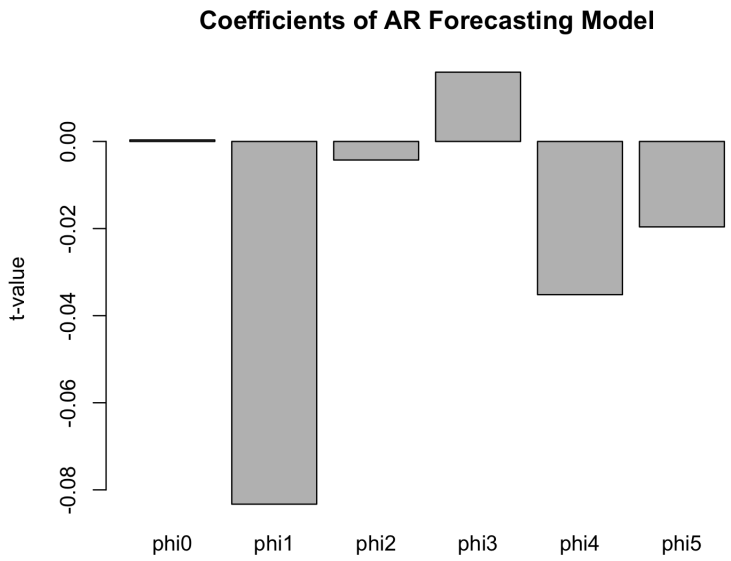
$$f_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n}$$

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 <- predinv %*% respv
> # 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="t-value", col="grey",
+   main="Coefficients of AR Forecasting Model")
```

The t-values of the Autoregressive Coefficients

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

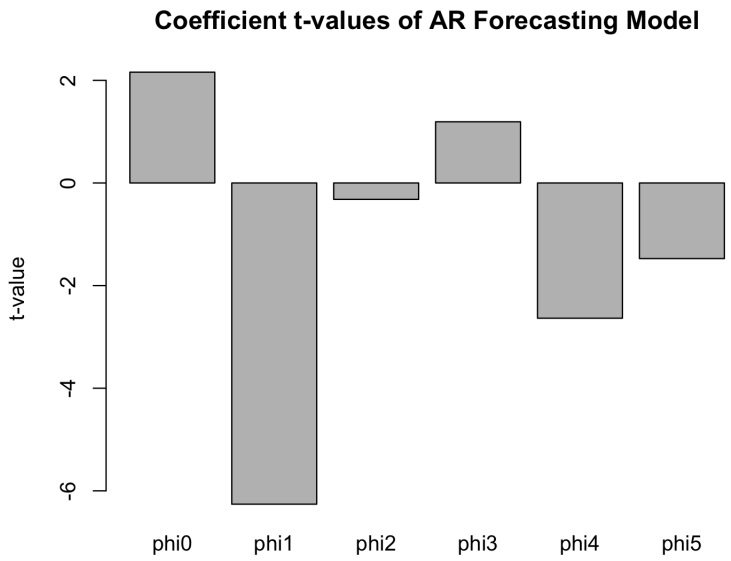
The variance of the autoregressive coefficients σ^2_{φ} is equal to the variance of the forecast residuals σ^2_{ε} divided by the squared *predictor matrix* $\mathbb{P}$:

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

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)
> coeffsd <- sqrt(diag(covmat))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coeffsd)
> coeffn <- paste0("phi", 0:(NROW(coefft)-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")
```

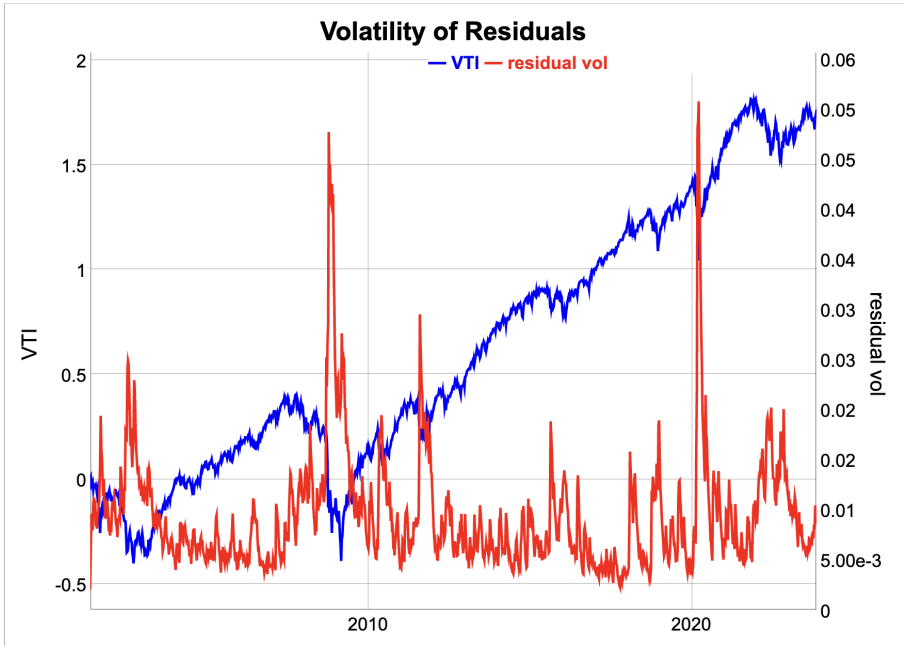
Residuals of Autoregressive Forecasting Model

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))
```




```
> # 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", strokeWidth=2, col="blue") %>%
+   dySeries(name="residual vol", axis="y2", strokeWidth=2, col="red
+   dyLegend(show="always", width=300)
```

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.

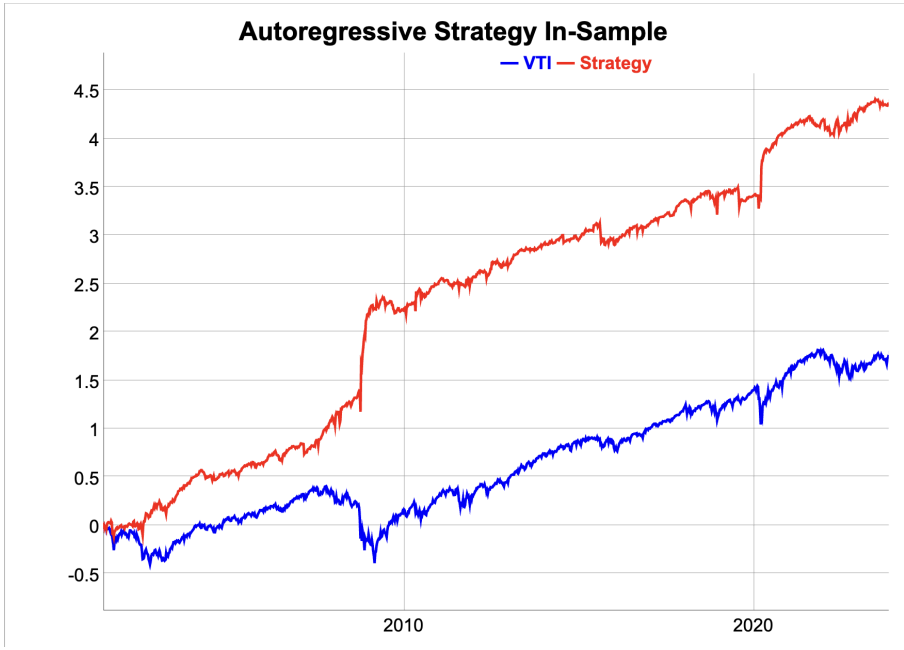
The autoregressive strategy invests dollar amounts of *VTI* stock proportional to the in-sample forecasts.

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.

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.




```
> # Scale the forecasts by their volatility
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.2))
> posv <- ifelse(fcastv > 0, fcasts/fcastv, 0)
> # Simulate autoregressive strategy in-sample
> pnls <- retp*posv
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
> # 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 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)
```

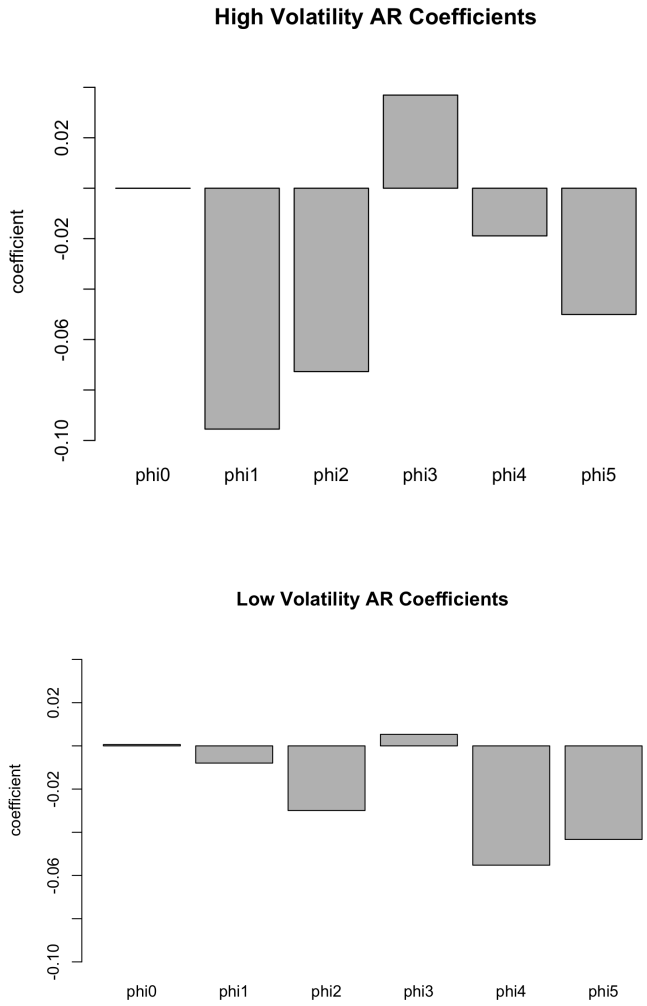
Autoregressive Coefficients in Periods of Low and High Volatility

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.

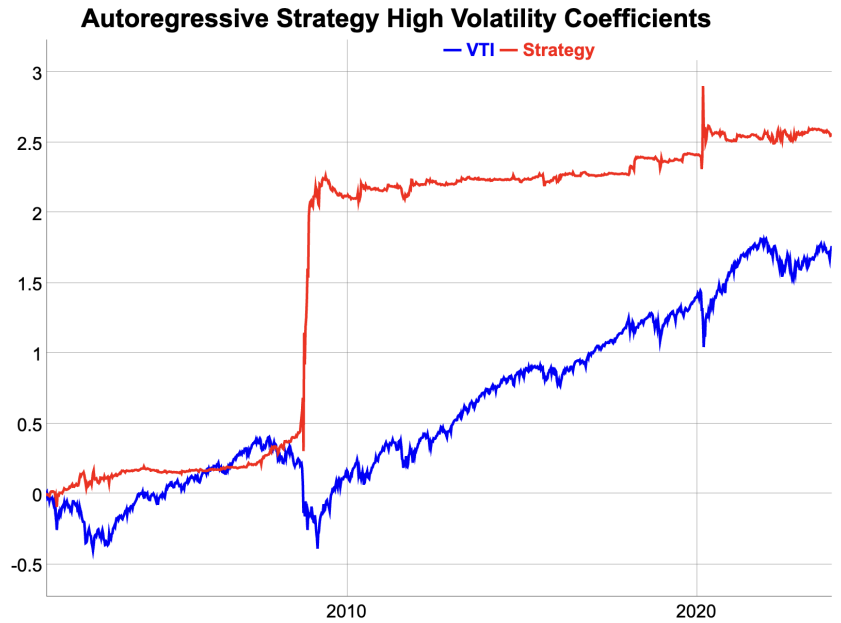
```
> # Calculate the high volatility AR coefficients
> respv <- retp["2008/2011"]
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> predinv <- MASS::ginv(predm)
> coeffh <- drop(predinv %*% respv)
> coeffn <- paste0("phi", 0:(NROW(coeffh)-1))
> barplot(coeffh ~ coeffn, 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)
> barplot(coeffl ~ coeffn, main="Low Volatility AR Coefficients",
+   xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
```

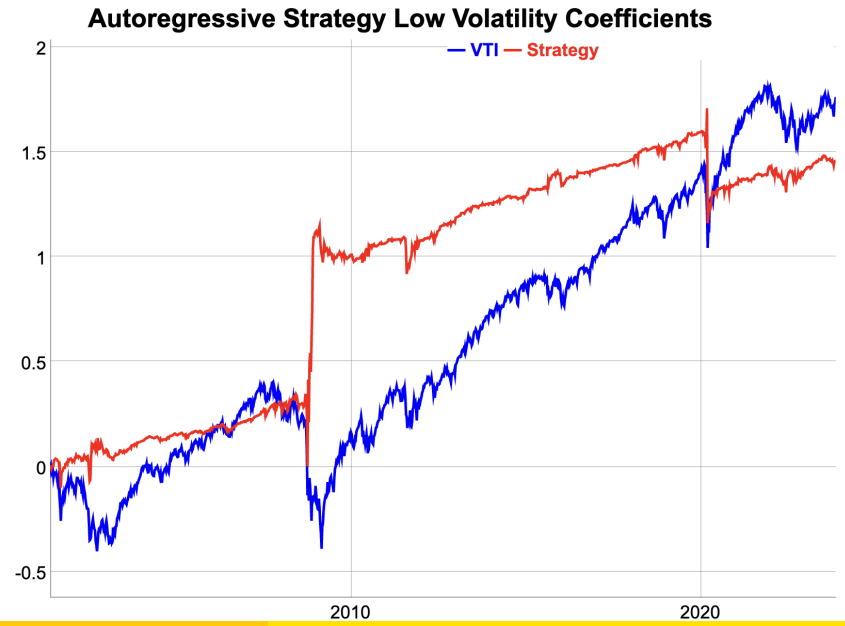


Performance of Low and High Volatility Autoregressive Coefficients

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 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
> 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 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 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.

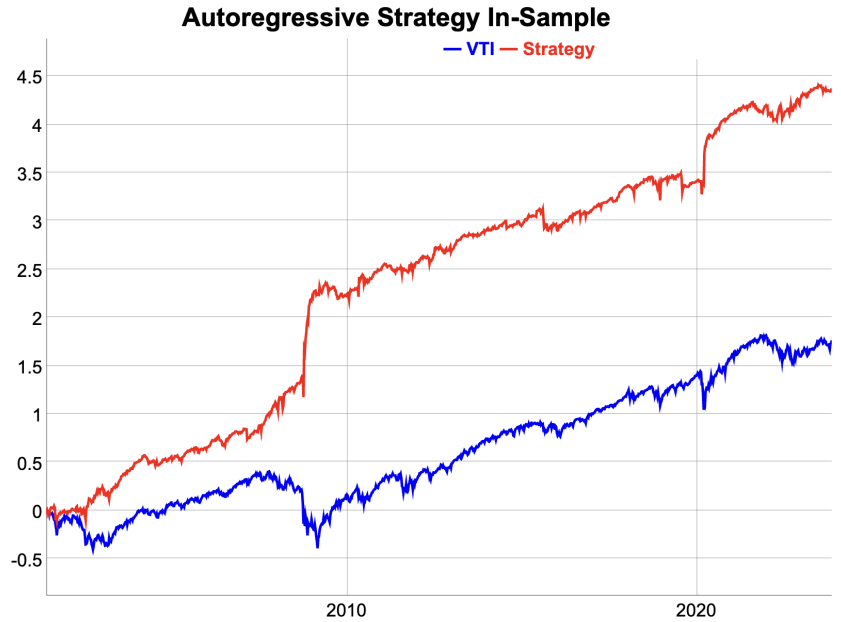
Doesn't work because the square forecast errors are similar.

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 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:

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

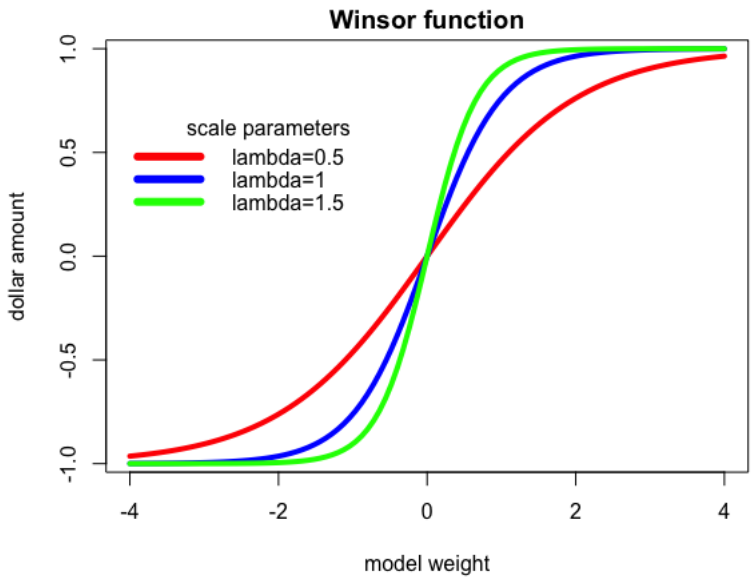
Where λ is the scale parameter.

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.



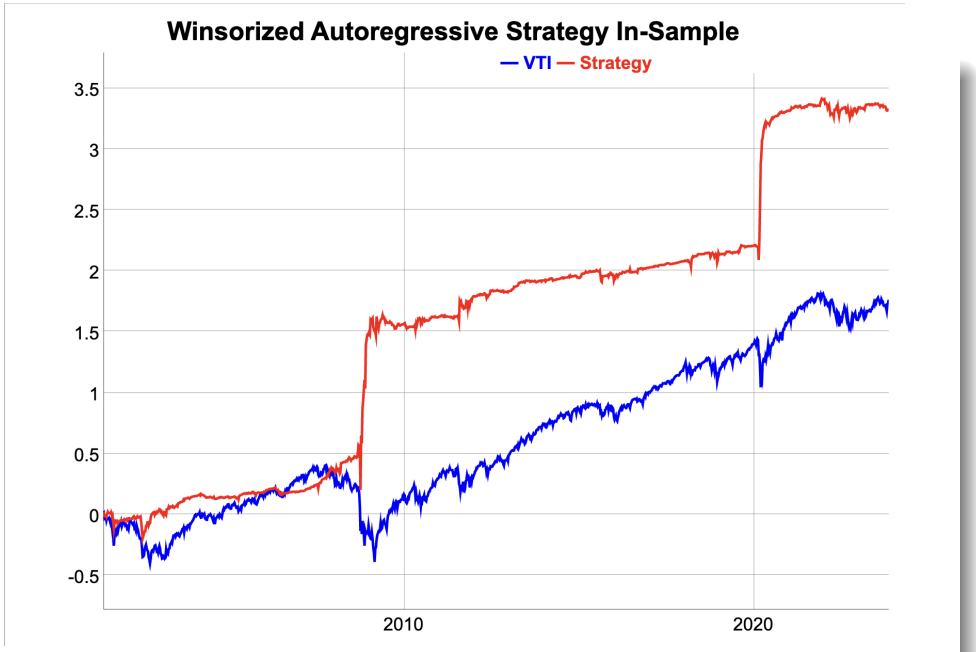

```
> lambdav <- c(0.5, 1, 1.5)
> colorv <- c("red", "blue", "green")
> # Define the winsor function
> winsorfun <- function(retp, lambdaf) tanh(lambdaf*retp)
> # Plot three curves in loop
> for (indeks in 1:3) {
+   curve(expr=winsorfun(x, lambda=lambdav[indeks]),
+ xlim=c(-4, 4), type="l", lwd=4,
+ 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 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[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 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)
```

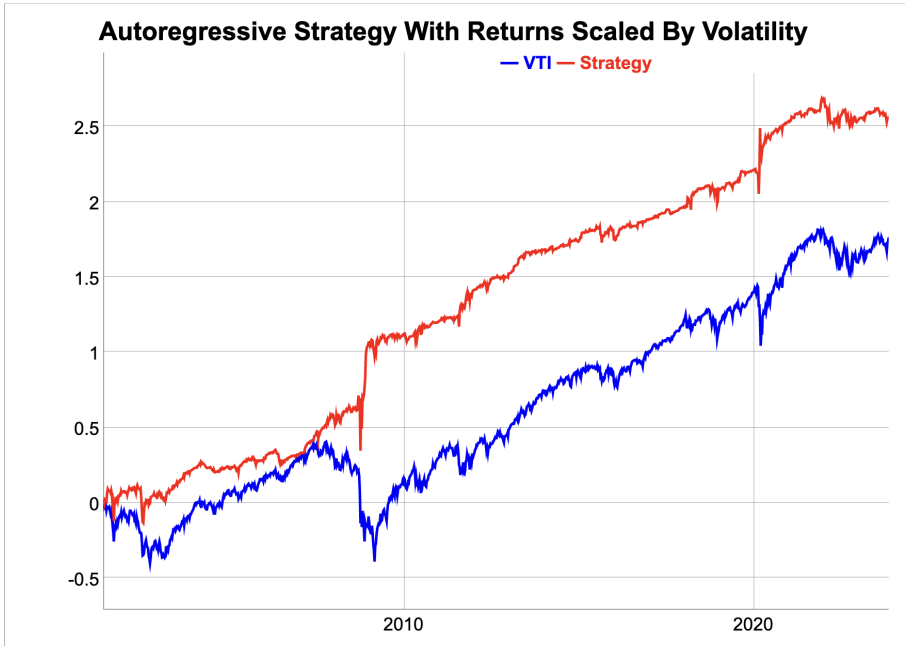
Autoregressive Strategy With Returns Scaled By Volatility

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.

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)
> 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[retp<0])/sd(pnls[pnls<0])
```



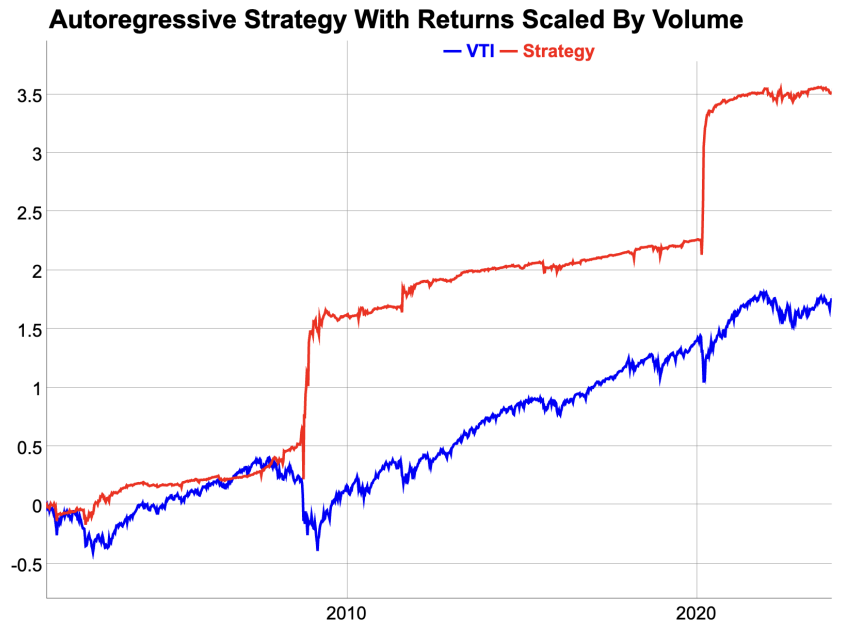

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   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=0.25)
> # Scale the returns using volume clock to trading time
> retsc <- ifelse(volumv > 0, volumr*retp/volumv, 0)
> # Calculate the AR coefficients
> respv <- retsc
> 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 <- predinv %*% respv
> # 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[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 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)
```

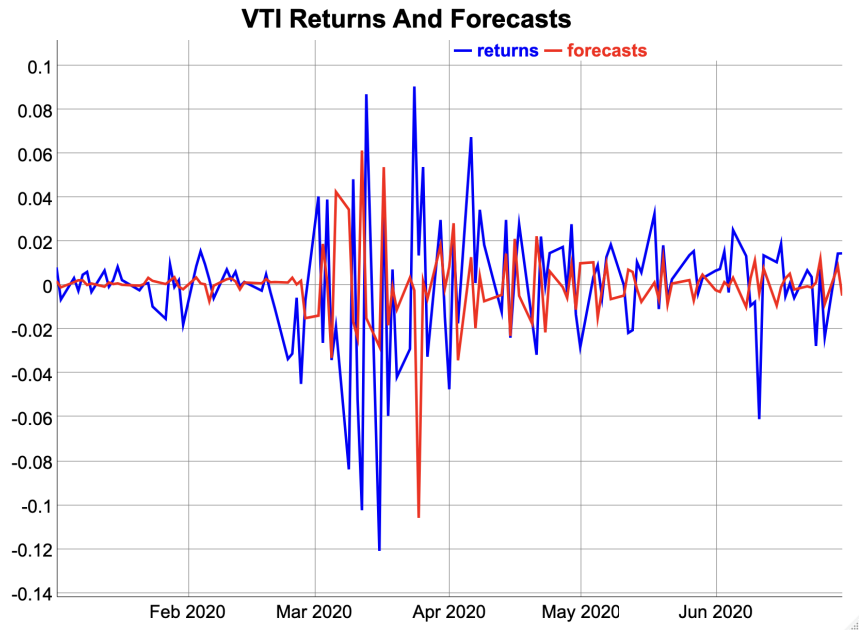
Mean Squared Error of the Autoregressive Forecasting Model

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: $\varepsilon_i = f_t - r_t$:

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

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

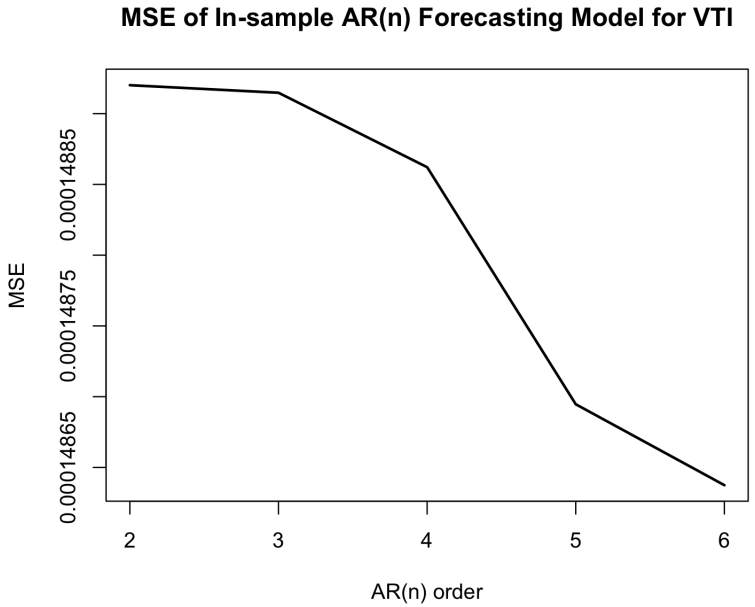
In-sample Order Selection of Autoregressive Forecasting

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")
```

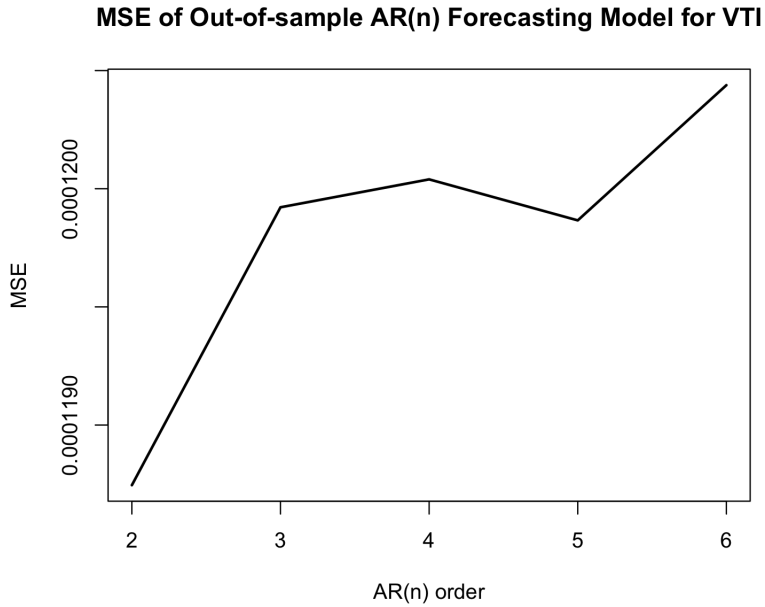
Out-of-Sample Forecasting Using Autoregressive Models

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")
```

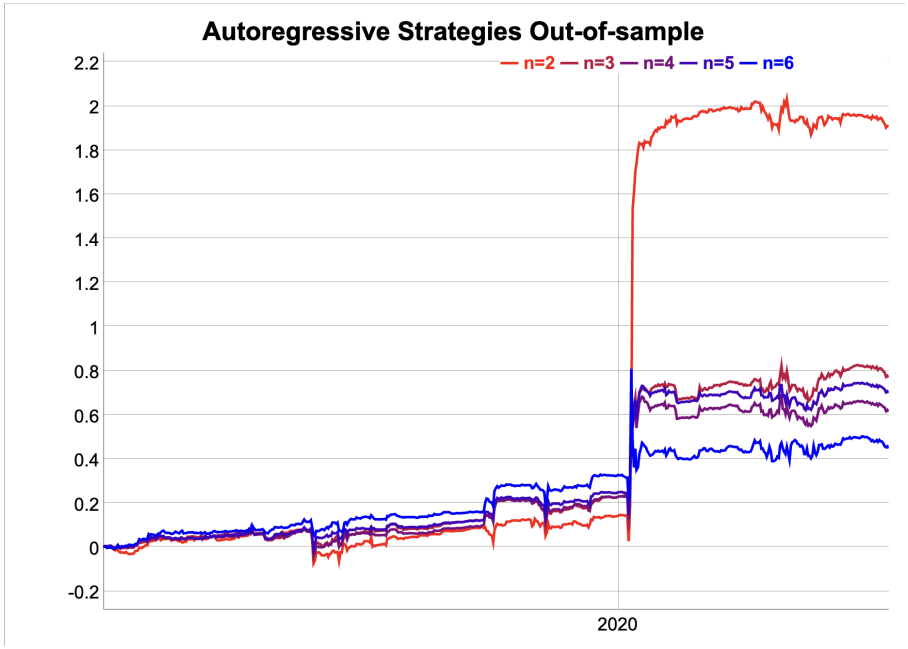
Out-of-Sample Autoregressive Strategy

The autoregressive strategy invests a dollar amount 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 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(fcast) {
+   pnls <- fcast*retp[outsample]
+   pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
+ })  # 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(cumsum(pnls)[endd],
+   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)
> coeffsd <- sqrt(diag(covmat))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coeffsd)
> # 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.

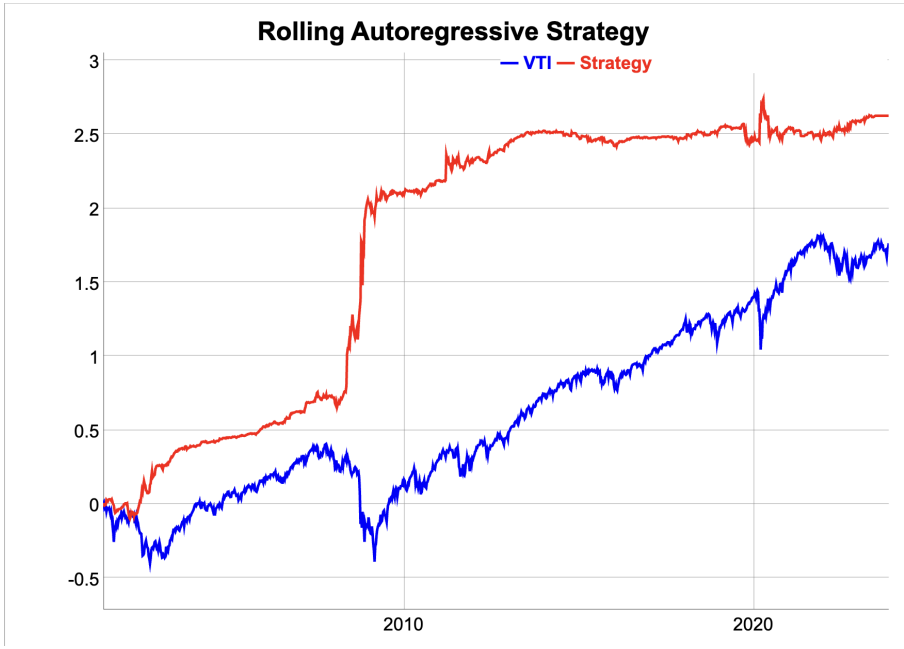
The *backtesting* procedure allows determining the optimal *meta-parameters* of the forecasting model: the order *n* of the *AR(n)* model and the length of look-back interval (`lookb`).

```
> # Perform rolling forecasting
> lookb <- 100
> fcasts <- sapply((lookb+1):nrows, function(tday) {
+   # 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)
+   coeffsd <- sqrt(diag(covmat))
+   coefft <- drop(coeff/coeffsd) # 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))
+   c(sd(resps), fcast=fcast, fstderr=sqrt(varf), coefft=coefft)
+ })  # end sapply
> # Coerce fcasts to a time series
> fcasts <- t(fcasts)
> ncols <- NCOL(fcasts)
> fcasts <- rbind(fcasts, matrix(numeric(ncols*lookb), nc=ncols))
> colnames(fcasts) <- c("volvti", "fcasts", "fstderr", colnames(pred
> fcasts <- xts::xts(fcasts, zoo::index(retp))
```

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.




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

draft: Forecasting Returns Using Rolling Autoregressive Models

Adopt this text to some other slides.

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 (`lookb`).

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) {
+     # Define the rolling look-back range
+     startp <- max(1, tday-lookb)
+     # Expanding look-back range
+     # startp <- 1
+     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))
```

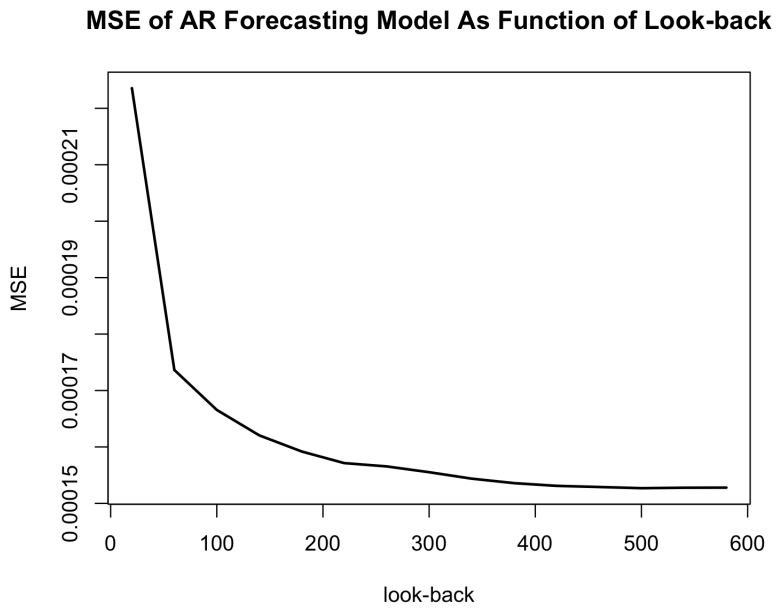
Forecasting Dependence On the Look-back Interval

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")
```

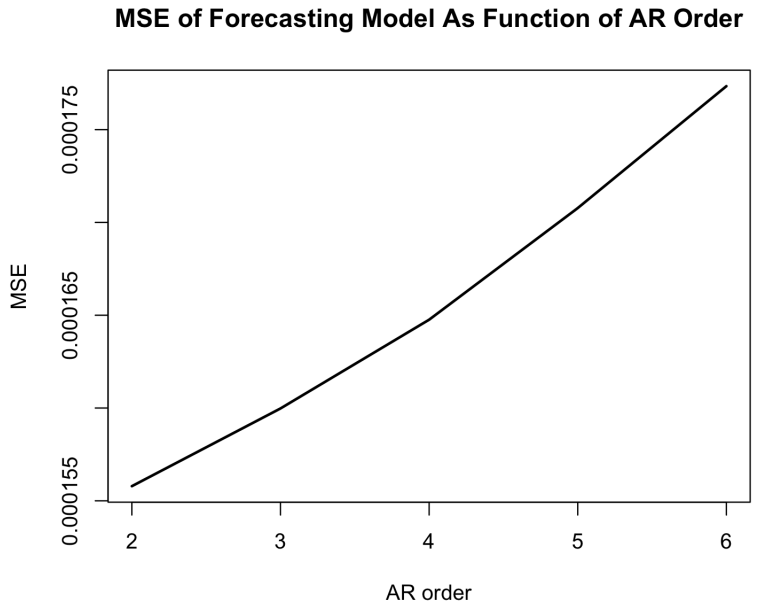
Order Dependence With Fixed Look-back

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)[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

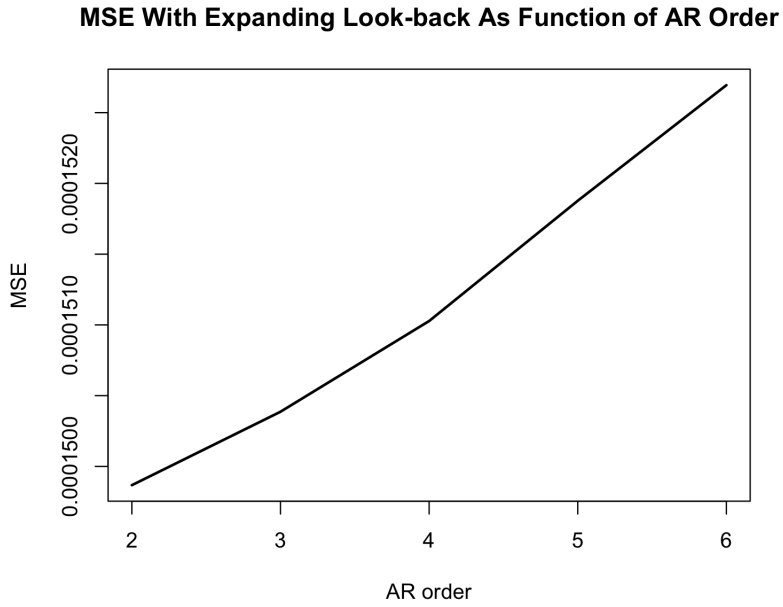
The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

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

Longer look-back intervals (`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)
```



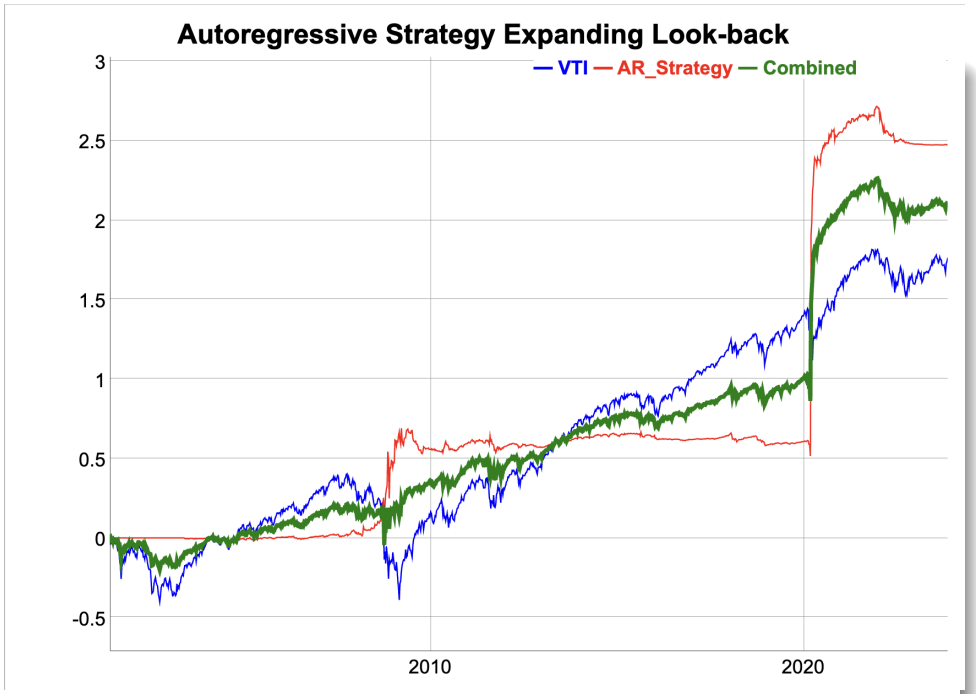

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, 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)[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: 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:

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

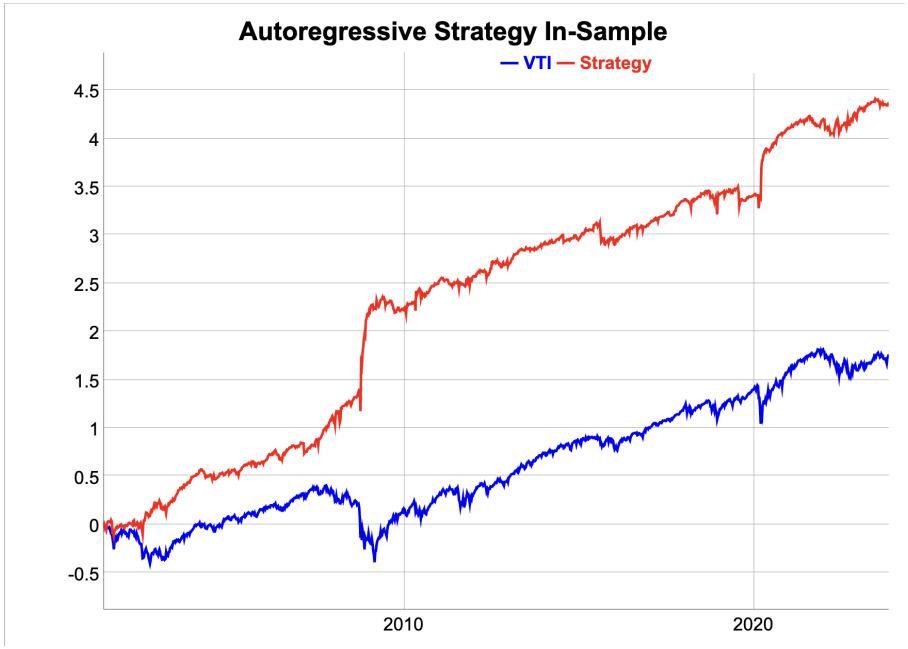
The *residuals* ε_t are assumed to be normally distributed, independent, and stationary.

```
> # 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("V
> 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)
```

Daytime and Overnight Stock Strategies

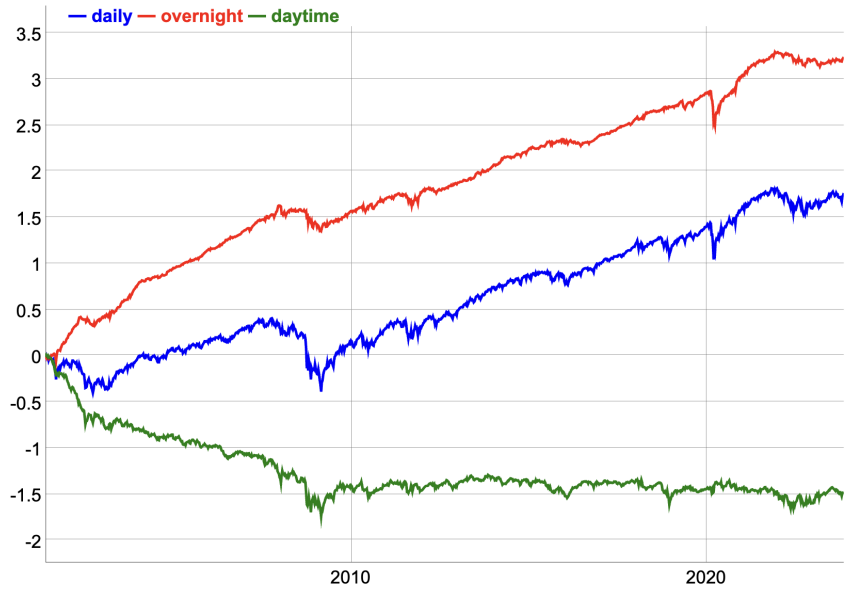
The overnight stock strategy consists of holding a long position only overnight (buying at the market close and selling at the open the next day).

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 log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   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.

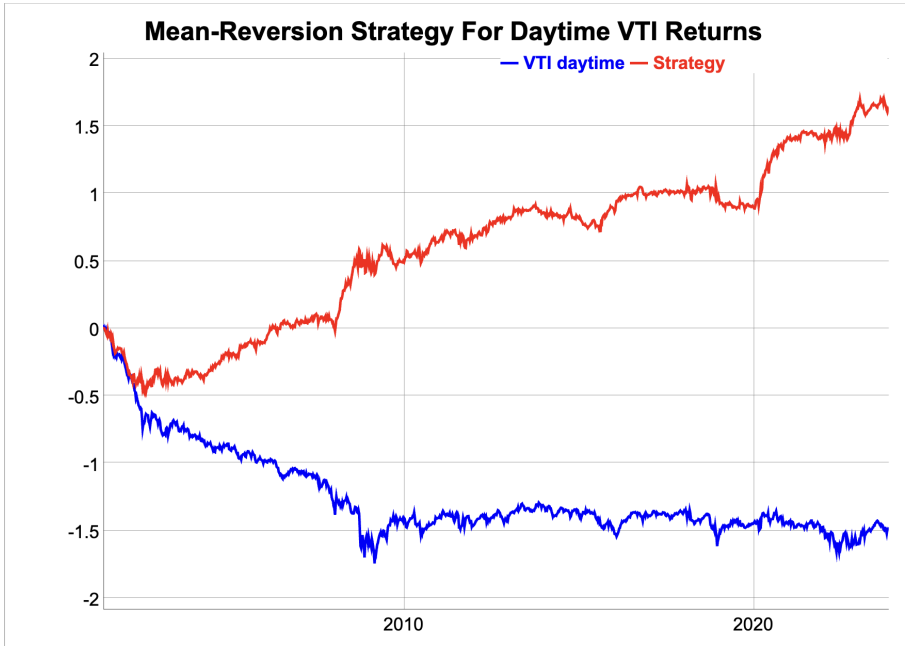
The *EMA* mean-reversion strategy holds stock positions equal to the *sign* of the trailing *EMA* of past returns.

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 # Bid-ask spread equal to 1 basis point
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
> # 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
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   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

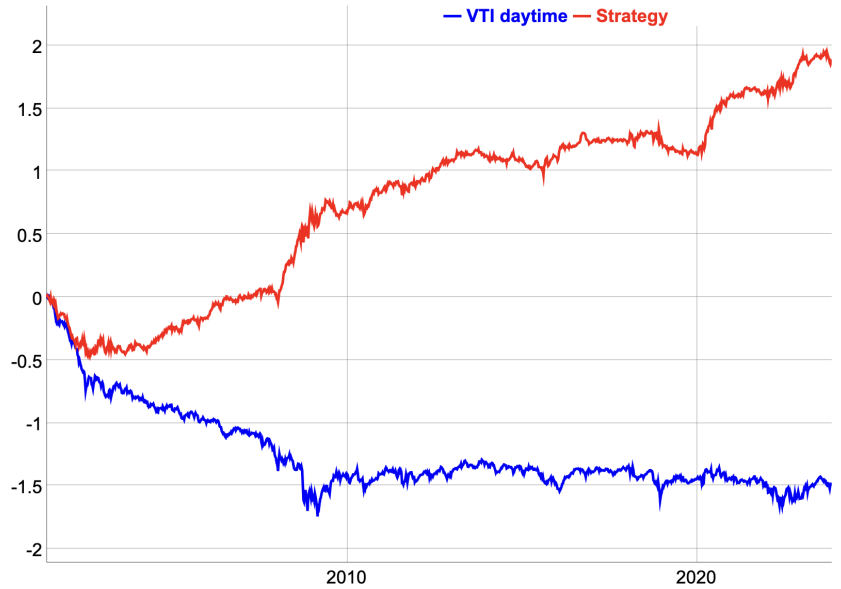
The Bollinger *z-score* for daytime returns z_t, is equal to the difference between the cumulative returns $p_t = \sum r_t$ minus their trailing mean $\bar{p}_t$, divided by their volatility σ_t:

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

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
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
> # 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)[endd],
+   main="Bollinger strategy For Daytime VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

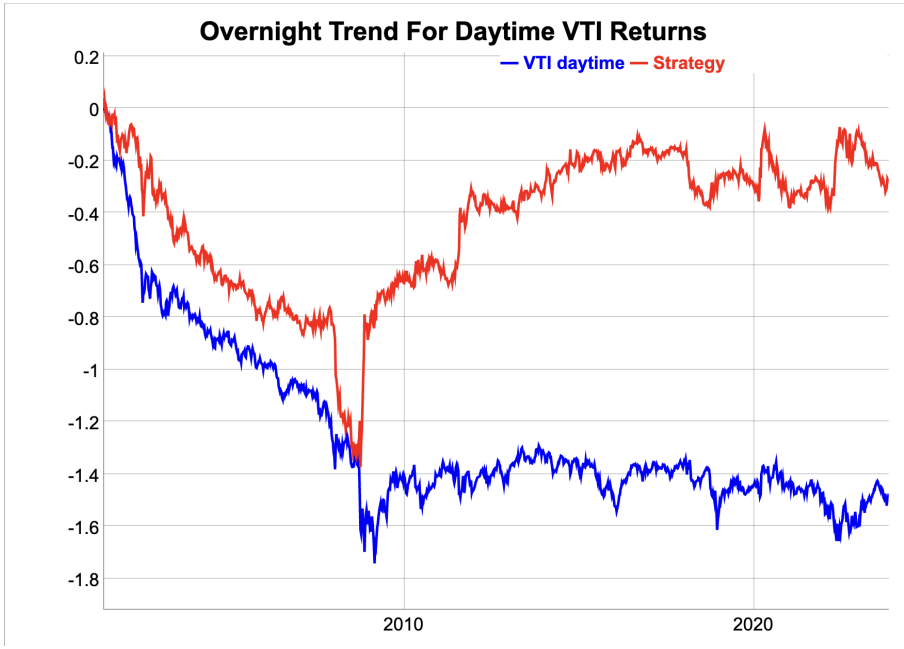
Overnight Trend Strategy

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
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
```

```
> # 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)[endd],
+ main="Overnight Trend For Daytime VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Bollinger Bands

The *Bollinger Bands* improve the moving average feature by adding information about the volatility.

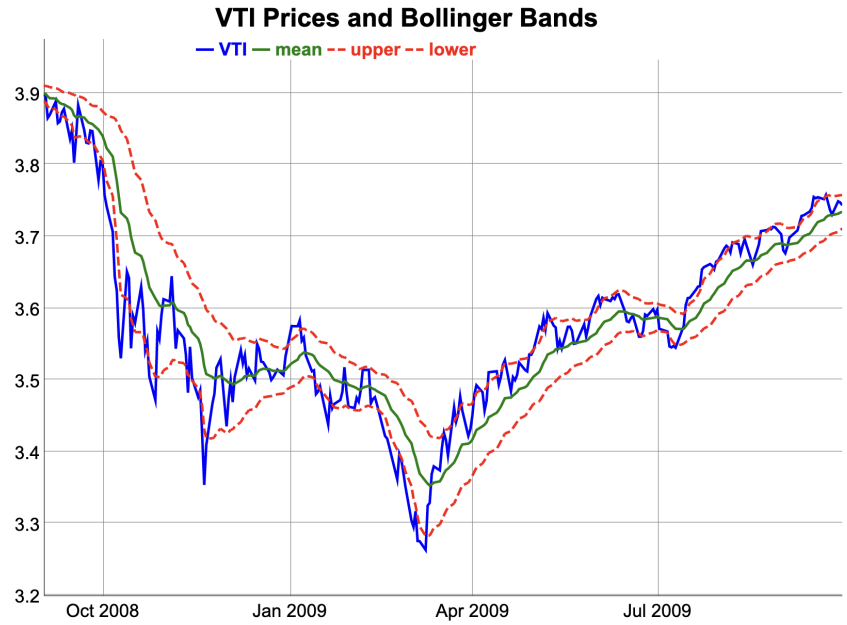
The Bollinger Bands are three time series, with the middle band equal to the trailing mean prices, the upper band equal to the mean prices plus the trailing standard deviation, and the lower band equal to the mean prices minus the standard deviation.

The Bollinger Bands are often used to indicate that prices are cheap if they are below the lower band, and rich (expensive) if they are above the upper band.

The decay factor λ determines the rate of decay of the weights, with smaller values of λ producing faster decay, giving more weight to recent prices, and vice versa.

The functions `HighFreq::run_mean()` and `HighFreq::run_var()` calculate the trailing mean and variance by recursively updating the past estimates with the new values, using the weight decay factor λ.

```
> # Extract the log VTI prices
> pricev <- log(na.omit(rutils::etfenv$prices$VTI))
> nrows <- NROW(pricev)
> # Calculate the trailing mean prices
> lambdaf <- 0.9
> pricem <- HighFreq::run_mean(pricev, lambda=lambdaf)
> # Calculate the trailing volatilities
> volv <- HighFreq::run_var(pricev, lambda=lambdaf)
> volv <- sqrt(volv)
```




```
> # Dygraphs plot of Bollinger bands
> priceb <- cbind(pricev, pricem, pricem+volv, pricem-volv)
> colnames(priceb)[2:4] <- c("mean", "upper", "lower")
> colnamev <- colnames(priceb)
> dygraphs::dygraph(priceb["2008-09/2009-09"], main="VTI Prices and
+   dySeries(name=colnamev[1], strokeWidth=2, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=2, col="green") %>%
+   dySeries(name=colnamev[3], strokeWidth=2, strokePattern="dashed"
+   dySeries(name=colnamev[4], strokeWidth=2, strokePattern="dashed"
+   dyLegend(show="always", width=200)
```

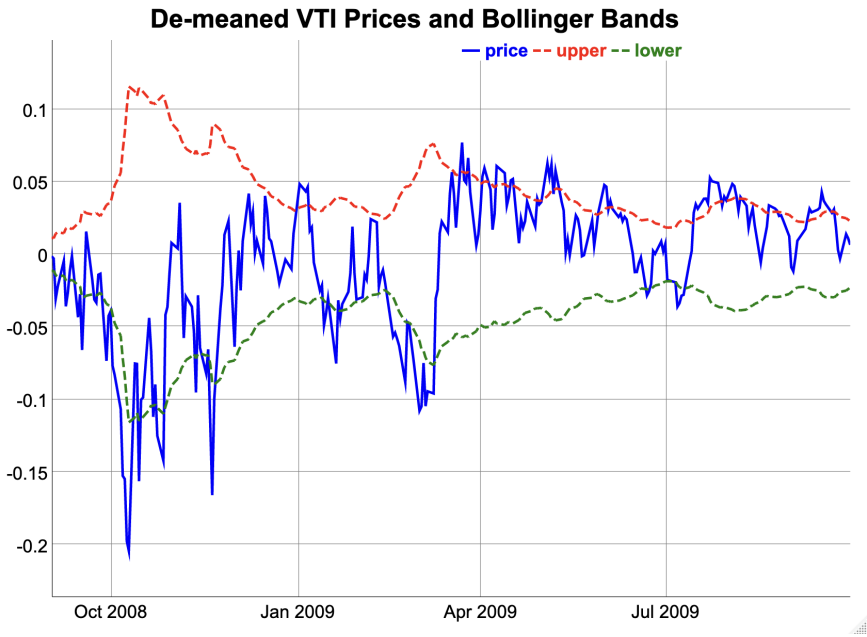
Bollinger Bands With Centered Prices

Centering (de-meaning) the prices provides a better view of the *Bollinger Bands*.

The centered prices tend to be range-bound between the *Bollinger Bands*.

When the centered price is below the lower band it's considered cheap, and if it's above the upper band it's considered rich (expensive).

When the centered price is close to zero it's considered fair (neutral).




```
> # Center the prices
> pricec <- pricev - pricem
> # Dygraphs plot of Bollinger bands
> priceb <- cbind(pricec, volv, -volv)
> colnames(priceb) <- c("price", "upper", "lower")
> colnamev <- colnames(priceb)
> dygraphs::dygraph(priceb["2008-09/2009-09"],
+   main="Centered VTI Prices and Bollinger Bands") %>%
+   dySeries(name=colnamev[1], strokeWidth=2, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=2, strokePattern="dashed"
+   dySeries(name=colnamev[3], strokeWidth=2, strokePattern="dashed"
+   dyLegend(show="always", width=200)
```

The Bollinger Band Strategy

The *Bollinger Band* strategy switches to long \$1 dollar of the stock when prices are cheap (below the lower band), and sells short -\$1 dollar of stock when prices are rich (expensive - above the upper band). It goes flat \$0 dollar of stock (unwinds) if the stock reaches a fair (mean) price.

The strategy is therefore always either long \$1 dollar of stock, or short -\$1 dollar of stock, or flat \$0 dollar of stock.

The upper and lower Bollinger bands can be chosen to be a multiple of the standard deviations above and below the mean prices.

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

The Bollinger strategy is a type of *statistical arbitrage* strategy.

Statistical arbitrage strategies try to exploit short-term anomalies in prices, when prices diverge from their equilibrium values and then revert back to them.

```
> # Calculate the trailing mean prices and volatilities
> lambdaf <- 0.1
> pricem <- HighFreq::run_mean(pricev, lambda=lambdaf)
> volv <- HighFreq::run_var(pricev, lambda=lambdaf)
> volv <- sqrt(volv)
> # Prepare the simulation parameters
> pricen <- as.numeric(pricev) # Numeric price
> pricec <- pricen - pricem # Centered price
> threshv <- volv
> posv <- integer(nrows) # Stock positions
> posv[1] <- 0 # Initial position
> # Calculate the positions from Bollinger bands
> for (it in 2:nrows) {
+   if (pricec[it-1] > threshv[it-1]) {
+     # Enter short
+     posv[it] <- (-1)
+   } else if (pricec[it-1] < (-threshv[it-1])) {
+     # Enter long
+     posv[it] <- 1
+   } else if ((posv[it-1] < 0) && (pricec[it-1] < 0)) {
+     # Unwind short
+     posv[it] <- 0
+   } else if ((posv[it-1] > 0) && (pricec[it-1] > 0)) {
+     # Unwind long
+     posv[it] <- 0
+   } else {
+     # Do nothing
+     posv[it] <- posv[it-1]
+   }  # end if
+ }  # end for
> # Calculate the number of trades
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
> # Calculate the pnls
> retp <- rutils::diffit(pricev)
> pnls <- retp*posv
```

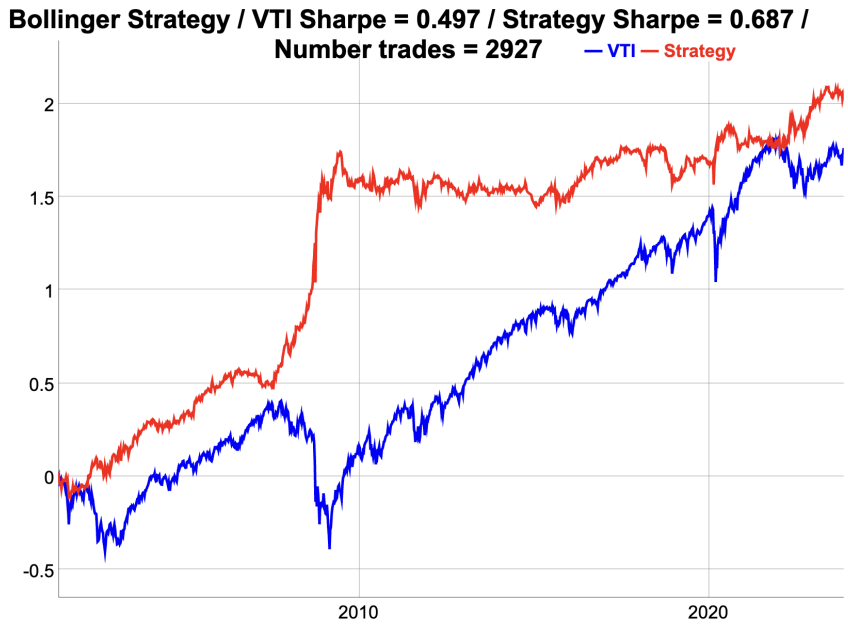
Bollinger Strategy Performance

The *Bollinger Band* strategy has two parameters: the weight decay factor λ and the standard deviation multiple n.

The best strategy parameters can be found using backtest simulation, but it risks overfitting the parameters to the in-sample data, and poor performance out-of-sample.

The *Bollinger Band* strategy has performed well for *VTI* with $\lambda = 0.1$, but it hasn't performed well for most other stocks.

The *Bollinger Band* strategy had its best performance for *VTI* prior to the financial crisis of 2008-2009.



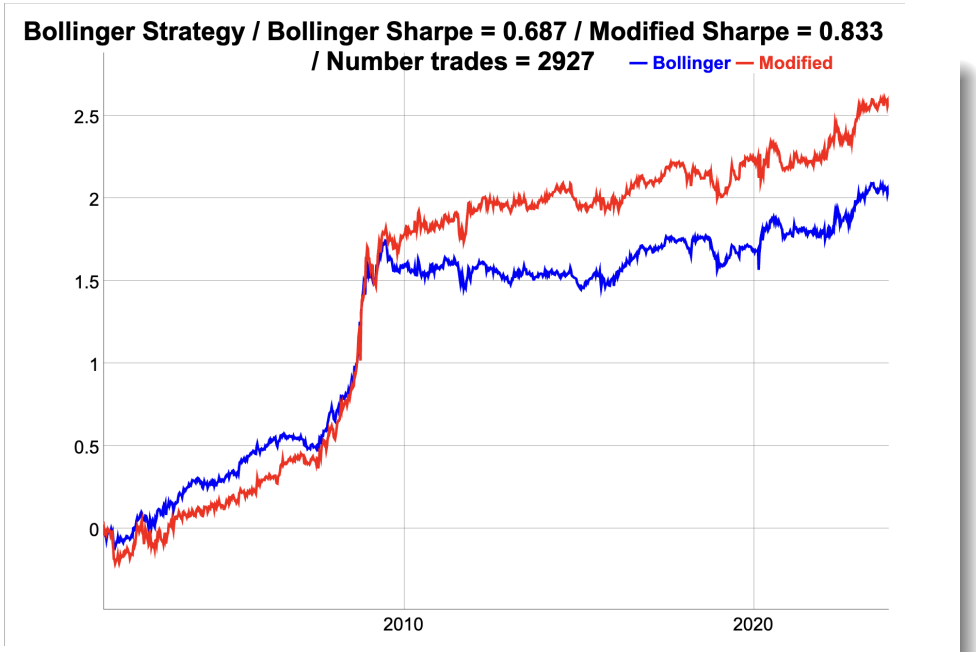

```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]
> sharper <- round(sharper, 3)
> # Dygraphs plot of Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Bollinger Strategy", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=" / "
+   "Number trades =", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

The Modified Bollinger Strategy

Unwinding the position when the stock reaches a fair (mean) price doesn't necessarily produce better performance.

In the modified Bollinger Strategy the positions are held until the price reaches the opposite extreme, so that the strategy is always either long \$1 dollar of stock or short \$1 dollar of stock.

```
> # Simulate the modified Bollinger strategy
> posv <- integer(nrows) # Stock positions
> posv[1] <- 0 # Initial position
> for (it in 2:nrows) {
+   if (pricec[it-1] > threshv[it-1]) {
+     # Enter short
+     posv[it] <- (-1)
+   } else if (pricec[it-1] < (-threshv[it-1])) {
+     # Enter long
+     posv[it] <- 1
+   } else {
+     # Do nothing
+     posv[it] <- posv[it-1]
+   }  # end if
+ }  # end for
> # Calculate the PnLs
> pnls2 <- retp*posv
```




```
> # Calculate the Sharpe ratios
> wealthv <- cbind(pnls, pnls2)
> colnames(wealthv) <- c("Bollinger", "Modified")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]
> sharper <- round(sharper, 3)
> # Dygraphs plot of Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Bollinger Strategy", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=" / "
+   "Number trades =", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Fast Bollinger Strategy Simulation

The Bollinger Strategy is path-dependent so simulating it requires performing a loop, which can be slow in R.

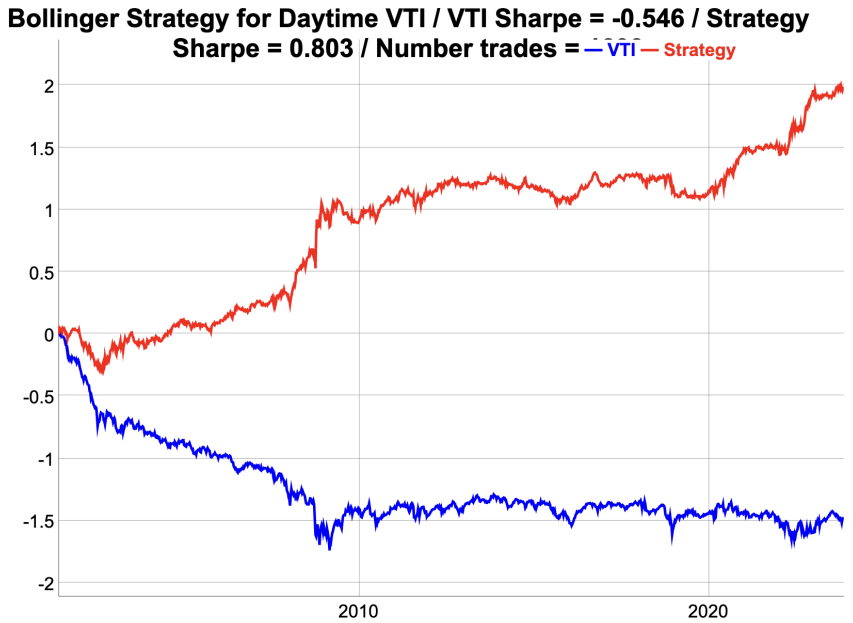
The modified Bollinger Strategy can be simulated quickly using the compiled C++ functions `ifelse()` and `zoo::na.locf()`.

```
> # Simulate the modified Bollinger strategy quickly
> posf <- rep(NA_integer_, nrows)
> posf[1] <- 0
> posf <- ifelse(pricec > threshv, -1, posf)
> posf <- ifelse(pricec < -threshv, 1, posf)
> posf <- zoo::na.locf(posf)
> # Lag the positions to trade in the next period
> posf <- rutils::lagit(posf, lagg=1)
> # Compare the positions
> all.equal(posv, posf)
```

Bollinger Strategy For Daytime VTI Returns

The Bollinger Strategy has performed well for daytime returns of the *VTI* ETF, because daytime returns exhibit significant mean-reversion.

This simulation doesn't account for transaction costs, which would likely erase all profits if market orders were used for trade executions. But the strategy could be profitable if limit orders were used for trade executions.




```
> # Calculate the daytime open-to-close VTI returns
> ohlc <- log(rutils::etfenv$VTI)
> nrows <- NROW(ohlc)
> openp <- quantmod::Op(ohlc)
> highp <- quantmod::Hi(ohlc)
> lowp <- quantmod::Lo(ohlc)
> closep <- quantmod::Cl(ohlc)
> retd <- (closep - openp)
> # Calculate the cumulative daytime VTI returns
> priced <- cumsum(retd)
> lambdaf <- 0.1
> pricem <- HighFreq::run_mean(priced, lambda=lambdaf)
> volv <- HighFreq::run_var(priced, lambda=lambdaf)
> volv <- sqrt(volv)
> # Calculate the positions from Bollinger bands
> threshv <- volv
> pricec <- zoo::coredata(priced - pricem)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(pricec > threshv, -1, posv)
> posv <- ifelse(pricec < -threshv, 1, posv)
> posv <- zoo::na.locf(posv)
> # Lag the positions to trade in the next period
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the number of trades and the PnLs
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
> pnls <- retd*posv
```

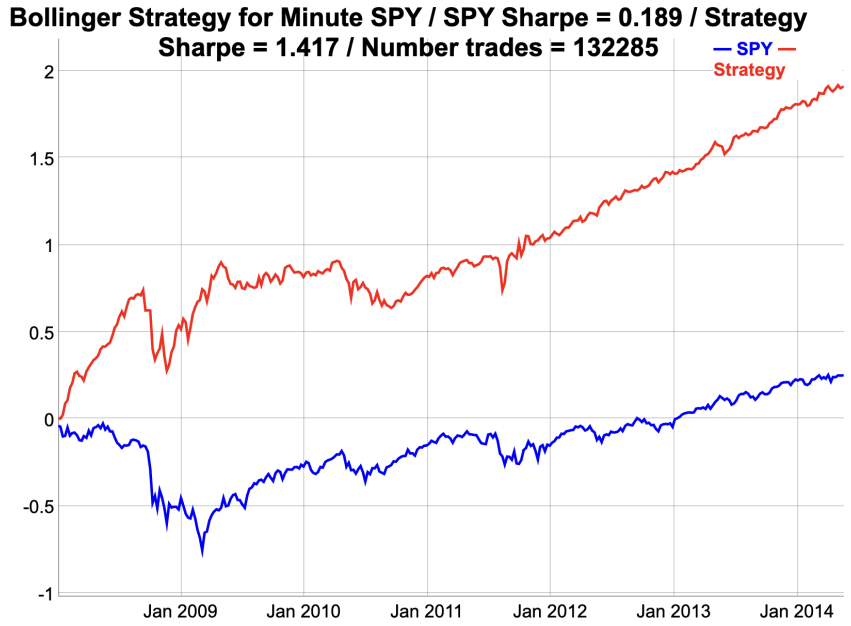
```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> nyears <- as.numeric(end(priced)-start(priced))/365
> sharper <- sqrt(nrows/nyears)*sapply(wealthv, function(x) mean(x)/
> sharper <- round(sharper, 3)
> # Dygraphs plot of Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Bollinger Strategy for Daytime VTI", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=" / "
+   "Number trades =", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Bollinger Strategy For Intraday SPY Returns

The Bollinger Strategy has performed well for 1-minute prices of the *SPY* ETF, because intraday returns exhibit significant mean-reversion.

This simulation doesn't account for transaction costs, which would likely erase all profits if market orders were used for trade executions. But the strategy could be profitable if limit orders were used for trade executions.

```
> # Calculate the trailing mean prices and volatilities of SPY
> pricev <- log(quantmod::Cl(HighFreq::SPY))
> nrows <- NROW(pricev)
> lambdaf <- 0.1
> pricem <- HighFreq::run_mean(pricev, lambda=lambdaf)
> volv <- HighFreq::run_var(pricev, lambda=lambdaf)
> volv <- sqrt(volv)
> # Calculate the positions from Bollinger bands
> threshv <- volv
> pricec <- zoo::coredata(pricev - pricem)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(pricec > threshv, -1, posv)
> posv <- ifelse(pricec < -threshv, 1, posv)
> posv <- zoo::na.locf(posv)
> # Lag the positions to trade in the next period
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the number of trades and the PnLs
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
> retp <- rutils::diffit(pricev)
> pnls <- retp*posv
> # Subtract transaction costs from the pnls
> bidask <- 0.0001 # Bid-ask spread equal to 1 basis point
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
```




```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("SPY", "Strategy")
> nyears <- as.numeric(end(pricev)-start(pricev))/365
> sharper <- sqrt(nrows/nyears)*sapply(wealthv, function(x) mean(x)/
> sharper <- round(sharper, 3)
> # Dygraphs plot of Bollinger strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Bollinger Strategy for Minute SPY", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=" / "
+   "Number trades =", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=100)
```

The Hampel Filter Bands

The Bollinger bands can be improved by using nonparametric measures of location (*median*) and dispersion (*MAD*).

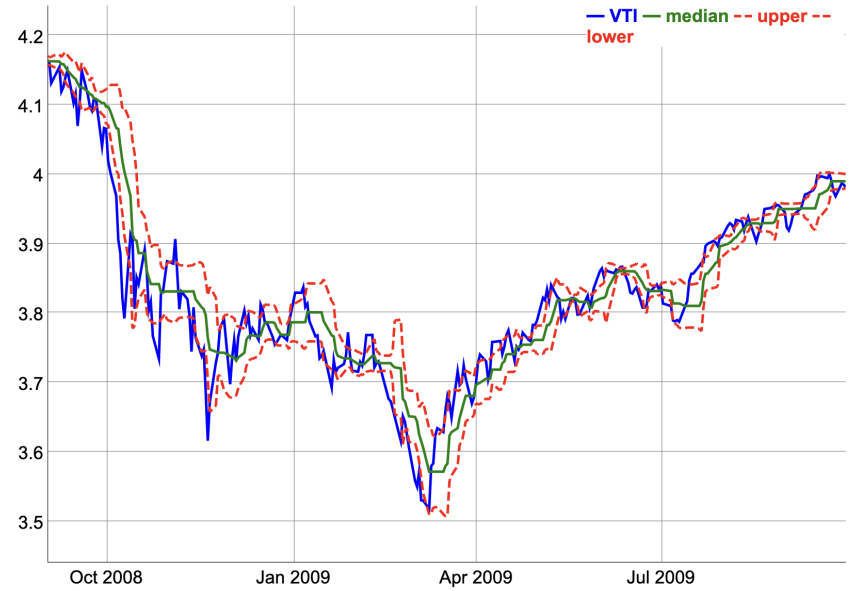
The *Median Absolute Deviation* (*MAD*) is a nonparametric measure of dispersion (variability):

$$\text{MAD} = \text{median}(\text{abs}(p_t - \text{median}(\mathbf{p})))$$

The *Hampel z-score* is equal to the deviation from the median divided by the *MAD*:

$$z_i = \frac{p_t - \text{median}(\mathbf{p})}{\text{MAD}}$$

A time series of *z-scores* over past data can be calculated using a trailing look-back window.




```
> # Extract time series of VTI log prices
> pricev <- log(na.omit(rutils::etfenv$prices$VTI))
> nrows <- NROW(pricev)
> # Define look-back window
> lookb <- 11
> # Calculate time series of trailing medians
> medianv <- HighFreq::roll_mean(pricev, lookb, method="nonparametr
> # medianv <- TTR::runMedian(pricev, n=lookb)
> # Calculate time series of MAD
> madv <- HighFreq::roll_var(pricev, lookb=lookb, method="nonparam
> # madv <- TTR::runMAD(pricev, n=lookb)
> # Calculate time series of z-scores
> zscores <- ifelse(madv > 0, (pricev - medianv)/madv, 0)
> zscores[1:lookb, ] <- 0
> tail(zscores, lookb)
> range(zscores)
```

```
> # Plot histogram of z-scores
> histp <- hist(zscores, col="lightgrey",
+   xlab="z-scores", breaks=50, xlim=c(-4, 4),
+   ylab="frequency", freq=FALSE, main="Hampel Z-Scores histogram")
> lines(density(zscores, adjust=1.5), lwd=3, col="blue")
> # Dygraphs plot of Hampel bands
> priceb <- cbind(pricev, medianv, medianv+madv, medianv-madv)
> colnames(priceb)[2:4] <- c("median", "upper", "lower")
> colnamev <- colnames(priceb)
> dygraphs::dygraph(priceb["2008-09/2009-09"], main="VTI Prices and
+   dySeries(name=colnamev[1], strokeWidth=2, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=2, col="green") %>%
+   dySeries(name=colnamev[3], strokeWidth=2, strokePattern="dashed"
+   dySeries(name=colnamev[4], strokeWidth=2, strokePattern="dashed"
+   dyLegend(show="always", width=200)
```

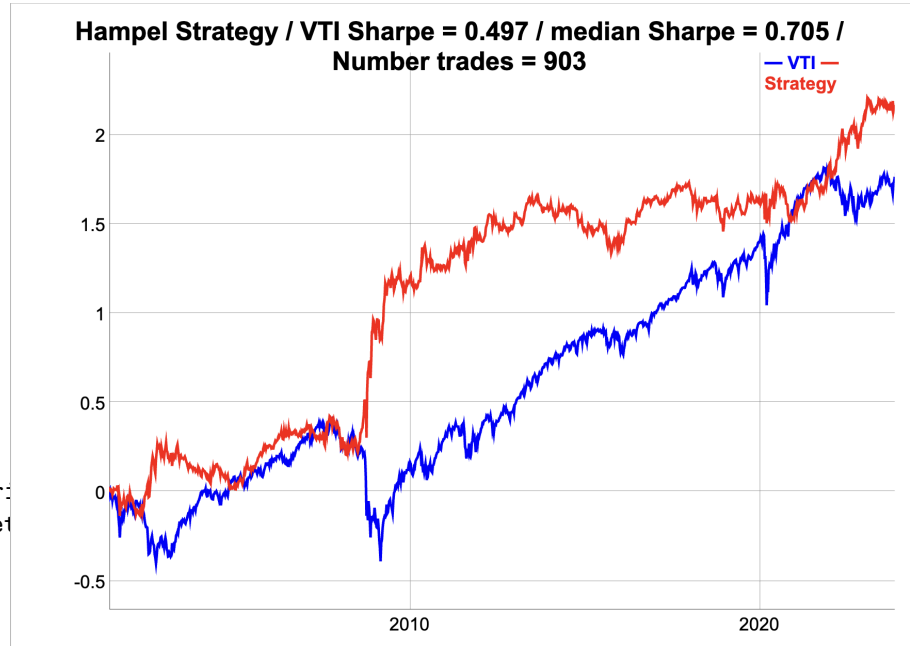
Hampel Filter Strategy

The Hampel filter strategy is a contrarian strategy that uses Hampel z-scores to establish long and short positions.

The Hampel strategy has two meta-parameters: the look-back interval and the threshold level.

The best choice of the meta-parameters can be determined through simulation.

```
> # Calculate the time series of trailing medians and MAD
> lookb <- 3
> medianv <- HighFreq::roll_mean(pricev, lookb, method="nonparametri
> madv <- HighFreq::roll_var(pricev, lookb=lookb, method="nonparamet
> # Calculate the time series of z-scores
> zscores <- ifelse(madv > 0, (pricev - medianv)/madv, 0)
> zscores[1:lookb, ] <- 0
> range(zscores)
> # Calculate the positions
> threshv <- 1
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv[zscores > threshv] <- (-1)
> posv[zscores < -threshv] <- 1
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv)
> # Calculate the number of trades and the PnLs
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
> retp <- rutils::diffit(pricev)
> pnls <- retp*posv
```



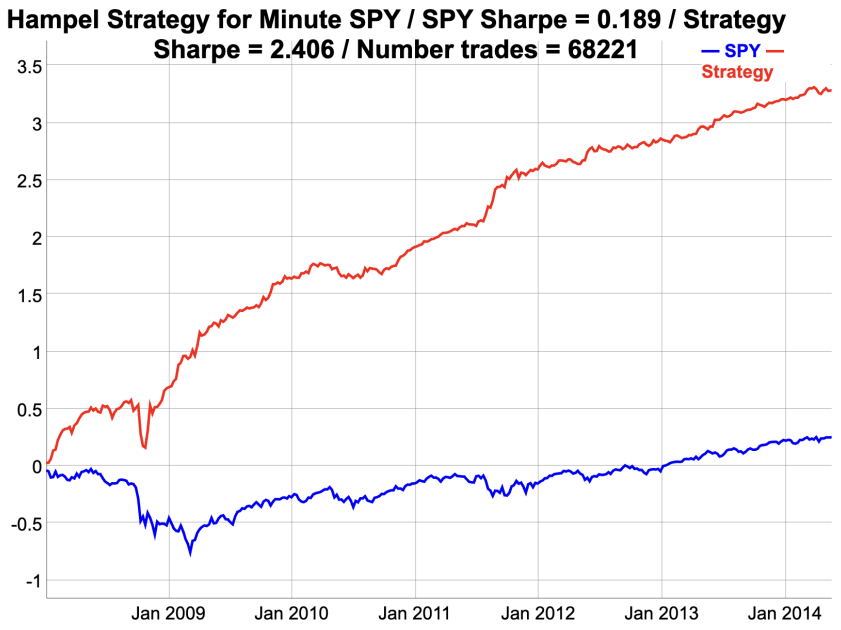

```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]
> sharper <- round(sharper, 3)
> # Dygraphs plot of Hampel strategy
> captiont <- paste("Hampel Strategy", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=" / "
+   "Number trades =", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=100)
```

Hampel Filter Strategy For Intraday SPY Returns

The Hampel Filter strategy has performed well for 1-minute prices of the *SPY* ETF, because intraday returns exhibit significant mean-reversion.

This simulation doesn't account for transaction costs, which would likely erase all profits if market orders were used for trade executions. But the strategy could be profitable if limit orders were used for trade executions.

```
> # Calculate the trailing mean prices and volatilities of SPY
> pricev <- log(quantmod::Cl(HighFreq::SPY))
> nrows <- NROW(pricev)
> # Calculate the price medians and MAD
> lookb <- 3
> medianv <- HighFreq::roll_mean(pricev, lookb, method="nonparametri
> madv <- HighFreq::roll_var(pricev, lookb=lookb, method="nonparamet
> # Calculate the time series of z-scores
> zscores <- ifelse(madv > 0, (pricev - medianv)/madv, 0)
> zscores[1:lookb, ] <- 0
> # Calculate the positions
> threshv <- 1
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv[zscores < -threshv] <- 1
> posv[zscores > threshv] <- (-1)
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv)
> # Calculate the number of trades and the PnLs
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
> retp <- rutils::diffit(pricev)
> pnls <- retp*posv
> # Subtract transaction costs from the pnls
> costs <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costs)
```




```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("SPY", "Strategy")
> nyears <- as.numeric(end(pricev)-start(pricev))/365
> sharper <- sqrt(nrows/nyears)*sapply(wealthv, function(x) mean(x)/
> sharper <- round(sharper, 3)
> # Dygraphs plot of Hampel strategy
> colnamev <- colnames(wealthv)
> captiont <- paste("Hampel Strategy for Minute SPY", "/ \n",
+   paste0(paste(colnamev[1:2], "Sharpe =", sharper), collapse=" / "
+   "Number trades =", ntrades)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=100)
```

Homework Assignment

Required

- Study all the lecture slides in *FRE7241_Lecture_4.pdf*, and run all the code in *FRE7241_Lecture_4.R*

Recommended

- Download from NYU Classes and read about momentum strategies:
  Moskowitz Time Series Momentum.pdf
  Bouchaud Momentum Mean Reversion Equity Returns.pdf
  Hurst Pedersen AQR Momentum Evidence.pdf