FRE7241 Algorithmic Portfolio Management

Lecture#6, Spring 2023

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

May 2, 2023

NYU | TANDON SCHOOL OF ENGINEERING

Portfolio Weight Constraints

Constraints on the portfolio weights are applied to satisfy investment objectives and risk limits.

Let w_i be the portfolio weights produced by a model, which may not satisfy the constraints, so they must be transformed into new weights: w_i'.

For example, the weights can be centered so their sum is equal to 0: $\sum_{i=1}^{n} w_i' = 0$, by shifting them by their mean value:

$$w_i' = w_i - \frac{1}{n}\sum_{i=1}^{n} w_i$$

The advantage of centering is that it produces portfolios that are more risk neutral - less long or short risk.

The disadvantage is that it shifts the mean of the weights, and it allows highly leveraged portfolios, with very large positive and negative weights.

```
> # Load daily S&P500 percentage stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> # Overwrite NA values in returns
> retp <- returns100
> datev <- zoo::index(retp) # dates
> nrows <- NROW(retp) # number of rows
> nstocks <- NCOL(retp) # number of stocks
> # Objective function equal to Kelly ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   varv <- var(retp)
+   if (varv > 0) mean(retp)/varv else 0
+ }  # end objfun
> # Calculate performance statistics for all stocks
> perfstat <- sapply(retp, objfun)
> sum(is.na(perfstat))
> sum(!is.finite(perfstat))
> # Calculate weights proportional to performance statistic
> weightm <- perfstat
> hist(weightm)
> # Center the weights
> weightm <- weightm - mean(weightm)
> sum(weightm)
> sort(weightm)
```

Quadratic Weight Constraint

Another way of satisfying the constraints is by scaling (multiplying) the weights by a factor.

Under the *quadratic* constraint, the sum of the *squared* weights is equal to 1: $\sum_{i=1}^{n} w_i'^2 = 1$, after they are scaled:

$$w_i' = \frac{w_i}{\sqrt{\sum_{i=1}^{n} w_i^2}}$$

Scaling the weights modifies the portfolio *leverage* (the ratio of the portfolio risk divided by the capital), while maintaining the relative weights.

The disadvantage of the *quadratic* constraint is that it can produce portfolios with very low leverage.

```
> # Quadratic constraint
> weightv <- weightm/sqrt(sum(weightm^2))
> sum(weightv^2)
> sum(weightv)
> weightv
```

Linear Weight Constraint

A widely used constraint is setting the sum of the weights equal to 1: $\sum_{i=1}^{n} w_i' = 1$, by dividing them by their sum:

$$w_i' = \frac{w_i}{\sum_{i=1}^{n} w_i}$$

The *linear* constraint is equivalent to distributing a unit of capital among a stock portfolio.

The disadvantage of the *linear* constraint is that it has a long risk bias. When the sum of the weights is negative, it switches their sign to positive.

```
> # Apply the linear constraint
> weightv <- weightm/sum(weightm)
> sum(weightv)
> weightv
```

Volatility Weight Constraint

The weights can be scaled to satisfy a volatility target.

For example, they can be scaled so that the in-sample portfolio volatility σ is the same as the volatility of the equal weight portfolio σ_{ew}:

$$w_i' = \frac{\sigma_{ew}}{\sigma} w_i$$

This produces portfolios with a leverage corresponding to the current market volatility.

Or the weights can be scaled so that the in-sample portfolio volatility σ is equal to a target volatility σ_t:

$$w_i' = \frac{\sigma_t}{\sigma} w_i$$

This produces portfolios with a volatility close to the target, irrespective of the market volatility.

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.

The function `HighFreq::mult_mat()` multiplies the rows or columns of a *matrix* times a *vector*, element-wise.

```
> # Calculate the weighted returns using transpose
> retw <- t(t(retp)*weightm)
> # Or using Rcpp
> retf <- HighFreq::mult_mat(weightm, retp)
> all.equal(retw, retf, check.attributes=FALSE)
> # Calculate the in-sample portfolio volatility
> volis <- sd(rowMeans(retw, na.rm=TRUE))
> # Calculate the equal weight portfolio volatility
> volew <- sd(rowMeans(retp, na.rm=TRUE))
> # Apply the volatility constraint
> weightv <- volew*weightm/volis
> # Calculate the in-sample portfolio volatility
> retw <- t(t(retp)*weightv)
> all.equal(sd(rowMeans(retw, na.rm=TRUE)), volew)
> # Apply the volatility target constraint
> volt <- 1e-2
> weightv <- volt*weightm/volis
> retw <- t(t(retp)*weightv)
> all.equal(sd(rowMeans(retw, na.rm=TRUE)), volt)
> # Compare speed of R with Rcpp
> summary(microbenchmark(
+   trans=t(t(retp)*weightm),
+   rcpp=HighFreq::mult_mat(weightm, retp),
+   times=10))[, c(1, 4, 5)]
```

Box Constraints

Box constraints limit the individual weights, for example: $0 \leq w_i \leq 1$.

Box constraints are often applied when constructing long-only portfolios, or when limiting the exposure to certain stocks.

```
> # Box constraints
> weightv[weightv > 1] <- 1
> weightv[weightv < 0] <- 0
> weightv
```

Stock Performance Measures

Various performance measures can be used to select stocks.

```
> # Objective function equal to sum of returns
> objfun <- function(retp) sum(na.omit(retp))
> # Objective function equal to Sharpe ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   stdev <- sd(retp)
+   if (stdev > 0) mean(retp)/stdev else 0
+ }  # end objfun
> # Objective function equal to Kelly ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   varv <- var(retp)
+   if (varv > 0) mean(retp)/varv else 0
+ }  # end objfun
```

Momentum Portfolio Weights

The portfolio weights of *momentum* strategies can be calculated based on the past performance of the assets in many different ways:

- Invest equal dollar amounts in the top `n` best performing stocks and short the `n` worst performing stocks,
- Invest dollar amounts proportional to the past performance - purchase stocks with positive performance, and short stocks with negative performance,
- Apply the weight constraints.

The *momentum* weights can then be applied in the out-of-sample interval.

```
> # Calculate the performance statistics for all stocks
> perfstat <- sapply(retp, objfun)
> sum(is.na(perfstat))
> # Calculate the best and worst performing stocks
> perfstat <- sort(perfstat, decreasing=TRUE)
> topstocks <- 10
> symbolb <- names(head(perfstat, topstocks))
> symbolw <- names(tail(perfstat, topstocks))
> # Calculate equal weights for the best and worst performing stocks
> weightv <- numeric(NCOL(retp))
> names(weightv) <- colnames(retp)
> weightv[symbolb] <- 1
> weightv[symbolw] <- (-1)
> # Calculate weights proportional to performance statistic
> weightv <- perfstat
> # Center weights so sum is equal to 0
> weightv <- weightv - mean(weightv)
> # Scale weights so sum of squares is equal to 1
> weightv <- weightv/sqrt(sum(weightv^2))
> # Calculate the in-sample momentum strategy pnls
> pnls <- t(t(retp)*weightv)
> # Or using Rcpp
> pnls2 <- HighFreq::mult_mat(weightv, retp)
> all.equal(pnls, pnls2, check.attributes=FALSE)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> # Scale weights so in-sample pnl volatility is same as equal weigh
> indeks <- rowMeans(retp, na.rm=TRUE)
> scalev <- sd(indeks)/sd(pnls)
> weightv <- scalev*weightv
> # Calculate the momentum strategy pnls
> pnls <- t(t(retp)*weightv)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> all.equal(sd(indeks), sd(pnls))
```

Rolling Momentum Strategy

In a *rolling momentum strategy*, the portfolio is rebalanced periodically and held out-of-sample.

Momentum strategies can be *backtested* by specifying the portfolio rebalancing frequency, the formation interval, and the holding period:

- Specify a portfolio of stocks and their returns,
- Calculate the *end points* for portfolio rebalancing,
- Define an objective function for calculating the past performance of the stocks,
- Calculate the past performance over the *look-back* formation intervals,
- Calculate the portfolio weights from the past (in-sample) performance,
- Calculate the out-of-sample momentum strategy returns by applying the portfolio weights to the future returns,
- Apply a volatility scaling factor to the out-of-sample returns,
- Calculate the transaction costs and subtract them from the strategy returns.

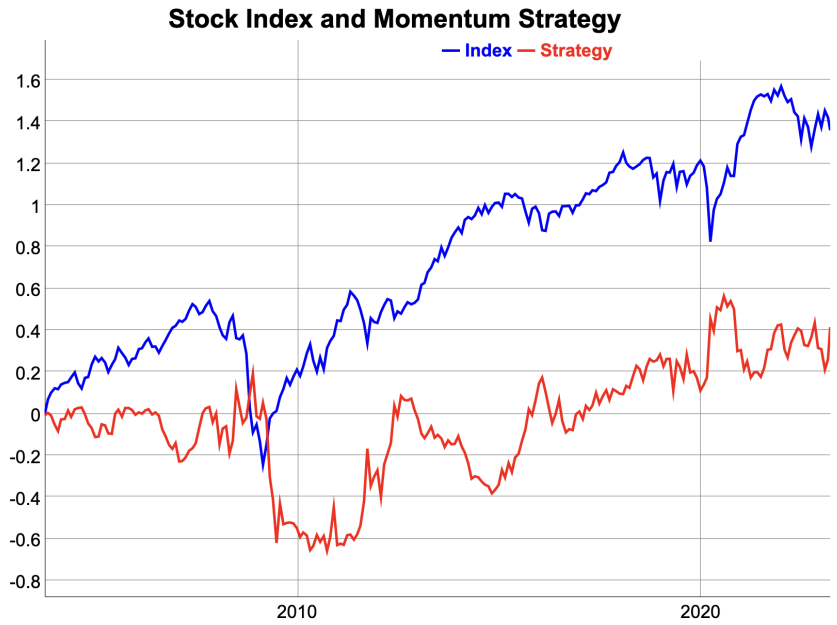
```
> # Objective function equal to Kelly ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   if (NROW(retp) > 2) {
+     varv <- var(retp)
+     if (varv > 0) mean(retp)/varv else 0
+   }
+   else 0
+ }  # end objfun
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> npts <- NROW(endd)
> # Perform loop over the end points
> look_back <- 8
> pnls <- lapply(2:(npts-1), function(ep) {
+   # Select the look-back returns
+   startp <- endd[max(1, ep-look_back)]
+   retis <- retp[startp:endd[ep], ]
+   # Calculate the best and worst performing stocks in-sample
+   perfstat <- sapply(retis, objfun)
+   perfstat <- sort(perfstat, decreasing=TRUE)
+   symbolb <- names(head(perfstat, topstocks))
+   symbolw <- names(tail(perfstat, topstocks))
+   # Calculate the momentum weights
+   weightv <- numeric(NCOL(retp))
+   names(weightv) <- colnames(retp)
+   weightv[symbolb] <- 1
+   weightv[symbolw] <- (-1)
+   # Calculate the in-sample momentum pnls
+   pnls <- HighFreq::mult_mat(weightv, retis)
+   pnls <- rowMeans(pnls, na.rm=TRUE)
+   # Scale weights so in-sample pnl volatility is same as equal wei
+   weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnls)
+   # Calculate the out-of-sample momentum returns
+   pnls <- HighFreq::mult_mat(weightv, retp[(endd[ep]+1):endd[ep+1]
+   pnls <- rowMeans(pnls, na.rm=TRUE)
+   drop(pnls)
+ })  # end lapply
```

Performance of Stock Momentum Strategy

The initial stock momentum strategy underperforms the index because of a poor choice of the model parameters.

The momentum strategy may be improved by a better choice of the model parameters: the length of look-back interval and the number of stocks.

```
> # Calculate the average of all stock returns
> indeks <- rowMeans(retp, na.rm=TRUE)
> indeks <- xts::xts(indeks, order.by=datev)
> colnames(indeks) <- "Index"
> # Add initial startup interval to the momentum returns
> pnls <- c(rowMeans(retp[endd[1]:endd[2], ], na.rm=TRUE), pnls)
> pnls <- xts::xts(pnls, order.by=datev)
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot dygraph of stock index and momentum strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Stock Index and Momentum Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Momentum Strategy Functional

Performing a *backtest* allows finding the optimal *momentum* (trading) strategy parameters, such as the *look-back interval*.

The function `btmomtop()` simulates (backtests) a *momentum strategy* which buys equal dollar amounts of the best performing stocks.

The function `btmomtop()` can be used to find the best choice of *momentum strategy* parameters.

```
> btmomtop <- function(retp, objfun, look_back=12, rebalf="months",
+                bid_offer=0.0, endd=rutils::calc_endpoints(retp, in
+   # Perform loop over end points
+   npts <- NROW(endd)
+   pnls <- lapply(2:(npts-1), function(ep) {
+     # Select the look-back returns
+     startp <- endd[max(1, ep-look_back)]
+     retis <- retp[startp:endd[ep], ]
+     # Calculate the best and worst performing stocks in-sample
+     perfstat <- sapply(retis, objfun)
+     perfstat <- sort(perfstat, decreasing=TRUE)
+     symbolb <- names(head(perfstat, topstocks))
+     symbolw <- names(tail(perfstat, topstocks))
+     # Calculate the momentum weights
+     weightv <- numeric(NCOL(retp))
+     names(weightv) <- colnames(retp)
+     weightv[symbolb] <- 1
+     weightv[symbolw] <- (-1)
+     # Calculate the in-sample momentum pnls
+     pnls <- HighFreq::mult_mat(weightv, retis)
+     pnls <- rowMeans(pnls, na.rm=TRUE)
+     # Scale weights so in-sample pnl volatility is same as equal w
+     weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnls)
+     # Calculate the out-of-sample momentum returns
+     pnls <- HighFreq::mult_mat(weightv, retp[(endd[ep]+1):endd[ep+
+     pnls <- rowMeans(pnls, na.rm=TRUE)
+     drop(pnls)
+   })  # end lapply
+   pnls <- rutils::do_call(c, pnls)
+   pnls
+ }  # end btmomtop
```

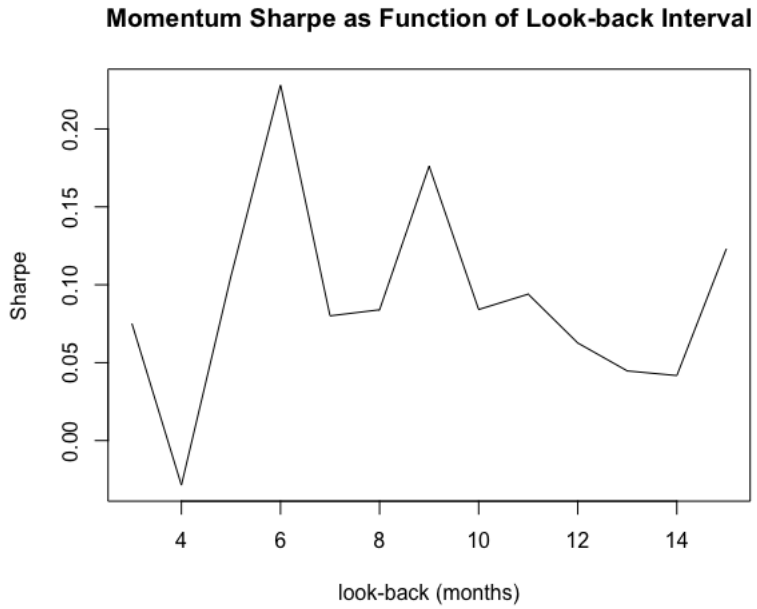
Optimization of Momentum Strategy Parameters

The performance of the *momentum* strategy depends on the length of the *look-back interval* used for calculating the past performance.

Research indicates that the optimal length of the *look-back interval* for momentum is about 8 to 12 months.

The dependence on the length of the *look-back interval* is an example of the *bias-variance tradeoff*. If the *look-back interval* is too short, the past performance estimates have high *variance*, but if the *look-back interval* is too long, the past estimates have high *bias*.

Performing many *backtests* on multiple trading strategies risks identifying inherently unprofitable trading strategies as profitable, purely by chance (known as *p-value hacking*).




```
> # Perform backtests for vector of look-back intervals
> look_backs <- seq(3, 15, by=1)
> endd <- rutils::calc_endpoints(retp, interval="months")
> # Warning - takes very long
> pnll <- lapply(look_backs, btmomtop, retp=retp, endd=endd, objfun=objfun)
> # Perform parallel loop under Mac-OSX or Linux
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnll <- mclapply(look_backs, btmomtop, retp=retp, endd=endd, objfun=objfun, mc.cores=ncores)
> sharper <- sqrt(252)*sapply(pnll, function(pnl) mean(pnl)/sd(pnl))
```

```
> # Plot Sharpe ratios of momentum strategies
> plot(x=look_backs, y=sharper, t="l",
+   main="Momentum Sharpe as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="Sharpe")
```

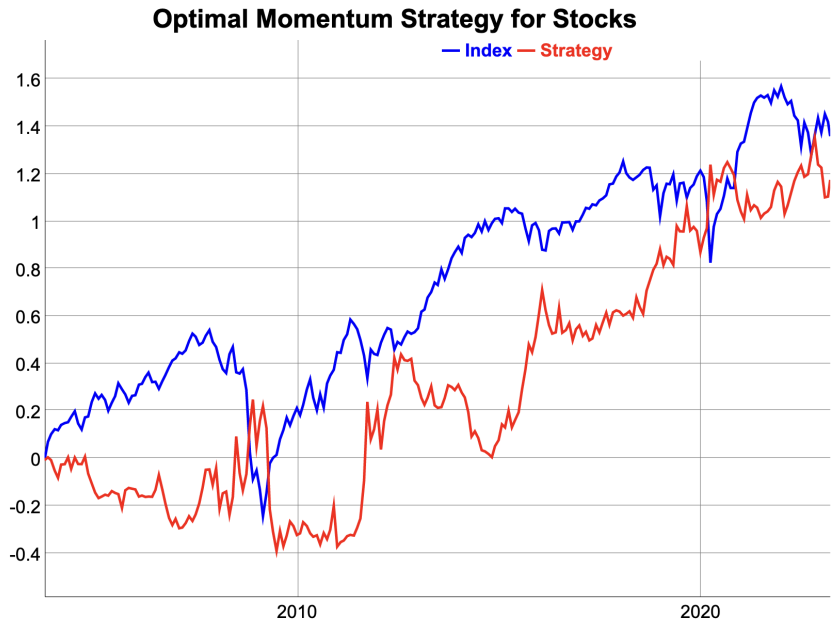
Optimal Stock Momentum Strategy

The best stock momentum strategy underperforms the index because of a poor choice of the model type.

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

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

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




```
> # Calculate best pnls of momentum strategy
> whichmax <- which.max(sharper)
> look_backs[whichmax]
> pnls <- pnll[[whichmax]]
> # Add stub period
> pnls <- c(rowMeans(retp[endd[1]:endd[2], ], na.rm=TRUE), pnls)
> pnls <- xts::xts(pnls, order.by=datev)
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```

```
> # Plot dygraph of stock index and momentum strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Weighted Momentum Strategy Functional

Performing a *backtest* allows finding the optimal *momentum* (trading) strategy parameters, such as the *look-back interval*.

The function `btmomweight()` simulates (backtests) a *momentum strategy* which buys dollar amounts proportional to the past performance of the stocks.

The function `btmomweight()` can be used to find the best choice of *momentum strategy* parameters.

```
> btmomweight <- function(retp, objfun, look_back=12, rebalf="months
+   bid_offer=0.0, endd=rutils::calc_endpoints(retp, interval=rebalf
+   # Perform loop over end points
+   npts <- NROW(endd)
+   pnls <- lapply(2:(npts-1), function(ep) {
+     # Select the look-back returns
+     startp <- endd[max(1, ep-look_back)]
+     retis <- retp[startp:endd[ep], ]
+     # Calculate weights proportional to performance
+     perfstat <- sapply(retis, objfun)
+     weightv <- perfstat
+     # Calculate the in-sample portfolio returns
+     pnls <- HighFreq::mult_mat(weightv, retis)
+     pnls <- rowMeans(pnls, na.rm=TRUE)
+     # Scale weights so in-sample pnl volatility is same as equal w
+     weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnls)
+     # Calculate the out-of-sample momentum returns
+     pnls <- HighFreq::mult_mat(weightv, retp[(endd[ep]+1):endd[ep+
+     pnls <- rowMeans(pnls, na.rm=TRUE)
+     drop(pnls)
+   })  # end lapply
+   rutils::do_call(c, pnls)
+ }  # end btmomweight
```

Optimal Weighted Stock Momentum Strategy

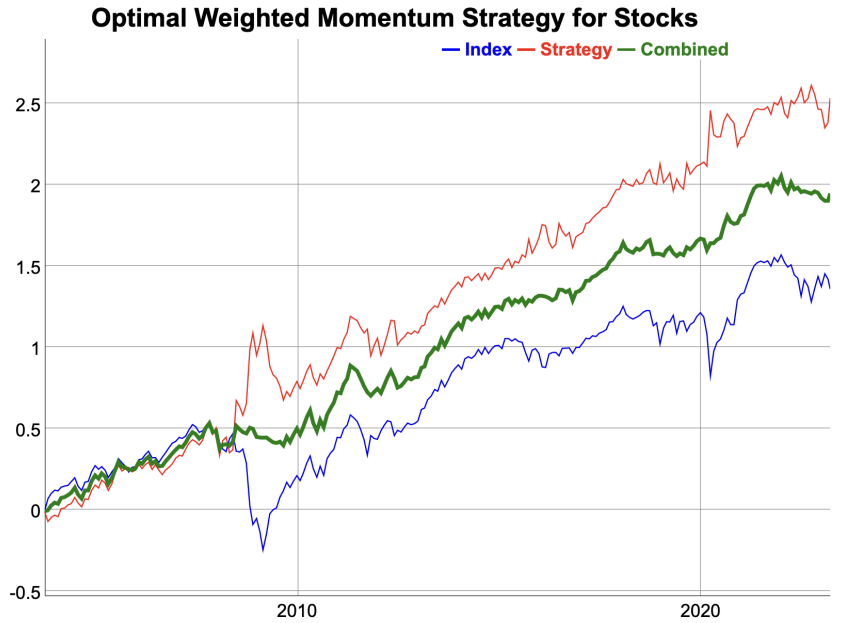
The stock momentum strategy produces a similar absolute return as the index, and also a similar Sharpe ratio.

The advantage of the momentum strategy is that it has a low correlation to stocks, so it can provide significant risk diversification when combined with stocks.

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

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

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




```
> # Perform backtests for vector of look-back intervals
> look_backs <- seq(3, 15, by=1)
> pnll <- lapply(look_backs, btmomweight, retp=retp, endd=endd, ob
> # Or perform parallel loop under Mac-OSX or Linux
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnll <- mclapply(look_backs, btmomweight, retp=retp, endd=endd,
> sharper <- sqrt(252)*sapply(pnll, function(pnl) mean(pnl)/sd(pnl
> # Plot Sharpe ratios of momentum strategies
> plot(x=look_backs, y=sharper, t="l",
+   main="Momentum Sharpe as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="Sharpe")
```

```
> # Calculate best pnls of momentum strategy
> whichmax <- which.max(sharper)
> look_backs[whichmax]
> pnls <- pnll[[whichmax]]
> pnls <- c(rowMeans(retp[endd[1]:endd[2], ], na.rm=TRUE), pnls)
> pnls <- xts::xts(pnls, order.by=datev)
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls, 0.5*(indeks+pnls))
> colnames(wealthv)[3] <- "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 stock index and momentum strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Weighted Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
```

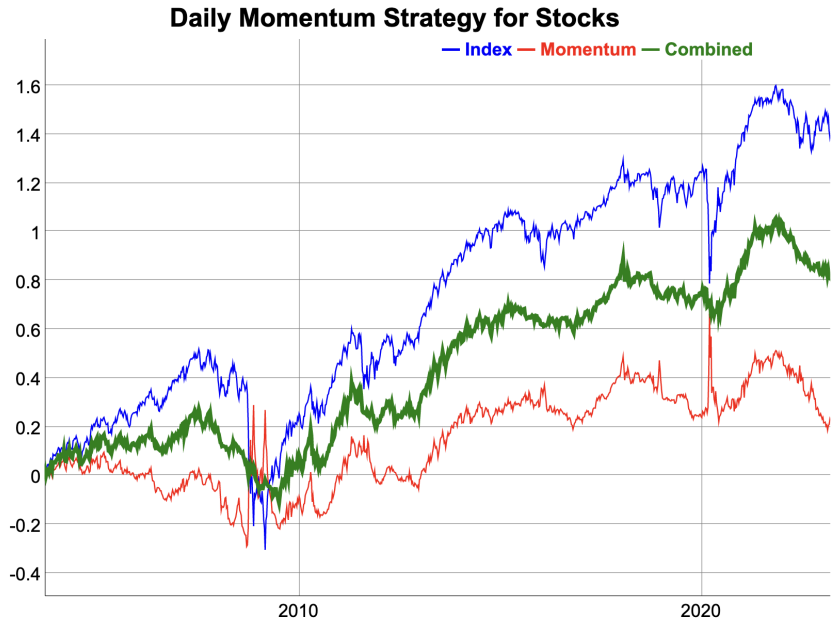
Momentum Strategy With Daily Rebalancing

In a momentum strategy with *daily rebalancing*, the weights are updated every day and the portfolio is rebalanced accordingly.

The momentum strategy with *daily rebalancing* performs worse than with *monthly rebalancing* because of the daily variance of the weights.

A momentum strategy with *daily rebalancing* requires more computations so compiled `C++` functions must be used instead of `apply()` loops.

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.




```
> # To simplify, set NAs to zero
> retp[is.na(retp)] <- 0
> # Calculate the trailing average returns and variance using C++ c
> lambda <- 0.99
> meanm <- HighFreq::run_mean(retp, lambda=lambda)
> varm <- HighFreq::run_var(retp, lambda=lambda)
> # Calculate the trailing Kelly ratio
> weightv <- ifelse(varm > 0, meanm/varm, 0)
> weightv[1, ] <- 1
> weightv <- weightv/sqrt(rowSums(weightv^2))
> weightv <- rutils::lagit(weightv)
> # Calculate the momentum profits and losses
> pnls <- rowSums(weightv*retp)
> # Calculate the transaction costs
> bid_offer <- 0.0
> costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weightv)))
> pnls <- (pnls - costs)
```

```
> # Scale the momentum volatility to the equal weight index
> indeksd <- sd(indeks)
> pnls <- indeksd*pnls/sd(pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls, 0.5*(indeks+pnls))
> colnames(wealthv)[2:3] <- c("Momentum", "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 stock index and momentum strategy
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Daily Momentum Strategy Functional

The function `btmomdaily()` simulates a momentum strategy with *daily rebalancing*.

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

If the argument `trend = -1` then it simulates a mean-reverting strategy (buys the worst performing stocks and sells the best performing).

Performing a *backtest* allows finding the optimal *momentum* (trading) strategy parameters, such as the *look-back interval*.

The function `btmomdaily()` can be used to find the best choice of *momentum strategy* parameters.

```
> # Define backtest functional for daily momentum strategy
> # If trend=(-1) then it backtests a mean reverting strategy
> btmomdaily <- function(retp, lambda=0.9, trend=1, bid_offer=0.0,
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   # Calculate the trailing Kelly ratio
+   meanm <- HighFreq::run_mean(retp, lambda=lambda)
+   varm <- HighFreq::run_var(retp, lambda=lambda)
+   weightv <- ifelse(varm > 0, meanm/varm, 0)
+   weightv[1, ] <- 1
+   weightv <- weightv/sqrt(rowSums(weightv^2))
+   weightv <- rutils::lagit(weightv)
+   # Calculate the momentum profits and losses
+   pnls <- trend*rowSums(weightv*retp)
+   # Calculate the transaction costs
+   costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weightv)))
+   (pnls - costs)
+ }  # end btmomdaily
```

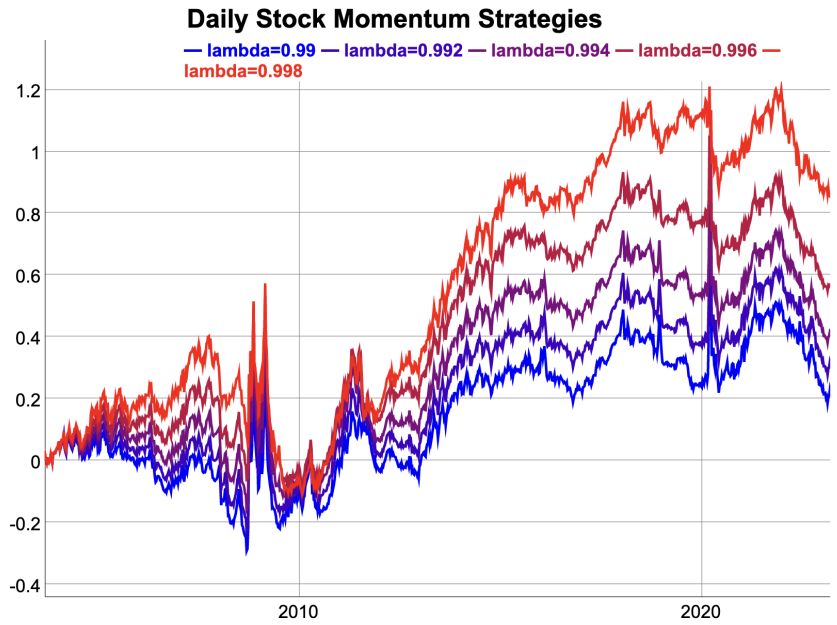
Multiple Daily Stock Momentum Strategies

Multiple daily momentum strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of λ parameters.

The best performing momentum strategies with *daily rebalancing* are with λ parameters close to 1.

The momentum strategies with *daily rebalancing* perform worse than with *monthly rebalancing* because of the daily variance of the weights.

```
> # Simulate multiple daily stock momentum strategies
> lambdas <- seq(0.99, 0.999, 0.002)
> pnls <- sapply(lambdas, btmomdaily, retp=retp)
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) indeksd*pnl/sd(pnl))
> colnames(pnls) <- paste0("lambda=", lambdas)
> pnls <- xts::xts(pnls, datev)
> tail(pnls)
```




```
> # Plot dygraph of daily stock momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily Stock Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot daily stock momentum strategies using quantmod
> plot_theme <- chart_theme()
> plot_theme$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls)[endd],
+   theme=plot_theme, name="Daily Stock Momentum Strategies")
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retp)),
+   col=plot_theme$col$line.col, bty="n")
```

Daily Momentum Strategy with Holding Period

The daily ETF momentum strategy can be improved by introducing a *holding period* for the portfolio.

Instead of holding the portfolio for only a day, it's held for several days and gradually liquidated. So many past momentum portfolios are held at the same time.

This is equivalent to averaging the portfolio weights over the past.

The best length of the *holding period* depends on the *bias-variance tradeoff*.

If the *holding period* is too short then the weights have too much day-over-day *variance*.

If the *holding period* is too long then the weights have too much *bias* (they are stale).

The decay parameter λ determines the length of the *holding period*. Smaller values of λ produce a faster decay corresponding to a shorter *holding period*, and vice versa.

The optimal value of the λ parameter can be determined by cross-validation (backtesting).

The function `btmomdailyhold()` simulates a momentum strategy with *daily rebalancing* with a holding period.

```
> # Define backtest functional for daily momentum strategy
> # If trend=(-1) then it backtests a mean reverting strategy
> btmomdailyhold <- function(retp, lambda=0.9, trend=1, bid_offer=0.
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   # Calculate the trailing Kelly ratio
+   meanm <- HighFreq::run_mean(retp, lambda=lambda)
+   varm <- HighFreq::run_var(retp, lambda=lambda)
+   weightv <- ifelse(varm > 0, meanm/varm, 0)
+   weightv[1, ] <- 1
+   weightv <- weightv/sqrt(rowSums(weightv^2))
+   # Average the past weights
+   weightv <- HighFreq::run_mean(weightv, lambda=lambda)
+   weightv <- rutils::lagit(weightv)
+   # Calculate the momentum profits and losses
+   pnls <- trend*rowSums(weightv*retp)
+   # Calculate the transaction costs
+   costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weightv)))
+   (pnls - costs)
+ }  # end btmomdailyhold
```

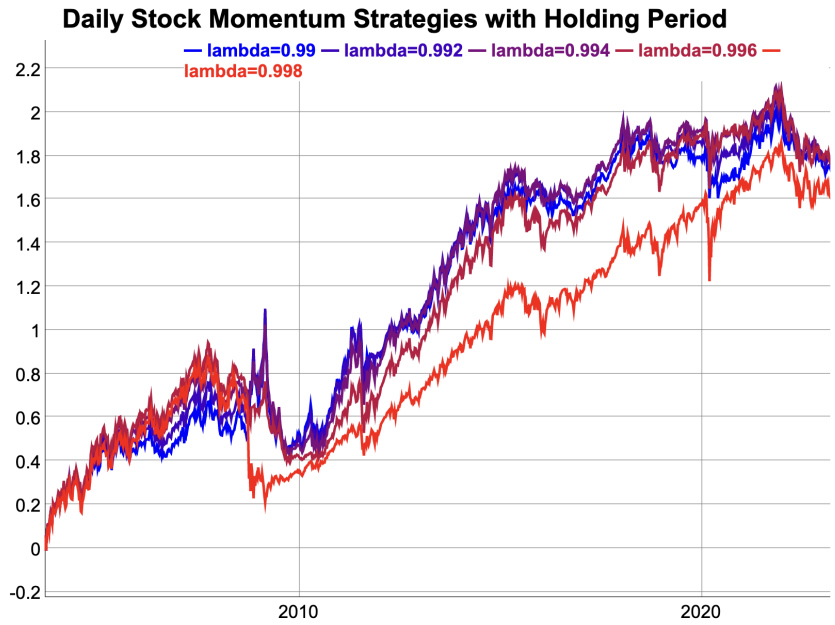
Multiple Daily Momentum Strategies With Holding Period

Multiple daily momentum strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of λ parameters (holding periods).

The daily momentum strategies with a holding period perform better than with daily rebalancing.

The reason is that a longer holding period averages the weights and reduces their variance. But this also increases their bias, so there's an optimal holding period for an optimal bias-variance tradeoff.

```
> # Simulate multiple daily stock momentum strategies with holding p
> lambdas <- seq(0.99, 0.999, 0.002)
> pnls <- sapply(lambdas, btmomdailyhold, retp=retp)
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) indeksd*pnl/sd(pnl))
> colnames(pnls) <- paste0("lambda=", lambdas)
> pnls <- xts::xts(pnls, datev)
```



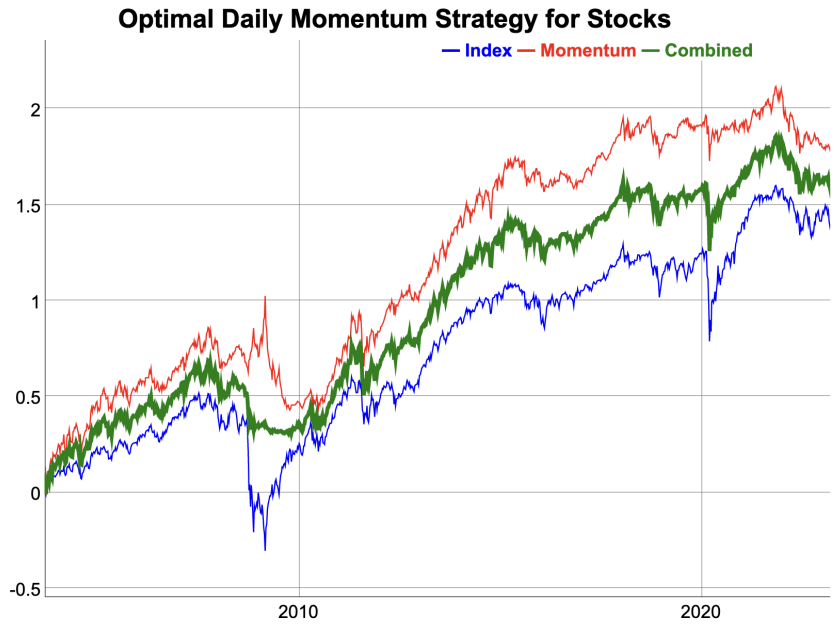

```
> # dygraph of daily stock momentum strategies with holding period
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily Stock Momentum Strategies with Holding Period") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot of daily stock momentum strategies with holding period
> plot_theme <- chart_theme()
> plot_theme$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls)[endd],
+   theme=plot_theme, name="Daily Stock Momentum Strategies with Ho
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retp)),
+   col=plot_theme$col$line.col, bty="n")
```

Optimal Momentum Strategy With Holding Period

The daily momentum strategies with a holding period perform better than with daily rebalancing.

The reason is that a longer holding period averages the weights and reduces their variance. But this also increases their bias, so there's an optimal holding period for an optimal bias-variance tradeoff.

```
> # Calculate best pnls of momentum strategy
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> whichmax <- which.max(sharper)
> lambdas[whichmax]
> pnls <- pnls[, whichmax]
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls, 0.5*(indeks+pnls))
> colnames(wealthv)[2:3] <- c("Momentum", "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 stock index and momentum strategy
> colorv <- c("blue", "red", "green")
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting Stock Momentum Strategies

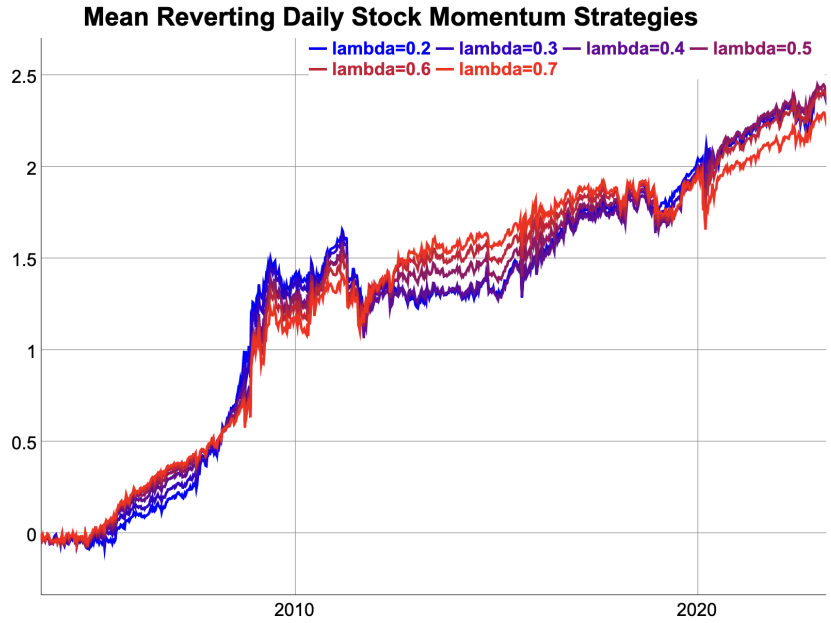
Multiple *mean reverting* stock momentum strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of λ parameters.

If the argument `trend = -1` then the function `btmomdaily()` simulates a mean-reverting strategy (buys the worst performing stocks and sells the best performing).

The *mean reverting* momentum strategies for the stock constituents perform the best for small λ parameters.

The *mean reverting* momentum strategies had their best performance prior to and during the 2008 financial crisis.

This simulation doesn't account for transaction costs, which could 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.




```
> # Perform sapply loop over look_backs
> lambdas <- seq(0.2, 0.7, 0.1)
> pnls <- sapply(lambdas, btmomdaily, retp=retp, trend=(-1))
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) indeksd*pnl/sd(pnl))
> colnames(pnls) <- paste0("lambda=", lambdas)
> pnls <- xts::xts(pnls, datev)
```

```
> # Plot dygraph of mean reverting daily stock momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Mean Reverting Daily Stock Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=400)
> # Plot mean reverting daily stock momentum strategies using quantm
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorRampPalette(c("blue", "red"))(NCOL
> quantmod::chart_Series(cumsum(pnls)[endd],
+   theme=plot_theme, name="Mean Reverting Daily Stock Momentum Stra
> legend("topleft", legend=colnames(pnls),
+   inset=0.05, bg="white", cex=0.7, lwd=rep(6, NCOL(retp)),
+   col=plot_theme$col$line.col, bty="n")
```

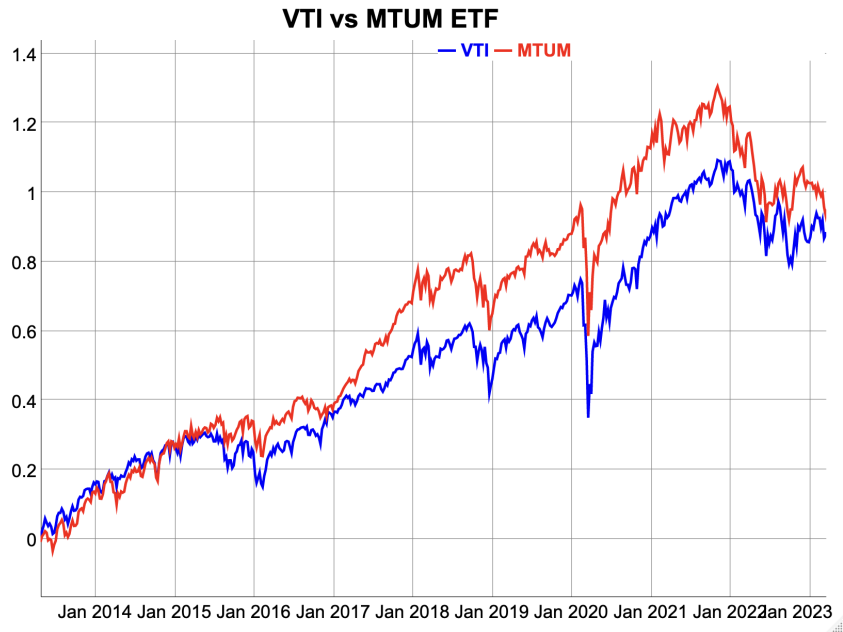
The MTUM Momentum ETF

The *MTUM* ETF is an actively managed ETF which follows a momentum strategy for stocks.

The *MTUM* ETF has a slightly higher absolute return than the *VTI* ETF, but it has a slightly lower Sharpe ratio.

The weak performance of the *MTUM* ETF demonstrates that it's difficult to implement a successful momentum strategy for individual stocks.

```
> # Calculate the scaled prices of VTI vs MTUM ETF
> wealthv <- na.omit(rutils::etfenv$returns[, c("VTI", "MTUM")])
> colnames(wealthv) <- c("VTI", "MTUM")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot of scaled prices of VTI vs MTUM ETF
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI vs MTUM ETF") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(width=300)
```

Momentum Weights for PCA Portfolios

The principal components are portfolios of stocks and can be traded directly as if they were single stocks.

The returns of the PCA portfolios are orthogonal to each other - the correlations of returns are equal to zero.

If the returns are orthogonal and if the momentum weights are proportional to the *Kelly ratios* (the returns divided by their variance):

$$w_i = \frac{\bar{r}_i}{\sigma_i^2}$$

Then the momentum weights are equal to the *maximum Sharpe* portfolio weights, equal to: $\mathbb{C}^{-1}\bar{r}$, where $\mathbb{C}$ is the covariance matrix (which is diagonal in this case).

So the momentum strategy for assets with orthogonal returns is equivalent to an optimal portfolio strategy.

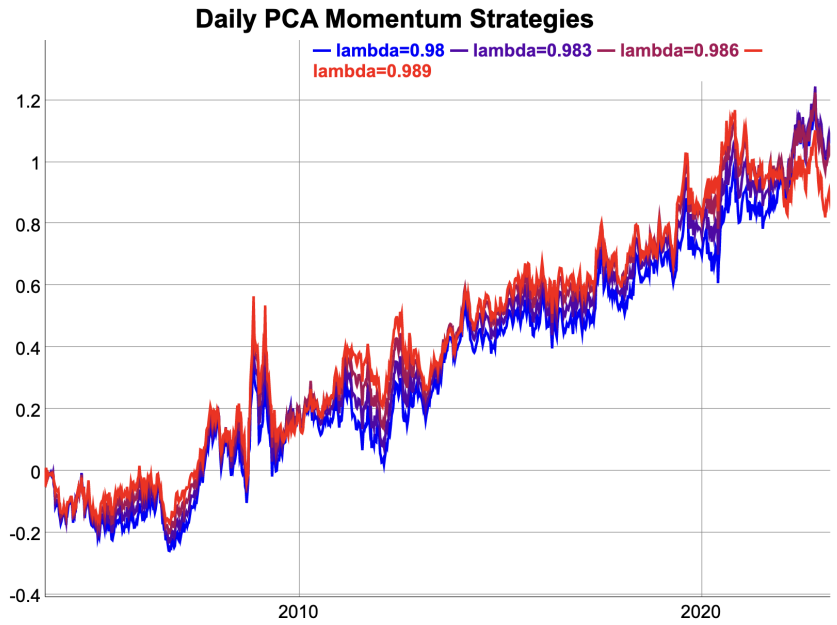
```
> # Calculate the PCA weights for standardized returns
> retsc <- lapply(retp, function(x) (x - mean(x))/sd(x))
> retsc <- do.call(cbind, retsc)
> covmat <- cov(retsc)
> pcad <- eigen(covmat)
> pcaw <- pcad$vectors
> rownames(pcaw) <- colnames(retp)
> sort(-pcaw[, 1], decreasing=TRUE)
> sort(pcaw[, 2], decreasing=TRUE)
> round((t(pcaw) %*% pcaw)[1:5, 1:5], 4)
> # Calculate the PCA time series from stock returns using PCA weigh
> retpca <- retsc %*% pcaw
> round((t(retpca) %*% retpca)[1:5, 1:5], 4)
> # Calculate the PCA using prcomp()
> pcad <- prcomp(retsc, center=FALSE, scale=FALSE)
> all.equal(abs(pcad$x), abs(retpca), check.attributes=FALSE)
> retpca <- xts::xts(retpca, order.by=datev)
```

Momentum Strategy for PCA Portfolios

The momentum strategy can be improved by applying it to PCA portfolios.

The lowest order principal components exhibit greater trending (positive autocorrelations), so they have better momentum strategy performance than individual stocks.

```
> # Simulate daily PCA momentum strategies for multiple lambda parar
> dimax <- 21
> lambdas <- seq(0.98, 0.99, 0.003)
> pnls <- mclapply(lambdas, btmomdailyhold, retp=retpca[, 1:dimax],
> pnls <- lapply(pnls, function(pnl) indeksd*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdas)
> pnls <- xts::xts(pnls, datev)
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdas, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Parameter",
+   xlab="lambda", ylab="Sharpe")
```




```
> # Plot dygraph of daily PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> endd <- rutils::calc_endpoints(retpca, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=400)
```

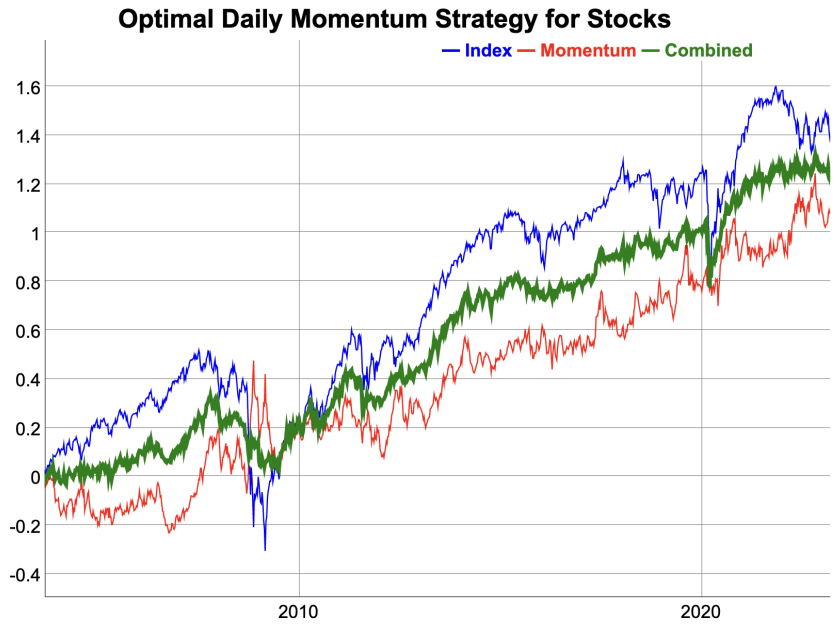
Optimal PCA Momentum Strategy

The PCA momentum strategy using only the lowest order principal components performs well when combined with the index.

But this is thanks to using the in-sample principal components.

The best performing PCA momentum strategy has a relatively small decay parameter λ, so it's able to quickly adjust to changes in market direction.

```
> # Calculate best pnls of PCA momentum strategy
> whichmax <- which.max(sharper)
> lambdas[whichmax]
> pnls <- pnls[, whichmax]
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls, 0.5*(indeks+pnls))
> colnames(wealthv)[2:3] <- c("Momentum", "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 stock index and PCA momentum strategy
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting PCA Momentum Strategies

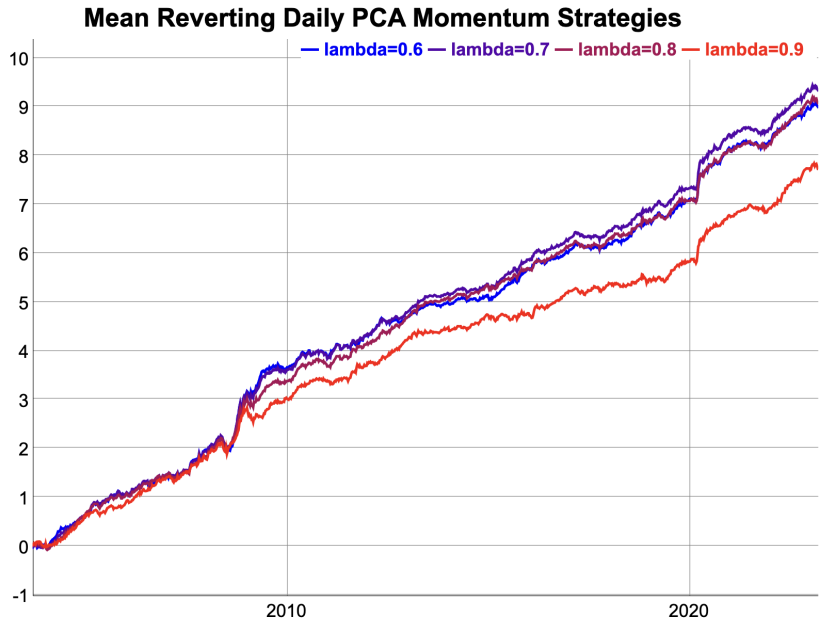
The *mean reverting* momentum strategy performs well for the higher order principal components.

This is because the higher order principal components exhibit greater mean reversion (negative autocorrelations) than individual stocks.

The *mean reverting* momentum strategies had their best performance in periods of high volatility, especially prior to and during the 2008 financial crisis.

This simulation doesn't account for transaction costs, which could 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.

If the argument `trend = -1` then the function `btmomdailyhold()` simulates a mean-reverting strategy (buys the worst performing stocks and sells the best performing).




```
> # Simulate daily PCA momentum strategies for multiple lambda para
> lambdas <- seq(0.6, 0.9, 0.1)
> pnls <- mclapply(lambdas, btmomdailyhold, retp=retpca[, (dimax+1):NCOL(retpca)],
+    trend=(-1), mc.cores=ncores)
> pnls <- lapply(pnls, function(pnl) indeksd*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdas)
> pnls <- xts::xts(pnls, datev)
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdas, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Parameter",
+   xlab="lambda", ylab="Sharpe")
```

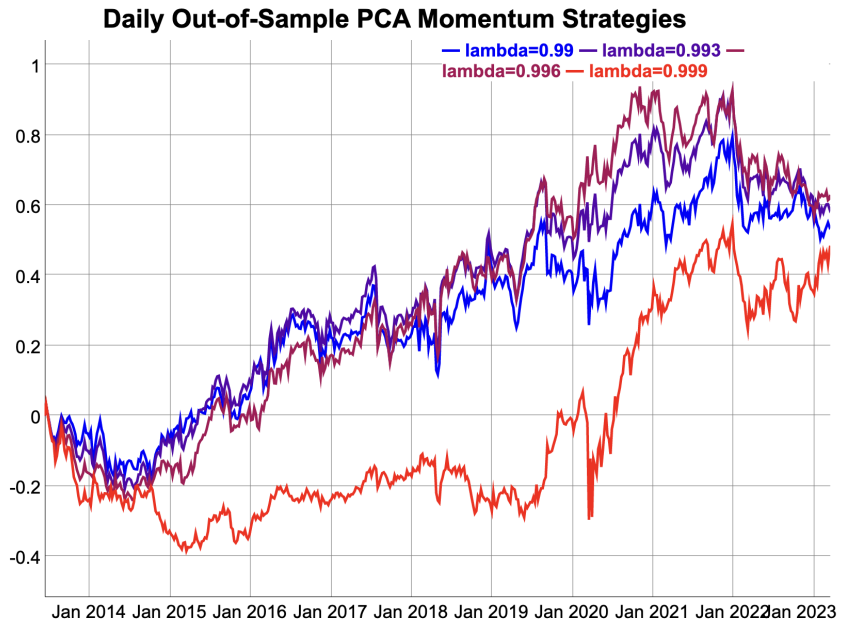
```
> # Plot dygraph of daily PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Mean Reverting Daily PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=400)
```

PCA Momentum Strategy Out-of-Sample

The principal component weights are calculated in-sample and applied out-of-sample.

The performance is much lower than in-sample, but it's still positive.

```
> # Define in-sample and out-of-sample intervals
> cutoff <- nrows %/% 2
> datev[cutoff]
> insample <- 1:cutoff
> outsample <- (cutoff + 1):nrows
> # Calculate the PCA weights in-sample
> pcad <- prcomp(retp[insample], center=FALSE, scale=TRUE)
> # Calculate the out-of-sample PCA time series
> retsc <- lapply(retp, function(x) x[outsample]/sd(x[insample]))
> retsc <- do.call(cbind, retsc)
> retpca <- xts::xts(retsc %*% pcad$rotation, order.by=datev[outsamp
> # Simulate daily PCA momentum strategies for multiple lambda param
> lambdas <- seq(0.99, 0.999, 0.003)
> pnls <- mclapply(lambdas, btmomdailyhold, retp=retpca[, 1:dimax]
> pnls <- lapply(pnls, function(pnl) indeksd*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdas)
> pnls <- xts::xts(pnls, datev[outsample])
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl
> plot(x=lambdas, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Parameter",
+   xlab="lambda", ylab="Sharpe")
```




```
> # Calculate a vector of weekly end points
> endd <- rutils::calc_endpoints(retpca, interval="weeks")
> # Plot dygraph of daily out-of-sample PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily Out-of-Sample PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

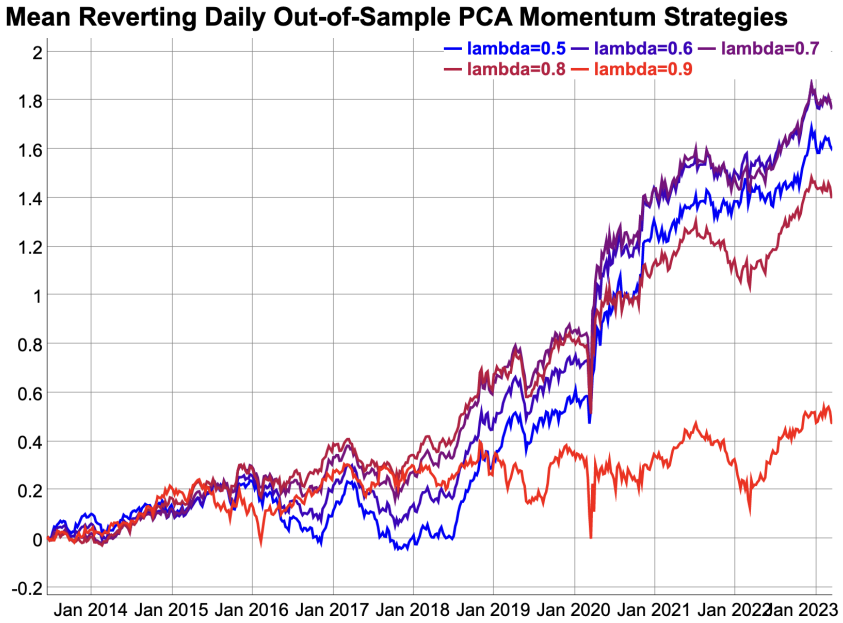
Mean Reverting PCA Momentum Strategy Out-of-Sample

The principal component weights are calculated in-sample and applied out-of-sample.

The performance is much lower than in-sample, but it's still positive.

This simulation doesn't account for transaction costs, which could 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.

```
> # Simulate daily PCA momentum strategies for multiple lambda parar
> lambdas <- seq(0.5, 0.9, 0.1)
> pnls <- mclapply(lambdas, btmomdailyhold, retp=retpca[, (dimax+1)
+    trend=(-1), mc.cores=ncores)
> pnls <- lapply(pnls, function(pnl) indeksd*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdas)
> pnls <- xts::xts(pnls, datev[outsample])
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl)
> plot(x=lambdas, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Parameter",
+   xlab="lambda", ylab="Sharpe")
```




```
> # Calculate a vector of weekly end points
> endd <- rutils::calc_endpoints(retpca, interval="weeks")
> # Plot dygraph of daily S&P500 momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Mean Reverting Daily Out-of-Sample PCA Momentum Strategies
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Momentum Strategy for an *ETF* Portfolio

The performance of the momentum strategy depends on the length of the *look-back interval* used for calculating the past performance.

Research indicates that the optimal length of the *look-back interval* for momentum is about 4 to 10 months.

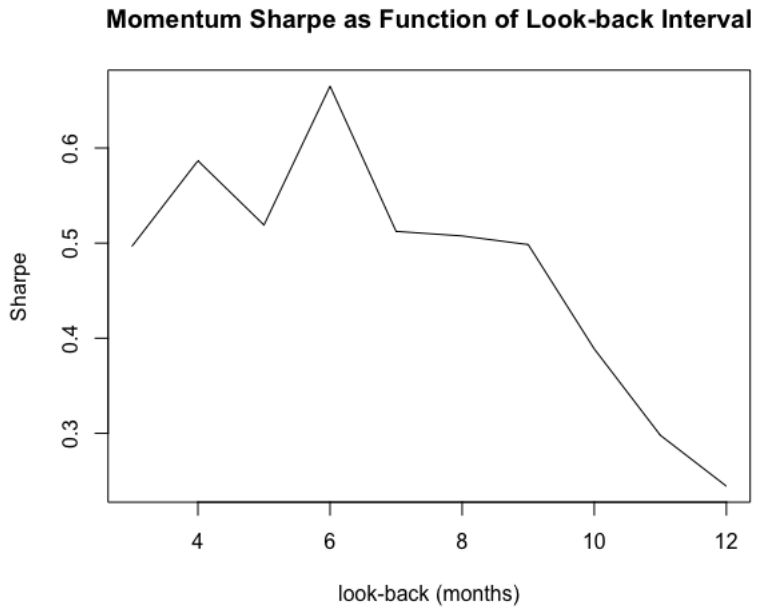
The dependence on the length of the *look-back interval* is an example of the *bias-variance tradeoff*. If the *look-back interval* is too short, the past performance estimates have high *variance*, but if the *look-back interval* is too long, the past estimates have high *bias*.

Performing many *backtests* on multiple trading strategies risks identifying inherently unprofitable trading strategies as profitable, purely by chance (known as *p-value hacking*).

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

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

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




```
> # Extract ETF returns
> symbolv <- c("VTI", "IEF", "DBC")
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> datev <- zoo::index(retp)
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> npts <- NROW(endd)
> # Perform backtests for vector of look-back intervals
> look_backs <- seq(3, 12, by=1)
> pnll <- lapply(look_backs, btmomweight, retp=retp, endd=endd, obj
> sharper <- sqrt(252)*sapply(pnll, function(pnl) mean(pnl)/sd(pnl))
> # Plot Sharpe ratios of momentum strategies
> plot(x=look_backs, y=sharper, t="l",
+   main="Momentum Sharpe as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="Sharpe")
```

Performance of Momentum Strategy for ETFs

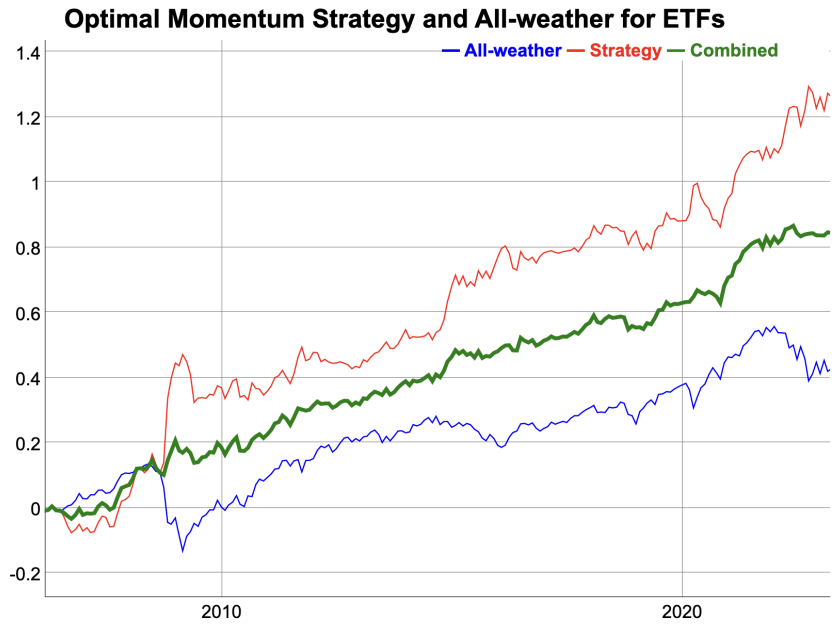
The momentum strategy for ETFs produces a higher absolute return and also a higher Sharpe ratio than the static *All-Weather* portfolio.

The momentum strategy for ETFs also has a very low correlation to the static *All-Weather* portfolio.

The momentum strategy works better for assets that are not correlated or are even anti-correlated.

The momentum strategy also works better for portfolios than for individual stocks because of risk diversification.

Portfolios of stocks can also be selected so that they are more autocorrelated - more trending - they have higher signal-to-noise ratios - larger Hurst exponents.




```
> # Calculate best pnls of momentum strategy
> whichmax <- which.max(sharper)
> look_backs[whichmax]
> pnls <- pnll[[whichmax]]
> pnls <- c(rowMeans(retp[endd[1]:endd[2], ]), pnls)
> # Define all-weather benchmark
> weightvaw <- c(0.30, 0.55, 0.15)
> all_weather <- retp %*% weightvaw
```

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(all_weather, pnls, 0.5*(all_weather+pnls))
> colnames(wealthv) <- c("All-weather", "Strategy", "Combined")
> cor(wealthv)
> wealthv <- xts::xts(wealthv, order.by=datev)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of stock index and momentum strategy
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Momentum Strategy and All-weather for ETFs") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

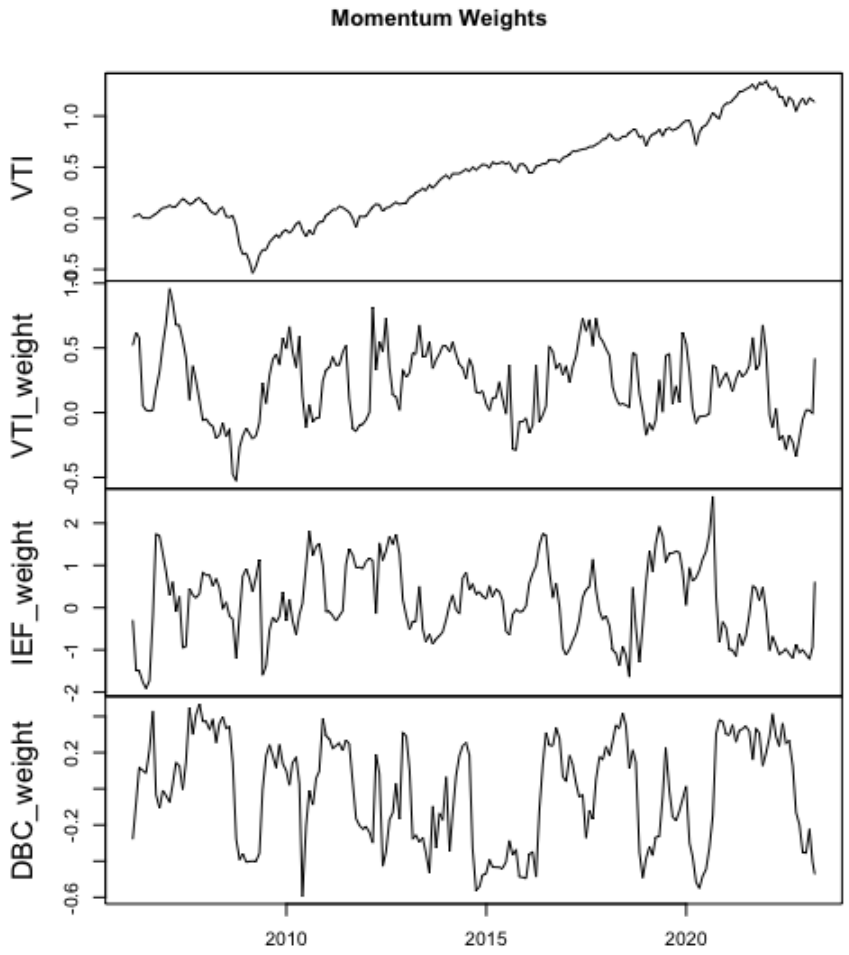
Time Series of Momentum Portfolio Weights

In momentum strategies, the portfolio weights are adjusted over time to be proportional to the past performance of the assets.

This way momentum strategies switch their weights to the best performing assets.

The weights are scaled to limit the portfolio *leverage* and its market *beta*.

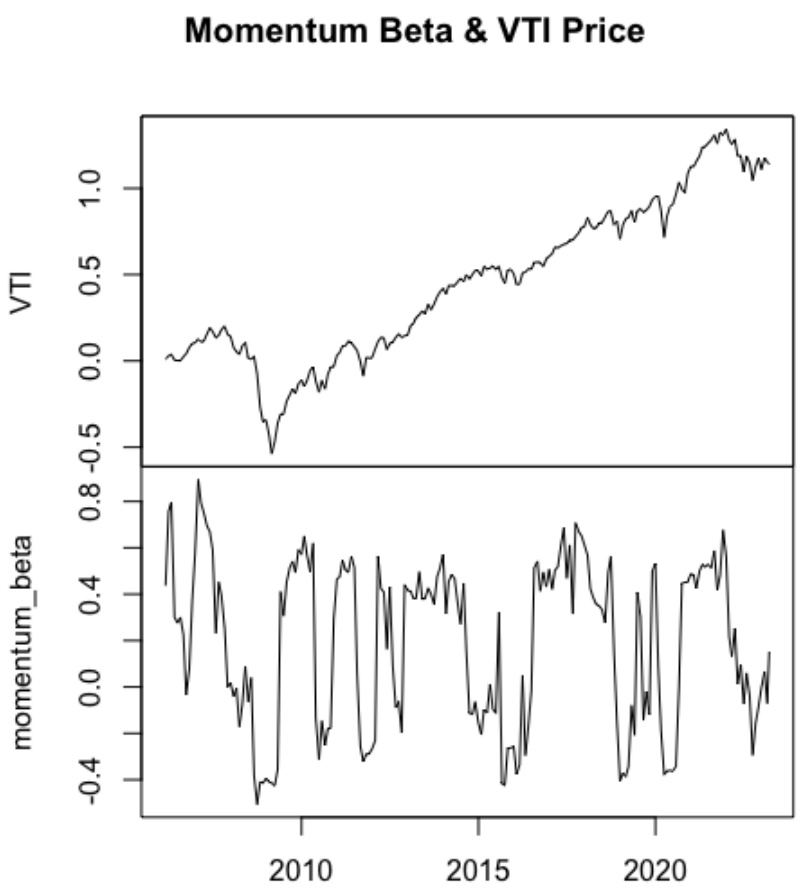
```
> # Calculate the momentum weights
> look_back <- look_backs[whichmax]
> weightv <- lapply(2:npts, function(ep) {
+   # Select the look-back returns
+   startp <- endd[max(1, ep-look_back)]
+   retis <- retp[startp:endd[ep], ]
+   # Calculate weights proportional to performance
+   perfstat <- sapply(retis, objfun)
+   weightv <- drop(perfstat)
+   # Scale weights so in-sample pnl volatility is same as equal wei
+   pnls <- retis %*% weightv
+   weightv*sd(rowMeans(retis))/sd(pnls)
+ })  # end lapply
> weightv <- rutils::do_call(rbind, weightv)
> # Plot of momentum weights
> retvti <- cumsum(retp$VTI)
> datav <- cbind(retvti[endd], weightv)
> colnames(datav) <- c("VTI", paste0(colnames(retp), "_weight"))
> zoo::plot.zoo(datav, xlab=NULL, main="Momentum Weights")
```



Momentum Strategy Market Beta

The momentum strategy market beta can be calculated by multiplying the *ETF* betas by the *ETF* portfolio weights.

```
> # Calculate ETF betas
> betasetf <- sapply(retp, function(x)
+   cov(retp$VTI, x)/var(retp$VTI))
> # Momentum beta is equal weights times ETF betas
> betav <- weightv %*% betasetf
> betav <- xts::xts(betav, order.by=datev[endd])
> colnames(betav) <- "momentum_beta"
> datav <- cbind(retvti[endd], betav)
> zoo::plot.zoo(datav, main="Momentum Beta & VTI Price", xlab="")
```

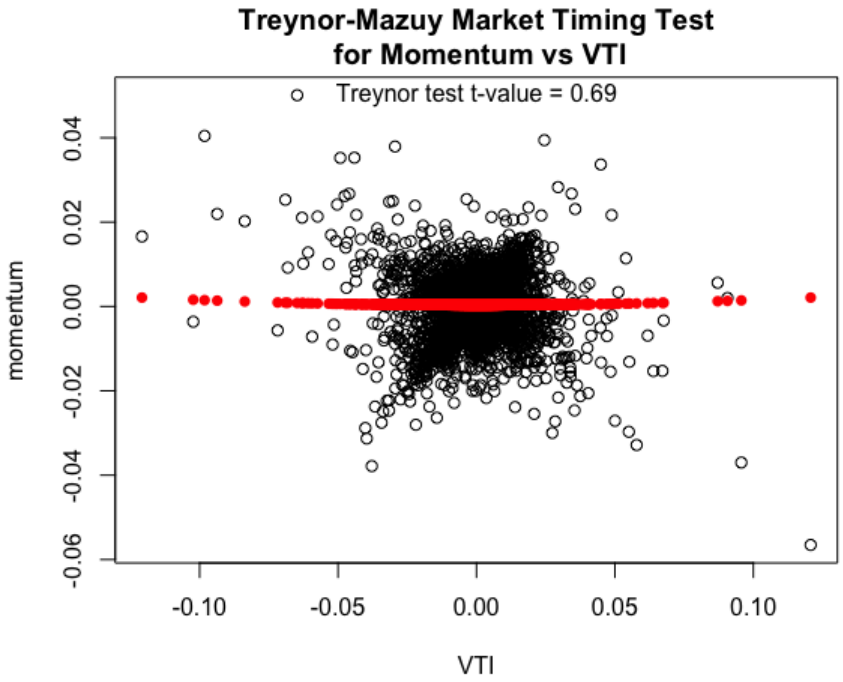


Momentum Strategy Market Timing Skill

Market timing skill is the ability to forecast the direction and magnitude of market returns.

The *Treynor-Mazuy* test shows that the momentum strategy has some *market timing* skill.

```
> # Merton-Henriksson test
> retvti <- retp$VTI
> predm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predm)[2:3] <- c("merton", "treynor")
> regmod <- lm(pnls ~ VTI + merton, data=predm); summary(regmod)
> # Treynor-Mazuy test
> regmod <- lm(pnls ~ VTI + treynor, data=predm); summary(regmod)
> # Plot residual scatterplot
> resids <- regmod$residuals
> plot.default(x=retvti, y=resids, xlab="VTI", ylab="momentum")
> title(main="Treynor-Mazuy Market Timing Test\n for Momentum vs VTI
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retvti, y=fitv, pch=16, col="red")
> text(x=0.0, y=max(resids), paste("Treynor test t-value =", tvalue))
```



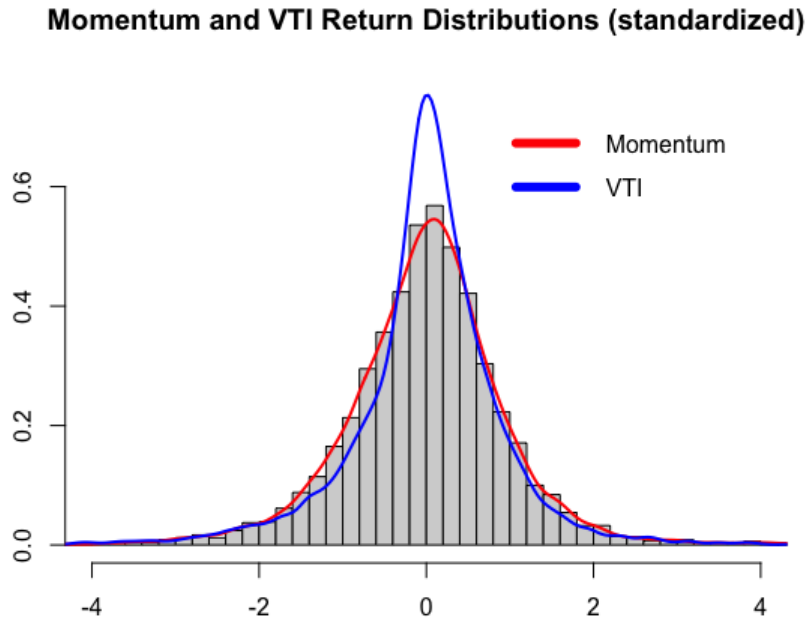
Skewness of Momentum Strategy Returns

Most assets with *positive returns* suffer from *negative skewness*.

The momentum strategy returns have more positive skewness compared to the negative skewness of *VTI*.

The momentum strategy is a genuine *market anomaly*, because it has both positive returns and positive skewness.

```
> # Standardize the returns
> pnlsd <- (pnls-mean(pnls))/sd(pnls)
> retvti <- (retvti-mean(retvti))/sd(retvti)
> # Calculate skewness and kurtosis
> apply(cbind(pnlsd, retvti), 2, function(x)
+   sapply(c(skew=3, kurt=4),
+     function(e) sum(x^e)))/NROW(retvti)
```




```
> # Calculate kernel density of VTI
> densvti <- density(retvti)
> # Plot histogram of momentum returns
> hist(pnlsd, breaks=80,
+   main="Momentum and VTI Return Distributions (standardized)",
+   xlim=c(-4, 4), ylim=range(densvti$y), xlab="", ylab="", freq=FAI
> # Draw kernel density of histogram
> lines(density(pnlsd), col='red', lwd=2)
> lines(densvti, col='blue', lwd=2)
> # Add legend
> legend("topright", inset=0.0, cex=1.0, title=NULL,
+  leg=c("Momentum", "VTI"), bty="n", y.intersp=0.5,
+  lwd=6, bg="white", col=c("red", "blue"))
```

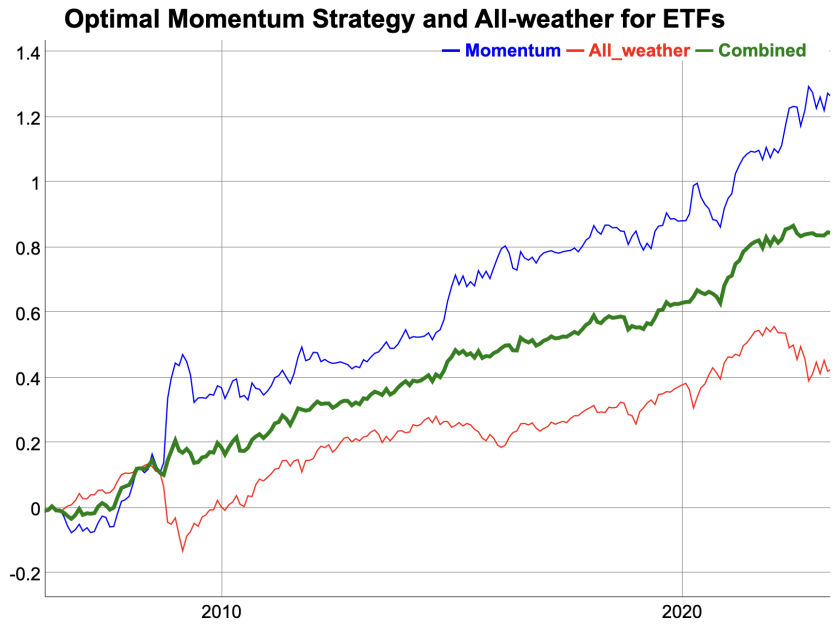
Combining Momentum with the *All-Weather* Portfolio

The momentum strategy has attractive returns compared to a static buy-and-hold strategy.

But the momentum strategy suffers from draw-downs called *momentum crashes*, especially after the market rallies from a sharp-sell-off.

This suggests that combining the momentum strategy with a static buy-and-hold strategy can achieve significant diversification of risk.

```
> # Combine momentum strategy with all-weather
> wealthv <- cbind(pnls, all_weather, 0.5*(pnls + all_weather))
> colnames(wealthv) <- c("Momentum", "All_weather", "Combined")
> wealthv <- xts::xts(wealthv, datev)
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Calculate strategy correlations
> cor(wealthv)
```




```
> # Plot ETF momentum strategy combined with All-Weather
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Momentum Strategy and All-weather for ETFs") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

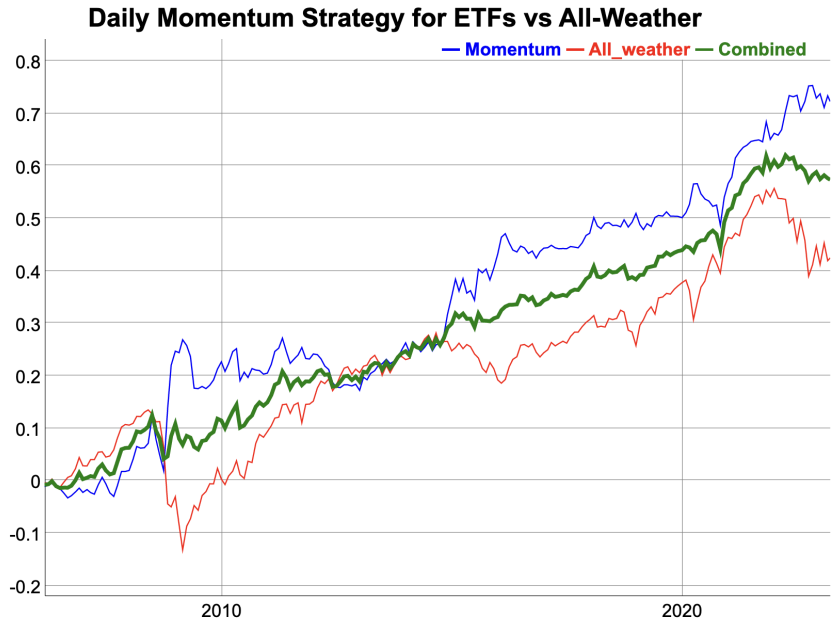
Momentum Strategy for ETFs With Daily Rebalancing

In a momentum strategy with *daily rebalancing*, the weights are updated every day and the portfolio is rebalanced accordingly.

A momentum strategy with *daily rebalancing* requires more computations so compiled C++ functions must be used instead of `apply()` loops.

The momentum strategy with *daily rebalancing* performs worse than the strategy with *monthly rebalancing* because of the daily variance of the weights.

```
> # Calculate the trailing variance
> look_back <- 152
> varm <- HighFreq::roll_var(retp, look_back=look_back)
> # Calculate the trailing Kelly ratio
> meanv <- HighFreq::roll_mean(retp, look_back=look_back)
> weightv <- ifelse(varm > 0, meanv/varm, 0)
> sum(is.na(weightv))
> weightv <- weightv/sqrt(rowSums(weightv^2))
> weightv <- rutils::lagit(weightv)
> # Calculate the momentum profits and losses
> pnls <- rowSums(weightv*retp)
> # Calculate the transaction costs
> bid_offer <- 0.0
> costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weightv)))
> pnls <- (pnls - costs)
```




```
> # Scale the momentum volatility to all_weather
> pnls <- sd(all_weather)*pnls/sd(pnls)
> # Calculate the wealth of momentum returns
> wealthv <- cbind(pnls, all_weather, 0.5*(pnls + all_weather))
> colnames(wealthv) <- c("Momentum", "All_weather", "Combined")
> wealthv <- xts::xts(wealthv, datev)
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> cor(wealthv)
> # Plot dygraph of the momentum strategy returns
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Momentum Strategy for ETFs vs All-Weather") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Portfolio Optimization Strategy

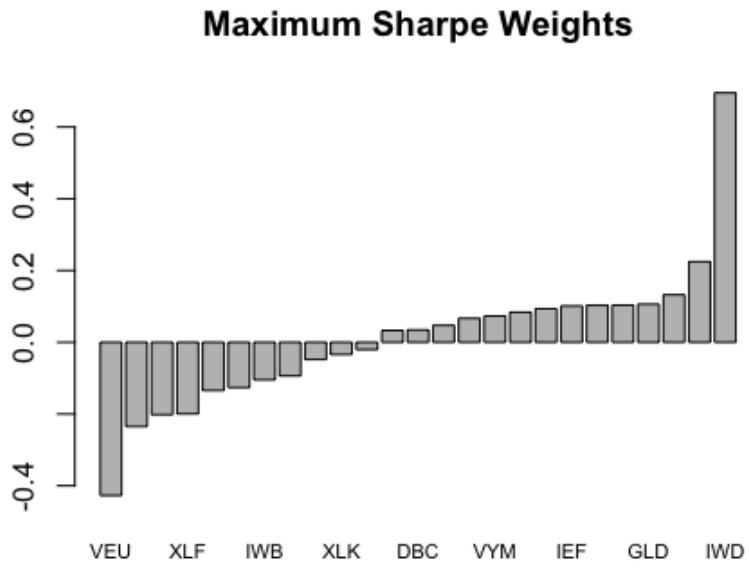
The *portfolio optimization* strategy invests in the best performing portfolio in the past *in-sample* interval, expecting that it will continue performing well *out-of-sample*.

The *portfolio optimization* strategy consists of:

1. Calculating the maximum Sharpe ratio portfolio weights in the *in-sample* interval,
2. Applying the weights and calculating the portfolio returns in the *out-of-sample* interval.

The optimal portfolio weights $\mathbf{w}$ are equal to the past in-sample excess returns $\mu = \mathbf{r} - r_f$ (in excess of the risk-free rate r_f) multiplied by the inverse of the covariance matrix $\mathbb{C}$:

$$\mathbf{w} = \mathbb{C}^{-1}\mu$$



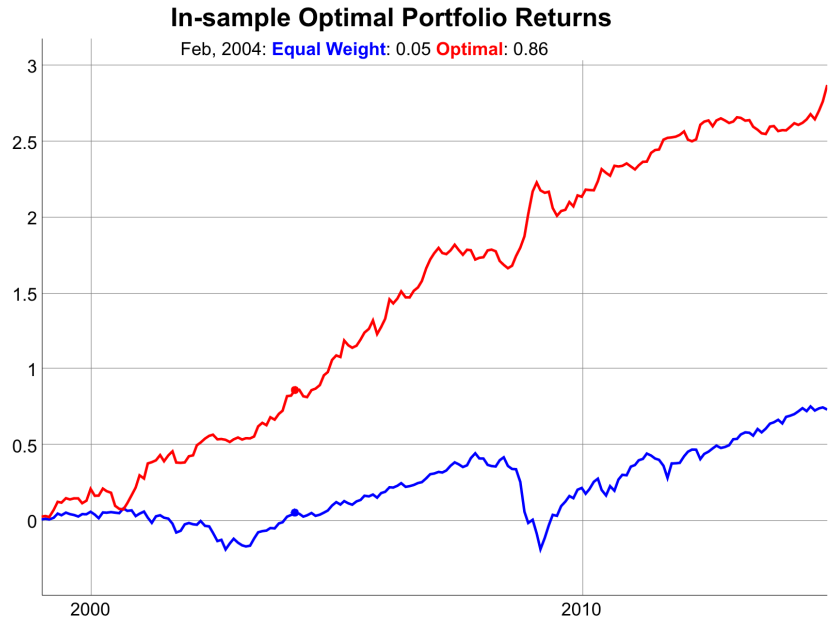

```
> # Select all the ETF symbols except "VXX", "SVXY" "MTUM", "QUAL"
> symbolv <- colnames(rutils::etfenv$returns)
> symbolv <- symbolv[!(symbolv %in% c("VXX", "SVXY", "MTUM", "QUAL
> # Extract columns of rutils::etfenv$returns and overwrite NA valu
> retp <- rutils::etfenv$returns[, symbolv]
> nstocks <- NCOL(retp)
> # retp <- na.omit(retp)
> retp[1, is.na(retp[1, ])] <- 0
> retp <- zoo::na.locf(retp, na.rm=FALSE)
> datev <- zoo::index(retp)
> # Returns in excess of risk-free rate
> riskf <- 0.03/252
> retx <- (retp - riskf)
```

```
> # Maximum Sharpe weights in-sample interval
> retis <- retp["/2014"]
> invmat <- MASS::ginv(cov(retis))
> weightv <- invmat %*% colMeans(retx["/2014"])
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retp)
> # Plot portfolio weights
> x11(width=6, height=5)
> par(mar=c(3, 3, 2, 1), oma=c(0, 0, 0, 0), mgp=c(2, 1, 0))
> barplot(sort(weightv), main="Maximum Sharpe Weights", cex.names=0.
```

Portfolio Optimization Strategy In-Sample

The in-sample performance of the optimal portfolio is much better than the equal weight portfolio.

```
> # Calculate in-sample portfolio returns
> insample <- xts::xts(retis %*% weightv, zoo::index(retis))
> indeks <- xts::xts(rowMeans(retis), zoo::index(retis))
> insample <- insample*sd(indeks)/sd(insample)
```



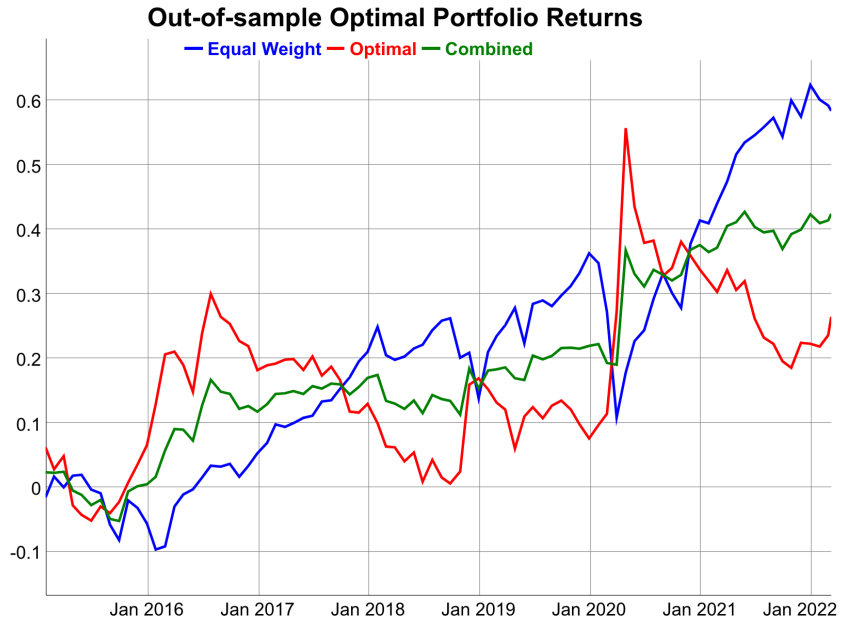

```
> # Plot cumulative portfolio returns
> pnls <- cbind(indeks, insample)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="In-sample Optimal Portfolio Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(width=300)
```

Portfolio Optimization Strategy Out-of-Sample

The out-of-sample performance of the optimal portfolio is not nearly as good as in-sample.

Combining the optimal portfolio with the equal weight portfolio produces and even better performing portfolio.

```
> # Calculate out-of-sample portfolio returns
> retsos <- retp["2015/"]
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
> indeks <- xts::xts(rowMeans(retsos), zoo::index(retsos))
> outsample <- outsample*sd(indeks)/sd(outsample)
> pnls <- cbind(indeks, outsample, (outsample + indeks)/2)
> colnames(pnls) <- c("Equal Weight", "Optimal", "Combined")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(pnls, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```



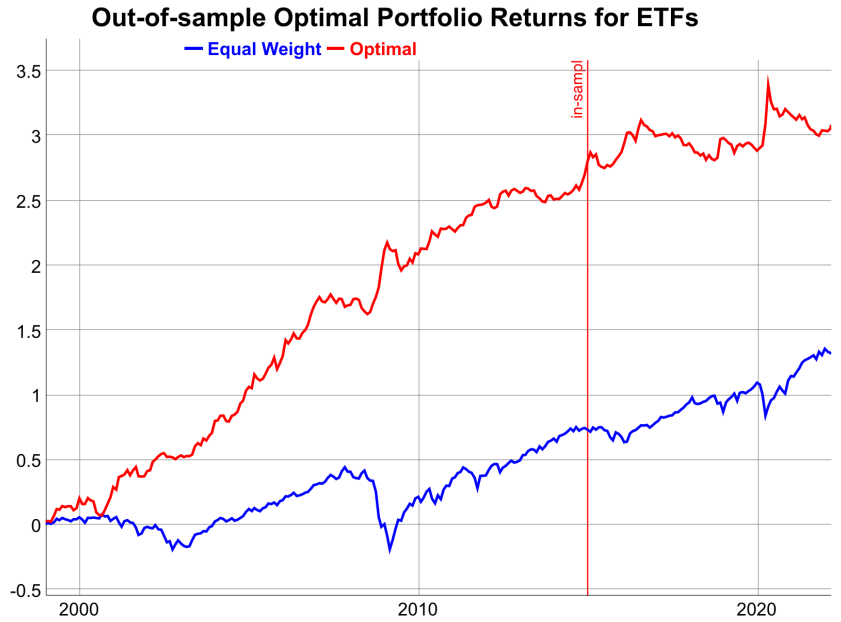

```
> # Plot cumulative portfolio returns
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Out-of-sample Optimal Portfolio Returns") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(width=300)
```

Portfolio Optimization Strategy for ETFs

The *portfolio optimization* strategy for ETFs is *overfitted* in the *in-sample* interval.

Therefore the strategy underperforms in the *out-of-sample* interval.

```
> # Maximum Sharpe weights in-sample interval
> invmat <- MASS::ginv(cov(retis))
> weightv <- invmat %*% colMeans(retx["/2014"])
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retp)
> # Calculate in-sample portfolio returns
> insample <- xts::xts(retis %*% weightv, zoo::index(retis))
> # Calculate out-of-sample portfolio returns
> retsos <- retp["2015/"]
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
```




```
> # Plot cumulative portfolio returns
> pnls <- rbind(insample, outsample)
> indeks <- xts::xts(rowMeans(retp), datev)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Out-of-sample Optimal Portfolio Returns for ETFs") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", stroke
+   dyLegend(width=300)
```

Regularized Inverse of Singular Covariance Matrix

The inverse of the covariance matrix of returns $\mathbb{C}$ can be calculated from its *eigenvalues* $\mathbb{D}$ and its *eigenvectors* $\mathbb{O}$:

$$\mathbb{C}^{-1} = \mathbb{O}\,\mathbb{D}^{-1}\,\mathbb{O}^{T}$$

If the number of time periods of returns (rows) is less than the number of stocks (columns), then some of the higher order eigenvalues are zero, and the above covariance matrix inverse is singular.

The *regularized inverse* $\mathbb{C}_n^{-1}$ is calculated by removing the zero eigenvalues, and keeping only the first n *eigenvalues*:

$$\mathbb{C}_n^{-1} = \mathbb{O}_n\,\mathbb{D}_n^{-1}\,\mathbb{O}_n^{T}$$

Where $\mathbb{D}_n$ and $\mathbb{O}_n$ are matrices with the higher order eigenvalues and eigenvectors removed.

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

```
> # Create rectangular matrix with collinear columns
> matrixv <- matrix(rnorm(10*8), nc=10)
> # Calculate covariance matrix
> covmat <- cov(matrixv)
> # Calculate inverse of covmat - error
> invmat <- solve(covmat)
> # Perform eigen decomposition
> eigend <- eigen(covmat)
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> # Set tolerance for determining zero singular values
> precv <- sqrt(.Machine$double.eps)
> # Calculate regularized inverse matrix
> notzero <- (eigenval > (precv*eigenval[1]))
> invreg <- eigenvec[, notzero] %*%
+   (t(eigenvec[, notzero]) / eigenval[notzero])
> # Verify inverse property of invreg
> all.equal(covmat, covmat %*% invreg %*% covmat)
> # Calculate regularized inverse of covmat
> invmat <- MASS::ginv(covmat)
> # Verify that invmat is same as invreg
> all.equal(invmat, invreg)
```

Dimension Reduction of the Covariance Matrix

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

The technique of *dimension reduction* calculates the inverse of a covariance matrix by removing the very small, higher order eigenvalues, to reduce the propagation of statistical noise and improve the signal-to-noise ratio:

$$\mathbb{C}_{DR}^{-1} = \mathbb{O}_{dimax}\, \mathbb{D}_{dimax}^{-1}\, \mathbb{O}_{dimax}^{T}$$

The parameter `dimax` specifies the number of eigenvalues used for calculating the *dimension reduction inverse* of the covariance matrix of returns.

Even though the *dimension reduction inverse* $\mathbb{C}_{DR}^{-1}$ does not satisfy the matrix inverse property (so it's biased), its out-of-sample forecasts are usually more accurate than those using the actual inverse matrix.

But removing a larger number of eigenvalues increases the bias of the covariance matrix, which is an example of the *bias-variance tradeoff*.

The optimal value of the parameter `dimax` can be determined using *backtesting* (*cross-validation*).

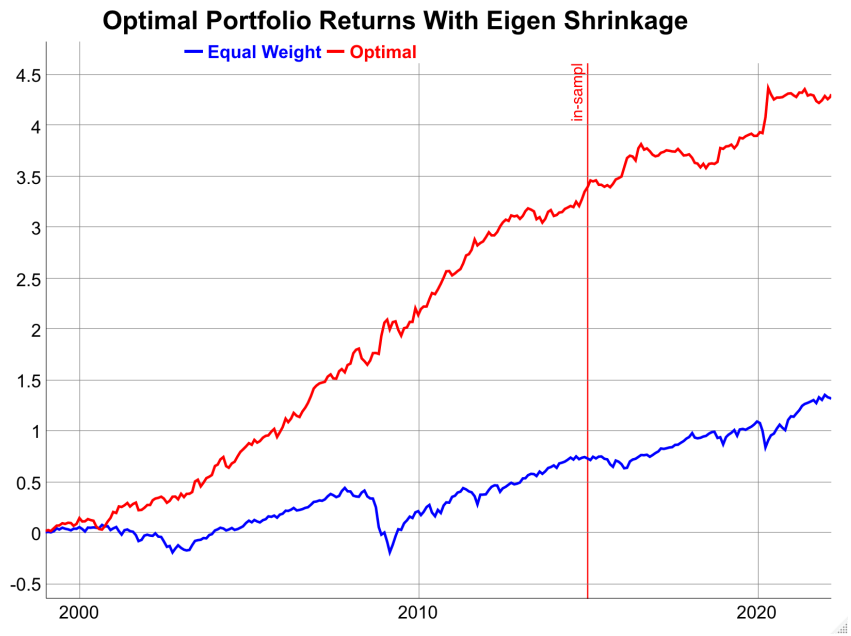
```
> # Calculate in-sample covariance matrix
> covmat <- cov(retis)
> eigend <- eigen(covmat)
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> # Calculate dimension reduction inverse of covariance matrix
> dimax <- 3
> covinv <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
> # Verify inverse property of inverse
> all.equal(covmat, covmat %*% covinv %*% covmat)
```

Portfolio Optimization for ETFs with Dimension Reduction

The *out-of-sample* performance of the *portfolio optimization* strategy is greatly improved by shrinking the inverse of the covariance matrix.

The *in-sample* performance is worse because shrinkage reduces *overfitting*.

```
> # Calculate portfolio weights
> weightv <- invmat %*% colMeans(retis)
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retp)
> # Calculate portfolio returns
> insample <- xts::xts(retis %*% weightv, zoo::index(retis))
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
```




```
> # Plot cumulative portfolio returns
> pnls <- rbind(insample, outsample)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Optimal Portfolio Returns With Dimension Reduction") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", stroke
+   dyLegend(width=300)
```

Portfolio Optimization With Return Shrinkage

To further reduce the statistical noise, the individual returns r_i can be *shrunk* to the average portfolio returns $\bar{r}$:

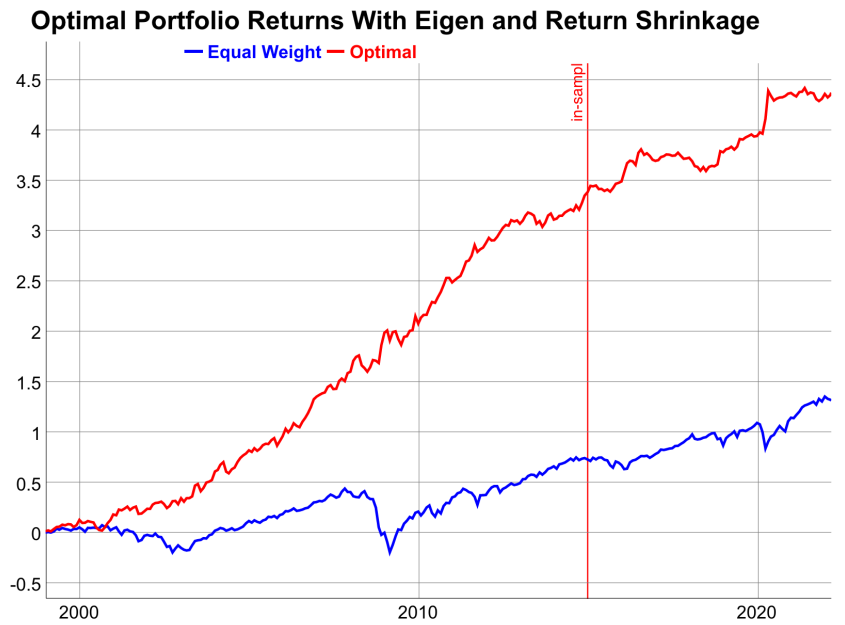
$$r_i' = (1 - \alpha)\, r_i + \alpha\, \bar{r}$$

The parameter α is the *shrinkage* intensity, and it determines the strength of the *shrinkage* of individual returns to their mean.

If $\alpha = 0$ then there is no *shrinkage*, while if $\alpha = 1$ then all the returns are *shrunk* to their common mean: $r_i = \bar{r}$.

The optimal value of the *shrinkage* intensity α can be determined using *backtesting* (*cross-validation*).

```
> # Shrink the in-sample returns to their mean
> alpha <- 0.7
> retxm <- rowMeans(retx["/2014"])
> retxis <- (1-alpha)*retx["/2014"] + alpha*retxm
> # Calculate portfolio weights
> weightv <- invmat %*% colMeans(retxis)
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> # Calculate portfolio returns
> insample <- xts::xts(retis %*% weightv, zoo::index(retis))
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
```



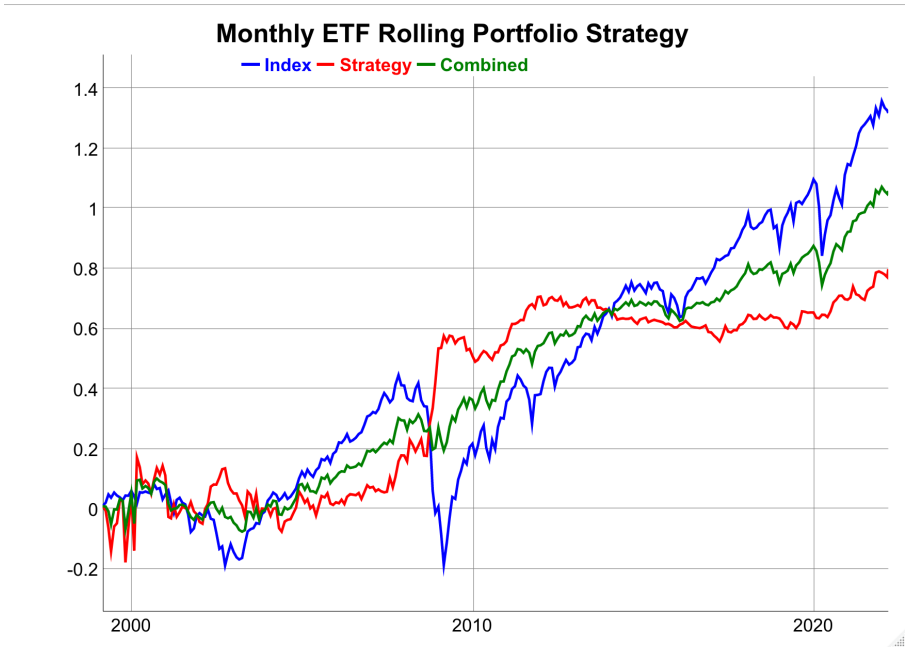

```
> # Plot cumulative portfolio returns
> pnls <- rbind(insample, outsample)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Optimal Portfolio Returns With Eigen and Return Shrinkage"
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeP
+   dyLegend(width=300)
```

Rolling Portfolio Optimization Strategy

In a *rolling portfolio optimization strategy*, the portfolio is optimized periodically and held out-of-sample.

- Calculate the *end points* for portfolio rebalancing,
- Define an objective function for optimizing the portfolio weights,
- Calculate the optimal portfolio weights from the past (in-sample) performance,
- Calculate the out-of-sample returns by applying the portfolio weights to the future returns.

```
> # Define monthly end points
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> endd <- endd[endd > (nstocks+1)]
> npts <- NROW(endd)
> look_back <- 3
> startp <- c(rep_len(0, look_back), endd[1:(npts-look_back)])
> # Perform loop over end points
> pnls <- lapply(2:npts, function(ep) {
+     # Calculate the portfolio weights
+     insample <- retx[startp[ep-1]:endd[ep-1], ]
+     invmat <- MASS::ginv(cov(insample))
+     weightv <- invmat %*% colMeans(insample)
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the out-of-sample portfolio returns
+     outsample <- retp[(endd[ep-1]+1):endd[ep], ]
+     xts::xts(outsample %*% weightv, zoo::index(outsample))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```



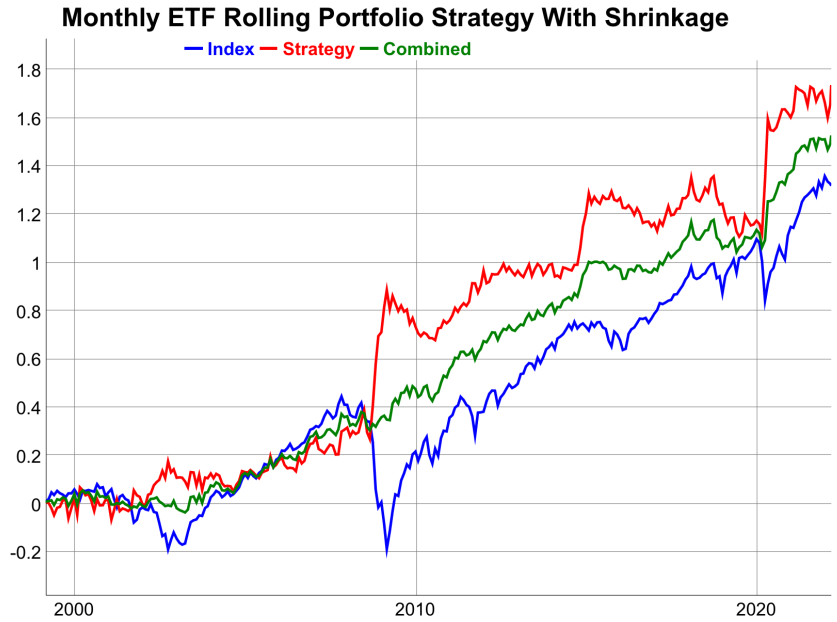

```
> # Plot dygraph of rolling ETF portfolio strategy
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Monthly ETF Rolling
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Portfolio Strategy With Dimension Reduction

The rolling portfolio optimization strategy with dimension reduction performs better than the standard strategy because dimension reduction suppresses the data noise.

The strategy performs especially well during sharp market selloffs, like in the years 2008 and 2020.

```
> # Define monthly end points
> look_back <- 3; dimax <- 9
> startp <- c(rep_len(0, look_back), endd[1:(npts-look_back)])
> # Perform loop over end points
> pnls <- lapply(2:npts, function(ep) {
+     # Calculate regularized inverse of covariance matrix
+     insample <- retx[startp[ep-1]:endd[ep-1], ]
+     eigend <- eigen(cov(insample))
+     eigenvec <- eigend$vectors
+     eigenval <- eigend$values
+     invmat <- eigenvec[, 1:dimax] %*%
+ (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
+     # Calculate the maximum Sharpe ratio portfolio weights
+     weightv <- invmat %*% colMeans(insample)
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the out-of-sample portfolio returns
+     outsample <- retp[(endd[ep-1]+1):endd[ep], ]
+     xts::xts(outsample %*% weightv, zoo::index(outsample))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```



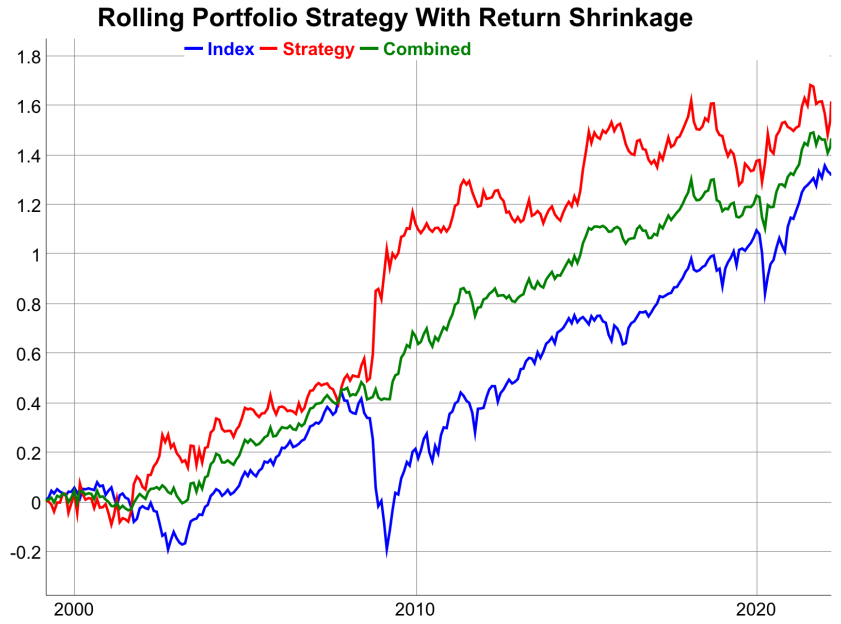

```
> # Plot dygraph of rolling ETF portfolio strategy
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Rolling Portfolio S
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Portfolio Strategy With Return Shrinkage

The rolling portfolio optimization strategy with return shrinkage performs better than the standard strategy because return shrinkage suppresses the data noise.

The strategy performs especially well during sharp market selloffs, like in the years 2008 and 2020.

```
> # Define the return shrinkage intensity
> alpha <- 0.7
> # Perform loop over end points
> pnls <- lapply(2:npts, function(ep) {
+     # Calculate regularized inverse of covariance matrix
+     insample <- retx[startp[ep-1]:endd[ep-1], ]
+     eigend <- eigen(cov(insample))
+     eigenvec <- eigend$vectors
+     eigenval <- eigend$values
+     invmat <- eigenvec[, 1:dimax] %*%
+ (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
+     # Shrink the in-sample returns to their mean
+     insample <- (1-alpha)*insample + alpha*rowMeans(insample)
+     # Calculate the maximum Sharpe ratio portfolio weights
+     weightv <- invmat %*% colMeans(insample)
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the out-of-sample portfolio returns
+     outsample <- retp[(endd[ep-1]+1):endd[ep], ]
+     xts::xts(outsample %*% weightv, zoo::index(outsample))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```




```
> # Plot dygraph of rolling ETF portfolio strategy
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Rolling Portfolio S
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

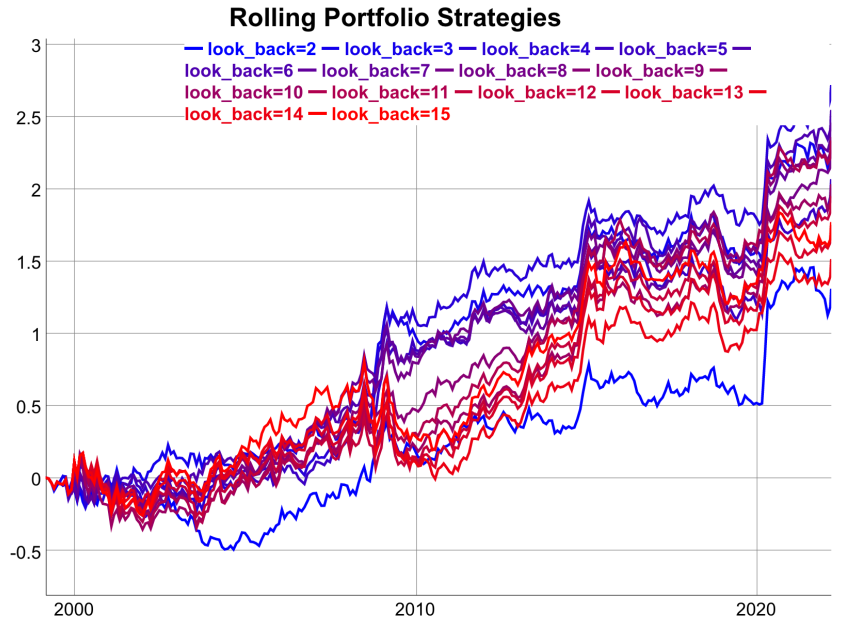
Function for Rolling Portfolio Optimization Strategy

```
> # Define backtest functional for rolling portfolio strategy
> roll_portf <- function(excess, # Excess returns
+                        returns, # Stock returns
+                        endd, # End points
+                        look_back=12, # Look-back interval
+                        dimax=3, # Dimension reduction intensity
+                        alpha=0.0, # Return shrinkage intensity
+                        bid_offer=0.0, # Bid-offer spread
+                        ...) {
+   npts <- NROW(endd)
+   startp <- c(rep_len(0, look_back), endd[1:(npts-look_back)])
+   pnls <- lapply(2:npts, function(ep) {
+     # Calculate regularized inverse of covariance matrix
+     insample <- excess[startp[ep-1]:endd[ep-1], ]
+     eigend <- eigen(cov(insample))
+     eigenvec <- eigend$vectors
+     eigenval <- eigend$values
+     invmat <- eigenvec[, 1:dimax] %*%
+ (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
+     # Shrink the in-sample returns to their mean
+     insample <- (1-alpha)*insample + alpha*rowMeans(insample)
+     # Calculate the maximum Sharpe ratio portfolio weights
+     weightv <- invmat %*% colMeans(insample)
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the out-of-sample portfolio returns
+     outsample <- returns[(endd[ep-1]+1):endd[ep], ]
+     xts::xts(outsample %*% weightv, zoo::index(outsample))
+   })  # end lapply
+   pnls <- do.call(rbind, pnls)
+   # Add warmup period to pnls
+   rbind(indeks[paste0("/", start(pnls)-1)], pnls)
+ }  # end roll_portf
```

Rolling Portfolio Optimization With Different Look-backs

Multiple *rolling portfolio optimization* strategies can be backtested by calling the function `roll_portf()` in a loop over a vector of *look-back* parameters.

```
> # Simulate a monthly ETF momentum strategy
> pnls <- roll_portf(excess=retx, returns=retp, endd=endd,
+   look_back=look_back, dimax=dimax)
> # Perform sapply loop over look_backs
> look_backs <- seq(2, 15, by=1)
> pnls <- lapply(look_backs, roll_portf,
+   returns=retp, excess=retx, endd=endd, dimax=dimax)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("look_back=", look_backs)
> pnlsums <- sapply(pnls, sum)
> look_back <- look_backs[which.max(pnlsums)]
```

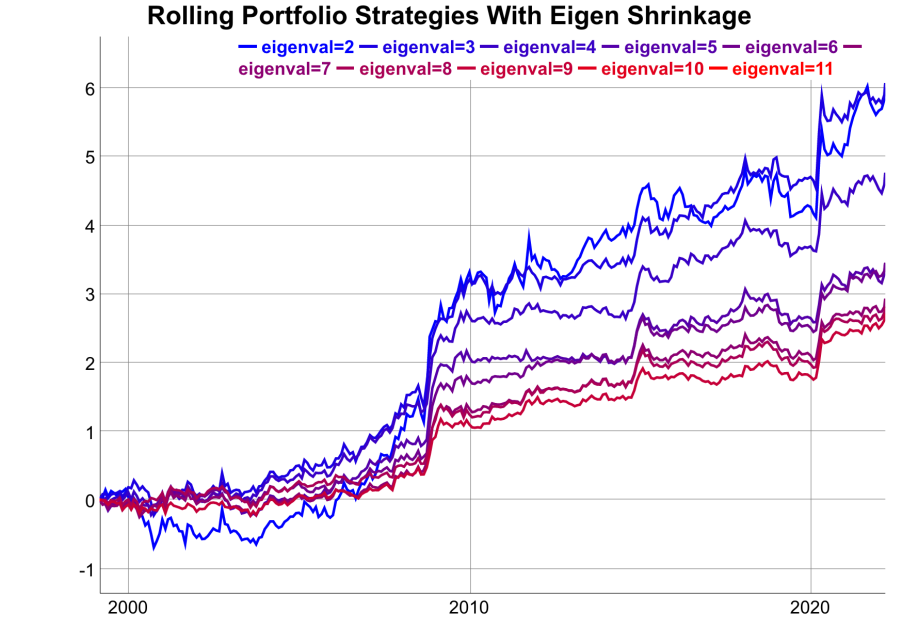



```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Plot EWMA strategies using quantmod
> plot_theme <- chart_theme()
> plot_theme$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls),
+   theme=plot_theme, name="Rolling Portfolio Strategies")
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retp)),
+   col=plot_theme$col$line.col, bty="n")
```

Rolling Portfolio Optimization With Different Dimension Reduction

Multiple *rolling portfolio optimization* strategies can be backtested by calling the function `roll_portf()` in a loop over a vector of the dimension reduction parameter.

```
> # Perform backtest for different dimax values
> eigenvals <- 2:11
> pnls <- lapply(eigenvals, roll_portf, excess=retx,
+   returns=retp, endd=endd, look_back=look_back)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("eigenval=", eigenvals)
> pnlsums <- sapply(pnls, sum)
> dimax <- eigenvals[which.max(pnlsums)]
```



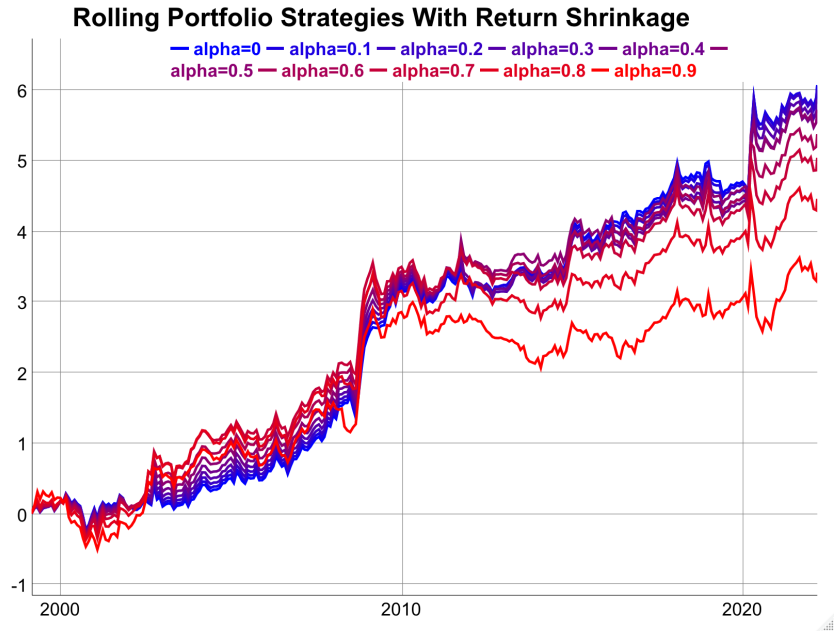

```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies With Dimension Reduction") %>
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Plot EWMA strategies using quantmod
> plot_theme <- chart_theme()
> plot_theme$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls),
+   theme=plot_theme, name="Rolling Portfolio Strategies")
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retp)),
+   col=plot_theme$col$line.col, bty="n")
```

Rolling Portfolio Optimization With Different Return Shrinkage

Multiple *rolling portfolio optimization* strategies can be backtested by calling the function `roll_portf()` in a loop over a vector of return shrinkage parameters.

The best return shrinkage parameter for ETFs is equal to 0, which means no return shrinkage.

```
> # Perform backtest over vector of return shrinkage intensities
> alphav <- seq(from=0.0, to=0.9, by=0.1)
> pnls <- lapply(alphav, roll_portf, excess=retx,
+   returns=retp, endd=endd, look_back=look_back, dimax=dimax)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("alpha=", alphav)
> pnlsums <- sapply(pnls, sum)
> alpha <- alphav[which.max(pnlsums)]
```



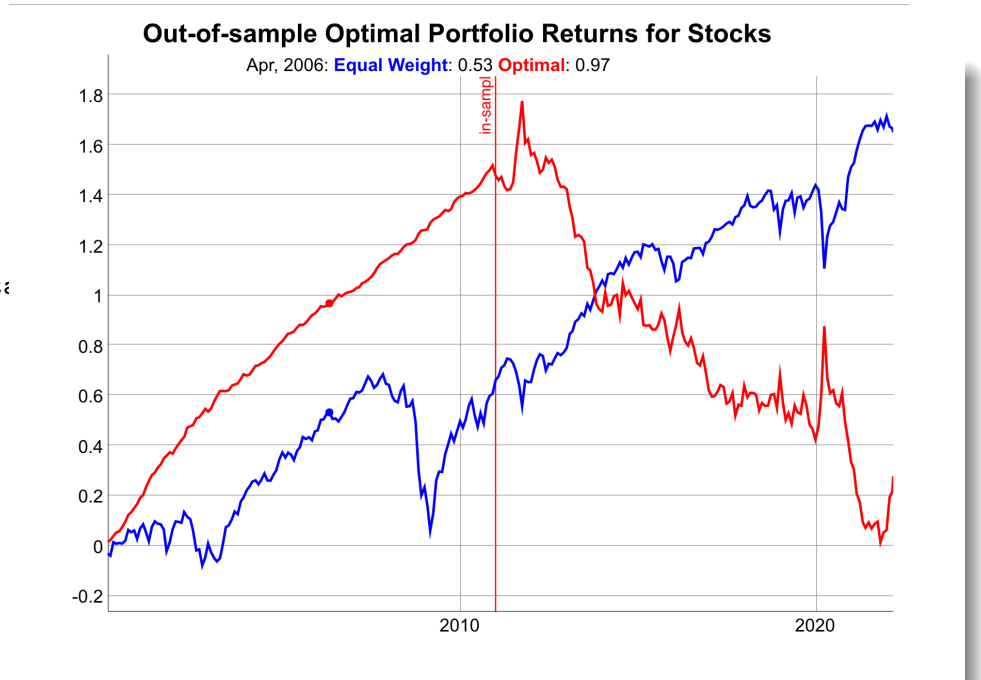

```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies With Return Shrinkage") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Plot EWMA strategies using quantmod
> plot_theme <- chart_theme()
> plot_theme$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls),
+   theme=plot_theme, name="Rolling Portfolio Strategies")
> legend("bottomleft", legend=colnames(pnls),
+   inset=0.02, bg="white", cex=0.7, lwd=rep(6, NCOL(retp)),
+   col=plot_theme$col$line.col, bty="n")
```

Portfolio Optimization Strategy for Stocks

The *portfolio optimization* strategy for stocks is *overfitted* in the *in-sample* interval.

Therefore the strategy completely fails in the *out-of-sample* interval.

```
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RDat
> # Overwrite NA values in returns
> retp <- returns["2000/"]
> nstocks <- NCOL(retp)
> retp[1, is.na(retp[1, ])] <- 0
> retp <- zoo::na.locf(retp, na.rm=FALSE)
> datev <- zoo::index(retp)
> riskf <- 0.03/252
> retx <- (retp - riskf)
> retis <- retp["/2010"]
> retsos <- retp["2011/"]
> # Maximum Sharpe weights in-sample interval
> covmat <- cov(retis)
> invmat <- MASS::ginv(covmat)
> weightv <- invmat %*% colMeans(retx["/2010"])
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retp)
> # Calculate portfolio returns
> insample <- xts::xts(retis %*% weightv, zoo::index(retis))
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
> indeks <- xts::xts(rowMeans(retp), datev)
```



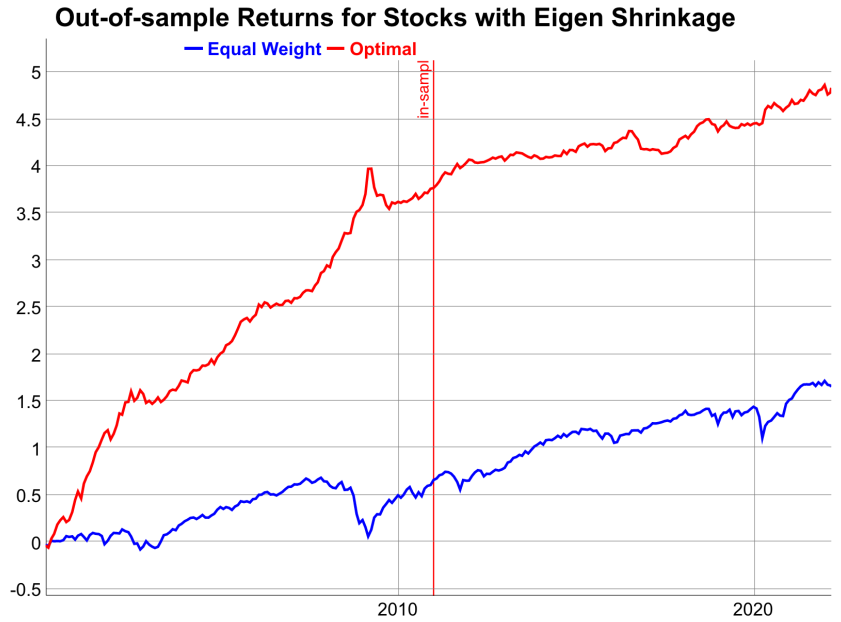

```
> # Combine in-sample and out-of-sample returns
> pnls <- rbind(insample, outsample)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(pnls[index(outsample)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative portfolio returns
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Out-of-sample Optimal Portfolio Returns for Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeP
+   dyLegend(width=300)
```

Portfolio Optimization for Stocks with Dimension Reduction

The *out-of-sample* performance of the *portfolio optimization* strategy is greatly improved by shrinking the inverse of the covariance matrix.

The *in-sample* performance is worse because shrinkage reduces *overfitting*.

```
> # Calculate regularized inverse of covariance matrix
> look_back <- 8; dimax <- 21
> eigend <- eigen(cov(retis))
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> invmat <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
> # Calculate portfolio weights
> weightv <- invmat %*% colMeans(retx["/2010"])
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retp)
> # Calculate portfolio returns
> insample <- xts::xts(retis %*% weightv, zoo::index(retis))
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
> indeks <- xts::xts(rowMeans(retp), datev)
```




```
> # Combine in-sample and out-of-sample returns
> pnls <- rbind(insample, outsample)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(pnls[index(outsample)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative portfolio returns
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Out-of-sample Returns for Stocks with Dimension Reduction"
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeP
+   dyLegend(width=300)
```

Optimal Stock Portfolio Weights With Return Shrinkage

To further reduce the statistical noise, the individual returns r_i can be *shrunk* to the average portfolio returns $\bar{r}$:

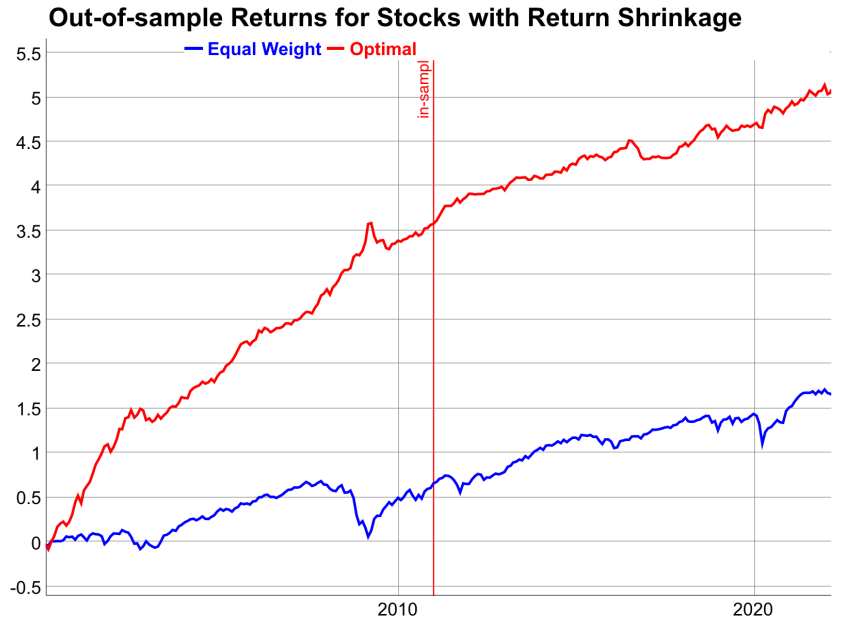
$$r_i' = (1 - \alpha)\, r_i + \alpha\, \bar{r}$$

The parameter α is the *shrinkage* intensity, and it determines the strength of the *shrinkage* of individual returns to their mean.

If $\alpha = 0$ then there is no *shrinkage*, while if $\alpha = 1$ then all the returns are *shrunk* to their common mean: $r_i = \bar{r}$.

The optimal value of the *shrinkage* intensity α can be determined using *backtesting* (*cross-validation*).

```
> # Shrink the in-sample returns to their mean
> alpha <- 0.7
> retxm <- rowMeans(retx["/2010"])
> retxis <- (1-alpha)*retx["/2010"] + alpha*retxm
> # Calculate portfolio weights
> weightv <- invmat %*% colMeans(retxis)
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> # Calculate portfolio returns
> insample <- xts::xts(retis %*% weightv, zoo::index(retis))
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
```




```
> # Combine in-sample and out-of-sample returns
> pnls <- rbind(insample, outsample)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(pnls[index(outsample)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative portfolio returns
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Out-of-sample Returns for Stocks with Return Shrinkage") %
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeP
+   dyLegend(width=300)
```

Fast Covariance Matrix Inverse Using *RcppArmadillo*

RcppArmadillo can be used to quickly calculate the regularized inverse of a covariance matrix.

```
> library(RcppArmadillo)
> # Source Rcpp functions from file
> Rcpp::sourceCpp("/Users/jerzy/Develop/lecture_slides/scripts/back
> # Create random matrix of returns
> matrixv <- matrix(rnorm(300), nc=5)
> # Regularized inverse of covariance matrix
> dimax <- 4
> eigend <- eigen(cov(matrixv))
> covinv <- eigend$vectors[, 1:dimax] %*%
+   (t(eigend$vectors[, 1:dimax]) / eigend$values[1:dimax])
> # Regularized inverse using RcppArmadillo
> covinv_arma <- calc_inv(matrixv, dimax)
> all.equal(covinv, covinv_arma)
> # Microbenchmark RcppArmadillo code
> library(microbenchmark)
> summary(microbenchmark(
+   rcode={eigend <- eigen(cov(matrixv))
+     eigend$vectors[, 1:dimax] %*%
+ (t(eigend$vectors[, 1:dimax]) / eigend$values[1:dimax])
+   },
+   cppcode=calc_inv(matrixv, dimax),
+   times=100))[, c(1, 4, 5)]  # end microbenchmark summary
```

```
arma::mat calc_inv(const arma::mat& matrixv,
                   arma::uword dimax = 0, // Max number
                   double eigen_thresh = 0.01) { // Thre

  if (dimax == 0) {
    // Calculate the inverse using arma::pinv()
    return arma::pinv(tseries, eigen_thresh);
  } else {
    // Calculate the regularized inverse using SVD decom

    // Allocate SVD
    arma::vec svdval;
    arma::mat svdu, svdv;

    // Calculate the SVD
    arma::svd(svdu, svdval, svdv, tseries);

    // Subset the SVD
    dimax = dimax - 1;
    // For no regularization: dimax = tseries.n_cols
    svdu = svdu.cols(0, dimax);
    svdv = svdv.cols(0, dimax);
    svdval = svdval.subvec(0, dimax);

    // Calculate the inverse from the SVD
    return svdv*arma::diagmat(1/svdval)*svdu.t();

  }  // end if

}  // end calc_inv
```

Portfolio Optimization Using *RcppArmadillo*

Fast portfolio optimization using matrix algebra can be implemented using *RcppArmadillo*.

```
arma::vec calc_weights(const arma::mat& returns, // Portfolio returns
                       stdev::string method = "ranksharpe",
                       double eigen_thresh = 0.001,
                       arma::uword dimax = 0,
                       double confi = 0.1,
                       double alpha = 0.0,
                       bool scale = true,
                       double vol_target = 0.01) {
  // Initialize
  arma::vec weightv(returns[ncols, fill::zeros);
  if (dimax == 0)  dimax = returns[ncols;

  // Switch for the different methods for weights
  switch(calc_method(method)) {
  case method::ranksharpe: {
    // Mean returns by columns
    arma::vec meancols = arma::trans(arma::mean(returns, 0));
    // Standard deviation by columns
    arma::vec sd_cols = arma::trans(arma::stddev(returns, 0));
    sd_cols.replace(0, 1);
    meancols = meancols/sd_cols;
    // Weights equal to ranks of Sharpe
    weightv = conv_to<vec>::from(arma::sort_index(arma::sort_index(meancols)));
    weightv = (weightv - arma::mean(weightv));
    break;
  }  // end ranksharpe
  case method::max_sharpe: {
    // Mean returns by columns
    arma::vec meancols = arma::trans(arma::mean(returns, 0));
    // Shrink meancols to the mean of returns
    meancols = ((1-alpha)*meancols + alpha*arma::mean(meancols));
    // Apply regularized inverse
    // arma::mat inverse = calc_inv(cov(returns), dimax);
    // weightv = calc_inv(cov(returns), dimax)*meancols;
    weightv = calc_inv(cov(returns), dimax, eigen_thresh)*meancols;
```

Strategy Backtesting Using *RcppArmadillo*

Fast backtesting of strategies can be implemented using *RcppArmadillo*.

```
arma::mat back_test(const arma::mat& retx, // Portfolio excess returns
                    const arma::mat& returns, // Portfolio returns
                    arma::uvec startp,
                    arma::uvec endd,
                    stdev::string method = "ranksharpe",
                    double eigen_thresh = 0.001,
                    arma::uword dimax = 0,
                    double confi = 0.1,
                    double alpha = 0.0,
                    bool scale = true,
                    double vol_target = 0.01,
                    double coeff = 1.0,
                    double bid_offer = 0.0) {

  arma::vec weightv(returns[ncols, fill::zeros);
  arma::vec weights_past = zeros(returns[ncols);
  arma::mat pnls = zeros(returns*nrows, 1);

  // Perform loop over the end points
  for (arma::uword it = 1; it < endd.size(); it++) {
    // cout << "it: " << it << endl;
    // Calculate portfolio weights
    weightv = coeff*calc_weights(retx.rows(startp(it-1), endd(it-1)), method, dimax, eigen_thresh, confi, alpha
    // Calculate out-of-sample returns
    pnls.rows(endd(it-1)+1, endd(it)) = returns.rows(endd(it-1)+1, endd(it))*weightv;
    // Add transaction costs
    pnls.row(endd(it-1)+1) -= bid_offer*sum(abs(weightv - weights_past))/2;
    weights_past = weightv;
  }  // end for

  // Return the strategy pnls
  return pnls;

}  // end back_test
```

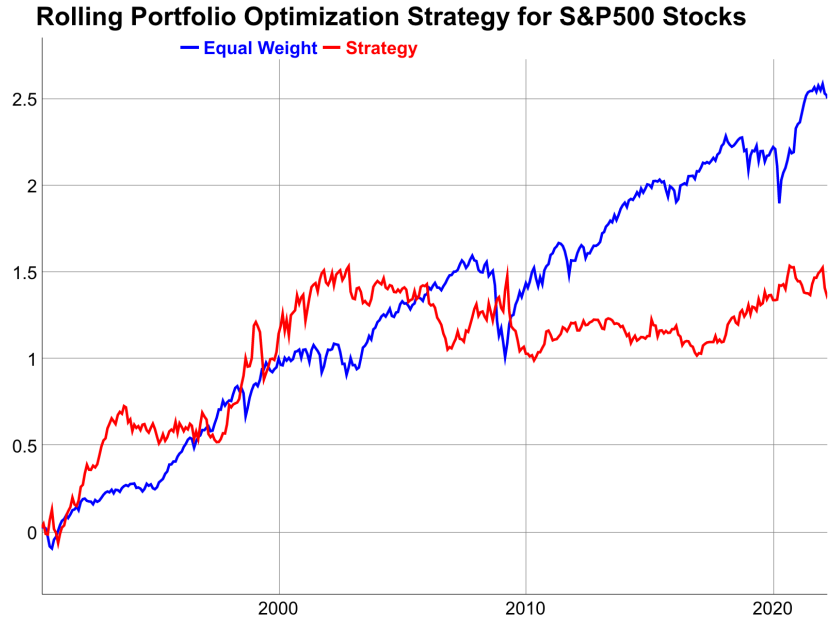
Rolling Portfolio Optimization Strategy for S&P500 Stocks

A *rolling portfolio optimization* strategy consists of rebalancing a portfolio over the end points:

1. Calculate the maximum Sharpe ratio portfolio weights at each end point,
2. Apply the weights in the next interval and calculate the out-of-sample portfolio returns.

The strategy parameters are: the rebalancing frequency (annual, monthly, etc.), and the length of look-back interval.

```
> # Overwrite NA values in returns
> retp <- returns100
> retp[1, is.na(retp[1, ])] <- 0
> retp <- zoo::na.locf(retp, na.rm=FALSE)
> retx <- (retp - riskf)
> nstocks <- NCOL(retp) ; datev <- zoo::index(retp)
> # Define monthly end points
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> endd <- endd[endd > (nstocks+1)]
> npts <- NROW(endd) ; look_back <- 12
> startp <- c(rep_len(0, look_back), endd[1:(npts-look_back)])
> # Perform loop over end points - takes very long !!!
> pnls <- lapply(2:npts, function(ep) {
+     # Subset the excess returns
+     insample <- retx[startp[ep-1]:endd[ep-1], ]
+     invmat <- MASS::ginv(cov(insample))
+     # Calculate the maximum Sharpe ratio portfolio weights
+     weightv <- invmat %*% colMeans(insample)
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the out-of-sample portfolio returns
+     outsample <- retp[(endd[ep-1]+1):endd[ep], ]
+     xts::xts(outsample %*% weightv, zoo::index(outsample))
+ })  # end lapply
```

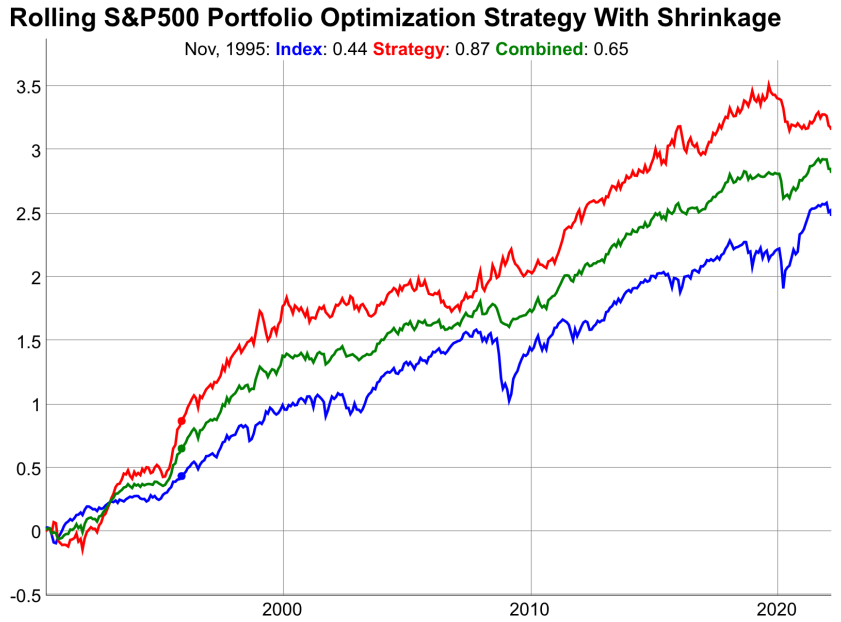



```
> # Calculate returns of equal weight portfolio
> indeks <- xts::xts(rowMeans(retp), datev)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls*sd(indeks)/
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> colnames(wealthv) <- c("Equal Weight", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot cumulative strategy returns
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Rolling Portfolio O
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Portfolio Optimization Strategy With Shrinkage

The *rolling portfolio optimization* strategy can be improved by applying both dimension reduction and return shrinkage.

```
> # Shift end points to C++ convention
> endd <- (endd - 1)
> endd[endd < 0] <- 0
> startp <- (startp - 1)
> startp[startp < 0] <- 0
> # Specify dimension reduction and return shrinkage using list of p
> controlv <- HighFreq::param_portf(method="maxsharpe", dimax=21, al
> # Perform backtest in Rcpp
> pnls <- HighFreq::back_test(excess=retx, returns=retp,
+   startp=startp, endd=endd, controlv=controlv)
> pnls <- pnls*sd(indeks)/sd(pnls)
```




```
> # Plot cumulative strategy returns
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Rolling S&P500 Port
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

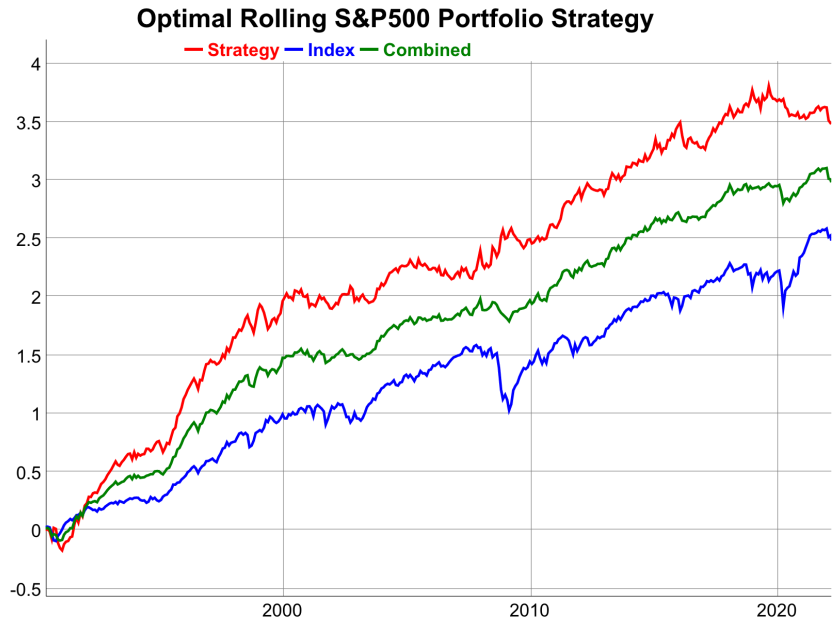
Determining Shrinkage Parameters Using Backtesting

The optimal values of the dimension reduction parameter dimax and the return shrinkage intensity parameter α can be determined using *backtesting*.

The best dimension reduction parameter for this portfolio of stocks is equal to dimax=33, which means relatively weak dimension reduction.

The best return shrinkage parameter for this portfolio of stocks is equal to $\alpha = 0.81$, which means strong return shrinkage.

```
> # Perform backtest over vector of return shrinkage intensities
> alphav <- seq(from=0.01, to=0.91, by=0.1)
> pnls <- lapply(alphav, function(alpha) {
+   HighFreq::back_test(excess=retx, returns=retp,
+   startp=startp, endd=endd, controlv=controlv)
+ })  # end lapply
> profilev <- sapply(pnls, sum)
> plot(x=alphav, y=profilev, t="l",
+   main="Rolling Strategy as Function of Return Shrinkage",
+   xlab="Shrinkage Intensity Alpha", ylab="pnl")
> whichmax <- which.max(profilev)
> alpha <- alphav[whichmax]
> pnls <- pnls[[whichmax]]
> # Perform backtest over vector of dimension reduction eigenvals
> eigenvals <- seq(from=3, to=40, by=2)
> pnls <- lapply(eigenvals, function(dimax) {
+   HighFreq::back_test(excess=retx, returns=retp,
+     startp=startp, endd=endd, controlv=controlv)
+ })  # end lapply
> profilev <- sapply(pnls, sum)
```



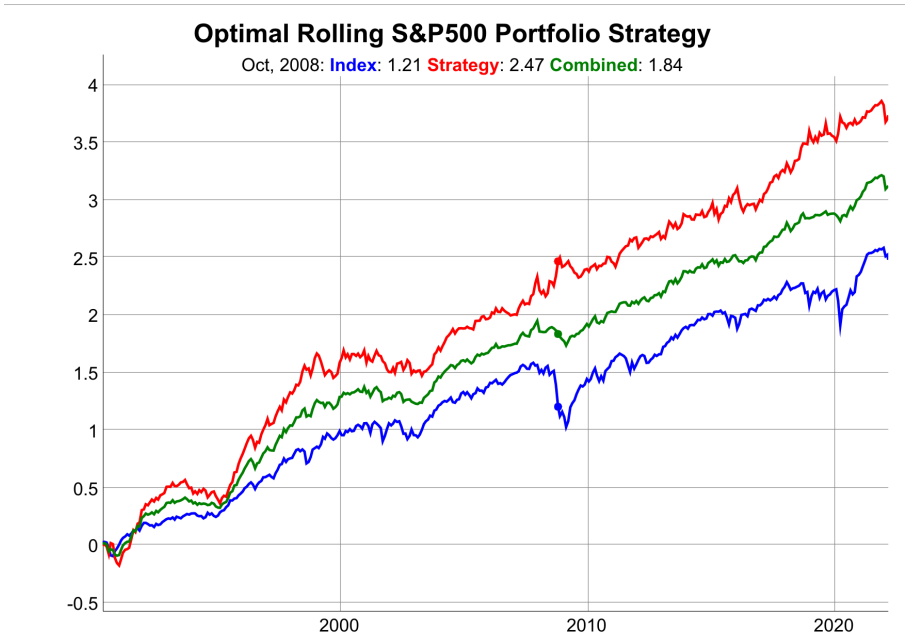

```
> plot(x=eigenvals, y=profilev, t="l",
+   main="Strategy PnL as Function of dimax",
+   xlab="dimax", ylab="pnl")
> whichmax <- which.max(profilev)
> dimax <- eigenvals[whichmax]
> pnls <- pnls[[whichmax]]
> pnls <- pnls*sd(indeks)/sd(pnls)
> # Plot cumulative strategy returns
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Optimal Rolling S&P
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Determining Look-back Interval Using Backtesting

The optimal value of the look-back interval can be determined using *backtesting*.

The optimal value of the look-back interval for this portfolio of stocks is equal to `look_back=9` months, which roughly agrees with the research literature on momentum strategies.

```
> # Perform backtest over look-backs
> look_backs <- seq(from=3, to=12, by=1)
> pnls <- lapply(look_backs, function(look_back) {
+   startp <- c(rep_len(0, look_back), endd[1:(npts-look_back)])
+   startp <- (startp - 1)
+   startp[startp < 0] <- 0
+   HighFreq::back_test(excess=retx, returns=retp,
+     startp=startp, endd=endd, controlv=controlv)
+ })  # end lapply
> profilev <- sapply(pnls, sum)
> plot(x=look_backs, y=profilev, t="l", main="Strategy PnL as Funct
+   xlab="Look-back Interval", ylab="pnl")
> whichmax <- which.max(profilev)
> look_back <- look_backs[whichmax]
> pnls <- pnls[[whichmax]]
> pnls <- pnls*sd(indeks)/sd(pnls)
```




```
> # Plot cumulative strategy returns
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Optimal Rolling S&P
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Homework Assignment

Required

Study all the lecture slides in `FRE7241_Lecture_6.pdf`, and run all the code in `FRE7241_Lecture_6.R`

Recommended

- Read about *estimator shrinkage*:
  Aswani Regression Shrinkage Bias Variance Tradeoff.pdf
  Blei Regression Lasso Shrinkage Bias Variance Tradeoff.pdf
- Read about *optimization methods*:
  Bolker Optimization Methods.pdf
  Yollin Optimization.pdf
  DEoptim Introduction.pdf
  Ardia DEoptim Portfolio Optimization.pdf
  Boudt DEoptim Portfolio Optimization.pdf
  Boudt DEoptim Large Portfolio Optimization.pdf
  Mullen Package DEoptim.pdf
- Read about *momentum*:
  Bouchaud Momentum Mean Reversion Equity Returns.pdf