

# FRE7241 Algorithmic Portfolio Management

## Lecture #6, Spring 2024

Jerzy Pawlowski [jp3900@nyu.edu](mailto:jp3900@nyu.edu)

NYU Tandon School of Engineering

April 23, 2024



# Optimal Stock Weights

If an investor's utility of wealth is logarithmic:  
 $u(w) = \log(w)$ , then to maximize their utility, they should invest the fraction  $k_f$  of their wealth in a stock:

$$k_f = \frac{\bar{r}}{\sigma^2}$$

Where  $\bar{r}$  is the expected stock return and  $\sigma^2$  is the expected variance. The fraction  $k_f$  is called the *Kelly fraction* or the *Kelly ratio*.

The expected *logarithmic utility*  $u$  can be expanded to second order as:

$$\begin{aligned} u &= \mathbb{E}[\log(1 + k_f r)] = \mathbb{E}[(k_f r - \frac{(k_f r)^2}{2})] = \\ &= k_f \bar{r} - \frac{k_f^2 \sigma^2}{2} \end{aligned}$$

The *Kelly ratio* which maximizes the utility is found by equating the derivative of utility to zero:

$$\frac{du}{dk_f} = \bar{r} - k_f \sigma^2 = 0 \rightarrow k_f = \frac{\bar{r}}{\sigma^2}$$

```
> # Objective function equal to the sum of returns
> objfun <- function(retp) sum(na.omit(retp))
> # Objective function equal to the Sharpe ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   if (NROW(retp) > 12) {
+     stdev <- sd(retp)
+     if (stdev > 0) mean(retp)/stdev else 0
+   } else 0
+ } # end objfun
> # Objective function equal to the Kelly ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   if (NROW(retp) > 12) {
+     varv <- var(retp)
+     if (varv > 0) mean(retp)/varv else 0
+   } else 0
+ } # end objfun
```

In portfolio optimization, assuming zero correlations, the *maximum Sharpe* portfolio weights are proportional to the *Kelly ratios*:

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

This is also a consequence of logarithmic utility.

# 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.RData")
> # Select returns of 100 stocks
> retp <- retstock100["2000/"]
> datev <- zoo::index(retp) # Dates vector
> nrows <- NROW(retp) # number of rows
> nstocks <- NCOL(retp) # number of stocks
> # Objective function equal to the Kelly ratio
> objfun <- function(retp) {
+   retp <- na.omit(retp)
+   if (NROW(retp) > 12) {
+     varv <- var(retp)
+     if (varv > 0) mean(retp)/varv else 0
+   } else 0
+ } # end objfun
> # Calculate performance statistics for all stocks
> perfstat <- sapply(retp, objfun)
> sum(is.na(perfstat))
> sum(!is.finite(perfstat))
> hist(perfstat)
> sort(perfstat, decreasing=TRUE)
```

## 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.

```
> # Calculate weights proportional to performance statistic  
> # With quadratic constraint  
> weightv <- perfstat/sqrt(sum(perfstat^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.

```
> # Calculate weights proportional to performance statistic  
> # With linear constraint  
> weightv <- perfstat/sum(perfstat)  
> sum(weightv^2)  
> 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  $\sigma$  is the same as the volatility of the equal weight portfolio  $\sigma_{ew}$ :

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

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  $\sigma$  is equal to a target volatility  $\sigma_t$ :

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

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)*perfstat)
> # Or using Rcpp
> retf <- HighFreq::mult_mat(perfstat, 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*perfstat/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 <- 0.01
> weightv <- volt*perfstat/volis
> retw <- t(t(retp)*weightv)
> all.equal(sd(rowMeans(retw, na.rm=TRUE)), volt)
> # Compare speed of R with Rcpp
> library(microbenchmark)
> summary(microbenchmark(
+   trans=t(t(retp)*perfstat),
+   rcpp=HighFreq::mult_mat(perfstat, 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
```

# 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 the 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)
> pnls[1] <- 0
> # Scale the pnls so their volatility is the same as equal weight
> retew <- rowMeans(retp, na.rm=TRUE)
> retew[1] <- 0
> pnls <- sd(retew)/sd(pnls)*pnls
> wealthv <- xts(cbind(retew, pnls), datev)
> colnames(wealthv) <- c("Equal Weight", "Momentum")
> dygraph(cumsum(wealthv))
```

# 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.

```
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> npts <- NROW(endd)
> # Perform loop over the end points
> lookb <- 8
> pnls <- lapply(3:(npts-1), function(tday) {
+   # Select the look-back returns
+   startp <- endd[max(1, tday-lookb)]
+   retis <- retp[startp:endd[tday], ]
+   # 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 weight
+   weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnls)
+   # Calculate the out-of-sample momentum returns
+   pnls <- HighFreq::mult_mat(weightv, retp[(endd[tday]+1):endd[tda]])
+   pnls <- rowMeans(pnls, na.rm=TRUE)
+   drop(pnls)
+ }) # end lapply
> pnls <- rutils::do_call(c, pnls)
```

# 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  
> retew <- rowMeans(rtp, na.rm=TRUE)  
> # Add initial startup interval to the momentum returns  
> pnls <- c(retew[ennd[1]:ennd[3]], pnls)  
> # Calculate the Sharpe and Sortino ratios  
> wealthv <- cbind(retew, pnls)  
> wealthv <- xts::xts(wealthv, order.by=datev)  
> colnames(wealthv) <- c("Index", "Strategy")  
> 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)[ennd],  
+   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, lookb=12, rebalf="months", topa
+   bidask=0.0, endd=rutils::calc_endpoints(retp, interval=rebalf),
+   # Perform loop over end points
+   npts <- NROW(endd)
+   pnls <- lapply(3:(npts-1), function(tday) {
+     # Select the look-back returns
+     startp <- endd[max(1, tday-lookb)]
+     retis <- retp[startp:endd[tday], ]
+     # 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[tday]+1):endd[1
+     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*.

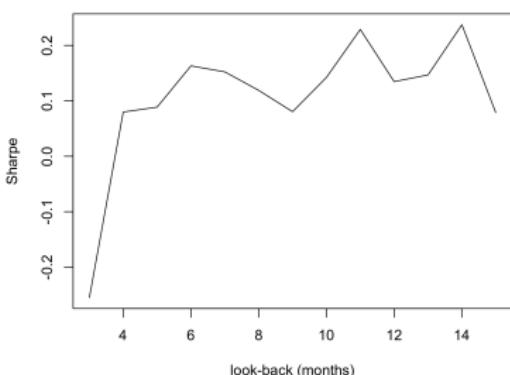
If the *look-back interval* is too long, then the data has large *bias* because the distant past may have little relevance to today.

But if the *look-back interval* is too short, then there's not enough data, and estimates will have high *variance*.

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
> lookbv <- seq(3, 15, by=1)
> endd <- rutils::calc_endpoints(retp, interval="months")
> # Warning - takes very long
> pnll <- lapply(lookbv, 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(lookbv, btmomtop, retp=retp, endd=endd, objfun=objfun, mc.cores=ncores)
> sharper <- sqrt(252)*sapply(pnll, function(pnl) mean(pnl)/sd(pnl))
```

Momentum Sharpe as Function of Look-back Interval



```
> # Plot Sharpe ratios of momentum strategies
> plot(x=lookbv, 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)
> lookbv[whichmax]
> pnls <- pnll[[whichmax]]
> # Add initial startup interval to the momentum returns
> pnls <- c(retew[ennd[1]:ennd[3]], pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("Index", "Strategy")
> 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)[ennd],
+   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, lookb=12, rebalf="months",
+   bidask=0.0, endd=rutils::calc_endpoints(retp, interval=rebalf),
+   # Perform loop over end points
+   npts <- NROW(endd)
+   pnls <- lapply(3:(npts-1), function(tday) {
+     # Select the look-back returns
+     startp <- endd[max(1, tday-lookb)]
+     retis <- retp[startp:endd[tday], ]
+     # 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 wts
+     weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnls)
+     # Calculate the out-of-sample momentum returns
+     pnls <- HighFreq::mult_mat(weightv, retp[(endd[tday]+1):endd[1]])
+     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
> lookbv <- seq(3, 15, by=1)
> pnll <- lapply(lookbv, btmomweight, retp=retp, endd=endd, objfun=
> # Or perform parallel loop under Mac-OSX or Linux
> library(parallel) # Load package parallel
> ncores <- detectCores() - 1
> pnll <- mclapply(lookbv, btmomweight, retp=retp, endd=endd, objf=
> sharper <- sqrt(252)*sapply(pnll, function(pnl) mean(pnl)/sd(pnl))
> # Plot Sharpe ratios of momentum strategies
> plot(x=lookbv, 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)
> lookbv[whichmax]
> pnls <- pnll[[whichmax]]
> # Add initial startup interval to the momentum returns
> pnls <- c(retew[ennd[1]:ennd[3]], pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, 0.5*(retew + pnls))
> wealthv <- xts::xts(wealthv, order.by=dates)
> colnames(wealthv) <- c("Index", "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
> dygraphs::dygraph(cumsum(wealthv)[ennd],
+   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  $\lambda$ .

```
> # To simplify, set NAs to zero
> retp[is.na(retp)] <- 0
> # Calculate the trailing average returns and variance using C++
> lambdaf <- 0.995
> meanm <- HighFreq::run_mean(retp, lambda=lambdaf)
> varm <- HighFreq::run_var(retp, lambda=lambdaf)
> meanm <- ifelse(is.na(meannm), 0, meanm)
> varm <- ifelse(is.na(varm), 1, varm)
> # Calculate the trailing Kelly ratios
> 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, na.rm=TRUE)
> # Calculate the transaction costs
> bidask <- 0.0
> costs <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)))

```

Daily Momentum Strategy for Stocks



```
> # Scale the momentum volatility to the equal weight index
> volew <- sd(retew)
> pnls <- volew*pnls/sd(pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, 0.5*(retew + pnls))
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Index", "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 `btmpmdaily()` simulates a momentum strategy with *daily rebalancing*.

The decay factor  $\lambda$  determines the rate of decay of the weights applied to the returns, with smaller values of  $\lambda$  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 `btmpmdaily()` 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
> btmpmdaily <- function(retp, lambdaf=0.9, trend=1, bidask=0.0,
+   stopifnot("package:quantmod" %in% search()) || require("quantmod")
+   # Calculate the trailing Kelly ratio
+   meanm <- HighFreq::run_mean(retp, lambda=lambdaf)
+   varm <- HighFreq::run_var(retp, lambda=lambdaf)
+   meanm <- ifelse(is.na(meann), 0, meanm)
+   varm <- ifelse(is.na(varm), 1, varm)
+   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, na.rm=TRUE)
+   # Calculate the transaction costs
+   costs <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)))
+   (pnls - costs)
+ } # end btmpmdaily
```

# Multiple Daily Stock Momentum Strategies

Multiple daily momentum strategies can be backtested by calling the function `btdmomdaily()` in a loop over a vector of  $\lambda$  parameters.

The best performing momentum strategies with *daily rebalancing* are with  $\lambda$  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
> lambdav <- seq(0.99, 0.998, 0.001)
> pnls <- sapply(lambdav, btdmomdaily, retp=retp)
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) volew*pnl/sd(pnl))
> colnames(pnls) <- paste0("lambda=", lambdav)
> 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, in order to reduce the variance of the portfolio weights.

Instead of holding the portfolio for only one day, it's held for several days and gradually liquidated. So that 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 factor  $\lambda$  determines the length of the *holding period*. Smaller values of  $\lambda$  produce a faster decay corresponding to a shorter *holding period*, and vice versa.

The optimal value of the  $\lambda$  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, bidask=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))
+ # 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*bidask*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 `btdmomdaily()` in a loop over a vector of  $\lambda$  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 period
> lambdav <- seq(0.99, 0.998, 0.001)
> pnls <- sapply(lambdav, btdmomdailyhold, retp=retp)
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) volew*pnl/sd(pnl))
> colnames(pnls) <- paste0("lambda=", lambdav)
> 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 Holding Period")
> 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)
> lambdav[whichmax]
> pnls <- pnls[, whichmax]
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, 0.5*(retew + pnls))
> colnames(wealthv) <- c("Index", "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="Optimal 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)
```

# 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  $\lambda$  parameters.

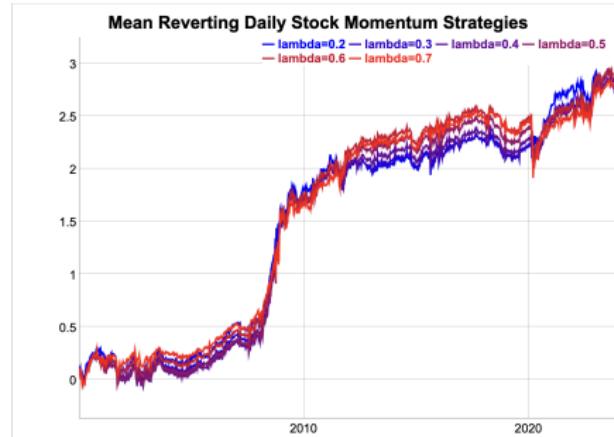
If the argument `trend = -1` then the function `btmomdaily()` simulates a mean-reverting strategy (buys the worst performing stocks and sells the best performing).

The *mean reverting* momentum strategies for the stock constituents perform the best for small  $\lambda$  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 lookbv
> lambdav <- seq(0.2, 0.7, 0.1)
> pnls <- sapply(lambdav, btmomdaily, retp=retp, trend=(-1))
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) volew*pnl/sd(pnl))
> colnames(pnls) <- paste0("lambda=", lambdav)
> 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 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="Mean Reverting Daily Stock Momentum Strategies")
> 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$prices[, c("VTI", "MTUM")])
> wealthv <- rutils:::diffit(log(wealthv))
> 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)
```

## 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 uncorrelated to each other - the correlations of returns are equal to zero.

If the stock returns are uncorrelated then the weights of the *maximum Sharpe* portfolio are proportional to the *Kelly ratios* (the returns divided by their variance):

$$w_i = \mathbb{C}^{-1} \bar{r} = \frac{\bar{r}_i}{\sigma_i^2}$$

Where  $\mathbb{C}$  is the covariance matrix of returns (which is diagonal in this case).

If the momentum weights are chosen to be the *Kelly ratios*, then the momentum strategy is equivalent to the *maximum Sharpe* optimal portfolio strategy.

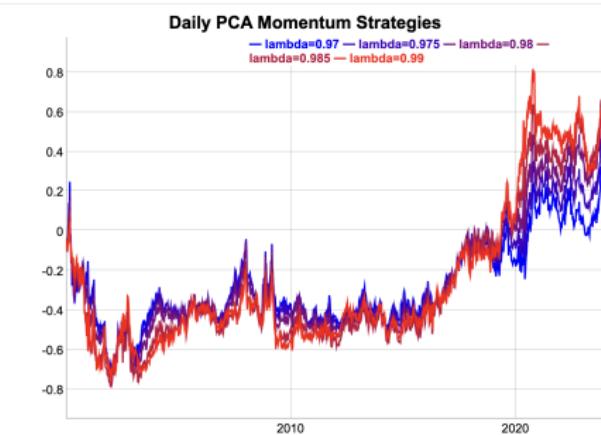
```
> # Calculate the PCA loadings for standardized returns
> retsc <- lapply(retp, function(x) (x - mean(x))/sd(x))
> retsc <- do.call(cbind, retsc)
> covmat <- cov(retsc)
> pcad <- eigen(covmat)
> pcal <- pcad$vectors # The PCA loadings
> rownames(pcal) <- colnames(retp)
> sort(-pcal[, 1], decreasing=TRUE)
> sort(pcal[, 2], decreasing=TRUE)
> round((t(pcal) %*% pcal)[1:5, 1:5], 4)
> # Calculate the PCA time series from the stock returns and the PCAs
> retpca <- retsc %*% pcal
> round((t(retpca) %*% retpca)[1:5, 1:5], 4)
> # Calculate the PCA using prcomp()
> pcad <- prcomp(retsc, center=FALSE, scale=FALSE)
> all.equal(abs(pcad$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 parameters
> dimax <- 11
> lambdav <- seq(0.97, 0.99, 0.005)
> pnls <- mclapply(lambdav, btmomdailyhold, retp=retPCA[, 1:dimax],
+ pnls <- lapply(pnls, function(pnl) volew*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, datev)
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdav, y=sharper, t="l",
+ main="PCA Momentum Sharpe as Function of Decay Factor",
+ xlab="lambda", ylab="Sharpe")
```



```
> # Plot dygraph of daily PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> endd <- rutils::calc_endpoints(pnls, 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 factor  $\lambda$ , so it's able to quickly adjust to changes in market direction.

```
> # Calculate best pnls of PCA momentum strategy
> whichmax <- which.max(sharper)
> lambdadav[whichmax]
> pnls <- pnls[, whichmax]
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, 0.5*(retew + pnls))
> colnames(wealthv) <- c("Index", "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
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal 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)
```

# 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.

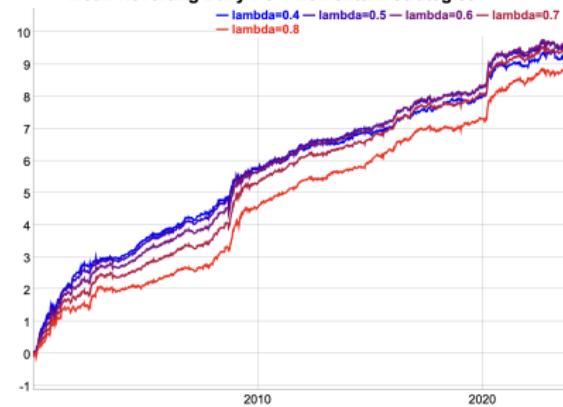
The *mean reverting* momentum strategies had their best performance in periods of high volatility, especially prior to and during the 2008 financial crisis.

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 factors
> lambdav <- seq(0.4, 0.8, 0.1)
> pnls <- mclapply(lambdav, btmomdailyhold, retp=retPCA[, (dimX+1):NCOL(retPCA)],
+   trend=(-1), mc.cores=ncores)
> pnls <- lapply(pnls, function(pnl) volew*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, datev)
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdav, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Factor",
+   xlab="lambda factor", ylab="Sharpe")
```

Mean Reverting Daily PCA Momentum Strategies



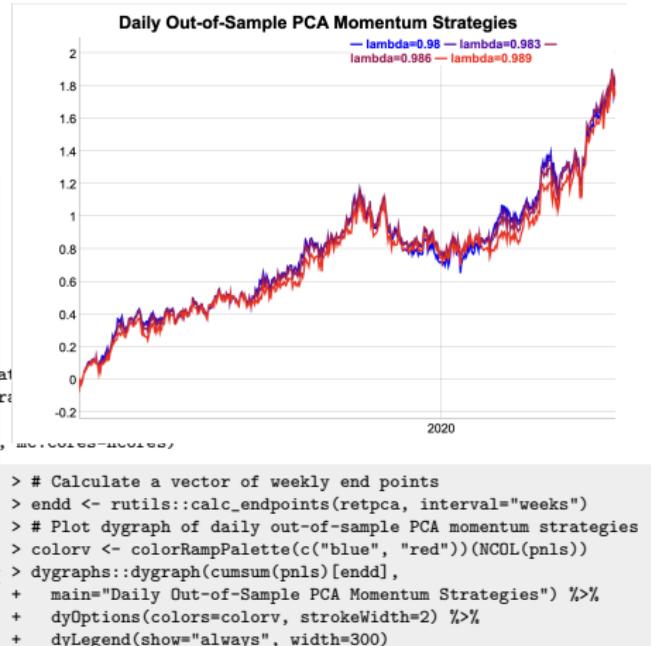
```
> # Plot dygraph of daily PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Mean Reverting Daily PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=400)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

# 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 <- nrow %/% 2
> datev[cutoff]
> insample <- 1:cutoff
> outsample <- (cutoff + 1):nrows
> # Calculate the PCA loadings in-sample
> pcomp <- prcomp(retp[insample])
> # Calculate the out-of-sample PCA time series
> retpca <- xts::xts(retp[outsample] %*% pcomp$rotation, order.by=datev)
> # Simulate daily PCA momentum strategies for multiple lambdaf parameters
> lambdav <- seq(0.98, 0.99, 0.003)
> pnls <- mclapply(lambdav, btmomdailyhold, retp=retpca[, 1:dimmax], method="mle")
> pnls <- lapply(pnls, function(pnl) volew*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts(pnls, datev[outsample])
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdav, y=sharper, t="l",
+     main="PCA Momentum Sharpe as Function of Decay Factor",
+     xlab="lambdaf", ylab="Sharpe")
```



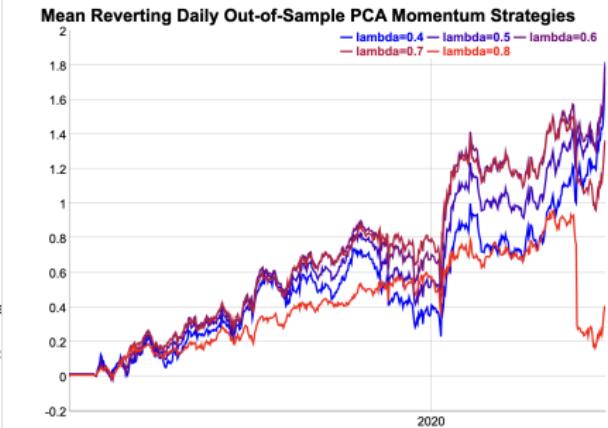
## Mean Reverting PCA Momentum Strategy Out-of-Sample

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 lambdaf parameters
> lambdav <- seq(0.4, 0.8, 0.1)
> pnls <- mclapply(lambdav, btmomdailyhold, retp=retPCA[, (dimax+1):
+   trend=(-1), mc.cores=ncores]
> pnls <- lapply(pnls, function(pnl) volew*pnl/sd(pnl))
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
> pnls <- xts::xts(pnls, date[outsample])
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdav, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Factor",
+   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 long, then the data has large *bias* because the distant past may have little relevance to today.

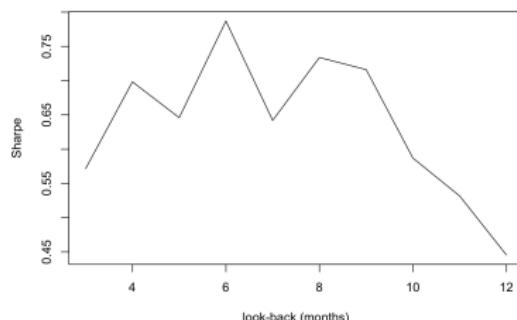
But if the *look-back interval* is too short, then there's not enough data, and estimates will have high *variance*.

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

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.

Momentum Sharpe as Function of Look-back Interval



```
> # 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
> lookbv <- seq(3, 12, by=1)
> pnll <- lapply(lookbv, btmomweight, retp=retp, endd=endd, objfun=)
> sharper <- sqrt(252)*sapply(pnll, function(pnl) mean(pnl)/sd(pnl))
> # Plot Sharpe ratios of momentum strategies
> plot(x=lookbv, 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.

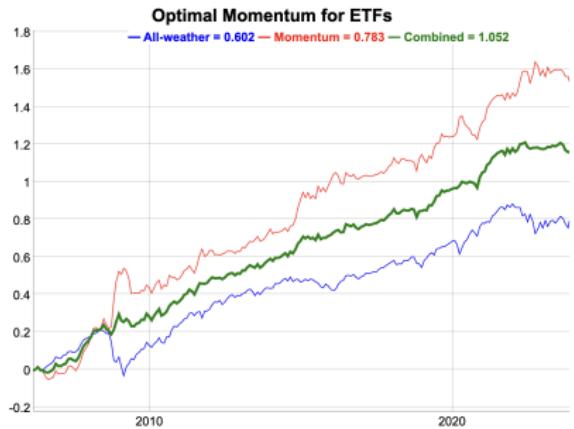
The momentum strategy for ETFs also has a very low correlation to the static *All-Weather* portfolio.

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)
> lookby[whichmax]
> pnls <- pnll[[whichmax]]
> retew <- rowMeans(retp)
> pnls <- c(retew[ennd[1]:ennd[3]], pnls)
> # Calculate returns of all-weather benchmark
> weightaw <- c(0.30, 0.55, 0.15)
> retaw <- retp %*% weightaw
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retaw, pnls, 0.5*(retaw+pnls))
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("All-weather", "Momentum", "Combined")
> cor(wealthv)
> wealthv <- xts::xts(wealthv, order.by=datev)
> sharper <- sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0]))))
> sharper
```



```
> captiont <- "Optimal Momentum for ETFs"
> colnames(wealthv) <- paste(colnames(wealthv), round(sharper[1, ], 1))
> # Plot dygraph of all-weather benchmark and momentum strategy
> dygraphs::dygraph(cumsum(wealthv)[ennd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name=colnames(wealthv)[3], strokeWidth=3) %>%
+   dyLegend(show="always", width=500)
```

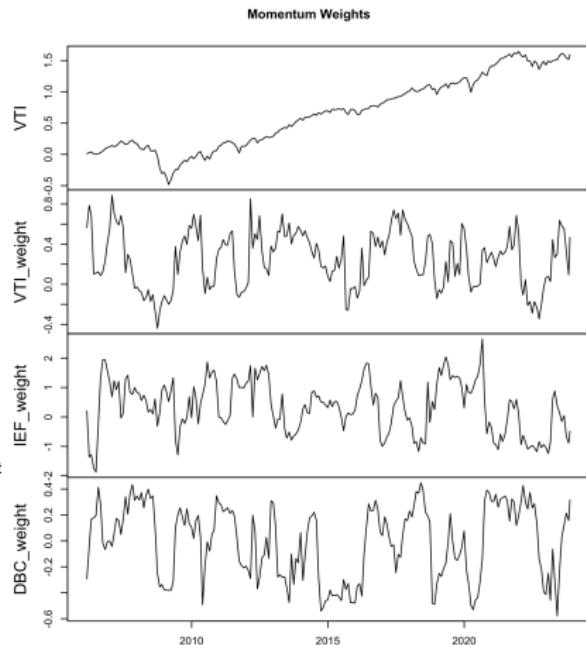
# Time Series of Momentum Portfolio Weights

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
> lookb <- lookbv[whichmax]
> weightv <- lapply(2:npts, function(tday) {
+   # Select the look-back returns
+   startp <- endd[max(1, tday-lookb)]
+   retis <- retp[startp:endd[tday], ]
+   # Calculate weights proportional to performance
+   perfstat <- sapply(retis, objfun)
+   weightv <- drop(perfstat)
+   # Scale weights so in-sample pnl volatility is same as equal we:
+   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")
```

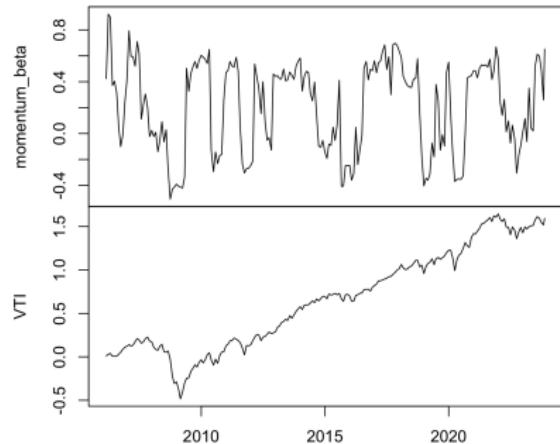


# Momentum Strategy Market Beta

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
> betac <- weightv %*% betasetf
> betac <- xts::xts(betac, order.by=datev[endd])
> colnames(betac) <- "momentum_beta"
> datav <- cbind(betac, retvti[endd])
> zoo::plot.zoo(datav, main="Momentum Beta & VTI Price", xlab="")
```

Momentum Beta & VTI Price



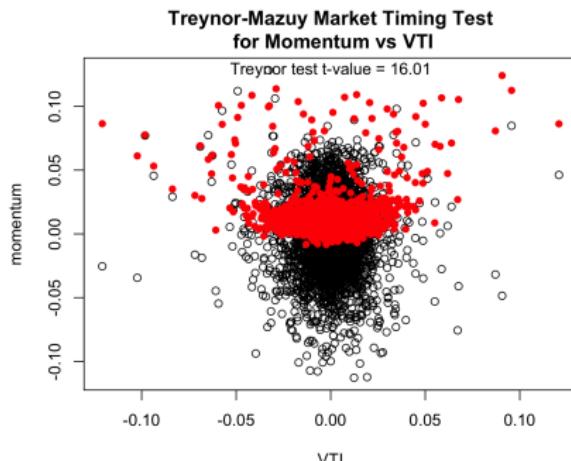
# Momentum Strategy Market Timing Skill

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

Since the momentum strategy forecasts over one month intervals, we should also aggregate the returns to one month intervals.

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

```
> # Aggregate the returns to monthly intervals
> retvti <- retp$VTI
> predm <- cbind(pnls, retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> predm <- HighFreq::roll_sumep(predm, lookbb=22)
> colnames(predm) <- c("pnls", "VTI", "merton", "treynor")
> predm <- as.data.frame(predm)
> # Merton-Henriksson test
> regmod <- lm(pnls ~ VTI + merton, data=predm); summary(regmod)
> # Treynor-Mazuy test
> regmod <- lm(pnls ~ VTI + treynor, data=predm); summary(regmod)
> # Plot residual scatterplot
> resid <- regmod$residuals
> plot.default(x=retvti, y=resid, xlab="VTI", ylab="momentum")
> title(main="Treynor-Mazuy Market Timing Test\nfor Momentum vs VTI", line=0.5)
> # 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=fity, pch=16, col="red")
> text(x=0.0, y=max(resid), paste("Treynor test t-value =", tvalue))
```



# Skewness of Momentum Strategy Returns

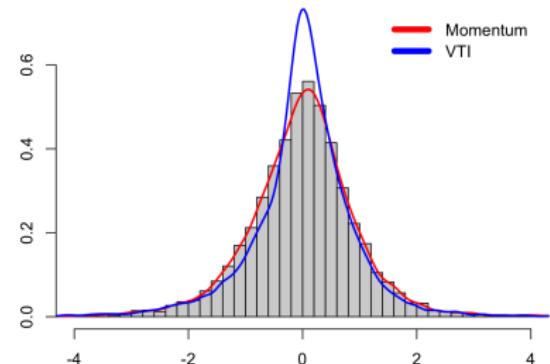
Most assets with *positive returns* suffer from *negative skewness*.

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)
```

Momentum and VTI Return Distributions (standardized)



```
> # 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=FALSE)
> # 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"))
```

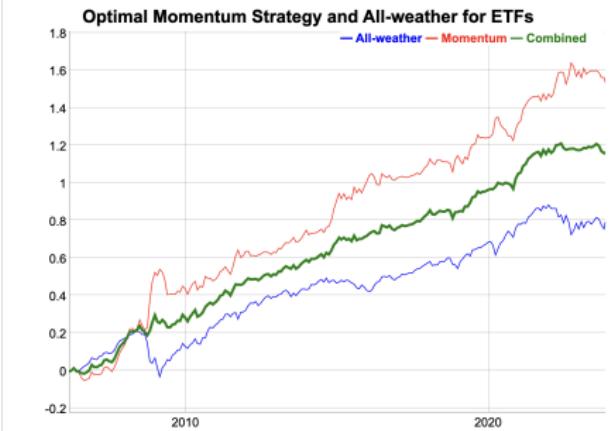
# Combining Momentum with the *All-Weather Portfolio*

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(retaw, pnls, 0.5*(pnls + retaw))
> colnames(wealthv) <- c("All-weather", "Momentum", "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
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Momentum Strategy and All-weather for ETFs") %>%
+   dyOptions(colors=c("blue", "red", "green"), 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
> lookb <- 152
> varm <- HighFreq::roll_var(retlp, lookb=lookb)
> # Calculate the trailing Kelly ratio
> meanv <- HighFreq::roll_mean(retlp, lookb=lookb)
> 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*retlp)
> # Calculate the transaction costs
> bidask <- 0.0
> costs <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)))
> pnls <- (pnls - costs)
```

Daily Momentum Strategy for ETFs vs All-Weather



```
> # Scale the momentum volatility to all-weather
> pnls <- sd(retaw)*pnls/sd(pnls)
> # Calculate the wealth of momentum returns
> wealthv <- cbind(retaw, pnls, 0.5*(pnls + retaw))
> colnames(wealthv) <- c("All-weather", "Momentum", "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
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Momentum Strategy for ETFs vs All-Weather") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

# Backtesting of Strategies

*Backtesting* is the simulation of a trading or investment strategy on historical data.

*Backtesting* is performed by *training* the strategy (calibrating the model parameters) on the past in-sample data and *testing* it on future out-of-sample data.

*Backtesting* is a type of *cross-validation* applied to investment strategies.

Backtest simulations can be useful for determining the relative advantages of different strategy features and model parameters.

Backtesting can be used to identify strategies that have little chance of being profitable.

Because if a strategy is not profitable in backtesting then it has little chance of being profitable in live trading.

Backtesting can be used to identify strategies that have a better chance of being profitable than others.

Backtesting can be used to rank strategies from the most promising to the least promising.

But backtesting cannot ensure that a strategy will be profitable in live trading, just because it is profitable in simulation.

## Backtesting of Momentum Strategies

Backtesting is performed by *training* the strategy (calibrating the model parameters) on the past in-sample data and *testing* it on future out-of-sample data.

The *momentum* portfolio weights are calculated using the past stock returns, which are determined by the length of the *look-back* interval.

The *momentum* strategy returns (pnls) are calculated by multiplying the out-of-sample stock returns times the forecast portfolio weights.

The *transaction costs* are equal to half the *bid-ask spread*  $\delta$  times the absolute value of the traded dollar amounts of the *risky assets*.

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 long, then the portfolio weights have large *bias* because the distant past may have little relevance to today.

But if the *look-back interval* is too short, then there's not enough data, and the portfolio weights will have high *variance*.

Backtesting can help to determine the optimal length of the *look-back interval*.

# Limitations of Backtest Simulations

The simulated strategy pnls from backtesting may not be realistic for several reasons:

- Transaction costs (broker commissions, bidask spread, market impact).
- Costs of stock borrowing (stocks must be borrowed in order to short them).
- Limits on stock borrowing and shorting (stocks may not be available for borrow, or they may be prohibited from shorting).

So the backtest simulations should be thought of as experiments to explore which features and parameters have better potential for profit.

But the backtest simulations cannot guarantee future profits.

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

# The Covariance of Stock Returns

Estimating the covariance of stock returns is complicated because their date ranges may not overlap in time. Stocks may trade over different date ranges because of IPOs and corporate events (takeovers, mergers).

The function `cov()` calculates the covariance matrix of time series. The argument `use="pairwise.complete.obs"` removes NA values from pairs of stock returns.

But removing NA values in pairs of stock returns can produce covariance matrices which are not positive semi-definite.

The reason is because the covariance are calculated over different time intervals for different pairs of stock returns.

Matrices which are not positive semi-definite may not have an inverse matrix, but they have a generalized inverse.

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

```
> # 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 values  
> retp <- rutils::etfenv$returns[, symbolv]  
> retp[1, ] <- 0.0  
> nstocks <- NCOL(retp)  
> datev <- zoo::index(retp)  
> # Calculate the covariance ignoring NA values  
> covmat <- cov(retp, use="pairwise.complete.obs")  
> sum(is.na(covmat))  
> # Calculate the inverse of covmat  
> invmat <- solve(covmat)  
> # Calculate the generalized inverse of covmat  
> invreg <- MASS::ginv(covmat)  
> all.equal(unname(invmat), invreg)
```

# Generalized Inverse of Singular Covariance Matrices

The standard inverse of a positive semi-definite matrix  $\mathbb{C}$  can be calculated from its *eigenvalues*  $\mathbb{D}$  and its *eigenvectors*  $\mathbb{O}$  as follows:

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

The covariance matrix may not be positive semi-definite if the number of time periods of returns (rows) is less than the number of stocks (columns).

In that case some of the higher order eigenvalues are zero, and the above covariance matrix inverse is singular.

But a non-positive semi-definite covariance matrix may still have a *generalized inverse*.

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

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

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

The generalized inverse  $\mathbb{C}_g^{-1}$  of the matrix  $\mathbb{C}$  satisfies the equation:

$$\mathbb{C} \mathbb{C}_g^{-1} \mathbb{C} = \mathbb{C}$$

Which is a generalization of the standard inverse property:  $\mathbb{C}^{-1} \mathbb{C} = \mathbb{I}$

```
> # Create rectangular matrix with collinear columns
> matv <- matrix(rnorm(10*8), nc=10)
> # Calculate covariance matrix
> covmat <- cov(matv)
> # 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 generalized inverse from the eigen decomposition
> nonzero <- (eigenval > (precv*eigenval[1]))
> inveigen <- eigenvec[, nonzero] %*%
+   (t(eigenvec[, nonzero]) / eigenval[nonzero])
> # Verify inverse property of invreg
> all.equal(covmat, inveigen %*% covmat)
> # Verify generalized inverse property of invreg
> all.equal(covmat, covmat %*% inveigen %*% covmat)
> # Calculate generalized inverse of covmat
> invreg <- MASS::ginv(covmat)
> # Verify that inveigen is the same as invreg
> all.equal(inveigen, invreg)
```

# Portfolio Optimization Strategy

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  $\mathbb{C}$ :

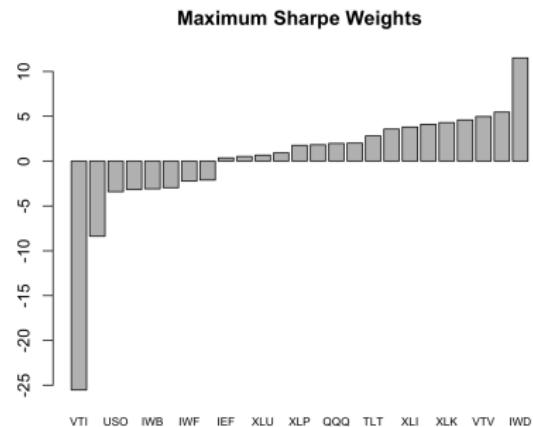
$$\mathbf{w} = \mathbb{C}^{-1} \mu$$

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.

```
> # Returns in excess of risk-free rate
> riskf <- 0.03/252
> retx <- (retp - riskf)
> # Maximum Sharpe weights in-sample interval
> retis <- retp["/2014"]
> invreg <- MASS::ginv(cov(retis, use="pairwise.complete.obs"))
> weightv <- drop(invreg %*% colMeans(retx["/2014"], na.rm=TRUE))
> # Calculate in-sample portfolio returns
> pnlis <- HighFreq::mult_mat(weightv, retis)
> pnlis <- rowMeans(pnlis, na.rm=TRUE)
> # Scale the weights to volatility target
> weightv <- weightv*0.01/sd(pnlis)
> names(weightv) <- colnames(retp)
```



```
> # Plot portfolio weights
> 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.

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

```
> # Calculate the equal weight index
> retew <- xts::xts(rowMeans(retis, na.rm=TRUE), zoo::index(retis))
> # Calculate the in-sample weighted returns using transpose
> pnlis <- unname(t(t(retis)*weightv))
> # Or using Rcpp
> # pnlis <- HighFreq::mult_mat(weightv, retis)
> pnlis <- rowMeans(pnlis, na.rm=TRUE)
> pnlis <- pnlis*sd(retew)/sd(pnlis)
```



```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnlis, (pnlis + retew)/2)
> colnames(wealthv) <- c("Equal Weight", "Optimal", "Combined")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Dygraph cumulative wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="In-Sample Optimal Portfolio Returns") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(width=300)
```

# Portfolio Optimization Strategy Out-of-Sample

The out-of-sample performance of the optimal in-sample portfolio is not nearly as good as in-sample, but still better than the equal weight portfolio.

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

```
> # Calculate the equal weight index
> retos <- rtp["2015/"]
> retew <- xts::xts(rowMeans(retos, na.rm=TRUE), zoo::index(retos))
> # Calculate out-of-sample portfolio returns
> pnlos <- HighFreq::mult_mat(weightv, retos)
> pnlos <- rowMeans(pnlos, na.rm=TRUE)
> pnlos <- pnlos*sd(retew)/sd(pnlos)
> wealthv <- cbind(retew, pnlos, (pnlos + retew)/2)
> colnames(wealthv) <- c("Equal Weight", "Optimal", "Combined")
> # 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])))
```



```
> # Dygraph cumulative wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Out-of-Sample Optimal Portfolio Returns") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(width=300)
```

# Portfolio Optimization Strategy for ETFs

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

Therefore the strategy doesn't perform as well in the *out-of-sample* interval as in the *in-sample* interval.

```
> # Maximum Sharpe weights in-sample interval
> invreg <- MASS::ginv(cov(retis, use="pairwise.complete.obs"))
> weightv <- invreg %*% colMeans(retis["/2014"], na.rm=TRUE)
> names(weightv) <- colnames(retis)
> # Calculate in-sample portfolio returns
> pnlis <- HighFreq::mult_mat(weightv, retis)
> pnlis <- rowMeans(pnlis, na.rm=TRUE)
> # Scale the weights to the volatility target
> weightv <- weightv*0.01/sd(pnlis)
> # Calculate out-of-sample portfolio returns
> pnlos <- HighFreq::mult_mat(weightv, retos)
> pnlos <- rowMeans(pnlos, na.rm=TRUE)
> # Calculate cumulative wealth
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(retew)/sd(pnls)
> retew <- xts::xts(rowMeans(retp, na.rm=TRUE), datev)
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Equal Weight", "Optimal", "Combined")
> # 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])))
```



```
> # Dygraph cumulative wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Out-of-Sample Optimal Portfolio Returns for ETFs") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeDash=c(5, 5))
+   dyLegend(width=400)
```

# 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 *reduced inverse*  $\mathbb{C}_R^{-1}$  is calculated from the largest (lowest order) eigenvalues, up to  $dimax = d$ :

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

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

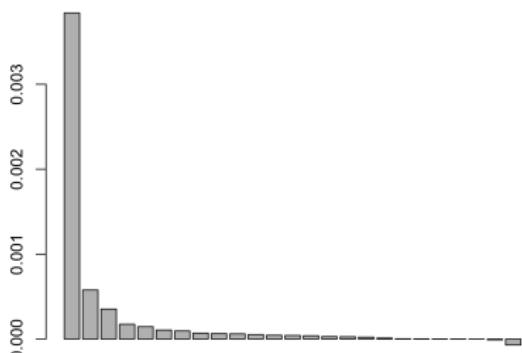
The *dimension reduction* technique calculates the *reduced inverse* of a covariance matrix by removing the very small, higher order eigenvalues, to reduce the propagation of statistical noise and improve the signal-to-noise ratio:

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

**ETF Covariance Eigenvalues**



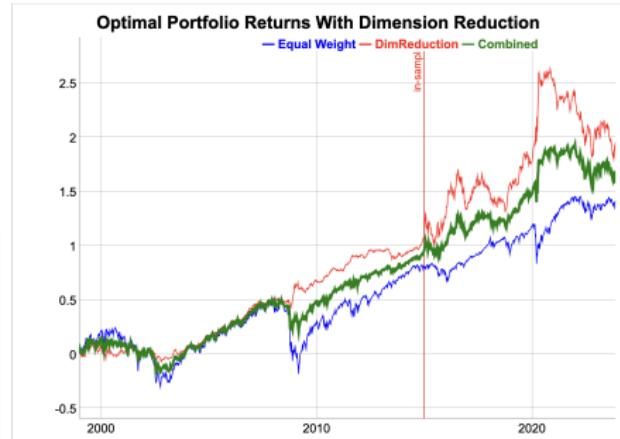
```
> # Calculate in-sample covariance matrix
> covmat <- cov(retis, use="pairwise.complete.obs")
> eigend <- eigen(covmat)
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> # Plot the eigenvalues
> barplot(eigenval, main="ETF Covariance Eigenvalues", cex.names=0.7)
> # Calculate reduced inverse of covariance matrix
> dimax <- 9
> invred <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
> # Reduced inverse does not satisfy matrix inverse property
> all.equal(covmat, covmat %*% invred %*% covmat)
```

# Portfolio Optimization for ETFs with Dimension Reduction

The *out-of-sample* performance of the *portfolio optimization* strategy is greatly improved by applying dimension reduction to the inverse of the covariance matrix.

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

```
> # Calculate portfolio weights
> weightv <- invred %*% colMeans(retis, na.rm=TRUE)
> names(weightv) <- colnames(retp)
> # Calculate in-sample portfolio returns
> pnlis <- HighFreq::mult_mat(weightv, retis)
> pnlis <- rowMeans(pnlis, na.rm=TRUE)
> # Scale the weights to volatility target
> weightv <- weightv*0.01/sd(pnlis)
> # Calculate out-of-sample portfolio returns
> pnlos <- HighFreq::mult_mat(weightv, retos)
> pnlos <- rowMeans(pnlos, na.rm=TRUE)
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Equal Weight", "DimReduction", "Combined")
> # 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])))
```



```
> # Dygraph cumulative wealth
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Portfolio Returns With Dimension Reduction") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", stroke
```

# 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  $\alpha$  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  $\alpha$  can be determined using *backtesting* (*cross-validation*).

```
> # Shrink the in-sample returns to their mean
> alphac <- 0.7
> retxm <- rowMeans(retx[~/2014"], na.rm=TRUE)
> retxis <- (1-alphac)*retx[~/2014"] + alphac*retxm
> # Calculate portfolio weights
> weightv <- invred %*% colMeans(retxis, na.rm=TRUE)
> # Calculate in-sample portfolio returns
> pnlis <- HighFreq::mult_mat(weightv, retis)
> pnlis <- rowMeans(pnlis, na.rm=TRUE)
> # Scale the weights to volatility target
> weightv <- weightv*0.01/sd(pnlis)
> # Calculate out-of-sample portfolio returns
> pnlos <- HighFreq::mult_mat(weightv, retos)
> pnlos <- rowMeans(pnlos, na.rm=TRUE)
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Equal Weight", "Optimal", "Combined")
```

Optimal Portfolio With Dimension Reduction and Return Shrinkage



```
> # 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])))
> # Dygraph cumulative wealth
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Portfolio With Dimension Reduction and Return Shrinkage",
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeDash=c(5, 5))
+   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="months")
> endd <- endd[endd > (nstocks+1)]
> npts <- NROW(endd)
> lookb <- 3
> startp <- c(rep_len(0, lookb), endd[1:(npts-lookb)])
> # Perform loop over end points
> pnls <- lapply(1:(npts-1), function(tday) {
+   # Calculate the portfolio weights
+   retis <- retx[startp[tday]:endd[tday], ]
+   covmat <- cov(retis, use="pairwise.complete.obs")
+   covmat[is.na(covmat)] <- 0
+   invreg <- MASS::ginv(covmat)
+   colm <- colMeans(retis, na.rm=TRUE)
+   colm[is.na(colm)] <- 0
+   weightv <- invreg %*% colm
+   # Scale the weights to volatility target
+   pnlis <- HighFreq::mult_mat(weightv, retis)
+   pnlis <- rowMeans(pnlis, na.rm=TRUE)
+   weightv <- weightv*0.01/sd(pnlis)
+   # Calculate the out-of-sample portfolio returns
+   retos <- retp[(endd[tday]+1):endd[tday+1], ]
+   pnlos <- HighFreq::mult_mat(weightv, retos)
+   pnlos <- rowMeans(pnlos, na.rm=TRUE)
+   xts::xts(pnlos, zoo::index(retos))
+ }) # end lapply
> pnls <- do.call(rbind, pnls)
> pnls <- rbind(retew[paste0("/", start(pnls)-1)], pnls)
```

# Rolling Portfolio Strategy Performance

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

```
> # Calculate the Sharpe and Sortino ratios  
> pnls <- pnls*sd(retew)/sd(pnls)  
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)  
> colnames(wealthv) <- c("Index", "PortfStrat", "Combined")  
> sqrt(252)*sapply(wealthv, function(x)  
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```



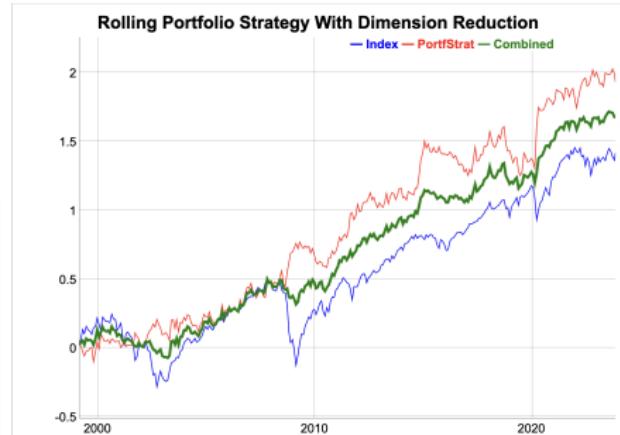
```
> # Dygraph cumulative wealth  
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Monthly ETF Rolling"  
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %%  
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %%  
+   dyLegend(show="always", width=300)
```

# Rolling Portfolio Strategy With Dimension Reduction

Dimension reduction improves the performance of the rolling portfolio strategy because it suppresses the data noise.

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

```
> # Perform loop over end points
> dimax <- 9
> pnls <- lapply(1:(npts-1), function(tday) {
+   # Calculate the portfolio weights
+   retis <- retx[startp[tday]:endd[tday], ]
+   covmat <- cov(retis, use="pairwise.complete.obs")
+   covmat[is.na(covmat)] <- 0
+   eigend <- eigen(covmat)
+   eigenvec <- eigend$vectors
+   eigenval <- eigend$values
+   invred <- eigenvec[, 1:dimax] %*%
+ (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
+   colm <- colMeans(retis, na.rm=TRUE)
+   colm[is.na(colm)] <- 0
+   weightv <- invred %*% colm
+   # Scale the weights to volatility target
+   pnlis <- HighFreq::mult_mat(weightv, retis)
+   pnlis <- rowMeans(pnlis, na.rm=TRUE)
+   weightv <- weightv*0.01/sd(pnlis)
+   # Calculate the out-of-sample portfolio returns
+   retos <- retx[(endd[tday]+1):endd[tday+1], ]
+   pnlos <- HighFreq::mult_mat(weightv, retos)
+   pnlos <- rowMeans(pnlos, na.rm=TRUE)
+   xts::xts(pnlos, zoo::index(retos))
+ }) # end lapply
> pnls <- do.call(rbind, pnls)
> pnls <- rbind(retew[paste0("/", start(pnls)-1)], pnls)
```



```
> # Calculate the Sharpe and Sortino ratios
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Index", "PortfStrat", "Combined")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Dygraph cumulative wealth
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Rolling Portfolio S")
+ dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+ dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+ dyLegend(show="always", width=300)
```

# Rolling Portfolio Strategy With Return Shrinkage

Shrinkage averages the stock returns, which creates bias, but it also reduces the variance.

Return shrinkage can be applied to improve the performance of the rolling portfolio strategy.

```
> alphac <- 0.7 # Return shrinkage intensity
> # Perform loop over end points
> pnls <- lapply(1:(npts-1), function(tday) {
+   # Shrink the in-sample returns to their mean
+   retis <- retx[startp[tday]:endd[tday], ]
+   rowm <- rowMeans(retis, na.rm=TRUE)
+   rowm[is.na(rowm)] <- 0
+   retis <- (1-alphac)*retis + alphac*rowm
+   # Calculate the portfolio weights
+   covmat <- cov(retis, use="pairwise.complete.obs")
+   covmat[is.na(covmat)] <- 0
+   eigend <- eigen(covmat)
+   eigenvec <- eigend$vectors
+   eigenval <- eigend$values
+   invred <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
+   colm <- colMeans(retis, na.rm=TRUE)
+   colm[is.na(colm)] <- 0
+   weightv <- invred %*% colm
+   # Scale the weights to volatility target
+   pnlis <- HighFreq::mult_mat(weightv, retis)
+   pnlis <- rowMeans(pnlis, na.rm=TRUE)
+   weightv <- weightv*0.01/sd(pnlis)
+   # Calculate the out-of-sample portfolio returns
+   retos <- retp[(endd[tday]+1):endd[tday+1], ]
+   pnlos <- HighFreq::mult_mat(weightv, retos)
+   pnlos <- rowMeans(pnlos, na.rm=TRUE)
+   xts::xts(pnlos, zoo::index(retos))
+ }) # end lapply
> pnls <- do.call(rbind, pnls)
> pnls <- rbind(retew[paste0("/", start(pnls)-1)], pnls)
```

**Rolling Portfolio Strategy With Return Shrinkage**



```
> # Calculate the Sharpe and Sortino ratios
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Index", "PortfStrat", "Combined")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Dygraph cumulative wealth
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Rolling Portfolio S")
+ dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+ dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+ dyLegend(show="always", width=300)
```

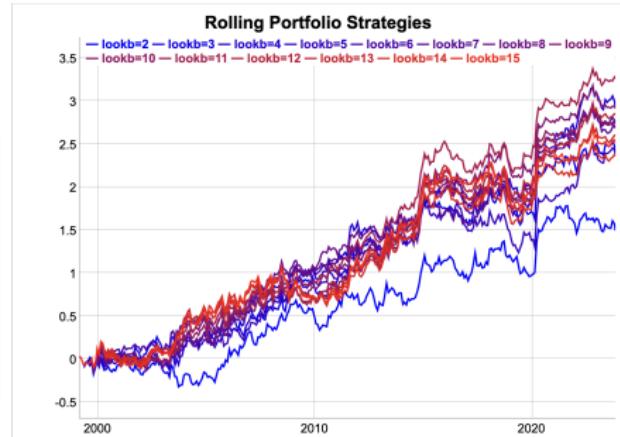
# Function for Rolling Portfolio Optimization Strategy

```
> # Define backtest functional for rolling portfolio strategy
> roll_portf <- function(retx, # Excess returns
+                         retp, # Stock returns
+                         endd, # End points
+                         lookb=12, # Look-back interval
+                         dimax=3, # Dimension reduction parameter
+                         alphac=0.0, # Return shrinkage intensity
+                         bidask=0.0, # Bid-offer spread
+                         ...) {
+   npts <- NROW(endd)
+   startp <- c(rep_len(0, lookb), endd[1:(npts-lookb)])
+   pnls <- lapply(1:(npts-1), function(tday) {
+     retis <- retx[startp[tday]:endd[tday], ]
+     # Shrink the in-sample returns to their mean
+     if (alphac > 0) {
+       rowm <- rowMeans(retis, na.rm=TRUE)
+       rowm[is.na(rowm)] <- 0
+       retis <- (1-alphac)*retis + alphac*rowm
+     } # end if
+     # Calculate the portfolio weights
+     covmat <- cov(retis, use="pairwise.complete.obs")
+     covmat[is.na(covmat)] <- 0
+     eigend <- eigen(covmat)
+     eigenvec <- eigend$vectors
+     eigenval <- eigend$values
+     invred <- eigenvec[, 1:dimax] %*% (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
+     colm <- colMeans(retis, na.rm=TRUE)
+     colm[is.na(colm)] <- 0
+     weightv <- invred %*% colm
+     # Scale the weights to volatility target
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     weightv <- weightv*0.01/sd(pnlis)
+     # Calculate the out-of-sample portfolio returns
+     retos <- retp[(endd[tday]+1):endd[tday+1], ]
+     pnlos <- HighFreq::mult_mat(weightv, retos)
+     pnlos <- rowMeans(pnlos, na.rm=TRUE)
+     #return(pnlos)
+   })
+ }
```

# 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 portfolio strategy
> pnls <- roll_portf(retx=retx, retp=retp, endd=endd,
+   lookb=lookb, dimax=dimax)
> # Perform sapply loop over lookbv
> lookbv <- seq(2, 15, by=1)
> pnls <- lapply(lookbv, roll_portf,
+   retp=retp, retx=retx, endd=endd, dimax=dimax)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lookb=", lookbv)
> pnlsums <- sapply(pnls, sum)
> lookb <- lookbv[which.max(pnlsums)]
```



```
> # Plot dygraph of monthly ETF portfolio 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=600)
> # Plot portfolio 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
> dimv <- 2:11
> pnls <- lapply(dimv, roll_portf, retx=retx,
+   rtretp=rtrp, endd=endd, lookb=lookb)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("dimax=", dimv)
> pnlsums <- sapply(pnls, sum)
> dimax <- dimv[which.max(pnlsums)]
```

**Rolling Portfolio Strategies With Dimension Reduction**



```
> # Plot dygraph of monthly ETF portfolio strategies
> colrv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies With Dimension Reduction") %>%
+   dyOptions(colors=colrv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot portfolio 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(rtrp)),
+   col=plot_theme$col$line.col, bty="n")
```

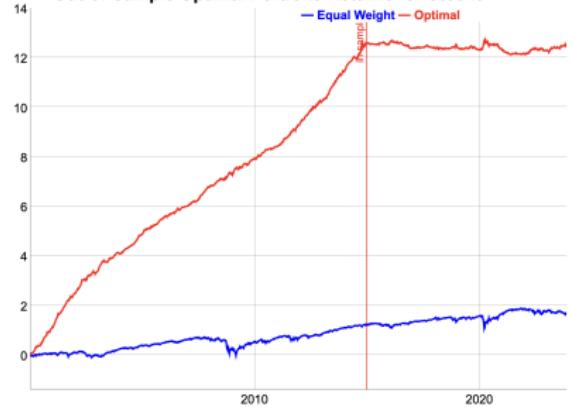
# Portfolio Optimization Strategy for Stocks

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

Therefore the strategy is mediocre in the *out-of-sample* interval.

```
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RData")
> # Overwrite NA values in returns
> retp <- retstock["2000/"]
> retp[is.na(retp)] <- 0
> nstocks <- NCOL(retp)
> datev <- zoo::index(retp)
> riskf <- 0.03/252
> retx <- (retp - riskf)
> retis <- retx["/2014"]
> retos <- retx["2015/"]
> # Maximum Sharpe weights in-sample interval
> covmat <- cov(retis, use="pairwise.complete.obs")
> covmat[is.na(covmat)] <- 0
> invreg <- MASS::ginv(covmat)
> colmeanv <- colMeans(retx["/2014"], na.rm=TRUE)
> colmeanv[is.nan(colmeanv)] <- 0
> weightv <- invreg %*% colmeanv
> rownames(weightv) <- colnames(retp)
> weightv <- weightv*0.01/sd(pnlis)
> # Calculate in-sample portfolio returns
> pnlis <- (retis %*% weightv)
> # Scale the weights to volatility target
> weightv <- weightv*0.01/sd(pnlis)
> # Calculate out-of-sample portfolio returns
> pnlos <- (retos %*% weightv)
> retew <- xts::xts(rowMeans(retp, na.rm=TRUE), datev)
> # Combine in-sample and out-of-sample returns
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sdev(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls)
> colnames(wealthv) <- c("Equal Weight", "Optimal")
```

Out-of-Sample Optimal Portfolio Returns for Stocks



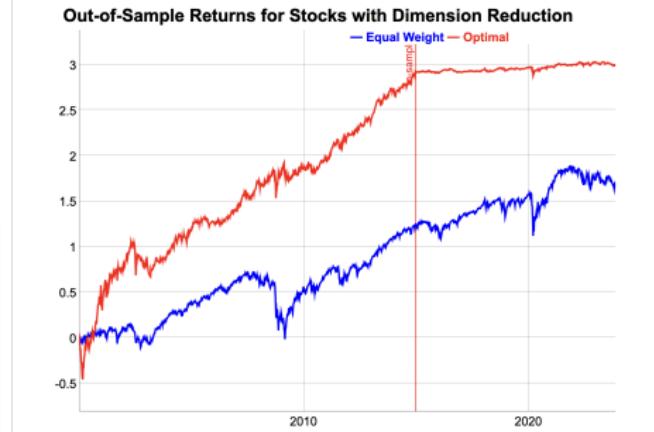
```
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[index(retis)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[index(retos)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative portfolio returns
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[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", strokeWidth=3)
+   dyLegend(width=300)
```

# Portfolio Strategy for Stocks with Dimension Reduction

The *out-of-sample* performance of the *portfolio optimization* strategy is greatly improved by applying dimension reduction to the inverse of the covariance matrix.

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

```
> # Calculate reduced inverse of covariance matrix
> dimax <- 5
> eigend <- eigen(cov(retis, use="pairwise.complete.obs"))
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> invred <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
> # Calculate portfolio weights
> weightv <- invred %*% colMeans(retx["/2014"], na.rm=TRUE)
> names(weightv) <- colnames(retp)
> # Calculate in-sample portfolio returns
> pnlis <- (retis %*% weightv)
> # Scale the weights to volatility target
> weightv <- weightv*0.01/sd(pnlis)
> # Calculate out-of-sample portfolio returns
> pnlos <- (retos %*% weightv)
```



```
> # Combine in-sample and out-of-sample returns
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls)
> colnames(wealthv) <- c("Equal Weight", "Optimal")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[index(retos)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative portfolio returns
> dygraphs::dygraph(cumsum(wealthv)[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", strokeWidth=300)
+   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  $\alpha$  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  $\alpha$  can be determined using *backtesting* (*cross-validation*).

```
> # Shrink the in-sample returns to their mean
> alphac <- 0.7
> retxm <- rowMeans(retx["/2014"], na.rm=TRUE)
> retxis <- (1-alphac)*retx["/2014"] + alphac*retxm
> # Calculate portfolio weights
> weightv <- invred %*% colMeans(retxis, na.rm=TRUE)
> # Calculate in-sample portfolio returns
> pnlis <- (retis %*% weightv)
> # Scale the weights to volatility target
> weightv <- weightv*0.01/sd(pnlis)
> # Calculate out-of-sample portfolio returns
> pnlos <- (retos %*% weightv)
```



```
> # Combine in-sample and out-of-sample returns
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls)
> colnames(wealthv) <- c("Equal Weight", "Optimal")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[index(retos)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative portfolio returns
> dygraphs::dygraph(cumsum(wealthv)[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", strokeDash=c(5, 5)) %
+   dyLegend(width=300)
```

# Rolling Portfolio 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 stock returns
> retp <- retstock100["2000/"]
> retp[is.na(retp)] <- 0
> retx <- retp # Set risk-free rate to zero
> nstocks <- NCOL(retp)
> datev <- zoo::index(retp)
> # Define monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> endd <- endd[endd > (nstocks+1)]
> npts <- NROW(endd) ; lookb <- 12
> startp <- c(rep_len(0, lookb), endd[1:(npts-lookb)])
> # Perform parallel loop over end points
> library(parallel) # Load package parallel
> ncores <- detectCores() - 1
> pnls <- mclapply(1:(npts-1), function(tday) {
+   # Subset the excess returns
+   retis <- retx[startp[tday]:endd[tday], ]
+   invreg <- MASS::ginv(cov(retis, use="pairwise.complete.obs"))
+   # Calculate the maximum Sharpe ratio portfolio weights
+   weightv <- invreg %*% colMeans(retis, na.rm=TRUE)
+   # Zero weights if sparse data
+   zerov <- sapply(retis, function(x) (sum(x == 0) > 5))
+   weightv[zerov] <- 0
+   # Calculate in-sample portfolio returns
+   pnls <- (retis %*% weightv)
+   # Scale the weights to volatility target
+   weightv <- weightv*0.01/sd(pnls)
+   # Calculate the out-of-sample portfolio returns
+   retos <- retp[(endd[tday]+1):endd[tday+1], ]
+   xts::xts(retos %*% weightv, zoo::index(retos))
+ }) # end lapply
> pnls <- rutils::do_call(rbind, pnls)
```

# Performance of Rolling Portfolio Strategy for Stocks

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

```
> # Calculate returns of equal weight portfolio  
> retew <- xts::xts(rowMeans(retp), datev)  
> pnls <- rbind(retew[paste0("/", start(pnls)-1)], pnls*sd(retew)/sd(pnls))  
> # Calculate the Sharpe and Sortino ratios  
> wealthv <- cbind(retew, 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])))
```

**Rolling Portfolio Strategy for S&P500 Stocks**



```
> # Plot cumulative strategy returns  
> dygraphs::dygraph(cumsum(wealthv)[endd],  
+   main="Rolling Portfolio Strategy for S&P500 Stocks") %>%  
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%  
+   dyLegend(show="always", width=300)
```

# Fast Covariance Matrix Inverse Using *RcppArmadillo*

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

```
> # Create random matrix of returns
> matv <- matrix(rnorm(300), ncol=5)
> # Reduced inverse of covariance matrix
> dimax <- 3
> eigend <- eigen(covmat)
> invred <- eigend$vectors[, 1:dimax] %*%
+   (t(eigend$vectors[, 1:dimax]) / eigend$values[1:dimax])
> # Reduced inverse using RcppArmadillo
> invarma <- HighFreq::calc_inv(covmat, dimax)
> all.equal(invred, invarma)
> # Microbenchmark RcppArmadillo code
> library(microbenchmark)
> summary(microbenchmark(
+   rcode={eigend <- eigen(covmat)
+     eigend$vectors[, 1:dimax] %*%
+   (t(eigend$vectors[, 1:dimax]) / eigend$values[1:dimax])
+   },
+   rcpp=calc_inv(covmat, dimax),
+   times=10) [, c(1, 4, 5)] # end microbenchmark summary
```

```
arma::mat calc_inv(const arma::mat& matv,
                    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 reduced inverse using SVD decompositi

        // 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, // Asset returns
                      Rcpp::List controlv) { // List of portfolio optimization parameters

    // Unpack the control list of portfolio optimization parameters
    // Type of portfolio optimization model
    std::string method = Rcpp::as<std::string>(controlv["method"]);
    // Threshold level for discarding small singular values
    double eigen_thresh = Rcpp::as<double>(controlv["eigen_thresh"]);
    // Dimension reduction
    arma::uword dimax = Rcpp::as<int>(controlv["dimax"]);
    // Confidence level for calculating the quantiles of returns
    double confl = Rcpp::as<double>(controlv["confl"]);
    // Shrinkage intensity of returns
    double alpha = Rcpp::as<double>(controlv["alpha"]);
    // Should the weights be ranked?
    bool rankw = Rcpp::as<int>(controlv["rankw"]);
    // Should the weights be centered?
    bool centerw = Rcpp::as<int>(controlv["centerw"]);
    // Method for scaling the weights
    std::string scalew = Rcpp::as<std::string>(controlv["scalew"]);
    // Volatility target for scaling the weights
    double vol_target = Rcpp::as<double>(controlv["vol_target"]);

    // Initialize the variables
    arma::uword ncols = returns.n_cols;
    arma::vec weightv(ncols, fill::zeros);
    // If no regularization then set dimax to ncols
    if (dimax == 0) dimax = ncols;
    // Calculate the covariance matrix
    arma::mat covmat = calc_covar(returns);

    // Apply different calculation methods for the weights
    switch(calc_method(method)) {
        case methodenum::maxsharpe: {
            // Mean returns of columns
            ... (code for maxsharpe method)
        }
        ...
    }
}
```

# Strategy Backtesting Using *RcppArmadillo*

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

```
arma::mat back_test(const arma::mat& retx, // Asset excess returns
                    const arma::mat& retp, // Asset returns
                    Rcpp::List controlv, // List of portfolio optimization model parameters
                    arma::uvec startp, // Start points
                    arma::uvec endp, // End points
                    double lambdaf = 0.0, // Decay factor for averaging the portfolio weights
                    double coeff = 1.0, // Multiplier of strategy returns
                    double bidask = 0.0) { // The bid-ask spread

    double lambda1 = 1-lambdaf;
    arma::uword nweights = retp.n_cols;
    arma::vec weightv(nweights, fill::zeros);
    arma::vec weights_past = arma::ones(nweights)/std::sqrt(nweights);
    arma::mat pnls = arma::zeros(retp.n_rows, 1);

    // Perform loop over the end points
    for (arma::uword it = 1; it < endp.size(); it++) {
        // cout << "it: " << it << endl;
        // Calculate the portfolio weights
        weightv = coeff*calc_weights(retx.rows(startp(it-1), endp(it-1)), controlv);
        // Calculate the weights as the weighted sum with past weights
        weightv = lambda1*weightv + lambdaf*weights_past;
        // Calculate out-of-sample returns
        pnls.rows(endp(it-1)+1, endp(it)) = retp.rows(endp(it-1)+1, endp(it))*weightv;
        // Add transaction costs
        pnls.row(endp(it-1)+1) -= bidask*sum(abs(weightv - weights_past))/2;
        // Copy the weights
        weights_past = weightv;
    } // end for

    // Return the strategy pnls
    return pnls;
} // end back_test
```

# Rolling Portfolio Strategy With Dimension Reduction and Shrinkage

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

The backtest simulation can be performed very quickly using C++ code and package *RcppArmadillo*.

```
> # 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 I
> dimax <- 9
> alphac <- 0.7
> controlv <- HighFreq::param_portf(method="maxsharpe", dimax=dimax,
> # Perform backtest in Rcpp
> pnls <- HighFreq::back_test(retx=retx, retp=retp,
+   startp=startp, endd=endd, controlv=controlv)
> pnls <- pnls*sd(retew)/sd(pnls)
```



```
> # Plot cumulative strategy returns
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Index", "PortfStrat", "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 Portfolio Strategy With Shrinkage") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

# Optimal Dimension Reduction Parameter

The optimal value of the dimension reduction parameter  $\text{dimax}$  can be determined using *backtesting*.

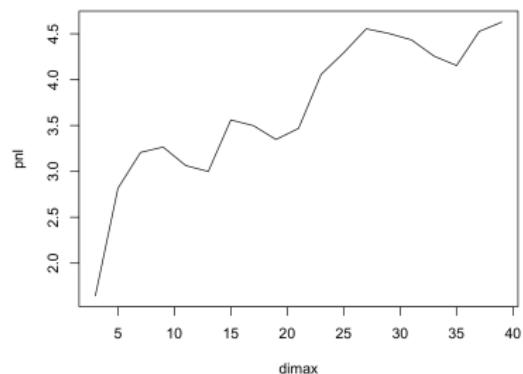
The best dimension reduction parameter for this portfolio of stocks is equal to  $\text{dimax}=21$ , which means relatively weak dimension reduction.

The dependence of the out-of-sample returns on the  $\text{dimax}$  parameter reflects the *bias-variance tradeoff*.

If the  $\text{dimax}$  parameter is too small, it creates excessive bias, but if it's too large, it doesn't suppress the variance enough.

```
> # Perform backtest over vector of dimension reduction parameters
> dimv <- seq(from=3, to=40, by=2)
> pnls <- mclapply(dimv, function(dimax) {
+   controlv <- HighFreq::param_portf(method="maxsharpe",
+   dimax=dimax, alpha=alphac)
+   HighFreq:::back_test(retx=retx, retp=retp,
+   startp=startp, endd=endd, controlv=controlv)
+ }) # end lapply
> profilev <- sapply(pnls, sum)
> whichmax <- which.max(profilev)
> dimax <- dimv[whichmax]
```

**Rolling Strategy PnL as Function of dimax**



```
> plot(x=dimv, y=profilev, t="l", xlab="dimax", ylab="pnl",
+   main="Rolling Strategy PnL as Function of dimax")
```

# Optimal Shrinkage Parameter

The optimal value of the return shrinkage intensity parameter  $\alpha$  can be determined using *backtesting*.

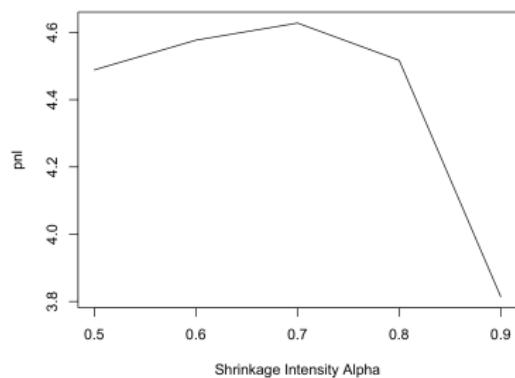
The best return shrinkage parameter for this portfolio of stocks is equal to  $\alpha = 0.89$ , which means strong return shrinkage.

The dependence of the out-of-sample returns on the  $\alpha$  parameter reflects the *bias-variance tradeoff*.

If the  $\alpha$  parameter is too large, it creates excessive bias, but if it's too small, it doesn't suppress the variance enough.

```
> # Perform backtest over vector of return shrinkage intensities
> alphav <- seq(from=0.5, to=0.9, by=0.1)
> pnls <- mclapply(alphav, function(alphac) {
+   controlv <- HighFreq::param_portf(method="maxsharpe",
+     dimax=dimax, alpha=alphac)
+   HighFreq::back_test(retx=retx, retp=retp,
+     startp=startp, endd=endd, controlv=controlv)
+ }) # end lapply
> profilev <- sapply(pnls, sum)
> whichmax <- which.max(profilev)
> alphac <- alphav[whichmax]
```

**Rolling Strategy PnL as Function of Return Shrinkage**



```
> plot(x=alphav, y=profilev, t="l",
+   main="Rolling Strategy PnL as Function of Return Shrinkage",
+   xlab="Shrinkage Intensity Alpha", ylab="pnl")
```

# Determining Look-back Interval Using Backtesting

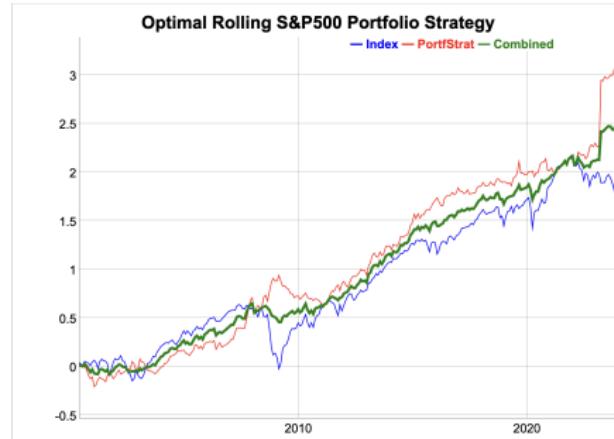
The optimal value of the look-back interval can be determined using *backtesting*.

The optimal value of the look-back interval for this portfolio of stocks is equal to `lookb=10` months, which roughly agrees with the research literature on momentum strategies.

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 long, then the data has large *bias* because the distant past may have little relevance to today.

But if the *look-back interval* is too short, then there's not enough data, and estimates will have high *variance*.



```
> # Create list of model parameters
> controlv <- HighFreq::param_portf(method="maxsharpe",
+   dimax=dimax, alpha=alphac)
> # Perform backtest over look-backs
> lookbv <- seq(from=5, to=16, by=1)
> pnls <- mclapply(lookbv, function(lookb) {
+   startp <- c(rep_len(0, lookb), endd[1:(npts-lookb)])
+   startp <- (startp - 1) ; startp[startp < 0] <- 0
+   HighFreq::back_test(retx=retx, retp=retp,
+     startp=startp, endd=endd, controlv=controlv)
+ }) # end lapply
> profilev <- sapply(pnls, sum)
> plot(x=lookbv, y=profilev, t="l", main="Strategy PnL as Function
+   of Look-back Interval", ylab="pnl")
```

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> whichmax <- which.max(profilev)
> lookb <- lookbv[whichmax]
> pnls <- pnls[[whichmax]]
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Index", "PortfStrat", "Combined")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Dygraph the cumulative wealth
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Rolling S&P500 Portfolio Strategy") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   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](#)