FRE7241 Algorithmic Portfolio Management
Lecture#7, Fall 2022

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

October 25, 2022

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*.

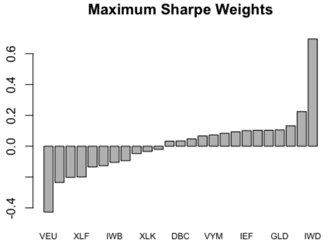
The *portfolio optimization* strategy consists of:

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 **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
> retsp <- rutils::etfenv$returns[, symbolv]
> nstocks <- NCOL(retsp)
> # retsp <- na.omit(retsp)
> retsp[1, is.na(retsp[1, ])] <- 0
> retsp <- zoo::na.locf(retsp, na.rm=FALSE)
> datev <- zoo::index(retsp)
> # Returns in excess of risk-free rate
> riskf <- 0.03/252
> retsx <- (retsp - riskf)
```

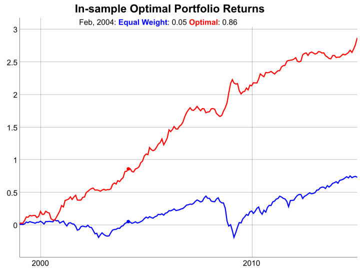



```
> # Maximum Sharpe weights in-sample interval
> retsis <- retsp["/2014"]
> invmat <- MASS::ginv(cov(retsis))
> weightv <- invmat %*% colMeans(retsx["/2014"])
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retsp)
> # Plot portfolio weights
> x11(width=6, height=5)
> par(mar=c(3,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

Portfolio Optimization Strategy In-Sample

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

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



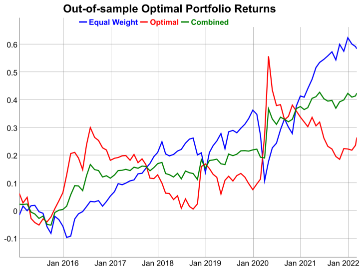

```
> # Plot cumulative portfolio returns
> pnls <- cbind(indeks, insample)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> endp <- rutils::calc_endpoints(pnls, interval="months")
> dygraphs::dygraph(cumsum(pnls)[endp], main="In-sample Optimal Port
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(width=500)
```

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 <- retsp["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")
> sqrt(252)*sapply(pnls, function(x) mean(x)/sd(x))
```



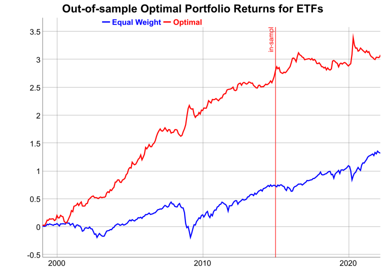

```
> # Plot cumulative portfolio returns
> endp <- rutils::calc_endpoints(pnls, interval="months")
> dygraphs::dygraph(cumsum(pnls)[endp], main="Out-of-sample Optimal
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(width=500)
```

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(retsx["/2014"])
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retsp)
> # Calculate in-sample portfolio returns
> insample <- xts::xts(retsis %*% weightv, zoo::index(retsis))
> # Calculate out-of-sample portfolio returns
> retsos <- retsp["2015/"]
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
```




```
> # Plot cumulative portfolio returns
> pnls <- rbind(insample, outsample)
> indeks <- xts::xts(rowMeans(retsp), datev)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> endp <- rutils::calc_endpoints(pnls, interval="months")
> dygraphs::dygraph(cumsum(pnls)[endp], main="Out-of-sample Optimal
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retsis[, 1])), label="in-sample", stroke
+   dyLegend(width=500)
```

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*).

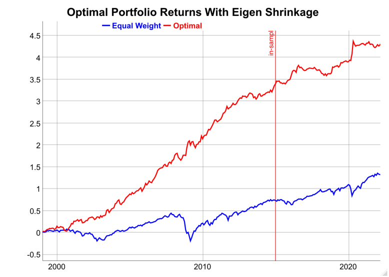
```
> # Calculate in-sample covariance matrix
> covmat <- cov(retsis)
> eigend <- eigen(covmat)
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> # Calculate dimension reduction inverse of covariance matrix
> dimax <- 3
> covinv <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
> # 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(weightv) <- colnames(retsp)
> # 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)[endp], main="Optimal Portfolio Retu
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retsis[, 1])), label="in-sample", stroke
+   dyLegend(width=500)
```

Portfolio Optimization With Return Shrinkage

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

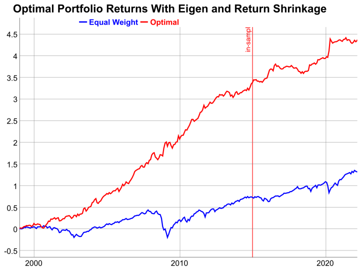
$$r_i^{\prime} = (1 - \alpha) \, r_i + \alpha \, \bar{r}$$

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
> retsxm <- rowMeans(retsx["/2014"])
> retsxis <- (1-alpha)*retsx["/2014"] + alpha*retsxm
> # Calculate portfolio weights
> weightv <- invmat %*% colMeans(retsxis)
> 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))
```

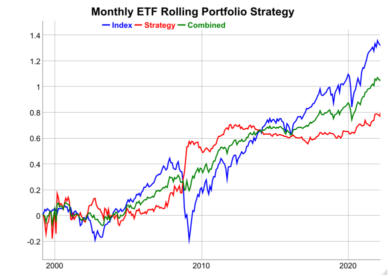



```
> # Plot cumulative portfolio returns
> pnls <- rbind(insample, outsample)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> dygraphs::dygraph(cumsum(pnls)[endp], main="Optimal Portfolio Retu
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retsis[, 1])), label="in-sample", stroke
+   dyLegend(width=500)
```

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
> endp <- rutils::calc_endpoints(retsp, interval="months")
> endp <- endp[endp > (nstocks+1)]
> npts <- NROW(endp)
> look_back <- 3
> startp <- c(rep_len(0, look_back), endp[1:(npts-look_back)])
> # Perform loop over end points
> pnls <- lapply(2:npts, function(ep) {
+     # Calculate the portfolio weights
+     insample <- retsx[startp[ep-1]:endp[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 <- retsp[(endp[ep-1]+1):endp[ep], ]
+     xts::xts(outsample %*% weightv, zoo::index(outsample))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```

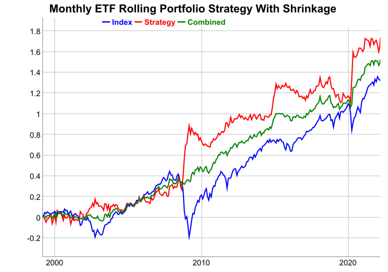
```
> # Plot dygraph of rolling ETF portfolio strategy
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Monthly ETF Rolling
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

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), endp[1:(npts-look_back)])
> # Perform loop over end points
> pnls <- lapply(2:npts, function(ep) {
+     # Calculate regularized inverse of covariance matrix
+     insample <- retsx[startp[ep-1]:endp[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 <- retsp[(endp[ep-1]+1):endp[ep], ]
+     xts::xts(outsample %*% weightv, zoo::index(outsample))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```



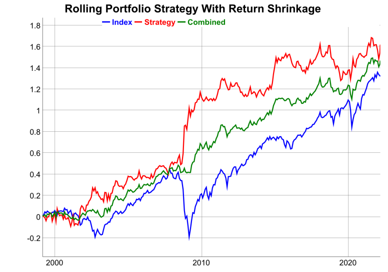

```
> # Plot dygraph of rolling ETF portfolio strategy
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Rolling Portfolio S
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

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 <- retsx[startp[ep-1]:endp[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 <- retsp[(endp[ep-1]+1):endp[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)[endp], main="Rolling Portfolio S
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

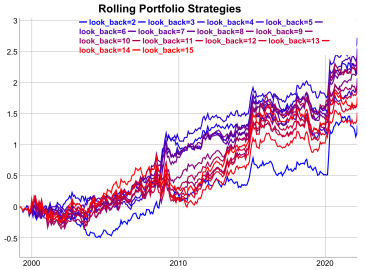
Function for Rolling Portfolio Optimization Strategy

```
> # Define backtest functional for rolling portfolio strategy
> roll_portf <- function(excess, # Excess returns
+                  returns, # Stock returns
+                  endp, # 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(endp)
+   startp <- c(rep_len(0, look_back), endp[1:(npts-look_back)])
+   pnls <- lapply(2:npts, function(ep) {
+     # Calculate regularized inverse of covariance matrix
+     insample <- excess[startp[ep-1]:endp[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[(endp[ep-1]+1):endp[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=retsx, returns=retsp, endp=endp,
+   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=retsp, excess=retsx, endp=endp, 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)]
```

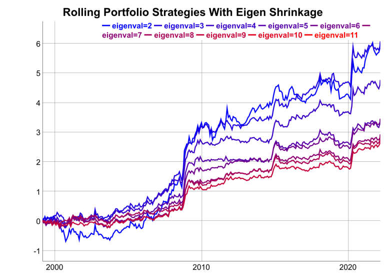



```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endp], main="Rolling Portfolio Stra
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot EWMA strategies with custom line colors
> 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(retsp)),
+   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=retsx,
+   returns=retsp, endp=endp, look_back=look_back)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("eigenval=", eigenvals)
> pnlsums <- sapply(pnls, sum)
> dimax <- eigenvals[which.max(pnlsums)]
```



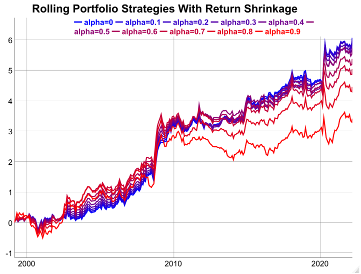

```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endp], main="Rolling Portfolio Stra
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot EWMA strategies with custom line colors
> 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(retsp)),
+   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=retsx,
+   returns=retsp, endp=endp, 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)]
```



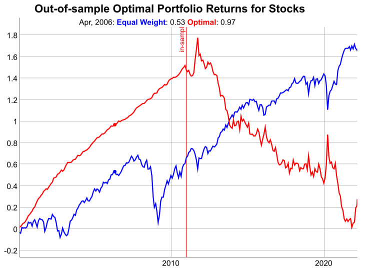

```
> # Plot dygraph of daily ETF momentum strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endp], main="Rolling Portfolio Stra
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot EWMA strategies with custom line colors
> 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(retsp)),
+   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
> retsp <- returns["2000/"]
> nstocks <- NCOL(retsp)
> retsp[1, is.na(retsp[1, ])] <- 0
> retsp <- zoo::na.locf(retsp, na.rm=FALSE)
> datev <- zoo::index(retsp)
> riskf <- 0.03/252
> retsx <- (retsp - riskf)
> retsis <- retsp["/2010"]
> retsos <- retsp["2011/"]
> # Maximum Sharpe weights in-sample interval
> covmat <- cov(retsis)
> invmat <- MASS::ginv(covmat)
> weightv <- invmat %*% colMeans(retsx["/2010"])
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retsp)
> # Calculate portfolio returns
> insample <- xts::xts(retsis %*% weightv, zoo::index(retsis))
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
> indeks <- xts::xts(rowMeans(retsp), datev)
```



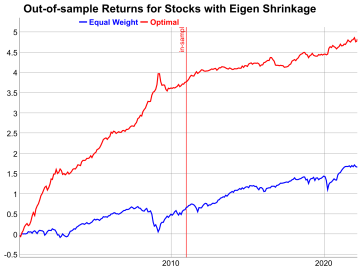

```
> # Combine in-sample and out-of-sample returns
> pnls <- rbind(insample, outsample)
> pnls <- pnls*sd(indeks)/sd(pnls)
> pnls <- cbind(indeks, pnls)
> colnames(pnls) <- c("Equal Weight", "Optimal")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(pnls[index(outsample)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot the cumulative portfolio returns
> endp <- rutils::calc_endpoints(pnls, interval="months")
> dygraphs::dygraph(cumsum(pnls)[endp], main="Out-of-sample Optimal
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retsis[, 1])), label="in-sample", stroke
+   dyLegend(width=500)
```

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(retsx["/2010"])
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retsp)
> # Calculate portfolio returns
> insample <- xts::xts(retsis %*% weightv, zoo::index(retsis))
> outsample <- xts::xts(retsos %*% weightv, zoo::index(retsos))
> indeks <- xts::xts(rowMeans(retsp), 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 the cumulative portfolio returns
> endp <- rutils::calc_endpoints(pnls, interval="months")
> dygraphs::dygraph(cumsum(pnls)[endp], main="Out-of-sample Returns
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retsis[, 1])), label="in-sample", stroke
+   dyLegend(width=500)
```

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 $\bar{r}$:

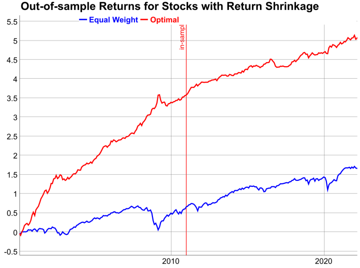
$$r_i' = (1 - \alpha)\, r_i + \alpha\, \bar{r}$$

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
> retsxm <- rowMeans(retsx["/2010"])
> retsxis <- (1-alpha)*retsx["/2010"] + alpha*retsxm
> # Calculate portfolio weights
> weightv <- invmat %*% colMeans(retsxis)
> 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 the cumulative portfolio returns
> dygraphs::dygraph(cumsum(pnls)[endp], main="Out-of-sample Returns
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retsis[, 1])), label="in-sample", stroke
+   dyLegend(width=500)
```

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& tseries,
                   double eigen_thresh = 0.001,
                   arma::uword dimax = 0) {

  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(weightv));
    break;
  } // end ranksharpe
  case method::max_sharpe: {
    // Mean returns by columns
    arma::vec meancols = arma::trans(arma::mean(returns, 0));
    // Shrink meancols to the mean of returns
    meancols = ((1-alpha)*meancols + alpha*arma::mean(meancols));
    // Apply regularized inverse
    // arma::mat inverse = calc_inv(cov(returns), dimax);
    // weightv = calc_inv(cov(returns), dimax)*meancols;
    weightv = calc_inv(cov(returns), eigen_thresh, dimax)*meancols;
```

Strategy Backtesting Using *RcppArmadillo*

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

```
arma::mat back_test(const arma::mat& retsx, // Portfolio excess returns
                    const arma::mat& returns, // Portfolio returns
                    arma::uvec startp,
                    arma::uvec endp,
                    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 < endp.size(); it++) {
    // cout << "it: " << it << endl;
    // Calculate portfolio weights
    weightv = coeff*calc_weights(retsx.rows(startp(it-1), endp(it-1)), method, eigen_thresh, dimax, confi, alpha
    // Calculate out-of-sample returns
    pnls.rows(endp(it-1)+1, endp(it)) = returns.rows(endp(it-1)+1, endp(it))*weightv;
    // Add transaction costs
    pnls.row(endp(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
```

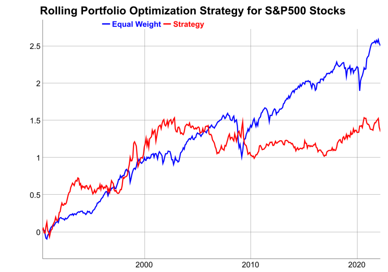
Rolling Portfolio Optimization Strategy for S&P500 Stocks

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 returns100
> retsp <- returns100
> retsp[1, is.na(retsp[1, ])] <- 0
> retsp <- zoo::na.locf(retsp, na.rm=FALSE)
> retsx <- (retsp - riskf)
> nstocks <- NCOL(retsp) ; datev <- zoo::index(retsp)
> # Define monthly end points
> endp <- rutils::calc_endpoints(retsp, interval="months")
> endp <- endp[endp > (nstocks+1)]
> npts <- NROW(endp) ; look_back <- 12
> startp <- c(rep_len(0, look_back), endp[1:(npts-look_back)])
> # Perform loop over end points - takes very long !!!
> pnls <- lapply(2:npts, function(ep) {
+     # Subset the excess returns
+     insample <- retsx[startp[ep-1]:endp[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 <- retsp[(endp[ep-1]+1):endp[ep], ]
+     xts::xts(outsample %*% weightv, zoo::index(outsample))
+ })  # end lapply
```

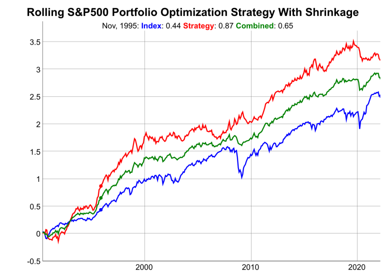



```
> # Calculate returns of equal weight portfolio
> indeks <- xts::xts(rowMeans(retsp), datev)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls*sd(indeks)/
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> colnames(wealthv) <- c("Equal Weight", "Strategy")
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot cumulative strategy returns
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Rolling Portfolio O
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

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
> endp <- (endp - 1)
> endp[endp < 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=retsx, returns=retsp,
+   startp=startp, endp=endp, 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)[endp], main="Rolling S&P500 Port
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

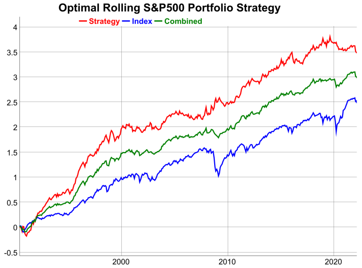
Determining Shrinkage Parameters Using Backtesting

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=retsx, returns=retsp,
+   startp=startp, endp=endp, controlv=controlv)
+ })  # end lapply
> profilev <- sapply(pnls, sum)
> plot(x=alphav, y=profilev, t="l", main="Rolling Strategy as Funct
+   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=retsx, returns=retsp,
+     startp=startp, endp=endp, 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]]
```



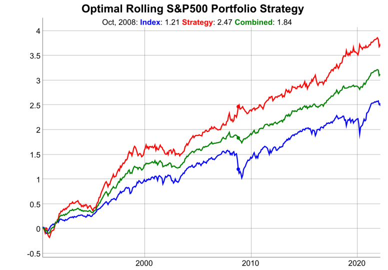

```
> # Plot cumulative strategy returns
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "Strategy", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endp], main="Optimal Rolling S&P
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

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), endp[1:(npts-look_back)])
+   startp <- (startp - 1)
+   startp[startp < 0] <- 0
+   HighFreq::back_test(excess=retsx, returns=retsp,
+     startp=startp, endp=endp, 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)[endp], main="Optimal Rolling S&P
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```

Vector and Matrix Calculus

Let $\mathbf{v}$ and $\mathbf{w}$ be vectors, with $\mathbf{v} = \{v_i\}_{i=1}^{i=n}$, and let $\mathbb{1}$ be the unit vector, with $\mathbb{1} = \{1\}_{i=1}^{i=n}$.

Then the inner product of $\mathbf{v}$ and $\mathbf{w}$ can be written as $\mathbf{v}^T\mathbf{w} = \mathbf{w}^T\mathbf{v} = \sum_{i=1}^{n} v_i w_i$.

We can then express the sum of the elements of $\mathbf{v}$ as the inner product: $\mathbf{v}^T\mathbb{1} = \mathbb{1}^T\mathbf{v} = \sum_{i=1}^{n} v_i$.

And the sum of squares of $\mathbf{v}$ as the inner product: $\mathbf{v}^T\mathbf{v} = \sum_{i=1}^{n} v_i^2$.

Let $\mathbb{A}$ be a matrix, with $\mathbb{A} = \{A_{ij}\}_{i,j=1}^{i,j=n}$.

Then the inner product of matrix $\mathbb{A}$ with vectors $\mathbf{v}$ and $\mathbf{w}$ can be written as:

$$\mathbf{v}^T\mathbb{A}\,\mathbf{w} = \mathbf{w}^T\mathbb{A}^T\mathbf{v} = \sum_{i,j=1}^{n} A_{ij} v_i w_j$$

The derivative of a scalar variable with respect to a vector variable is a vector, for example:

$$\begin{aligned}\frac{d(\mathbf{v}^T\mathbb{1})}{d\mathbf{v}} = d_v[\mathbf{v}^T\mathbb{1}] = d_v[\mathbb{1}^T\mathbf{v}] &= \mathbb{1}^T \\ d_v[\mathbf{v}^T\mathbf{w}] = d_v[\mathbf{w}^T\mathbf{v}] &= \mathbf{w}^T \\ d_v[\mathbf{v}^T\mathbb{A}\,\mathbf{w}] &= \mathbf{w}^T\mathbb{A}^T \\ d_v[\mathbf{v}^T\mathbb{A}\,\mathbf{v}] &= \mathbf{v}^T\mathbb{A} + \mathbf{v}^T\mathbb{A}^T\end{aligned}$$

Portfolio Weight Constraints

Portfolio optimization requires constraints on the portfolio weights to prevent excessive leverage (the size of positions relative to the capital).

Portfolio-level constraints limit the combined size of the weights.

For example, under *linear* constraints the sum of the weights is equal to 1: $\mathbf{w}^T \mathbb{1} = \sum_{i=1}^{n} w_i = 1$, so that the weights are constrained to a *hyperplane*.

The weights can be shifted by an amount x in order to satisfy the *linear* constraint: $w_i' = w_i - x$. This is equivalent to subtracting an equal-weighted portfolio from the weights.

The disadvantage of *linear* constraints is that they allow highly leveraged portfolios, with very large positive and negative weights.

Under *quadratic* constraints the sum of the *squared* weights is equal to 1: $\mathbf{w}^T \mathbf{w} = \sum_{i=1}^{n} w_i^2 = 1$, so that the weights are constrained to a *hypersphere*.

The weights can be scaled by a factor x in order to satisfy the *quadratic* constraint: $w_i' = x w_i$. This is equivalent to deleveraging the portfolio.

```
> # Linear constraint
> weightv <- weightv/sum(weightv)
> # Quadratic constraint
> weightv <- weightv/sqrt(sum(weightv^2))
> # Box constraints
> weightv[weightv > 1] <- 1
> weightv[weightv < 0] <- 0
```

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 some stocks.

Maximum Return Portfolio Using Linear Programming

The weights of the maximum return portfolio are obtained by maximizing the portfolio returns:

$$w_{max} = \arg\max_{w} [\, \mathbf{r}^T \mathbf{w} \,] = \arg\max_{w} [\, \sum_{i=1}^{n} w_i r_i \,]$$

Where $\mathbf{r}$ is the vector of returns, and $\mathbf{w}$ is the vector of portfolio weights, with a linear constraint:

$$\mathbf{w}^T \mathbb{1} = \sum_{i=1}^{n} w_i = 1$$

And a box constraint:

$$0 \leq w_i \leq 1$$

The weights of the maximum return portfolio can be calculated using linear programming (*LP*), which is the optimization of linear objective functions subject to linear constraints.

The function `Rglpk_solve_LP()` from package *Rglpk* solves linear programming problems by calling the *GNU Linear Programming Kit* library.

```
> library(rutils)
> library(Rglpk)
> # Vector of symbol names
> symbolv <- c("VTI", "IEF", "DBC")
> nstocks <- NROW(symbolv)
> # Calculate mean returns
> retsp <- na.omit(rutils::etfenv$returns[, symbolv])
> retsm <- colMeans(retsp)
> # Specify linear constraint coefficients
> lincon <- matrix(c(rep(1, nstocks), 1, 1, 0),
+                  nc=nstocks, byrow=TRUE)
> directs <- c("==", "<=")
> rhs <- c(1, 0)
> # Specify box constraints (-1, 1) (default is c(0, Inf))
> boxc <- list(lower=list(ind=1:nstocks, val=rep(-1, nstocks)),
+         upper=list(ind=1:nstocks, val=rep(1, nstocks)))
> # Perform optimization
> optiml <- Rglpk::Rglpk_solve_LP(
+   obj=retsm,
+   mat=lincon,
+   dir=directs,
+   rhs=rhs,
+   bounds=boxc,
+   max=TRUE)
> unlist(optiml[1:2])
```

The *Minimum Variance* Portfolio Under *Linear* Constraints

The portfolio variance is equal to: $\mathbf{w}^T \mathbb{C} \, \mathbf{w}$, where $\mathbb{C}$ is the covariance matrix of returns.

If the portfolio weights $\mathbf{w}$ are subject to *linear* constraints: $\mathbf{w}^T \mathbb{1} = \sum_{i=1}^{n} w_i = 1$, then the weights that minimize the portfolio variance can be found by minimizing the *Lagrangian*:

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

Where λ is a *Lagrange multiplier*.

The derivative of a scalar variable with respect to a vector variable is a vector, for example:

$$d_w[\mathbf{w}^T \mathbb{1}] = d_w[\mathbb{1}^T \mathbf{w}] = \mathbb{1}^T$$

$$d_w[\mathbf{w}^T \mathbf{r}] = d_w[\mathbf{r}^T \mathbf{w}] = \mathbf{r}^T$$

$$d_w[\mathbf{w}^T \mathbb{C} \, \mathbf{w}] = \mathbf{w}^T \mathbb{C} + \mathbf{w}^T \mathbb{C}^T$$

Where $\mathbb{1}$ is the unit vector, and $\mathbf{w}^T \mathbb{1} = \mathbb{1}^T \mathbf{w} = \sum_{i=1}^{n} x_i$

The derivative of the *Lagrangian* $\mathcal{L}$ with respect to $\mathbf{w}$ is given by:

$$d_w \mathcal{L} = 2\mathbf{w}^T \mathbb{C} - \lambda \mathbb{1}^T$$

By setting the derivative to zero we find $\mathbf{w}$ equal to:

$$\mathbf{w} = \frac{1}{2} \lambda \, \mathbb{C}^{-1} \mathbb{1}$$

By multiplying the above from the left by $\mathbb{1}^T$, and using $\mathbf{w}^T \mathbb{1} = 1$, we find λ to be equal to:

$$\lambda = \frac{2}{\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}}$$

And finally the portfolio weights are then equal to:

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

If the portfolio weights are subject to *quadratic* constraints: $\mathbf{w}^T \mathbf{w} = 1$ then the minimum variance weights are equal to the highest order *principal component* (with the smallest eigenvalue) of the covariance matrix $\mathbb{C}$.

Variance of the *Minimum Variance* Portfolio

The weights of the *minimum variance* portfolio under the constraint $\mathbf{w}^T \mathbb{1} = 1$ can be calculated using the inverse of the covariance matrix:

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

The variance of the *minimum variance* portfolio is equal to:

$$\sigma^2 = \frac{\mathbb{1}^T \mathbb{C}^{-1} \mathbb{C} \, \mathbb{C}^{-1} \mathbb{1}}{(\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1})^2} = \frac{1}{\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}}$$

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

The `%*%` operator performs *inner* (*scalar*) multiplication of vectors and matrices.

Inner multiplication multiplies the rows of one matrix with the columns of another matrix, so that each pair produces a single number:

The function `drop()` removes any dimensions of length *one*.

```
> # Calculate covariance matrix of returns and its inverse
> covmat <- cov(retsp)
> covinv <- solve(a=covmat)
> unitv <- rep(1, NCOL(covmat))
> # Minimum variance weights with constraint
> # weightv <- solve(a=covmat, b=unitv)
> weightv <- covinv %*% unitv
> weightv <- weightv/drop(t(unitv) %*% weightv)
> # Minimum variance
> t(weightv) %*% covmat %*% weightv
> 1/(t(unitv) %*% covinv %*% unitv)
```

The *Efficient Portfolios*

A portfolio which has the smallest variance, given a target return, is an *efficient portfolio*.

The *efficient portfolio* weights have two constraints: the sum of portfolio weights $\mathbf{w}$ is equal to 1: $\mathbf{w}^T \mathbb{1} = \sum_{i=1}^{n} w_i = 1$, and the mean portfolio return is equal to the target return r_t: $\mathbf{w}^T \mathbf{r} = \sum_{i=1}^{n} w_i r_i = r_t$.

The weights that minimize the portfolio variance under these constraints can be found by minimizing the *Lagrangian*:

$$\mathcal{L} = \mathbf{w}^T \mathbb{C} \, \mathbf{w} - \lambda_1 \, (\mathbf{w}^T \mathbb{1} - 1) - \lambda_2 \, (\mathbf{w}^T \mathbf{r} - r_t)$$

Where λ_1 and λ_2 are the *Lagrange multipliers*.

The derivative of the *Lagrangian* $\mathcal{L}$ with respect to $\mathbf{w}$ is given by:

$$d_w \mathcal{L} = 2\mathbf{w}^T \mathbb{C} - \lambda_1 \mathbb{1}^T - \lambda_2 \mathbf{r}^T$$

By setting the derivative to zero we obtain the *efficient portfolio* weights $\mathbf{w}$:

$$\mathbf{w} = \frac{1}{2} (\lambda_1 \, \mathbb{C}^{-1} \mathbb{1} + \lambda_2 \, \mathbb{C}^{-1} \mathbf{r})$$

By multiplying the above from the left first by $\mathbb{1}^T$, and then by $\mathbf{r}^T$, we obtain a system of two equations for λ_1 and λ_2:

$$2\mathbb{1}^T \mathbf{w} = \lambda_1 \, \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1} + \lambda_2 \, \mathbb{1}^T \mathbb{C}^{-1} \mathbf{r} = 2$$

$$2\mathbf{r}^T \mathbf{w} = \lambda_1 \, \mathbf{r}^T \mathbb{C}^{-1} \mathbb{1} + \lambda_2 \, \mathbf{r}^T \mathbb{C}^{-1} \mathbf{r} = 2r_t$$

The above can be written in matrix notation as:

$$\begin{bmatrix} \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1} & \mathbb{1}^T \mathbb{C}^{-1} \mathbf{r} \\ \mathbf{r}^T \mathbb{C}^{-1} \mathbb{1} & \mathbf{r}^T \mathbb{C}^{-1} \mathbf{r} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2r_t \end{bmatrix}$$

Or:

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \mathbb{F} \lambda = 2 \begin{bmatrix} 1 \\ r_t \end{bmatrix} = 2u$$

With $a = \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$, $b = \mathbb{1}^T \mathbb{C}^{-1} \mathbf{r}$, $c = \mathbf{r}^T \mathbb{C}^{-1} \mathbf{r}$, $\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$, $u = \begin{bmatrix} 1 \\ r_t \end{bmatrix}$, and $\mathbb{F} = u^T \mathbb{C}^{-1} u = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$.

The *Lagrange multipliers* can be solved as:

$$\lambda = 2\mathbb{F}^{-1} u$$

The *Efficient Portfolio* Weights

The *efficient portfolio* weights $\mathbf{w}$ can now be solved as:

$$\mathbf{w} = \frac{1}{2}(\lambda_1 \mathbb{C}^{-1}\mathbb{1} + \lambda_2 \mathbb{C}^{-1}\mathbf{r}) =$$

$$\frac{1}{2}\begin{bmatrix} \mathbb{C}^{-1}\mathbb{1} \\ \mathbb{C}^{-1}\mathbf{r} \end{bmatrix}^T \lambda = \begin{bmatrix} \mathbb{C}^{-1}\mathbb{1} \\ \mathbb{C}^{-1}\mathbf{r} \end{bmatrix}^T \mathbb{F}^{-1} u =$$

$$\frac{1}{ac - b^2}\begin{bmatrix} \mathbb{C}^{-1}\mathbb{1} \\ \mathbb{C}^{-1}\mathbf{r} \end{bmatrix}^T \begin{bmatrix} c & -b \\ -b & a \end{bmatrix} \begin{bmatrix} 1 \\ r_t \end{bmatrix} =$$

$$\frac{(c - br_t)\,\mathbb{C}^{-1}\mathbb{1} + (ar_t - b)\,\mathbb{C}^{-1}\mathbf{r}}{ac - b^2}$$

With $a = \mathbb{1}^T\mathbb{C}^{-1}\mathbb{1}$, $b = \mathbb{1}^T\mathbb{C}^{-1}\mathbf{r}$, $c = \mathbf{r}^T\mathbb{C}^{-1}\mathbf{r}$.

The above formula shows that a convex sum of two *efficient portfolio* weights: $w = \alpha w_1 + (1 - \alpha) w_2$
Are also the weights of an *efficient portfolio*, with target return equal to: $r_t = \alpha r_1 + (1 - \alpha) r_2$

```
> # Calculate vector of mean returns
> retsm <- colMeans(retsp)
> # Specify the target return
> rett <- 1.5*mean(retsp)
> # Products of inverse with mean returns and unit vector
> fmat <- matrix(c(
+   t(unitv) %*% covinv %*% unitv,
+   t(unitv) %*% covinv %*% retsm,
+   t(retsm) %*% covinv %*% unitv,
+   t(retsm) %*% covinv %*% retsm), nc=2)
> # Solve for the Lagrange multipliers
> lagm <- solve(a=fmat, b=c(2, 2*rett))
> # Calculate weights
> weightv <- drop(0.5*covinv %*% cbind(unitv, retsm) %*% lagm)
> # Calculate constraints
> all.equal(1, sum(weightv))
> all.equal(rett, sum(retsm*weightv))
```

Variance of the *Efficient Portfolios*

The *efficient portfolio* variance is equal to:

$$\sigma^2 = \mathbf{w}^T \mathbb{C} \, \mathbf{w} = \frac{1}{4} \lambda^T \mathbb{F} \, \lambda = u^T \mathbb{F}^{-1} \, u = \frac{1}{ac - b^2} \begin{bmatrix} 1 \\ r_t \end{bmatrix}^T \begin{bmatrix} c & -b \\ -b & a \end{bmatrix} \begin{bmatrix} 1 \\ r_t \end{bmatrix} = \frac{a r_t^2 - 2b r_t + c}{ac - b^2}$$

The above formula shows that the variance of the *efficient portfolios* is a *parabola* with respect to the target return r_t.

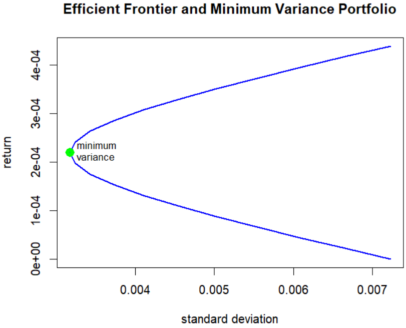
The vertex of the *parabola* is at $r_t = \mathbb{1}^T \mathbb{C}^{-1} \mathbf{r} / \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$ and $\sigma^2 = 1 / \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$.

```
> # Calculate portfolio return and standard deviation
> retsp <- drop(retsp %*% weightv)
> c(return=mean(retsp), sd=sd(retsp))
> all.equal(mean(retsp), rett)
> # Calculate portfolio variance
> uu <- c(1, rett)
> finv <- solve(fmat)
> all.equal(var(retsp), drop(t(uu) %*% finv %*% uu))
> # Calculate vertex of variance parabola
> weightv <- drop(covinv %*% unitv /
+   drop(t(unitv) %*% covinv %*% unitv))
> retsp <- drop(retsp %*% weightv)
> retsv <- drop(t(unitv) %*% covinv %*% retsm /
+   t(unitv) %*% covinv %*% unitv)
> all.equal(mean(retsp), retsv)
> varmin <- drop(1/t(unitv) %*% covinv %*% unitv)
> all.equal(var(retsp), varmin)
```

The *Efficient Frontier*

The *efficient frontier* is the plot of the *efficient portfolio* standard deviations with respect to the target return r_t, which is a *hyperbola*.

```
> # Calculate efficient frontier from target returns
> retst <- retsv*(1+seq(from=(-1), to=1, by=0.1))
> effront <- sapply(retst, function(rett) {
+   uu <- c(1, rett)
+   sqrt(drop(t(uu) %*% finv %*% uu))
+ })  # end sapply
> # Plot efficient frontier
> x11(width=6, height=5)
> plot(x=effront, y=retst, t="l", col="blue", lwd=2,
+      main="Efficient Frontier and Minimum Variance Portfolio",
+      xlab="standard deviation", ylab="return")
> points(x=sqrt(varmin), y=retsv, col="green", lwd=6)
> text(x=sqrt(varmin), y=retsv, labels="minimum \nvariance",
+      pos=4, cex=0.8)
```



The *Tangent Line* and the *Risk-free* Rate

A *tangent* line can be drawn at every point on the *efficient frontier*.

The slope β of the *tangent* line can be calculated by differentiating the variance σ^2 by the target return r_t:

$$\frac{d\sigma^2}{dr_t} = 2\sigma \frac{d\sigma}{dr_t} = \frac{2ar_t - 2b}{ac - b^2}$$
$$\frac{d\sigma}{dr_t} = \frac{ar_t - b}{\sigma\,(ac - b^2)}$$
$$\beta = \frac{\sigma\,(ac - b^2)}{ar_t - b}$$

The *tangent* line connects the *tangent* point on the *efficient frontier* with a *risk-free* rate r_f.

The *risk-free* rate r_f can be calculated as the intercept of the tangent line:

$$r_f = r_t - \sigma\,\beta = r_t - \frac{\sigma^2\,(ac - b^2)}{ar_t - b} =$$
$$r_t - \frac{ar_t^2 - 2br_t + c}{ac - b^2}\,\frac{ac - b^2}{ar_t - b} =$$
$$r_t - \frac{ar_t^2 - 2br_t + c}{ar_t - b} = \frac{br_t - c}{ar_t - b}$$

```
> # Calculate portfolio standard deviation
> stdev <- sqrt(drop(t(uu) %*% finv %*% uu))
> # Calculate the slope of the tangent line
> slopev <- (stdev*det(fmat))/(fmat[1, 1]*rett-fmat[1, 2])
> # Calculate the risk-free rate as intercept of the tangent line
> riskf <- rett - slopev*stdev
> # Calculate the risk-free rate from target return
> riskf <- (rett*fmat[1, 2]-fmat[2, 2]) /
+   (rett*fmat[1, 1]-fmat[1, 2])
```

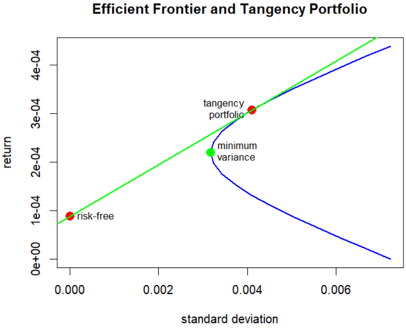
The *Tangent Line* on the *Efficient Frontier*

The *efficient portfolios* are also called *tangency portfolios*, since they are the tangent points on the *efficient frontier*.

The *tangency portfolio* is the *market portfolio* corresponding to the given *risk-free* rate.

The *tangent line* at the *market portfolio* is known as the *Capital Market Line* (CML).

```
> # Plot efficient frontier
> plot(x=effront, y=retst, t="l", col="blue", lwd=2,
+      xlim=c(0.0, max(effront)),
+      main="Efficient Frontier and Tangency Portfolio",
+      xlab="standard deviation", ylab="return")
> # Plot minimum variance
> points(x=sqrt(varmin), y=retsv, col="green", lwd=6)
> text(x=sqrt(varmin), y=retsv, labels="minimum \nvariance",
+      pos=4, cex=0.8)
> # Plot tangent point
> points(x=stdev, y=rett, col="red", lwd=6)
> text(x=stdev, y=rett, labels="tangency\nportfolio", pos=2, cex=0.8)
> # Plot risk-free point
> points(x=0, y=riskf, col="red", lwd=6)
> text(x=0, y=riskf, labels="risk-free", pos=4, cex=0.8)
> # Plot tangent line
> abline(a=riskf, b=slopev, lwd=2, col="green")
```



Maximum *Sharpe* Portfolio Weights

The *Sharpe* ratio is equal to the ratio of excess returns divided by the portfolio standard deviation:

$$SR = \frac{\mathbf{w}^T \mu}{\sigma}$$

Where $\mu = \mathbf{r} - r_f$ is the vector of excess returns (in excess of the risk-free rate r_f), $\mathbf{w}$ is the vector of portfolio weights, and $\sigma = \sqrt{\mathbf{w}^T \mathbb{C}\, \mathbf{w}}$, where $\mathbb{C}$ is the covariance matrix of returns.

We can calculate the maximum *Sharpe* portfolio weights by setting the derivative of the *Sharpe* ratio with respect to the weights, to zero:

$$d_w SR = \frac{1}{\sigma}(\mu^T - \frac{(\mathbf{w}^T \mu)(\mathbf{w}^T \mathbb{C})}{\sigma^2}) = 0$$

We then get:

$$(\mathbf{w}^T \mathbb{C}\, \mathbf{w})\, \mu = (\mathbf{w}^T \mu)\, \mathbb{C}\mathbf{w}$$

We can multiply the above equation by $\mathbb{C}^{-1}$ to get:

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

We can finally rescale the weights so that they satisfy the linear constraint $\mathbf{w}^T \mathbb{1} = 1$:

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

These are the weights of the maximum *Sharpe* portfolio, with the vector of excess returns equal to μ, and the covariance matrix equal to $\mathbb{C}$.

The maximum *Sharpe* portfolio is an *efficient portfolio*, and so its mean return is equal to some target return r_t: $\mathbf{w}^T \mathbf{r} = \sum_{i=1}^n w_i r_i = r_t$.

The mean portfolio return can be written as:

$$\mathbf{r}^T \mathbf{w} = \frac{\mathbf{r}^T \mathbb{C}^{-1} \mu}{\mathbb{1}^T \mathbb{C}^{-1} \mu} = \frac{\mathbf{r}^T \mathbb{C}^{-1} (\mathbf{r} - r_f)}{\mathbb{1}^T \mathbb{C}^{-1} (\mathbf{r} - r_f)} =$$

$$r_t = \frac{\mathbf{r}^T \mathbb{C}^{-1} \mathbb{1}\; r_f - \mathbf{r}^T \mathbb{C}^{-1} \mathbf{r}}{\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}\; r_f - \mathbf{r}^T \mathbb{C}^{-1} \mathbb{1}}$$

The above formula calculates the target return r_t from the risk-free rate r_f.

Returns and Variance of Maximum *Sharpe* Portfolio

The weights of the maximum *Sharpe* portfolio are equal to:

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

Where μ is the vector of excess returns, and $\mathbb{C}$ is the covariance matrix.

The excess returns of the maximum *Sharpe* portfolio are equal to:

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

The variance of the maximum *Sharpe* portfolio is equal to:

$$\sigma^2 = \frac{\mu^T\mathbb{C}^{-1}\mathbb{C}\,\mathbb{C}^{-1}\mu}{(\mathbb{1}^T\mathbb{C}^{-1}\mu)^2} = \frac{\mu^T\mathbb{C}^{-1}\mu}{(\mathbb{1}^T\mathbb{C}^{-1}\mu)^2}$$

The *Sharpe* ratio is equal to:

$$SR = \sqrt{\mu^T\mathbb{C}^{-1}\mu}$$

```
> # Calculate excess returns
> riskf <- 0.03/252
> retsx <- (retsp - riskf)
> # Calculate covariance and inverse matrix
> covmat <- cov(retsp)
> unitv <- rep(1, NCOL(covmat))
> covinv <- solve(a=covmat)
> # Calculate mean excess returns
> retsx <- sapply(retsx, mean)
> # Weights of maximum Sharpe portfolio
> # weightv <- solve(a=covmat, b=returns)
> weightv <- covinv %*% retsx
> weightv <- weightv/drop(t(unitv) %*% weightv)
> # Sharpe ratios
> sqrt(252)*sum(weightv*retsx) /
+   sqrt(drop(weightv %*% covmat %*% weightv))
> sapply(retsp - riskf, function(x) sqrt(252)*mean(x)/sd(x))
> maxsharpe <- weightv
```

Optimal Portfolios Under Zero Correlation

If the correlations of returns are equal to zero, then the covariance matrix is diagonal:

$$\mathbb{C} = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n^2 \end{pmatrix}$$

Where σ_i^2 is the variance of returns of asset `i`.

The inverse of $\mathbb{C}$ is then simply:

$$\mathbb{C}^{-1} = \begin{pmatrix} \sigma_1^{-2} & 0 & \cdots & 0 \\ 0 & \sigma_2^{-2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n^{-2} \end{pmatrix}$$

The *minimum variance* portfolio weights are proportional to the inverse of the individual variances:

$$w_i = \frac{1}{\sigma_i^2 \sum_{i=1}^n \sigma_i^{-2}}$$

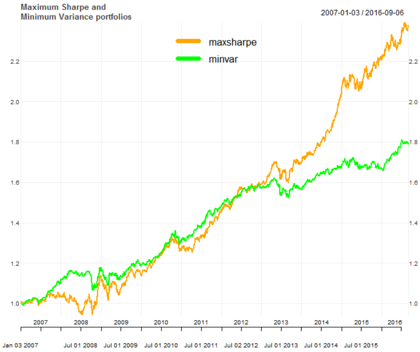
The maximum *Sharpe* portfolio weights are proportional to the ratio of excess returns divided by the individual variances:

$$w_i = \frac{\mu_i}{\sigma_i^2 \sum_{i=1}^n \mu_i \sigma_i^{-2}}$$

Maximum *Sharpe* and *Minimum Variance* Performance

The maximum *Sharpe* and *Minimum Variance* portfolios are both *efficient portfolios*, with the lowest risk (standard deviation) for the given level of return.

```
> library(rutils)
> # Calculate minimum variance weights
> weightv <- covinv %*% unitv
> minvar <- weightv/drop(t(unitv) %*% weightv)
> # Calculate optimal portfolio returns
> retsoptim <- xts(
+   x=cbind(exp(cumsum(retsp %*% maxsharpe)),
+     exp(cumsum(retsp %*% minvar))),
+   order.by=zoo::index(retsp))
> colnames(retsoptim) <- c("maxsharpe", "minvar")
> # Plot optimal portfolio returns, with custom line colors
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("orange", "green")
> x11(width=6, height=5)
> chart_Series(retsoptim, theme=plot_theme,
+   name="Maximum Sharpe and
+   Minimum Variance portfolios")
> legend("top", legend=colnames(retsoptim), cex=0.8,
+  inset=0.1, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```



The *Efficient Frontier* and *Capital Market Line*

The maximum *Sharpe* portfolio weights depend on the value of the risk-free rate r_f,

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

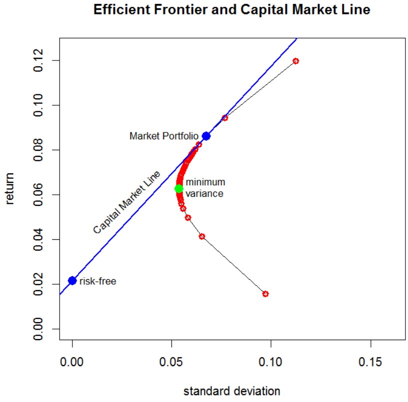
The *Efficient Frontier* is the set of *efficient portfolios*, that have the lowest risk (standard deviation) for the given level of return.

The maximum *Sharpe* portfolios are *efficient portfolios*, and they lie on the *Efficient Frontier*, forming a tangent line from the risk-free rate to the *Efficient Frontier*, known as the *Capital Market Line* (CML).

The maximum *Sharpe* portfolios are considered to be the *market portfolios*, corresponding to different values of the risk-free rate r_f.

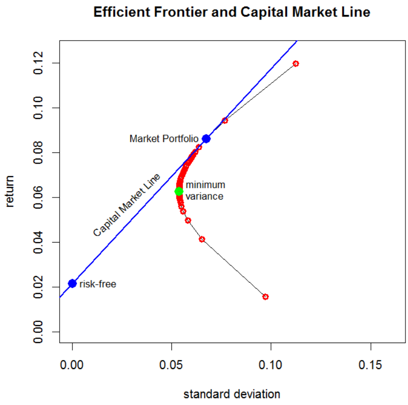
The maximum *Sharpe* portfolios are also called *tangency* portfolios, since they are the tangent point on the *Efficient Frontier*.

The *Capital Market Line* is the line drawn from the *risk-free* rate to the *market portfolio* on the *Efficient Frontier*.



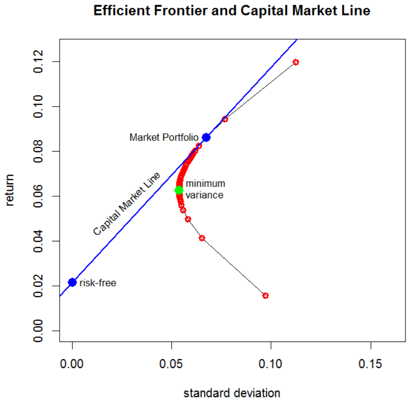
Plotting *Efficient Frontier* and Maximum *Sharpe* Portfolios

```
> # Calculate minimum variance weights
> weightv <- covinv %*% unitv
> weightv <- weightv/drop(t(unitv) %*% weightv)
> # Minimum standard deviation and return
> stdev <- sqrt(252*drop(weightv %*% covmat %*% weightv))
> retsp <- 252*sum(weightv*retsm)
> # Calculate maximum Sharpe portfolios
> riskf <- (retsp * seq(-10, 10, by=0.1)^3)/252
> effront <- sapply(riskf, function(riskf) {
+   weightv <- covinv %*% (retsm - riskf)
+   weightv <- weightv/drop(t(unitv) %*% weightv)
+   # Portfolio return and standard deviation
+   c(return=252*sum(weightv*retsm),
+     stddev=sqrt(252*drop(weightv %*% covmat %*% weightv)))
+ })  # end sapply
> effront <- cbind(252*riskf, t(effront))
> colnames(effront)[1] <- "risk-free"
> effront <- effront[is.finite(effront[, "stddev"]), ]
> effront <- effront[order(effront[, "return"]), ]
> # Plot maximum Sharpe portfolios
> plot(x=effront[, "stddev"],
+      y=effront[, "return"], t="l",
+      xlim=c(0.0*stdev, 3.0*stdev),
+      ylim=c(0.0*retsp, 2.0*retsp),
+      main="Efficient Frontier and Capital Market Line",
+      xlab="standard deviation", ylab="return")
> points(x=effront[, "stddev"], y=effront[, "return"],
+  col="red", lwd=3)
```



Plotting the *Capital Market Line*

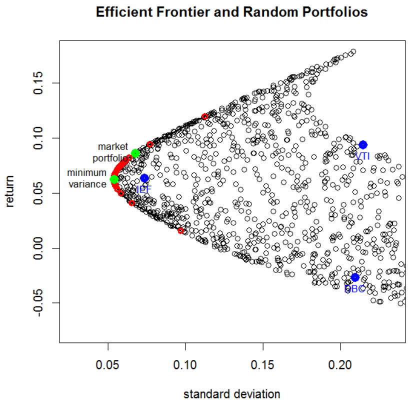
```
> # Plot minimum variance portfolio
> points(x=stdev, y=retsp, col="green", lwd=6)
> text(stdev, retsp, labels="minimum \nvariance",
+      pos=4, cex=0.8)
> # Draw Capital Market Line
> sortv <- sort(effront[, 1])
> riskf <- sortv[findInterval(x=0.5*retsp, vec=sortv)]
> points(x=0, y=riskf, col="blue", lwd=6)
> text(x=0, y=riskf, labels="risk-free",
+      pos=4, cex=0.8)
> marketp <- match(riskf, effront[, 1])
> points(x=effront[marketp, "stddev"],
+  y=effront[marketp, "return"],
+  col="blue", lwd=6)
> text(x=effront[marketp, "stddev"],
+      y=effront[marketp, "return"],
+      labels="market portfolio",
+      pos=2, cex=0.8)
> sharper <- (effront[marketp, "return"]-riskf)/
+   effront[marketp, "stddev"]
> abline(a=riskf, b=sharper, col="blue", lwd=2)
> text(x=0.7*effront[marketp, "stddev"],
+      y=0.7*effront[marketp, "return"]+0.01,
+      labels="Capital Market Line", pos=2, cex=0.8,
+      srt=45*atan(sharper*heightp/widthp)/(0.25*pi))
```



The *Capital Market Line* represents delevered and levered portfolios, consisting of the *market portfolio* combined with the *risk-free* rate.

Plotting Random Portfolios

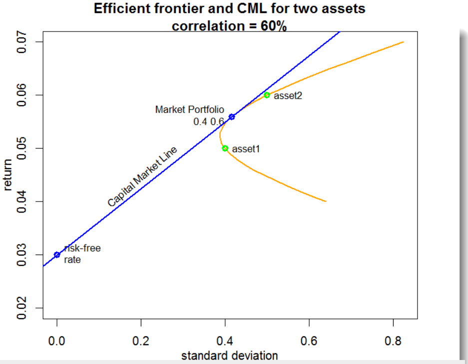
```
> # Calculate random portfolios
> nportf <- 1000
> randportf <- sapply(1:nportf, function(it) {
+   weightv <- runif(nstocks-1, min=-0.25, max=1.0)
+   weightv <- c(weightv, 1-sum(weightv))
+   # Portfolio return and standard deviation
+   c(return=252*sum(weightv*retsm),
+     stddev=sqrt(252*drop(weightv %*% covmat %*% weightv)))
+ })  # end sapply
> # Plot scatterplot of random portfolios
> x11(widthp <- 6, heightp <- 6)
> plot(x=randportf["stddev", ], y=randportf["return", ],
+      main="Efficient Frontier and Random Portfolios",
+      xlim=c(0.5*stdev, 0.8*max(randportf["stddev", ])),
+      xlab="standard deviation", ylab="return")
> # Plot maximum Sharpe portfolios
> lines(x=effront[, "stddev"],
+      y=effront[, "return"], lwd=2)
> points(x=effront[, "stddev"], y=effront[, "return"],
+  col="red", lwd=3)
> # Plot minimum variance portfolio
> points(x=stdev, y=retsp, col="green", lwd=6)
> text(stdev, retsp, labels="minimum\nvariance",
+      pos=2, cex=0.8)
> # Plot market portfolio
> points(x=effront[marketp, "stddev"],
+  y=effront[marketp, "return"], col="green", lwd=6)
> text(x=effront[marketp, "stddev"],
+      y=effront[marketp, "return"],
+      labels="market\nportfolio",
+      pos=2, cex=0.8)
```




```
> # Plot individual assets
> points(x=sqrt(252*diag(covmat)),
+  y=252*retsm, col="blue", lwd=6)
> text(x=sqrt(252*diag(covmat)), y=252*retsm,
+      labels=names(retsm),
+      col="blue", pos=1, cex=0.8)
```

Plotting Efficient Frontier for Two-asset Portfolios

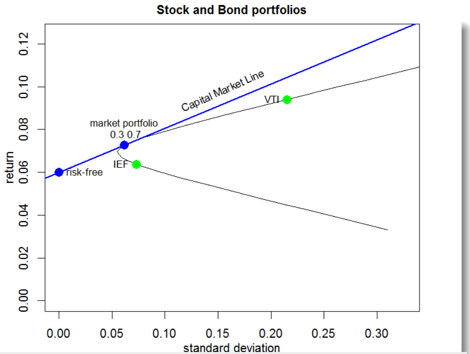
```
> riskf <- 0.03
> retsp <- c(asset1=0.05, asset2=0.06)
> stdevs <- c(asset1=0.4, asset2=0.5)
> corrp <- 0.6
> covmat <- matrix(c(1, corrp, corrp, 1), nc=2)
> covmat <- t(t(stdevs*covmat)*stdevs)
> weightv <- seq(from=(-1), to=2, length.out=31)
> weightv <- cbind(weightv, 1-weightv)
> retsp <- weightv %*% retsp
> portfsd <- sqrt(rowSums(weightv*(weightv %*% covmat)))
> sharper <- (retsp-riskf)/portfsd
> whichmax <- which.max(sharper)
> sharpem <- max(sharper)
> # Plot efficient frontier
> x11(widthp <- 6, heightp <- 5)
> par(mar=c(3,3,2,1)+0.1, oma=c(0, 0, 0, 0), mgp=c(2, 1, 0))
> plot(portfsd, retsp, t="l",
+  main=paste0("Efficient frontier and CML for two assets\ncorrelat:
+  xlab="standard deviation", ylab="return",
+  lwd=2, col="orange",
+  xlim=c(0, max(portfsd)),
+  ylim=c(0.02, max(retsp)))
> # Add Market Portfolio (maximum Sharpe ratio portfolio)
> points(portfsd[whichmax], retsp[whichmax],
+  col="blue", lwd=3)
> text(x=portfsd[whichmax], y=retsp[whichmax],
+      labels=paste(c("market portfolio\n",
+  structure(c(weightv[whichmax], 1-weightv[whichmax]),
+         names=names(retsp))), collapse=" "),
+      pos=2, cex=0.8)
```




```
> # Plot individual assets
> points(stdevs, retsp, col="green", lwd=3)
> text(stdevs, retsp, labels=names(retsp), pos=4, cex=0.8)
> # Add point at risk-free rate and draw Capital Market Line
> points(x=0, y=riskf, col="blue", lwd=3)
> text(0, riskf, labels="risk-free\nrate", pos=4, cex=0.8)
> abline(a=riskf, b=sharpem, lwd=2, col="blue")
> range_s <- par("usr")
> text(portfsd[whichmax]/2, (retsp[whichmax]+riskf)/2,
+      labels="Capital Market Line", cex=0.8, , pos=3,
+      srt=45*atan(sharpem*(range_s[2]-range_s[1])/
+              (range_s[4]-range_s[3])*
+              heightp/widthp)/(0.25*pi))
```

Efficient Frontier of Stock and Bond Portfolios

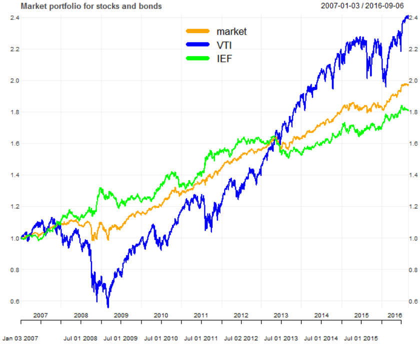
```
> # Vector of symbol names
> symbolv <- c("VTI", "IEF")
> # Matrix of portfolio weights
> weightv <- seq(from=(-1), to=2, length.out=31)
> weightv <- cbind(weightv, 1-weightv)
> # Calculate portfolio returns and volatilities
> retsp <- rutils::etfenv$returns[, symbolv]
> retsp <- retsp %*% t(weightv)
> portfv <- cbind(252*colMeans(retsp),
+   sqrt(252)*matrixStats::colSds(retsp))
> colnames(portfv) <- c("returns", "stddev")
> riskf <- 0.06
> portfv <- cbind(portfv,
+   (portfv[, "returns"]-riskf)/portfv[, "stddev"])
> colnames(portfv)[3] <- "Sharpe"
> whichmax <- which.max(portfv[, "Sharpe"])
> sharpem <- portfv[whichmax, "Sharpe"]
> plot(x=portfv[, "stddev"], y=portfv[, "returns"],
+      main="Stock and Bond portfolios", t="l",
+      xlim=c(0, 0.7*max(portfv[, "stddev"])), ylim=c(0, max(portfv
+      xlab="standard deviation", ylab="return")
> # Add blue point for market portfolio
> points(x=portfv[whichmax, "stddev"], y=portfv[whichmax, "returns'
> text(x=portfv[whichmax, "stddev"], y=portfv[whichmax, "returns"]
+      labels=paste(c("market portfolio\n",
+   structure(c(weightv[whichmax, 1], weightv[whichmax, 2]), names=
+      pos=3, cex=0.8)
```




```
> # Plot individual assets
> retsm <- 252*sapply(retsp, mean)
> stdevs <- sqrt(252)*sapply(retsp, sd)
> points(stdevs, retsm, col="green", lwd=6)
> text(stdevs, retsm, labels=names(retsp), pos=2, cex=0.8)
> # Add point at risk-free rate and draw Capital Market Line
> points(x=0, y=riskf, col="blue", lwd=6)
> text(0, riskf, labels="risk-free", pos=4, cex=0.8)
> abline(a=riskf, b=sharpem, col="blue", lwd=2)
> range_s <- par("usr")
> text(max(portfv[, "stddev"])/3, 0.75*max(portfv[, "returns"]),
+      labels="Capital Market Line", cex=0.8, , pos=3,
+      srt=45*atan(sharpem*(range_s[2]-range_s[1])/
+              (range_s[4]-range_s[3])*
+              heightp/widthp)/(0.25*pi))
```

Performance of Market Portfolio for Stocks and Bonds

```
> # Calculate cumulative returns of VTI and IEF
> retsoptim <- lapply(retsp,
+   function(retsp) exp(cumsum(retsp)))
> retsoptim <- rutils::do_call(cbind, retsoptim)
> # Calculate market portfolio returns
> retsoptim <- cbind(exp(cumsum(retsp %*%
+     c(weightv[whichmax], 1-weightv[whichmax]))),
+   retsoptim)
> colnames(retsoptim)[1] <- "market"
> # Plot market portfolio with custom line colors
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("orange", "blue", "green")
> chart_Series(retsoptim, theme=plot_theme,
+        name="Market portfolio for stocks and bonds")
> legend("top", legend=colnames(retsoptim),
+  cex=0.8, inset=0.1, bg="white", lty=1,
+  lwd=6, col=plot_theme$col$line.col, bty="n")
```



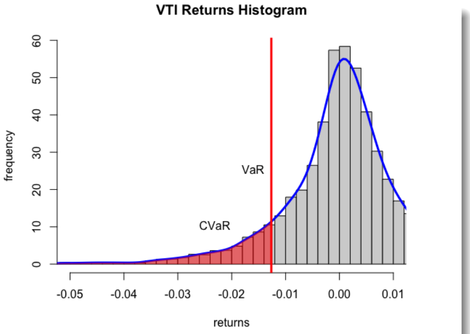
Conditional Value at Risk (*CVaR*)

The *Conditional Value at Risk* (*CVaR*) is equal to the average of the *VaR* for confidence levels less than a given confidence level α:

$$\mathrm{CVaR} = \frac{1}{\alpha} \int_{0}^{\alpha} \mathrm{VaR}(p) \, \mathrm{d}p$$

The *Conditional Value at Risk* is also called the Expected Shortfall (*ES*), or the Expected Tail Loss (*ETL*).

The function density() calculates a kernel estimate of the probability density for a sample of data, and returns a list with a vector of loss values and a vector of corresponding densities.




```
> # VTI percentage returns
> retsp <- rutils::diffit(log(quantmod::Cl(rutils::etfenv$VTI)))
> confl <- 0.1
> varisk <- quantile(retsp, confl)
> cvar <- mean(retsp[retsp < varisk])
> # Or
> sortv <- sort(as.numeric(retsp))
> varind <- round(confl*NROW(retsp))
> varisk <- sortv[varind]
> cvar <- mean(sortv[1:varind])
> # Plot histogram of VTI returns
> varmin <- (-0.05)
> histp <- hist(retsp, col="lightgrey",
+   xlab="returns", breaks=100, xlim=c(varmin, 0.01),
+   ylab="frequency", freq=FALSE, main="VTI Returns Histogram")
```

```
> # Plot density of losses
> densv <- density(retsp, adjust=1.5)
> lines(densv, lwd=3, col="blue")
> # Add line for VaR
> abline(v=varisk, col="red", lwd=3)
> ymax <- max(densv$y)
> text(x=varisk, y=2*ymax/3, labels="VaR", lwd=2, pos=2)
> # Add shading for CVaR
> rangev <- (densv$x < varisk) & (densv$x > varmin)
> polygon(
+   c(varmin, densv$x[rangev], varisk),
+   c(0, densv$y[rangev], 0),
+   col=rgb(1, 0, 0,0.5), border=NA)
> text(x=1.5*varisk, y=ymax/7, labels="CVaR", lwd=2, pos=2)
```

CVaR Portfolio Weights Using Linear Programming

The weights of the minimum *CVaR* portfolio can be calculated using linear programming (*LP*), which is the optimization of linear objective functions subject to linear constraints,

$$w_{min} = \arg\max_{w} [\sum_{i=1}^{n} w_i b_i]$$

Where b_i is the negative objective vector, and $\mathbf{w}$ is the vector of portfolio weights, with a linear constraint:

$$\mathbf{w}^T \mathbb{1} = \sum_{i=1}^{n} w_i = 1$$

And a box constraint:

$$0 \leq w_i \leq 1$$

The function `Rglpk_solve_LP()` from package *Rglpk* solves linear programming problems by calling the *GNU Linear Programming Kit* library.

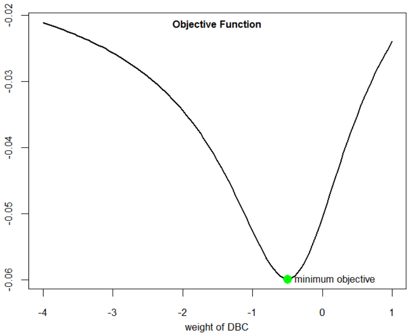
```
> library(rutils)  # Load rutils
> library(Rglpk)
> # Vector of symbol names and returns
> symbolv <- c("VTI", "IEF", "DBC")
> nstocks <- NROW(symbolv)
> retsp <- na.omit(rutils::etfenv$returns[, symbolv])
> retsm <- colMeans(retsp)
> confl <- 0.05
> rmin <- 0 ; wmin <- 0 ; wmax <- 1
> weightsum <- 1
> ncols <- NCOL(retsp) # number of assets
> nrows <- NROW(retsp) # number of rows
> # Create objective vector
> objvec <- c(numeric(ncols), rep(-1/(confl/nrows), nrows), -1)
> # Specify linear constraint coefficients
> lincon <- rbind(cbind(rbind(1, retsm),
+                 matrix(data=0, nrow=2, ncol=(nrows+1))),
+           cbind(coredata(retsp), diag(nrows), 1))
> rhs <- c(weightsum, rmin, rep(0, nrows))
> directs <- c("==", ">=", rep(">=", nrows))
> # Specify box constraints (wmin, wmax) (default is c(0, Inf))
> boxc <- list(lower=list(ind=1:ncols, val=rep(wmin, ncols)),
+          upper=list(ind=1:ncols, val=rep(wmax, ncols)))
> # Perform optimization
> optiml <- Rglpk_solve_LP(obj=objvec, mat=lincon, dir=directs, rhs=
> optiml$solution
> lincon %*% optiml$solution
> objvec %*% optiml$solution
> as.numeric(optiml$solution[1:ncols])
```

Sharpe Ratio Objective Function

The function `optimize()` performs *one-dimensional* optimization over a single independent variable.

`optimize()` searches for the minimum of the objective function with respect to its first argument, in the specified interval.

```
> # Create initial vector of portfolio weights
> weightv <- rep(1, NROW(symbolv))
> names(weightv) <- symbolv
> # Objective equal to minus Sharpe ratio
> objfun <- function(weightv, retsp) {
+   retsp <- retsp %*% weightv
+   if (sd(retsp) == 0)
+     return(0)
+   else
+     -return(mean(retsp)/sd(retsp))
+ }  # end objfun
> # Objective for equal weight portfolio
> objfun(weightv, retsp=retsp)
> optiml <- unlist(optimize(
+   f=function(weight)
+     objfun(c(1, 1, weight), retsp=retsp),
+   interval=c(-4, 1)))
> # Vectorize objective function with respect to third weight
> objvec <- function(weightv) sapply(weightv,
+   function(weight) objfun(c(1, 1, weight),
+     retsp=retsp))
> # Or
> objvec <- Vectorize(FUN=function(weight)
+     objfun(c(1, 1, weight), retsp=retsp),
+   vectorize.args="weight")  # end Vectorize
> objvec(1)
> objvec(1:3)
```




```
> # Plot objective function with respect to third weight
> curve(expr=objvec,
+       type="l", xlim=c(-4.0, 1.0),
+       xlab=paste("weight of", names(weightv[3])),
+       ylab="", lwd=2)
> title(main="Objective Function", line=(-1))  # Add title
> points(x=optiml[1], y=optiml[2], col="green", lwd=6)
> text(x=optiml[1], y=optiml[2],
+      labels="minimum objective", pos=4, cex=0.8)
>
> ### below is simplified code for plotting objective function
> # Create vector of DBC weights
> weightv <- seq(from=-4, to=1, by=0.1)
> obj_val <- sapply(weightv,
+   function(weight) objfun(c(1, 1, weight)))
> plot(x=weightv, y=obj_val, t="l",
```

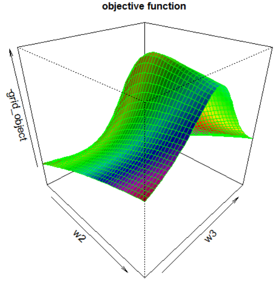
Perspective Plot of Portfolio Objective Function

The function persp() plots a 3d perspective surface plot of a function specified over a grid of argument values.

The function outer() calculates the values of a function over a grid spanned by two variables, and returns a matrix of function values.

The package *rgl* allows creating *interactive* 3d scatterplots and surface plots including perspective plots, based on the *OpenGL* framework.

```
> # Vectorize function with respect to all weights
> objvec <- Vectorize(
+   FUN=function(w1, w2, w3) objfun(c(w1, w2, w3)),
+   vectorize.args=c("w2", "w3"))  # end Vectorize
> # Calculate objective on 2-d (w2 x w3) parameter grid
> w2 <- seq(-3, 7, length=50)
> w3 <- seq(-5, 5, length=50)
> grid_object <- outer(w2, w3, FUN=objvec, w1=1)
> rownames(grid_object) <- round(w2, 2)
> colnames(grid_object) <- round(w3, 2)
> # Perspective plot of objective function
> persp(w2, w3, -grid_object,
+ theta=45, phi=30, shade=0.5,
+ col=rainbow(50), border="green",
+ main="objective function")
```




```
> # Interactive perspective plot of objective function
> library(rgl)
> rgl::persp3d(z=-grid_object, zlab="objective",
+   col="green", main="objective function")
> rgl::persp3d(
+   x=function(w2, w3) {-objvec(w1=1, w2, w3)},
+   xlim=c(-3, 7), ylim=c(-5, 5),
+   col="green", axes=FALSE)
```

Multi-dimensional Portfolio Optimization

The functional optim() performs *multi-dimensional* optimization.

The argument par are the initial parameter values.

The argument fn is the objective function to be minimized.

The argument of the objective function which is to be optimized, must be a vector argument.

optim() accepts additional parameters bound to the dots "..." argument, and passes them to the fn objective function.

The arguments lower and upper specify the search range for the variables of the objective function fn.

method="L-BFGS-B" specifies the quasi-Newton optimization method.

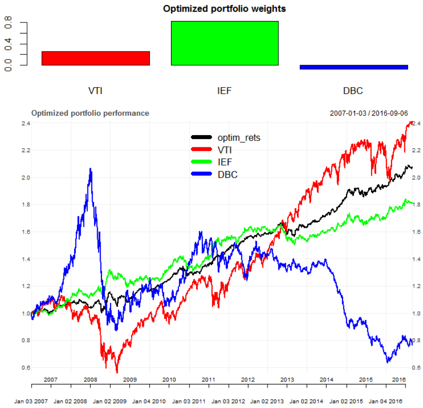
optim() returns a list containing the location of the minimum and the objective function value.

```
> # Optimization to find weights with maximum Sharpe ratio
> optiml <- optim(par=weightv,
+              fn=objfun,
+              retsp=retsp,
+              method="L-BFGS-B",
+              upper=c(1.1, 10, 10),
+              lower=c(0.9, -10, -10))
> # Optimal parameters
> optiml$par
> optiml$par <- optiml$par/sum(optiml$par)
> # Optimal Sharpe ratio
> -objfun(optiml$par)
```

Optimized Portfolio Performance

The optimized portfolio has both long and short positions, and outperforms its individual component assets.

```
> # Plot in two vertical panels
> layout(matrix(c(1,2), 2),
+  widths=c(1,1), heights=c(1,3))
> # barplot of optimal portfolio weights
> barplot(optiml$par, col=c("red", "green", "blue"),
+   main="Optimized portfolio weights")
> # Calculate cumulative returns of VTI, IEF, DBC
> retc <- lapply(retsp,
+   function(retsp) exp(cumsum(retsp)))
> retc <- rutils::do_call(cbind, retc)
> # Calculate optimal portfolio returns with VTI, IEF, DBC
> retsoptim <- cbind(
+   exp(cumsum(retsp %*% optiml$par)),
+   retc)
> colnames(retsoptim)[1] <- "retsoptim"
> # Plot optimal returns with VTI, IEF, DBC
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- c("black", "red", "green", "blue")
> chart_Series(retsoptim, theme=plot_theme,
+        name="Optimized portfolio performance")
> legend("top", legend=colnames(retsoptim), cex=0.8,
+  inset=0.1, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
> # Or plot non-compounded (simple) cumulative returns
> PerformanceAnalytics::chart.CumReturns(
+   cbind(retsp %*% optiml$par, retsp),
+   lwd=2, ylab="", legend.loc="topleft", main="")
```



Package *quadprog* for Quadratic Programming

Quadratic programming (*QP*) is the optimization of quadratic objective functions subject to linear constraints.

Let $O(x)$ be an objective function that is quadratic with respect to a vector variable x:

$$O(x) = \frac{1}{2} x^T \mathbb{Q} x - d^T x$$

Where $\mathbb{Q}$ is a *positive definite* matrix ($x^T \mathbb{Q} x > 0$), and d is a vector.

An example of a *positive definite* matrix is the covariance matrix of linearly independent variables.

Let the linear constraints on the variable x be specified as:

$$\mathbb{A} x \geq b$$

Where $\mathbb{A}$ is a matrix, and b is a vector.

The function `solve.QP()` from package *quadprog* performs optimization of quadratic objective functions subject to linear constraints.

```
> library(quadprog)
> # Minimum variance weights without constraints
> optiml <- solve.QP(Dmat=2*covmat,
+             dvec=rep(0, 2),
+             Amat=matrix(0, nr=2, nc=1),
+             bvec=0)
> # Minimum variance weights sum equal to 1
> optiml <- solve.QP(Dmat=2*covmat,
+             dvec=rep(0, 2),
+             Amat=matrix(1, nr=2, nc=1),
+             bvec=1)
> # Optimal value of objective function
> t(optiml$solution) %*% covmat %*% optiml$solution
> ## Perform simple optimization for reference
> # Objective function for simple optimization
> objfun <- function(x) {
+   x <- c(x, 1-x)
+   t(x) %*% covmat %*% x
+ }  # end objfun
> unlist(optimize(f=objfun, interval=c(-1, 2)))
```

Portfolio Optimization Using Package *quadprog*

The objective function is designed to minimize portfolio variance and maximize its returns:

$$O(x) = \mathbf{w}^T \mathbb{C} \, \mathbf{w} - \mathbf{w}^T \mathbf{r}$$

Where $\mathbb{C}$ is the covariance matrix of returns, $\mathbf{r}$ is the vector of returns, and $\mathbf{w}$ is the vector of portfolio weights.

The portfolio weights $\mathbf{w}$ are constrained as:

$$\mathbf{w}^T \mathbb{1} = \sum_{i=1}^{n} w_i = 1$$

$$0 \leq w_i \leq 1$$

The function `solve.QP()` has the arguments:

`Dmat` and `dvec` are the matrix and vector defining the quadratic objective function.

`Amat` and `bvec` are the matrix and vector defining the constraints.

`meq` specifies the number of equality constraints (the first `meq` constraints are equalities, and the rest are inequalities).

```
> # Calculate daily percentage returns
> symbolv <- c("VTI", "IEF", "DBC")
> retsp <- rutils::etfenv$returns[, symbolv]
> # Calculate the covariance matrix
> covmat <- cov(retsp)
> # Minimum variance weights, with sum equal to 1
> optiml <- quadprog::solve.QP(Dmat=2*covmat,
+             dvec=numeric(3),
+             Amat=matrix(1, nr=3, nc=1),
+             bvec=1)
> # Minimum variance, maximum returns
> optiml <- quadprog::solve.QP(Dmat=2*covmat,
+             dvec=apply(0.1*retsp, 2, mean),
+             Amat=matrix(1, nr=3, nc=1),
+             bvec=1)
> # Minimum variance positive weights, sum equal to 1
> a_mat <- cbind(matrix(1, nr=3, nc=1),
+        diag(3), -diag(3))
> b_vec <- c(1, rep(0, 3), rep(-1, 3))
> optiml <- quadprog::solve.QP(Dmat=2*covmat,
+             dvec=numeric(3),
+             Amat=a_mat,
+             bvec=b_vec,
+             meq=1)
```

Package *DEoptim* for Global Optimization

The function DEoptim() from package *DEoptim* performs *global* optimization using the *Differential Evolution* algorithm.

Differential Evolution is a genetic algorithm which evolves a population of solutions over several generations,
https://link.springer.com/content/pdf/10.1023/A:1008202821328.pdf

The first generation of solutions is selected randomly.

Each new generation is obtained by combining solutions from the previous generation.

The best solutions are selected for creating the next generation.

The *Differential Evolution* algorithm is well suited for very large multi-dimensional optimization problems, such as portfolio optimization.

Gradient optimization methods are more efficient than *Differential Evolution* for smooth objective functions with no local minima.

```
> # Rastrigin function with vector argument for optimization
> rastrigin <- function(vectorv, param=25){
+   sum(vectorv^2 - param*cos(vectorv))
+ }  # end rastrigin
> vectorv <- c(pi/6, pi/6)
> rastrigin(vectorv=vectorv)
> library(DEoptim)
> # Optimize rastrigin using DEoptim
> optiml <-  DEoptim(rastrigin,
+   upper=c(6, 6), lower=c(-6, -6),
+   DEoptim.control(trace=FALSE, itermax=50))
> # Optimal parameters and value
> optiml$optim$bestmem
> rastrigin(optiml$optim$bestmem)
> summary(optiml)
> plot(optiml)
```

Portfolio Optimization Using Package *Deoptim*

The *Differential Evolution* algorithm is well suited for very large multi-dimensional optimization problems, such as portfolio optimization.

```
> # Calculate daily percentage returns
> retsp <- rutils::etfenv$returns[, symbolv]
> # Objective equal to minus Sharpe ratio
> objfun <- function(weightv, retsp) {
+   retsp <- retsp %*% weightv
+   if (sd(retsp) == 0)
+     return(0)
+   else
+     -return(mean(retsp)/sd(retsp))
+ }  # end objfun
> # Perform optimization using DEoptim
> optiml <- DEoptim::DEoptim(fn=objfun,
+   upper=rep(10, NCOL(retsp)),
+   lower=rep(-10, NCOL(retsp)),
+   retsp=retsp,
+   control=list(trace=FALSE, itermax=100, parallelType=1))
> weightv <- optiml$optim$bestmem/sum(abs(optiml$optim$bestmem))
> names(weightv) <- colnames(retsp)
```

Portfolio Optimization Using *Shrinkage*

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

The *shrinkage* technique adds a penalty term to the objective function.

The *elastic net* regularization is a combination of *ridge* regularization and *Lasso* regularization:

$$w_{max} = \arg\max_{w} [\frac{\mathbf{w}^T \mu}{\sigma} - \lambda((1-\alpha) \sum_{i=1}^{n} w_i^2 + \alpha \sum_{i=1}^{n} |w_i|)]$$

The portfolio weights $\mathbf{w}$ are shrunk to zero as the parameters λ and α increase.

```
> # Objective with shrinkage penalty
> objfun <- function(weightv, retsp, lambda, alpha) {
+   retsp <- retsp %*% weightv
+   if (sd(retsp) == 0)
+     return(0)
+   else {
+     penaltyv <- lambda*((1-alpha)*sum(weightv^2) +
+ alpha*sum(abs(weightv)))
+     -return(mean(retsp)/sd(retsp) + penaltyv)
+   }
+ }  # end objfun
> # Objective for equal weight portfolio
> weightv <- rep(1, NROW(symbolv))
> names(weightv) <- symbolv
> lambda <- 0.5 ; alpha <- 0.5
> objfun(weightv, retsp=retsp, lambda=lambda, alpha=alpha)
> # Perform optimization using DEoptim
> optiml <- DEoptim::DEoptim(fn=objfun,
+   upper=rep(10, NCOL(retsp)),
+   lower=rep(-10, NCOL(retsp)),
+   retsp=retsp,
+   lambda=lambda,
+   alpha=alpha,
+   control=list(trace=FALSE, itermax=100, parallelType=1))
> weightv <- optiml$optim$bestmem/sum(abs(optiml$optim$bestmem))
> names(weightv) <- colnames(retsp)
```

Homework Assignment

No homework!