FRE7241 Algorithmic Portfolio Management

Lecture#5, Spring 2023

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

April 25, 2023

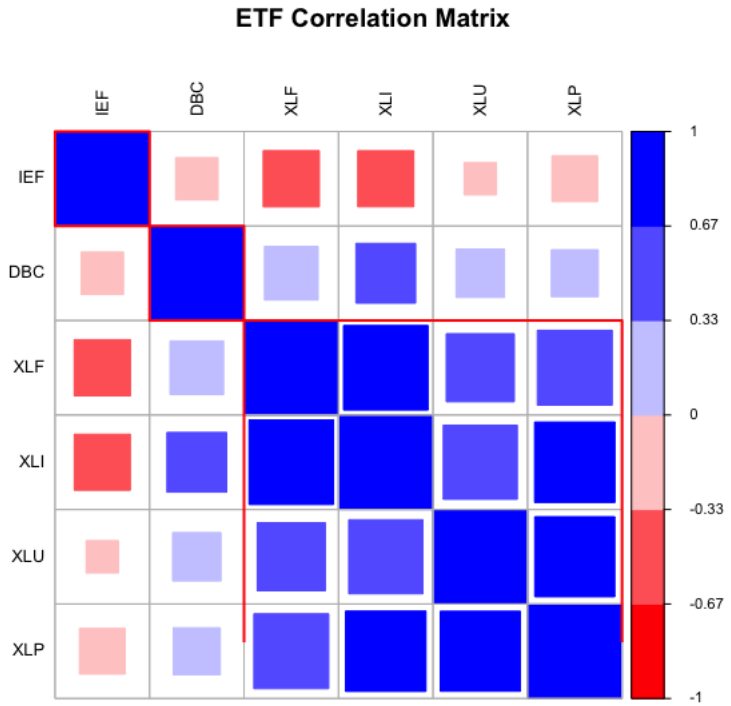
Covariance Matrix of ETF Returns

The covariance matrix $\mathbb{C}$, of the return matrix $\mathbf{r}$ is given by:

$$\mathbb{C} = \frac{(\mathbf{r} - \bar{\mathbf{r}})^T (\mathbf{r} - \bar{\mathbf{r}})}{n - 1}$$

If the returns are *standardized* (de-meaned and scaled) then the covariance matrix is equal to the correlation matrix.

```
> # Select ETF symbols
> symbolv <- c("IEF", "DBC", "XLU", "XLF", "XLP", "XLI")
> # Calculate ETF prices and log returns
> pricev <- rutils::etfenv$prices[, symbolv]
> # Applying zoo::na.locf() can produce bias of the correlations
> # pricev <- zoo::na.locf(pricev, na.rm=FALSE)
> # pricev <- zoo::na.locf(pricev, fromLast=TRUE)
> pricev <- na.omit(pricev)
> retp <- rutils::diffit(log(pricev))
> # Calculate covariance matrix
> covmat <- cov(retp)
> # Standardize (de-mean and scale) the returns
> retp <- lapply(retp, function(x) {(x - mean(x))/sd(x)})
> retp <- rutils::do_call(cbind, retp)
> round(sapply(retp, mean), 6)
> sapply(retp, sd)
> # Alternative (much slower) center (de-mean) and scale the retur
> # retp <- apply(retp, 2, scale)
> # retp <- xts::xts(retp, zoo::index(pricev))
> # Alternative (much slower) center (de-mean) and scale the retur
> # retp <- scale(retp, center=TRUE, scale=TRUE)
> # retp <- xts::xts(retp, zoo::index(pricev))
> # Alternative (much slower) center (de-mean) and scale the retur
> # retp <- t(retp) - colMeans(retp)
> # retp <- retp/sqrt(rowSums(retp^2)/(NCOL(retp)-1))
> # retp <- t(retp)
```




```
> # Calculate correlation matrix
> cormat <- cor(retp)
> # Reorder correlation matrix based on clusters
> library(corrplot)
> ordern <- corrMatOrder(cormat, order="hclust",
+   hclust.method="complete")
> cormat <- cormat[ordern, ordern]
> # Plot the correlation matrix
> colorv <- colorRampPalette(c("red", "white", "blue"))
> # x11(width=6, height=6)
> corrplot(cormat, title=NA, tl.col="black", mar=c(0,0,0,0),
+     method="square", col=colorv(NCOL(cormat)), tl.cex=0.8,
+     cl.offset=0.75, cl.cex=0.7, cl.align.text="l", cl.ratio=0.25)
```

Principal Component Vectors

Principal components are linear combinations of the k return vectors $\mathbf{r}_i$:

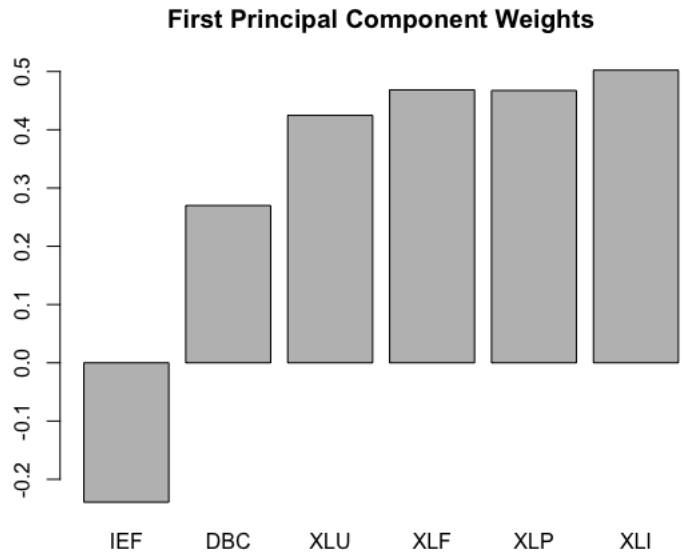
$$\mathbf{pc}_j = \sum_{i=1}^{k} w_{ij}\, \mathbf{r}_i$$

Where $\mathbf{w}_j$ is a vector of weights (loadings) of the *principal component* j, with $\mathbf{w}_j^T \mathbf{w}_j = 1$.

The weights $\mathbf{w}_j$ are chosen to maximize the variance of the *principal components*, under the condition that they are orthogonal:

$$\mathbf{w}_j = \arg\max \left\{ \mathbf{pc}_j^T\, \mathbf{pc}_j \right\}$$

$$\mathbf{pc}_i^T\, \mathbf{pc}_j = 0 \; (i \neq j)$$




```
> # Create initial vector of portfolio weights
> nweights <- NROW(symbolv)
> weightv <- rep(1/sqrt(nweights), nweights)
> names(weightv) <- symbolv
> # Objective function equal to minus portfolio variance
> objfun <- function(weightv, retp) {
+   retp <- retp %*% weightv
+   -sum(retp^2) + 1e4*(1 - sum(weightv^2))^2
+ }  # end objfun
> # Objective for equal weight portfolio
> objfun(weightv, retp)
> # Compare speed of vector multiplication methods
> summary(microbenchmark(
+   transp=(t(retp[, 1]) %*% retp[, 1]),
+   sumv=sum(retp[, 1]^2),
+   times=10))[, c(1, 4, 5)]
```

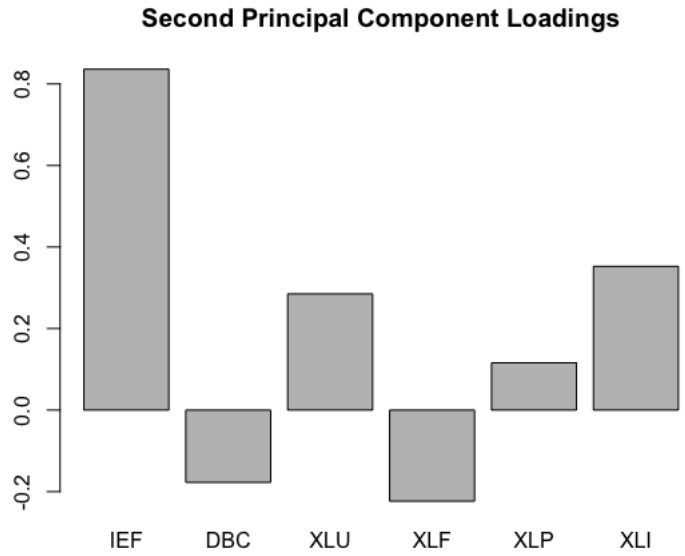
```
> # Find weights with maximum variance
> optim1 <- optim(par=weightv,
+   fn=objfun,
+   retp=retp,
+   method="L-BFGS-B",
+   upper=rep(10.0, nweights),
+   lower=rep(-10.0, nweights))
> # Optimal weights and maximum variance
> weights1 <- optim1$par
> -objfun(weights1, retp)
> # Plot first principal component weights
> barplot(weights1, names.arg=names(weights1), xlab="", ylab="",
+   main="First Principal Component Weights")
```

Higher Order Principal Components

The second *principal component* can be calculated by maximizing its variance, under the constraint that it must be orthogonal to the first *principal component*.

Similarly, higher order *principal components* can be calculated by maximizing their variances, under the constraint that they must be orthogonal to all the previous *principal components*.

```
> # PC1 returns
> pc1 <- drop(retp %*% weights1)
> # Redefine objective function
> objfun <- function(weightv, retp) {
+   retp <- retp %*% weightv
+   -sum(retp^2) + 1e4*(1 - sum(weightv^2))^2 +
+     1e4*(sum(weights1*weightv))^2
+ }  # end objfun
> # Find second PC weights using parallel DEoptim
> optim1 <- DEoptim::DEoptim(fn=objfun,
+   upper=rep(10, NCOL(retp)),
+   lower=rep(-10, NCOL(retp)),
+   retp=retp, control=list(parVar="weights1",
+     trace=FALSE, itermax=1000, parallelType=1))
```




```
> # PC2 weights
> weights2 <- optim1$optim$bestmem
> names(weights2) <- colnames(retp)
> sum(weights2^2)
> sum(weights1*weights2)
> # PC2 returns
> pc2 <- drop(retp %*% weights2)
> # Plot second principal component loadings
> barplot(weights2, names.arg=names(weights2), xlab="", ylab="",
+   main="Second Principal Component Loadings")
```

Eigenvalues of the Correlation Matrix

The portfolio variance: $\mathbf{w}^T \mathbb{C} \mathbf{w}$ can be maximized under the *quadratic* weights constraint $\mathbf{w}^T \mathbf{w} = 1$, by maximizing the *Lagrangian* $\mathcal{L}$:

$$\mathcal{L} = \mathbf{w}^T \mathbb{C} \mathbf{w} - \lambda (\mathbf{w}^T \mathbf{w} - 1)$$

Where λ is a *Lagrange multiplier*.

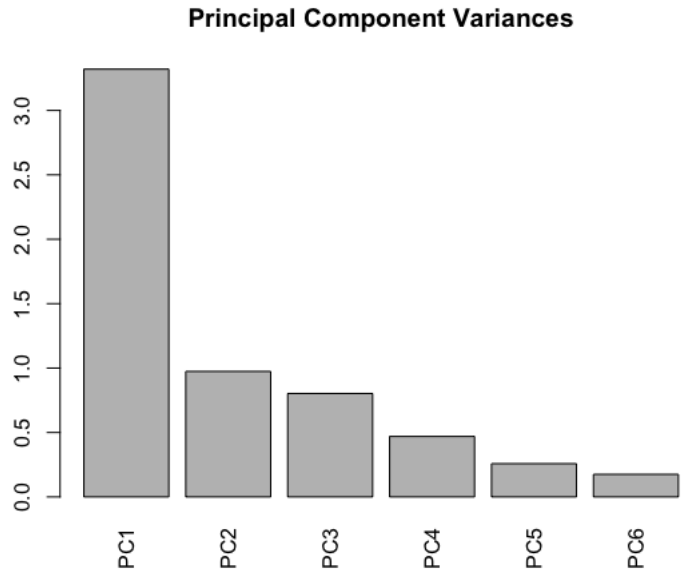
The maximum variance portfolio weights can be found by differentiating $\mathcal{L}$ with respect to $\mathbf{w}$ and setting it to zero:

$$\mathbb{C} \mathbf{w} = \lambda \mathbf{w}$$

This is the *eigenvalue* equation of the covariance matrix $\mathbb{C}$, with the optimal weights $\mathbf{w}$ forming an *eigenvector*, and λ is the *eigenvalue* corresponding to the *eigenvector* $\mathbf{w}$.

The *eigenvalues* are the variances of the *eigenvectors*, and their sum is equal to the sum of the return variances:

$$\sum_{i=1}^{k} \lambda_i = \frac{1}{1-k} \sum_{i=1}^{k} \mathbf{r}_i^T \mathbf{r}_i$$




```
> # Calculate the eigenvalues and eigenvectors
> eigend <- eigen(cormat)
> eigend$vectors
> # Compare with optimization
> all.equal(sum(diag(cormat)), sum(eigend$values))
> all.equal(abs(eigend$vectors[, 1]), abs(weights1), check.attribute
> all.equal(abs(eigend$vectors[, 2]), abs(weights2), check.attribute
> all.equal(eigend$values[1], var(pc1), check.attributes=FALSE)
> all.equal(eigend$values[2], var(pc2), check.attributes=FALSE)
> # Eigenvalue equations
> (cormat %*% weights1) / weights1 / var(pc1)
> (cormat %*% weights2) / weights2 / var(pc2)
> # Plot eigenvalues
> barplot(eigend$values, names.arg=paste0("PC", 1:nweights),
+   las=3, xlab="", ylab="", main="Principal Component Variances")
```

Principal Component Analysis Versus *Eigen Decomposition*

Principal Component Analysis (*PCA*) is equivalent to the *eigen decomposition* of either the correlation or the covariance matrix.

If the input time series *are* scaled, then *PCA* is equivalent to the eigen decomposition of the *correlation matrix*.

If the input time series *are not* scaled, then *PCA* is equivalent to the eigen decomposition of the *covariance matrix*.

Scaling the input time series improves the accuracy of the *PCA dimension reduction*, allowing a smaller number of *principal components* to more accurately capture the data contained in the input time series.

The number of *eigenvalues* is equal to the dimension of the covariance matrix.

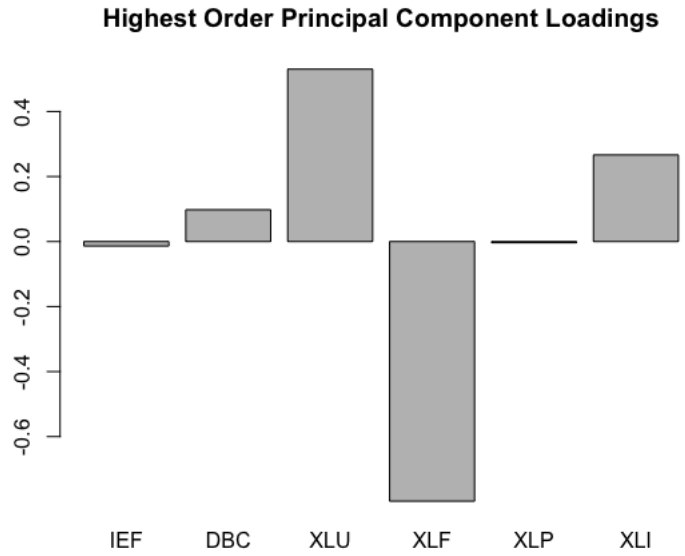
```
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> # Perform PCA with scaling
> pcad <- prcomp(retp, scale=TRUE)
> # Compare outputs
> all.equal(eigend$values, pcad$sdev^2)
> all.equal(abs(eigend$vectors), abs(pcad$rotation),
+     check.attributes=FALSE)
> # Eigen decomposition of covariance matrix
> eigend <- eigen(covmat)
> # Perform PCA without scaling
> pcad <- prcomp(retp, scale=FALSE)
> # Compare outputs
> all.equal(eigend$values, pcad$sdev^2)
> all.equal(abs(eigend$vectors), abs(pcad$rotation),
+     check.attributes=FALSE)
```

Minimum Variance Portfolio

The highest order *principal component*, with the smallest eigenvalue, has the lowest possible variance, under the *quadratic* weights constraint: $\mathbf{w}^T\mathbf{w} = 1$.

So the highest order *principal component* is equal to the *Minimum Variance Portfolio*.

```
> # Redefine objective function to minimize variance
> objfun <- function(weightv, retp) {
+   retp <- retp %*% weightv
+   sum(retp^2) + 1e4*(1 - sum(weightv^2))^2
+ }  # end objfun
> # Find highest order PC weights using parallel DEoptim
> optiml <- DEoptim::DEoptim(fn=objfun,
+   upper=rep(10, NCOL(retp)),
+   lower=rep(-10, NCOL(retp)),
+   retp=retp, control=list(trace=FALSE,
+     itermax=1000, parallelType=1))
> # PC6 weights and returns
> weights6 <- optiml$optim$bestmem
> names(weights6) <- colnames(retp)
> sum(weights6^2)
> sum(weights1*weights6)
> # Compare with eigend vector
> weights6
> eigend$vectors[, 6]
> # Calculate objective function
> objfun(weights6, retp)
> objfun(eigend$vectors[, 6], retp)
```




```
> # Plot highest order principal component loadings
> weights6 <- eigend$vectors[, 6]
> names(weights6) <- colnames(retp)
> barplot(weights6, names.arg=names(weights6), xlab="", ylab="",
+   main="Highest Order Principal Component Loadings")
```

Principal Component Analysis of ETF Returns

Principal Component Analysis (*PCA*) is a *dimension reduction* technique, that explains the returns of a large number of correlated time series as linear combinations of a smaller number of principal component time series.

The input time series are often scaled by their standard deviations, to improve the accuracy of *PCA dimension reduction*, so that more information is retained by the first few *principal component* time series.

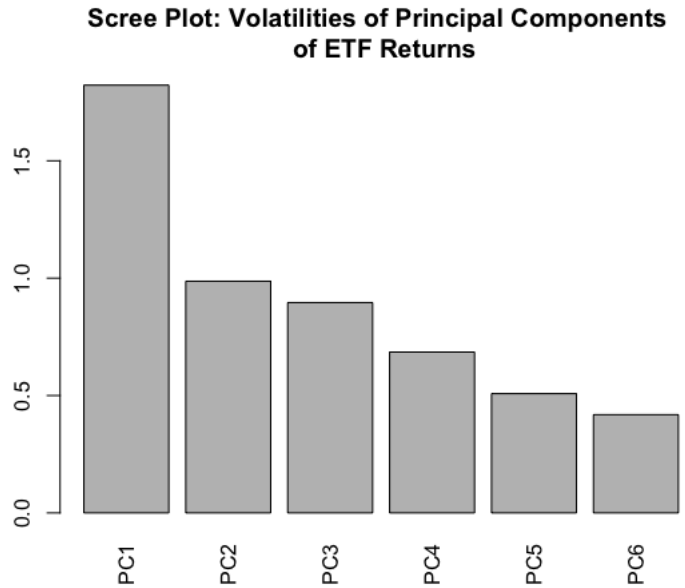
If the input time series are not scaled, then *PCA* analysis is equivalent to the *eigen decomposition* of the covariance matrix, and if they are scaled, then *PCA* analysis is equivalent to the *eigen decomposition* of the correlation matrix.

The function `prcomp()` performs *Principal Component Analysis* on a matrix of data (with the time series as columns), and returns the results as a list of class `prcomp`.

The `prcomp()` argument `scale=TRUE` specifies that the input time series should be scaled by their standard deviations.

The *Kaiser-Guttman* rule uses only *principal components* with *variance* greater than 1.

Another rule is to use the *principal components* with the largest standard deviations which sum up to 80% of the total variance of returns.



A *scree plot* is a bar plot of the volatilities of the *principal components*.

```
> # Perform principal component analysis PCA
> pcad <- prcomp(retp, scale=TRUE)
> # Plot standard deviations of principal components
> barplot(pcad$sdev, names.arg=colnames(pcad$rotation),
+   las=3, xlab="", ylab="",
+   main="Scree Plot: Volatilities of Principal Components \n of ETF
> # Calculate the number of principal components which sum up to at
> pcavar <- pcad$sdev^2
> which(cumsum(pcavar)/sum(pcavar) > 0.8)[1]
```

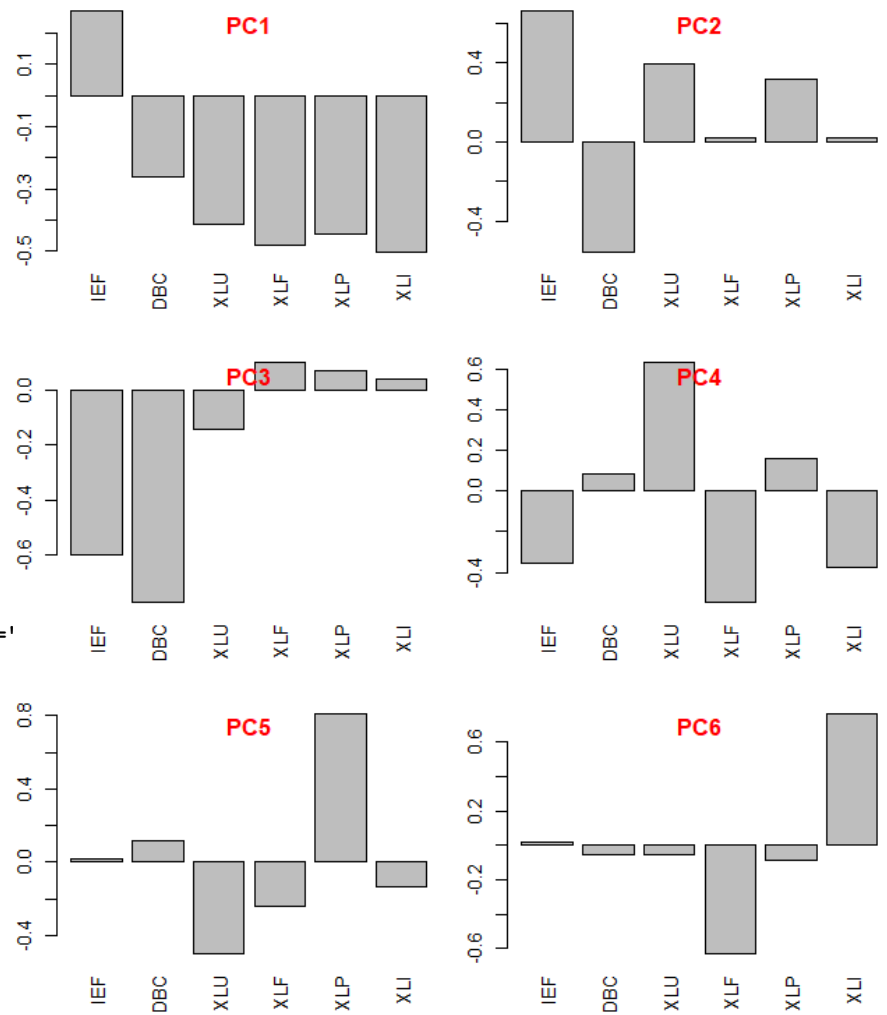
Principal Component Loadings (Weights)

Principal component loadings are the weights of portfolios which have mutually orthogonal returns.

The *principal component* (*PC*) portfolios represent the different orthogonal modes of the return variance.

The *PC* portfolios typically consist of long or short positions of highly correlated groups of assets (clusters), so that they represent relative value portfolios.

```
> # Plot barplots with PCA loadings (weights) in multiple panels
> pcad$rotation
> # x11(width=6, height=7)
> par(mfrow=c(nweights/2, 2))
> par(mar=c(3, 2, 2, 1), oma=c(0, 0, 0, 0))
> for (ordern in 1:nweights) {
+   barplot(pcad$rotation[, ordern], las=3, xlab="", ylab="", main='
+   title(paste0("PC", ordern), line=-1, col.main="red")
+ }  # end for
```



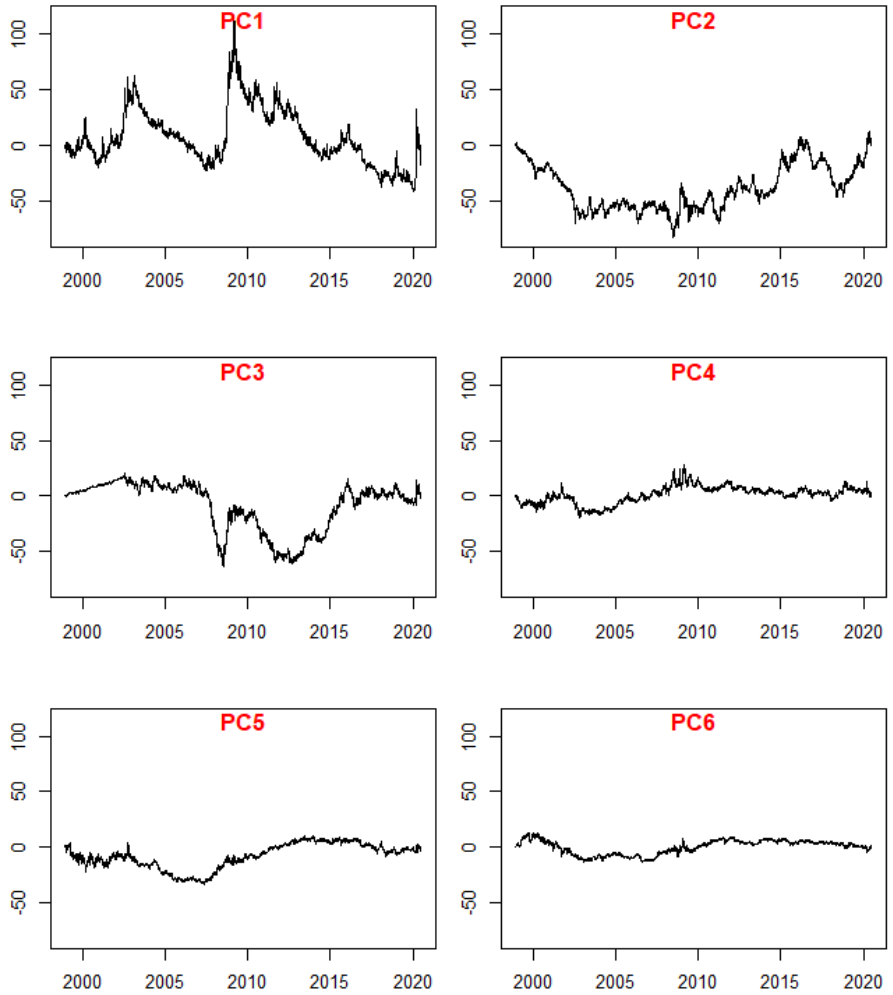
Principal Component Time Series

The time series of the *principal components* can be calculated by multiplying the loadings (weights) times the original data.

The *principal component* time series have mutually orthogonal returns.

Higher order *principal components* are gradually less volatile.

```
> # Calculate products of principal component time series
> round(t(pcad$x) %*% pcad$x, 2)
> # Calculate principal component time series from returns
> datev <- zoo::index(pricev)
> retpca <- xts::xts(retp %*% pcad$rotation, order.by=datev)
> round(cov(retpca), 3)
> all.equal(coredata(retpca), pcad$x, check.attributes=FALSE)
> pcacum <- cumsum(retpca)
> # Plot principal component time series in multiple panels
> rangev <- range(pcacum)
> for (ordern in 1:nweights) {
+   plot.zoo(pcacum[, ordern], ylim=rangev, xlab="", ylab="")
+   title(paste0("PC", ordern), line=-1, col.main="red")
+ }  # end for
```



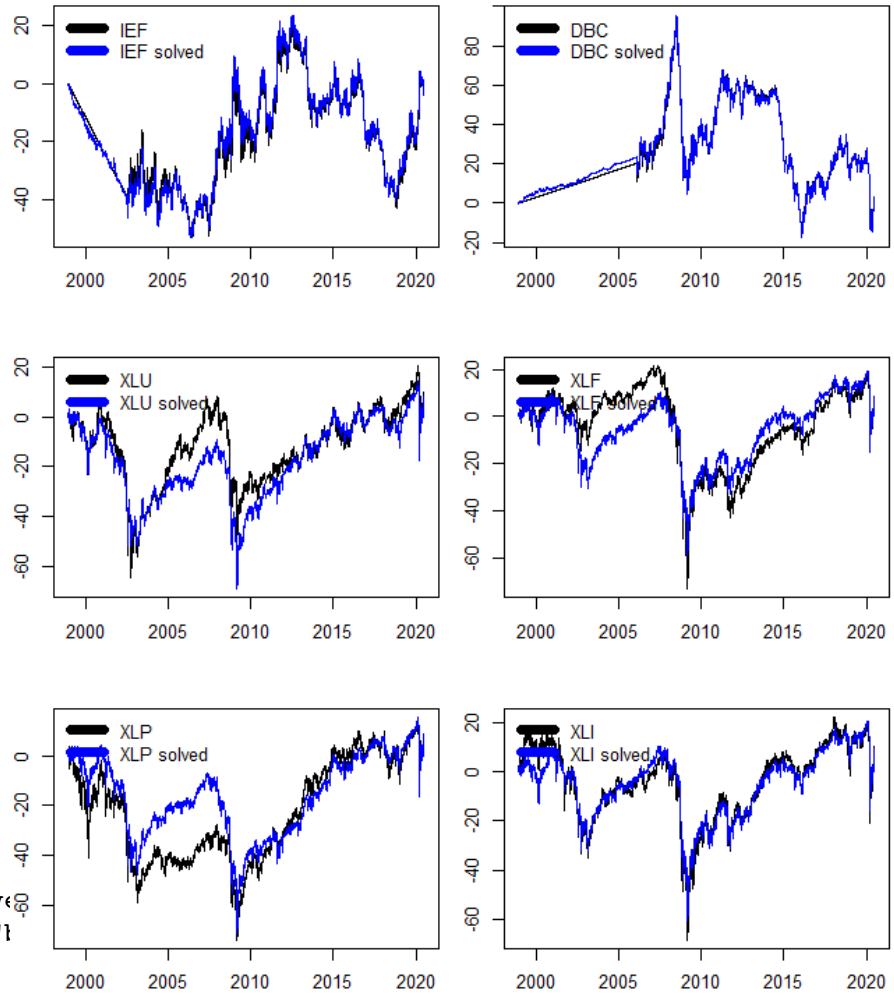
Dimension Reduction Using Principal Component Analysis

The original time series can be calculated exactly from the time series of all the *principal components*, by inverting the loadings matrix.

The original time series can be calculated approximately from just the first few *principal components*, which demonstrates that *PCA* is a form of *dimension reduction*.

The function `solve()` solves systems of linear equations, and also inverts square matrices.

```
> # Invert all the principal component time series
> retpca <- retp %*% pcad$rotation
> solved <- retpca %*% solve(pcad$rotation)
> all.equal(coredata(retp), solved)
> # Invert first 3 principal component time series
> solved <- retpca[, 1:3] %*% solve(pcad$rotation)[1:3, ]
> solved <- xts::xts(solved, datev)
> solved <- cumsum(solved)
> retc <- cumsum(retp)
> # Plot the solved returns
> for (symbol in symbolv) {
+   plot.zoo(cbind(retc[, symbol], solved[, symbol]),
+     plot.type="single", col=c("black", "blue"), xlab="", ylab="")
+   legend(x="topleft", bty="n", legend=paste0(symbol, c("", " solve
+    title=NULL, inset=0.0, cex=1.0, lwd=6, lty=1, col=c("black", "b
+ }  # end for
```



Condition Number of Correlation Matrices

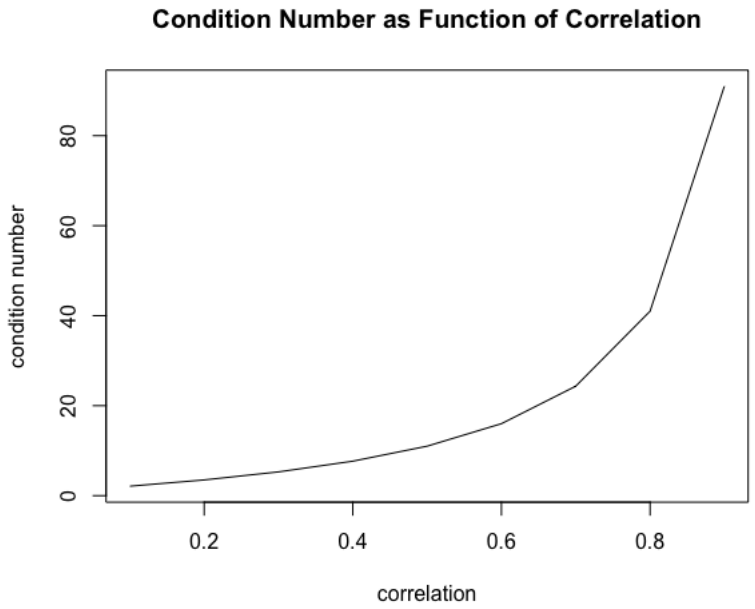
The condition number κ of a correlation matrix is equal to the ratio of its largest eigenvalue divided by the smallest:

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}}$$

The condition number depends on the level of correlations. If correlations are small then the eigenvalues are close to 1 and the condition number is also close to 1. If the correlations are close to 1 then the condition number is large.

A large condition number indicates the presence of small eigenvalues, and a correlation matrix close to *singular*, with a poorly defined inverse matrix.

A very large condition number indicates that the correlation matrix is close to being *singular*.




```
> # Create a matrix with low correlation
> ndata <- 10
> cormat <- matrix(rep(0.1, ndata^2), nc=ndata)
> diag(cormat) <- rep(1, ndata)
> # Calculate the condition number
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> max(eigenval)/min(eigenval)
> # Create a matrix with high correlation
> cormat <- matrix(rep(0.9, ndata^2), nc=ndata)
> diag(cormat) <- rep(1, ndata)
> # Calculate the condition number
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> max(eigenval)/min(eigenval)
```

```
> # Calculate the condition numbers as function correlation
> corvec <- seq(0.1, 0.9, 0.1)
> condvec <- sapply(corvec, function(corv) {
+   cormat <- matrix(rep(corv, ndata^2), nc=ndata)
+   diag(cormat) <- rep(1, ndata)
+   eigend <- eigen(cormat)
+   eigenval <- eigend$values
+   max(eigenval)/min(eigenval)
+ })  # end sapply
> # Plot the condition numbers
> plot(x=corvec, y=condvec, t="l",
+   main="Condition Number as Function of Correlation",
+   xlab="correlation", ylab="condition number")
```

Condition Number for Small Number of Observations

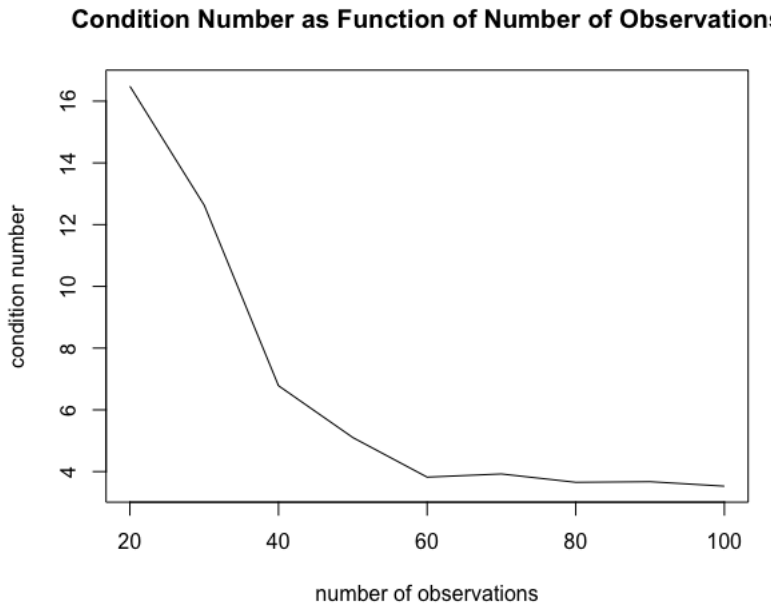
The condition number also depends on the number of observations.

If the number of observations (rows of data) is small compared to the number of stocks (columns), then the condition number can be large, even if the returns are not correlated.

That's because as the number of rows of data decreases, the returns become more *collinear*, and the sample correlation matrix becomes more *singular*, with some very small eigenvalues.

In practice, calculating the inverse correlation matrix of returns faces two challenges: not enough rows of data and correlated returns.

In both cases, the problem is that the columns of returns are close to *collinear*.




```
> # Simulate uncorrelated stock returns
> nstocks <- 10
> nrows <- 100
> set.seed(1121)  # Initialize random number generator
> retp <- matrix(rnorm(nstocks*nrows), nc=nstocks)
> # Calculate the condition numbers as function of number of observations
> obsvec <- seq(20, nrows, 10)
> condvec <- sapply(obsvec, function(nobs) {
+   cormat <- cor(retp[1:nobs, ])
+   eigend <- eigen(cormat)
+   eigenval <- eigend$values
+   max(eigenval)/min(eigenval)
+ })  # end sapply
```

```
> # Plot the condition numbers
> plot(x=obsvec, y=condvec, t="l",
+   main="Condition Number as Function of Number of Observations",
+   xlab="number of observations", ylab="condition number")
```

The Correlations of Stock Returns

Estimating the correlations 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 `cor()` calculates the correlation 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 correlation matrices which are not positive semi-definite.

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

```
> # Load daily S&P500 log percentage stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> # Calculate the number of NA values in returns
> retp <- returns
> colSums(is.na(retp))
> # Calculate the correlations ignoring NA values
> cor(retp$DAL, retp$FOXA, use="pairwise.complete.obs")
> cor(na.omit(retp[, c("DAL", "FOXA")]))[2]
> cormat <- cor(retp, use="pairwise.complete.obs")
> sum(is.na(cormat))
> cormat[is.na(cormat)] <- 0
```

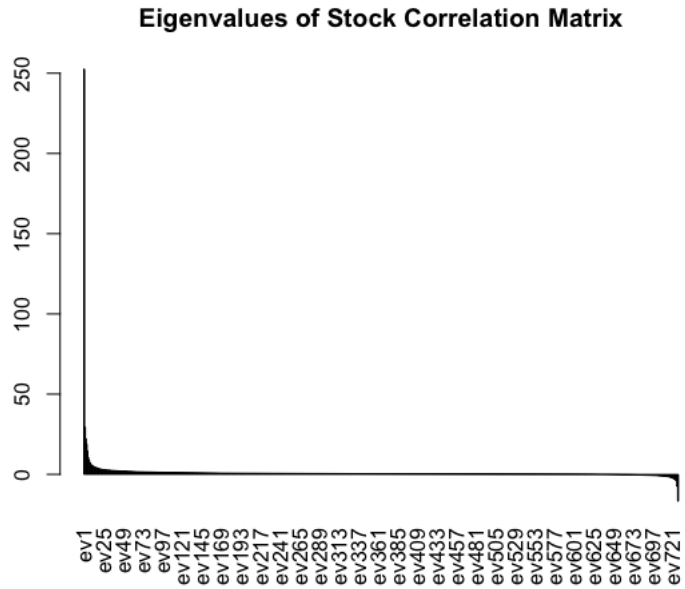
Principal Component Analysis of Stock Returns

Removing `NA` values in pairs of stock returns can produce correlation matrices which are not positive semi-definite.

The function `prcomp()` produces an error when the correlation matrix is not positive semi-definite, so instead, *eigen decomposition* can be applied to perform *Principal Component Analysis*.

If some of the eigenvalues are negative, then the condition number is calculated using the eigenvalue with the smallest absolute value.

```
> # Perform principal component analysis PCA - produces error
> pcad <- prcomp(retp, scale=TRUE)
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> # Calculate the eigenvalues and eigenvectors
> eigenval <- eigend$values
> eigenvec <- eigend$vectors
> # Calculate the number of negative eigenvalues
> sum(eigenval<0)
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
> # Calculate the number of eigenvalues which sum up to at least 80% of the total variance
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
```




```
> # Plot the eigenvalues
> barplot(eigenval, xlab="", ylab="", las=3,
+   names.arg=paste0("ev", 1:NROW(eigenval)),
+   main="Eigenvalues of Stock Correlation Matrix")
```

Principal Component Analysis of Low and High Volatility Stocks

Low and high volatility stocks have different correlations and principal components.

Low volatility stocks have higher correlations than high volatility stocks, so their correlation matrix has a larger condition number than high volatility stocks.

But low volatility stocks can be explained by a smaller number of principal components, compared to high volatility stocks.

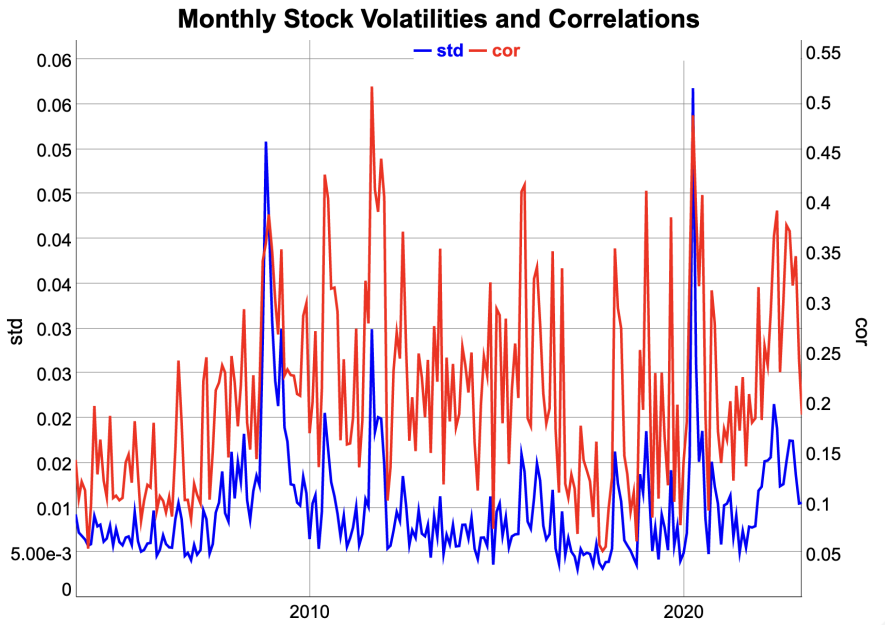
```
> # Calculate the stock variance
> varv <- sapply(retp, var, na.rm=TRUE)
> # Calculate the returns of low and high volatility stocks
> nstocks <- NCOL(retp)
> medianv <- median(varv)
> retlow <- retp[, varv <= medianv]
> rethigh <- retp[, varv > medianv]
> # Calculate the correlations of low volatility stocks
> cormat <- cor(retlow, use="pairwise.complete.obs")
> cormat[is.na(cormat)] <- 0
> # Calculate the mean correlations
> mean(cormat[upper.tri(cormat)])
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> # Calculate the number of negative eigenvalues
> sum(eigenval < 0)
> # Calculate the number of eigenvalues which sum up to at least 80%
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
> # Calculate the correlations of high volatility stocks
> cormat <- cor(rethigh, use="pairwise.complete.obs")
> cormat[is.na(cormat)] <- 0
> # Calculate the mean correlations
> mean(cormat[upper.tri(cormat)])
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> # Calculate the number of negative eigenvalues
> sum(eigenval < 0)
> # Calculate the number of eigenvalues which sum up to at least 80%
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
```

Stock Correlations in Periods of High and Low Volatility

Correlations of stock returns are higher in time intervals with high volatility.

Stock returns have *high correlations* in time intervals with *high volatility*, and vice versa.

```
> # Subset (select) the stock returns after the start date of VTI
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> retp <- returns[zoo::index(retvti)]
> datev <- zoo::index(retp)
> retvti <- retvti[datev]
> nrows <- NROW(retp)
> nstocks <- NCOL(retp)
> head(retp[, 1:5])
> # Calculate the monthly end points
> endd <- rutils::calc_endpoints(retvti, interval="months")
> retvti[head(endd)]
> retvti[tail(endd)]
> # Remove stub interval at the end
> endd <- endd[-NROW(endd)]
> npts <- NROW(endd)
> # Calculate the monthly stock volatilities and correlations
> stdcor <- sapply(2:npts, function(endp) {
+   # cat("endp = ", endp, "\n")
+   retp <- retp[endd[endp-1]:endd[endp]]
+   cormat <- cor(retp, use="pairwise.complete.obs")
+   cormat[is.na(cormat)] <- 0
+   c(std=sd(retvti[endd[endp-1]:endd[endp]]),
+     cor=mean(cormat[upper.tri(cormat)]))
+ })  # end sapply
> stdcor <- t(stdcor)
```




```
> # Scatterplot of stock volatilities and correlations
> plot(x=stdcor[, "std"], y=stdcor[, "cor"],
+  xlab="volatility", ylab="correlation",
+  main="Monthly Stock Volatilities and Correlations")
> # Plot stock volatilities and correlations
> colnamev <- colnames(stdcor)
> stdcor <- xts(stdcor, zoo::index(retvti[endd]))
> dygraphs::dygraph(stdcor,
+   main="Monthly Stock Volatilities and Correlations") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
+   dyLegend(show="always", width=300)
```

Principal Component Analysis in Periods of High and Low Volatility

Stock returns in time intervals with *high volatility* have *high correlations* and therefore require fewer eigenvalues to explain 80% of their total variance.

Stock returns in time intervals with *low volatility* have *low correlations* and therefore require more eigenvalues to explain 80% of their total variance.

```
> # Calculate the median VTI volatility
> medianv <- median(stdcor[, "std"])
> # Calculate the stock returns of low volatility intervals
> retlow <- lapply(2:npts, function(endp) {
+   if (stdcor[endp-1, "std"] <= medianv)
+     retp[endd[endp-1]:endd[endp]]
+ })  # end lapply
> retlow <- rutils::do_call(rbind, retlow)
> # Calculate the stock returns of high volatility intervals
> rethigh <- lapply(2:npts, function(endp) {
+   if (stdcor[endp-1, "std"] > medianv)
+     retp[endd[endp-1]:endd[endp]]
+ })  # end lapply
> rethigh <- rutils::do_call(rbind, rethigh)
```

```
> # Calculate the correlations of low volatility intervals
> cormat <- cor(retlow, use="pairwise.complete.obs")
> cormat[is.na(cormat)] <- 0
> mean(cormat[upper.tri(cormat)])
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> sum(eigenval < 0)
> # Calculate the number of eigenvalues which sum up to at least 80%
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
> # Calculate the correlations of high volatility intervals
> cormat <- cor(rethigh, use="pairwise.complete.obs")
> cormat[is.na(cormat)] <- 0
> mean(cormat[upper.tri(cormat)])
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> eigenval <- eigend$values
> sum(eigenval < 0)
> # Calculate the number of eigenvalues which sum up to at least 80%
> which(cumsum(eigenval)/sum(eigenval) > 0.8)[1]
> # Calculate the condition number
> max(eigenval)/min(abs(eigenval))
```

Trailing Correlations of Stock Returns

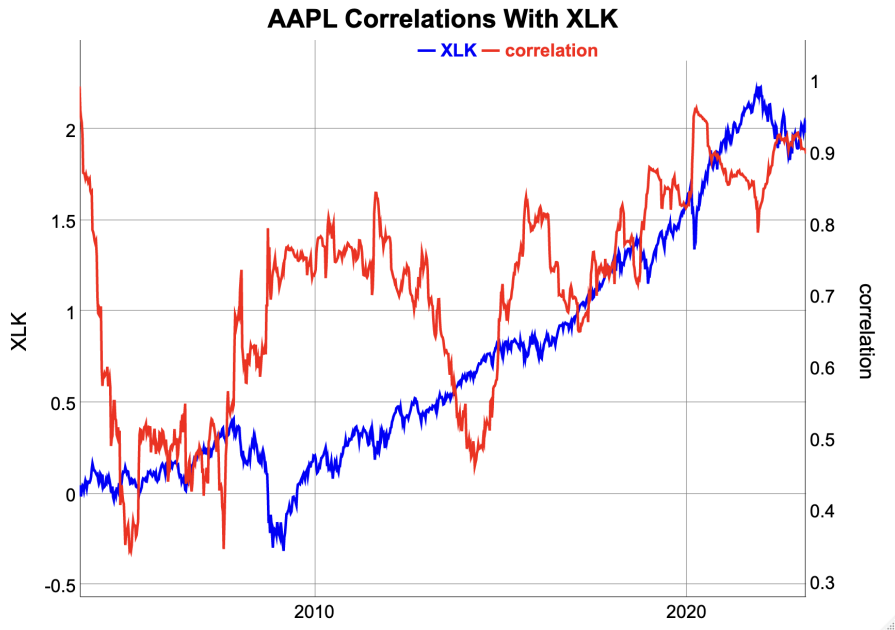
The trailing covariance can be updated using *online* recursive formulas with the weight decay factor λ:

$$\bar{x}_t = \lambda \bar{x}_{t-1} + (1-\lambda)x_t$$
$$\bar{y}_t = \lambda \bar{y}_{t-1} + (1-\lambda)y_t$$
$$\sigma^2_{xt} = \lambda \sigma^2_{x(t-1)} + (1-\lambda)(x_t - \bar{x}_t)^2$$
$$\sigma^2_{yt} = \lambda \sigma^2_{y(t-1)} + (1-\lambda)(y_t - \bar{y}_t)^2$$
$$\mathrm{cov}_t = \lambda\, \mathrm{cov}_{t-1} + (1-\lambda)(x_t - \bar{x}_t)(y_t - \bar{y}_t)$$

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 the trailing mean values have a stronger dependence on past returns. This is equivalent to a long look-back interval. 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*.

```
> # Calculate AAPL and XLK returns
> retp <- na.omit(cbind(returns$AAPL, rutils::etfenv$returns$XLK))
> # Calculate the trailing correlations
> lambda <- 0.99
> covarv <- HighFreq::run_covar(retp, lambda)
> correlv <- covarv[, 1, drop=FALSE]/sqrt(covarv[, 2]*covarv[, 3])
```



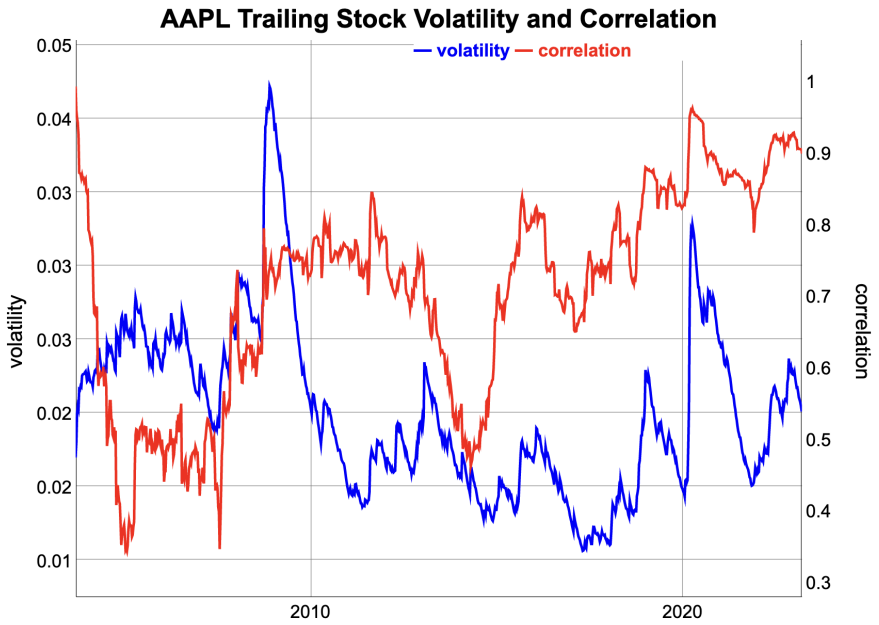

```
> # Plot dygraph of XLK returns and AAPL correlations
> datav <- cbind(cumsum(retp$XLK), correlv)
> colnames(datav)[2] <- "correlation"
> colnamev <- colnames(datav)
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(datav[endd], main="AAPL Correlations With XLK")
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
+   dyLegend(show="always", width=300)
```

Trailing Stock Correlations and Volatilities

The correlations of stock returns are typically higher in periods of higher volatility, and vice versa.

But stock correlations have increased after the 2008-09 financial crisis, while volatilities have decreased.

The correlation of *AAPL* and *XLK* has increased over time because *AAPL* has become a much larger component of *XLK*, as its stock has rallied.




```
> # Scatterplot of trailing stock volatilities and correlations
> volv <- sqrt(covarv[, 2])
> plot(x=volv[endd], y=correlv[endd, ], pch=1, col="blue",
+  xlab="AAPL volatility", ylab="Correlation",
+  main="Trailing Volatilities and Correlations of AAPL vs XLK")
> # Interactive scatterplot of trailing stock volatilities and corre
> datev <- zoo::index(retp[endd])
> datav <- data.frame(datev, volv[endd], correlv[endd, ])
> colnames(datav) <- c("date", "volatility", "correlation")
> library(plotly)
> plotly::plot_ly(data=datav, x=~volatility, y=~correlation,
+   type="scatter", mode="markers", text=datev) %>%
+   layout(title="Trailing Volatilities and Correlations of AAPL vs
```

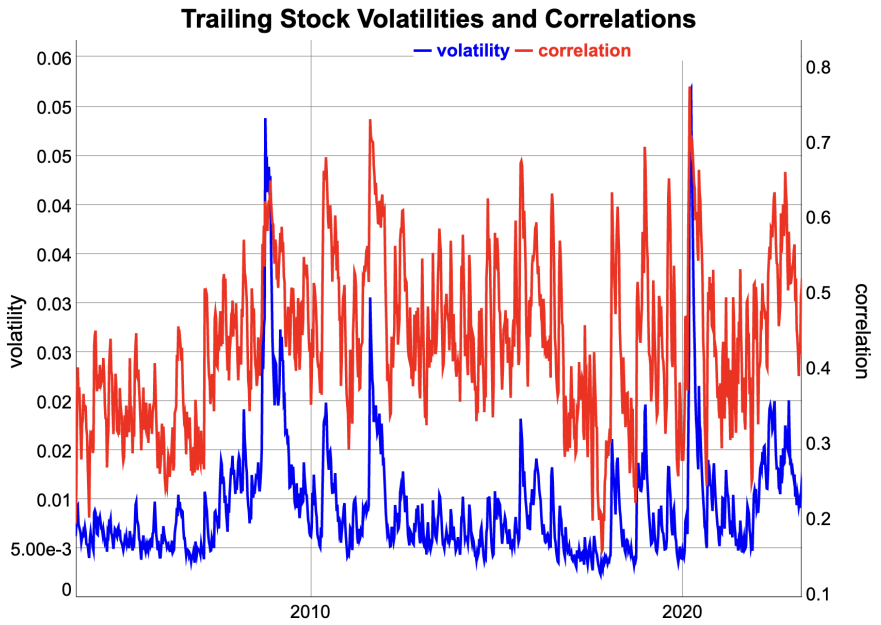
```
> # Plot trailing stock volatilities and correlations
> datav <- xts(cbind(volv, correlv), zoo::index(retp))
> colnames(datav) <- c("volatility", "correlation")
> colnamev <- colnames(datav)
> dygraphs::dygraph(datav[endd], main="AAPL Trailing Stock Volatilit
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
+   dyLegend(show="always", width=300)
```

Stock Portfolio Correlations and Volatilities

The average correlations of a stock portfolio are typically higher in periods of higher volatility, and vice versa.

But stock correlations have increased after the 2008-09 financial crisis, while volatilities have decreased.

```
> # Calculate portfolio returns
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> datev <- zoo::index(retvti)
> retp <- returns100
> retp[is.na(retp)] <- 0
> retp <- retp[datev]
> nrows <- NROW(retp)
> nstocks <- NCOL(retp)
> head(retp[, 1:5])
> # Calculate the average trailing portfolio correlations
> lambda <- 0.9
> correlv <- sapply(retp, function(retp) {
+   covarv <- HighFreq::run_covar(cbind(retvti, retp), lambda)
+   covarv[, 1, drop=FALSE]/sqrt(covarv[, 2]*covarv[, 3])
+ })  # end sapply
> correlv[is.na(correlv)] <- 0
> correlp <- rowMeans(correlv)
> # Scatterplot of trailing stock volatilities and correlations
> volvti <- sqrt(HighFreq::run_var(retvti, lambda))
> endd <- rutils::calc_endpoints(retvti, interval="weeks")
> plot(x=volvti[endd], y=correlp[endd],
+  xlab="volatility", ylab="correlation",
+  main="Trailing Stock Volatilities and Correlations")
```




```
> # Plot trailing stock volatilities and correlations
> datav <- xts(cbind(volvti, correlp), datev)
> colnames(datav) <- c("volatility", "correlation")
> colnamev <- colnames(datav)
> dygraphs::dygraph(datav[endd],
+   main="Trailing Stock Volatilities and Correlations") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeW
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], stroke
+   dyLegend(show="always", width=300)
```

Daily Mean Reverting Strategy

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

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

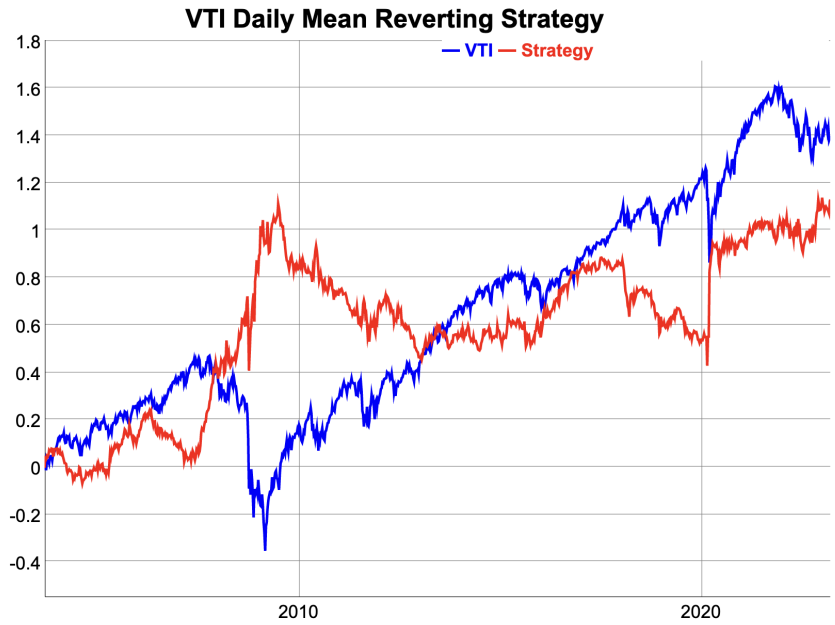
The function `rutils::plot_acf()` calculates and plots the autocorrelations of a time series.

Daily stock returns often exhibit some negative autocorrelations.

The daily mean reverting strategy buys or sells short \$1 of stock at the end of each day (depending on the sign of the previous daily return), and holds the position until the next day.

If the previous daily return was positive, it sells short \$1 of stock. If the previous daily return was negative, it buys \$1 of stock.

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



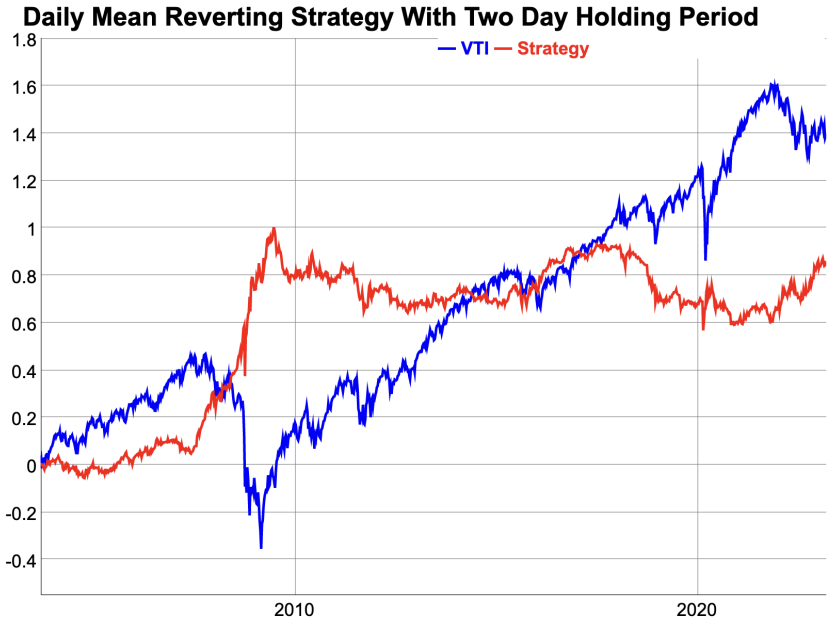

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv,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

Daily Mean Reverting Strategy With a Holding Period

The daily mean reverting strategy can be improved by combining the daily returns from the previous two days. This is equivalent to holding the position for two days, instead of rolling it daily.

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

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



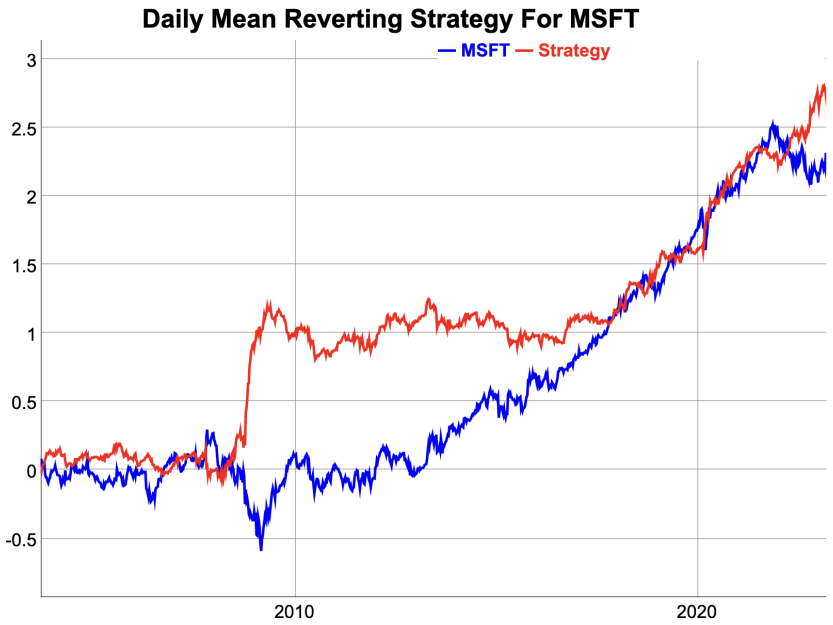

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv,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

Daily Mean Reverting Strategy For Stocks

Some daily stock returns exhibit stronger negative autocorrelations than ETFs.

But the daily mean reverting strategy doesn't perform well for many stocks.

```
> # Load daily S&P500 stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> retp <- na.omit(returns$MSFT)
> rutils::plot_acf(retp)
> # Simulate mean reverting strategy with two day holding period
> posv <- rutils::lagit(rutils::roll_sum(sign(retp), look_back=2))/2
> pnls <- (-retp*posv)
```

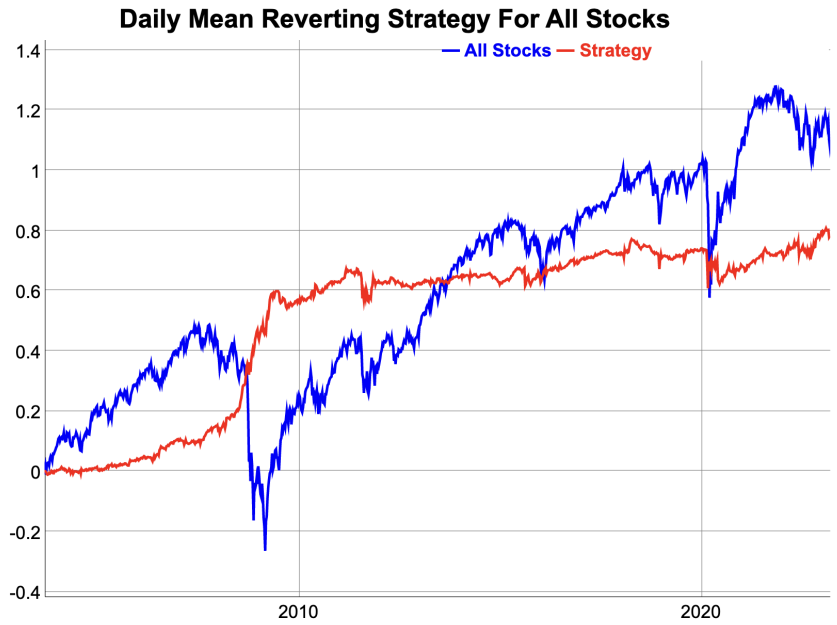



```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv,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

Daily Mean Reverting Strategy For All Stocks

The combined daily mean reverting strategy for all *S&P500* stocks performed well prior to and during the 2008 financial crisis, but was flat afterwards.

```
> # Calculate the average returns of all S&P500 stocks
> datev <- zoo::index(returns)
> retp <- returns
> retp[is.na(retp)] <- 0
> retp <- rowMeans(retp)
> # Simulate mean reverting strategy for all S&P500 stocks
> pnls <- lapply(returns, function(retp) {
+   retp <- na.omit(retp)
+   posv <- rutils::roll_sum(sign(retp), look_back=2)/2
+   posv <- rutils::lagit(posv)
+   pnls <- (-retp*posv)
+   pnls
+ }) # end lapply
> pnls <- do.call(cbind, pnls)
> pnls[is.na(pnls)] <- 0
> pnls <- rowMeans(pnls)
```




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

Autoregressive Model of Stock Returns

The stock returns r_t can be modeled using an *autoregressive* process *AR(n)* with a constant intercept term φ_0:

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

The autoregressive model can be written in matrix form as:

$$\mathbf{r} = \varphi \mathbb{P}$$

Where $\varphi = \{\varphi_0, \varphi_1, \varphi_2, \ldots \varphi_n\}$ is the vector of autoregressive coefficients.

The *response* is equal to the returns $\mathbf{r}$, and the columns of the *predictor matrix* $\mathbb{P}$ are equal to the lags of the returns.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> nrows <- NROW(retp)
> respv <- retp
> # Define the response and predictor matrices
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- paste0("pred", 1:NCOL(predm))
```

Forecasting Stock Returns Using Autoregressive Models

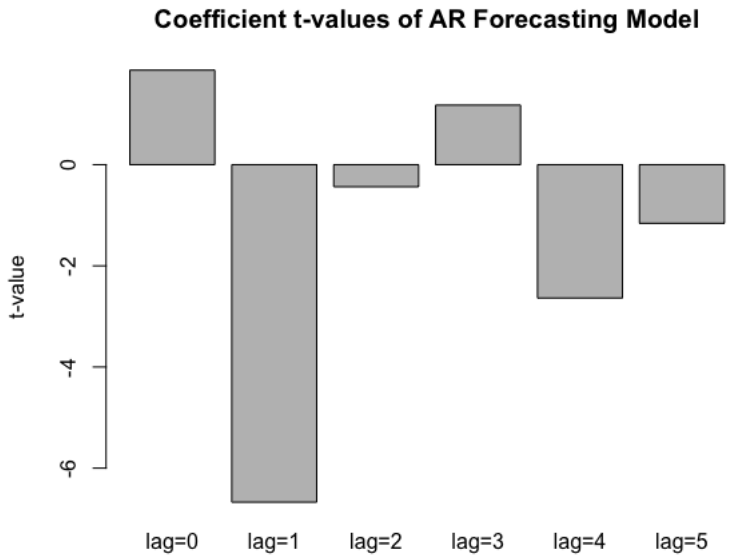
The fitted autoregressive coefficients φ are equal to the *response* $\mathbf{r}$ multiplied by the inverse of the *predictor matrix* $\mathbb{P}$:

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

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

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

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




```
> # Calculate the fitted autoregressive coefficients
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcast <- predm %*% coeff
> range(fcast)
> # Calculate the residuals (forecast errors)
> resids <- (fcast - retp)
> # The residuals are orthogonal to the forecasts
> cor(resids, fcast)
> # Calculate the variance of the residuals
> vares <- sum(resids^2)/(nrows-NROW(coeff))
> # Calculate the predictor matrix squared
> predm2 <- crossprod(predm)
> # Calculate the covariance matrix of the AR coefficients
> covar <- vares*MASS::ginv(predm2)
> coeffsd <- sqrt(diag(covar))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coeffsd)
```

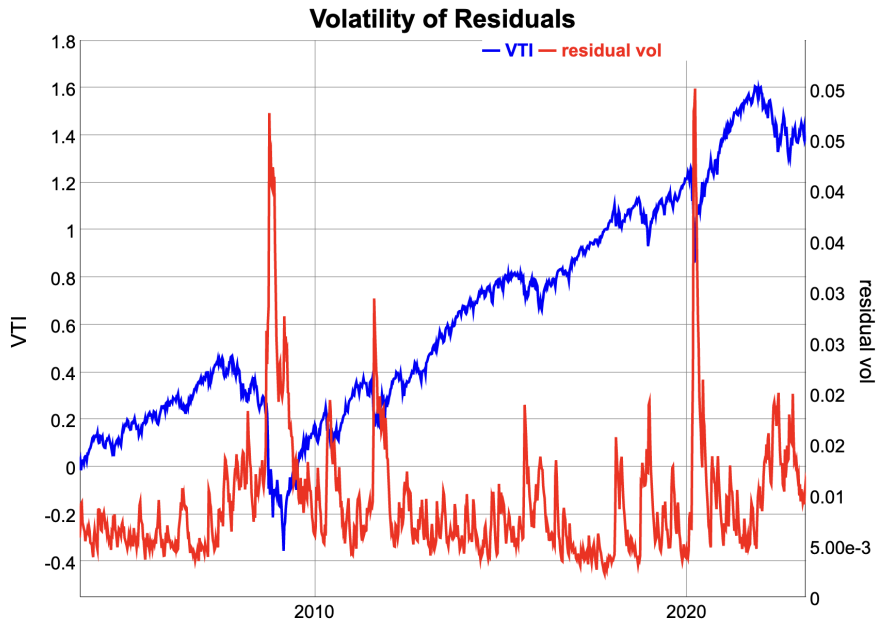
```
> # Plot the t-values of the AR coefficients
> lagv <- paste0("lag=", 0:5)
> barplot(coefft ~ lagv, xlab="", ylab="t-value",
+   main="Coefficient t-values of AR Forecasting Model")
```

Residuals of Autoregressive Forecasting Model

The autoregressive model assumes stationary returns and residuals, with similar volatility over time.

In reality stock volatility is highly time dependent, so the volatility of the residuals is also time dependent.

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




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

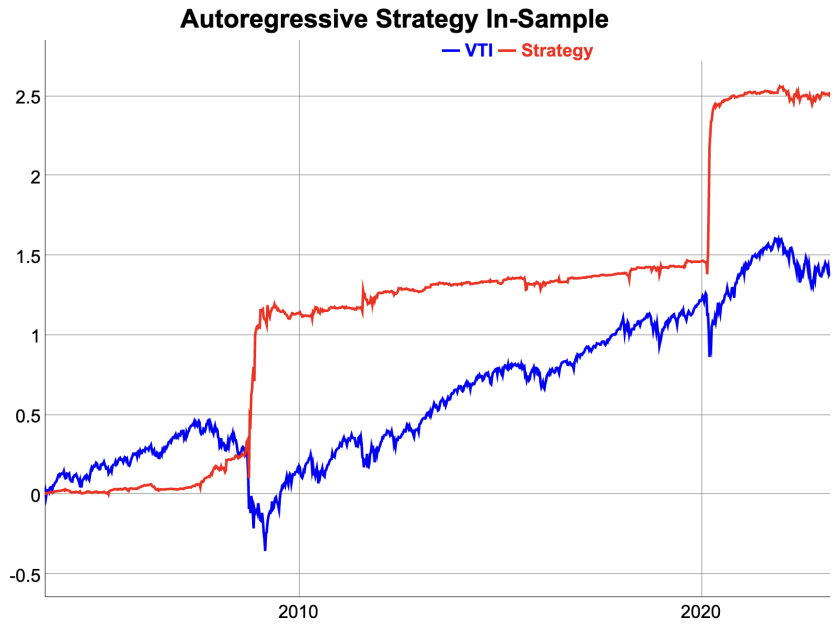
Autoregressive Strategy In-Sample

The first step in strategy development is optimizing it in-sample, even though in practice it can't be implemented. Because a strategy can't perform well out-of-sample if it doesn't perform well in-sample.

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

The in-sample autoregressive strategy performs well during periods of high volatility, but not as well in low volatility periods.

The dollar allocations of *VTI* stock are too large in periods of high volatility, which causes over-leverage and very high risk.




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

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

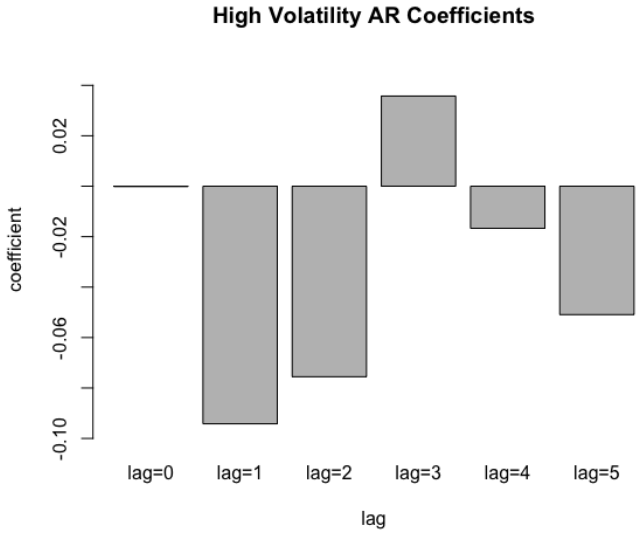
Autoregressive Coefficients in Periods of High and Low Volatility

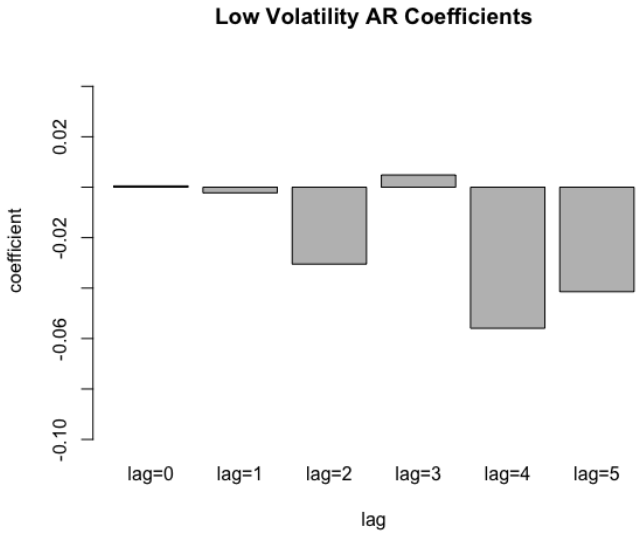
The autoregressive model assumes stationary returns and residuals, with similar volatility over time. In reality stock volatility is highly time dependent.

The autoregressive coefficients in periods of high volatility are very different from those under low volatility.

In periods of high volatility, there are larger negative autocorrelations than in low volatility.

```
> # Calculate the high volatility AR coefficients
> respv <- retp["2008/2011"]
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> predinv <- MASS::ginv(predm)
> coeffh <- drop(predinv %*% respv)
> lagv <- paste0("lag=", 0:5)
> barplot(coeffh ~ lagv, main="High Volatility AR Coefficients",
+   xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
> # Calculate the low volatility AR coefficients
> respv <- retp["2012/2019"]
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> predinv <- MASS::ginv(predm)
> coeffl <- drop(predinv %*% respv)
> barplot(coeffl ~ lagv, main="Low Volatility AR Coefficients",
+   xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
```





The *Winsor* Function

Some models produce very large dollar allocations, leading to large portfolio leverage (dollars invested divided by the capital).

The *winsor function* maps the *model weight* w into the dollar amount for investment. The hyperbolic tangent function can serve as a winsor function:

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

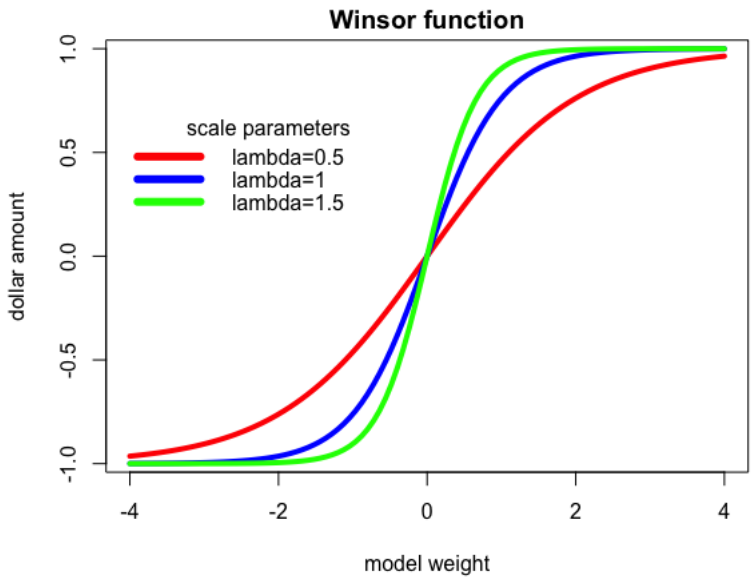
Where λ is the scale parameter.

The hyperbolic tangent is close to linear for small values of the *model weight* w, and saturates to $+1\$/-1\$$ for very large positive and negative values of the *model weight*.

The saturation effect limits (caps) the leverage in the strategy to $+1\$/-1\$$.

For very small values of the scale parameter λ, the invested dollar amount is linear for a wide range of *model weights*. So the strategy is mostly invested in dollar amounts proportional to the *model weights*.

For very large values of the scale parameter λ, the invested dollar amount jumps from $-1\$$ for negative *model weights* to $+1\$$ for positive *model weight* values. So the strategy is invested in either $-1\$$ or $+1\$$ dollar amounts.



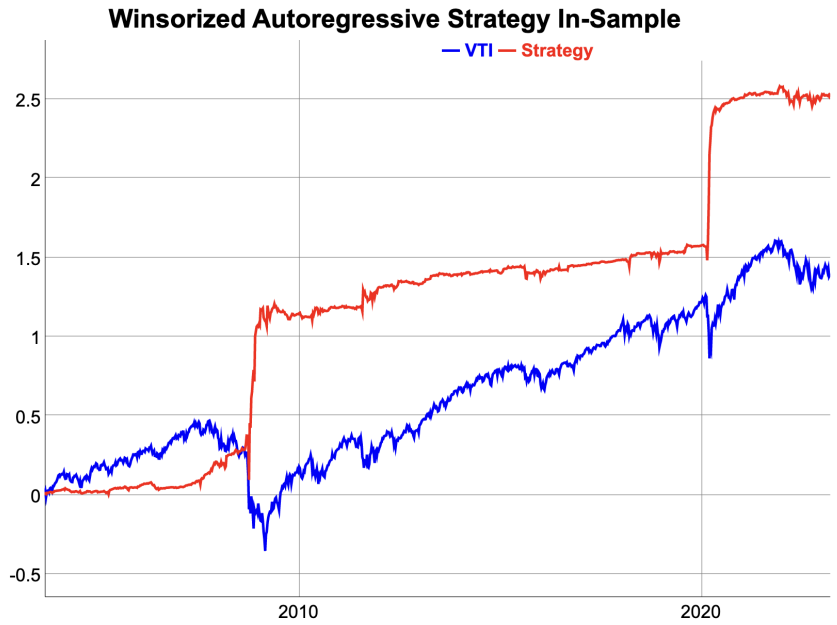

```
> lambdav <- c(0.5, 1, 1.5)
> colorv <- c("red", "blue", "green")
> # Define the winsor function
> winsorfun <- function(retp, lambda) tanh(lambda*retp)
> # Plot three curves in loop
> for (indeks in 1:3) {
+   curve(expr=winsorfun(x, lambda=lambdav[indeks]),
+ xlim=c(-4, 4), type="l", lwd=4,
+ xlab="model weight", ylab="dollar amount",
+ col=colorv[indeks], add=(indeks>1))
+ }  # end for
> # Add title and legend
> title(main="Winsor function", line=0.5)
> legend("topleft", title="scale parameters\n",
+    paste("lambda", lambdav, sep="="), inset=0.0, cex=1.0,
+    lwd=6, bty="n", y.intersp=0.3, lty=1, col=colorv)
```

Winsorized Autoregressive Strategy

The performance of the autoregressive strategy can be improved by fitting its coefficients using the *winsorized returns*, to reduce the effect of time-dependent volatility.

The performance can also be improved by *winsorizing* the forecasts, by reducing the leverage due to very large forecasts.

```
> # Winsorize the VTI returns
> retw <- winsorfun(retp/0.01, lambda=0.1)
> # Define the response and predictor matrices
> predm <- lapply(1:orderp, rutils::lagit, input=retw)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- paste0("pred", 1:NCOL(predm))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% retw
> # Calculate the in-sample forecasts of VTI
> fcast <- predm %*% coeff
> # Winsorize the forecasts
> # fcast <- winsorfun(fcast/mad(fcast), lambda=1.5)
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcast
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp)/sd(pnls)
```



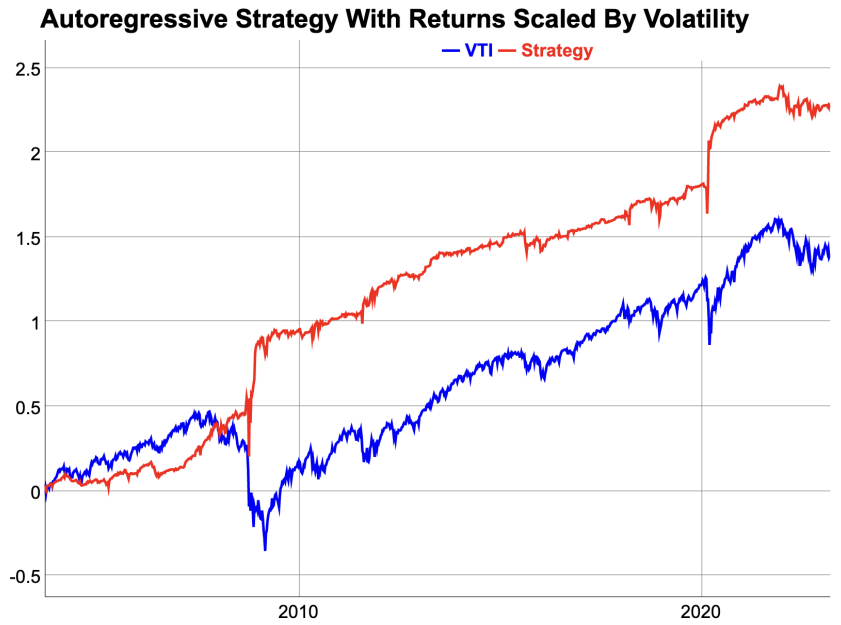

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of autoregressive strategy
> endd <- rutils::calc_endpoints(wealthv,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

Autoregressive Strategy With Returns Scaled By Volatility

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns divided by their trailing volatility.

Dividing the returns by their trailing volatility reduces the effect of time-dependent volatility.

```
> # Scale the returns by their standard deviation
> varv <- HighFreq::run_var(retp, lambda=0.99)
> retsc <- ifelse(varv > 0, retp/sqrt(varv), 0)
> # Calculate the AR coefficients
> predm <- lapply(1:orderp, rutils::lagit, input=retsc)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- paste0("pred", 1:NCOL(predm))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% retsc
> # Calculate the in-sample forecasts of VTI
> fcast <- predm %*% coeff
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcast
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp)/sd(pnls)
```



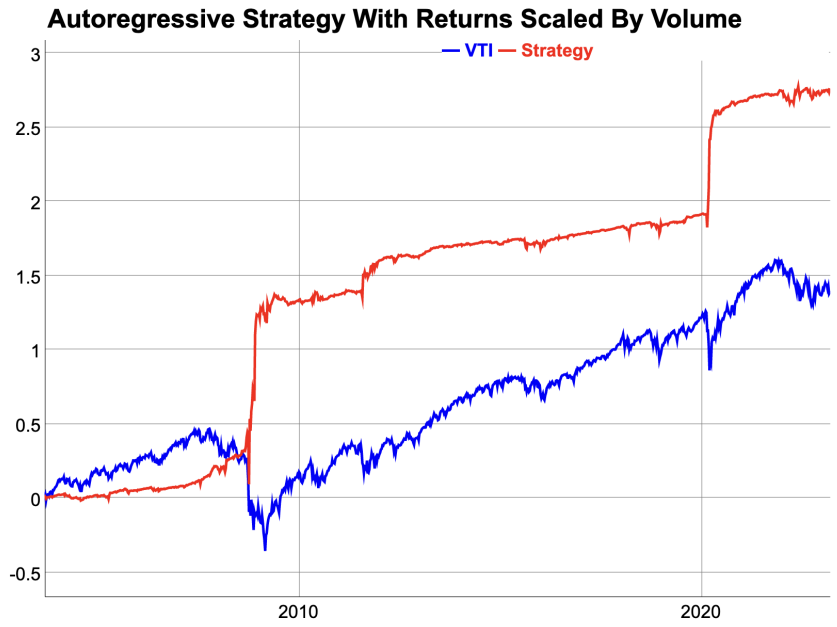

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy With Returns Scaled By Volatility"
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Autoregressive Strategy in Trading Time

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns divided by the trading volumes.

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

```
> # Calculate VTI returns and trading volumes
> ohlc <- rutils::etfenv$VTI
> datev <- zoo::index(ohlc)
> nrows <- NROW(ohlc)
> closep <- quantmod::Cl(ohlc)
> retp <- rutils::diffit(log(closep))
> volumv <- quantmod::Vo(ohlc)
> # Calculate trailing average volume
> volumr <- HighFreq::run_mean(volumv, lambda=0.7)
> # Scale the returns using volume clock to trading time
> retsc <- ifelse(volumv > 0, volumr*retp/volumv, 0)
> # Calculate the AR coefficients
> respv <- retsc
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- paste0("pred", 1:NCOL(predm))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> # Calculate the in-sample forecasts of VTI
> fcast <- predm %*% coeff
> # Simulate autoregressive strategy in-sample
> pnls <- retp*fcast
> pnls <- pnls*sd(retp)/sd(pnls)
```




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

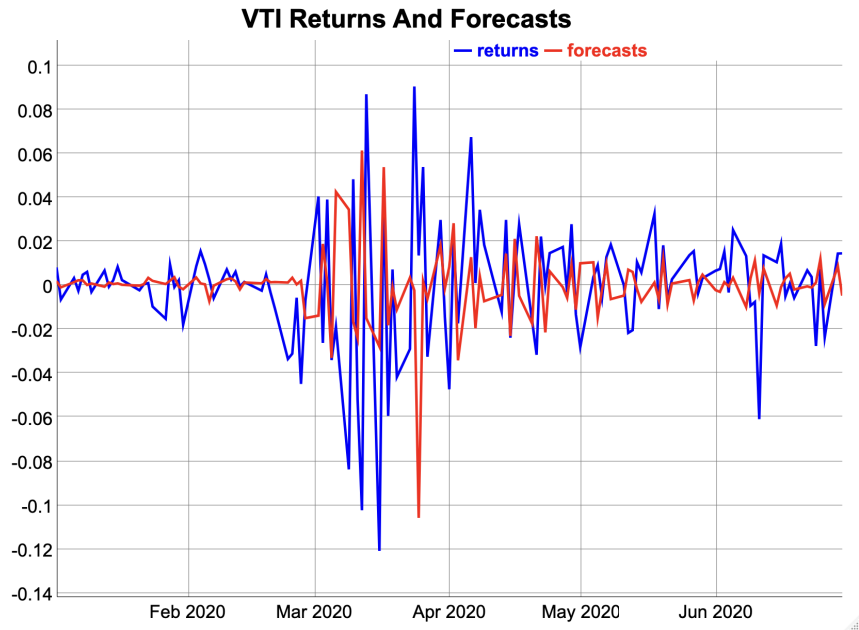
Mean Squared Error of the Autoregressive Forecasting Model

The accuracy of a forecasting model can be measured using the *mean squared error* and the *correlation*.

The mean squared error (*MSE*) of a forecasting model is the average of the squared forecasting errors ε_i, equal to the differences between the *forecasts* f_t minus the actual values r_t: $\varepsilon_i = f_t - r_t$:

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

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




```
> # Plot the forecasts
> datav <- cbind(retp, fcasts)["2020-01/2020-06"]
> colnames(datav) <- c("returns", "forecasts")
> dygraphs::dygraph(datav,
+   main="VTI Returns And Forecasts") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

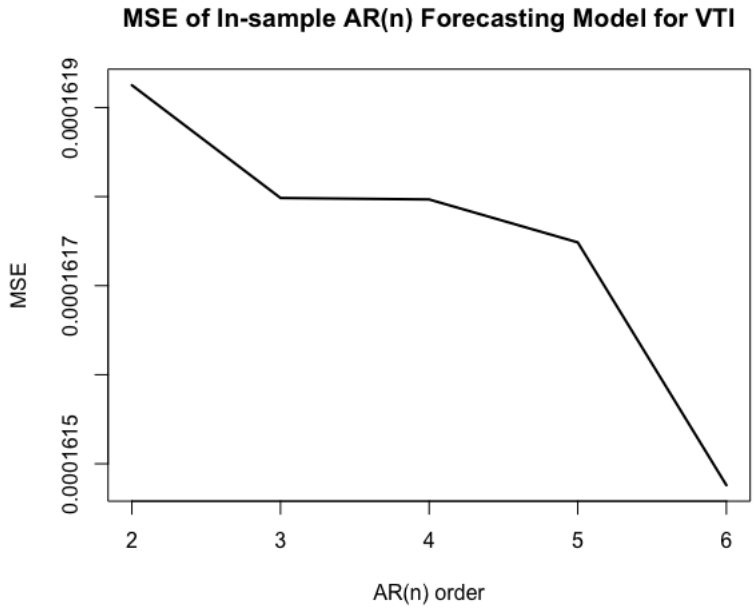
In-sample Order Selection of Autoregressive Forecasting

The mean squared errors (*MSE*) of the *in-sample* forecasts decrease steadily with the increasing order parameter *n* of the *AR(n)* forecasting model.

In-sample forecasting consists of first fitting an *AR(n)* model to the data, and calculating its coefficients.

The *in-sample* forecasts are calculated by multiplying the predictor matrix by the fitted AR coefficients.

```
> # Calculate the forecasts as function of the AR order
> fcasts <- lapply(2:NCOL(predm), function(ordern) {
+   # Calculate the fitted AR coefficients
+   predinv <- MASS::ginv(predm[, 1:ordern])
+   coeff <- predinv %*% respv
+   # Calculate the in-sample forecasts of VTI
+   drop(predm[, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcasts) <- paste0("n=", 2:NCOL(predm))
```




```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((respv - x)^2), cor=cor(respv, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- names(fcasts)
> # Plot forecasting MSE
> plot(x=2:NCOL(predm), y=mse[, 1],
+   xlab="AR(n) order", ylab="MSE", type="l", lwd=2,
+   main="MSE of In-sample AR(n) Forecasting Model for VTI")
```

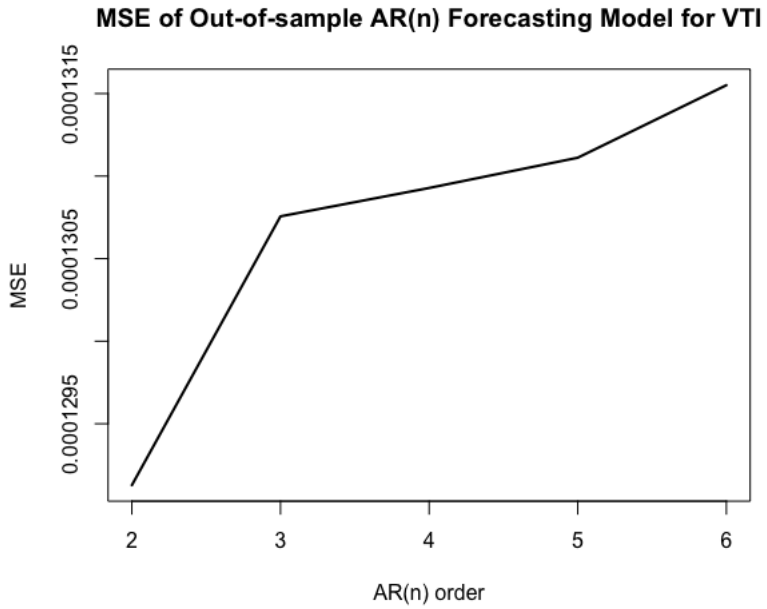
Out-of-Sample Forecasting Using Autoregressive Models

The mean squared errors (*MSE*) of the *out-of-sample* forecasts increase with the increasing order parameter *n* of the *AR(n)* model.

The reason for the increasing out-of-sample MSE is the *overfitting* of the coefficients to the training data for larger order parameters.

Out-of-sample forecasting consists of first fitting an *AR(n)* model to the training data, and calculating its coefficients.

The *out-of-sample* forecasts are calculated by multiplying the *out-of-sample* predictor matrix by the fitted AR coefficients.




```
> # Define in-sample and out-of-sample intervals
> nrows <- NROW(retp)
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> # Calculate the forecasts as function of the AR order
> fcasts <- lapply(2:NCOL(predm), function(ordern) {
+   # Calculate the fitted AR coefficients
+   predinv <- MASS::ginv(predm[insample, 1:ordern])
+   coeff <- predinv %*% respv[insample]
+   # Calculate the out-of-sample forecasts of VTI
+   drop(predm[outsample, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcasts) <- paste0("n=", 2:NCOL(predm))
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((respv[outsample] - x)^2), cor=cor(respv[outsample],
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- names(fcasts)
> # Plot forecasting MSE
> plot(x=2:NCOL(predm), y=mse[, 1],
+   xlab="AR(n) order", ylab="MSE", type="l", lwd=2,
+   main="MSE of Out-of-sample AR(n) Forecasting Model for VTI")
```

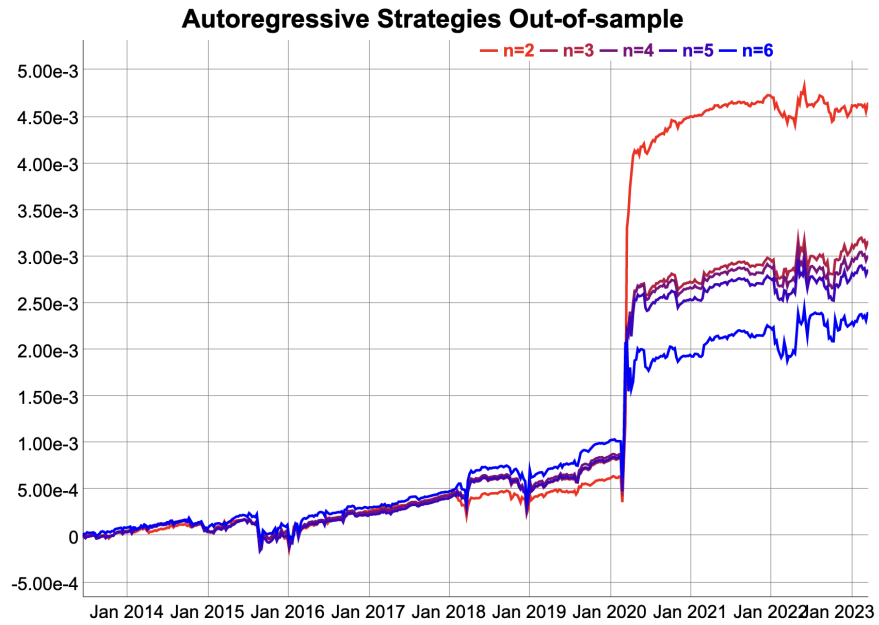
Out-of-Sample Autoregressive Strategy

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

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

The optimal order parameter is equal to 2, with a positive intercept coefficient φ_0 (since the average *VTI* returns were positive), and a negative coefficient φ_1 (because of strong negative autocorrelations in periods of high volatility).

Decreasing the order parameter of the autoregressive model is a form of *shrinkage* because it reduces the number of predictive variables.




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

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

Rolling Autoregressive Forecasting Model

The autoregressive coefficients can be calibrated dynamically over a *rolling* look-back interval, and applied to calculating the *out-of-sample* forecasts.

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

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

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

Backtesting Function for the Forecasting Model

The *meta-parameters* of the *backtesting* function are the order *n* of the *AR(n)* model and the length of the look-back interval (`look_back`).

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

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

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

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

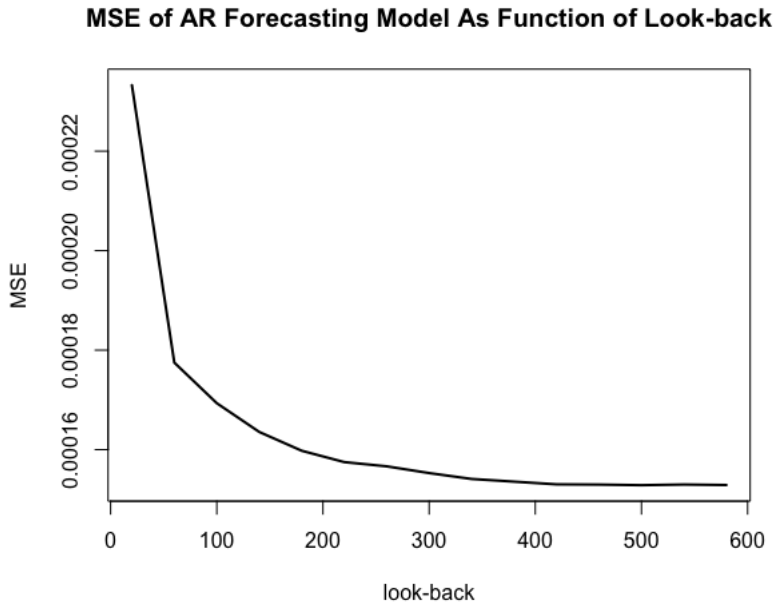
Forecasting Dependence On the Look-back Interval

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

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

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`look_back`).

The accuracy of the forecasting model increases with longer look-back intervals (`look_back`), because more data improves the estimates of the autoregressive coefficients.




```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> cluster <- makeCluster(ncores)
> # Perform parallel loop under Windows
> look_backs <- seq(20, 600, 40)
> fcasts <- parLapply(cluster, look_backs, sim_fcasts, ordern=6)
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(look_backs, sim_fcasts, ordern=6, mc.cores=nc
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- look_backs
> # Select optimal look_back interval
> look_back <- look_backs[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=look_backs, y=mse[, 1],
+   xlab="look-back", ylab="MSE", type="l", lwd=2,
+   main="MSE of AR Forecasting Model As Function of Look-back")
```

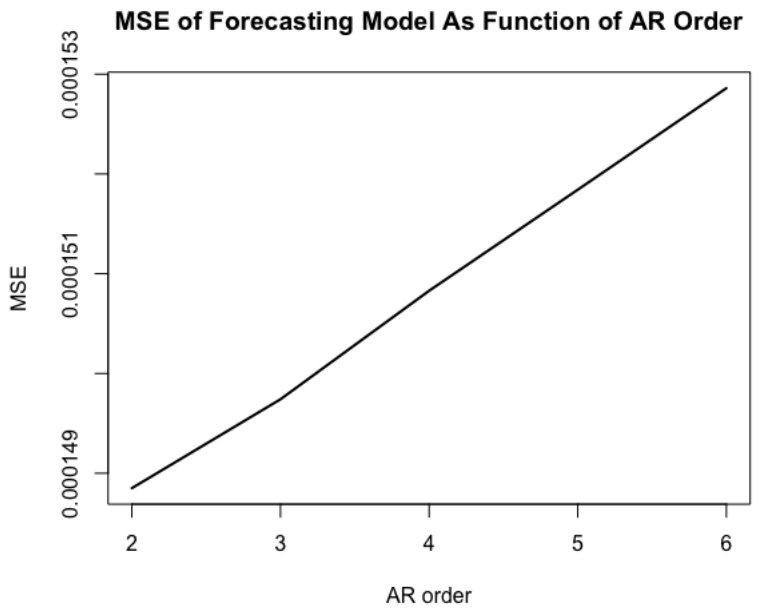
Order Dependence With Fixed Look-back

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

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

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`look_back`).

The accuracy of the forecasting model decreases for larger AR order parameters, because of overfitting in-sample.




```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> cluster <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(cluster, orderv, sim_fcasts,
+   look_back=look_back)
> stopCluster(cluster)  # Stop R processes over cluster under Windo
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   look_back=look_back, mc.cores=ncores)
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, y=mse[, 1],
+   xlab="AR order", ylab="MSE", type="l", lwd=2,
+   main="MSE of Forecasting Model As Function of AR Order")
```

Autoregressive Strategy With Fixed Look-back

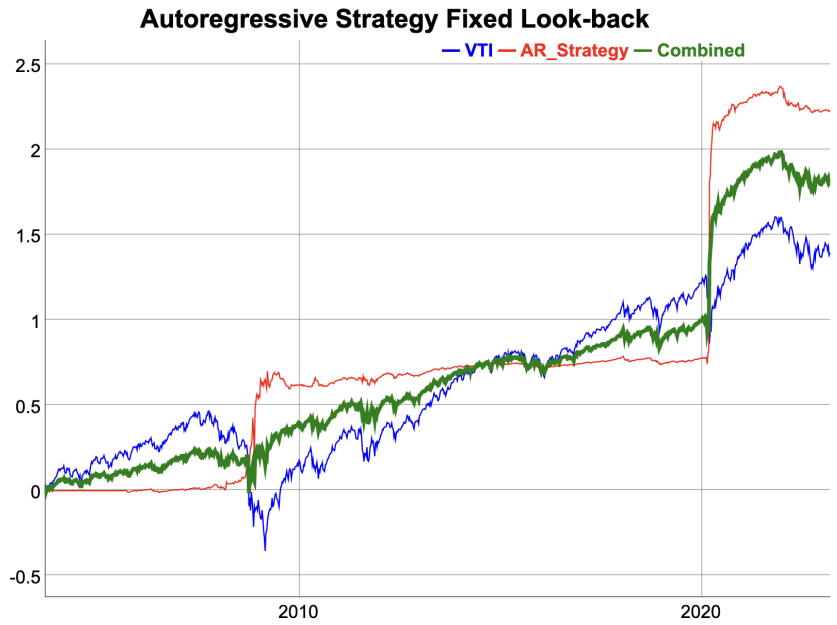
The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

Using the return forecasts as portfolio weights produces very large weights in periods of high volatility, and creates excessive risk.

To reduce excessive risk, a binary strategy can be used, with portfolio weights equal to the sign of the forecasts.

```
> # Simulate the rolling autoregressive forecasts
> fcasts <- sim_fcasts(look_back=look_back, ordern=ordern)
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp)/sd(pnls)
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```




```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy Fixed Look-back") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Order Dependence With Expanding Look-back

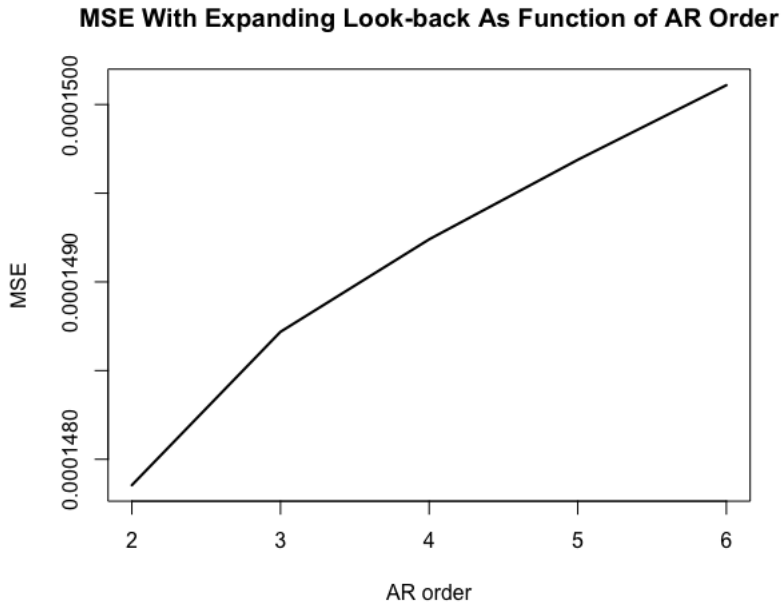
The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

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

Longer look-back intervals (`look_back`) are usually better for the autoregressive forecasting model.

The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> cluster <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(cluster, orderv, sim_fcasts,
+   look_back=look_back, fixedlb=FALSE)
> stopCluster(cluster)  # Stop R processes over cluster under Windo
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   look_back=look_back, fixedlb=FALSE, mc.cores=ncores)
```



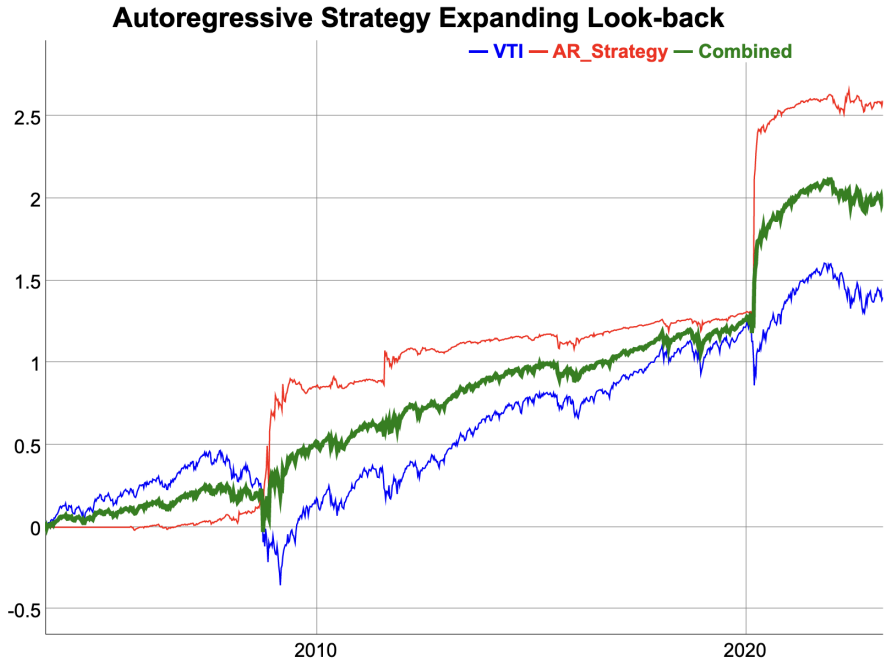

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, y=mse[, 1],
+   xlab="AR order", ylab="MSE", type="l", lwd=2,
+   main="MSE With Expanding Look-back As Function of AR Order")
```

Autoregressive Strategy With Expanding Look-back

The model with an *expanding* look-back interval has better performance compared to the *fixed* look-back interval.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

```
> # Simulate the autoregressive forecasts with expanding look-back
> fcasts <- sim_fcasts(look_back=look_back, ordern=ordern, fixedlb=F
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> pnls <- pnls*sd(retp)/sd(pnls)
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```




```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy Expanding Look-back") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Portfolio Weight Constraints

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

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

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

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

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

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

```
> # Load daily S&P500 percentage stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> # Overwrite NA values in returns
> retp <- returns100
> retp[is.na(retp)] <- 0
> # Remove stocks with very little data
> datev <- zoo::index(retp) # dates
> nrows <- NROW(retp) # number of rows
> nzeros <- colSums(retp == 0)
> sum(nzeros > nrows/2)
> retp <- retp[, nzeros < nrows/2]
> nstocks <- NCOL(retp) # number of stocks
> # Objective function equal to Kelly ratio
> objfun <- function(retp) {
+   varv <- var(retp)
+   if (varv > 0) mean(retp)/varv else 0
+ }  # end objfun
> # Calculate performance statistics for all stocks
> perfstat <- sapply(retp, objfun)
> perfstat[!is.finite(perfstat)] <- 0
> sum(is.na(perfstat))
> sum(!is.finite(perfstat))
> # Calculate weights proportional to performance statistic
> weightm <- perfstat
> hist(weightm)
> # Center the weights
> weightv <- weightm - mean(weightm)
> sum(weightv)
> sort(weightv)
```

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 allocations.

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

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

Linear Weight Constraint

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

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

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

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

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

Volatility Weight Constraint

The weights can be scaled to satisfy a volatility target.

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

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

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

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

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

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

```
> # Calculate in-sample portfolio volatility
> volis <- sd(drop(retp %*% weightm))
> # Calculate equal weight portfolio volatility
> volew <- sd(rowMeans(retp))
> # Apply the volatility constraint
> weightv <- volew*weightm/volis
> sqrt(var(drop(retp %*% weightv)))
> # Apply the volatility target constraint
> volt <- 1e-2
> weightv <- volt*weightm/volis
> sqrt(var(drop(retp %*% weightv)))
```

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.

```
> # Objective function equal to sum of returns
> objfun <- function(retp) sum(retp)
> # Objective function equal to Sharpe ratio
> objfun <- function(retp) mean(retp)/max(sd(retp), 1e-4)
> # Objective function equal to Kelly ratio
> objfun <- function(retp) {
+   varv <- var(retp)
+   if (varv > 0) mean(retp)/varv else 0
+ }  # end objfun
> # Calculate performance statistics for all stocks
> perfstat <- sapply(retp, objfun)
> perfstat[!is.finite(perfstat)] <- 0
> sum(is.na(perfstat))
> # Calculate the best and worst performing stocks
> perfstat <- sort(perfstat, decreasing=TRUE)
> topstocks <- 10
> symbolb <- names(head(perfstat, topstocks))
> symbolw <- names(tail(perfstat, topstocks))
> # Calculate equal weights for the best and worst performing stocks
> weightv <- numeric(NCOL(retp))
> names(weightv) <- colnames(retp)
> weightv[symbolb] <- 1
> weightv[symbolw] <- (-1)
> # Calculate weights proportional to performance statistic
> weightv <- perfstat
> # Center weights so sum is equal to 0
> weightv <- weightv - mean(weightv)
> # Scale weights so sum of squares is equal to 1
> weightv <- weightv/sqrt(sum(weightv^2))
> # Calculate the strategy returns
> retportf <- retp %*% weightv
> # Scale weights so in-sample portfolio volatility is same as equal
> scalev <- sd(rowMeans(retp))/sd(retportf)
> weightv <- scalev*weightv
```

Rolling Momentum Strategy

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

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

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

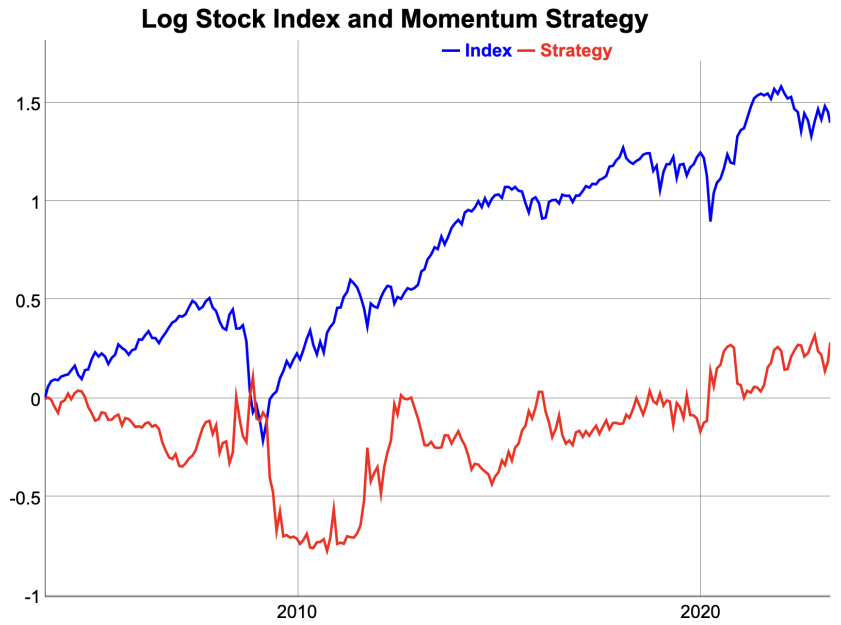
```
> # Objective function equal to Kelly ratio
> objfun <- function(retp) {
+   varv <- var(retp)
+   if (varv > 0) mean(retp)/varv else 0
+ }  # end objfun
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> npts <- NROW(endd)
> # Perform loop over the end points
> look_back <- 8
> pnls <- lapply(2:(npts-1), function(ep) {
+   # Select the look-back returns
+   startp <- endd[max(1, ep-look_back)]
+   retsis <- retp[startp:endd[ep], ]
+   # Calculate the best and worst performing stocks in-sample
+   perfstat <- sapply(retsis, objfun)
+   perfstat[!is.finite(perfstat)] <- 0
+   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 portfolio returns
+   retportf <- retsis %*% weightv
+   # Scale weights so in-sample portfolio volatility is same as equ
+   weightv <- weightv*sd(rowMeans(retsis))/sd(retportf)
+   # Calculate the out-of-sample momentum returns
+   drop(retp[(endd[ep]+1):endd[ep+1], ] %*% weightv)
+ })  # 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
> indeks <- rowMeans(retp)
> indeks <- xts::xts(indeks, order.by=datev)
> colnames(indeks) <- "Index"
> # Add initial startup interval to the momentum returns
> pnls <- c(rowMeans(retp[endd[1]:endd[2], ]), pnls)
> pnls <- xts::xts(pnls, order.by=datev)
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




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

Momentum Strategy Functional

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

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

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

```
> btmomtop <- function(retp, objfun, look_back=12, rebalf="months",
+   bid_offer=0.0, endd=rutils::calc_endpoints(retp, interval=rebali
+   # Perform loop over end points
+   npts <- NROW(endd)
+   pnls <- lapply(2:(npts-1), function(ep) {
+     # Select the look-back returns
+     startp <- endd[max(1, ep-look_back)]
+     retsis <- retp[startp:endd[ep], ]
+     # Calculate the best and worst performing stocks in-sample
+     perfstat <- sapply(retsis, objfun)
+     perfstat[!is.finite(perfstat)] <- 0
+     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 portfolio returns
+     retportf <- retsis %*% weightv
+     # Scale weights so in-sample portfolio volatility is same as e
+     weightv <- weightv*sd(rowMeans(retsis))/sd(retportf)
+     # Calculate the out-of-sample momentum returns
+     drop(retp[(endd[ep]+1):endd[ep+1], ] %*% weightv)
+   })  # end lapply
+   pnls <- rutils::do_call(c, pnls)
+   pnls
+ }  # end btmomtop
```

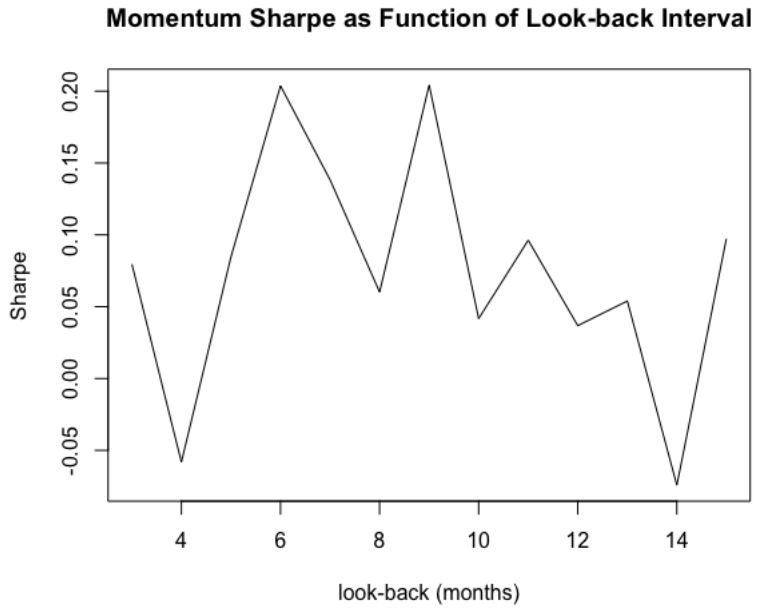
Optimization of Momentum Strategy Parameters

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

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

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

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




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

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

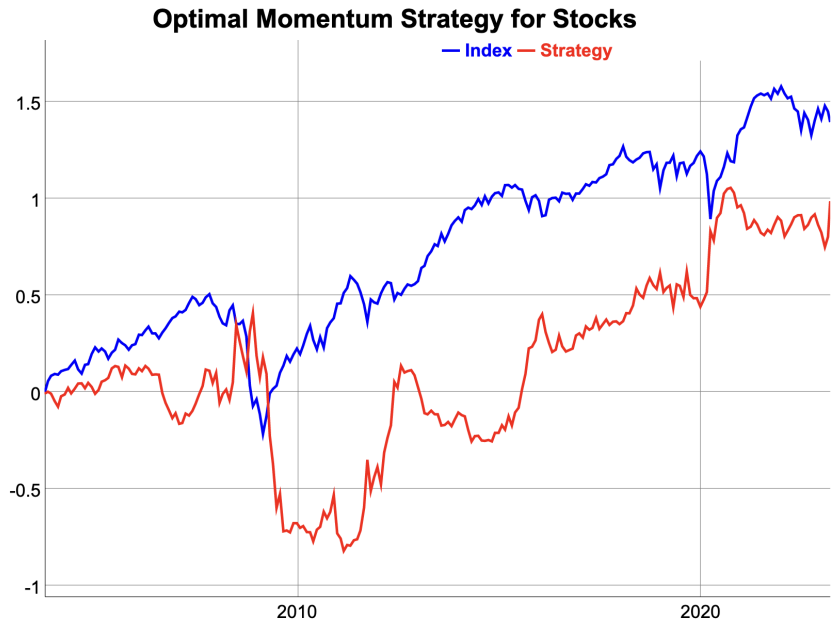
Optimal Stock Momentum Strategy

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

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

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

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




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

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

Weighted Momentum Strategy Functional

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

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

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

```
> btmomweight <- function(retp, objfun, look_back=12, rebalf="months
+   bid_offer=0.0, endd=rutils::calc_endpoints(retp, interval=rebalf
+   # Perform loop over end points
+   npts <- NROW(endd)
+   pnls <- lapply(2:(npts-1), function(ep) {
+     # Select the look-back returns
+     startp <- endd[max(1, ep-look_back)]
+     retsis <- retp[startp:endd[ep], ]
+     # Calculate weights proportional to performance
+     perfstat <- sapply(retsis, objfun)
+     perfstat[!is.finite(perfstat)] <- 0
+     weightv <- perfstat
+     # Calculate the in-sample portfolio returns
+     retportf <- retsis %*% weightv
+     # Scale weights so in-sample portfolio volatility is same as e
+     weightv <- weightv*sd(rowMeans(retsis))/sd(retportf)
+     # Calculate the out-of-sample momentum returns
+     retp[(endd[ep]+1):endd[ep+1], ] %*% weightv
+   })  # end lapply
+   rutils::do_call(c, pnls)
+ }  # end btmomweight
```

Optimal Weighted Stock Momentum Strategy

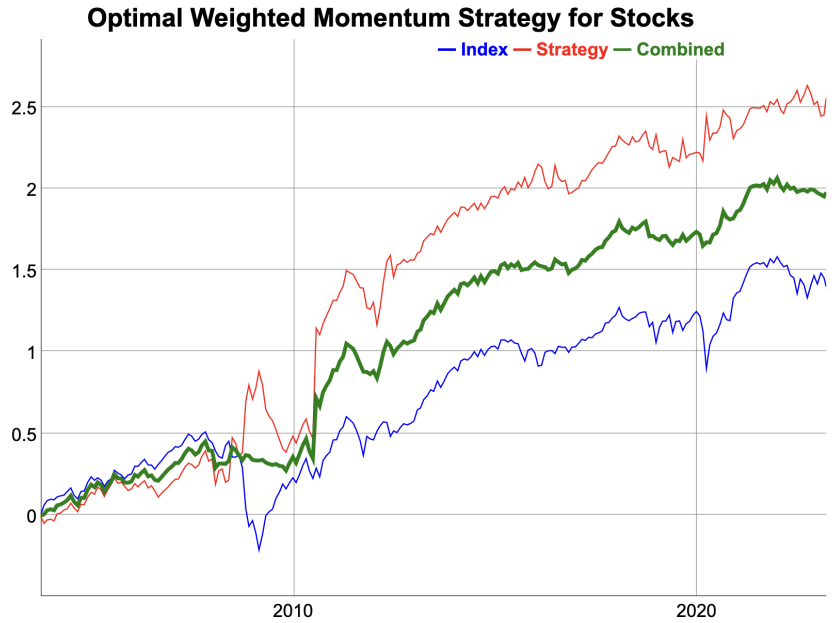
The stock momentum strategy produces a similar absolute return as the index, and also a similar Sharpe ratio.

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

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

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

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




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

```
> # Calculate best pnls of momentum strategy
> whichmax <- which.max(sharper)
> look_backs[whichmax]
> pnls <- pnll[[whichmax]]
> pnls <- c(rowMeans(retp[endd[1]:endd[2], ]), pnls)
> pnls <- xts::xts(pnls, order.by=datev)
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls, 0.5*(indeks+pnls))
> colnames(wealthv)[3] <- "Combined"
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of stock index and momentum strategy
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Weighted Momentum Strategy for Stocks") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
```

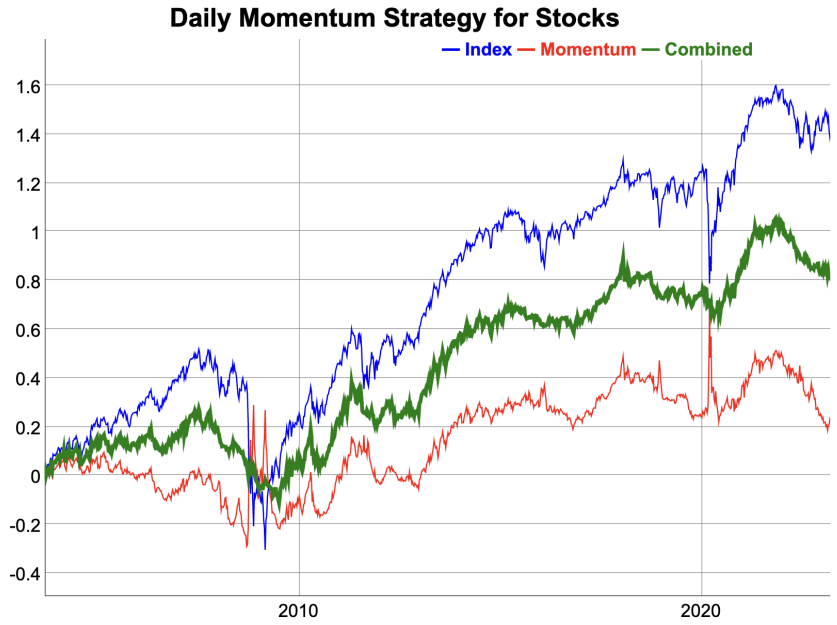
Momentum Strategy With Daily Rebalancing

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

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

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

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




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

```
> # Scale the momentum volatility to the equal weight index
> indeksd <- sd(indeks)
> pnls <- indeksd*pnls/sd(pnls)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls, 0.5*(indeks+pnls))
> colnames(wealthv)[2:3] <- c("Momentum", "Combined")
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of stock index and momentum strategy
> colorv <- c("blue", "red", "green")
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Daily Momentum Strategy Functional

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

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

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

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

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

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

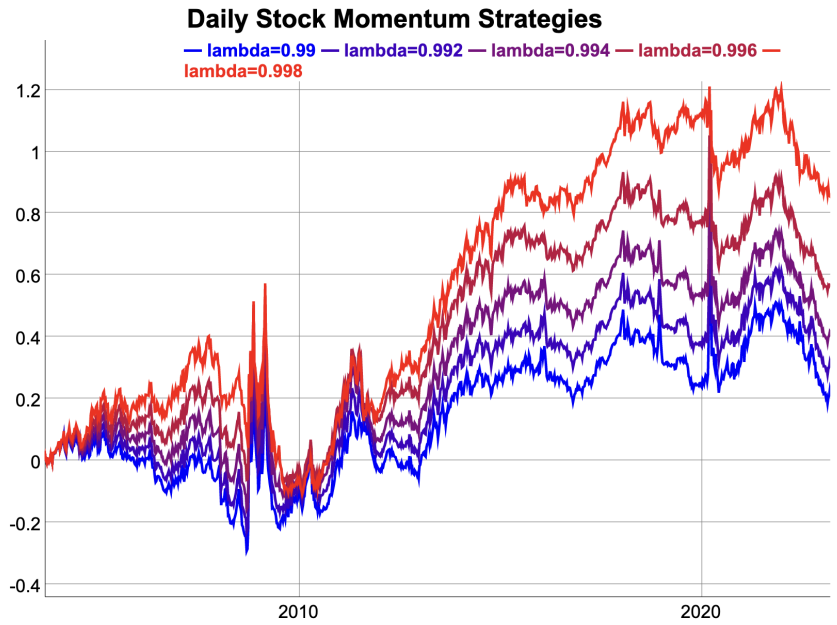
Multiple Daily Stock Momentum Strategies

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

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

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

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




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

Daily Momentum Strategy with Holding Period

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

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

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

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

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

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

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

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

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

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

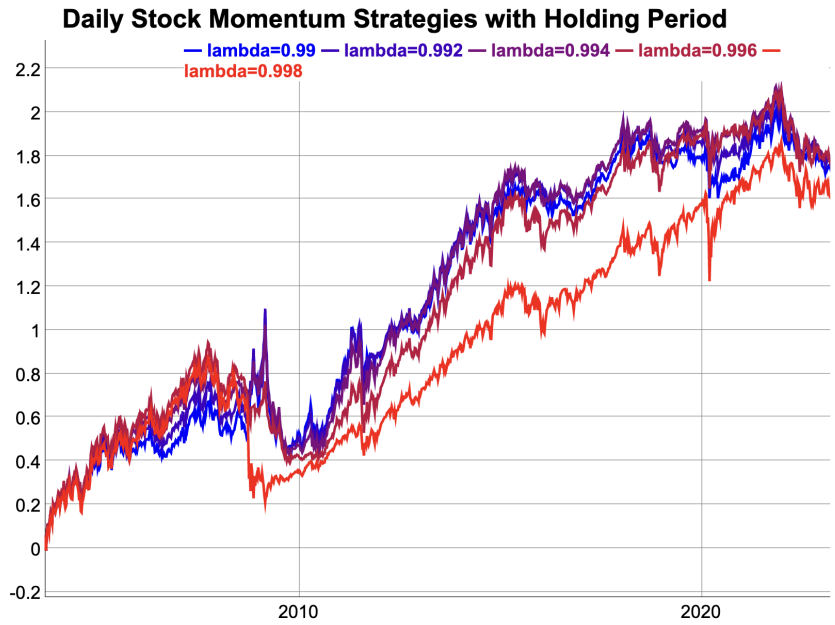
Multiple Daily Momentum Strategies With Holding Period

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

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

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

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



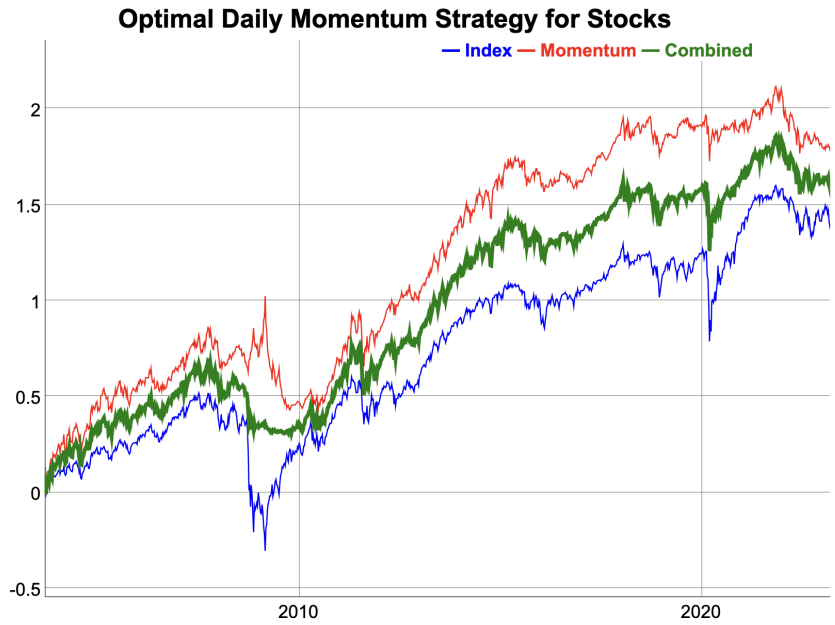

```
> # dygraph of daily stock momentum strategies with holding period
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily Stock Momentum Strategies with Holding Period") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot of daily stock momentum strategies with holding period
> plot_theme <- chart_theme()
> plot_theme$col$line.col <-
+   colorRampPalette(c("blue", "red"))(NCOL(pnls))
> quantmod::chart_Series(cumsum(pnls)[endd],
+   theme=plot_theme, name="Daily Stock Momentum Strategies with 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)
> lambdas[whichmax]
> pnls <- pnls[, whichmax]
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls, 0.5*(indeks+pnls))
> colnames(wealthv)[2:3] <- c("Momentum", "Combined")
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot dygraph of stock index and momentum strategy
> colorv <- c("blue", "red", "green")
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting Stock Momentum Strategies

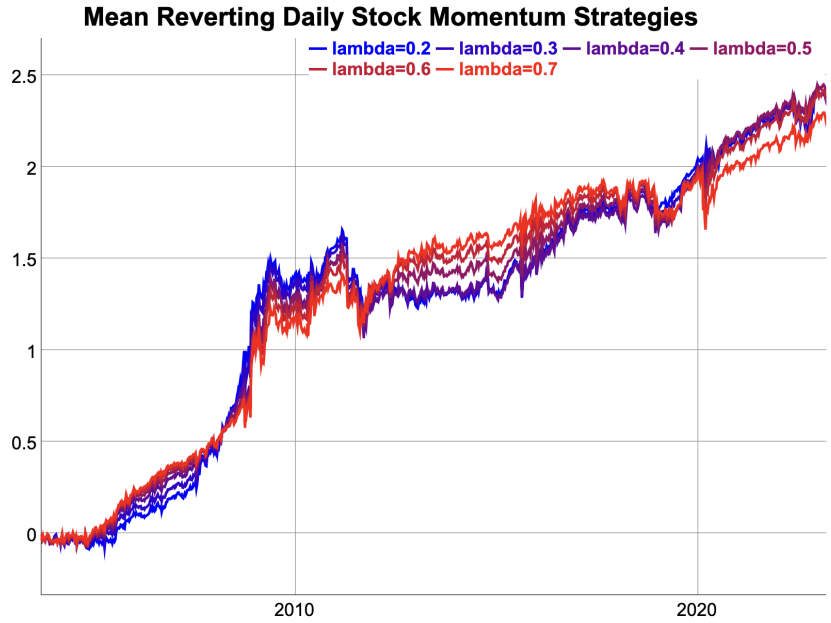
Multiple *mean reverting* stock momentum strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of λ parameters.

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

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

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

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




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

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

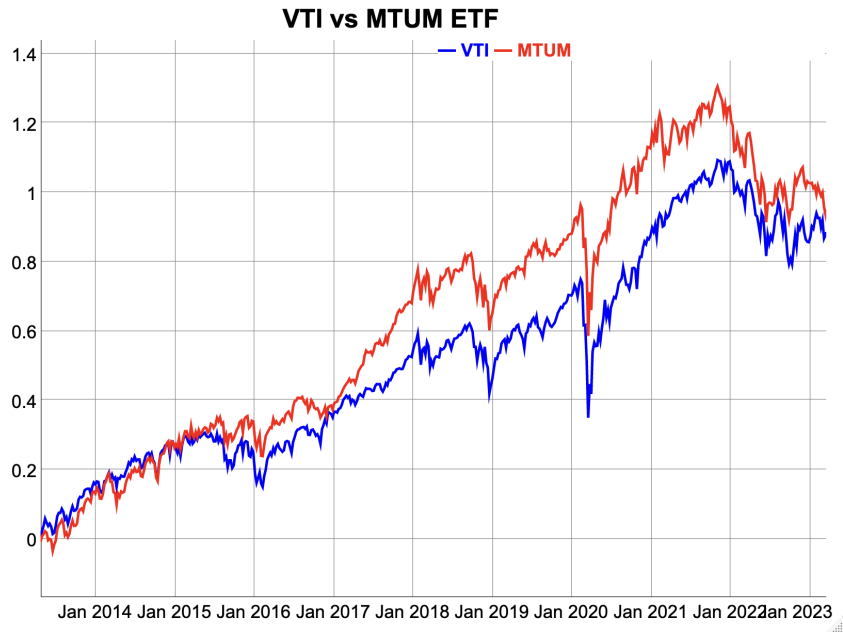
The MTUM Momentum ETF

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

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

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

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




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

Momentum Weights for PCA Portfolios

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

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

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

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

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

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

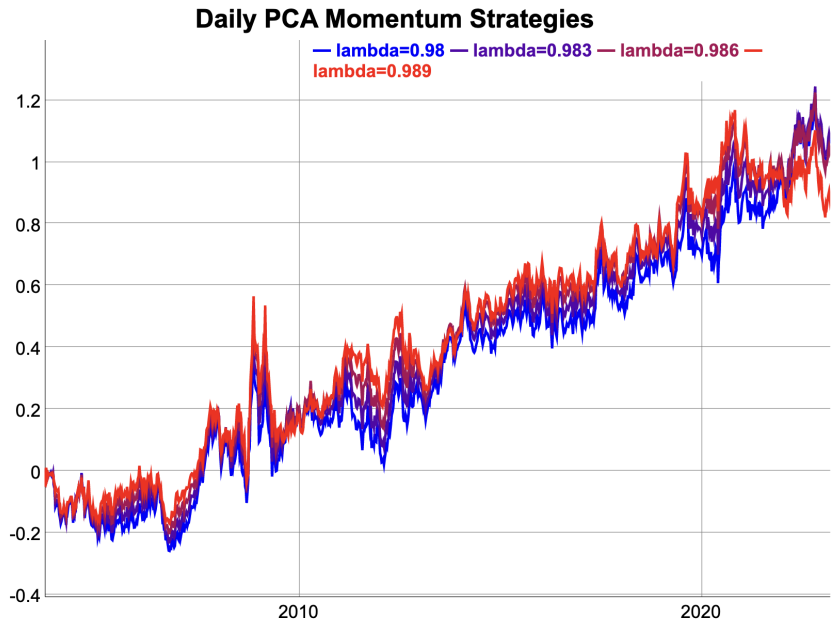
```
> # Calculate the PCA weights for standardized returns
> retsc <- lapply(retp, function(x) (x - mean(x))/sd(x))
> retsc <- do.call(cbind, retsc)
> covmat <- cov(retsc)
> pcad <- eigen(covmat)
> pcaw <- pcad$vectors
> rownames(pcaw) <- colnames(retp)
> sort(-pcaw[, 1], decreasing=TRUE)
> sort(pcaw[,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

Momentum Strategy for PCA Portfolios

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

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

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




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

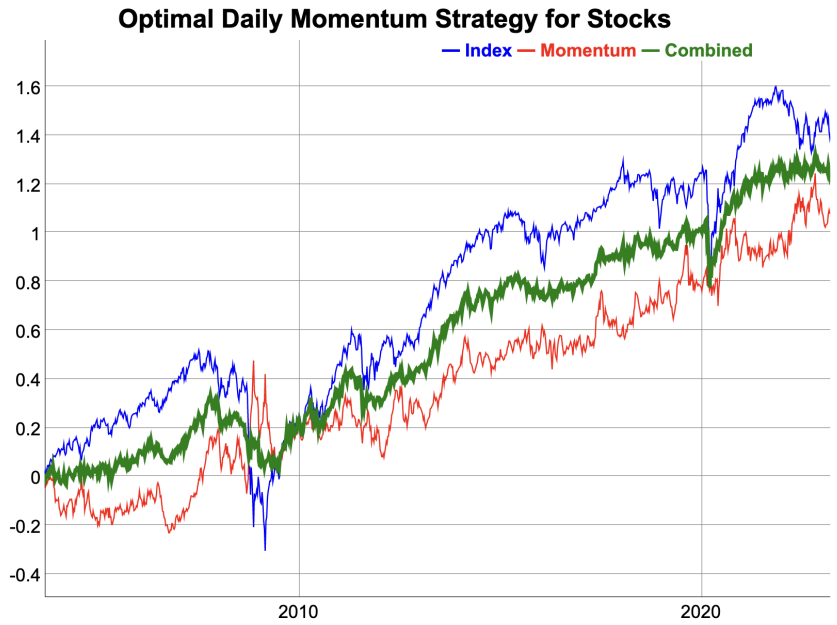
Optimal PCA Momentum Strategy

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

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

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

```
> # Calculate best pnls of PCA momentum strategy
> whichmax <- which.max(sharper)
> lambdas[whichmax]
> pnls <- pnls[, whichmax]
> colnames(pnls) <- "Strategy"
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls, 0.5*(indeks+pnls))
> colnames(wealthv)[2:3] <- c("Momentum", "Combined")
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot dygraph of stock index and PCA momentum strategy
> colorv <- c("blue", "red", "green")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting PCA Momentum Strategies

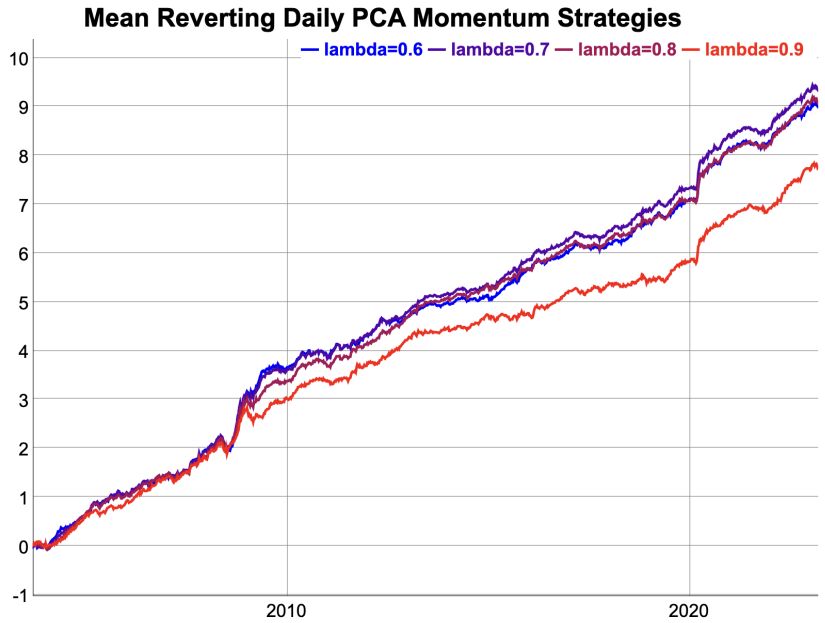
The *mean reverting* momentum strategy performs well for the higher order principal components.

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

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

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

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



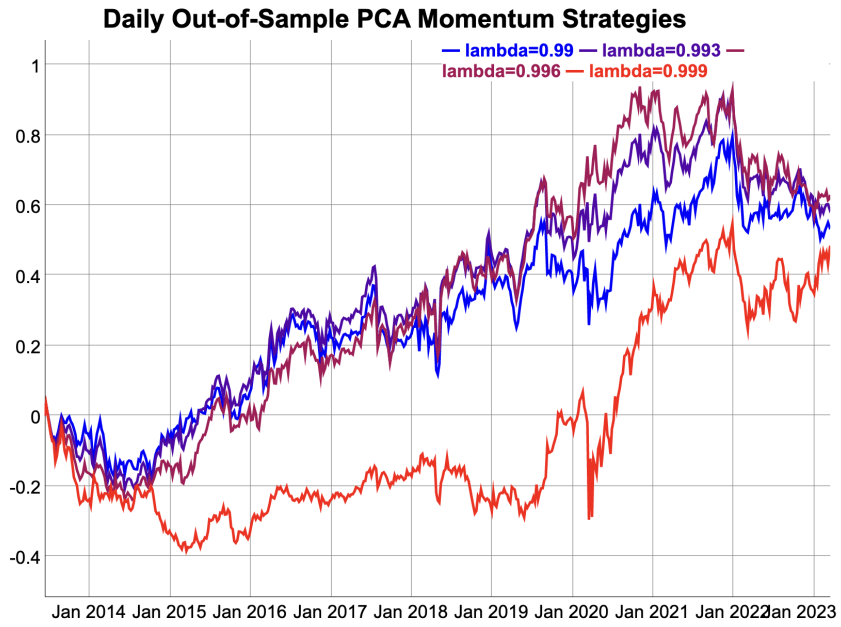

```
> # Plot dygraph of daily PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Mean Reverting Daily PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=400)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

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

PCA Momentum Strategy Out-of-Sample

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

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




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

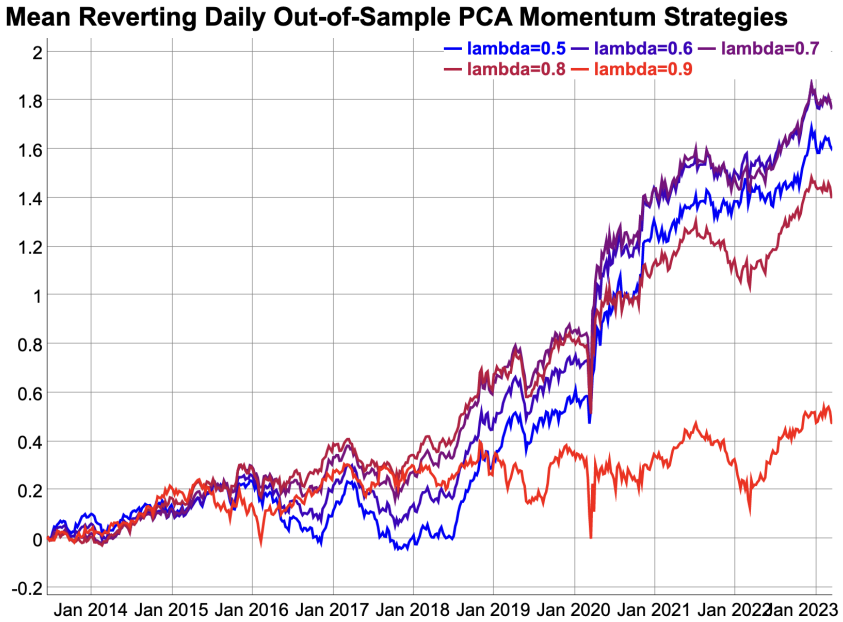
```
> # Calculate a vector of weekly end points
> endd <- rutils::calc_endpoints(retpca, interval="weeks")
> # Plot dygraph of daily out-of-sample PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily Out-of-Sample PCA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

Mean Reverting PCA Momentum Strategy Out-of-Sample

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

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

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




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

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

Momentum Strategy for an *ETF* Portfolio

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

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

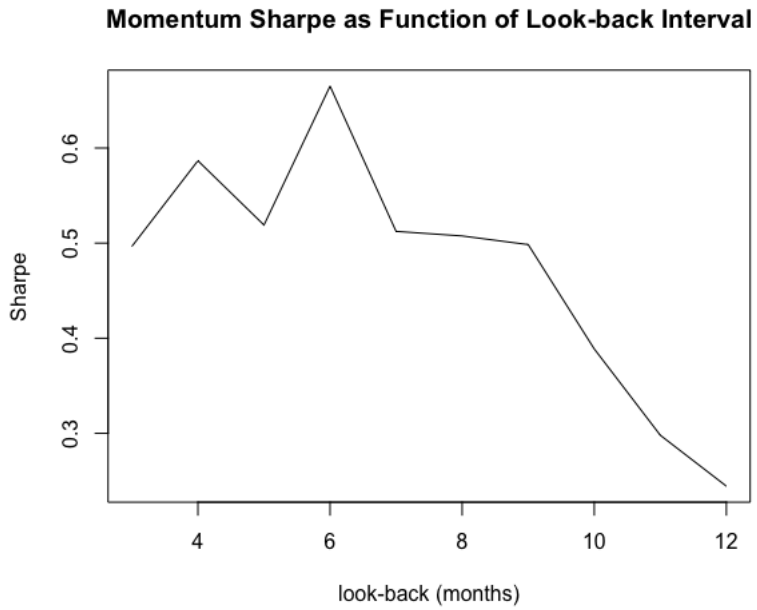
The dependence on the length of the *look-back interval* is an example of the *bias-variance tradeoff*. If the *look-back interval* is too short, the past performance estimates have high *variance*, but if the *look-back interval* is too long, the past estimates have high *bias*.

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

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

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

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




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

Performance of Momentum Strategy for ETFs

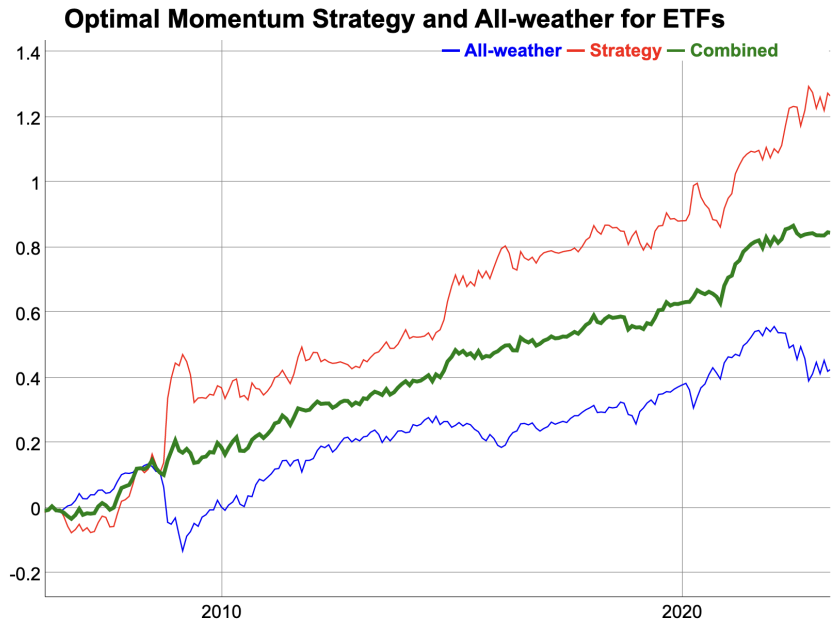
The momentum strategy for ETFs produces a higher absolute return and also a higher Sharpe ratio than the static *All-Weather* portfolio.

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

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

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

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




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

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

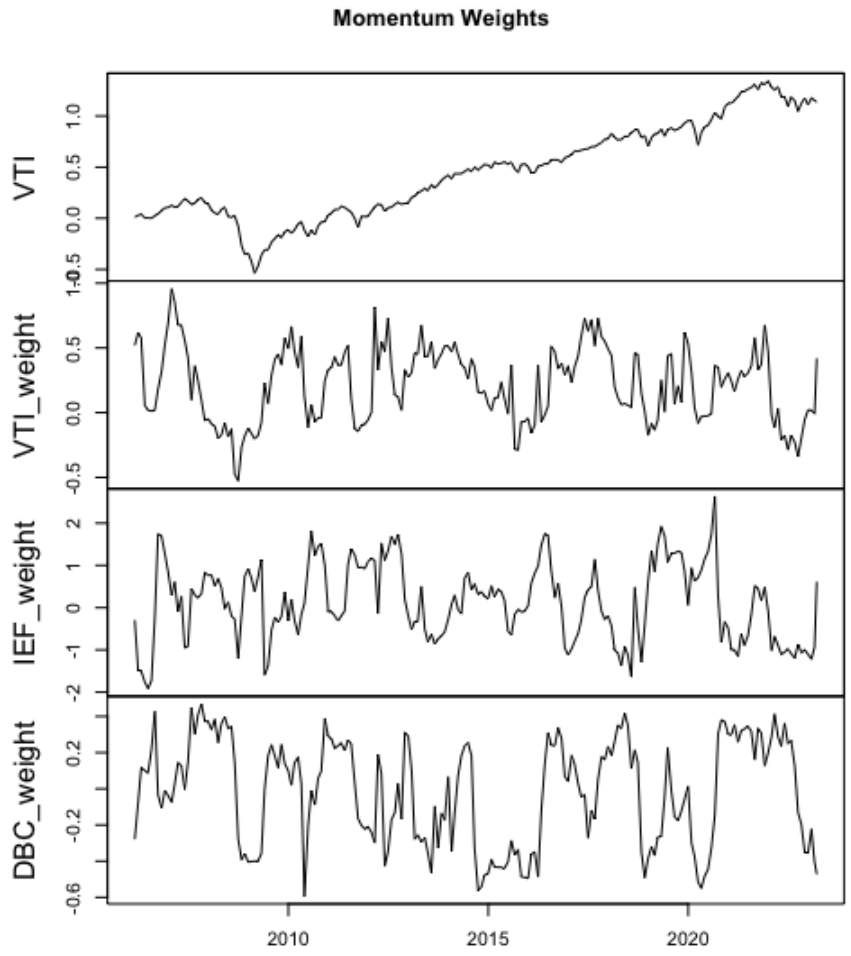
Time Series of Momentum Portfolio Weights

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

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

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

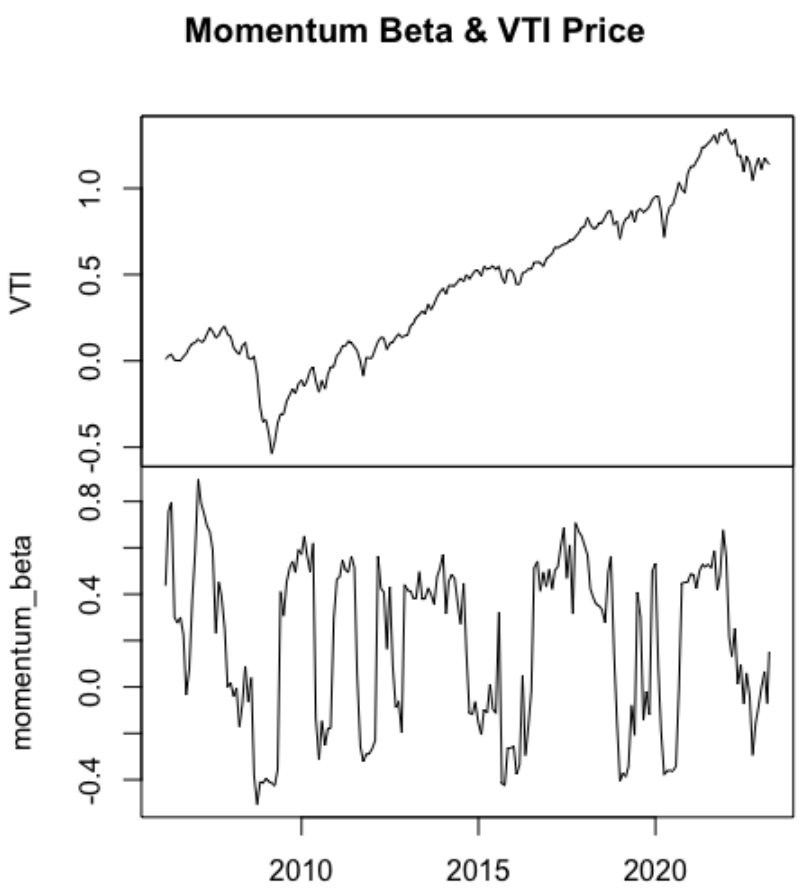
```
> # Calculate the momentum weights
> look_back <- look_backs[whichmax]
> weightv <- lapply(2:npts, function(ep) {
+   # Select the look-back returns
+   startp <- endd[max(1, ep-look_back)]
+   retsis <- retp[startp:endd[ep], ]
+   # Calculate weights proportional to performance
+   perfstat <- sapply(retsis,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



Momentum Strategy Market Beta

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

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

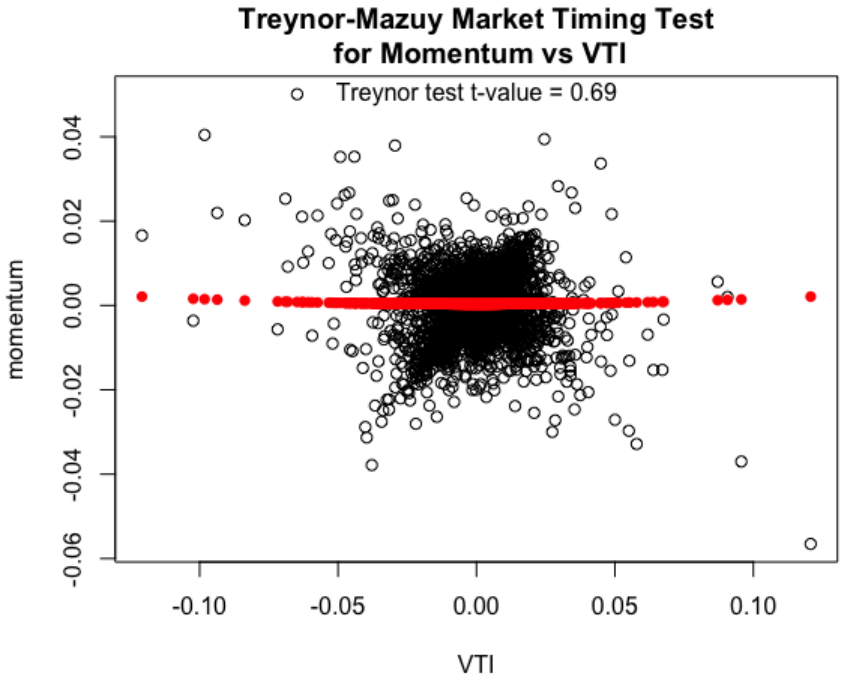


Momentum Strategy Market Timing Skill

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

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

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



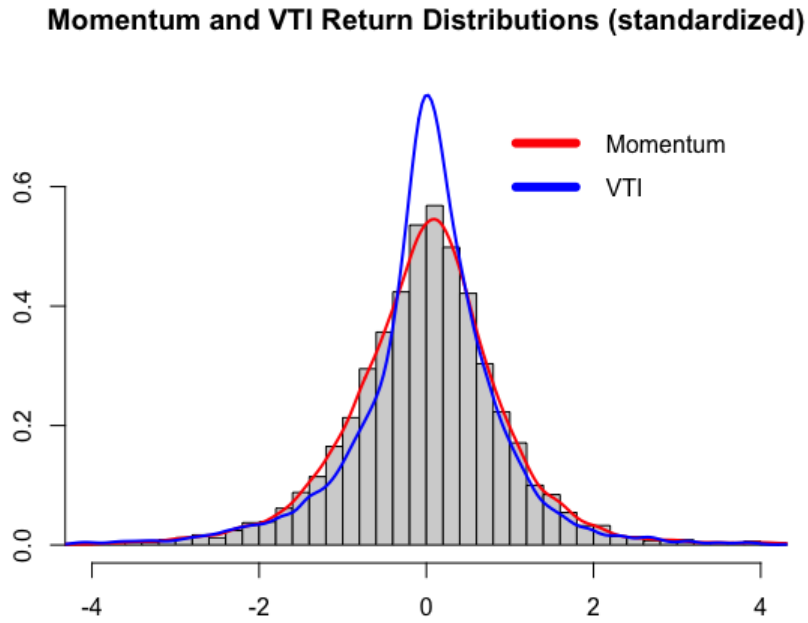
Skewness of Momentum Strategy Returns

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

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

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

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




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

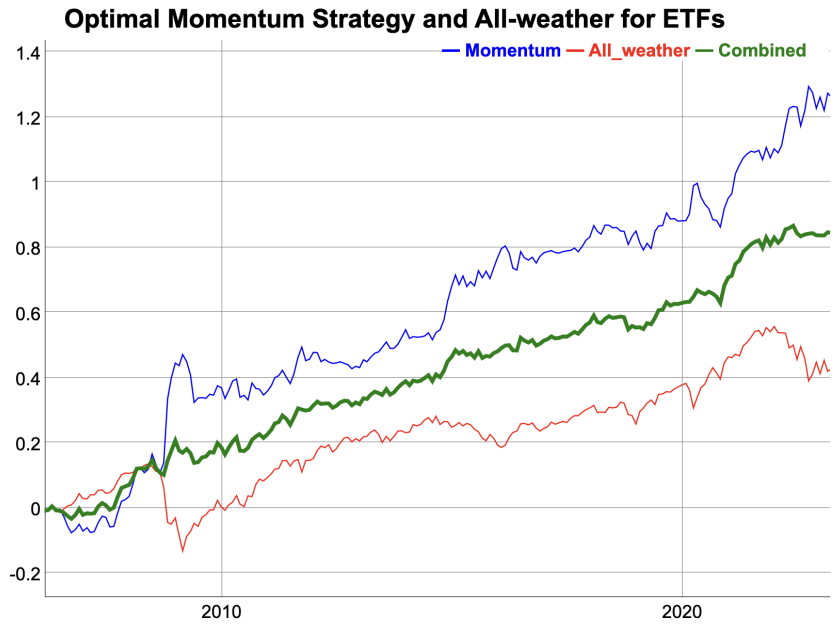
Combining Momentum with the *All-Weather* Portfolio

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

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

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

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




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

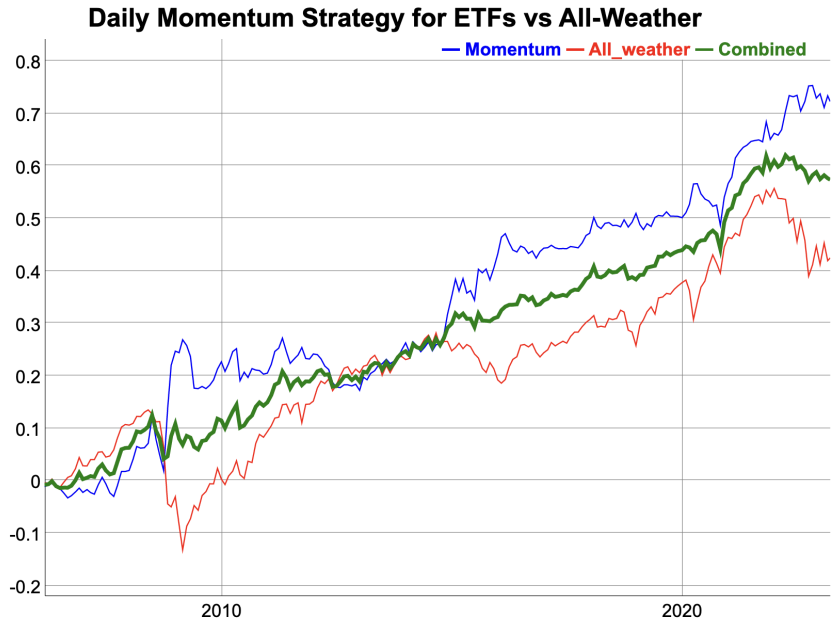
Momentum Strategy for ETFs With Daily Rebalancing

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

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

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

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




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

Homework Assignment

Required

Study all the lecture slides in `FRE7241_Lecture_5.pdf`, and run all the code in `FRE7241_Lecture_5.R`

Recommended

- Read about *optimization methods*:
  Bolker Optimization Methods.pdf
  Yollin Optimization.pdf
  Boudt DEoptim Large Portfolio Optimization.pdf
- Read about *PCA* in:
  pca-handout.pdf
  pcaTutorial.pdf