Multivariate Investment Strategies

FRE7241, Fall 2024

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

October 17, 2024

NYU | TANDON SCHOOL OF ENGINEERING

Idiosyncratic Stock Returns

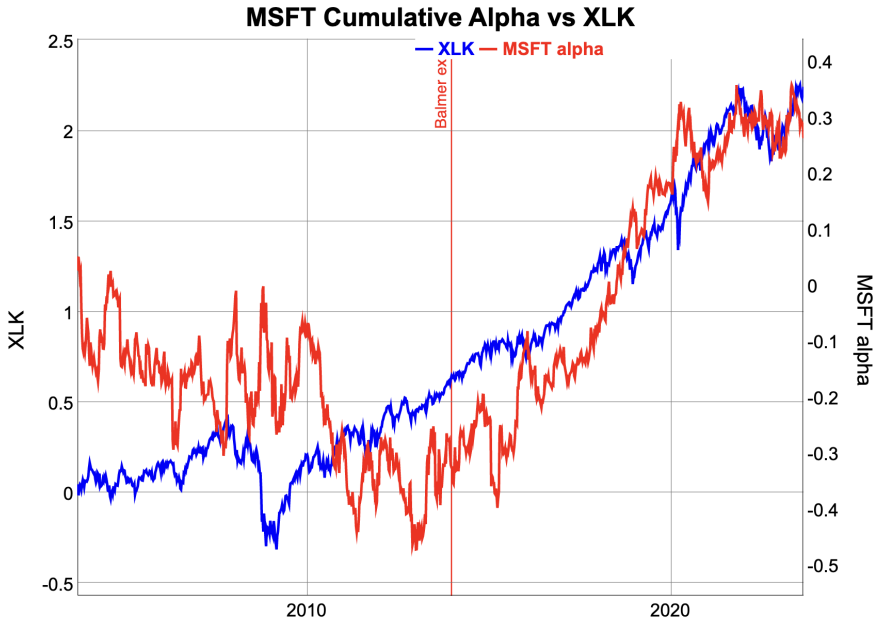
The daily stock returns $r_i - r_f$ in excess of the risk-free rate r_f, can be decomposed into *systematic* returns $\beta(r_m - r_f)$ (r_m are the market returns) plus *idiosyncratic* returns $\alpha + \varepsilon_i$ (which are uncorrelated to the market returns):

$$r_i - r_f = \alpha + \beta(r_m - r_f) + \varepsilon_i$$

The *alpha* α are the abnormal returns in excess of the risk premium $\beta(r_m - r_f)$, and ε_i are the regression residuals with zero mean.

We can simplify the formula by setting the risk-free rate to zero $r_f = 0$ and redefining the alpha α.

```
> # Load daily S&P500 stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RData")
> # Select ETF returns
> retetf <- rutils::etfenv$returns[, c("VTI", "XLK", "XLF", "XLE")]
> # Calculate the MSFT betas with respect to different ETFs
> betas <- sapply(retetf, function(retetf) {
+   retp <- na.omit(cbind(retstock$MSFT, retetf))
+   # Calculate the MSFT beta
+   drop(cov(retp$MSFT, retp[, 2])/var(retp[, 2]))
+ }) # end sapply
> # Combine MSFT and XLK returns
> retp <- cbind(retstock$MSFT, rutils::etfenv$returns$XLK)
> retp <- na.omit(retp)
> colnames(retp) <- c("MSFT", "XLK")
> # Calculate the beta and alpha of returns MSFT ~ XLK
> betac <- drop(cov(retp$MSFT, retp$XLK)/var(retp$XLK))
> alphac <- retp$MSFT - betac*retp$XLK
> # Scatterplot of returns
> plot(MSFT ~ XLK, data=retp, main="MSFT ~ XLK Returns",
+      xlab="XLK", ylab="MSFT", pch=1, col="blue")
```



The stock of Microsoft *MSFT* began outperforming the *XLK* ETF after Steve Ballmer was replaced as CEO in 2014.

```
> # dygraph plot of MSFT idiosyncratic returns vs XLK
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> datev <- zoo::index(retp)[endd]
> dateb <- datev[findInterval(as.Date("2014-01-01"), datev)] # Steve
> datav <- cbind(retp$XLK, alphac)
> colnames(datav)[2] <- "MSFT alpha"
> colv <- colnames(datav)
> dygraphs::dygraph(cumsum(datav)[endd], main="MSFT Cumulative Alpha
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyEvent(dateb, label="Balmer exit", strokePattern="solid", color
+   dyLegend(show="always", width=300)
```

Trailing Idiosyncratic Stock Returns

In practice, the stock beta should be updated over time and applied out-of-sample to calculate the trailing idiosyncratic stock returns.

The trailing beta β of a stock with returns r_t with respect to a stock index with returns R_t can be updated using these recursive formulas with the decay factor λ:

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

$$\bar{R}_t = \lambda \bar{R}_{t-1} + (1-\lambda) R_t$$

$$\sigma_t^2 = \lambda \sigma_{t-1}^2 + (1-\lambda)(R_t - \bar{R}_t)^2$$

$$\text{cov}_t = \lambda \, \text{cov}_{t-1} + (1-\lambda)(r_t - \bar{r}_t)(R_t - \bar{R}_t)$$

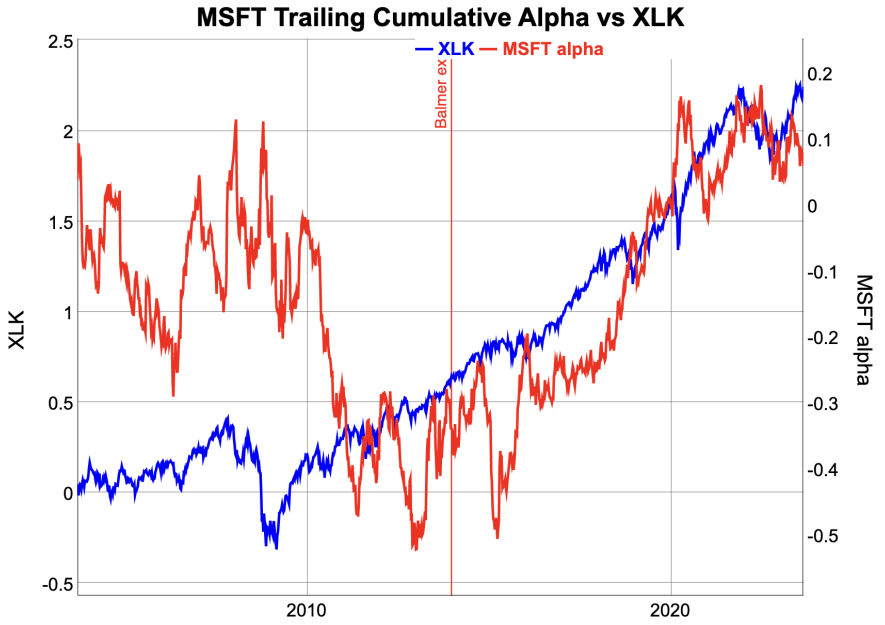
$$\beta_t = \frac{\text{cov}_t}{\sigma_t^2}$$

$$\alpha_t = r_t - \beta_t R_t$$

The parameter λ determines the rate of decay of the weight of past returns. If λ is close to 1 then the decay is weak and past returns have a greater weight. And vice versa if λ is close to 0.

The function `HighFreq::run_covar()` calculates the trailing variances, covariances, and means of two *time series*.

Using a dynamic beta produces a similar picture of *MSFT* stock performance versus the *XLK* ETF.




```
> # Calculate the trailing alphas and betas
> lambdaf <- 0.9
> covarv <- HighFreq::run_covar(retp, lambdaf)
> covarv[1, ] <- 1.0
> betac <- covarv[, 1]/covarv[, 3]
> alphac <- retp$MSFT - betac*retp$XLK
> # dygraph plot of trailing MSFT idiosyncratic returns vs XLK
> datav <- cbind(retp$XLK, alphac)
> colnames(datav)[2] <- "MSFT alpha"
> colv <- colnames(datav)
> dygraphs::dygraph(cumsum(datav)[endd], main="MSFT Trailing Cumulat
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyEvent(dateb, label="Balmer exit", strokePattern="solid", colo
+   dyLegend(show="always", width=300)
```

The Dickey-Fuller Process

The *Dickey-Fuller* process is a combination of an *Ornstein-Uhlenbeck* process and an *autoregressive* process.

The returns r_t are equal to the sum of a mean reverting term plus *autoregressive* terms:

$$r_t = \theta(\mu - p_{t-1}) + \varphi_1 r_{t-1} + \ldots + \varphi_n r_{t-n} + \sigma \xi_t$$

$$p_t = p_{t-1} + r_t$$

Where μ is the equilibrium price, σ is the volatility of returns, θ is the strength of mean reversion, and ξ_t are standard normal *innovations*.

Then the prices follow an *autoregressive* process:

$$p_t = \theta\mu + (1 + \varphi_1 - \theta)p_{t-1} + (\varphi_2 - \varphi_1)p_{t-2} + \ldots + (\varphi_n - \varphi_{n-1})p_{t-n} - \varphi_n p_{t-n-1} + \sigma \xi_t$$

The sum of the *autoregressive* coefficients is equal to $1 - \theta$, so if the mean reversion parameter θ is positive: $\theta > 0$, then the time series p_t exhibits mean reversion and has no *unit root*.

```
> # Define Dickey-Fuller parameters
> prici <- 0.0;  priceq <- 1.0
> thetav <- 0.01;  nrows <- 1000
> coeff <- c(0.1, 0.39, 0.5)
> # Initialize the data
> innov <- rnorm(nrows, sd=0.01)
> retp <- numeric(nrows)
> pricev <- numeric(nrows)
> # Simulate Dickey-Fuller process using recursive loop in R
> retp[1] <- innov[1]
> pricev[1] <- prici
> retp[2] <- thetav*(priceq - pricev[1]) + coeff[1]*retp[1] +
+   innov[2]
> pricev[2] <- pricev[1] + retp[2]
> retp[3] <- thetav*(priceq - pricev[2]) + coeff[1]*retp[2] +
+   coeff[2]*retp[1] + innov[3]
> pricev[3] <- pricev[2] + retp[3]
> for (it in 4:nrows) {
+   retp[it] <- thetav*(priceq - pricev[it-1]) +
+     retp[(it-1):(it-3)] %*% coeff + innov[it]
+   pricev[it] <- pricev[it-1] + retp[it]
+ }  # end for
> # Simulate Dickey-Fuller process in Rcpp
> pricecpp <- HighFreq::sim_df(prici=prici, priceq=priceq,
+    theta=thetav, coeff=matrix(coeff), innov=matrix(innov))
> # Compare prices
> all.equal(pricev, drop(pricecpp))
> # Compare the speed of R code with Rcpp
> library(microbenchmark)
> summary(microbenchmark(
+   Rcode={for (it in 4:nrows) {
+   retp[it] <- thetav*(priceq - pricev[it-1]) + retp[(it-1):(it-3)]
+   pricev[it] <- pricev[it-1] + retp[it]
+   }},
+   Rcpp=HighFreq::sim_df(prici=prici, priceq=priceq, theta=thetav,
+   times=10))[, c(1, 4, 5)]  # end microbenchmark summary
```

Augmented Dickey-Fuller ADF Test for Unit Roots

The *Augmented Dickey-Fuller ADF* test is designed to test the *null hypothesis* that a time series has a *unit root*.

The *ADF* test fits an autoregressive model for the prices p_t:

$$r_t = \theta(\mu - p_{t-1}) + \varphi_1 r_{t-1} + \ldots + \varphi_n r_{t-n} + \sigma \xi_t$$

$$p_t = p_{t-1} + r_t$$

Where μ is the equilibrium price, σ is the volatility of returns, and θ is the strength of mean reversion.

ε_i are the *residuals*, which are assumed to be standard normally distributed $\phi(0, \sigma_\varepsilon)$, independent, and stationary.

If the mean reversion parameter θ is positive: $\theta > 0$, then the time series p_t exhibits mean reversion and has no *unit root*.

The *null hypothesis* is that prices have a unit root ($\theta = 0$, no mean reversion), while the alternative hypothesis is that it's *stationary* ($\theta > 0$, mean reversion).

The *ADF* test statistic is equal to the *t*-value of the θ parameter: $t_\theta = \hat{\theta}/SE_\theta$ (which follows a different distribution from the t-distribution).

The function `tseries::adf.test()` performs the *ADF* test.

```
> # Simulate AR(1) process with coefficient=1, with unit root
> innov <- matrix(rnorm(1e4, sd=0.01))
> arimav <- HighFreq::sim_ar(coeff=matrix(1), innov=innov)
> plot(arimav, t="l", main="Brownian Motion")
> # Perform ADF test with lag = 1
> tseries::adf.test(arimav, k=1)
> # Perform standard Dickey-Fuller test
> tseries::adf.test(arimav, k=0)
> # Simulate AR(1) with coefficient close to 1, without unit root
> arimav <- HighFreq::sim_ar(coeff=matrix(0.99), innov=innov)
> plot(arimav, t="l", main="AR(1) coefficient = 0.99")
> tseries::adf.test(arimav, k=1)
> # Simulate Ornstein-Uhlenbeck OU process with mean reversion
> prici <- 0.0; priceq <- 0.0; thetav <- 0.1
> pricev <- HighFreq::sim_ou(prici=prici, priceq=priceq,
+   theta=thetav, innov=innov)
> plot(pricev, t="l", main=paste("OU coefficient =", thetav))
> tseries::adf.test(pricev, k=1)
> # Simulate Ornstein-Uhlenbeck OU process with zero reversion
> thetav <- 0.0
> pricev <- HighFreq::sim_ou(prici=prici, priceq=priceq,
+   theta=thetav, innov=innov)
> plot(pricev, t="l", main=paste("OU coefficient =", thetav))
> tseries::adf.test(pricev, k=1)
```

The common practice is to use a small number of lags in the *ADF* test, and if the residuals are autocorrelated, then to increase them until the correlations are no longer significant.

If the number of lags in the regression is zero: $n = 0$ then the *ADF* test becomes the standard *Dickey-Fuller* test: $r_t = \theta(\mu - p_{t-1}) + \varepsilon_i$.

Cointegration of Stocks Prices

A group of stocks is *cointegrated* if there is a portfolio of the stocks whose price is range-bound.

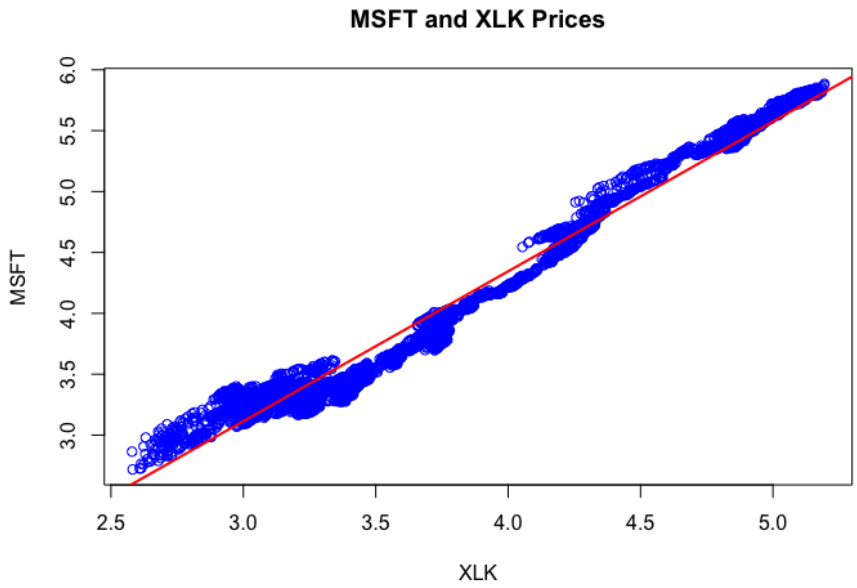
For two stocks, the cointegrating factor β can be obtained from the regression of the stock prices. The cointegrated portfolio price R is the residual of the regression:

$$R = p_1 - \beta p_2$$

Regressing the stock *prices* produces a *cointegrated* portfolio, with a small variance of its *prices*.

Regressing the stock *returns* produces a *correlated* portfolio, with a small variance of its *returns*.

The standard confidence intervals are not valid for the regression of prices, because prices are not stationary and are not normally distributed.




```
> # Load daily S&P500 stock prices
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_prices
> # Combine MSFT and XLK prices
> pricev <- cbind(pricestock$MSFT, rutils::etfenv$prices$XLK)
> pricev <- log(na.omit(pricev))
> colnames(pricev) <- c("MSFT", "XLK")
> datev <- zoo::index(pricev)
> # Calculate the beta regression coefficient of prices MSFT ~ XLK
> betac <- drop(cov(pricev$MSFT, pricev$XLK)/var(pricev$XLK))
> # Calculate the cointegrated portfolio prices
> pricec <- pricev$MSFT - betac*pricev$XLK
> colnames(pricec) <- "MSFT Coint XLK"
```

```
> # Scatterplot of MSFT and XLK prices
> plot(MSFT ~ XLK, data=pricev, main="MSFT and XLK Prices",
+      xlab="XLK", ylab="MSFT", pch=1, col="blue")
> abline(a=mean(pricec), b=betac, col="red", lwd=2)
> # Plot time series of prices
> endd <- rutils::calc_endpoints(pricev, interval="weeks")
> dygraphs::dygraph(pricev[endd], main="MSFT and XLK Log Prices") %>
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
```

Cointegrated Portfolio Prices

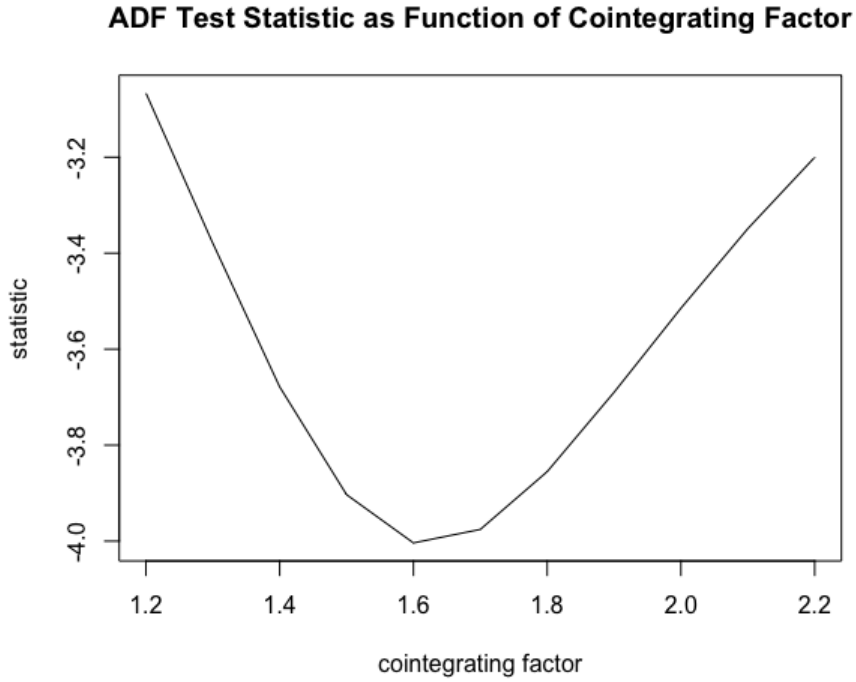
The Engle-Granger two-step procedure can be used to test the cointegrated portfolio of two stocks:

- Perform a regression of the stock prices to calculate the cointegrating factor β,
- Apply the *ADF* test to the cointegrated portfolio price, to determine if it has a unit root (the portfolio price diverges), or if it's mean reverting.

The *p*-value of the *ADF* test for the cointegrated portfolio of *MSFT* and *XLK* is not small, so the *null hypothesis* that it has a *unit root* (it diverges) cannot be rejected.

The *null hypothesis* of the *ADF* test is that the time series has a *unit root* (it diverges). So a small *p*-value suggests that the *null hypothesis* is `FALSE` and that the time series is range-bound.

The *ADF* test statistic for the cointegrated portfolio is smaller than for either *MSFT* or *XLK* alone, which indicates that it's more mean-reverting.




```
> # Plot histogram of the cointegrated portfolio prices
> hist(pricec, breaks=100, xlab="Prices",
+   main="Histogram of Cointegrated Portfolio Prices")
> # Plot of cointegrated portfolio prices
> datav <- cbind(pricev$XLK, pricec)[endd]
> colnames(datav)[2] <- "Cointegrated Portfolio"
> colv <- colnames(datav)
> dygraphs::dygraph(datav, main="MSFT and XLK Cointegrated Portfol
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
```

```
> # Perform ADF test on the individual stocks
> sapply(pricev, tseries::adf.test, k=1)
> # Perform ADF test on the cointegrated portfolio
> tseries::adf.test(pricec, k=1)
> # Perform ADF test for vector of cointegrating factors
> betas <- seq(1.2, 2.2, 0.1)
> adfstat <- sapply(betas, function(betac) {
+   pricec <- (pricev$MSFT - betac*pricev$XLK)
+   tseries::adf.test(pricec, k=1)$statistic
+ })  # end sapply
> # Plot ADF statistics for vector of cointegrating factors
> plot(x=betas, y=adfstat, type="l", xlab="cointegrating factor", yl
+  main="ADF Test Statistic as Function of Cointegrating Factor")
```

Bollinger Band Strategy for Cointegrated Pairs

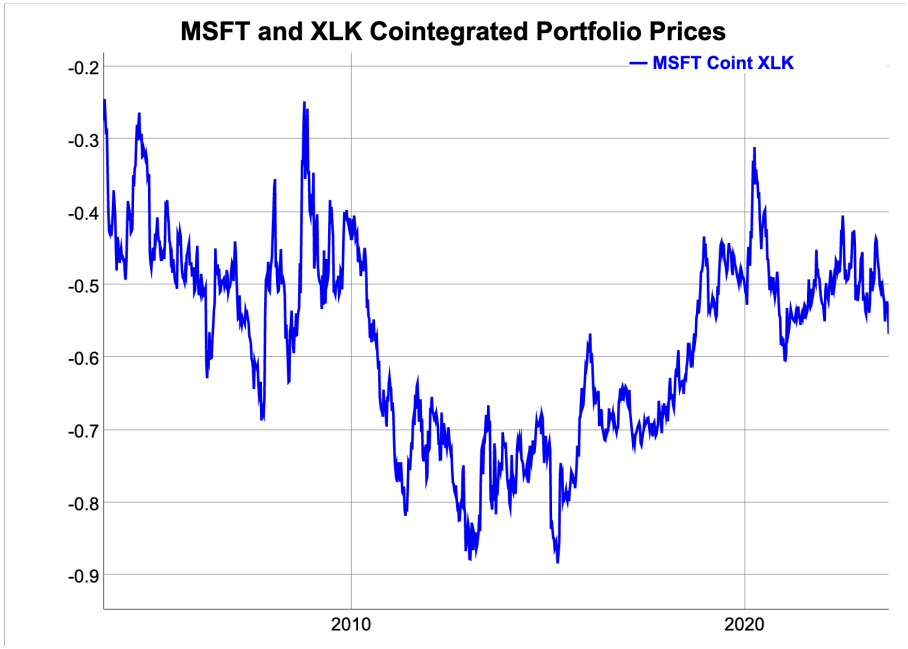
The returns of the cointegrated portfolio have negative autocorrelations, so it can be traded using a mean-reverting strategy.

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

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

The portfolio is cheap if its price is below its mean price minus the portfolio standard deviation, and it's rich (expensive) if its price is above the mean plus the standard deviation. The portfolio price is fair if it's equal to the mean price.

The pairs strategy is path dependent because it depends on the risk position. So the simulation requires performing a loop.



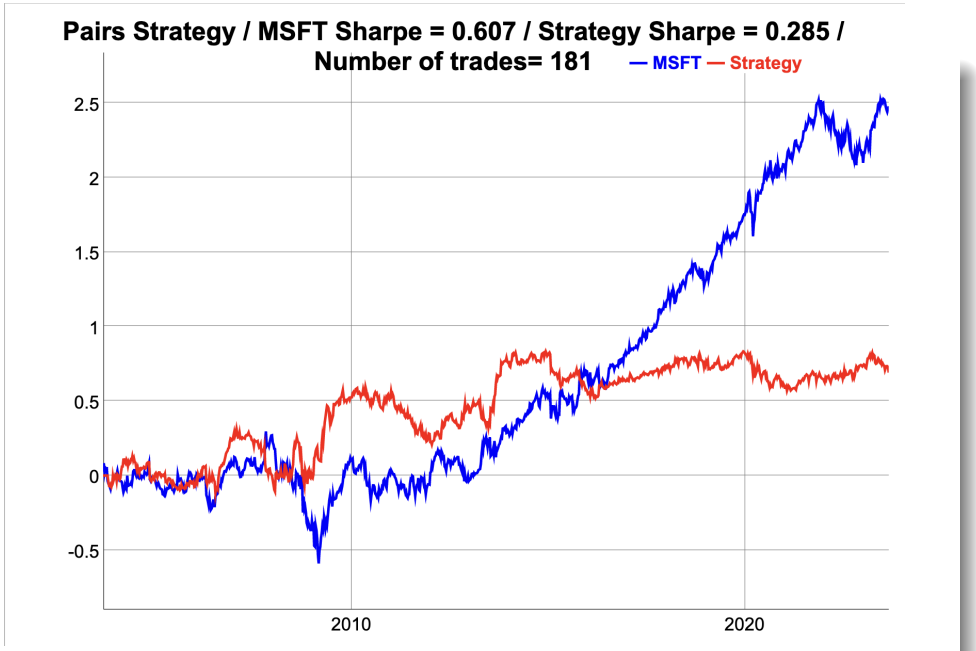

```
> # Plot of PACF of the cointegrated portfolio returns
> pricen <- zoo::coredata(pricec) # Numeric price
> retd <- rutils::diffit(pricen)
> pacf(retd, lag=10, xlab=NA, ylab=NA,
+      main="PACF of Cointegrated Portfolio Returns")
> # Dygraphs plot of cointegrated portfolio prices
> endd <- rutils::calc_endpoints(pricec, interval="weeks")
> dygraphs::dygraph(pricec[endd], main=
+   "MSFT and XLK Cointegrated Portfolio Prices") %>%
+   dyOptions(colors=c("blue"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Pairs Strategy With Fixed Beta

The pairs strategy performs well because it's in-sample - it uses the in-sample beta.

A more realistic strategy requires calculating the trailing betas on past prices, updating the pairs price, and updating the trailing mean and volatility.

```
> # Calculate the trailing mean prices and volatilities
> lambdaf <- 0.9
> volp <- HighFreq::run_var(pricen, lambda=lambdaf)
> meanv <- volp[, 1]
> volp <- sqrt(volp[, 2])
> # Simulate the pairs Bollinger strategy
> pricem <- pricen - meanv # De-meaned price
> nrows <- NROW(pricec)
> threshd <- rutils::lagit(volp)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(pricem > threshd, -1, posv)
> posv <- ifelse(pricem < -threshd, 1, posv)
> posv <- zoo::na.locf(posv)
> # Lag the positions to trade in the next period
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the pnls and the number of trades
> retp <- rutils::diffit(pricev)
> pnls <-  posv*(retp$MSFT - betac*retp$XLK)
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
```



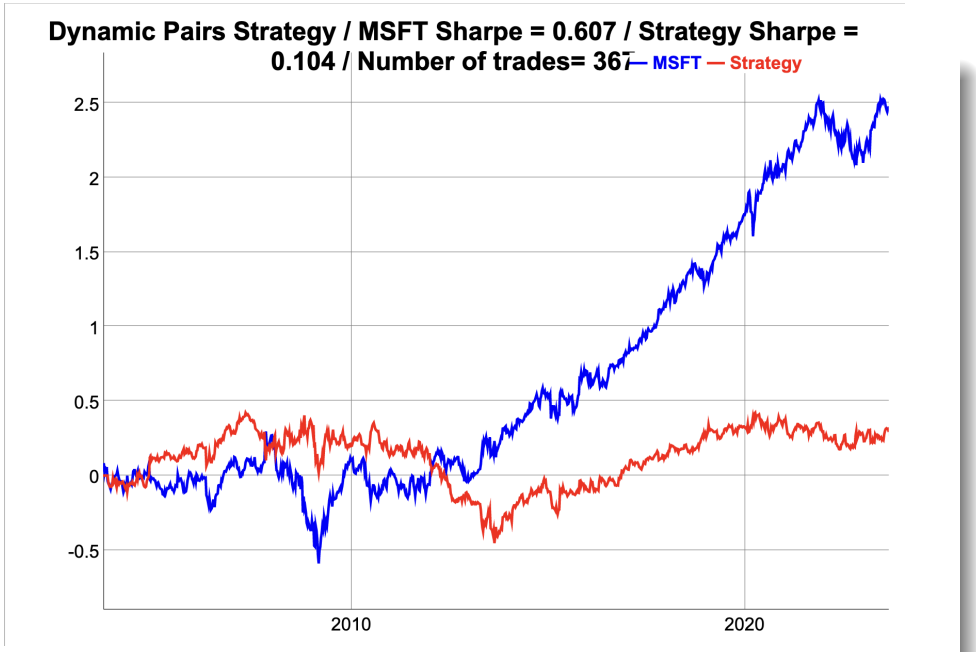

```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp$MSFT, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]
> sharper <- round(sharper, 3)
> # Dygraphs plot of pairs Bollinger strategy
> colv <- colnames(wealthv)
> captiont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

Pairs Strategy With Dynamic Beta

In practice, the stock beta should be updated over time and applied out-of-sample to calculate the trailing cointegrated pair prices.

Using a dynamic beta produces poor performance for many stock and ETF pairs.

```
> # Calculate the trailing cointegrated pair prices
> covarv <- HighFreq::run_covar(pricev, lambdaf)
> betac <- covarv[, 1]/covarv[, 3]
> pricec <- (pricev$MSFT - betac*pricev$XLK)
> # Recalculate the mean of cointegrated portfolio prices
> volp <- HighFreq::run_var(pricec, lambda=lambdaf)
> meanv <- volp[, 1]
> volp <- sqrt(volp[, 2])
> # Simulate the pairs Bollinger strategy
> pricen <- zoo::coredata(pricec) # Numeric price
> pricem <- pricen - meanv # De-meaned price
> threshd <- rutils::lagit(volp)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(pricem > threshd, -1, posv)
> posv <- ifelse(pricem < -threshd, 1, posv)
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the pnls and the number of trades
> retp <- rutils::diffit(pricev)
> pnls <-  posv*(retp$MSFT - betac*retp$XLK)
> ntrades <- sum(abs(rutils::diffit(posv)) > 0)
```



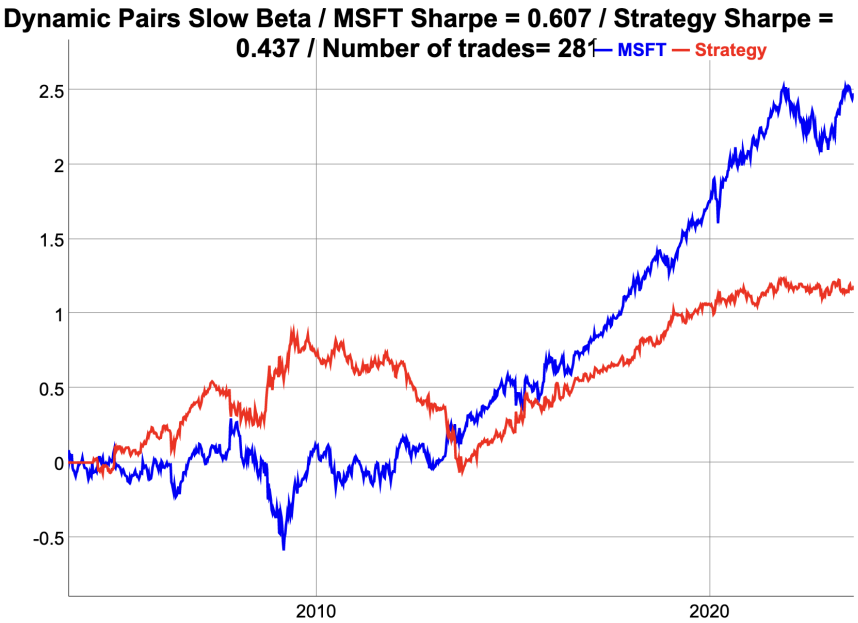

```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp$MSFT, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]
> sharper <- round(sharper, 3)
> # Dygraphs plot of pairs Bollinger strategy
> colv <- colnames(wealthv)
> captiont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

Pairs Strategy With Slow Beta

The pairs strategy can be improved by introducing an independent decay factor λ_β for calculating the trailing stock β.

The pairs strategy performs better with fast decay for calculating the trailing pairs volatility and slow decay for calculating the trailing stock β.

Independent decay factors for the trailing stock β and for the trailing pairs volatility improve the pairs strategy performance, but they also increase the risk of overfitting the model, because the more model parameters, the greater the risk of overfitting.




```
> # Calculate the trailing cointegrated pair prices
> covarv <- HighFreq::run_covar(pricev, lambda=0.95)
> betac <- covarv[, 1]/covarv[, 3]
> pricec <- (pricev$MSFT - betac*pricev$XLK)
> # Recalculate the mean of cointegrated portfolio prices
> volp <- HighFreq::run_var(pricec, lambda=0.3)
> meanv <- volp[, 1]
> volp <- sqrt(volp[, 2])
> # Simulate the pairs Bollinger strategy
> pricen <- zoo::coredata(pricec) # Numeric price
> pricem <- pricen - meanv # De-meaned price
> threshd <- rutils::lagit(volp)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(pricem > threshd, -1, posv)
> posv <- ifelse(pricem < -threshd, 1, posv)
> posv <- zoo::na.locf(posv)
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the pnls and the number of trades
> retp <- rutils::diffit(pricev)
> pnls <-  posv*(retp$MSFT - betac*retp$XLK)
```

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

Stock Index Weighting Methods

Stock market indices can be either capitalization-weighted, price-weighted, or equal wealth.

The cap-weighted index is equal to the average of the market capitalizations of all its companies (stock price times number of shares). The *S&P500* index is cap-weighted.

The cap-weighted index is equivalent to owning a fixed number of shares, proportional to the number of shares outstanding. So if company X has twice as many shares outstanding as company Y, then the cap-weighted index will own twice as many shares of company X as company Y.

The price-weighted index is equal to the average of the stock prices. The price-weighted index is equivalent to owning a fixed and equal number of shares. The *DJIA* index is price-weighted.

The equal wealth index invests equal dollar amounts in each stock, and it rebalances its weights as market prices change.

The cap-weighted and price-weighted indices are overweight large-cap stocks, compared to the equal wealth index which has larger weights for small-cap stocks.

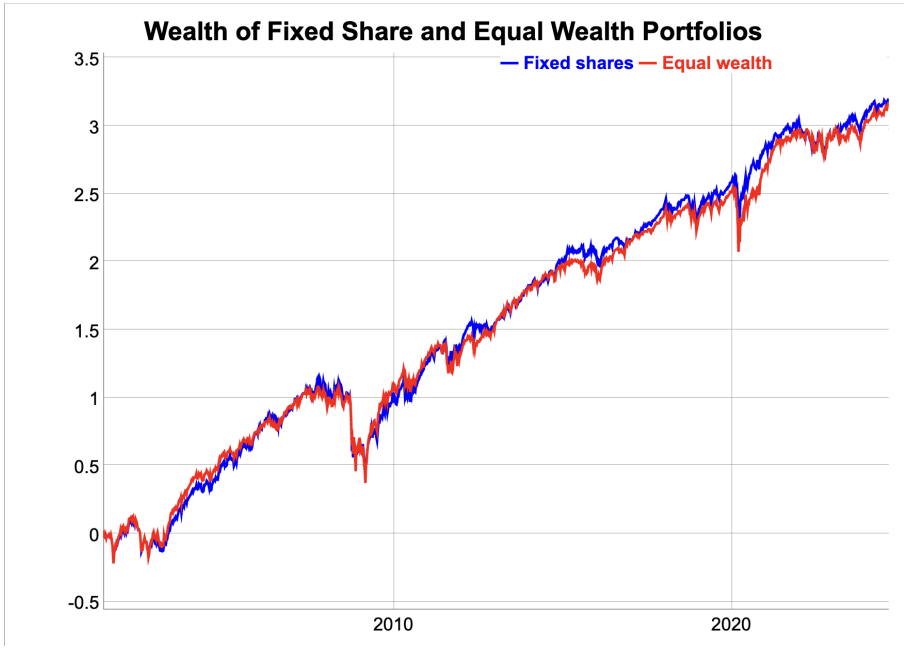
```
> # Calculate the percentage VTI returns
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> datev <- zoo::index(retvti)
> nrows <- NROW(retvti)
> # Load daily S&P500 stock prices
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_prices.R
> # Select the stock prices since VTI
> pricestock <- pricestock[datev]
> # Select stocks with no NA values in their prices
> numna <- sapply(pricestock, function(x) sum(is.na(x)))
> pricestock <- pricestock[, numna == 0]
> # Drop penny stocks
> pricel <- last(pricestock)
> pricel <- drop(coredata(pricel))
> pricestock <- pricestock[, pricel > 1]
> # Calculate the dollar and percentage stock returns
> retd <- rutils::diffit(pricestock)
> retp <- retd/rutils::lagit(pricestock)
> retp[1, ] <- 0
```

The Equal Wealth Portfolio

The equal wealth portfolio rebalances its weights - it sells the stocks with higher returns and buys stocks with lower returns. So it's a *mean reverting* (contrarian) strategy.

The equal wealth portfolio can often underperform the cap-weighted and fixed share indices because it gradually overweights underperforming stocks, as it rebalances to maintain equal wealth allocations.

In periods when a small number of stocks dominate returns, the cap-weighted and fixed share indices outperform the equal wealth index.




```
> # Wealth of fixed shares portfolio
> wealthfs <- rowMeans(cumprod(1 + retp))
> # Wealth of equal wealth portfolio (with rebalancing)
> wealthew <- cumprod(1 + rowMeans(retp))
```

```
> # Calculate combined log wealth
> wealthv <- cbind(wealthfs, wealthew)
> wealthv <- log(wealthv)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Fixed shares", "Equal wealth")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of combined log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Fixed Share and Equal Wealth Portfolios") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Random Stock Selection

A random portfolio is a sub-portfolio of stocks selected at random.

Random portfolios are used as a benchmark for stock pickers (portfolio managers).

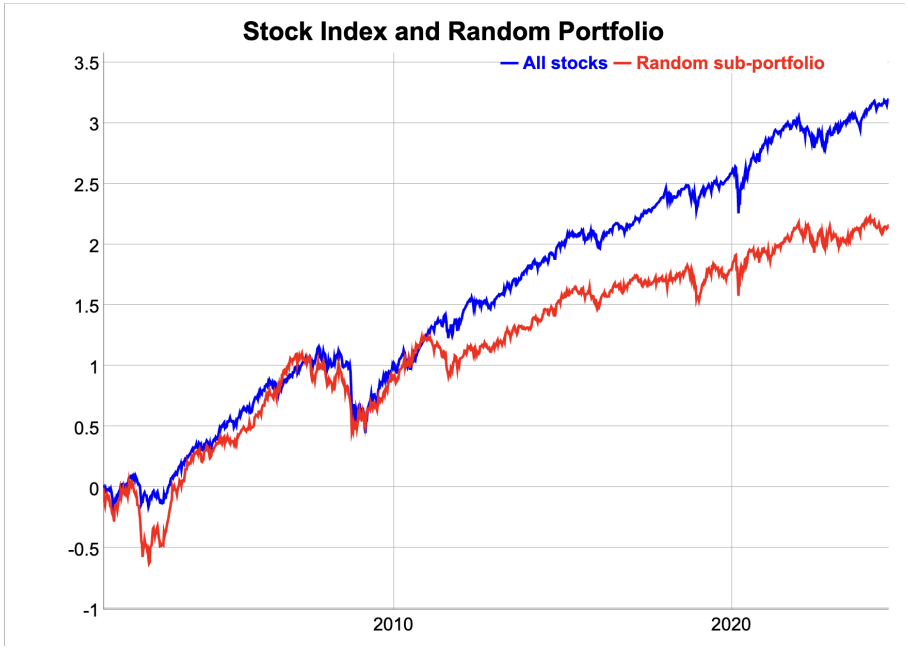
If a portfolio manager outperforms the median of random portfolios, then they may have stock picking skill.

According to S&P Global, 95% of large-cap actively managed funds have underperformed their benchmark. And the underperformance increases for longer holding periods, because the distribution of stock prices becomes more and more skewed over time.

Charlie Munger (vice chairman of Berkshire Hathaway) said that "Most money managers are little more than fortune tellers or astrologers."

John Bogle (founder of The Vanguard Group) said, "Don't look for the needle in the haystack. Just buy the whole haystack."

```
> # Select a random, fixed share sub-portfolio of 5 stocks
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> nstocks <- NCOL(retp)
> samplev <- sample.int(n=nstocks, size=5, replace=FALSE)
> wealthr <- rowMeans(cumprod(1 + retp[, samplev]))
```




```
> # Plot dygraph of all stocks and random sub-portfolio
> wealthv <- cbind(wealthfs, wealthr)
> wealthv <- log(wealthv)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("All stocks", "Random sub-portfolio")
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endd], main="Stock Index and Random Port
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

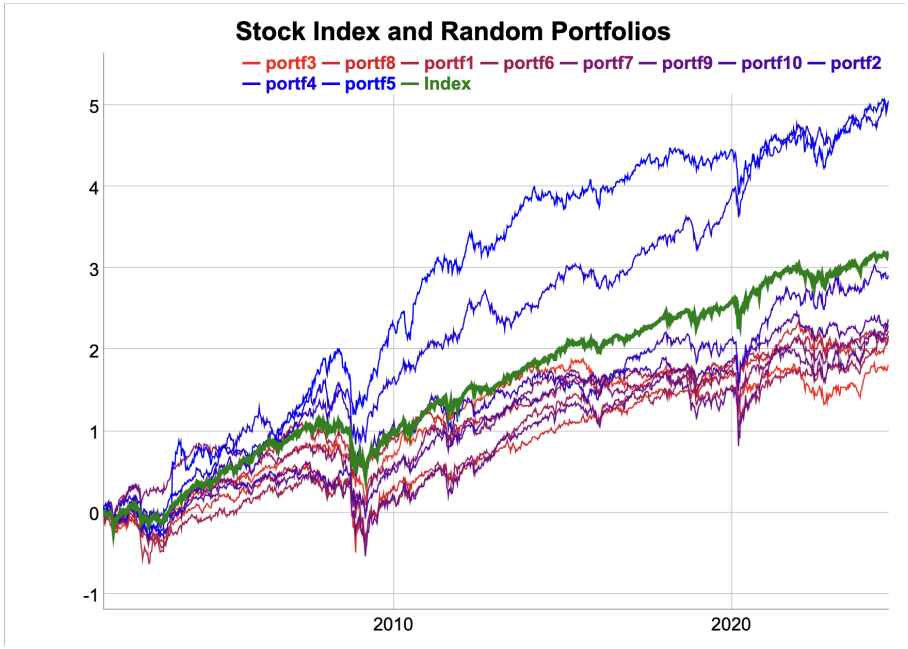
Random Stock Portfolios

Most random portfolios underperform the index, so picking a portfolio which outperforms the stock index requires great skill.

An investor without skill, who selects stocks at random, has a high probability of underperforming the index, because they will most likely miss selecting the best performing stocks.

Therefore the proper benchmark for a stock picker is the median of random portfolios, not the stock index, which is the mean of all the stock prices.

Performing as well as the index requires *significant* investment skill, while outperforming the index requires *exceptional* investment skill.




```
> # Select 10 random fixed share sub-portfolios
> set.seed(1121, "Mersenne-Twister", sample.kind="Rejection")
> nportf <- 10
> wealthr <- sapply(1:nportf, function(x) {
+   samplev <- sample.int(n=nstocks, size=5, replace=FALSE)
+   rowMeans(cumprod(1 + retp[, samplev]))
+ })  # end sapply
> wealthr <- xts::xts(wealthr, order.by=datev)
> colnames(wealthr) <- paste0("portf", 1:nportf)
> # Sort the sub-portfolios according to performance
> wealthr <- wealthr[, order(wealthr[nrows])]
> round(head(wealthr), 3)
> round(tail(wealthr), 3)
```

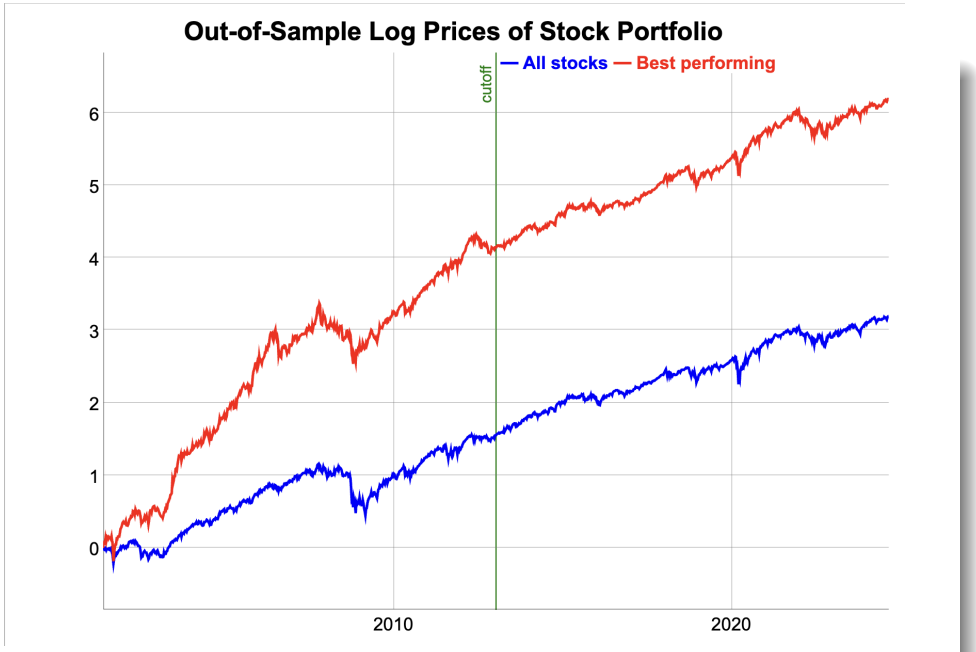
```
> # Plot dygraph of all stock index and random sub-portfolios
> colorv <- colorRampPalette(c("red", "blue"))(nportf)
> colorv <- c("green", colorv)
> wealthv <- cbind(wealthfs, wealthr)
> wealthv <- log(wealthv)
> colnames(wealthv)[1] <- "Index"
> colv <- colnames(wealthv)
> dygraphs::dygraph(wealthv[endd], main="Stock Index and Random Port
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name=colv[1], strokeWidth=3) %>%
+   dyLegend(show="always", width=500)
```

Stock Portfolio Selection Out-of-Sample

The strategy selects the 10 best performing stocks at the end of the in-sample interval, and invests in them in the out-of-sample interval.

The strategy buys equal and fixed number of shares of stocks, and at the end of the in-sample interval, selects the 10 best performing stocks. It then invests the same number of shares in the out-of-sample interval.

The out-of-sample performance of the best performing stocks is not any better than the index.




```
> # Define in-sample and out-of-sample intervals
> cutoff <- nrows %/% 2
> datev[cutoff]
> insample <- 1:cutoff
> outsample <- (cutoff + 1):nrows
> # Calculate the 10 best performing stocks in-sample
> pricev <- cumprod(1 + retp)
> pricet <- pricev[cutoff, ]
> pricet <- drop(coredata(pricet))
> pricet <- sort(pricet, decreasing=TRUE)
> symbolv <- names(head(pricet, 10))
> # Calculate the wealth of the 10 best performing stocks
> wealthb <- rowMeans(pricev[insample, symbolv])
> wealthos <- wealthb[cutoff]*rowMeans(cumprod(1 + retp[outsample,
> wealthb <- c(wealthb, wealthos)
```

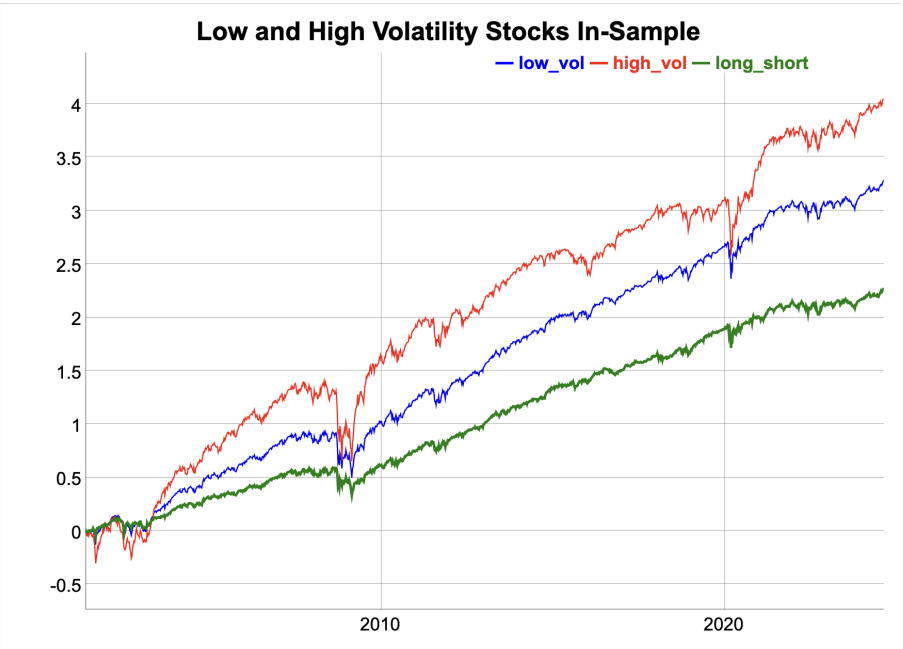
```
> # Combine the fixed share wealth with the 10 best performing stock
> wealthv <- cbind(wealthfs, wealthb)
> wealthv <- xts::xts(log(wealthv), order.by=datev)
> colnames(wealthv) <- c("All stocks", "Best performing")
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv[insample, ]),
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv[outsample, ]),
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot out-of-sample stock portfolio returns
> dygraphs::dygraph(wealthv[endd], main="Out-of-Sample Log Prices of
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(datev[cutoff], label="cutoff", strokePattern="solid", co
+   dyLegend(width=300)
```

Low and High Volatility Stock Portfolios

Research by Robeco, Eric Falkenstein, and others has shown that low volatility stocks have outperformed high volatility stocks.

Betting against volatility is a strategy which invests in low volatility stocks and shorts high volatility stocks.

USMV is an *ETF* that holds low volatility stocks, although it hasn't met expectations.




```
> # Calculate the stock volatilities, betas, and alphas
> # Perform parallel loop under Mac-OSX or Linux
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> riskret <- mclapply(retp, function(rets) {
+   rets <- na.omit(rets)
+   stdev <- sd(rets)
+   retvti <- retvti[zoo::index(rets)]
+   varvti <- drop(var(retvti))
+   meanvti <- mean(retvti)
+   betac <- drop(cov(rets, retvti))/varvti
+   resid <- rets - betac*retvti
+   alphac <- mean(rets) - betac*meanvti
+   c(alpha=alphac, beta=betac, stdev=stdev, ivol=sd(resid))
+ }, mc.cores=ncores)  # end mclapply
> riskret <- do.call(rbind, riskret)
> tail(riskret)
> # Calculate the median volatility
> riskv <- riskret[, "stdev"]
> medianv <- median(riskv)
> # Calculate the returns of low and high volatility stocks
> retlow <- rowMeans(retp[, (riskv <= medianv)], na.rm=TRUE)
> rethigh <- rowMeans(retp[, (riskv > medianv)], na.rm=TRUE)
```

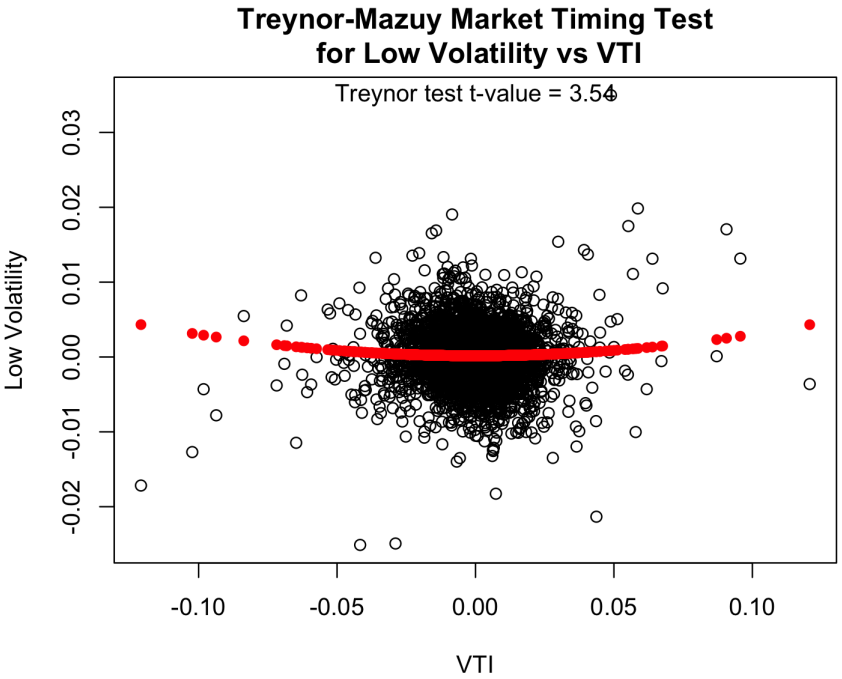
```
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colv <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colv
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high volatility stocks
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Low and High Volati
+   dySeries(name=colv[1], col="blue", strokeWidth=1) %>%
+   dySeries(name=colv[2], col="red", strokeWidth=1) %>%
+   dySeries(name=colv[3], col="green", strokeWidth=2) %>%
+   dyLegend(width=300)
```

Low Volatility Stock Portfolio Market Timing Skill

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

The *Treynor-Mazuy* test shows that the *betting against volatility* strategy has very small *market timing* skill.

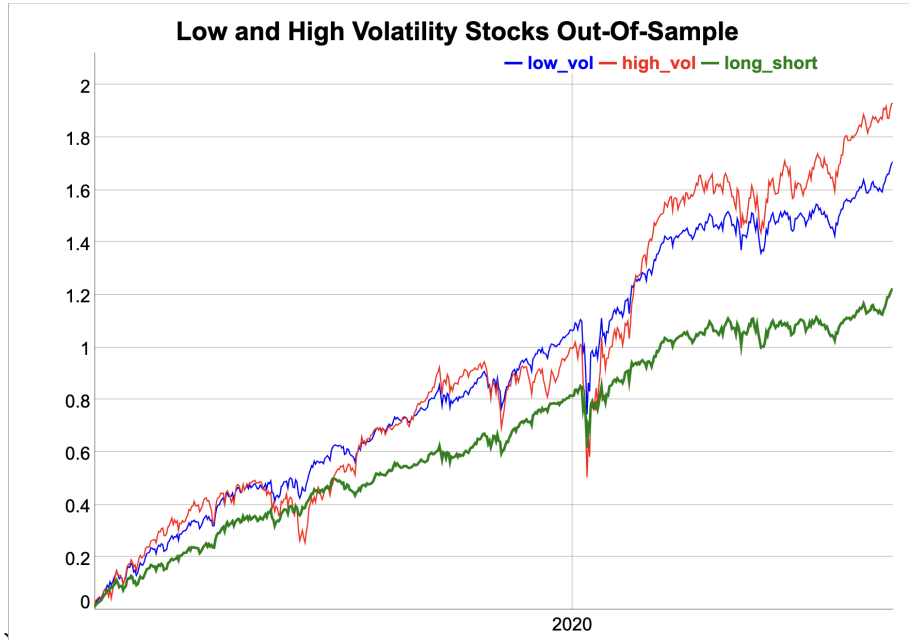
```
> # Merton-Henriksson test
> desm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(desm)[2:3] <- c("Merton", "Treynor")
> regmod <- lm(wealthv$long_short ~ VTI + Merton, data=desm); summar
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + Treynor, data=desm); summa
> # Plot residual scatterplot
> resids <- regmod$residuals
> plot.default(x=retvti, y=resids, xlab="VTI", ylab="Low Volatility'
> title(main="Treynor-Mazuy Market Timing Test\n for Low Volatility
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retvti
> tvalue <- round(coefreg["Treynor", "t value"], 2)
> points.default(x=retvti, y=fitv, pch=16, col="red")
> text(x=0.0, y=max(resids), paste("Treynor test t-value =", tvalue))
```



Low and High Volatility Stock Portfolios Out-Of-Sample

The low volatility stocks selected in-sample also have a higher *Sharpe ratio* in the out-of-sample period than the high volatility stocks, although their absolute returns are lower.

```
> # Calculate the in-sample stock volatilities, betas, and alphas
> riskretis <- mclapply(retp[insample], function(rets) {
+   combv <- na.omit(cbind(rets, retvti))
+   if (NROW(combv) > 11) {
+     rets <- na.omit(rets)
+     stdev <- sd(rets)
+     retvti <- retvti[zoo::index(rets)]
+     varvti <- drop(var(retvti))
+     meanvti <- mean(retvti)
+     betac <- drop(cov(rets, retvti))/varvti
+     resid <- rets - betac*retvti
+     alphac <- mean(rets) - betac*meanvti
+     return(c(alpha=alphac, beta=betac, stdev=stdev, ivol=sd(resid)))
+   } else {
+     return(c(alpha=0, beta=0, stdev=0, ivol=0))
+   }  # end if
+ }, mc.cores=ncores)  # end mclapply
> riskretis <- do.call(rbind, riskretis)
> tail(riskretis)
> # Calculate the median volatility
> riskv <- riskretis[, "stdev"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high volatility
> retlow <- rowMeans(retp[outsample, (riskv <= medianv)], na.rm=TRU
> rethigh <- rowMeans(retp[outsample, (riskv > medianv)], na.rm=TRUE)
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colv <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colv
```



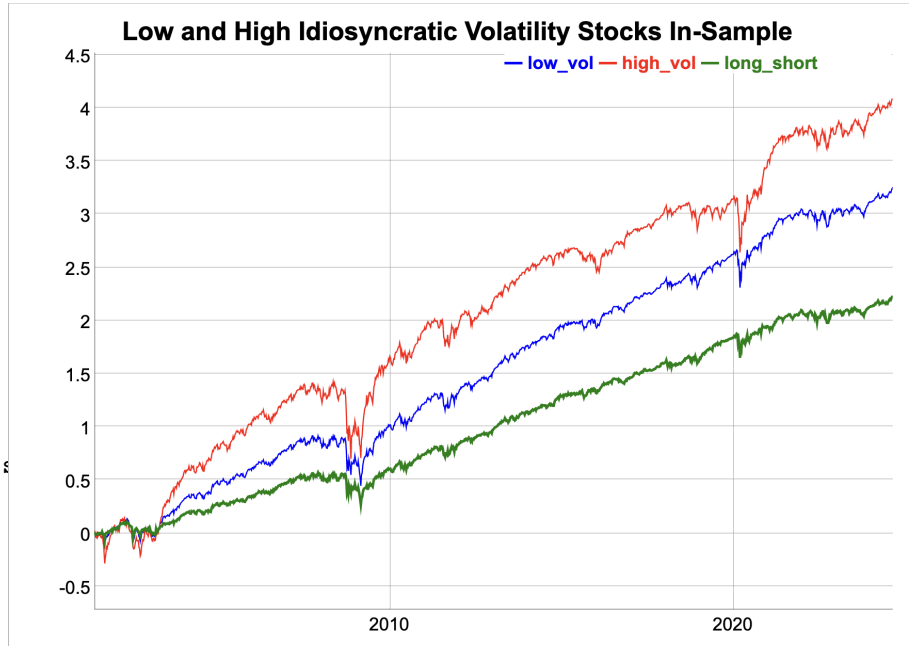

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high volatility stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Low and High Volati
+   dySeries(name=colv[1], col="blue", strokeWidth=1) %>%
+   dySeries(name=colv[2], col="red", strokeWidth=1) %>%
+   dySeries(name=colv[3], col="green", strokeWidth=2) %>%
+   dyLegend(width=300)
```

Low and High Idiosyncratic Volatility Stock Portfolios

Research by Robeco, Eric Falkenstein, and others has shown that low idiosyncratic volatility stocks have outperformed high volatility stocks.

Betting against idiosyncratic volatility is a strategy which invests in low idiosyncratic volatility stocks and shorts high volatility stocks.

```
> # Calculate the median idiosyncratic volatility
> riskv <- riskret[, "ivol"]
> medianv <- median(riskv)
> # Calculate the returns of low and high idiosyncratic volatility s
> retlow <- rowMeans(retp[, (riskv <= medianv)], na.rm=TRUE)
> rethigh <- rowMeans(retp[, (riskv > medianv)], na.rm=TRUE)
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colv <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colv
```



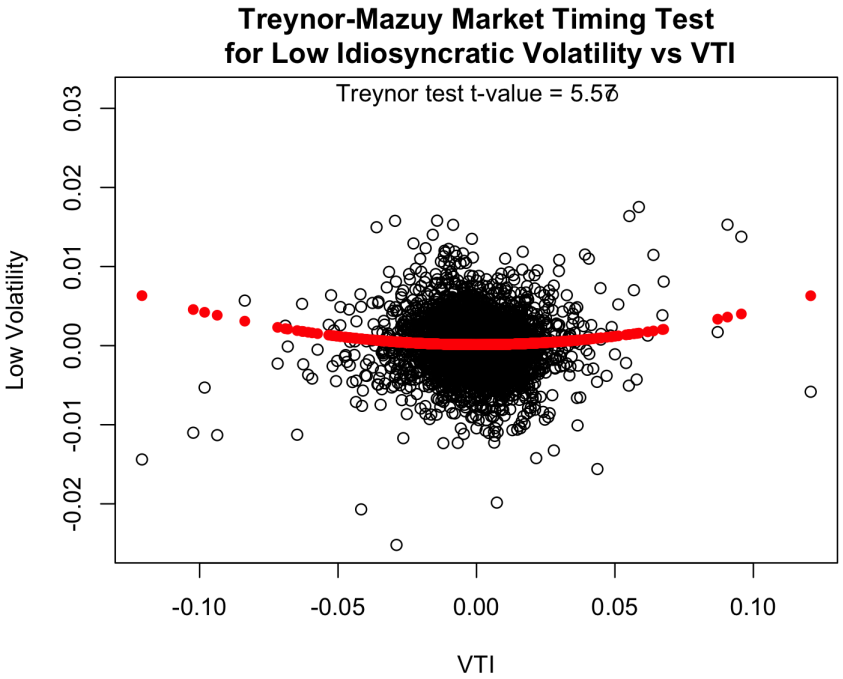

```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of returns of low and high idiosyncratic volatility stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Low and High Idiosy
+   dySeries(name=colv[1], col="blue", strokeWidth=1) %>%
+   dySeries(name=colv[2], col="red", strokeWidth=1) %>%
+   dySeries(name=colv[3], col="green", strokeWidth=2) %>%
+   dyLegend(width=300)
```

Low Idiosyncratic Volatility Stock Portfolio Market Timing Skill

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

The *Treynor-Mazuy* test shows that the *betting against idiosyncratic volatility* strategy has some *market timing* skill.

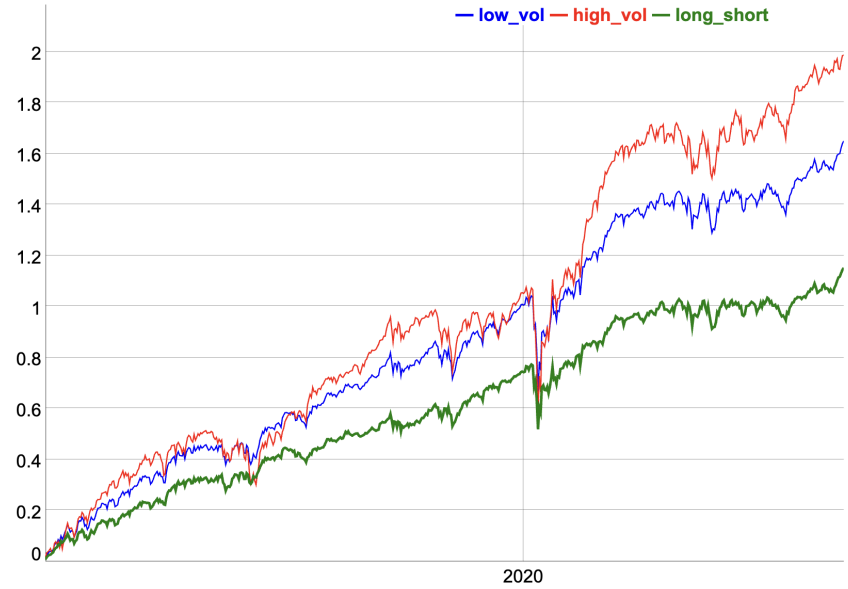
```
> # Merton-Henriksson test
> desm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(desm)[2:3] <- c("Merton", "Treynor")
> regmod <- lm(wealthv$long_short ~ VTI + Merton, data=desm); summar
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + Treynor, data=desm); summa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



Low and High Idiosyncratic Volatility Stock Portfolios Out-Of-Sample

The low idiosyncratic volatility stocks selected in-sample also have a higher *Sharpe ratio* in the out-of-sample period than the high idiosyncratic volatility stocks, although their absolute returns are lower.

```
> # Calculate the median in-sample idiosyncratic volatility
> riskv <- riskretis[, "ivol"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high idiosyncrati
> retlow <- rowMeans(retp[outsample, (riskv <= medianv)], na.rm=TRUE
> rethigh <- rowMeans(retp[outsample, (riskv > medianv)], na.rm=TRUE
> wealthv <- cbind(retlow, rethigh, retlow - 0.25*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colv <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colv
```



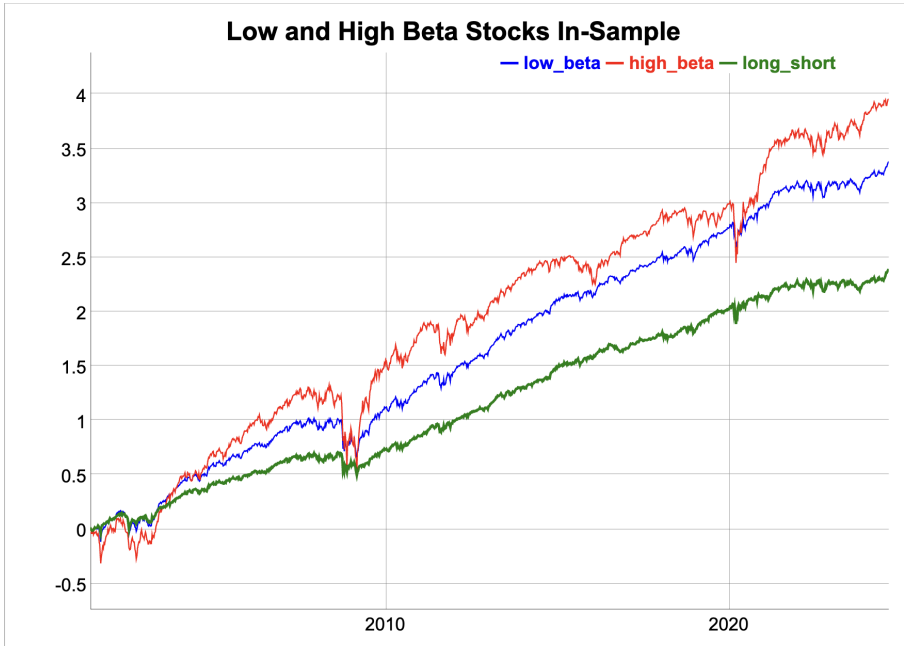

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of out-of-sample returns of low and high volatility stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Low and High Idiosy
+   dySeries(name=colv[1], col="blue", strokeWidth=1) %>%
+   dySeries(name=colv[2], col="red", strokeWidth=1) %>%
+   dySeries(name=colv[3], col="green", strokeWidth=2) %>%
+   dyLegend(width=300)
```

Low and High Beta Stock Portfolios

Research by NYU professors Andrea Frazzini and Lasse Heje Pedersen has shown that low beta stocks have outperformed high beta stocks, contrary to the *CAPM* model.

The low beta stocks are mostly from defensive stock sectors, like consumer staples, healthcare, etc., which investors buy when they fear a market selloff.

The strategy of investing in low beta stocks and shorting high beta stocks is known as betting against beta.




```
> # Calculate the median beta
> riskv <- riskret[, "beta"]
> medianv <- median(riskv)
> # Calculate the returns of low and high beta stocks
> betalow <- rowMeans(retp[, names(riskv[riskv <= medianv])], na.rm=TRUE)
> betahigh <- rowMeans(retp[, names(riskv[riskv > medianv])], na.rm
> wealthv <- cbind(betalow, betahigh, betalow - 0.25*betahigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colv <- c("low_beta", "high_beta", "long_short")
> colnames(wealthv) <- colv
```

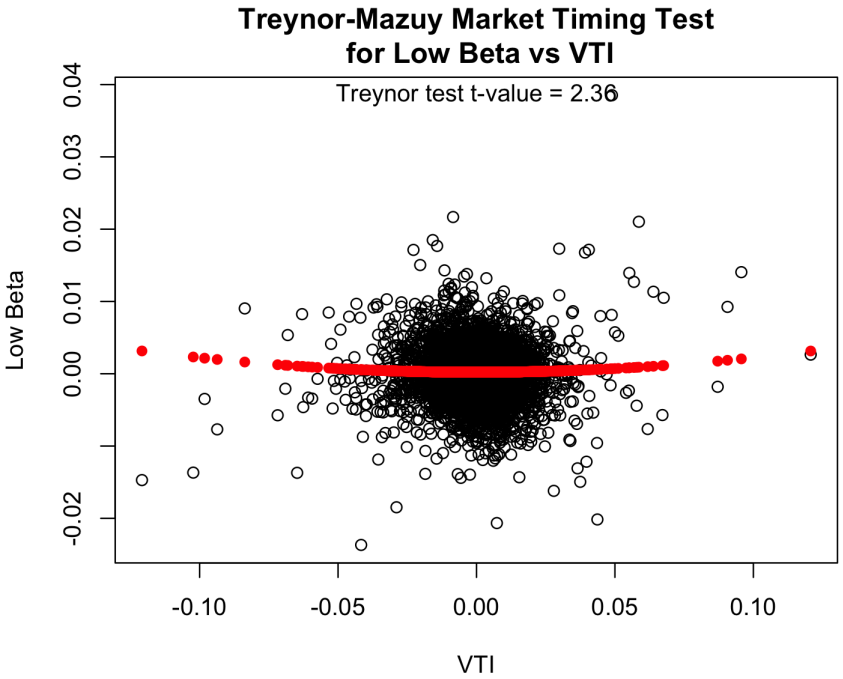
```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high beta stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Low and High Beta S
+   dySeries(name=colv[1], col="blue", strokeWidth=1) %>%
+   dySeries(name=colv[2], col="red", strokeWidth=1) %>%
+   dySeries(name=colv[3], col="green", strokeWidth=2) %>%
+   dyLegend(width=300)
```

Low Beta Stock Portfolio Market Timing Skill

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

The *Treynor-Mazuy* test shows that the *betting against beta* strategy does not have significant *market timing* skill.

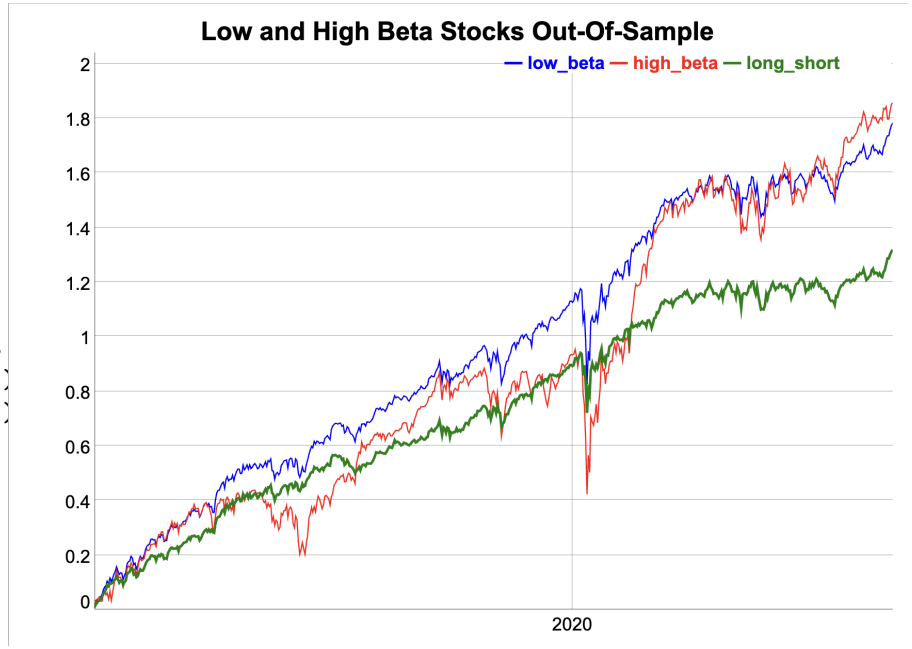
```
> # Merton-Henriksson test
> desm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(desm)[2:3] <- c("Merton", "Treynor")
> regmod <- lm(wealthv$long_short ~ VTI + Merton, data=desm); summa
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + Treynor, data=desm); summa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



Low and High Beta Stock Portfolios Out-Of-Sample

The low beta stocks selected in-sample also have a higher *Sharpe ratio* in the out-of-sample period than the high beta stocks, although their absolute returns are lower.

```
> # Calculate the median beta
> riskv <- riskretis[, "beta"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high beta stocks
> betalow <- rowMeans(retp[outsample, names(riskv[riskv <= medianv])])
> betahigh <- rowMeans(retp[outsample, names(riskv[riskv > medianv])])
> wealthv <- cbind(betalow, betahigh, betalow - 0.25*betahigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colv <- c("low_beta", "high_beta", "long_short")
> colnames(wealthv) <- colv
```




```
> # 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])))
> # Plot of out-of-sample returns of low and high beta stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Low and High Beta S
+   dySeries(name=colv[1], col="blue", strokeWidth=1) %>%
+   dySeries(name=colv[2], col="red", strokeWidth=1) %>%
+   dySeries(name=colv[3], col="green", strokeWidth=2) %>%
+   dyLegend(width=300)
```

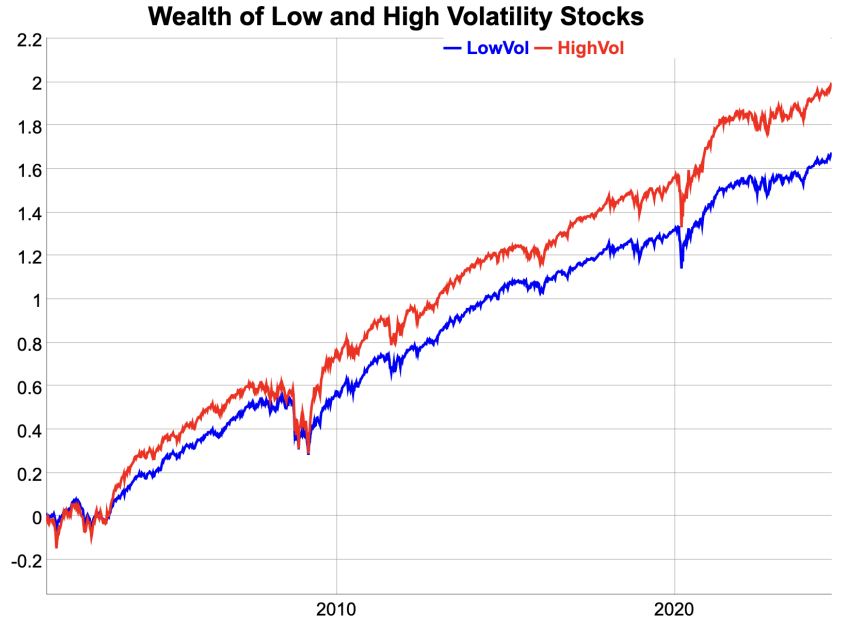
Stocks With Low and High Trailing Volatilities

The trailing volatilities can be used to create low and high volatility portfolios, and test their performance out-of-sample.

The low volatility portfolio consists of stocks with trailing volatilities less than the median, and the high portfolio with trailing volatilities greater than the median.

The portfolios are rebalanced daily, as the volatility changes.

The low volatility portfolio has a higher *Sharpe ratio*, but lower absolute returns than the high volatility portfolio.




```
> # Calculate the trailing percentage volatilities
> volp <- HighFreq::run_var(retp, lambda=0.15)
> volp <- sqrt(volp[, (nstocks+1):(2*nstocks)])
> volp <- rutils::lagit(volp)
> volp[volp == 0] <- 1
> # Calculate the median volatilities
> medianv <- matrixStats::rowMedians(volp)
> # Calculate the wealth of low volatility stocks
> weightv <- (volp <= medianv)
> weightv <- rutils::lagit(weightv)
> retlow <- rowMeans(weightv*retp)
> # Calculate the wealth of high volatility stocks
> weightv <- (volp > medianv)
> weightv <- rutils::lagit(weightv)
> rethigh <- rowMeans(weightv*retp)
```

```
> # Combined wealth
> wealthv <- cbind(retlow, rethigh)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("LowVol", "HighVol")
> # 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 log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Wealth of Low and High Volatility Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

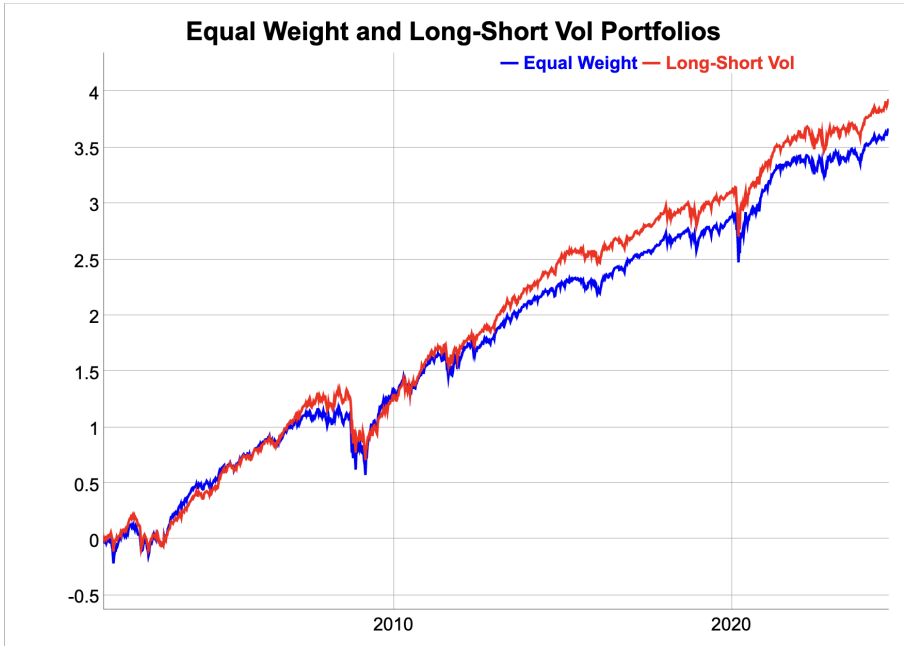
Long-Short Stock Volatility Strategy

The *Long-Short Volatility* strategy buys the low volatility stock portfolio and shorts the high volatility portfolio.

The high volatility portfolio returns are multiplied by a factor to compensate for their higher volatility.

The *Long-Short Volatility* strategy has a higher *Sharpe ratio* and also higher absolute returns than the equal wealth strategy.

```
> # Calculate the returns of equal wealth portfolio
> retew <- rowMeans(retp, na.rm=TRUE)
> retew[1] <- 0
> # Calculate the long-short volatility returns
> retls <- (retlow - 0.25*rethigh)
> # Scale the PnL volatility to that of wealthew
> retls <- retls*sd(retew)/sd(retls)
> # Combined wealth
> wealthv <- cbind(retew, retls)
> wealthv <- xts::xts(wealthv, datev)
> colv <- c("Equal Weight", "Long-Short Vol")
> colnames(wealthv) <- colv
> # 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 log wealth
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Equal Weight and Long-Short Vol Portfolios") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Risk Parity Strategy Weights

The dollar amount of a stock with price p_i that has unit dollar volatility is equal to: $\frac{p_i}{\sigma_i^d}$.

Where σ_i^d is the dollar volatility.

So the weights of the risk parity strategy are proportional to the inverse of the stock dollar volatilities: $w_i \propto \frac{1}{\sigma_i^d}$.

The weights are rebalanced daily so that the dollar volatilities of the stock allocations (*dollar amounts*) remain equal.

The stock allocations increase when the volatility is low, and vice versa.

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

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

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

Where σ_t^2 is the trailing variance at time t.

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

```
> # Calculate the trailing dollar volatilities
> lambdav <- 0.99
> vold <- HighFreq::run_var(retd, lambda=lambdav)
> vold <- vold[, (nstocks+1):(2*nstocks)]
> vold <- sqrt(vold)
> vold[vold == 0] <- 1
> # Calculate the rolling risk parity weights
> weightv <- 1/vold
> weightv <- weightv/rowSums(weightv)
```

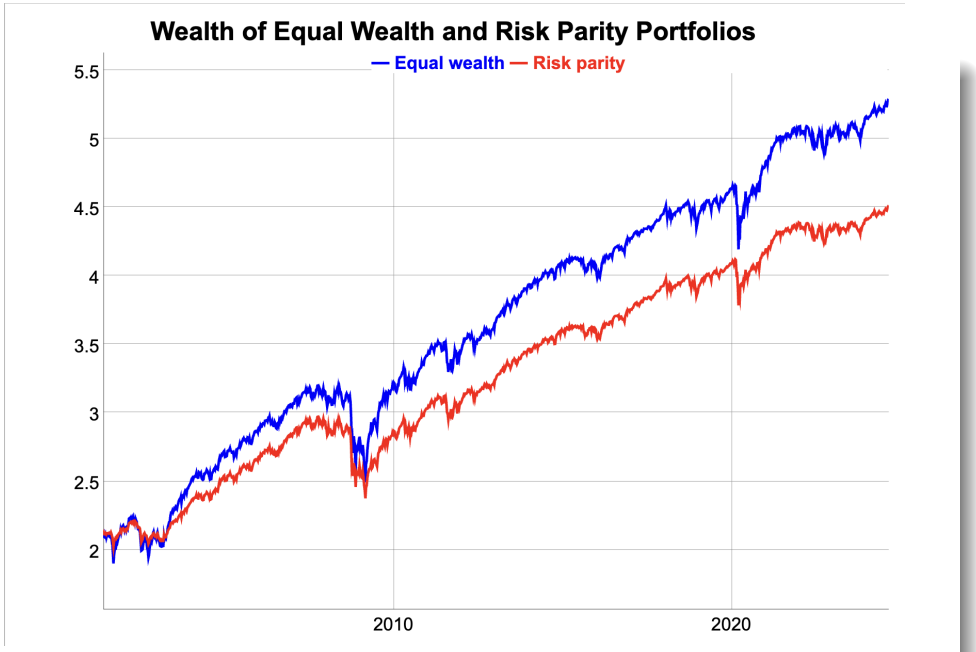
Risk Parity Strategy for Stocks

The risk parity strategy for stocks has a similar *Sharpe ratio* to the equal wealth strategy. But it has lower absolute returns.

The risk parity strategy also has higher transaction costs.

Risk parity for stocks doesn't perform as well as for assets with negative or low correlations, like stocks and bonds, because stock returns have positive correlations.

```
> # Wealth of equal wealth portfolio (with rebalancing)
> wealthew <- cumprod(1 + rowMeans(retp))
> # Calculate the risk parity allocations
> pricerp <- pricestock*weightv
> # Calculate the dollar returns of risk parity
> retrp <- retp*rutils::lagit(pricerp)
> retrp[1, ] <- pricerp[1, ]
> # Calculate the wealth of risk parity
> wealthrp <- cumsum(rowSums(retrp))
```




```
> # Combined wealth
> wealthv <- cbind(wealthrp[1]*wealthew, wealthrp)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Equal wealth", "Risk parity")
> wealthv <- log(wealthv)
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv), function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Equal Wealth and Risk Parity Portfolios") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=400)
```

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 σ^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:

$$u = \mathbb{E}[\log(1 + k_f\, r)] = \mathbb{E}[(k_f\, r - \frac{(k_f\, r)^2}{2}] =$$

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

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

$$\frac{\mathrm{d}u}{\mathrm{d}k_f} = \bar{r} - k_f \sigma^2 = 0 \;\rightarrow\; k_f = \frac{\bar{r}}{\sigma^2}$$

```
> # 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.

```
> # VTI returns
> retv <- na.omit(rutils::etfenv$returns$VTI)
> datev <- zoo::index(retv) # Dates vector
> # Load daily S&P500 percentage stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> retp <- retstock[datev]
> 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, breaks=100, main="Performance Statistics")
> 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 σ is the same as the volatility of the equal weight portfolio σ_{ew}:

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

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

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

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

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

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

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

```
> # Calculate the weighted returns using transpose
> retw <- t(t(retp)*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.

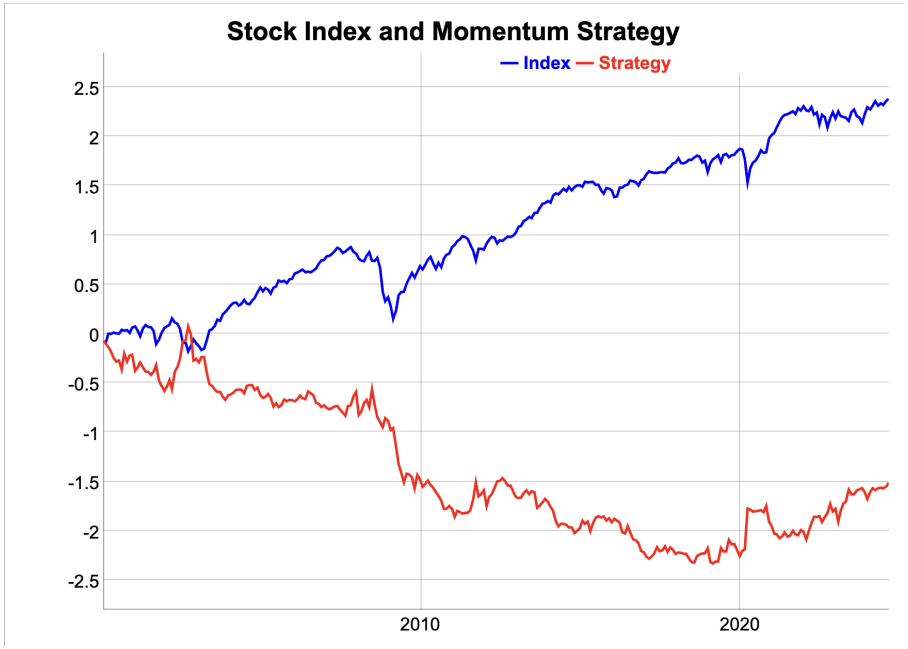
```
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> npts <- NROW(endd)
> # Perform loop over the end points
> lookb <- 8
> pnls <- lapply(3:(npts-1), function(tday) {
+   # Select the look-back returns
+   startp <- endd[max(1, tday-lookb)]
+   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
+   pnlis <- HighFreq::mult_mat(weightv, retis)
+   pnlis <- rowMeans(pnlis, na.rm=TRUE)
+   # Scale weights so in-sample pnl volatility is same as equal wei
+   weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnlis)
+   # Calculate the out-of-sample momentum returns
+   pnlos <- HighFreq::mult_mat(weightv, retp[(endd[tday]+1):endd[to
+   pnlos <- rowMeans(pnlos, na.rm=TRUE)
+   drop(pnlos)
+ })  # 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(retp, na.rm=TRUE)
> # Add initial startup interval to the momentum returns
> pnls <- c(retew[endd[1]:endd[3]], pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("Equal", "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)[endd],
+   main="Stock Index and Momentum Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Momentum Strategy Functional

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

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

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

```
> btmomtop <- function(retp, objfun, lookb=12, rebalf="months", tops
+  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
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     # Scale weights so in-sample pnl volatility is same as equal
+     weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnlis)
+     # Calculate the out-of-sample momentum returns
+     pnlos <- HighFreq::mult_mat(weightv, retp[(endd[tday]+1):endd
+     pnlos <- rowMeans(pnlos, na.rm=TRUE)
+     drop(pnlos)
+   })  # 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.

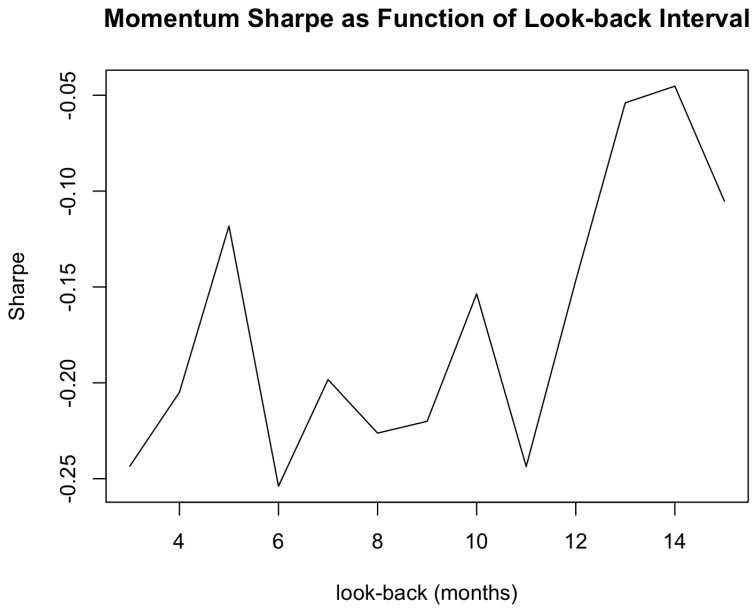
Research indicates that the optimal length of the *look-back interval* for momentum is about 8 to 12 months.

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*).




```
> # 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")
```

```
> # 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))
```

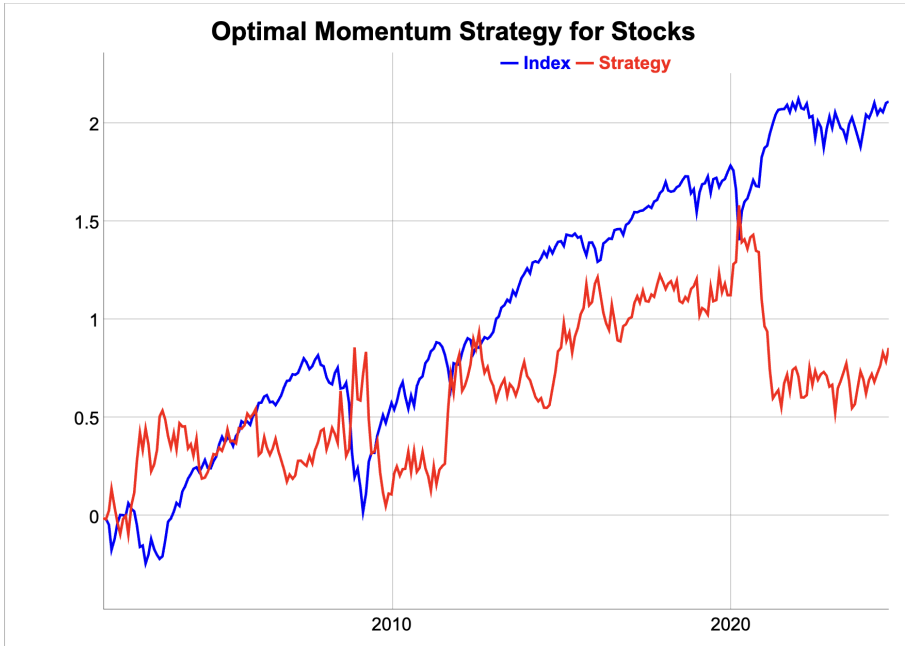
Optimal Stock Momentum Strategy

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[endd[1]:endd[3]], pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("Equal", "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)[endd],
+   main="Optimal Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Weighted Momentum Strategy Functional

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

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

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

```
> btmomweight <- function(retp, objfun, 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
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     # Scale weights so in-sample pnl volatility is same as equal w
+     weightv <- weightv*sd(rowMeans(retis, na.rm=TRUE))/sd(pnlis)
+     # Calculate the out-of-sample momentum returns
+     pnlos <- HighFreq::mult_mat(weightv, retp[(endd[tday]+1):endd
+     pnlos <- rowMeans(pnlos, na.rm=TRUE)
+     drop(pnlos)
+   })  # end lapply
+   rutils::do_call(c, pnls)
+ }  # end btmomweight
```

Optimal Weighted Stock Momentum Strategy

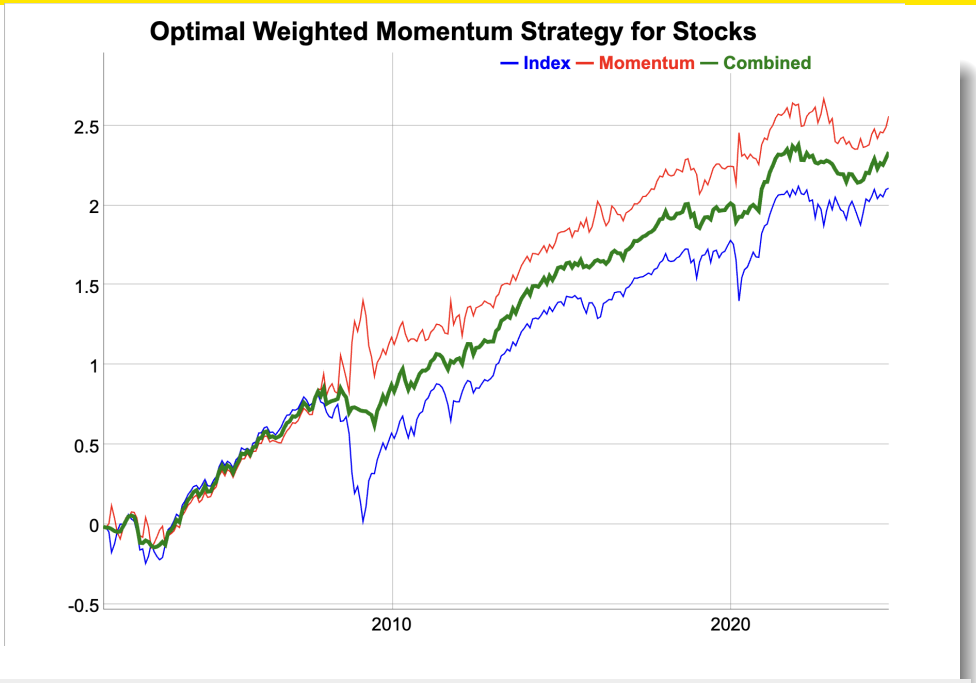
The stock momentum strategy produces a similar absolute return as the index, and also a similar Sharpe ratio.

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, objfu
> # 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, objfu
> 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[endd[1]:endd[3]], pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, 0.5*(retew + pnls))
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnames(wealthv) <- c("Equal", "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)[endd],
+   main="Optimal Weighted Momentum Strategy for Stocks") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
```

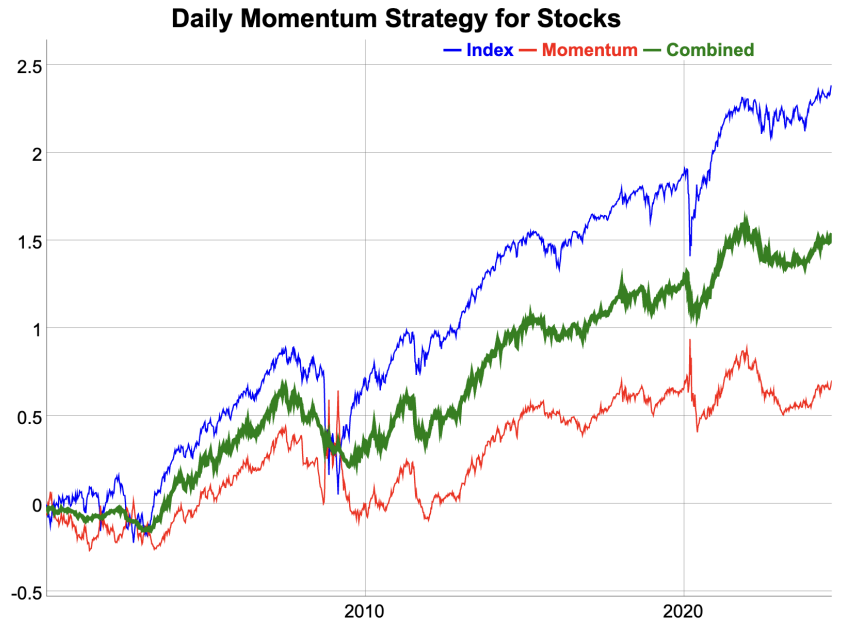
Momentum Strategy With Daily Rebalancing

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 decay factor λ.




```
> # Calculate the trailing average returns and variance using C++ c
> lambdaf <- 0.99
> varm <- HighFreq::run_var(retp, lambda=lambdaf)
> meanm <- varm[, 1:nstocks]
> varm <- varm[, (nstocks+1):(2*nstocks)]
> # Calculate the trailing Kelly ratios
> weightv <- meanm/varm
> weightv <- weightv/sqrt(rowSums(weightv^2, na.rm=TRUE))
> weightv <- rutils::lagit(weightv)
> # Calculate the momentum profits and losses
> pnls <- rowSums(weightv*retp, na.rm=TRUE)
> # Calculate the transaction costs
> bidask <- 0.0
> costv <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)), na.rm=
> pnls <- (pnls - costv)
```

```
> # 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("Equal", "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", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Daily Momentum Strategy Functional

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

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

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

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

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

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

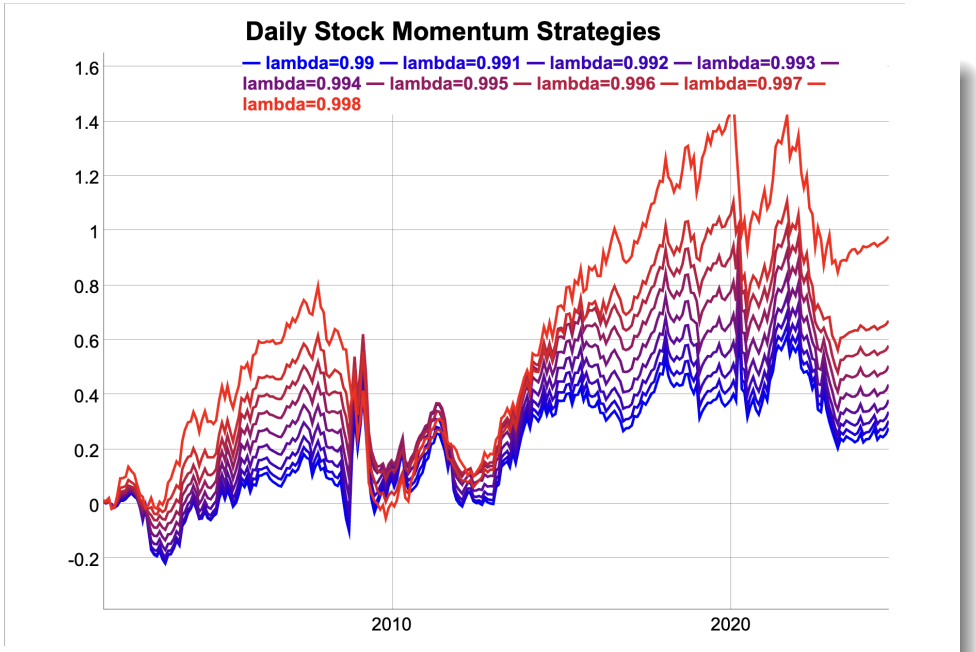
Multiple Daily Stock Momentum Strategies

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

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

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

```
> # Simulate multiple daily stock momentum strategies
> lambdav <- seq(0.99, 0.998, 0.001)
> pnls <- sapply(lambdav, btmomdaily, 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 λ determines the length of the *holding period*. Smaller values of λ produce a faster decay corresponding to a shorter *holding period*, and vice versa.

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

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

```
> # Define backtest functional for daily momentum strategy
> # If trend=(-1) then it backtests a mean reverting strategy
> btmomdailyhold <- function(retp, lambdaf=0.9, trend=1, bidask=0.0
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   # Calculate the trailing Kelly ratio
+   nstocks <- NCOL(retp)
+   varm <- HighFreq::run_var(retp, lambda=lambdaf)
+   meanm <- varm[, 1:nstocks]
+   vars <- varm[, (nstocks+1):(2*nstocks)]
+   weightv <- meanm/vars
+   weightv <- weightv/sqrt(rowSums(weightv^2, na.rm=TRUE))
+   # Average the past weights
+   weightv <- HighFreq::run_mean(weightv, lambda=lambdaf)
+   weightv <- rutils::lagit(weightv)
+   # Calculate the momentum profits and losses
+   pnls <- trend*rowSums(weightv*retp, na.rm=TRUE)
+   # Calculate the transaction costs
+   costv <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)), na.rm=
+   pnls <- (pnls - costv)
+   pnls[1:21] <- 0 # Set the warmup PnLs to zero
+   return(pnls)
+ }  # end btmomdailyhold
```

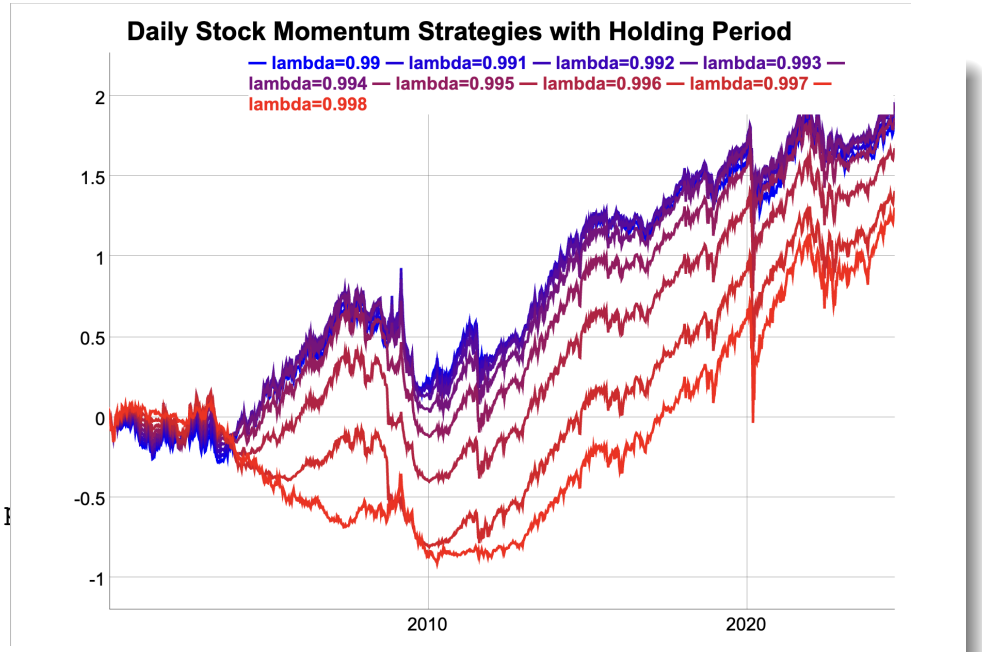
Multiple Daily Momentum Strategies With Holding Period

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

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

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

```
> # Simulate multiple daily stock momentum strategies with holding p
> lambdav <- seq(0.99, 0.998, 0.001)
> pnls <- sapply(lambdav, btmomdailyhold, 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)
```



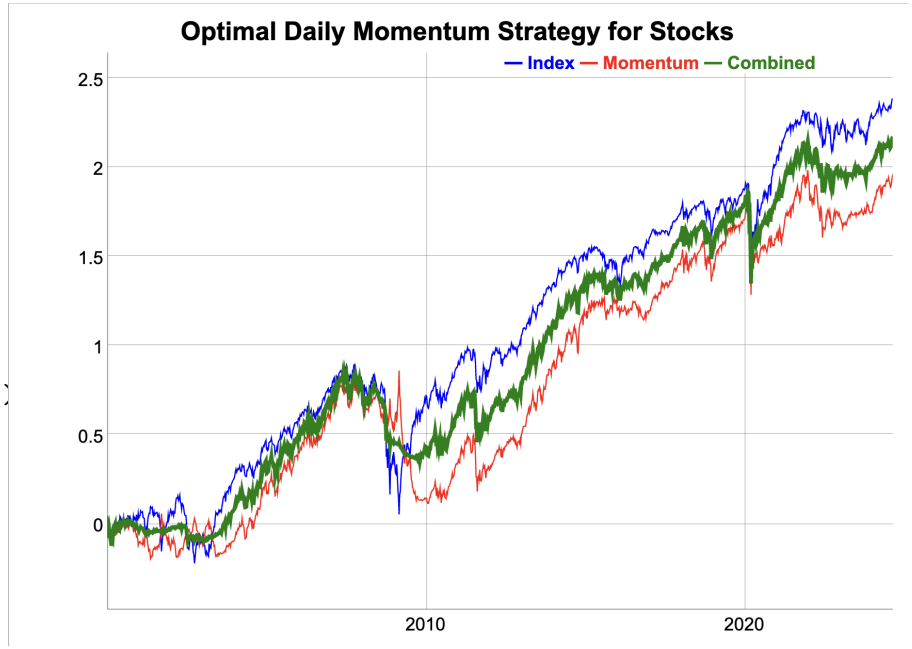

```
> # dygraph of daily stock momentum strategies with holding period
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily Stock Momentum Strategies with Holding Period") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", 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 Hol
> 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("Equal", "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", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting Stock Momentum Strategies

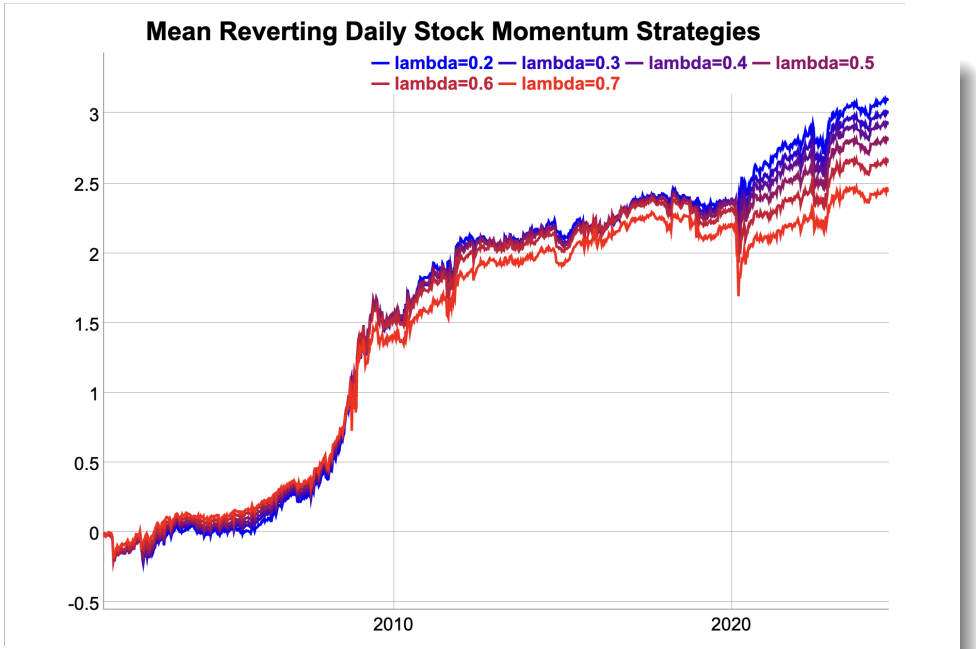
Multiple *mean reverting* stock momentum strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of λ parameters.

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

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

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

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




```
> # Perform sapply loop over lambdav
> 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)
```

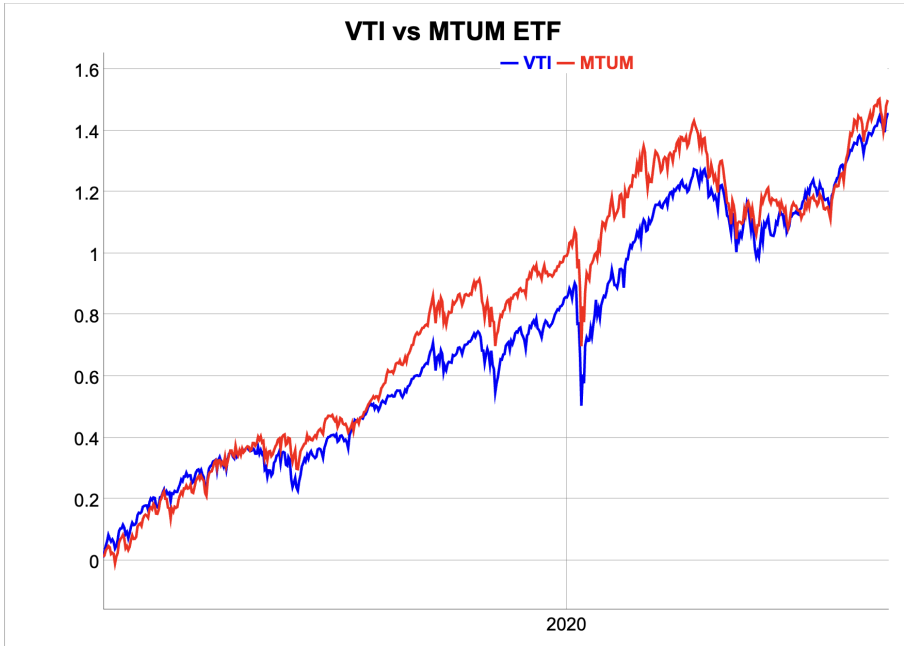
```
> # Plot dygraph of mean reverting daily stock momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Mean Reverting Daily Stock Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always",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

The MTUM Momentum ETF

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

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

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




```
> # Calculate the scaled prices of VTI vs MTUM ETF
> wealthv <- na.omit(rutils::etfenv$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

Principal Component Analysis (*PCA*) can be used to construct portfolios of stocks.

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

PCA performs better if the stock returns are standardized (mean zero and unit variance).

PCA is equivalent to the *eigen decomposition* of either the correlation or the covariance matrix.

The function `eigen()` calculates the *eigenvectors* and *eigenvalues* of numeric matrices.

Performing *PCA* on stock returns with `NA` values is more complicated, so we replace the `NA` values with zeros.

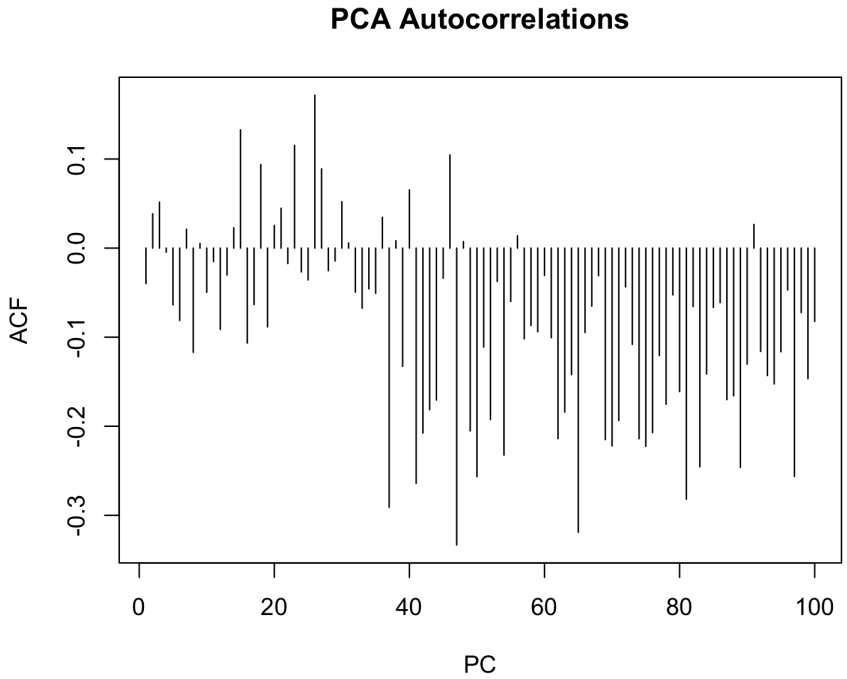
```
> # Set the NA values to zero
> retp[is.na(retp)] <- 0
> # Calculate the standardized returns
> retsc <- lapply(retp, function(x) (x - mean(x))/sd(x))
> retsc <- do.call(cbind, retsc)
> # Calculate the PCA loadings
> covmat <- cov(retsc)
> pcad <- eigen(covmat)
> pcal <- pcad$vectors # The PCA loadings
> 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 PCA
> retpca <- retp %*% pcal
> colnames(retpca) <- paste0("PC", 1:nstocks)
> round((t(retpca) %*% retpca)[1:5, 1:5], 4)
```

Autocorrelations of PCA Returns

The *Principal Component* portfolios (*PC*s) have either trending or mean-reverting characteristics.

The lower order *PCs* exhibit greater trending (positive autocorrelations) than individual stocks.

The higher order *PCs* exhibit greater mean reversion (negative autocorrelations) than the lower order ones.




```
> # Calculate the autocorrelations of the PCA time series
> pacv <- apply(retpca[, 1:100], 2, function(x)
+   sum(pacf(x, lag=10, plot=FALSE)$acf))
> plot(pacv, type="h", main="PCA Autocorrelations",
+      xlab="PC", ylab="PACF")
```

Momentum Strategy for PCA Portfolios

The momentum strategy performs better for *PCA* portfolios than for individual stocks for two reasons:

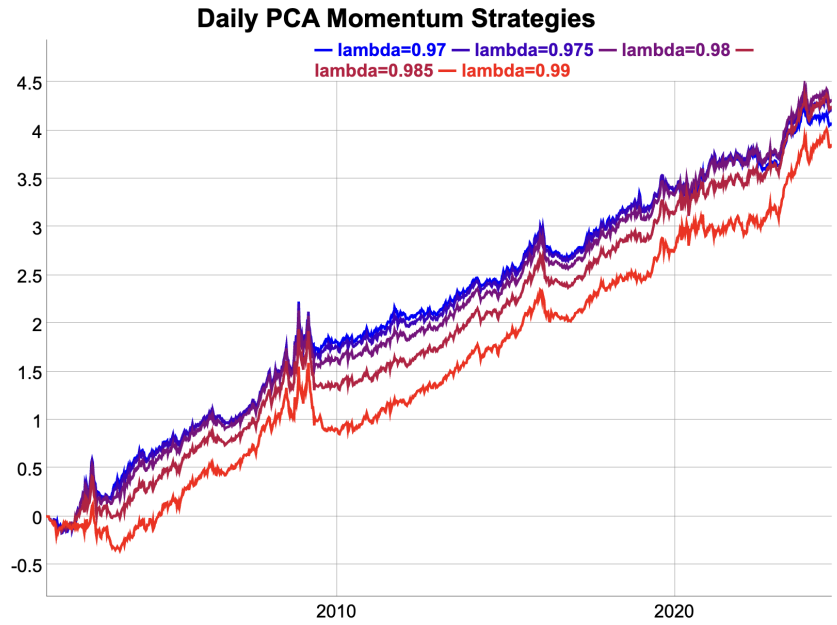
- The *PCA* returns are uncorrelated to each other,
- The lower order *PCA* returns have more positive autocorrelations than individual stocks.
- The higher order *PCA* returns have more negative autocorrelations than individual stocks.

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.




```
> # Simulate daily PCA momentum strategies for multiple lambdaf pa
> dimax <- 30
> 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="lambdaf", 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)
```

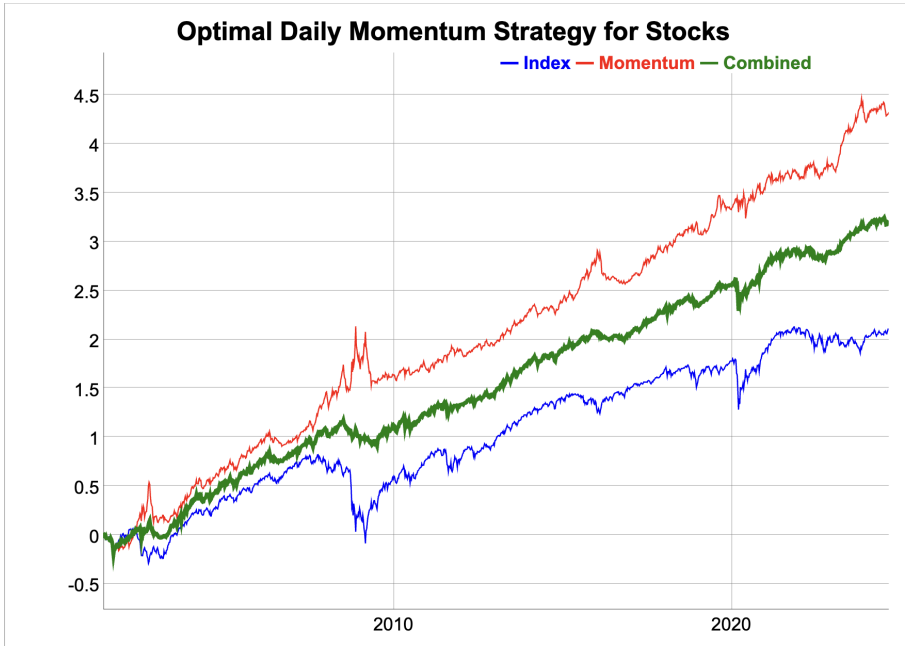
Optimal PCA Momentum Strategy

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

But this is thanks to using the in-sample *PCs*.

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

```
> # Calculate best pnls of PCA momentum strategy
> 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("Equal", "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", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting PCA Momentum Strategy

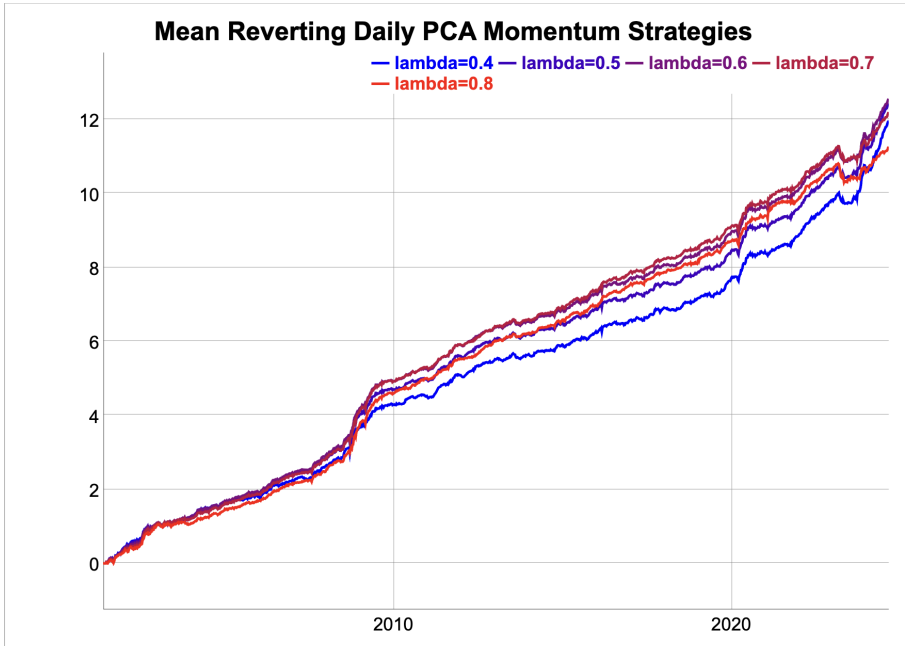
The *mean reverting* momentum strategy buys the worst performing stocks and sells the best.

If the argument `trend` is set to `-1`, then the function `btmomdailyhold()` simulates a mean-reverting strategy.

The *mean reverting* strategy performs well for the higher order *PCs*, but not the highest order, because those have very low volatilities.

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

The *mean reverting* strategy trades very frequently, so it's very sensitive to the transaction costs. If the transaction costs are too high then the strategy will lose money.




```
> # Simulate daily PCA momentum strategies for multiple lambdaf pa
> lambdav <- seq(0.4, 0.8, 0.1)
> pnls <- mclapply(lambdav, btmomdailyhold, retp=retpca[, (dimax+1
+    trend=(-1), bidask=0.0001, 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)
```

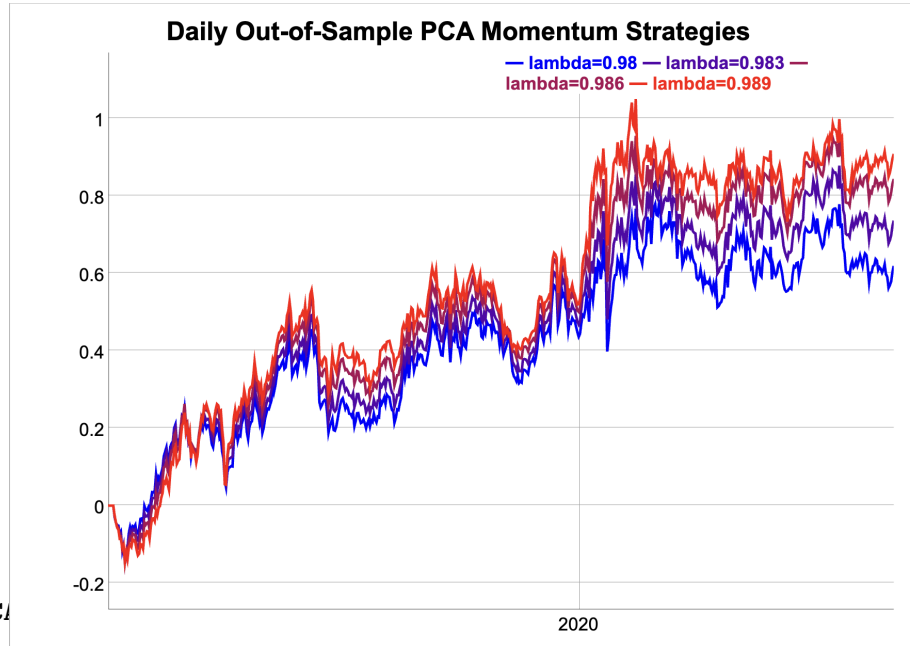
```
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=lambdav, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Decay Factor",
+   xlab="lambdaf", ylab="Sharpe")
> # Plot dygraph of daily PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Mean Reverting Daily PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=400)
```

PCA Momentum Strategy Out-of-Sample

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

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

```
> # Define in-sample and out-of-sample intervals
> cutoff <- nrows %/% 2
> datev[cutoff]
> insample <- 1:cutoff
> outsample <- (cutoff + 1):nrows
> # Calculate the PCA loadings in-sample
> covmat <- cov(retsc[insample])
> pcad <- eigen(covmat)
> pcal <- pcad$vectors # The PCA loadings
> rownames(pcal) <- colnames(retp)
> # Calculate the PCA time series from the stock returns and the PCA
> retpca <- retp %*% pcal
> colnames(retpca) <- paste0("PC", 1:nstocks)
> # Calculate the out-of-sample PCA time series
> retpca <- xts::xts(retpca[outsample, ], order.by=datev[outsample])
> # Simulate daily PCA momentum strategies for multiple lambdaf pa
> lambdav <- seq(0.98, 0.99, 0.003)
> 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[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")
```



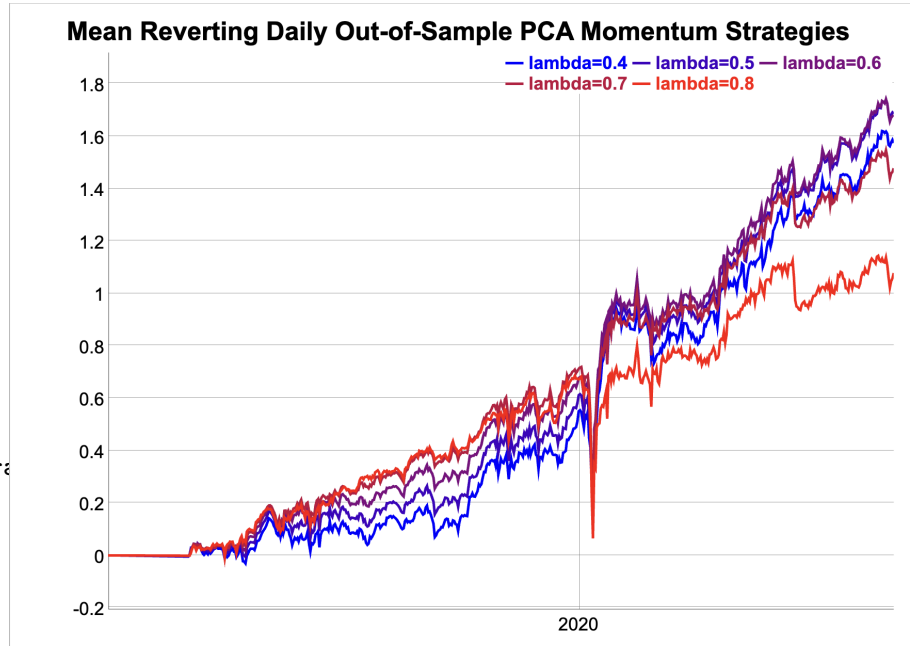

```
> # Calculate a vector of weekly end points
> endd <- rutils::calc_endpoints(retpca, interval="weeks")
> # Plot dygraph of daily out-of-sample PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily Out-of-Sample PCA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

Mean Reverting PCA Momentum Strategy Out-of-Sample

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

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

The *mean reverting* strategy trades very frequently, so it's very sensitive to the transaction costs. If the transaction costs are too high then the strategy will lose money.




```
> # Simulate daily PCA momentum strategies for multiple lambdaf para
> lambdav <- seq(0.4, 0.8, 0.1)
> pnls <- mclapply(lambdav, btmomdailyhold,
+    retp=retpca[, (dimax+1):(NCOL(retpca)-dimax)],
+    trend=(-1), bidask=0.0001, 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[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")
```

```
> # 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*.

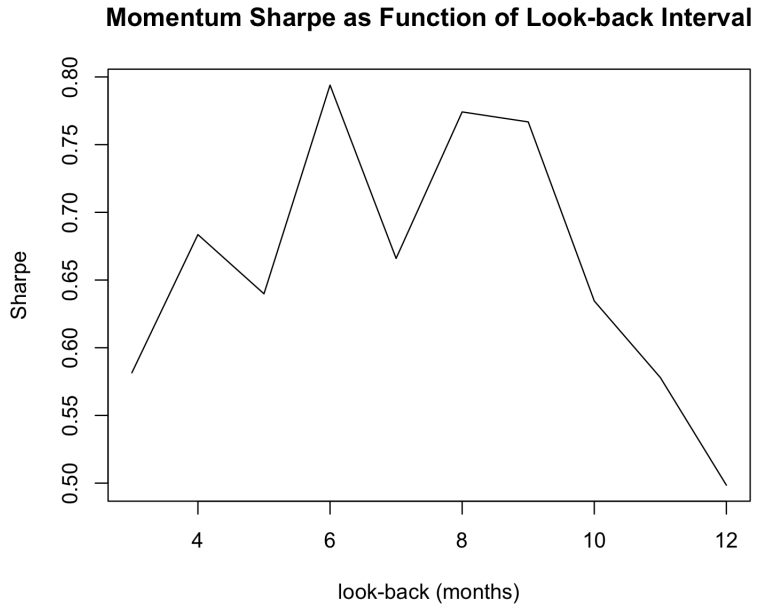
If the *look-back interval* is too long, then the data has large *bias* because the distant past may have little relevance to today.

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.




```
> # 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

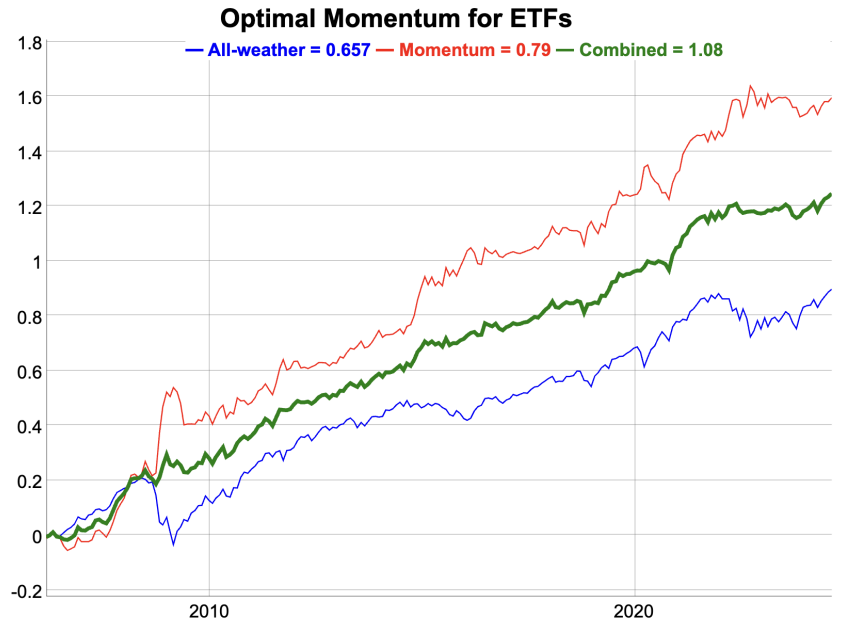
The momentum strategy for ETFs produces a higher absolute return and also a higher Sharpe ratio than the static *All-Weather* portfolio.

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)
> lookbv[whichmax]
> pnls <- pnll[[whichmax]]
> retew <- rowMeans(retp)
> pnls <- c(retew[endd[1]:endd[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, ],
> # Plot dygraph of all-weather benchmark and momentum strategy
> dygraphs::dygraph(cumsum(wealthv)[endd], main=captiont) %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name=colnames(wealthv)[3], strokeWidth=3) %>%
+   dyLegend(show="always", width=500)
```

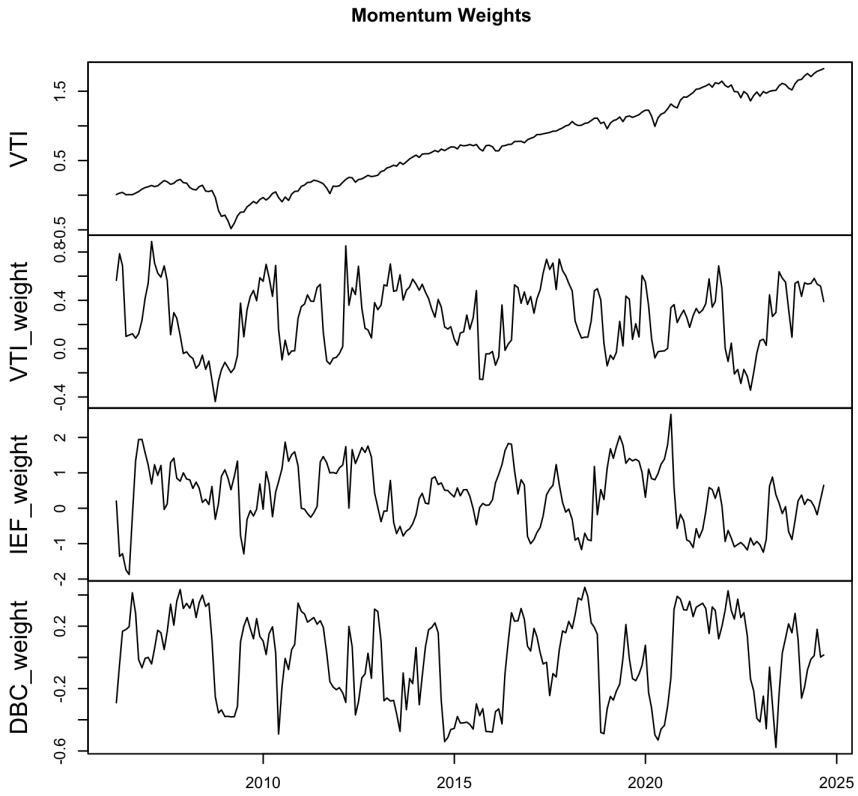
Time Series of Momentum Portfolio Weights

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*.

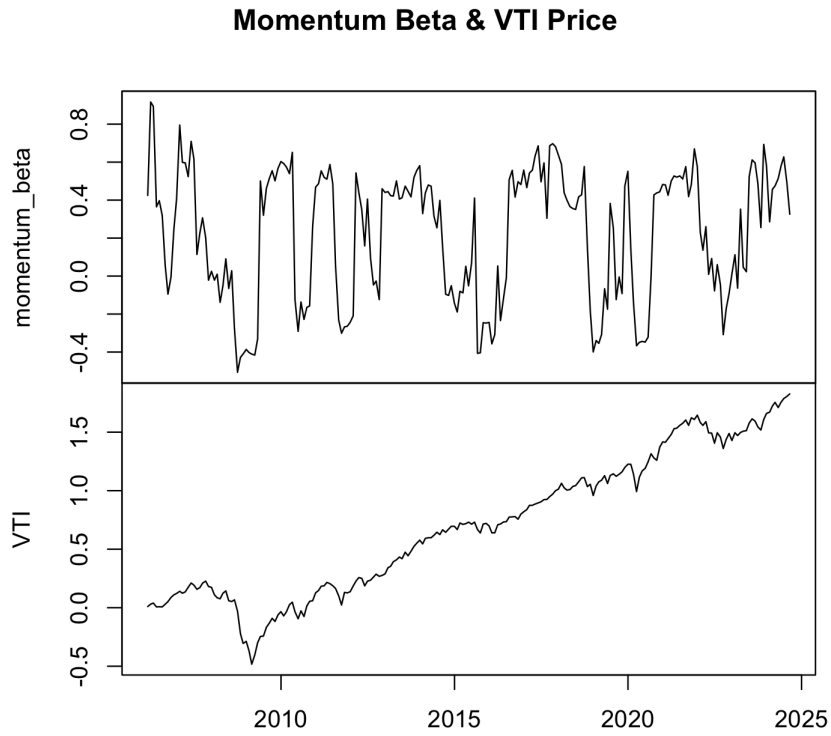
```
> # Calculate the momentum weights
> lookb <- lookbv[whichmax]
> weightv <- lapply(2:npts, function(tday) {
+   # Select the look-back returns
+   startp <- endd[max(1, tday-lookb)]
+   retis <- retp[startp:endd[tday], ]
+   # Calculate weights proportional to performance
+   perfstat <- sapply(retis,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



Momentum Strategy Market Beta

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

```
> # Calculate ETF betas
> betasetf <- sapply(retp, function(x) cov(retp$VTI, x)/var(retp$VTI
> # Momentum beta is equal weights times ETF betas
> 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="")
```



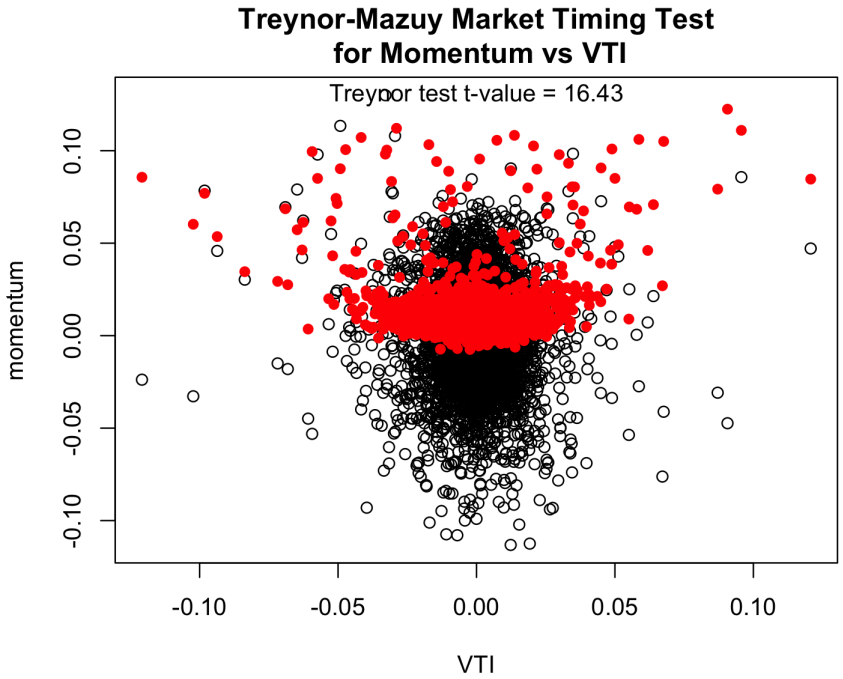
Momentum Strategy Market Timing Skill

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$VTI
> desm <- cbind(pnls, retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> desm <- HighFreq::roll_sumep(desm, lookb=22)
> colnames(desm) <- c("pnls", "VTI", "Merton", "Treynor")
> desm <- as.data.frame(desm)
> # Merton-Henriksson test
> regmod <- lm(pnls ~ VTI + Merton, data=desm); summary(regmod)
> # Treynor-Mazuy test
> regmod <- lm(pnls ~ VTI + Treynor, data=desm); summary(regmod)
> # Plot residual scatterplot
> resids <- regmod$residuals
> plot.default(x=retvti, y=resids, xlab="VTI", ylab="momentum")
> title(main="Treynor-Mazuy Market Timing Test\n for Momentum vs VTI", 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=fitv, pch=16, col="red")
> text(x=0.0, y=max(resids), paste("Treynor test t-value =", tvalue))
```



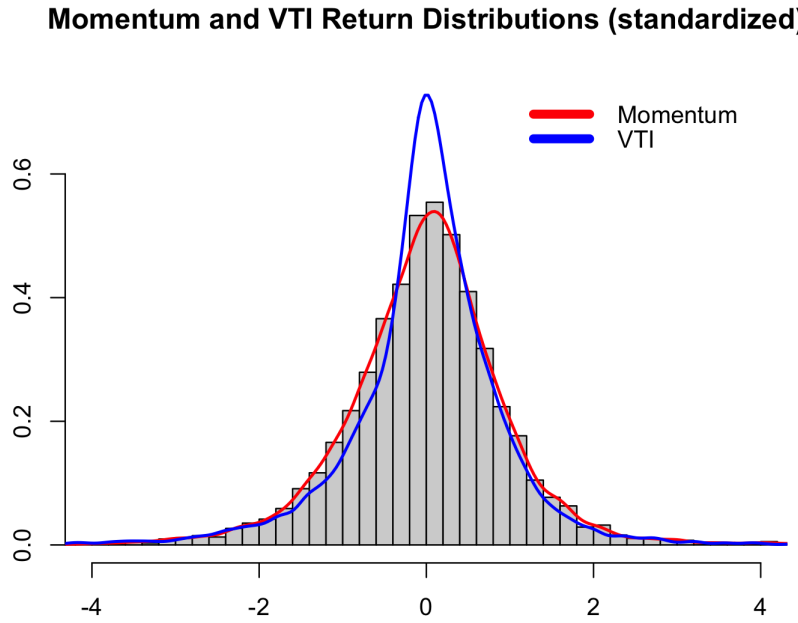
Skewness of Momentum Strategy Returns

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

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

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

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




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

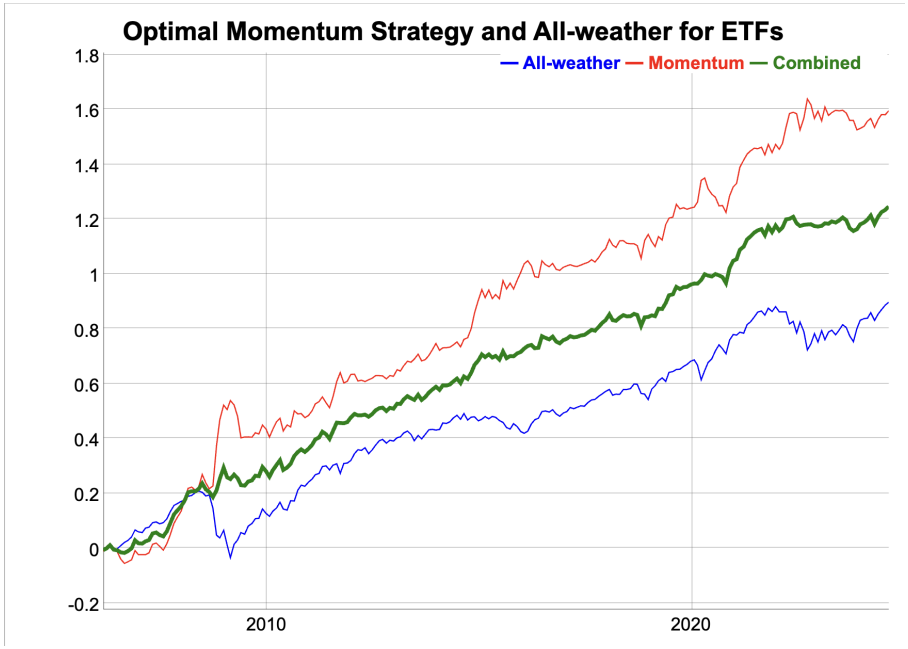
Combining Momentum with the *All-Weather* Portfolio

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

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

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

```
> # Combine momentum strategy with all-weather
> wealthv <- cbind(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", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

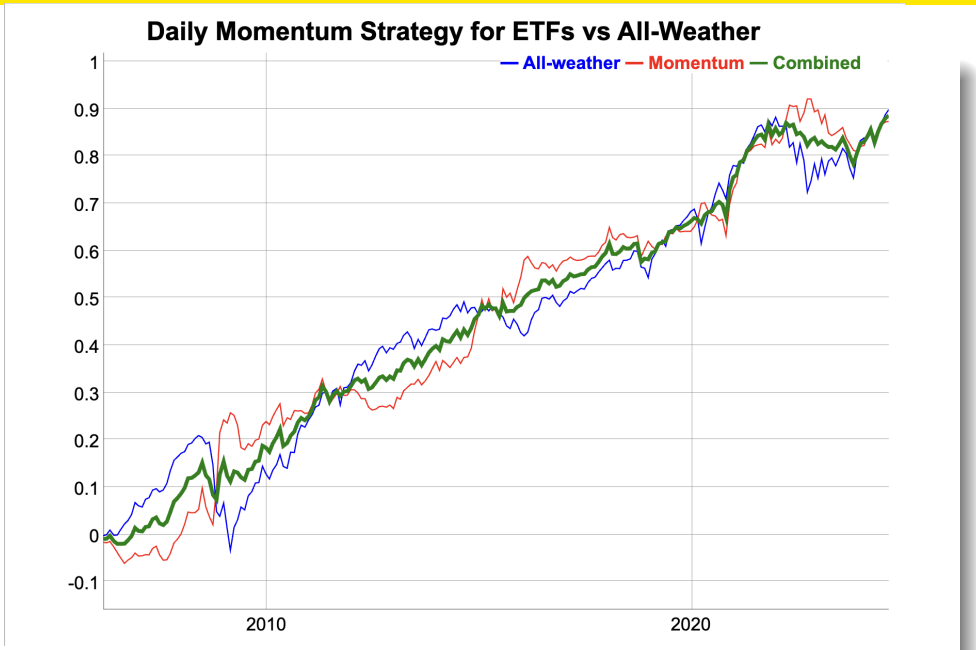
Momentum Strategy for ETFs With Daily Rebalancing

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(retp, lookb=lookb)
> # Calculate the trailing Kelly ratio
> meanv <- HighFreq::roll_mean(retp, 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*retp)
> # Calculate the transaction costs
> bidask <- 0.0
> costv <- 0.5*bidask*rowSums(abs(rutils::diffit(weightv)))
> pnls <- (pnls - costv)
```



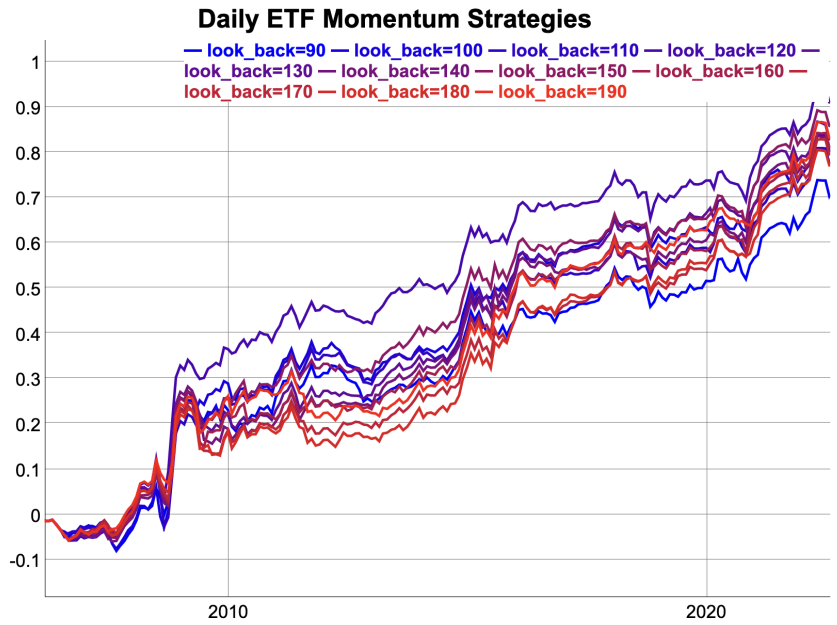

```
> # Scale the momentum volatility to all-weather
> pnls <- sd(retaw)*pnls/sd(pnls)
> # Calculate the wealth of momentum returns
> wealthv <- cbind(retaw, pnls, 0.5*(pnls + retaw))
> colnames(wealthv) <- c("All-weather", "Momentum", "Combined")
> wealthv <- xts::xts(wealthv, datev)
> # Calculate the 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", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

draft: Multiple Daily ETF Momentum Strategies

Multiple daily ETF *momentum* strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of *look-back* parameters.

The best performing daily ETF *momentum* strategies are with *look-back* parameters between 100 and 120 days.

The *momentum* strategies do not perform well, especially the ones with a long *look-back* parameter.




```
> # Simulate a daily ETF momentum strategy
> pnls <- btmomdaily(retp=retp, lookb=152, bidask=bidask)
> # Perform sapply loop over lookbv
> lookbv <- seq(90, 190, by=10)
> pnls <- sapply(lookbv, btmomdaily,
+   retp=retp, bidask=bidask)
> # Scale the momentum volatility to all-weather
> pnls <- apply(pnls, MARGIN=2,
+   function(pnl) sd(retaw)*pnl/sd(pnl))
> colnames(pnls) <- paste0("lookb=", lookbv)
> pnls <- xts::xts(pnls, datev)
> tail(pnls)
```

```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily ETF Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Plot daily ETF 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="Cumulative Returns of Daily ETF Momentum
> 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")
```

draft: Momentum Strategy for an *ETF* Portfolio

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

- Specify a portfolio of *ETFs*, stocks, or other assets, and a time series of their returns,
- Specify *end points* for the portfolio rebalancing frequency,
- Specify *look-back* intervals for portfolio formation, and *look-forward* intervals for portfolio holding,
- Specify a performance function to calculate the past performance of the assets,
- Calculate the past performance over the *look-back* formation intervals,
- Calculate the portfolio weights from the past (in-sample) performance,
- Calculate the future returns over the *look-forward* holding intervals,
- Calculate the out-of-sample momentum strategy returns by applying the portfolio weights to the future returns,
- Calculate the transaction costs and subtract them from the strategy returns.

```
> # Extract ETF returns
> symbolv <- c("VTI", "IEF", "DBC")
> retp <- rutils::etfenv$returns[, symbolv]
> retp <- na.omit(retp)
> # Or, select rows with IEF data
> # retp <- retp[zoo::index(rutils::etfenv$IEF)]
> # Copy over NA values
> # retp[1, is.na(retp[1, ])] <- 0
> # retp <- zoo::na.locf(retp, na.rm=FALSE)
```

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* δ 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 value
> retp <- rutils::etfenv$returns["1999/", symbolv]
> sum(is.na(retp)) # Number of NA values
> 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)
> round(invmat %*% covmat, digits=5)
> # 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: $\mathbb{C}^{-1}\mathbb{C} = \mathbb{1}$

```
> # 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
> notzero <- (eigenval > (precv*eigenval[1]))
> inveigen <- eigenvec[, notzero] %*%
+   (t(eigenvec[, notzero]) / eigenval[notzero])
> # Inverse property of invreg doesn't hold
> all.equal(covmat, inveigen %*% covmat)
> # Generalized inverse property of invreg holds
> all.equal(covmat, covmat %*% inveigen %*% covmat)
> # Calculate generalized inverse using MASS::ginv()
> invreg <- MASS::ginv(covmat)
> # Verify that inveigen is the same as invreg
> all.equal(inveigen, invreg)
```

Portfolio Optimization Strategy

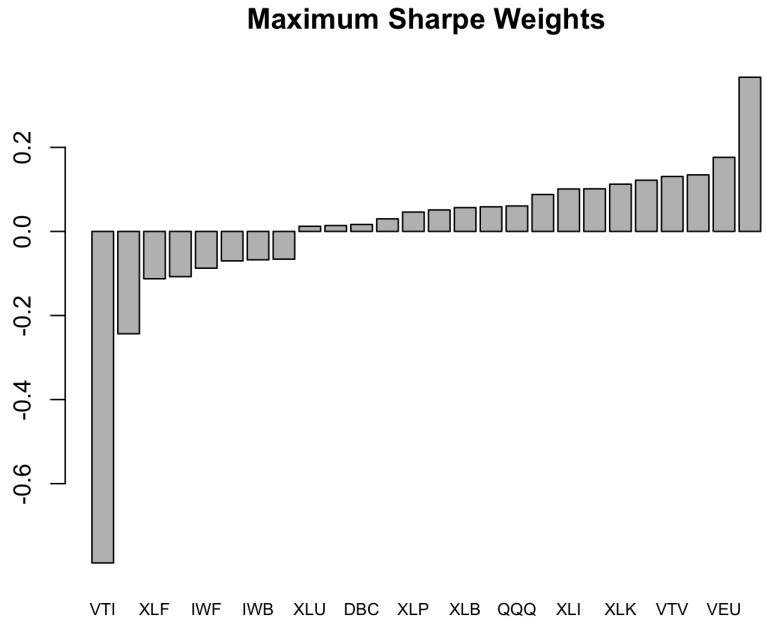
The optimal portfolio weights $\mathbf{w}$ are equal to the past in-sample excess returns $\mu = \mathbf{r} - r_f$ (in excess of the risk-free rate r_f) multiplied by the inverse of the covariance matrix $\mathbb{C}$:

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

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.




```
> # Maximum Sharpe weights in-sample interval
> retis <- retp["/2014"]
> raterf <- 0.03/252
> retx <- (retis - raterf)
> invreg <- MASS::ginv(cov(retis, use="pairwise.complete.obs"))
> weightv <- drop(invreg %*% colMeans(retx, na.rm=TRUE))
> weightv <- weightv/sqrt(sum(weightv^2))
> names(weightv) <- colnames(retp)
```

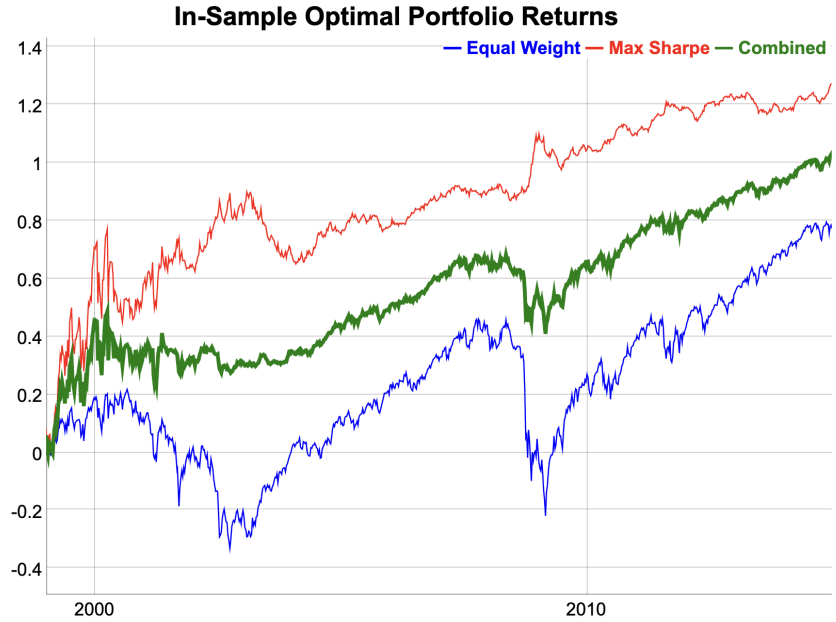
```
> # Plot portfolio weights
> barplot(sort(weightv), main="Maximum Sharpe Weights", cex.names=0
```

Portfolio Optimization Strategy In-Sample

The in-sample performance of the *maximum Sharpe* 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(retp, na.rm=TRUE), datev)
> # Calculate the in-sample weighted returns using Rcpp
> pnlis <- HighFreq::mult_mat(weightv, retis)
> # Or using the transpose
> # pnlis <- unname(t(t(retis)*weightv))
> pnlis <- rowMeans(pnlis, na.rm=TRUE)
> pnlis <- pnlis*sd(retew["/2014"])/sd(pnlis)
```



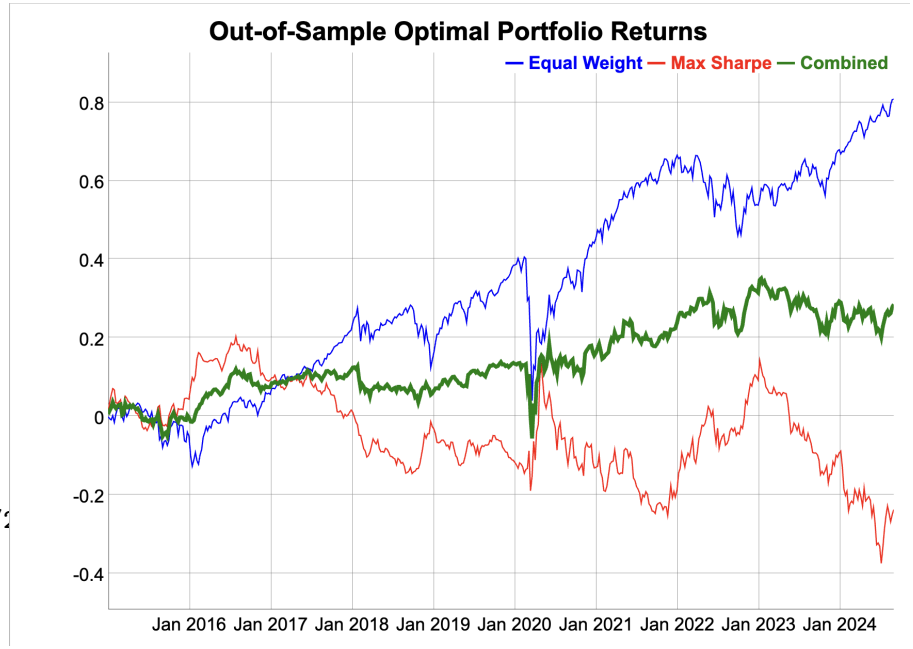

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retew["/2014"], pnlis, (pnlis + retew["/2014"])/2
> colnames(wealthv) <- c("Equal Weight", "Max Sharpe", "Combined")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Dygraph cumulative 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", 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.

Combining the *maximum Sharpe* portfolio with the equal weight portfolio produces and even better performing portfolio.

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



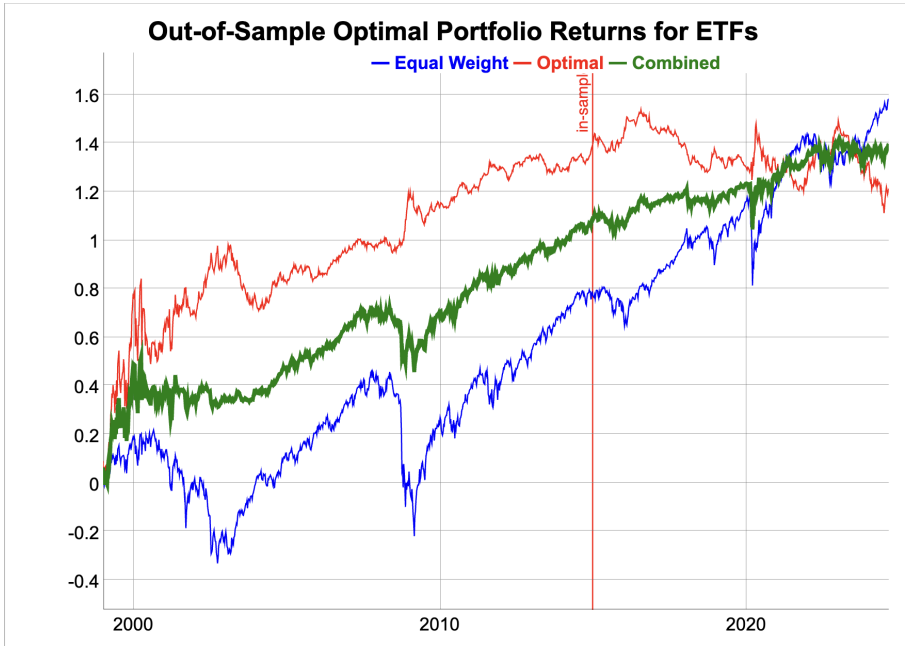

```
> # Dygraph cumulative wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Out-of-Sample Optimal Portfolio Returns") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   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(retx, na.rm=TRUE)
> names(weightv) <- colnames(retp)
> # Calculate cumulative wealth
> pnls <- HighFreq::mult_mat(weightv, retp)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> pnls <- pnls*sd(retew)/sd(pnls)
> 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", strokeWidth=3) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeP
+   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 $dimax = d$:

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

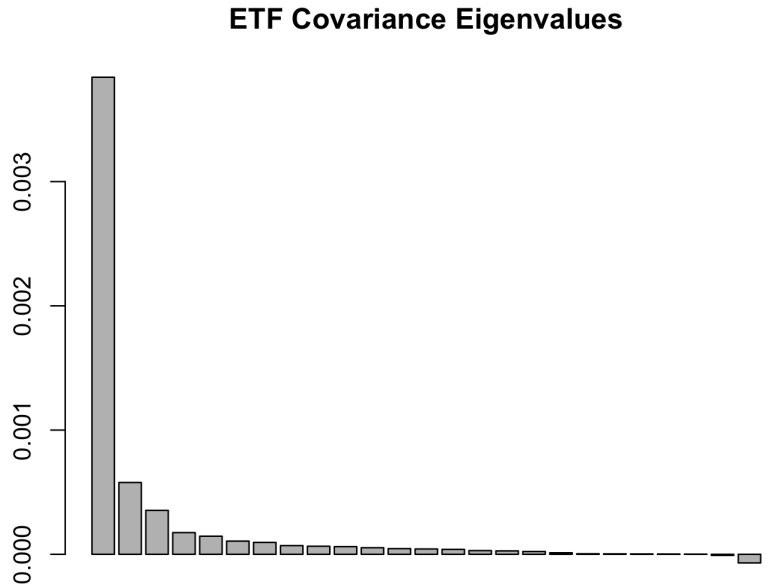
The parameter *dimax* specifies the number of eigenvalues used for calculating the *reduced inverse* of the covariance matrix of returns.

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*).



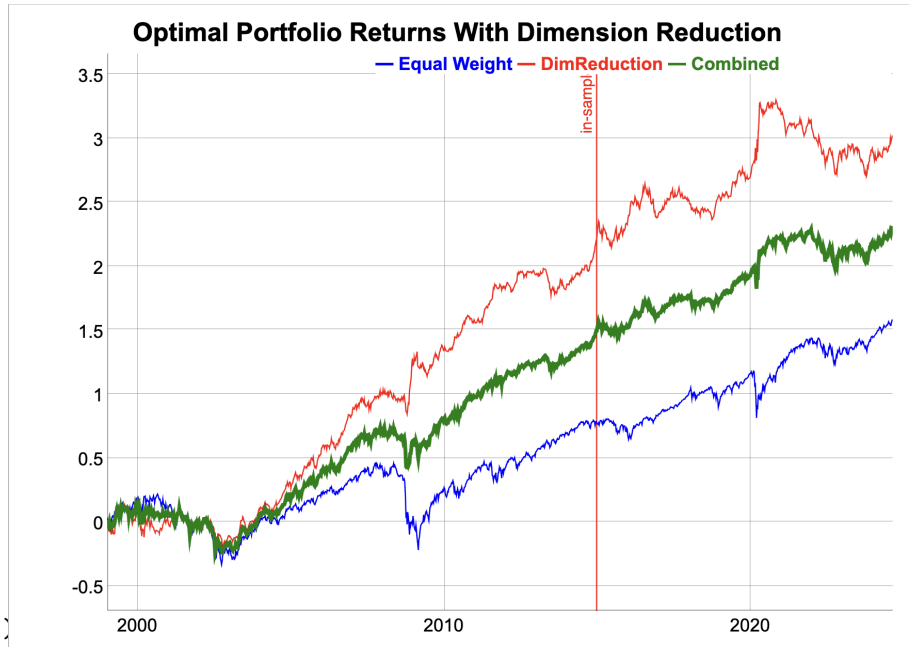

```
> # Calculate in-sample covariance matrix
> covmat <- cov(retis, use="pairwise.complete.obs")
> eigend <- eigen(covmat)
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> # Plot the eigenvalues
> barplot(eigenval, main="ETF Covariance Eigenvalues", cex.names=0.7
> # Calculate reduced inverse 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 cumulative wealth
> pnls <- HighFreq::mult_mat(weightv, retp)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> 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", strokeWidth=3) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeP
+   dyLegend(width=400)
```

Portfolio Optimization With Return Shrinkage

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

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

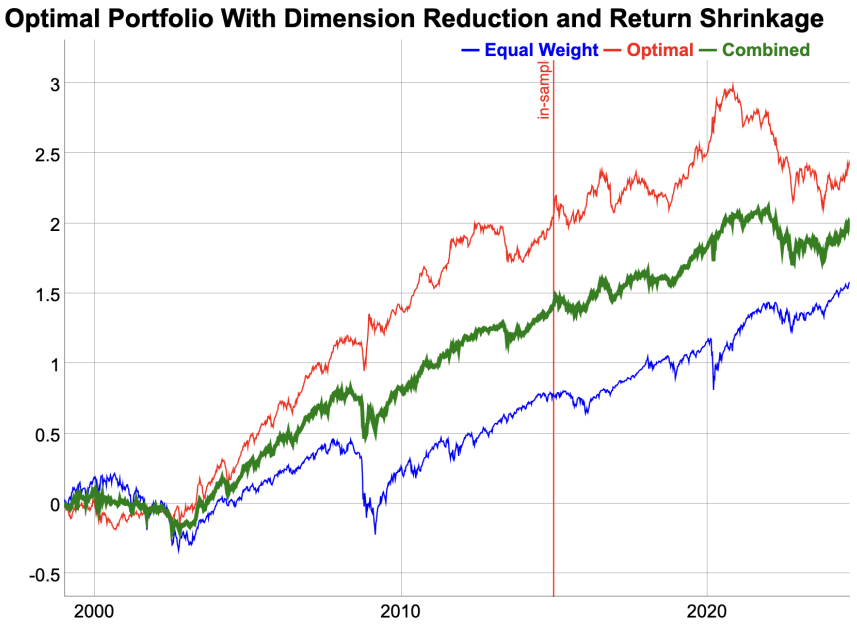
The parameter α is the *shrinkage* intensity, and it determines the strength of the *shrinkage* of individual returns to their mean.

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

If $\alpha = 1$ then the weights are equal to the minimum variance portfolio weights. So an intermediate value of α gives weights that are an average of the maximum Sharpe weights and the minimum variance weights.

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

```
> # Shrink the in-sample returns to their mean
> alphac <- 0.7
> retxm <- rowMeans(retx, na.rm=TRUE)
> retxis <- (1-alphac)*retx + alphac*retxm
> # Calculate portfolio weights
> weightv <- invred %*% colMeans(retxis, na.rm=TRUE)
> # Calculate cumulative wealth
> pnls <- HighFreq::mult_mat(weightv, retp)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> pnls <- pnls*sd(retew)/sd(pnls)
> 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
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Portfolio With Dimension Reduction and Return Shri
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeP
+   dyLegend(width=300)
```

Rolling Portfolio Optimization Strategy

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

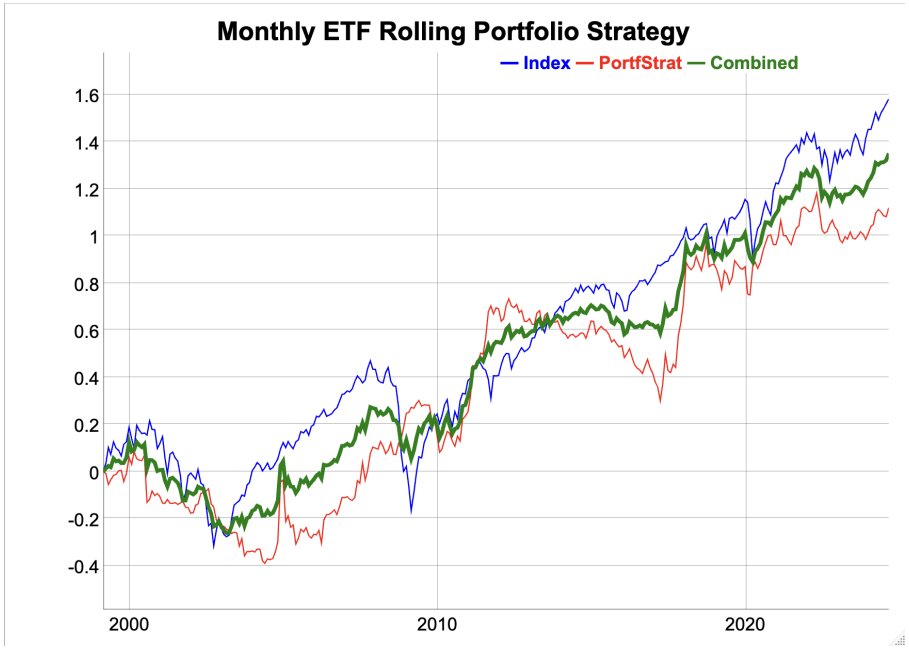
- Calculate the *end points* for portfolio rebalancing,
- Define an objective function for optimizing the portfolio weights,
- Calculate the optimal portfolio weights from the past (in-sample) performance,
- Calculate the out-of-sample returns by applying the portfolio weights to the future returns.

```
> # Define monthly end points
> endd <- rutils::calc_endpoints(retp, interval="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
+     # Calculate the in-sample portfolio returns
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     # 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("Equal", "PortfStrat", "Combined")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```



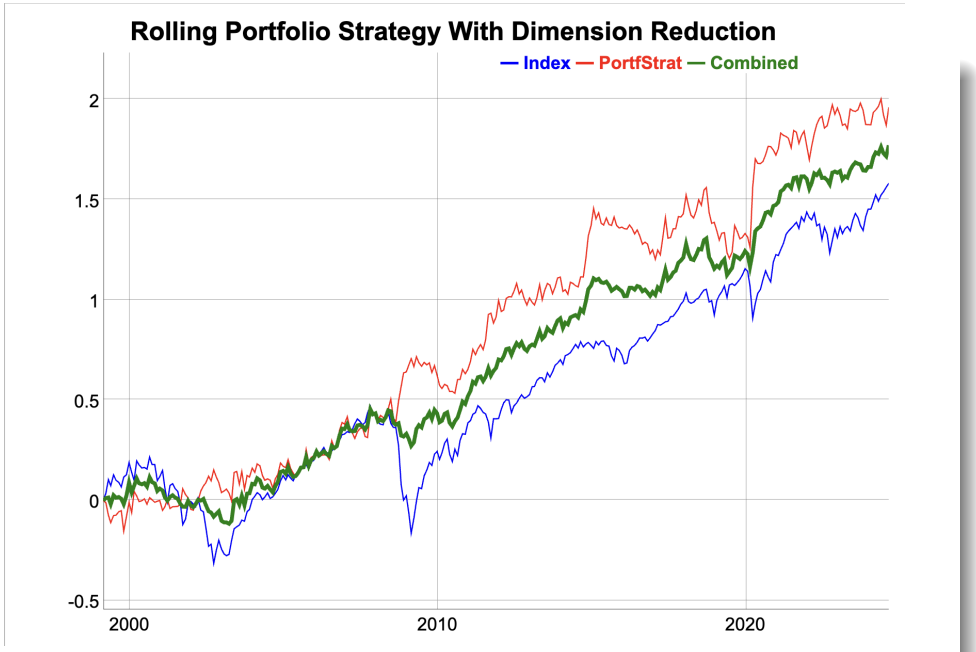

```
> # Dygraph cumulative wealth
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Monthly ETF Rolling
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

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
+     # Calculate the in-sample portfolio returns
+     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)
```




```
> # Calculate the Sharpe and Sortino ratios
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Equal", "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 Strategy With Dimension Reduction") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="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.

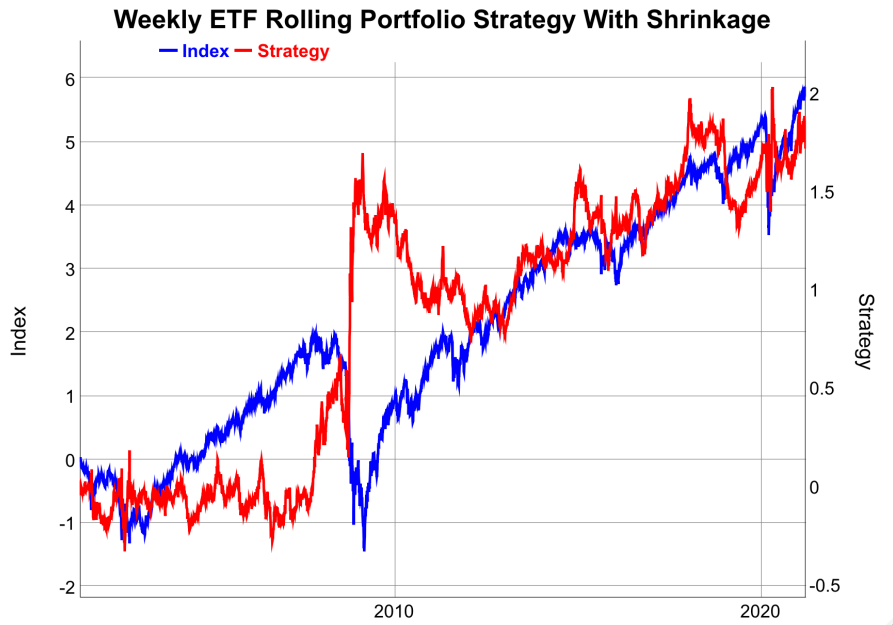
```
> alphac <- 0.7 # Return shrinkage intensity
> # Perform loop over end points
> pnls <- lapply(1:(npts-1), function(tday) {
+     # Shrink the in-sample returns to their mean
+     retis <- retx[startp[tday]:endd[tday], ]
+     rowm <- rowMeans(retis, na.rm=TRUE)
+     rowm[is.na(rowm)] <- 0
+     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)
```


```
> # Calculate the Sharpe and Sortino ratios
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Equal", "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 Strategy With Return Shrinkage") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

draft: Weekly ETF Rolling Portfolio Strategy With Shrinkage

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

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




```
> # Define weekly dates
> weeks <- seq.Date(from=as.Date("2001-01-01"), to=as.Date("2021-04-
> # Perform loop over monthly dates
> lookb <- 21
> dimax <- 9
> pnls <- lapply((lookb+1):(NROW(weeks)-1), function(tday) {
+   # Define in-sample and out-of-sample returns
+   insample <- (datev > weeks[tday-lookb]) & (datev < weeks[tday]]
+   outsample <- (datev > weeks[tday]) & (datev < weeks[tday+1])
+   retis <- retp[insample]
+   retos <- retp[outsample]
+   # Calculate reduced inverse of covariance matrix
+   # invreg <- MASS::ginv(cov(retis, use="pairwise.complete.obs")]
+   invred <- HighFreq::calc_invsd(cov(retis, use="pairwise.complet
+   weightv <- invred %*% colMeans(retis - raterf, na.rm=TRUE)
+   # weightv <- drop(weightv/sqrt(sum(weightv^2)))
+   # Calculate portfolio pnls out-of-sample
+   xts::xts(retos %*% weightv, zoo::index(retos))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```

```
> # Plot dygraph of weekly rolling ETF portfolio strategy
> vti <- rutils::diffit(cumsum(retew)[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("Equal", "Strategy")
> colv <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],                    :t
+   main="Weekly ETF Rolling Portfolio Strategy With dimension reduc
+   dyAxis("y", label=colv[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colv[2], independentTicks=TRUE) %>%
+   dySeries(name=colv[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colv[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

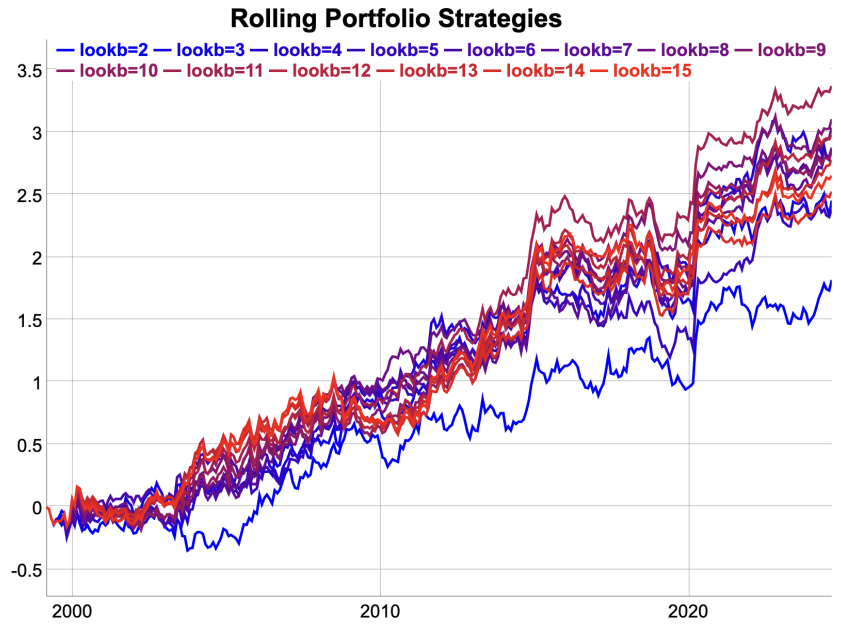
Function for Rolling Portfolio Optimization Strategy

```
> # 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)
+     xts::xts(pnlos, zoo::index(retos))
```

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)
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnls <- mclapply(lookbv, roll_portf, retp=retp, retx=retx,
+   endd=endd, dimax=dimax, mc.cores=ncores)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lookb=", lookbv)
> pnlsums <- sapply(pnls, sum)
> lookb <- lookbv[which.max(pnlsums)]
```

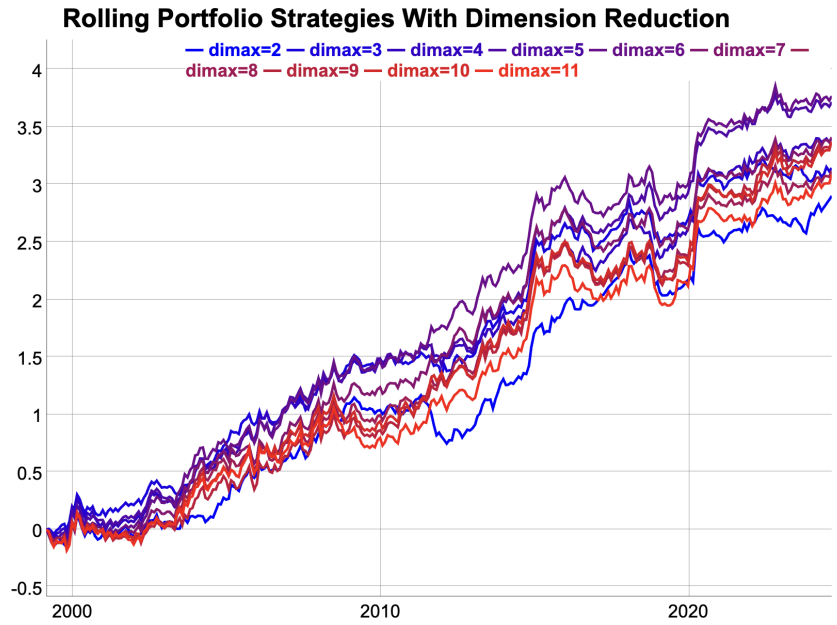



```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(pnls, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of monthly ETF portfolio strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd], main="Rolling Portfolio Stra
+   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 <- mclapply(dimv, roll_portf, retp=retp, retx=retx,
+   endd=endd, lookb=lookb, mc.zcores=ncores)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("dimax=", dimv)
> pnls <- sapply(pnls, sum)
> dimax <- dimv[which.max(pnls)]
```



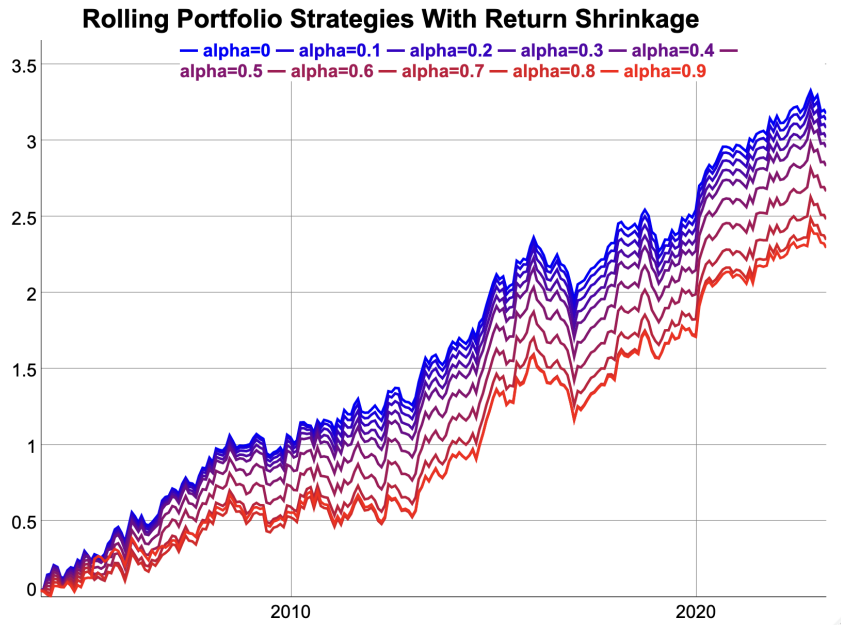

```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(pnls, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of monthly ETF portfolio strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies With Dimension Reduction") %>
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=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(retp)),
+   col=plot_theme$col$line.col, bty="n")
```

draft: Rolling Portfolio Optimization With Different Return Shrinkage

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

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

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

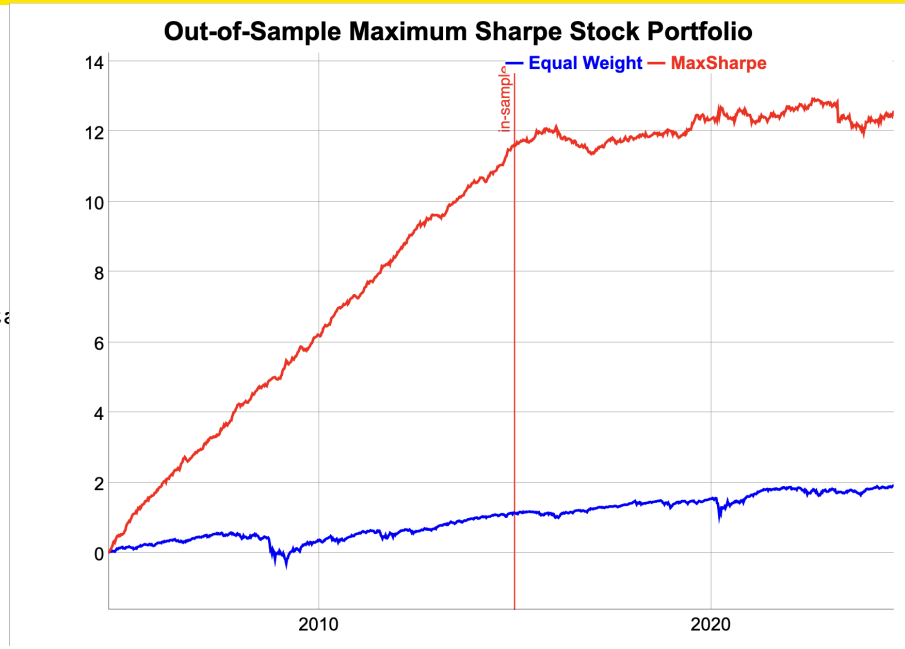



```
> # Plot dygraph of monthly ETF portfolio strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies With Return Shrinkage") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)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

Maximum Sharpe Stock Portfolio Out-of-Sample

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

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




```
> # Load stock returns
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RData
> datev <- zoo::index(na.omit(retstock$GOOGL))
> retp <- retstock[datev] # Subset the returns to GOOGL
> # Remove the stocks with any NA values
> nonas <- sapply(retp, function(x) sum(is.na(x)))
> nonas <- !(nonas > 0)
> retp <- retp[, nonas]
> nstocks <- NCOL(retp)
> datev <- zoo::index(retp)
> colv <- colnames(retp)
> retis <- retp["/2014"] # In-sample returns
> raterf <- 0.03/252
> retx <- (retis - raterf) # Excess returns
> # Maximum Sharpe weights in-sample interval
> covmat <- cov(retis, use="pairwise.complete.obs")
> invreg <- MASS::ginv(covmat)
> colmeanv <- colMeans(retx, na.rm=TRUE)
> weightv <- drop(invreg %*% colmeanv)
> names(weightv) <- colv
> head(sort(weightv))
> tail(sort(weightv))
> # Calculate the portfolio returns
> pnls <- HighFreq::mult_mat(weightv, retp)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> pnls <- xts::xts(pnls, datev)
> retew <- xts::xts(rowMeans(retp, na.rm=TRUE), datev)
> pnls <- pnls*sd(retew["/2014"])/sd(pnls["/2014"])
> wealthv <- cbind(retew, pnls)
> colnames(wealthv) <- c("Equal Weight", "MaxSharpe")
```

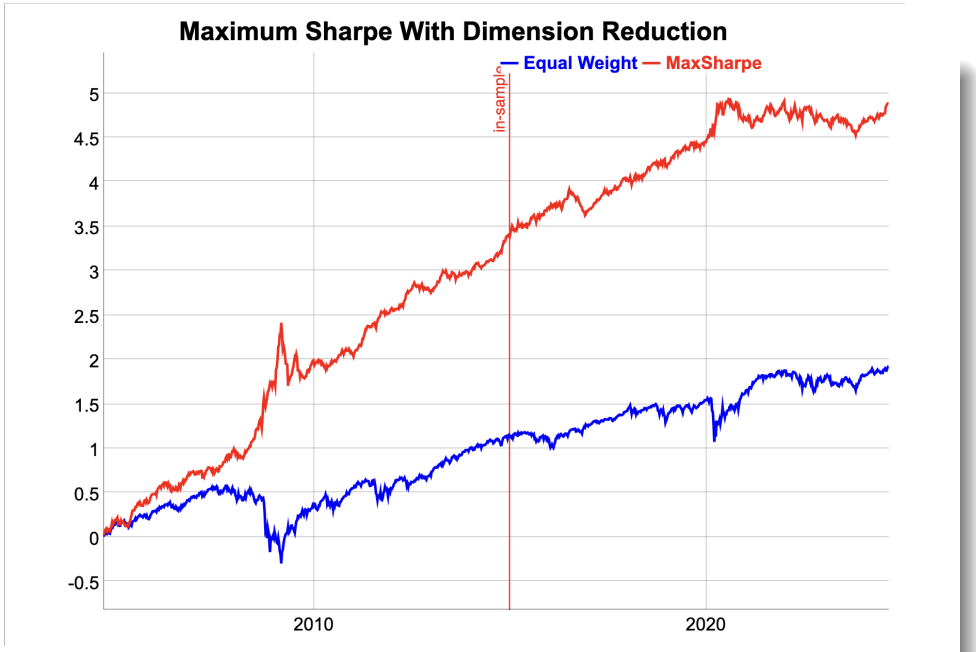
```
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv["/2014"], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv["2015/"], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative portfolio returns
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Out-of-Sample Maximum Sharpe Stock Portfolio") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeP
+   dyLegend(width=300)
```

Maximum Sharpe Portfolio 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 <- 10
> eigend <- eigen(covmat)
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> invred <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
> # Calculate portfolio weights and returns
> weightv <- invred %*% colmeanv
> rownames(weightv) <- colv
> pnls <- HighFreq::mult_mat(weightv, retp)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> pnls <- xts::xts(pnls, datev)
> pnls <- pnls*sd(retew["/2014"])/sd(pnls["/2014"])
```




```
> # Combine with equal weight
> wealthv <- cbind(retew, pnls)
> colnames(wealthv) <- c("Equal Weight", "MaxSharpe")
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv["/2014"], 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["2015/"], 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="Maximum Sharpe With Dimension Reduction") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", stroke
+   dyLegend(width=300)
```

Maximum Sharpe Portfolio With Return Shrinkage

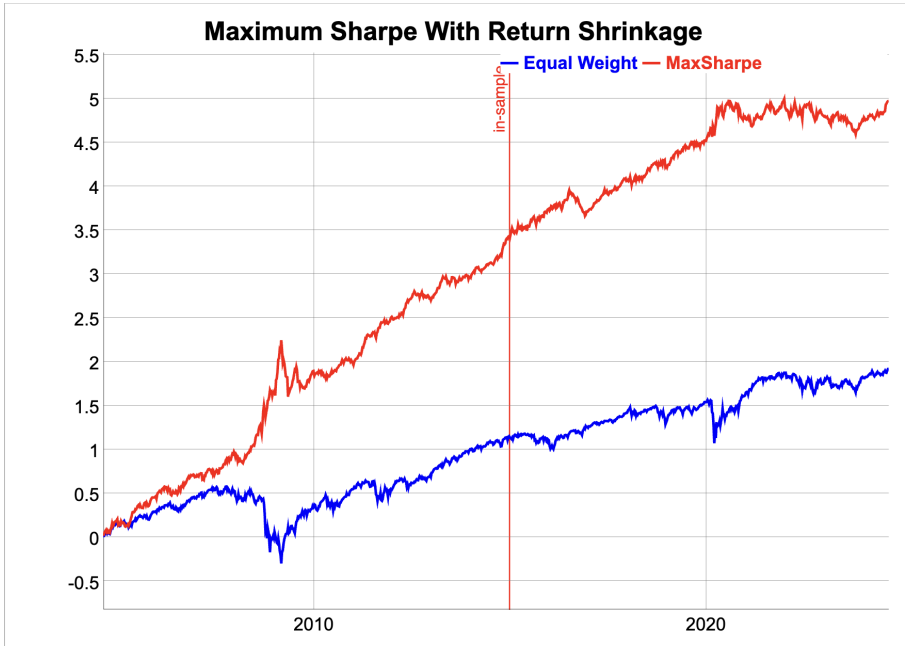
To further reduce the statistical noise, the individual returns r_i can be *shrunk* to the average portfolio returns $\bar{r}$:

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

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

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

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




```
> # Shrink the in-sample returns to their mean
> alphac <- 0.3
> retxm <- rowMeans(retx, na.rm=TRUE)
> retxis <- (1-alphac)*retx + alphac*retxm
> # Calculate portfolio weights and returns
> weightv <- invred %*% colMeans(retxis, na.rm=TRUE)
> rownames(weightv) <- colv
> pnls <- HighFreq::mult_mat(weightv, retp)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> pnls <- xts::xts(pnls, datev)
> pnls <- pnls*sd(retew["/2014"])/sd(pnls["/2014"])
```

```
> # Combine with equal weight
> wealthv <- cbind(retew, pnls)
> colnames(wealthv) <- c("Equal Weight", "MaxSharpe")
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv["/2014"], 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["2015/"], 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="Maximum Sharpe With Return Shrinkage") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", stroke
+   dyLegend(width=300)
```

Rolling Maximum Sharpe Stock Portfolio Strategy

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.

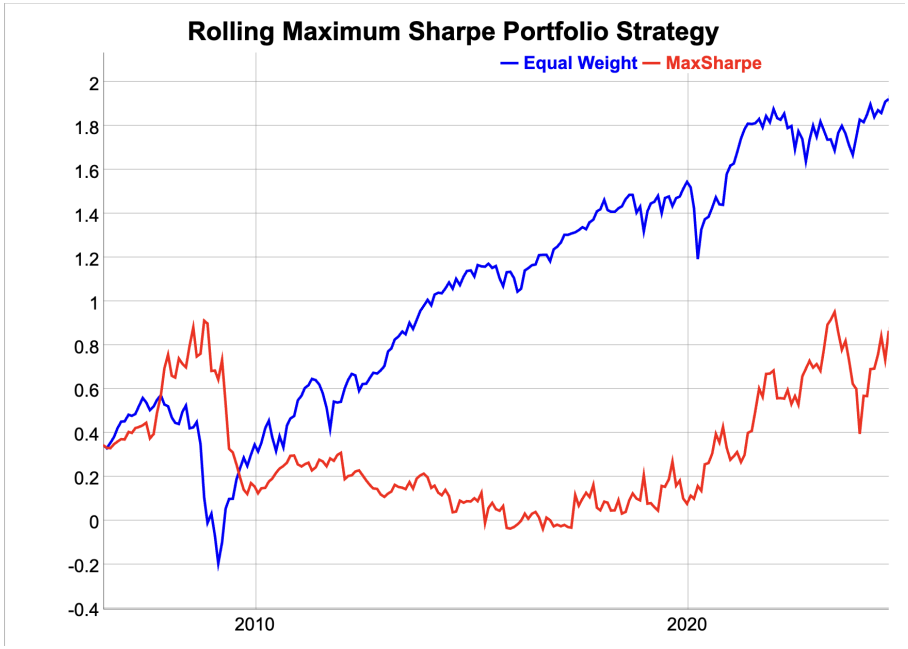
The strategy parameters are: the rebalancing frequency (annual, monthly, etc.), and the length of look-back interval.

```
> # 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 - takes very long!!!
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnls <- mclapply(1:(npts-1), function(tday) {
+     # Subset the excess returns
+     retis <- retp[startp[tday]:endd[tday], ]
+     retis[is.na(retis)] <- 0
+     invreg <- MASS::ginv(cov(retis, use="pairwise.complete.obs"))
+     # Calculate the maximum Sharpe ratio portfolio weights
+     retm <- colMeans(retis, na.rm=TRUE)
+     weightv <- invreg %*% retm
+     # Zero weights if sparse data
+     zerov <- sapply(retis, function(x) (sum(x == 0) > 5))
+     weightv[zerov] <- 0
+     # Calculate in-sample portfolio returns
+     pnlis <- (retis %*% weightv)
+     # Scale the weights to the in-sample volatility
+     weightv <- weightv*sd(retm)/sd(pnlis)
+     # Calculate the out-of-sample portfolio returns
+     retos <- retp[(endd[tday]+1):endd[tday+1], ]
+     pnls <- HighFreq::mult_mat(weightv, retos)
+     pnls <- rowMeans(pnls, na.rm=TRUE)
+     xts::xts(pnls, zoo::index(retos))
+ })  # end lapply
> pnls <- rutils::do_call(rbind, pnls)
> pnls <- rbind(retew[paste0("/", start(pnls)-1)], pnls*sd(retew)/sd
```

Performance of Rolling Maximum Sharpe Strategy

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

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




```
> # Plot cumulative strategy returns
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Maximum Sharpe Portfolio Strategy") %>%
+   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), nc=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 decomposi

    // 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
```

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
```

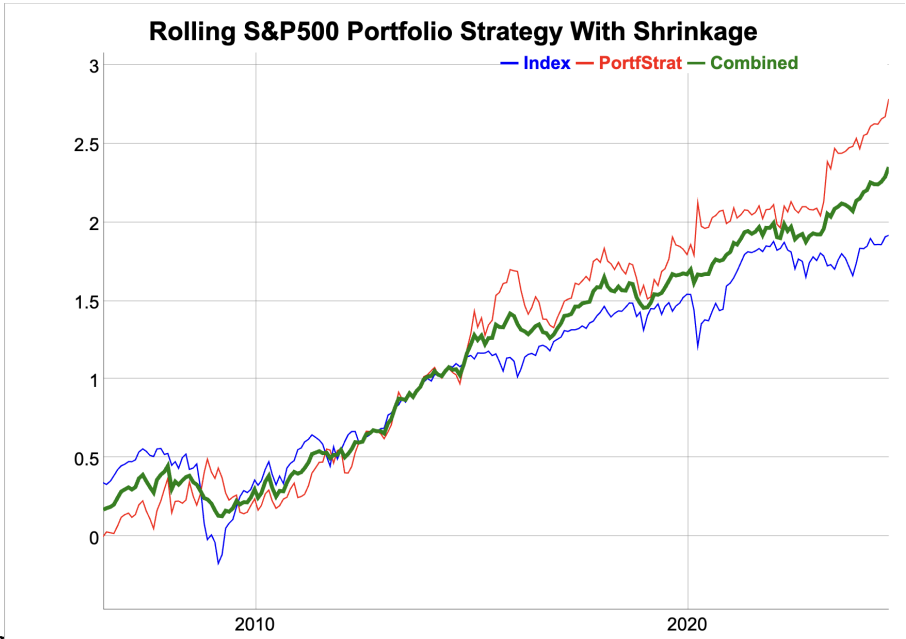
Rolling Max Sharpe Strategy With Dimension Reduction and Shrinkage

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

The *rolling portfolio* strategy has a higher *Sharpe* ratio and also higher absolute returns than the equal weight portfolio.

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

```
> # Shift the end points to C++ convention
> endd <- (endd - 1)
> endd[endd < 0] <- 0
> startp <- (startp - 1)
> startp[startp < 0] <- 0
> # Specify dimension reduction and return shrinkage using list of p
> dimax <- 9
> alphac <- 0.8
> controlv <- HighFreq::param_portf(method="maxsharpe",
+   dimax=dimax, alpha=alphac)
> # Perform backtest in Rcpp - takes very long!!!
> retx <- (retp - raterf)
> pnls <- HighFreq::back_test(retx=retx, retp=retp,
+   startp=startp, endd=endd, controlv=controlv)
> pnls <- pnls*sd(retew)/sd(pnls)
```




```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Equal", "MaxSharpe", "Combined")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot cumulative strategy returns
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling 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

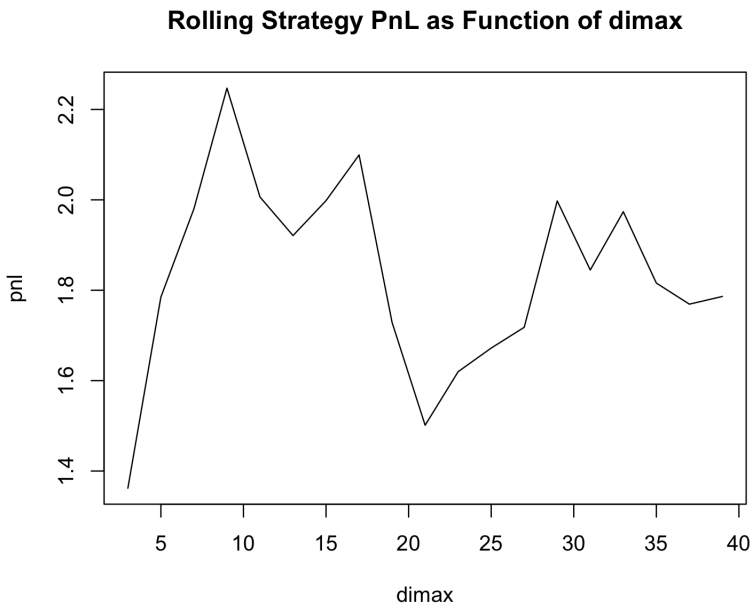
The optimal value of the dimension reduction parameter *dimax* can be determined using *backtesting*.

The best dimension reduction parameter for this portfolio of stocks is equal to `dimax=9`, which means relatively weak dimension reduction.

The dependence of the out-of-sample returns on the `dimax` parameter reflects the *bias-variance tradeoff*.

If the `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]
```




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

Optimal Shrinkage Parameter

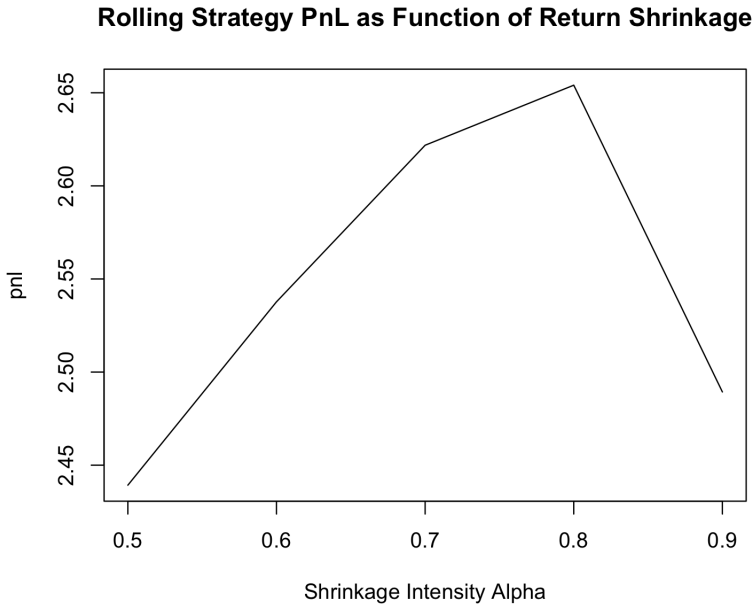
The optimal value of the return shrinkage intensity parameter α can be determined using *backtesting*.

The best return shrinkage parameter for this portfolio of stocks is equal to $\alpha = 0.8$, which means strong return shrinkage.

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

If the α 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]
```




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

Optimal Look-back Interval

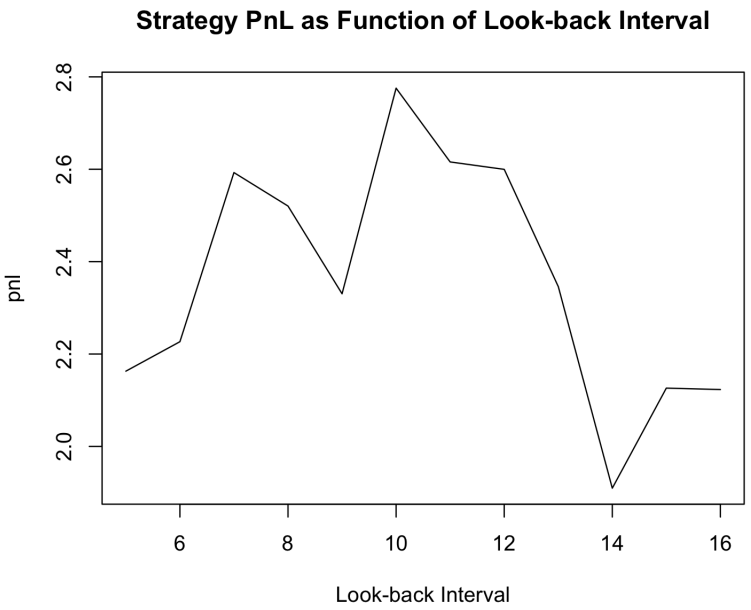
The optimal length of the look-back interval can be determined using *backtesting*.

The optimal length 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*.



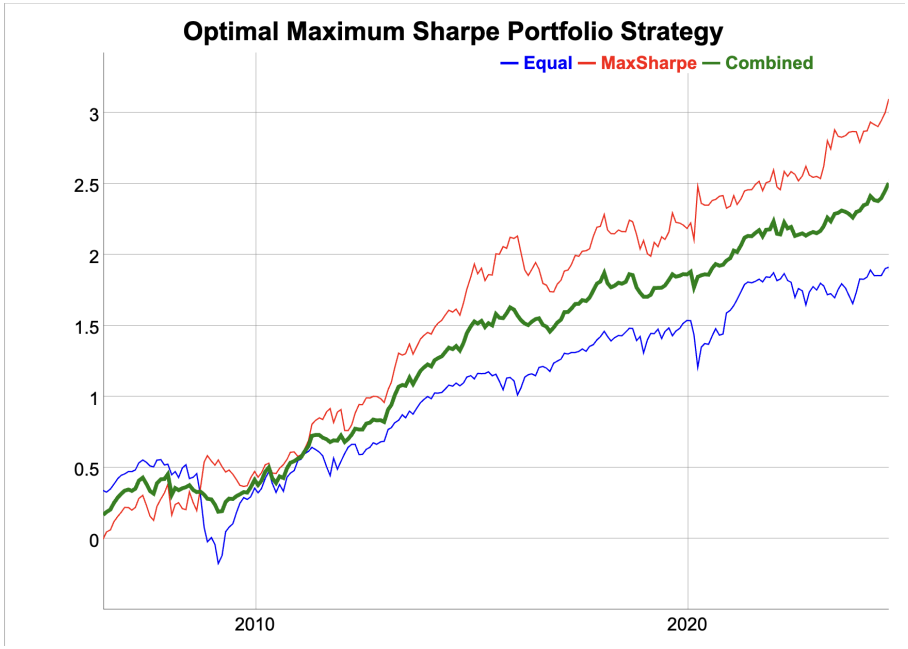

```
> # Create list of model parameters
> controlv <- HighFreq::param_portf(method="maxsharpe",
+       dimax=dimax, alpha=alphac)
> # Perform backtest over look-backs
> lookbv <- seq(from=5, to=16, by=1)
> pnls <- mclapply(lookbv, function(lookb) {
+   startp <- c(rep_len(0, lookb), endd[1:(npts-lookb)])
+   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)
> whichmax <- which.max(profilev)
> lookb <- lookbv[whichmax]
```

```
> # Dygraph the cumulative wealth
> plot(x=lookbv, y=profilev, t="l", main="MaxSharpe PnL as Function
+   xlab="Look-back Interval", ylab="pnl")
```

Optimal Max Sharpe Portfolio Optimization Strategy

The optimal *rolling portfolio optimization* strategy, with dimension reduction and return shrinkage, has both a higher *Sharpe* ratio and higher absolute returns than the equal weight portfolio.

Combining the optimal rolling portfolio strategy with the equal weight portfolio results in an even higher *Sharpe* ratio.




```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> pnls <- pnls[[whichmax]]
> pnls <- pnls*sd(retew)/sd(pnls)
> wealthv <- cbind(retew, pnls, (pnls + retew)/2)
> colnames(wealthv) <- c("Equal", "MaxSharpe", "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 Maximum Sharpe Portfolio Strategy") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

draft: Applications of Machine Learning to Finance

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

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

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

The *generalized inverse* $\mathbb{C}_g^{-1}$ is calculated by removing the zero eigenvalues, and keeping only the first n *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 function `MASS::ginv()` calculates the *generalized* inverse of a matrix.

```
> # 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 matrix
> notzero <- (eigenval > (precv*eigenval[1]))
> invred <- eigenvec[, notzero] %*%
+   (t(eigenvec[, notzero]) / eigenval[notzero])
> # Verify inverse property of invred
> all.equal(covmat, covmat %*% invred %*% covmat)
> # Calculate generalized inverse of covmat
> invreg <- MASS::ginv(covmat)
> # Verify that invreg is same as invred
> all.equal(invreg, invred)
```