Multivariate Investment Strategies

FRE7241, Spring 2023

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

April 12, 2023

Interest Rate Yield Curve and Stock Returns

Daily stock returns have insignificant correlations with the daily changes in interest rates, with the possible exception of the 10-year bond yield.

And these correlations change significantly over time.

```
> # Load constant maturity Treasury rates
> load(file="/Users/jerzy/Develop/lecture_slides/data/rates_data.RI
> # Combine rates into single xts series
> rates <- do.call(cbind, as.list(ratesenv))
> # Sort the columns of rates according bond maturity
> namesv <- colnames(rates)
> namesv <- substr(namesv, start=4, stop=10)
> namesv <- as.numeric(names)
> indeks <- order(names)
> rates <- rates[, indeks]
> # Align rates dates with VTI prices
> closep <- log(quantmod::Cl(rutils::etfenv$VTI))
> colnames(closep) <- "VTI"
> nrows <- NROW(closep)
> datev <- zoo::index(closep)
> rates <- na.omit(rates[datev])
> closep <- closep[zoo::index(rates)]
> datev <- zoo::index(closep)
```

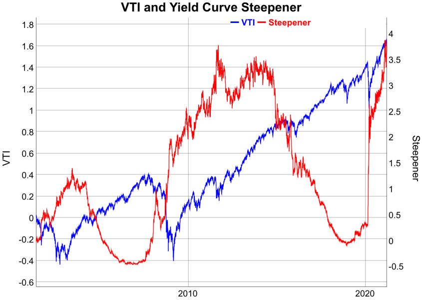
```
> # Calculate VTI returns and IR changes
> retp <- rutils::diffit(log(closep))
> retr <- rutils::diffit(log(rates))
> # Regress VTI returns versus the lagged rate differences
> predm <- rutils::lagit(retr)
> regmod <- lm(retp ~ predm)
> summary(regmod)
> # Regress VTI returns before and after 2012
> summary(lm(retp["/2012"] ~ predm["/2012"]))
> summary(lm(retp["2012/"] ~ predm["2012/"]))
```

Yield Curve Principal Components and Stock Returns

The principal components of the interest rate yield curve can also be used as predictors of stock indices.

The second principal component describes the steepening and flattening of the yield curve, and it's an indicator of investor risk appetite. So it's also related to bullish and bearish market periods.

```
> # Calculate PCA of rates correlation matrix
> eigend <- eigen(cor(retr))
> pcar <- -(retr %*% eigend$vectors)
> colnames(pcar) <- paste0("PC", 1:6)
> # Define predictor as the YC PCAs
> predm <- rutils::lagit(pcar)
> regmod <- lm(retp ~ predm)
> summary(regmod)
```




```
> # Plot YC steepener principal component with VTI
> datav <- cbind(retp, pcar[, 2])
> colnames(datav) <- c("VTI", "Steepener")
> colnamev <- colnames(datav)
> dygraphs::dygraph(cumsum(datav),
+   main="VTI and Yield Curve Steepener") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeWi
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], strokeW
+   dyLegend(show="always", width=300)
```

Yield Curve Strategy In-Sample

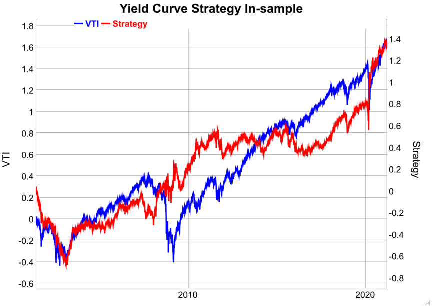
For in-sample forecasts, the training set and the test set are the same. The model is calibrated on the data that is used for forecasting.

Yield Curve Strategy

Although it's not realistic to achieve the in-sample performance, it's useful because it provides insights into how the model works.

The in-sample strategy performs well in periods of high volatility, but otherwise it's flat.

```
> # Define predictor with intercept term
> predm <- rutils::lagit(retr)
> predm <- cbind(rep(1, NROW(predm)), predm)
> colnames(predm)[1] <- "intercept"
> # Calculate inverse of predictor
> invmat <- MASS::ginv(predm)
> # Calculate coefficients from response and inverse of predictor
> respv <- retp
> coeff <- drop(invmat %*% respv)
> # Calculate forecasts and pnls in-sample
> fcast <- (predm %*% coeff)
> pnls <- sign(fcast)*response
> # Calculate in-sample factors
> factors <- (predm*coeff)
> apply(factors, 2, sd)
```



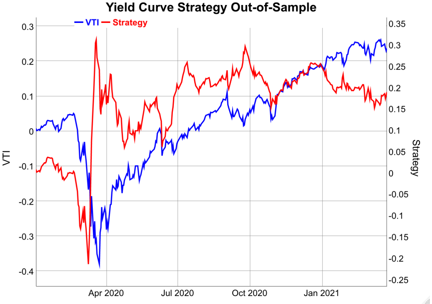

```
> # Plot dygraph of in-sample IR strategy
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Yield Curve Strategy In-sample") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

Yield Curve Strategy Out-of-Sample

For out-of-sample forecasts, the training set and the test set are separate. The model is calibrated on the training data, and forecasts are calculated using the test data.

The out-of-sample strategy performs well in periods of high volatility, but otherwise it's flat.

```
> # Define in-sample and out-of-sample intervals
> insample <- (datev < as.Date("2020-01-01"))
> outsample <- (datev >= as.Date("2020-01-01"))
> # Calculate inverse of predictor in-sample
> invmat <- MASS::ginv(predm[insample, ])
> # Calculate coefficients in-sample
> coeff <- drop(invmat %*% respv[insample, ])
> # Calculate forecasts and pnls out-of-sample
> fcast <- (predm[outsample, ] %*% coeff)
> pnls <- sign(fcast)*respv[outsample, ]
```



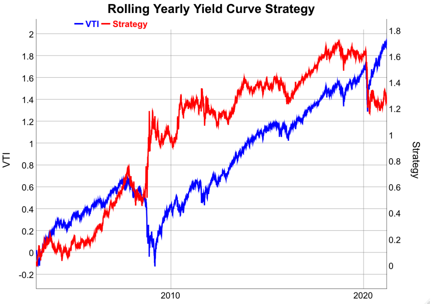

```
> # Plot dygraph of out-of-sample IR PCA strategy
> wealthv <- cbind(retp[outsample, ], pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Yield Curve Strategy Out-of-Sample") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

Rolling Yearly Yield Curve Strategy

In the rolling yearly yield curve strategy, the model is recalibrated at the end of every year using a training set of data from the past year. The coefficients are applied to calculate out-of-sample forecasts in the following year.

The rolling yearly strategy performs well in periods of high volatility, but otherwise it's flat.

```
> # Define yearly dates
> format(datev[1], "%Y")
> years <- paste0(seq(2001, 2022, 1), "-01-01")
> years <- as.Date(years)
> # Perform loop over yearly dates
> pnls <- lapply(3:(NROW(years)-1), function(ep) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > years[ep-1]) & (datev < years[ep])
+   outsample <- (datev >= years[ep]) & (datev < years[ep+1])
+   # Calculate coefficients in-sample
+   invmat <- MASS::ginv(predm[insample, ])
+   coeff <- drop(invmat %*% respv[insample, ])
+   # Calculate forecasts and pnls out-of-sample
+   fcast <- (predm[outsample, ] %*% coeff)
+   sign(fcast)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```




```
> # Plot dygraph of rolling yearly IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Yearly Yield Curve Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

Rolling Monthly Yield Curve Strategy

In the rolling monthly yield curve strategy, the model is recalibrated at the end of every month using a training set of the past 11 months. The coefficients are applied to perform out-of-sample forecasts in the following month.

Research shows that looking back roughly a year provides the best out-of-sample forecasts.

The rolling monthly strategy performs better than the yearly strategy, but mostly in periods of high volatility, and otherwise it's flat.

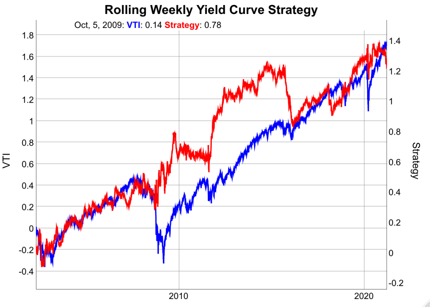

```
> # Define monthly dates
> format(datev[1], "%m-%Y")
> format(datev[NROW(datev)], "%m-%Y")
> months <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-01"), by="month")
> # Perform loop over monthly dates
> pnls <- lapply(12:(NROW(months)-1), function(ep) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > months[ep-11]) & (datev < months[ep])
+   outsample <- (datev > months[ep]) & (datev < months[ep+1])
+   # Calculate forecasts and pnls out-of-sample
+   invmat <- MASS::ginv(predm[insample, ])
+   coeff <- drop(invmat %*% respv[insample, ])
+   fcast <- (predm[outsample, ] %*% coeff)
+   sign(fcast)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```

```
> # Plot dygraph of rolling monthly IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Monthly Yield Curve Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2) %>%
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Weekly Yield Curve Strategy

In the rolling weekly yield curve strategy, the model is recalibrated at the end of every week using a training set of the past 10 weeks. The coefficients are applied to perform out-of-sample forecasts in the following week.

```
> # Define weekly dates
> weeks <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-
> # Perform loop over weekly dates
> pnls <- lapply(51:(NROW(weeks)-1), function(ep) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > weeks[ep-10]) & (datev < weeks[ep])
+   outsample <- (datev > weeks[ep]) & (datev < weeks[ep+1])
+   # Calculate forecasts and pnls out-of-sample
+   invmat <- MASS::ginv(predm[insample, ])
+   coeff <- drop(invmat %*% respv[insample, ])
+   fcast <- (predm[outsample, ] %*% coeff)
+   sign(fcast)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```




```
> # Plot dygraph of rolling weekly IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Weekly Yield Curve Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

Regularization of the Inverse Predictor Matrix

The *SVD* of a rectangular matrix $\mathbb{A}$ is defined as the factorization:

$$\mathbb{A} = \mathbb{U} \Sigma \mathbb{V}^T$$

Where $\mathbb{U}$ and $\mathbb{V}$ are the *singular matrices*, and Σ is a diagonal matrix of *singular values*.

The *generalized inverse* matrix $\mathbb{A}^{-1}$ satisfies the inverse equation: $\mathbb{A}\mathbb{A}^{-1}\mathbb{A} = \mathbb{A}$, and it can be expressed as a product of the *SVD* matrices as follows:

$$\mathbb{A}^{-1} = \mathbb{V} \, \Sigma^{-1} \, \mathbb{U}^T$$

If any of the *singular values* are zero then the *generalized inverse* does not exist.

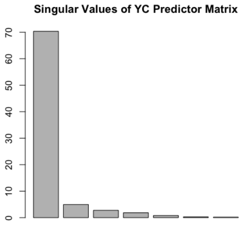
Regularization is the removal of zero singular values, to make calculating the inverse matrix possible.

The *regularized inverse* is obtained by removing the zero *singular values*:

$$\mathbb{A}^{-1} = \mathbb{V}_n \, \Sigma_n^{-1} \, \mathbb{U}_n^T$$

Where $\mathbb{U}_n$, $\mathbb{V}_n$ and Σ_n are the *SVD* matrices without the zero *singular values*.

The regularized inverse satisfies the inverse matrix equation: $\mathbb{A}\mathbb{A}^{-1}\mathbb{A} = \mathbb{A}$.




```
> # Calculate singular value decomposition of the predictor matrix
> svdec <- svd(predm)
> barplot(svdec$d, main="Singular Values of YC Predictor Matrix")
> # Calculate generalized inverse from SVD
> invsvd <- svdec$v %*% (t(svdec$u) / svdec$d)
> # Verify inverse property of inverse
> all.equal(zoo::coredata(predm), predm %*% invsvd %*% predm)
> # Calculate generalized inverse using MASS::ginv()
> invmat <- MASS::ginv(predm)
> all.equal(invmat, invsvd)
> # Set tolerance for determining zero singular values
> precv <- sqrt(.Machine$double.eps)
> # Check for zero singular values
> round(svdec$d, 12)
> notzero <- (svdec$d > (precv*svdec$d[1]))
> # Calculate regularized inverse from SVD
> invreg <- svdec$v[, notzero] %*%
+   (t(svdec$u[, notzero]) / svdec$d[notzero])
> # Verify inverse property of invreg
> all.equal(zoo::coredata(predm), predm %*% invreg %*% predm)
> all.equal(invreg, invmat)
```

Shrinkage Inverse of the Predictor Matrix

Regularization is the removal of zero singular values, to make calculating the inverse matrix possible.

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

Dimension reduction is achieved by the removal of small singular values, to improve the out-of-sample performance of the inverse matrix.

The *shrinkage inverse* is obtained by removing the very small *singular values*.

$$\mathbb{A}^{-1} = \mathbb{V}_n \, \Sigma_n^{-1} \, \mathbb{U}_n^T$$

This effectively reduces the number of parameters in the model.

The *shrinkage inverse* satisfies the inverse equation only approximately (it is *biased*), but it's often used in machine learning because it produces a lower *variance* of the forecasts than the exact inverse.

```
> # Calculate shrinkage inverse from SVD
> dimax <- 3
> invreg <- svdec$v[, 1:dimax] %*%
+   (t(svdec$u[, 1:dimax]) / svdec$d[1:dimax])
> # Inverse property fails for invreg
> all.equal(zoo::coredata(predm), predm %*% invreg %*% predm)
> # Calculate shrinkage inverse using RcppArmadillo
> inverse_rcpp <- HighFreq::calc_inv(predm, dimax=dimax)
> all.equal(invreg, inverse_rcpp, check.attributes=FALSE)
```

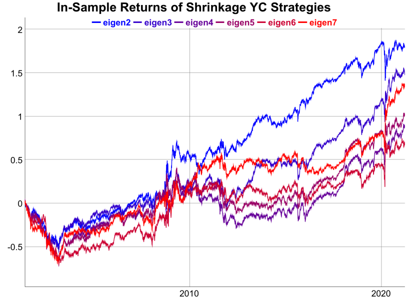
Yield Curve Strategy With Shrinkage In-Sample

The technique of *regularization* is designed to reduce the number of parameters in a model, for example in portfolio optimization.

Regularization of the inverse predictor matrix improves the in-sample performance of the yield curve strategy.

Although it's not realistic to achieve the in-sample performance, it's useful because it provides insights into how the model can be improved.

```
> # Calculate in-sample pnls for different dimax values
> eigenvals <- 2:7
> pnls <- lapply(eigenvals, function(dimax) {
+   invmat <- HighFreq::calc_inv(predm, dimax=dimax)
+   coeff <- drop(invmat %*% respv)
+   fcast <- (predm %*% coeff)
+   sign(fcast)*response
+ })
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("eigen", eigenvals)
```



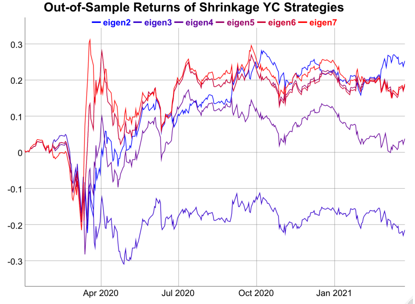

```
> # Plot dygraph of in-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="In-Sample Returns of Shrinka
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Yield Curve Strategy With Shrinkage Out-of-Sample

For out-of-sample forecasts, the training set and the test set are separate. The model is calibrated on the training data, and forecasts are calculated using the test data.

The out-of-sample strategy performs well in periods of high volatility, but otherwise it's flat.

```
> # Define in-sample and out-of-sample intervals
> insample <- (datev < as.Date("2020-01-01"))
> outsample <- (datev >= as.Date("2020-01-01"))
> # Calculate in-sample pnls for different dimax values
> eigenvals <- 2:7
> pnls <- lapply(eigenvals, function(x) {
+   invmat <- HighFreq::calc_inv(predm[insample, ], dimax=x)
+   coeff <- drop(invmat %*% respv[insample, ])
+   fcast <- (predm[outsample, ] %*% coeff)
+   sign(fcast)*respv[outsample, ]
+ })
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("eigen", eigenvals)
```



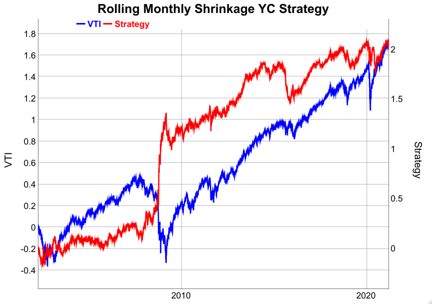

```
> # Plot dygraph of out-of-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="Out-of-Sample Returns of Sh
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Rolling Monthly Yield Curve Strategy With Dimension Reduction

The shrinkage rolling monthly strategy performs better than the standard strategy because regularization allows using shorter `look_back` intervals since it suppresses the response noise.

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

```
> # Define monthly dates
> format(datev[1], "%m-%Y")
> format(datev[NROW(datev)], "%m-%Y")
> months <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04
> # Perform loop over monthly dates
> look_back <- 6
> dimax <- 3
> pnls <- lapply((look_back+1):(NROW(months)-1), function(ep) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > months[ep-look_back]) & (datev < months[ep
+   outsample <- (datev > months[ep]) & (datev < months[ep+1])
+   # Calculate forecasts and pnls out-of-sample
+   invmat <- HighFreq::calc_inv(predm[insample, ], dimax=dimax)
+   coeff <- drop(invmat %*% respv[insample, ])
+   fcast <- (predm[outsample, ] %*% coeff)
+   sign(fcast)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```

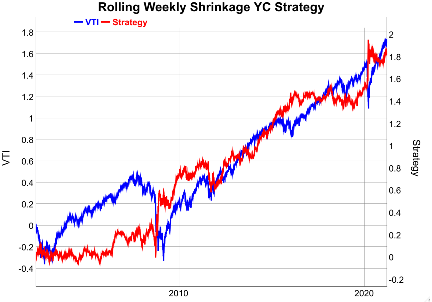



```
> # Plot dygraph of rolling monthly IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Monthly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

Rolling Weekly Yield Curve Strategy With Shrinkage

In the rolling weekly yield curve strategy, the model is recalibrated at the end of every week using a training set of the past 4 weeks. The coefficients are applied to perform out-of-sample forecasts in the following week.

```
> # Define weekly dates
> weeks <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-
> # Perform loop over weekly dates
> look_back <- 4
> dimax <- 4
> pnls <- lapply((look_back+1):(NROW(weeks)-1), function(ep) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > weeks[ep-look_back]) & (datev < weeks[ep])
+   outsample <- (datev > weeks[ep]) & (datev < weeks[ep+1])
+   # Calculate forecasts and pnls out-of-sample
+   invmat <- HighFreq::calc_inv(predm[insample, ], dimax=dimax)
+   coeff <- drop(invmat %*% respv[insample, ])
+   fcast <- (predm[outsample, ] %*% coeff)
+   sign(fcast)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```




```
> # Plot dygraph of rolling weekly IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Weekly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

draft: Combined Predictor Matrix

A "kitchen sink" strategy combines many different predictors into a large predictor matrix with many columns.

For example by combining the yield curve predictors with the lagged returns.

```
> # Load the yield curve data
> load(file="/Users/jerzy/Develop/lecture_slides/data/rates_data.RDa
> rates <- do.call(cbind, as.list(ratesenv))
> namesv <- colnames(rates)
> namesv <- substr(namesv, start=4, stop=10)
> namesv <- as.numeric(names)
> indeks <- order(names)
> rates <- rates[, indeks]
> closep <- log(quantmod::Cl(rutils::etfenv$VTI))
> colnames(closep) <- "VTI"
> nrows <- NROW(closep)
> datev <- zoo::index(closep)
> rates <- na.omit(rates[datev])
> closep <- closep[zoo::index(rates)]
> datev <- zoo::index(closep)
> retp <- rutils::diffit(log(closep))
> retr <- rutils::diffit(log(rates))
> # Create a combined predictor matrix
> ordmax <- 5
> predm <- sapply(1:ordmax, rutils::lagit, input=as.numeric(retp))
> colnames(predm) <- paste0("retslag", 1:NCOL(predm))
> predm <- cbind(predm, rutils::lagit(retr))
> predm <- cbind(rep(1, NROW(predm)), predm)
> colnames(predm)[1] <- "intercept"
> respv <- retp
```

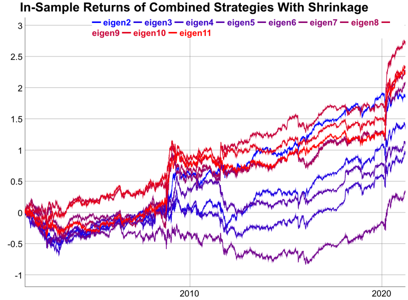
draft: Combined Strategy With Shrinkage In-Sample

The technique of *regularization* is designed to reduce the number of parameters in a model, for example in portfolio optimization.

Regularization of the inverse predictor matrix improves the in-sample performance of the yield curve strategy.

Although it's not realistic to achieve the in-sample performance, it's useful because it provides insights into how the model can be improved.

```
> # Calculate in-sample pnls for different dimax values
> eigenvals <- 2:11
> pnls <- lapply(eigenvals, function(dimax) {
+   invmat <- HighFreq::calc_inv(predm, dimax=dimax)
+   coeff <- drop(invmat %*% respv)
+   fcast <- (predm %*% coeff)
+   sign(fcast)*response
+ })
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("eigen", eigenvals)
```



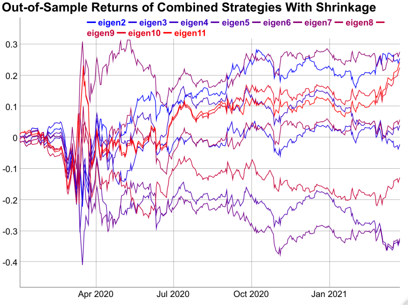

```
> # Plot dygraph of in-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="In-Sample Returns of Combine
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

draft: Combined Strategy With Shrinkage Out-of-Sample

For out-of-sample forecasts, the training set and the test set are separate. The model is calibrated on the training data, and forecasts are calculated using the test data.

The out-of-sample strategy performs well in periods of high volatility, but otherwise it's flat.

```
> # Define in-sample and out-of-sample intervals
> insample <- (datev < as.Date("2020-01-01"))
> outsample <- (datev >= as.Date("2020-01-01"))
> # Calculate in-sample pnls for different dimax values
> eigenvals <- 2:11
> pnls <- lapply(eigenvals, function(x) {
+   invmat <- HighFreq::calc_inv(predm[insample, ], dimax=x)
+   coeff <- drop(invmat %*% respv[insample, ])
+   fcast <- (predm[outsample, ] %*% coeff)
+   sign(fcast)*respv[outsample, ]
+ })
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("eigen", eigenvals)
```



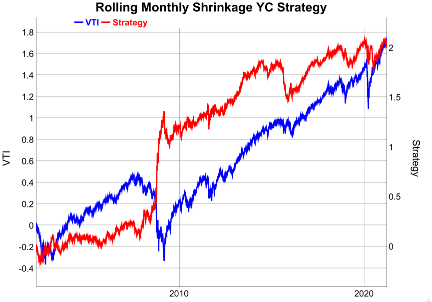

```
> # Plot dygraph of out-of-sample pnls
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls), main="Out-of-Sample Returns of Com
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

draft: Rolling Monthly Combined Strategy With Dimension Reduction

The shrinkage rolling monthly strategy performs better than the standard strategy because regularization allows using shorter look_back intervals since it suppresses the response noise.

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

```
> # Define monthly dates
> format(datev[1], "%m-%Y")
> format(datev[NROW(datev)], "%m-%Y")
> months <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04
> # Perform loop over monthly dates
> look_back <- 6
> dimax <- 3
> pnls <- lapply((look_back+1):(NROW(months)-1), function(ep) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > months[ep-look_back]) & (datev < months[ep
+   outsample <- (datev > months[ep]) & (datev < months[ep+1])
+   # Calculate forecasts and pnls out-of-sample
+   invmat <- HighFreq::calc_inv(predm[insample, ], dimax=dimax)
+   coeff <- drop(invmat %*% respv[insample, ])
+   fcast <- (predm[outsample, ] %*% coeff)
+   sign(fcast)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```

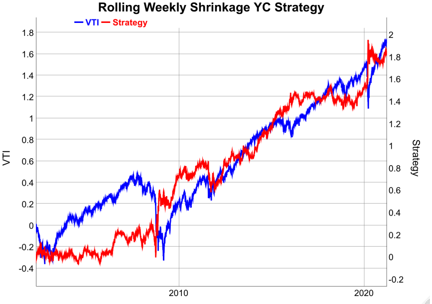



```
> # Plot dygraph of rolling monthly IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Monthly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

draft: Rolling Weekly Combined Strategy With Shrinkage

In the rolling weekly yield curve strategy, the model is recalibrated at the end of every week using a training set of the past 4 weeks. The coefficients are applied to perform out-of-sample forecasts in the following week.

```
> # Define weekly dates
> weeks <- seq.Date(from=as.Date("2001-05-01"), to=as.Date("2021-04-
> # Perform loop over weekly dates
> look_back <- 8
> dimax <- 4
> pnls <- lapply((look_back+1):(NROW(weeks)-1), function(ep) {
+   # Define in-sample and out-of-sample intervals
+   insample <- (datev > weeks[ep-look_back]) & (datev < weeks[ep])
+   outsample <- (datev > weeks[ep]) & (datev < weeks[ep+1])
+   # Calculate forecasts and pnls out-of-sample
+   invmat <- HighFreq::calc_inv(predm[insample, ], dimax=dimax)
+   coeff <- drop(invmat %*% respv[insample, ])
+   fcast <- (predm[outsample, ] %*% coeff)
+   sign(fcast)*respv[outsample, ]
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```




```
> # Plot dygraph of rolling weekly IR strategy
> vti <- rutils::diffit(closep[zoo::index(pnls),])
> wealthv <- cbind(vti, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Weekly Shrinkage YC Strategy") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

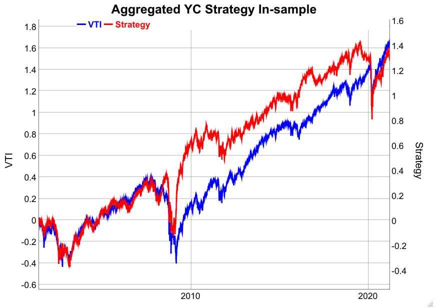
draft: Forecasts Using Aggregated Predictor

Needs more work to improve performance

Aggregating the predictor reduces its noise and increases the significance of correlations.

The optimal aggregation number can be found by maximizing the regression t-values.

```
> # Find optimal nagg for predictor
> naggs <- 5:100
> tvalues <- sapply(naggs, function(nagg) {
+   predm <- HighFreq::roll_mean(retr, look_back=nagg)
+   predm <- cbind(rep(1, NROW(predm)), predm)
+   predm <- rutils::lagit(predm)
+   regmod <- lm(respv ~ predm - 1)
+   modsum <- summary(regmod)
+   max(abs(modsum$coefficients[, 3][-1]))
+ })  # end sapply
> naggs[which.max(tvalues)]
> plot(naggs, tvalues, t="l", col="blue", lwd=2)
> # Calculate aggregated predictor
> nagg <- 53
> predm <- HighFreq::roll_mean(retr, look_back=nagg)
> predm <- rutils::lagit(predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> regmod <- lm(respv ~ predm - 1)
> summary(regmod)
```




```
> # Calculate forecasts and pnls in-sample
> invmat <- MASS::ginv(predm)
> coeff <- drop(invmat %*% respv)
> fcast <- (predm %*% coeff)
> pnls <- sign(fcast)*response
> # Plot dygraph of in-sample IR strategy
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Aggregated YC Strategy In-sample") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

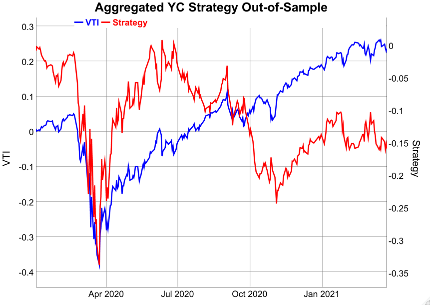
draft: Aggregated Forecasts Out-of-Sample

Needs more work to improve performance

For out-of-sample forecasts, the training set and the test set are separate. The model is calibrated on the training data, and forecasts are calculated using the test data.

The out-of-sample strategy performs well in periods of high volatility, but otherwise it's flat.

```
> # Define in-sample and out-of-sample intervals
> insample <- (datev < as.Date("2020-01-01"))
> outsample <- (datev >= as.Date("2020-01-01"))
> # Calculate forecasts and pnls out-of-sample
> invmat <- MASS::ginv(predm[insample, ])
> coeff <- drop(invmat %*% respv[insample, ])
> fcast <- (predm[outsample, ] %*% coeff)
> pnls <- sign(fcast)*respv[outsample, ]
```




```
> # Plot dygraph of out-of-sample YC strategy
> wealthv <- cbind(retp[outsample, ], pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> colnamev <- colnames(wealthv)
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Aggregated YC Strategy Out-of-Sample") %>%
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", col="blue", strokeWidth=2)
+   dySeries(name=colnamev[2], axis="y2", col="red", strokeWidth=2)
+   dyLegend(show="always", width=300)
```

Idiosyncratic Stock Returns

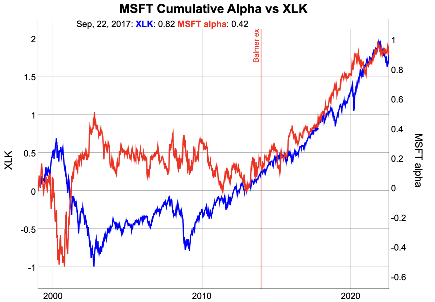
The daily stock returns $r_i - r_f$ in excess of the risk-free rate r_f, can be decomposed into *systematic* returns $\beta(r_m - r_f)$ (r_m are the market returns) plus *idiosyncratic* returns $\alpha + \varepsilon_i$ (which are uncorrelated to the market returns):

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

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

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

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



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

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

Trailing Idiosyncratic Stock Returns

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

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

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

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

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

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

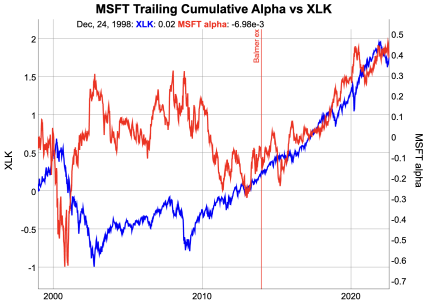
$$\beta_t = \frac{\mathrm{cov}_t}{\sigma^2_t}$$

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

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

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

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




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

Cointegration of Stocks Prices

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

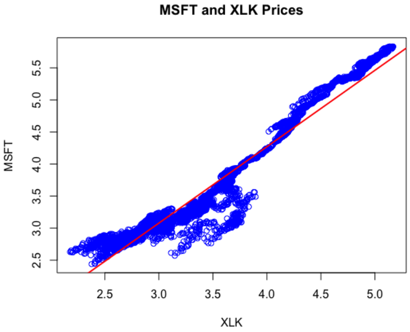
For two stocks, the cointegrating factor β can be obtained from the regression of the stock prices. The cointegrated portfolio price R is the residual of the regression:

$$R = p_1 - \beta p_2$$

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

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

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




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

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

Cointegrated Portfolio Prices

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

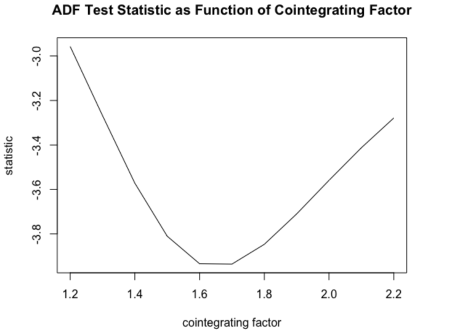
- Perform a regression of the stock prices to calculate the cointegrating factor β,
- Apply the *ADF* test to the cointegrated portfolio price, to determine if it has a unit root (the portfolio price diverges), or if it's mean reverting.

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

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

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

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




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

Bollinger Band Strategy for Cointegrated Pairs

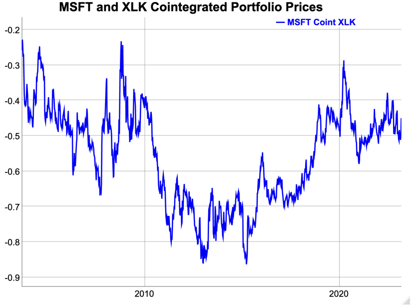
The returns of the cointegrated portfolio have negative autocorrelations, so it can be traded using a mean-reverting strategy.

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

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

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

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



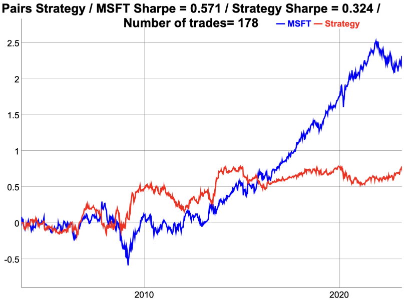

```
> # Plot of PACF of the cointegrated portfolio returns
> pricen <- zoo::coredata(pricec) # Numeric price
> retd <- rutils::diffit(pricen)
> pacf(retd, lag=10, xlab=NA, ylab=NA,
+      main="PACF of Cointegrated Portfolio Returns")
> # Dygraphs plot of cointegrated portfolio prices
> endd <- rutils::calc_endpoints(pricec,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

Pairs Strategy With Fixed Beta

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

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

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



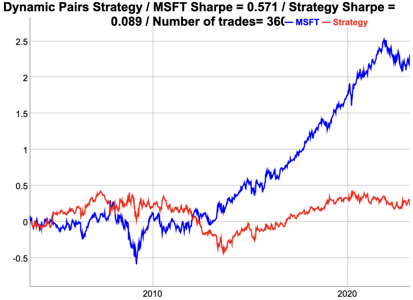

```
> # Calculate the Sharpe ratios
> wealthv <- cbind(retp$MSFT, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sharper <- sqrt(252)*sapply(wealthv, function(x) mean(x)/sd(x[x<0]
> sharper <- round(sharper, 3)
> # Dygraphs plot of pairs Bollinger strategy
> colnamev <- colnames(wealthv)
> caption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

Pairs Strategy With Dynamic Beta

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

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

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




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

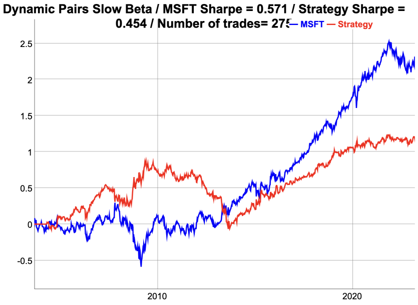
Pairs Strategy With Slow Beta

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

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

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

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




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

Stock Index Weighting Methods

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

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

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

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

The equal-dollar-weighted index invests equal dollar amounts in each stock, and it rebalances its allocations as market prices change.

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

```
> # Load daily S&P500 stock prices
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_prices.R
> # Calculate the percentage returns
> retp <- lapply(prices, function(x) xts::diff.xts(x)/rutils::lagit(
> retp <- rutils::do_call(cbind, retp)
> # Subset (select) the stock returns after the start date of VTI
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> retp <- retp[zoo::index(retvti)]
> datev <- zoo::index(retp)
> retvti <- retvti[datev]
> nrows <- NROW(retp)
> nstocks <- NCOL(retp)
> head(retp[, 1:5])
```

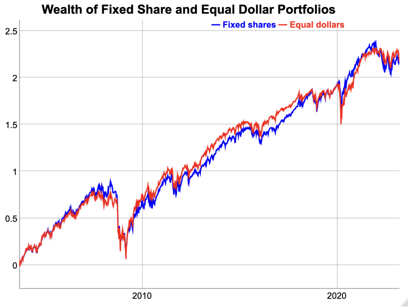
The Equal-Weight Portfolio

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

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

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

```
> # Select stocks with non-NA returns at beginning
> retna <- retp[, !is.na(retp[1, ])]
> # Replace remaining NA returns with zeros
> retna[is.na(retna)] <- 0
> # Wealth of fixed shares (price-weighted) portfolio
> wealthfs <- rowMeans(cumprod(1+retna))
> # Wealth of equal-dollar-weighted portfolio
> wealthew <- cumprod(1+rowMeans(retna))
```




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

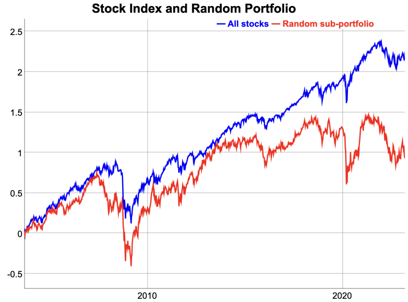
Random Stock Selection

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

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

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

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




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

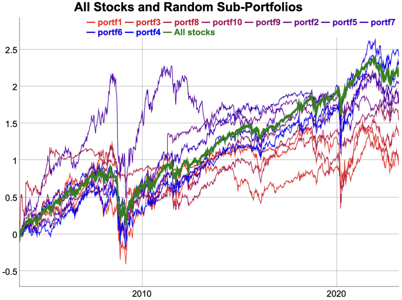
Random Stock Portfolios

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

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

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

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




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

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

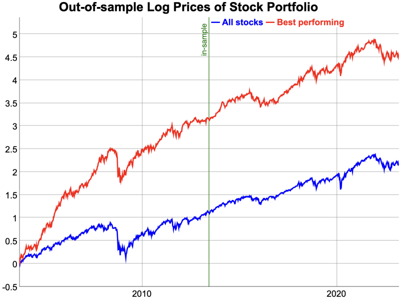
Stock Portfolio Selection Out-of-Sample

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

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

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

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




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

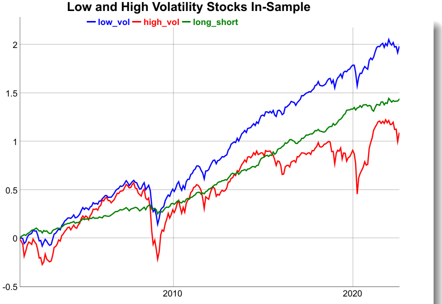
Low and High Volatility Stock Portfolios

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

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

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

```
> # Calculate the stock volatilities, betas, and alphas
> riskret <- sapply(retp, function(retp) {
+   retp <- na.omit(retp)
+   std <- sd(retp)
+   retvti <- retvti[zoo::index(retp)]
+   varvti <- drop(var(retvti))
+   meanvti <- mean(retvti)
+   betav <- drop(cov(retp, retvti))/varvti
+   resid <- retp - betav*retvti
+   alphav <- mean(retp) - betav*meanvti
+   c(alpha=alphav, beta=betav, std=std, ivol=sd(resid))
+ })  # end sapply
> riskret <- t(riskret)
> tail(riskret)
> # Calculate the median volatility
> riskv <- riskret[, "std"]
> medianv <- median(riskv)
> # Calculate the returns of low and high volatility stocks
> retlow <- rowMeans(retna[, names(riskv[riskv<=medianv])])
> rethigh <- rowMeans(retna[, names(riskv[riskv>medianv])])
> wealthv <- cbind(retlow, rethigh, retlow - 0.5*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnamev <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colnamev
```



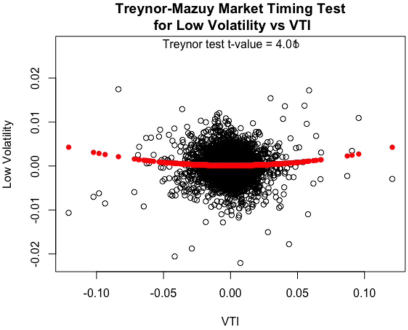

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high volatility stocks
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Low and High Volati
+   dySeries(name=colnamev[1],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

Low Volatility Stock Portfolio Market Timing Skill

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

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

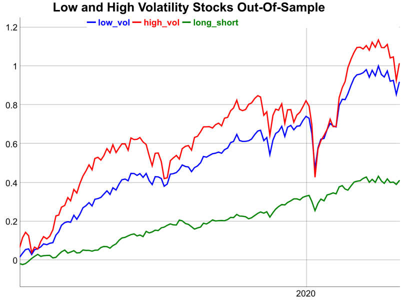
```
> # Merton-Henriksson test
> predm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predm)[2:3] <- c("merton", "treynor")
> regmod <- lm(wealthv$long_short ~ VTI + merton, data=predm); summa
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + treynor, data=predm);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



Low and High Volatility Stock Portfolios Out-Of-Sample

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

```
> # Calculate the in-sample stock volatilities, betas, and alphas
> riskretis <- sapply(retp[insample], function(retp) {
+   combv <- na.omit(cbind(retp, retvti))
+   if (NROW(combv) > 0) {
+     retp <- na.omit(retp)
+     std <- sd(retp)
+     retvti <- retvti[zoo::index(retp)]
+     varvti <- drop(var(retvti))
+     meanvti <- mean(retvti)
+     betav <- drop(cov(retp, retvti))/varvti
+     resid <- retp - betav*retvti
+     alphav <- mean(retp) - betav*meanvti
+     return(c(alpha=alphav, beta=betav, std=std, ivol=sd(resid)))
+   } else {
+     return(c(alpha=0, beta=0, std=0, ivol=0))
+   }  # end if
+ })  # end sapply
> riskretis <- t(riskretis)
> tail(riskretis)
> # Calculate the median volatility
> riskv <- riskretis[, "std"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high volatility
> retlow <- rowMeans(retna[outsample, names(riskv[riskv<=medianv])]
> rethigh <- rowMeans(retna[outsample, names(riskv[riskv>medianv])])
> wealthv <- cbind(retlow, rethigh, retlow - 0.5*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colnamev <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colnamev
```



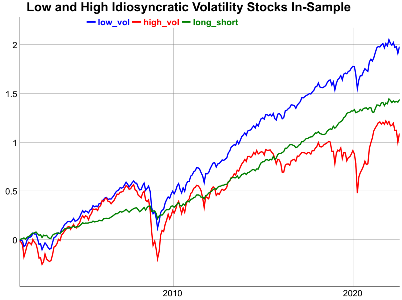

```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high volatility stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],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

Low and High Idiosyncratic Volatility Stock Portfolios

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

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

```
> # Calculate the median idiosyncratic volatility
> riskv <- riskret[, "ivol"]
> medianv <- median(riskv)
> # Calculate the returns of low and high idiosyncratic volatility s
> retlow <- rowMeans(retna[, names(riskv[riskv<=medianv])])
> rethigh <- rowMeans(retna[, names(riskv[riskv>medianv])])
> wealthv <- cbind(retlow, rethigh, retlow - 0.5*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnamev <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colnamev
```



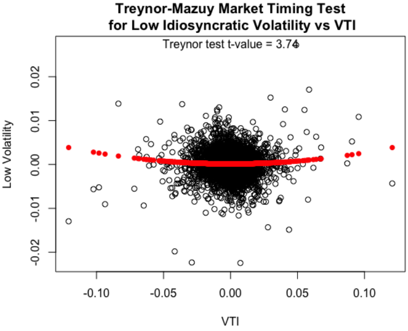

```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of returns of low and high idiosyncratic volatility stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],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

Low Idiosyncratic Volatility Stock Portfolio Market Timing Skill

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

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

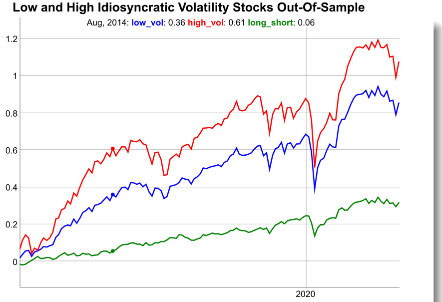
```
> # Merton-Henriksson test
> predm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predm)[2:3] <- c("merton", "treynor")
> regmod <- lm(wealthv$long_short ~ VTI + merton, data=predm); summa
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + treynor, data=predm);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



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

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

```
> # Calculate the median in-sample idiosyncratic volatility
> riskv <- riskretis[, "ivol"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high idiosyncrat
> retlow <- rowMeans(retna[outsample, names(riskv[riskv<=medianv])])
> rethigh <- rowMeans(retna[outsample, names(riskv[riskv>medianv])])
> wealthv <- cbind(retlow, rethigh, retlow - 0.5*rethigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colnamev <- c("low_vol", "high_vol", "long_short")
> colnames(wealthv) <- colnamev
```




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

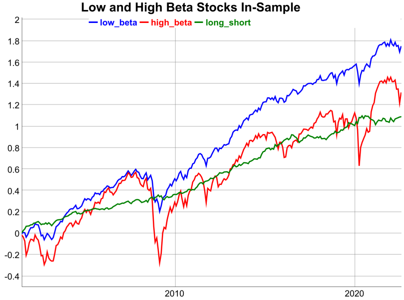
Low and High Beta Stock Portfolios

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

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

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

```
> # Calculate the median beta
> riskv <- riskret[, "beta"]
> medianv <- median(riskv)
> # Calculate the returns of low and high beta stocks
> betalow <- rowMeans(retna[, names(riskv[riskv<=medianv])])
> betahigh <- rowMeans(retna[, names(riskv[riskv>medianv])])
> wealthv <- cbind(betalow, betahigh, betalow - 0.5*betahigh)
> wealthv <- xts::xts(wealthv, order.by=datev)
> colnamev <- c("low_beta", "high_beta", "long_short")
> colnames(wealthv) <- colnamev
```



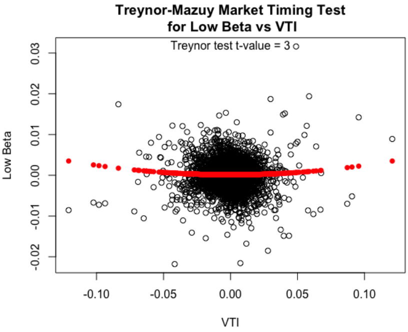

```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative returns of low and high beta stocks
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],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

Low Beta Stock Portfolio Market Timing Skill

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

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

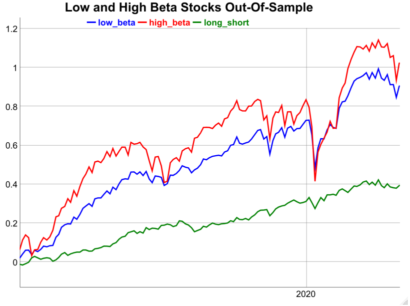
```
> # Merton-Henriksson test
> predm <- cbind(VTI=retvti, 0.5*(retvti+abs(retvti)), retvti^2)
> colnames(predm)[2:3] <- c("merton", "treynor")
> regmod <- lm(wealthv$long_short ~ VTI + merton, data=predm); summa
> # Treynor-Mazuy test
> regmod <- lm(wealthv$long_short ~ VTI + treynor, data=predm); sumr
> # Plot residual scatterplot
> resids <- regmod$residuals
> plot.default(x=retvti, y=resids, xlab="VTI", ylab="Low Beta")
> title(main="Treynor-Mazuy Market Timing Test\n for Low Beta vs VTI
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$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))
```



Low and High Beta Stock Portfolios Out-Of-Sample

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

```
> # Calculate the median beta
> riskv <- riskretis[, "beta"]
> medianv <- median(riskv)
> # Calculate the out-of-sample returns of low and high beta stocks
> betalow <- rowMeans(retna[outsample, names(riskv[riskv<=medianv])]
> betahigh <- rowMeans(retna[outsample, names(riskv[riskv>medianv])]
> wealthv <- cbind(betalow, betahigh, betalow - 0.5*betahigh)
> wealthv <- xts::xts(wealthv, order.by=datev[outsample])
> colnamev <- c("low_beta", "high_beta", "long_short")
> colnames(wealthv) <- colnamev
```




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

Risk Parity Strategy for Stocks

In the *Risk Parity* strategy the dollar portfolio allocations are rebalanced daily so that their dollar volatilities remain equal.

The dollar amount of stock that has unit dollar volatility is equal to the *standardized prices* $\frac{p_i}{\sigma_i^d}$. Where σ_i^d is the dollar volatility.

So the allocations a_i should be proportional to the *standardized prices*: $a_i \propto \frac{p_i}{\sigma_i^d}$,

But the *standardized prices* are equal to the inverse of the percentage volatilities σ_i: $\frac{p_i}{\sigma_i^d} = \frac{1}{\sigma_i}$, so the allocations a_i are proportional to the inverse of the percentage volatilities $a_i \propto \frac{1}{\sigma_i}$.

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

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

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

Where σ^2_t is the trailing variance at time t, and r_t is the *time series* of returns.

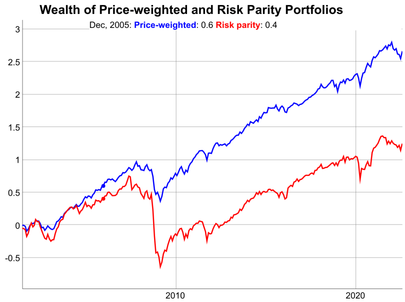
```
> # Calculate the trailing percentage volatilities
> lambda <- 0.6
> volat <- HighFreq::run_var(retna, lambda=lambda)
> volat <- sqrt(volat)
> # Calculate the risk parity portfolio allocations
> alloc <- ifelse(volat > 1e-4, 1/volat, 0)
> # Scale allocations to 1 dollar total
> allocs <- rowSums(alloc)
> alloc <- ifelse(allocs > 0, alloc/allocs, 0)
> # Lag the allocations
> alloc <- rutils::lagit(alloc)
```

Risk Parity Strategy for Stocks Performance

The risk parity strategy does not perform well for stocks because their correlations are positive.

The risk parity strategy performs better when the correlations of asset returns are negative, and worse when the correlations are positive.

```
> # Calculate the wealth of risk parity
> wealthrp <- exp(cumsum(rowSums(alloc*retp, na.rm=TRUE)))
> # Combined wealth
> wealthv <- cbind(wealthfs, wealthrp)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Price-weighted", "Risk parity")
> wealthv <- log(wealthv)
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv),
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot of log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Price-weighted and Risk Parity Portfolios") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

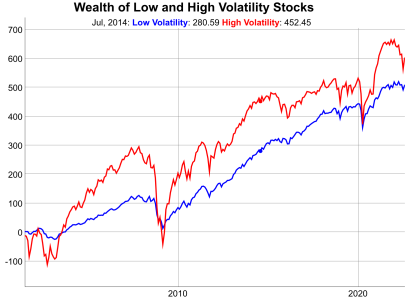
Stocks With Low and High Trailing Volatilities

The trailing volatilities can be used to create low and high volatility portfolios, which are rebalanced daily.

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

The low volatility portfolio has higher risk-adjusted returns than the high volatility portfolio, which contradicts the *CAPM* model.

```
> # Calculate the median volatilities
> medianv <- matrixStats::rowMedians(volat)
> # Calculate the wealth of low volatility stocks
> alloc <- matrix(integer(nrows*nstocks), ncol=nstocks)
> alloc[volat <= medianv] <- 1
> alloc <- rutils::lagit(alloc)
> retlow <- rowSums(alloc*retp, na.rm=TRUE)
> wealth_lovol <- exp(cumsum(retlow))
> # Calculate the wealth of high volatility stocks
> alloc <- matrix(integer(nrows*nstocks), ncol=nstocks)
> alloc[volat > medianv] <- 1
> alloc <- rutils::lagit(alloc)
> rethigh <- rowSums(alloc*retp, na.rm=TRUE)
> wealth_hivol <- exp(cumsum(rethigh))
```




```
> # Combined wealth
> wealthv <- cbind(wealth_lovol, wealth_hivol)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Low Volatility", "High Volatility")
> wealthv <- log(wealthv)
> # Calculate the Sharpe and Sortino ratios
> retvol <- rutils::diffit(wealthv)
> sqrt(252)*sapply(retvol, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Low and High Volatility Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

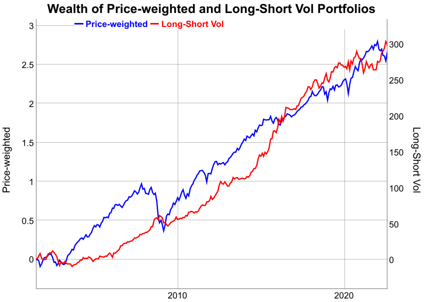
Long-Short Stock Volatility Strategy

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

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

The *Long-Short Volatility* strategy has higher risk-adjusted returns than the price-weighted portfolio.

```
> # Calculate the volatilities of the low and high volatility stocks
> volat <- HighFreq::run_var(retvol, lambda=lambda)
> volat <- sqrt(volat)
> volat[1:2, ] <- 1
> colnames(volat) <- c("Low Volatility", "High Volatility")
> # Multiply the high volatility portfolio returns by a factor
> factv <- volat[, 1]/volat[, 2]
> factv <- rutils::lagit(factv)
> # Calculate the long-short volatility returns
> retls <- (retlow - factv*rethigh)
> wealthls <- exp(cumsum(retls))
> # Combined wealth
> wealthv <- cbind(wealthfs, wealthls)
> wealthv <- xts::xts(wealthv, datev)
> colnamev <- c("Price-weighted", "Long-Short Vol")
> colnames(wealthv) <- colnamev
> wealthv <- log(wealthv)
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(rutils::diffit(wealthv),
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot of log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(wealthv[endd],
+   main="Wealth of Price-weighted and Long-Short Vol Portfolios") %
+   dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+   dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+   dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeWi
+   dySeries(name=colnamev[2], axis="y2", label=colnamev[2], strokeW
+   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: $w_i^{\prime}$.

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

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

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["2000/"]
> 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(weights)
> sort(weights)
```

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(weights)
> 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(weights)
> 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)
> nstocks <- 10
> symbolb <- names(head(perfstat, nstocks))
> symbolw <- names(tail(perfstat, nstocks))
> # Calculate equal weights for the best and worst performing stocks
> weightv <- numeric(NCOL(retp))
> names(weights) <- 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(weights)
> # Scale weights so sum of squares is equal to 1
> weightv <- weightv/sqrt(sum(weightv^2))
> # Calculate the momentum portfolio 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.

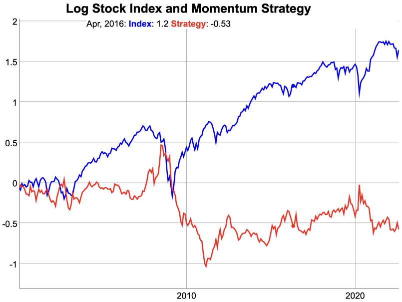
```
> # Objective function equal to Kelly ratio
> objfun <- function(retp) {
+   varv <- var(retp)
+   if (varv > 0) mean(retp)/varv else 0
+ }  # end objfun
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> npts <- NROW(endd)
> # Perform loop over 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, nstocks))
+   symbolw <- names(tail(perfstat, nstocks))
+   # Calculate the momentum weights
+   weightv <- numeric(NCOL(retp))
+   names(weights) <- 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 momentum portfolio 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")
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> 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="weeks", n
+   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 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, nstocks))
+     symbolw <- names(tail(perfstat, nstocks))
+     # Calculate the momentum weights
+     weightv <- numeric(NCOL(retp))
+     names(weights) <- 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 momentum portfolio returns
+     drop(retp[(endd[ep]+1):endd[ep+1], ] %*% weightv)
+   })  # end lapply
+   pnls <- rutils::do_call(c, pnls)
+   pnls
+ }  # end btmomtop
```

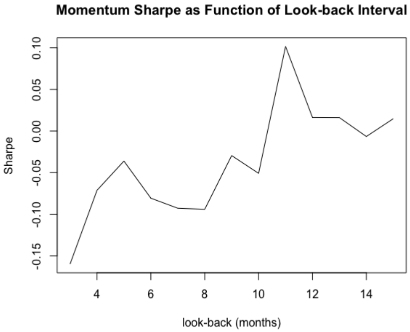
Optimization of Momentum Strategy Parameters

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="weeks")
> pnll <- lapply(look_backs, btmomtop, retp=retp, endd=endd, objfu
> # 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

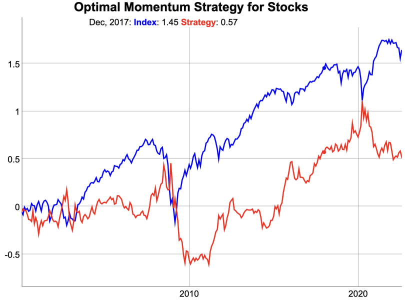
The best stock momentum strategy underperforms the index because of a poor choice of the model type.

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]]
> 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="weeks'
+   bid_offer=0.0, endd=rutils::calc_endpoints(retp, interval=rebal
+   # 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 momentum portfolio 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.

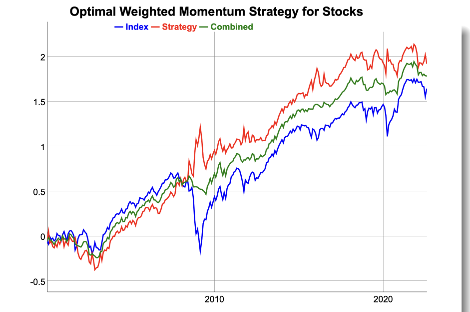
The advantage of the momentum strategy is that it has a low correlation to stocks, so it can provide significant risk diversification when combined with stocks.

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=2) %>%
```

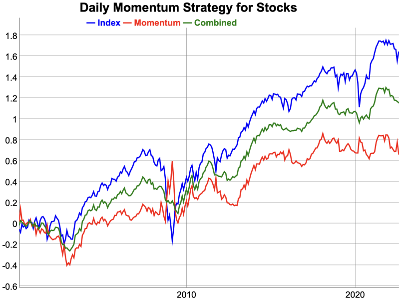
Momentum Strategy With Daily Rebalancing

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 package *roll* contains extremely fast functions for calculating rolling aggregations using compiled C++ code.

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)
> head(varm)
> # Calculate the trailing Kelly ratio
> perfstat <- HighFreq::roll_mean(retp, look_back=look_back)
> weightv <- perfstat/varm
> sum(is.na(weights))
> weightv <- weightv/sqrt(rowSums(weightv^2))
> weightv <- rutils::lagit(weights)
> # 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(weights)))
> 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")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Momentum Strategy for Stocks") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Daily Momentum Strategy Functional

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

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

A momentum strategy with *daily rebalancing* requires more computations so compiled C++ functions are preferred to apply() loops.

The package *roll* contains extremely fast functions for calculating rolling aggregations using compiled C++ code.

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

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, look_back=252, trend=1, bid_offer=0.0
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   # Calculate the trailing variance
+   varm <- HighFreq::roll_var(retp, look_back=look_back)
+ # Calculate the trailing Kelly ratio
+   perfstat <- HighFreq::roll_mean(retp, look_back=look_back)
+   weightv <- perfstat/varm
+   weightv <- weightv/sqrt(rowSums(weightv^2))
+   weightv <- rutils::lagit(weights)
+   # Calculate the momentum profits and losses
+   pnls <- trend*rowSums(weightv*retp)
+   # Calculate the transaction costs
+   costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weights)))
+   (pnls - costs)
+ }  # end btmomdaily
```

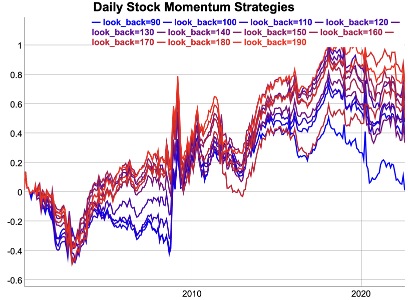
Multiple Daily Stock Momentum Strategies

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

The best performing daily stock momentum strategies are with *look-back* parameters between 140 and 190 days.

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

```
> # Simulate multiple daily stock momentum strategies
> look_backs <- seq(90, 190, by=10)
> pnls <- sapply(look_backs, 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("look_back=", look_backs)
> 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=300)
> # 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, its held for several days and then liquidated. So several portfolios are held at the same time.

This is equivalent to averaging the portfolio weights over several days from 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 optimal length of the *holding period* 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, look_back=252, holdp=5, trend=1,
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   # Calculate the trailing variance
+   varm <- HighFreq::roll_var(retp, look_back=look_back)
+   # Calculate the trailing Kelly ratio
+   perfstat <- HighFreq::roll_mean(retp, look_back=look_back)
+   weightv <- perfstat/varm
+   weightv <- weightv/sqrt(rowSums(weightv^2))
+   # Average the weights over holding period
+   weightv <- HighFreq::roll_mean(weightv, look_back=holdp)
+   weightv <- rutils::lagit(weights)
+   # Calculate the momentum profits and losses
+   pnls <- trend*rowSums(weightv*retp)
+   # Calculate the transaction costs
+   costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weights)))
+   (pnls - costs)
+ }  # end btmomdailyhold
```

Multiple Daily Momentum Strategies with Holding Period

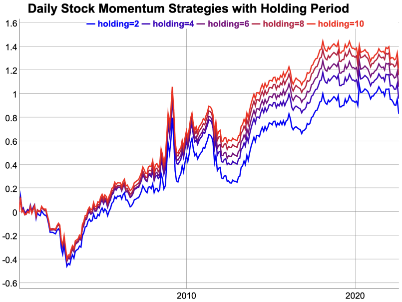
Multiple daily momentum strategies can be backtested by calling the function btmomdaily() in a loop over a vector of holding periods.

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

The daily momentum strategy with a longer holding period performs better than with a daily holding.

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.

```
> # Perform sapply loop over holding periods
> holdpv <- seq(2, 11, by=2)
> pnls <- sapply(holdpv, btmomdailyhold, look_back=180, 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("holding=", holdpv)
> 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=300)
> # 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")
```

Mean Reverting Stock Momentum Strategies

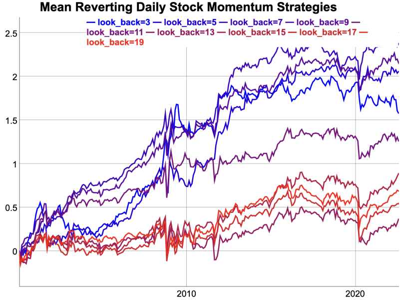
Multiple *mean reverting* stock momentum strategies can be backtested by calling the function `btmomdaily()` in a loop over a vector of *look-back* parameters.

The *mean reverting* momentum strategies for the stock constituents perform the best for short *look-back* parameters.

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

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

```
> # Perform sapply loop over look_backs
> look_backs <- seq(3, 20, by=2)
> pnls <- sapply(look_backs, 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("look_back=", look_backs)
> 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=300)
> # 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")
```

draft: Online Daily Momentum Strategy Functional

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

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

A momentum strategy with *daily rebalancing* requires more computations so compiled C++ functions are preferred to apply() loops.

The package *roll* contains extremely fast functions for calculating rolling aggregations using compiled C++ code.

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

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

The function btmomrun() 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
> btmomrun <- function(retp, lambda=0.9, trend=1, holdp=5, bid_offer
+   stopifnot("package:quantmod" %in% search() || require("quantmod"
+   # Calculate the trailing variance
+   varm <- HighFreq::run_var(retp, lambda=lambda)
+   # Calculate the trailing Kelly ratio
+   perfstat <- HighFreq::run_mean(retp, lambda=lambda)
+   weightv <- perfstat/varm
+   weightv[varm == 0] <- 0
+   weightv <- weightv/sqrt(rowSums(weightv^2))
+   # Average the weights over holding period
+   weightv <- HighFreq::roll_mean(weightv, look_back=holdp)
+   weightv <- rutils::lagit(weights)
+   # Calculate the momentum profits and losses
+   pnls <- trend*rowSums(weightv*retp)
+   # Calculate the transaction costs
+   costs <- 0.5*bid_offer*rowSums(abs(rutils::diffit(weights)))
+   (pnls - costs)
+ }  # end btmomrun
```

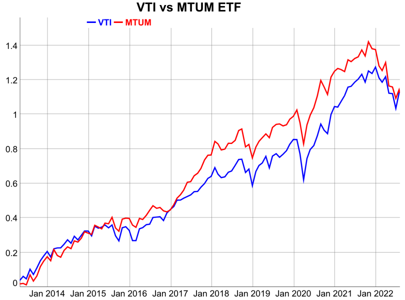
The MTUM Momentum ETF

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.

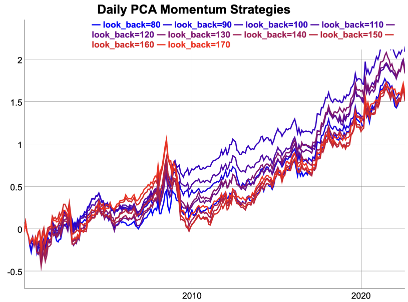
```
> # Calculate the PCA weights for scaled returns
> retscaled <- lapply(retp, function(x) x/sd(x))
> retscaled <- do.call(cbind, retscaled)
> pcad <- eigen((t(retscaled) %*% retscaled)/(nrows-1))
> pcaw <- pcad$vectors
> rownames(pcaw) <- colnames(retp)
> sort(-pcaw[, 1], decreasing=TRUE)
> sort(pcaw[, 2], decreasing=TRUE)
> round((t(pcaw) %*% pcaw)[1:5, 1:5], 4)
> # Calculate the PCA time series from stock returns using PCA weig
> retspca <- retscaled %*% pcaw
> round((t(retspca) %*% retspca)[1:5, 1:5], 4)
> retspca <- xts::xts(retspca, order.by=datev)
> # Calculate the PCA using prcomp()
> pcad <- prcomp(retscaled, center=FALSE, scale=FALSE)
> all.equal(pcad$x, retspca, check.attributes=FALSE)
> library(microbenchmark)
> summary(microbenchmark(
+   prcomp=prcomp(retscaled, center=FALSE, scale=FALSE),
+   eigen=eigen((t(retscaled) %*% retscaled)/(NROW(retscaled)-1)),
+   times=10))[, c(1, 4, 5)]
```

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 look-backs
> dimax <- 11
> look_backs <- seq(80, 170, by=10)
> pnll <- mclapply(look_backs, btmomdaily, retp=retspca[, 1:dimax],
+    mc.cores=ncores)
> pnll <- lapply(pnll, function(pnl) indeksd*pnl/sd(pnl))
> pnls <- do.call(cbind, pnll)
> colnames(pnls) <- paste0("look_back=", look_backs)
> pnls <- xts::xts(pnls, datev)
> # Plot Sharpe ratios of momentum strategies
> sharper <- sqrt(252)*sapply(pnls, function(pnl) mean(pnl)/sd(pnl))
> plot(x=look_backs, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="Sharpe")
```




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

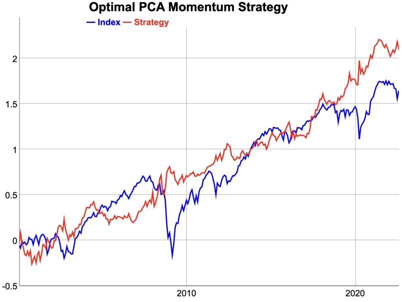
Optimal PCA Momentum Strategy

The PCA momentum strategy using only the lowest order principal components outperforms the index.

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

The best performing PCA momentum strategy has a relatively short look-back interval of only 100 days, so it's able to quickly adjust to changes in market direction.

```
> # Calculate best pnls of PCA momentum strategy
> whichmax <- which.max(sharper)
> look_backs[whichmax]
> pnls <- pnll[[whichmax]]
> 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 PCA momentum strategy
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal PCA Momentum Strategy") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   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.

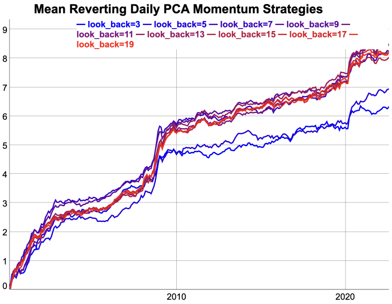
The *mean reverting* PCA momentum strategies perform the best for short look-back intervals of about 5 - 10 days.

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

The backtest simulations don't account for transaction costs, so they should be interpreted only as an indication of potential performance.

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

```
> # Perform sapply loop over look_backs
> look_backs <- seq(3, 20, by=2)
> pnls <- sapply(look_backs, btmomdaily, trend=(-1),
+   retp=retspca[, (dimax+1):NCOL(retspca)])
> colnames(pnls) <- paste0("look_back=", look_backs)
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) indeksd*pnl/sd(pnl))
> pnls <- xts::xts(pnls, datev)
```



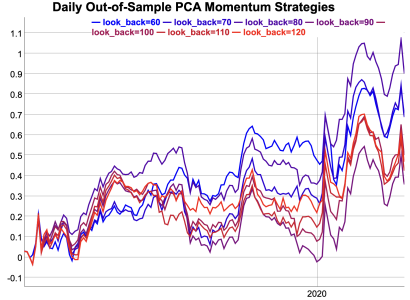

```
> # Plot dygraph of daily PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Mean Reverting Daily PCA Momentum Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)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

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
> retscaled <- lapply(retp, function(x) x[outsample]/sd(x[insample])
> retscaled <- do.call(cbind, retscaled)
> retspca <- xts::xts(retscaled %*% pcad$vectors, order.by=datev[out
> # Simulate daily PCA momentum strategies for multiple look-backs
> dimax <- 11
> look_backs <- seq(60, 120, by=10)
> pnll <- mclapply(look_backs, btmomdaily,
+    retp=retspca[, 1:dimax], mc.cores=ncores)
> pnll <- lapply(pnll, function(pnl) indeksd*pnl/sd(pnl))
> pnls <- do.call(cbind, pnll)
> colnames(pnls) <- paste0("look_back=", look_backs)
> 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=look_backs, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="Sharpe")
```



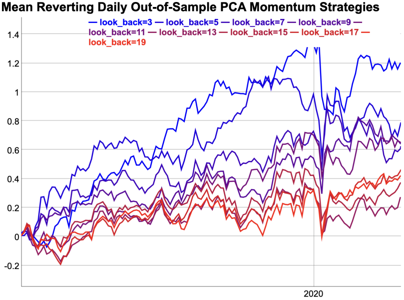

```
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retspca, interval="weeks")
> # Plot dygraph of daily out-of-sample PCA momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Daily Out-of-Sample 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.

```
> # Perform sapply loop over look_backs
> look_backs <- seq(3, 20, by=2)
> pnls <- sapply(look_backs, btmomdaily, trend=(-1),
+   retp=retspca[, (dimax+1):NCOL(retspca)])
> colnames(pnls) <- paste0("look_back=", look_backs)
> # Scale the momentum volatility to the equal weight index
> pnls <- apply(pnls, MARGIN=2, function(pnl) indeksd*pnl/sd(pnl))
> 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=look_backs, y=sharper, t="l",
+   main="PCA Momentum Sharpe as Function of Look-back Interval",
+   xlab="look-back (months)", ylab="Sharpe")
```




```
> # Calculate a vector of monthly end points
> endd <- rutils::calc_endpoints(retspca, 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.

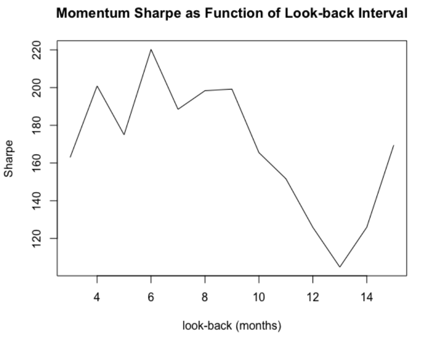
The dependence on the length of the *look-back interval* is an example of the *bias-variance tradeoff*. If the *look-back interval* is too short, the past performance estimates have high *variance*, but if the *look-back interval* is too long, the past estimates have high *bias*.

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="weeks")
> npts <- NROW(endd)
> # Perform backtests for vector of look-back intervals
> look_backs <- seq(3, 15, 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

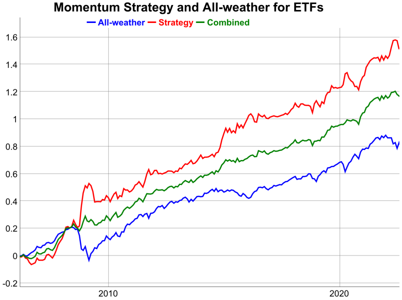
The momentum strategy for ETFs produces a higher absolute return and also a higher Sharpe ratio than the static *All-Weather* portfolio.

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
> weightsaw <- 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="Momentum Strategy and All-weather for ETFs") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

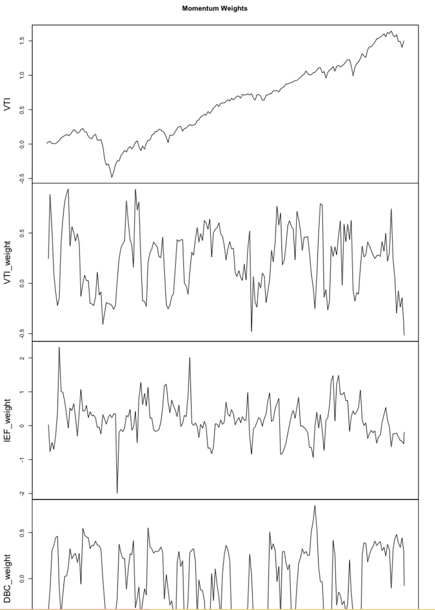
Time Series of Momentum Portfolio Weights

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*.

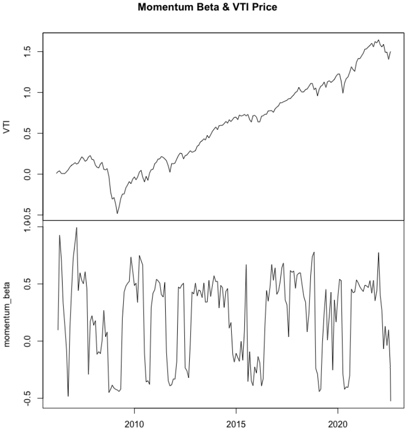
```
> # Calculate the momentum weights
> look_back <- look_backs[whichmax]
> weightv <- lapply(1:(npts-1), function(ep) {
+   # Select the look-back returns
+   startp <- endd[max(1, ep-look_back)]
+   retsis <- retp[startp:endd[ep], ]
+   # Calculate weights proportional to performance
+   perfstat <- sapply(retsis, objfun)
+   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="")
```

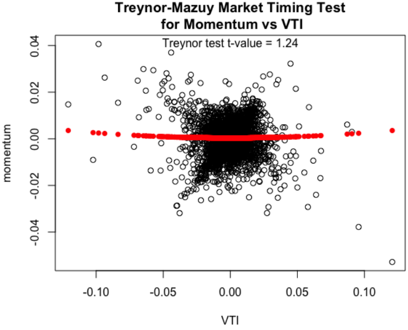


Momentum Strategy Market Timing Skill

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))
```



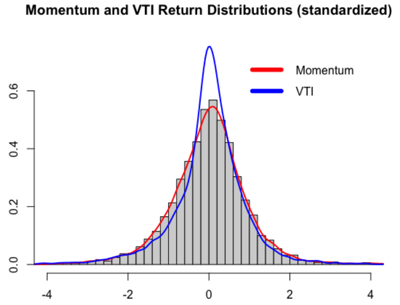
Skewness of Momentum Strategy Returns

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=FAL
> # 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"))
```

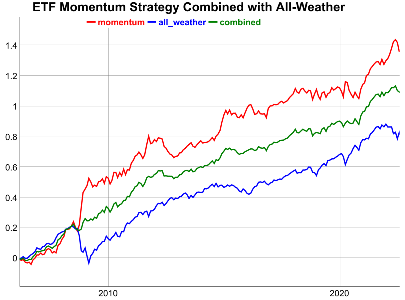
Combining Momentum with the *All-Weather* Portfolio

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
> dygraphs::dygraph(cumsum(wealthv)[endd], main="ETF Momentum Strate
+   dyOptions(colors=c("red", "blue", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Or
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("green", "blue", "red")
> quantmod::chart_Series(wealthv, theme=plot_theme,
+        name="ETF Momentum Strategy Combined with All-Weather")
> legend("topleft", legend=colnames(wealthv),
+   inset=0.1, bg="white", lty=1, lwd=6,
+   col=plot_theme$col$line.col, bty="n")
```

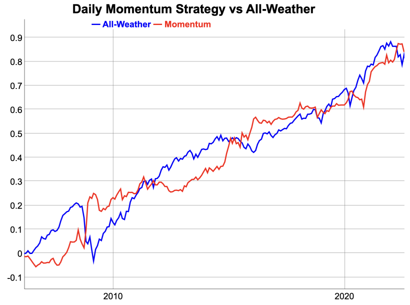
Momentum Strategy for ETFs With Daily Rebalancing

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 package *roll* contains extremely fast functions for calculating rolling aggregations using compiled C++ code.

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
> perfstat <- HighFreq::roll_mean(retp, look_back=look_back)
> weightv <- perfstat/varm
> weightv[varm == 0] <- 0
> sum(is.na(weights))
> weightv <- weightv/sqrt(rowSums(weightv^2))
> weightv <- rutils::lagit(weights)
> # 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(weights)))
> pnls <- (pnls - costs)
```

```
> # Define all-weather benchmark
> weightsaw <- c(0.30, 0.55, 0.15)
> all_weather <- retp %*% weightvaw
> # Scale the momentum volatility to all_weather
> pnls <- sd(all_weather)*pnls/sd(pnls)
> # Calculate the wealth of momentum returns
> wealthv <- xts::xts(cbind(all_weather, pnls), order.by=datev)
> colnames(wealthv) <- c("All-Weather", "Momentum")
> cor(wealthv)
> # 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 dygraph of the momentum strategy returns
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Daily Momentum Stra
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

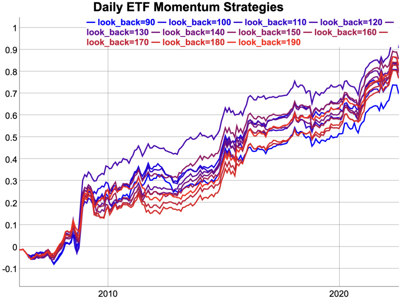
Multiple Daily ETF Momentum Strategies

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

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

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

```
> # Simulate a daily ETF momentum strategy
> source("/Users/jerzy/Develop/lecture_slides/scripts/back_test.R")
> pnls <- btmomdaily(retp=retp, look_back=152, bid_offer=bid_offer)
> # Perform sapply loop over look_backs
> look_backs <- seq(90, 190, by=10)
> pnls <- sapply(look_backs, btmomdaily,
+   retp=retp, bid_offer=bid_offer)
> # Scale the momentum volatility to all_weather
> pnls <- apply(pnls, MARGIN=2,
+   function(pnl) sd(all_weather)*pnl/sd(pnl))
> colnames(pnls) <- paste0("look_back=", look_backs)
> pnls <- xts::xts(pnls, datev)
> tail(pnls)
```




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

draft: Daily rank simple Momentum Strategy with Holding Period

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

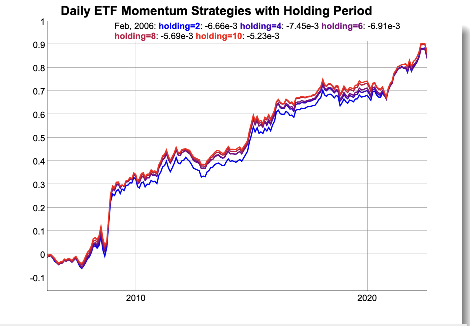
```
> # If trend=(-1) then it backtests a mean reverting strategy
> btmomdaily <- function(retp, look_back=252, holdp=5, trend=1, bid_offer=0.0, ...) {
+   stopifnot("package:quantmod" %in% search() || require("quantmod", quietly=TRUE))
+
+   posit <- matrixStats::rowRanks(retp)
+   posit <- (posit - rowMeans(posit))
+   posit <- HighFreq::lagit(posit, lagg=1)
+   trend*rowMeans(posit*retp)
+
+ }  # end btmomdaily
>
> # Load ETF data
> symbolv <- rutils::etfenv$symbolv
> symbolv <- symbolv[!(symbolv %in% c("TLT", "IEF", "MTUM", "QUAL", "VLUE", "USMV"))]
> retp <- rutils::etfenv$returns[, symbolv]
> retp[1, is.na(retp[1, ])] <- 0
> retp <- zoo::na.locf(retp, na.rm=FALSE)
>
> # Load S&P500 data
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RData")
> retp <- retp["2000/"]
> retp[1, is.na(retp[1, ])] <- 0
> retp <- zoo::na.locf(retp, na.rm=FALSE)
> nstocks <- NCOL(retp)
> retp <- retp[, !(retp[nstocks %/% 10, ] == 0)]
>
>
> pnls <- btmomdaily(retp=retp, trend=(-1))
> pnls <- xts::xts(pnls, datev)
> colnames(pnls) <- "PnL"
> dygraphs::dygraph(cumsum(pnls), main="Daily Momentum Strategy") %>%
+   dyOptions(colors="blue", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
>
```

draft: Backtesting Daily rank simple Momentum Strategy with Holding Period

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

The daily momentum strategies with a holding period perform much better.

```
> # Perform sapply loop over look_backs
> look_backs <- seq(50, 300, by=50)
> pnls <- sapply(look_backs, btmomdaily,
+   retp=retp, bid_offer=bid_offer)
> colnames(pnls) <- paste0("look_back=", look_backs)
> pnls <- xts::xts(pnls, datev)
>
> # Perform sapply loop over holding periods
> holdpv <- seq(2, 11, by=2)
> pnls <- sapply(holdpv, btmomdaily, look_back=120, retp=retp)
> colnames(pnls) <- paste0("holding=", holdpv)
> pnls <- xts::xts(pnls, datev)
```




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

depr: Backtesting the Momentum Strategy

Backtesting is simulating the performance of a investment strategy on historical data.

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

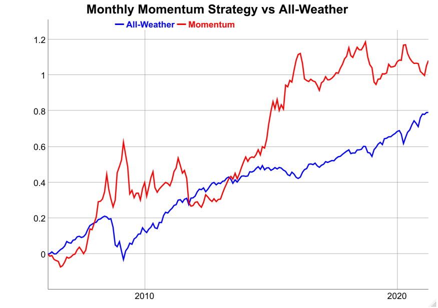
Backtesting is performed by *training* the model on past data and *testing* it on future out-of-sample data.

The *training* data is specified by the *look-back* intervals (`past`), and the model forecasts are applied to the future data defined by the *look-forward* intervals (`future`).

The out-of-sample *momentum* strategy returns can be calculated by multiplying the `future` returns by the forecast *ETF* portfolio weights.

The momentum returns are lagged so that they are attached to the end of the future interval, instead of at its beginning.

```
> # Calculate future out-of-sample performance
> retsos <- apply(look_fwds, 1, function(ep) {
+   sapply(retp[ep[1]:ep[2]], sum)
+ })  # end sapply
> retsos <- t(retsos)
> retsos[is.na(retsos)] <- 0
> tail(retsos)
```




```
> # Calculate the momentum pnls
> pnls <- rowSums(weightv*retsos)
> # Lag the future and momentum returns to proper dates
> retsos <- rutils::lagit(retsos)
> pnls <- rutils::lagit(pnls)
> # The momentum strategy has low correlation to stocks
> cor(pnls, retsos)
> # Define all-weather benchmark
> weightsaw <- c(0.30, 0.55, 0.15)
> all_weather <- retsos %*% weightvaw
> # Calculate the wealth of momentum returns
> wealthv <- xts::xts(cbind(all_weather, pnls), order.by=datev)
> colnames(wealthv) <- c("All-Weather", "Momentum")
> cor(wealthv)
> # Plot dygraph of the momentum strategy returns
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Monthly Momentum St
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

depr: Momentum Strategy for an *ETF* Portfolio

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

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

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

depr: Look-back and Look-forward Intervals

Performance aggregations are be calculated over a vector of overlapping in-sample *look-back* intervals attached at *end points*.

For example, aggregations at monthly *end points* over overlapping 12-month *look-back* intervals.

An example of a data aggregation are the cumulative past returns at each *end point*.

The variable look_back is equal to the number of *end points* in the *look-back* interval.

The *start points* are the *end points* lagged by the length of the *look-back* interval.

The *look-back* intervals are spanned by the vectors of *start points* and *end points*.

Performance aggregations are also be calculated over non-overlapping out-of-sample *look-forward* intervals.

The *look-forward* intervals should not overlap with the *look-back* intervals, in order to avoid data snooping.

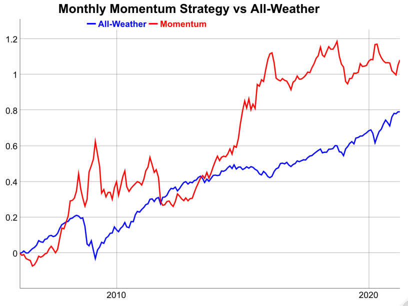
```
> # Define end of month end points
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> endd <- endd[-1]
> npts <- NROW(endd)
> datev <- zoo::index(retp)[endd]
> # Start points equal end points lagged by 12-month look-back inter
> look_back <- 12
> startp <- c(rep_len(1, look_back),
+   endd[1:(npts - look_back + 1)])
> # Calculate matrix of look-back intervals
> look_backs <- cbind(startp, endd)
> colnames(look_backs) <- c("start", "end")
> # Calculate matrix of look-forward intervals
> look_fwds <- cbind(endd + 1, rutils::lagit(endd, -1))
> look_fwds[npts, ] <- endd[npts]
> colnames(look_fwds) <- c("start", "end")
> # Inspect the intervals
> head(cbind(look_backs, look_fwds))
> tail(cbind(look_backs, look_fwds))
```

depr: Backtest of the Momentum Strategy

The *out-of-sample* momentum strategy returns are calculated by multiplying the weights times the future returns.

The *transaction costs* are equal to half the *bid-offer spread* δ times the absolute value of the traded dollar amounts of the *risky assets*.

```
> # Calculate the momentum profits and losses (pnls)
> pnls <- rowSums(weightv*retsos)
> # Lag the momentum returns and weights
> # to correspond with end of future interval
> pnls <- rutils::lagit(pnls)
> weightv <- rutils::lagit(weights)
> # bid_offer equal to 10 bps for liquid ETFs
> bid_offer <- 0.001
> # Calculate the transaction costs
> wealthv <- cumsum(pnls)
> costs <- 0.5*bid_offer*wealth*rowSums(abs(rutils::diffit(weights)
> wealthv <- cumsum(pnls - costs)
> datev <- zoo::index(retp[endd])
> wealthv <- xts::xts(wealthv, datev)
```




```
> # Define all-weather benchmark
> weightsaw <- c(0.30, 0.55, 0.15)
> retsaw <- retp %*% weightvaw
> wealthaw <- cumsum(retsaw)
> wealthaw <- xts::xts(wealthaw[endd], datev)
> # Plot of Momentum strategy and benchmark
> wealthv <- cbind(wealthv, wealthaw)
> colnames(wealthv) <- c("Momentum Strategy", "Benchmark")
> dygraphs::dygraph(wealthv, main="Momentum Strategy") %>%
+   dyAxis("y", label="Benchmark", independentTicks=TRUE) %>%
+   dyAxis("y2", label="Momentum Strategy", independentTicks=TRUE) %
+   dySeries(name="Momentum Strategy", axis="y2", label="Momentum St
+   dySeries(name="Benchmark", axis="y", label="Benchmark", strokeWi
+   dyLegend(show="always", width=300)
```

depr: Backtesting Functional for ETF Momentum Strategy

```
> # Define backtest functional
> btmomweight <- function(retp,
+                 objfun=function(retp) (sum(retp)/sd(retp)),
+                 look_back=12, rebalf="weeks", bid_offer=0.0,
+                 endd=rutils::calc_endpoints(retp, interval=rebalf)[-1],
+                 with_weights=FALSE, ...) {
+   stopifnot("package:rutils" %in% search() || require("rutils", quietly=TRUE))
+   # Define look-back and look-forward intervals
+   npts <- NROW(endd)
+   startp <- c(rep_len(1, look_back), endd[1:(npts-look_back)])
+   # Calculate look-back intervals
+   look_backs <- cbind(startp, endd)
+   # Calculate look-forward intervals
+   look_fwds <- cbind(endd + 1, rutils::lagit(endd, -1))
+   look_fwds[npts, ] <- endd[npts]
+   # Calculate past performance over look-back intervals
+   perfstat <- t(apply(look_backs, 1, function(ep) sapply(retp[ep[1]:ep[2]], objfun)))
+   perfstat[is.na(perfstat)] <- 0
+   # Calculate future performance
+   retsos <- t(apply(look_fwds, 1, function(ep) sapply(retp[ep[1]:ep[2]], sum)))
+   retsos[is.na(retsos)] <- 0
+   # Scale weights so sum of squares is equal to 1
+   weightv <- perfstat
+   weightv <- weightv/sqrt(rowSums(weightv^2))
+   weightv[is.na(weights)] <- 0  # Set NA values to zero
+   # Calculate the momentum profits and losses
+   pnls <- rowSums(weightv*retsos)
+   # Calculate the transaction costs
+   costs <- 0.5*bid_offer*cumsum(pnls)*rowSums(abs(rutils::diffit(weights)))
+   pnls <- (pnls - costs)
+   if (with_weightv)
+     rutils::lagit(cbind(pnls, weightv))
+   else
+     rutils::lagit(pnls)
+ }  # end btmomweight
```

Portfolio Optimization Strategy

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

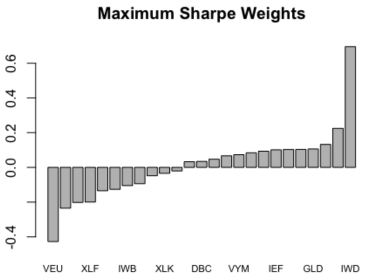
The *portfolio optimization* strategy consists of:

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

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

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

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

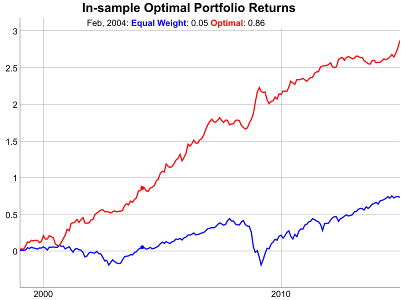



```
> # Maximum Sharpe weights in-sample interval
> retsis <- retp["/2014"]
> invmat <- MASS::ginv(cov(retsis))
> weightv <- invmat %*% colMeans(retx["/2014"])
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weights) <- colnames(retp)
> # Plot portfolio weights
> x11(width=6, height=5)
> par(mar=c(3,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

Portfolio Optimization Strategy In-Sample

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

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



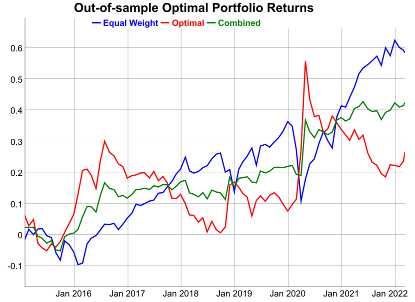

```
> # Plot cumulative portfolio returns
> pnls <- cbind(indeks, insample)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="In-sample Optimal Portfolio Returns") %>%
+   dyOptions(colors=c("blue",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

Portfolio Optimization Strategy Out-of-Sample

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

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

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



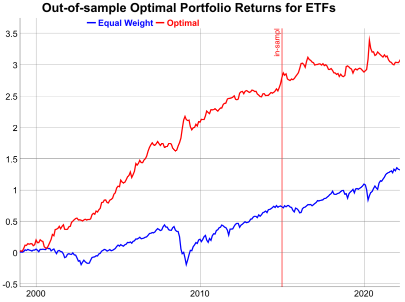

```
> # Plot cumulative portfolio returns
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Out-of-sample Optimal Portfolio Returns") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(width=300)
```

Portfolio Optimization Strategy for ETFs

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

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

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




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

Regularized Inverse of Singular Covariance Matrix

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

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

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

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

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

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

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

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

Dimension Reduction of the Covariance Matrix

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

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

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

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

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

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

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

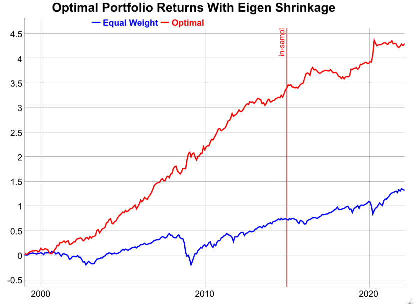
```
> # Calculate in-sample covariance matrix
> covmat <- cov(retsis)
> eigend <- eigen(covmat)
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> # Calculate dimension reduction inverse of covariance matrix
> dimax <- 3
> covinv <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])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

Portfolio Optimization for ETFs with Dimension Reduction

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

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

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




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

Portfolio Optimization With Return Shrinkage

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

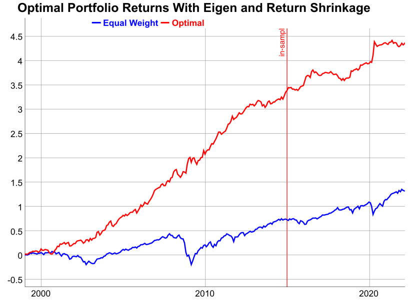
$$r_i' = (1 - \alpha)\, r_i + \alpha\, \bar{r}$$

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

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

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

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



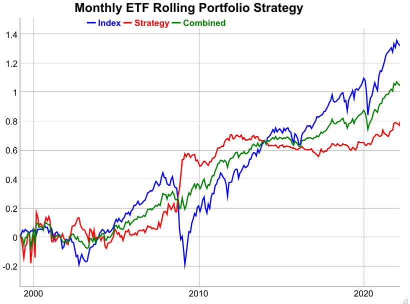

```
> # Plot cumulative portfolio returns
> pnls <- rbind(insample, outsample)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Optimal Portfolio Returns With Eigen and Return Shrinkage"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

Rolling Portfolio Optimization Strategy

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

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

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



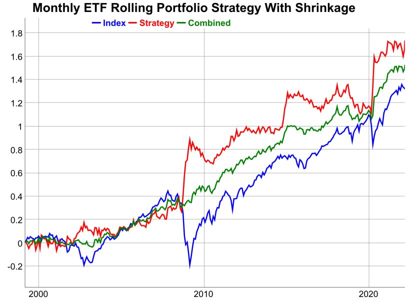

```
> # Plot dygraph of rolling ETF portfolio strategy
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino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

Rolling Portfolio Strategy With Dimension Reduction

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

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

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



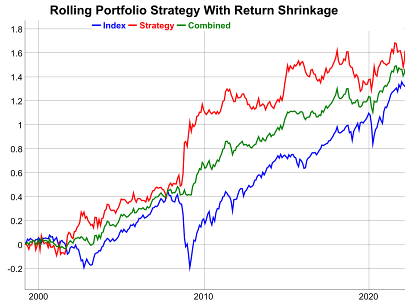

```
> # Plot dygraph of rolling ETF portfolio strategy
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino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

Rolling Portfolio Strategy With Return Shrinkage

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

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

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



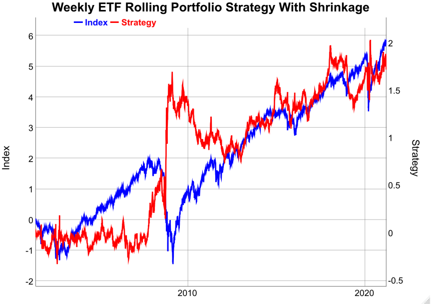

```
> # Plot dygraph of rolling ETF portfolio strategy
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino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

draft: Weekly ETF Rolling Portfolio Strategy With Shrinkage

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

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

```
> # Define weekly dates
> weeks <- seq.Date(from=as.Date("2001-01-01"), to=as.Date("2021-04-
> # Perform loop over monthly dates
> look_back <- 21
> dimax <- 3
> pnls <- lapply((look_back+1):(NROW(weeks)-1), function(ep) {
+   # Define in-sample and out-of-sample returns
+   insample <- (datev > weeks[ep-look_back]) & (datev < weeks[ep]]
+   outsample <- (datev > weeks[ep]) & (datev < weeks[ep+1])
+   retsis <- retp[insample]
+   retsos <- retp[outsample]
+   # Calculate regularized inverse of covariance matrix
+   # invmat <- MASS::ginv(cov(retsis))  # if VXX and SVXY are inc
+   invmat <- HighFreq::calc_inv(cov(retsis), dimax=dimax)
+   weightv <- invmat %*% colMeans(retsis - riskf)
+   weightv <- drop(weightv/sqrt(sum(weightv^2)))
+   # Calculate portfolio pnls out-of-sample
+   xts::xts(retsos %*% weightv, zoo::index(retsos))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```




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

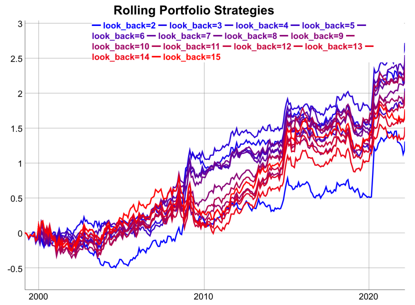
Function for Rolling Portfolio Optimization Strategy

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

Rolling Portfolio Optimization With Different Look-backs

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

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

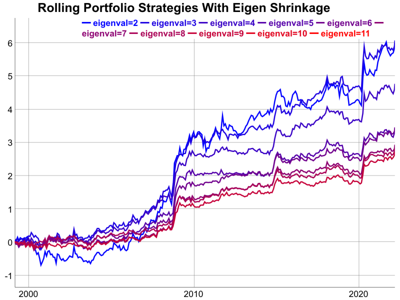



```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Plot EWMA strategies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

Rolling Portfolio Optimization With Different Dimension Reduction

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

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



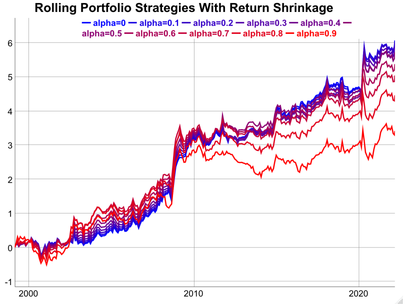

```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies With Dimension Reduction") %>
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)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

Rolling Portfolio Optimization With Different Return Shrinkage

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

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

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

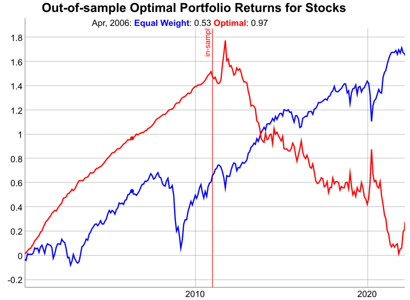



```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies With Return Shrinkage") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=300)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

Portfolio Optimization Strategy for Stocks

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

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




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

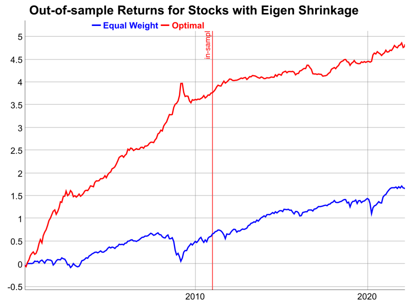
```
> # Combine in-sample and out-of-sample returns
> pnls <- rbind(insample, outsample)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(pnls[index(outsample)],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

Portfolio Optimization for Stocks with Dimension Reduction

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

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

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




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

Optimal Stock Portfolio Weights With Return Shrinkage

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

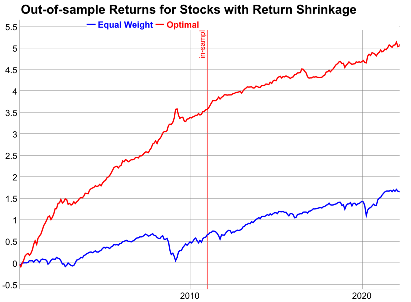
$$r_i' = (1 - \alpha)\, r_i + \alpha\, \bar{r}$$

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

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

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

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




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

Fast Covariance Matrix Inverse Using *RcppArmadillo*

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

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

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

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

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

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

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

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

  }  // end if

}  // end calc_inv
```

Portfolio Optimization Using *RcppArmadillo*

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

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

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

Strategy Backtesting Using *RcppArmadillo*

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

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

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

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

  // Return the strategy pnls
  return pnls;

}  // end back_test
```

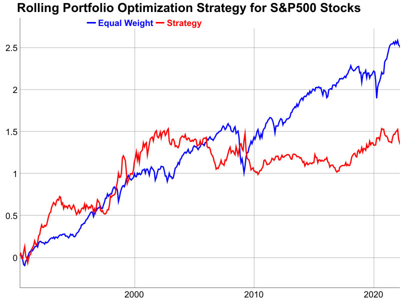
Rolling Portfolio Optimization Strategy for S&P500 Stocks

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

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

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

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

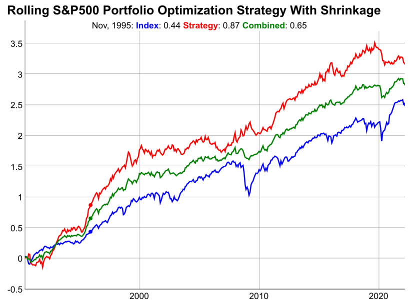



```
> # Calculate returns of equal weight portfolio
> indeks <- xts::xts(rowMeans(retp), datev)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls*sd(indeks)/
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> colnames(wealthv) <- c("Equal Weight", "Strategy")
> sqrt(252)*sapply(wealthv,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

Rolling Portfolio Optimization Strategy With Shrinkage

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

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




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

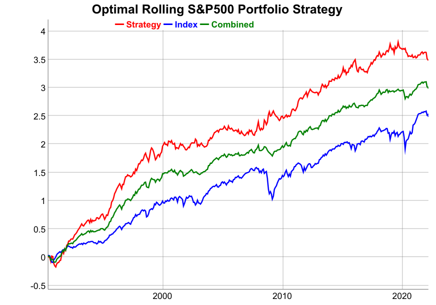
Determining Shrinkage Parameters Using Backtesting

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

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

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

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



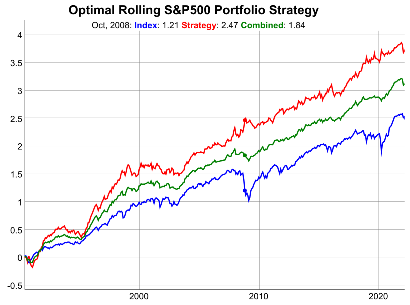

```
> plot(x=eigenvals, y=profilev, t="l",
+   main="Strategy PnL as Function of dimax",
+   xlab="dimax", ylab="pnl")
> whichmax <- which.max(profilev)
> dimax <- eigenvals[whichmax]
> pnls <- pnls[[whichmax]]
> pnls <- pnls*sd(indeks)/sd(pnls)
> # Plot cumulative strategy returns
> wealthv <- cbind(indeks,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

Determining Look-back Interval Using Backtesting

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

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

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




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

draft: Applications of Machine Learning to Finance

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

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

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

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

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

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

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

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