FRE7241 Algorithmic Portfolio Management
Lecture#7, Spring 2023

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

May 9, 2023

The Covariance of Stock Returns

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

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

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

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

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

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

```
> # Select all the ETF symbols except "VXX", "SVXY" "MTUM", "QUAL",
> symbolv <- colnames(rutils::etfenv$returns)
> # VYM has bad data in 2006
> symbolv <- symbolv[!(symbolv %in% c("VXX", "SVXY", "MTUM", "QUAL",
> # Extract columns of rutils::etfenv$returns and overwrite NA value
> retp <- rutils::etfenv$returns[, symbolv]
> retp[1, ] <- 0.01
> nstocks <- NCOL(retp)
> datev <- zoo::index(retp)
> # Calculate the covariance ignoring NA values
> covmat <- cor(retp, use="pairwise.complete.obs")
> sum(is.na(covmat))
> # Calculate the inverse of covmat
> invmat <- solve(covmat)
> # Calculate the generalized inverse of covmat
> invreg <- MASS::ginv(covmat)
> all.equal(unname(invmat), invreg)
```

Generalized Inverse of Singular Covariance Matrix

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

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

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

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

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

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

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

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

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

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

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

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

Portfolio Optimization Strategy

The *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*.

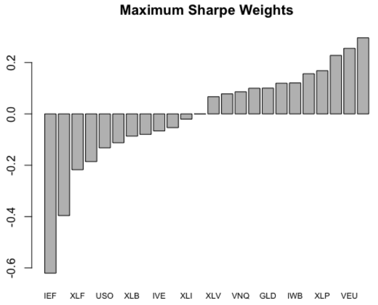
The *portfolio optimization* strategy consists of:

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 $\mathbb{C}$:

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

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



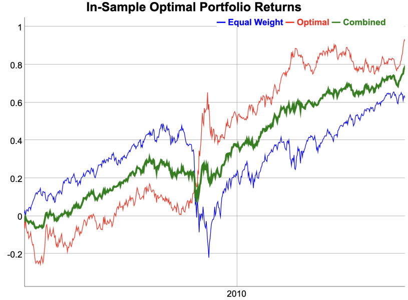

```
> # Plot portfolio weights
> barplot(sort(weightv), main="Maximum Sharpe Weights", cex.names=0.
```

Portfolio Optimization Strategy In-Sample

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

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

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



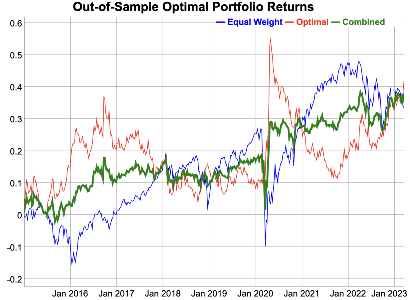

```
> # Dygraph cumulative wealth
> wealthv <- cbind(indeks, pnlis, (pnlis + indeks)/2)
> colnames(wealthv) <- c("Equal Weight", "Optimal", "Combined")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Dygraph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

Portfolio Optimization Strategy Out-of-Sample

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

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

```
> # Calculate the equal weight index
> retos <- retp["2015/"]
> indeks <- xts::xts(rowMeans(retos, na.rm=TRUE), zoo::index(retos))
> # Calculate out-of-sample portfolio returns
> pnlos <- HighFreq::mult_mat(weightv, retos)
> pnlos <- rowMeans(pnlos, na.rm=TRUE)
> pnlos <- pnlos*sd(indeks)/sd(pnlos)
> wealthv <- cbind(indeks, pnlos, (pnlos + indeks)/2)
> colnames(wealthv) <- c("Equal Weight", "Optimal", "Combined")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```



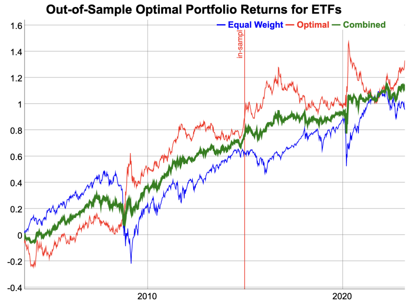

```
> # Dygraph cumulative wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Out-of-Sample Optimal Portfolio Returns") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined",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

Portfolio Optimization Strategy for ETFs

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

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

```
> # Maximum Sharpe weights in-sample interval
> invreg <- MASS::ginv(cov(retis, use="pairwise.complete.obs"))
> weightv <- invreg %*% colMeans(retx["/2014"], na.rm=TRUE)
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retp)
> # Calculate in-sample portfolio returns
> pnlis <- HighFreq::mult_mat(weightv, retis)
> pnlis <- rowMeans(pnlis, na.rm=TRUE)
> # Calculate out-of-sample portfolio returns
> pnlos <- HighFreq::mult_mat(weightv, retos)
> pnlos <- rowMeans(pnlos, na.rm=TRUE)
> # Calculate cumulative wealth
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(indeks)/sd(pnls)
> indeks <- xts::xts(rowMeans(retp, na.rm=TRUE), datev)
> wealthv <- cbind(indeks, pnls, (pnls + indeks)/2)
> colnames(wealthv) <- c("Equal Weight", "Optimal", "Combined")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




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

Dimension Reduction of the Covariance Matrix

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

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

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

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

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

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

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

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

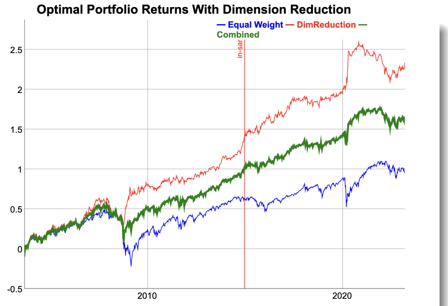
```
> # Calculate in-sample covariance matrix
> covmat <- cov(retis, use="pairwise.complete.obs")
> eigend <- eigen(covmat)
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> # Negative eigenvalues
> eigenval
> # Calculate reduced inverse of covariance matrix
> dimax <- 3
> invred <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
> # Verify inverse property of inverse
> all.equal(covmat, covmat %*% invred %*% covmat)
```

Portfolio Optimization for ETFs with Dimension Reduction

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

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

```
> # Calculate portfolio weights
> weightv <- invred %*% colMeans(retis, na.rm=TRUE)
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retp)
> # Calculate portfolio returns
> pnlis <- HighFreq::mult_mat(weightv, retis)
> pnlis <- rowMeans(pnlis, na.rm=TRUE)
> pnlos <- HighFreq::mult_mat(weightv, retos)
> pnlos <- rowMeans(pnlos, na.rm=TRUE)
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(indeks)/sd(pnls)
> wealthv <- cbind(indeks, pnls, (pnls + indeks)/2)
> colnames(wealthv) <- c("Equal Weight", "DimReduction", "Combined'
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




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

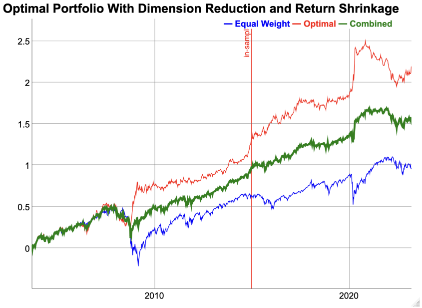
$$r_i' = (1 - \alpha)\, r_i + \alpha\, \bar{r}$$

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"], na.rm=TRUE)
> retxis <- (1-alpha)*retx["/2014"] + alpha*retxm
> # Calculate portfolio weights
> weightv <- invred %*% colMeans(retxis, na.rm=TRUE)
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> # Calculate portfolio returns
> pnlis <- HighFreq::mult_mat(weightv, retis)
> pnlis <- rowMeans(pnlis, na.rm=TRUE)
> pnlos <- HighFreq::mult_mat(weightv, retos)
> pnlos <- rowMeans(pnlos, na.rm=TRUE)
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(indeks)/sd(pnls)
> wealthv <- cbind(indeks, pnls, (pnls + indeks)/2)
> colnames(wealthv) <- c("Equal Weight", "Optimal", "Combined")
```




```
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Dygraph cumulative wealth
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Portfolio With Dimension Reduction and Return Shri
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeW
+   dyLegend(width=300)
```

Rolling Portfolio Optimization Strategy

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

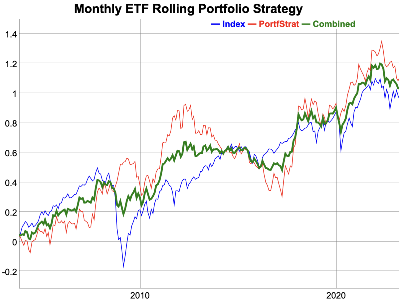
- Calculate the *end points* for portfolio rebalancing,
- Define an objective function for optimizing the portfolio weights,
- Calculate the optimal portfolio weights from the past (in-sample) performance,
- Calculate the out-of-sample returns by applying the portfolio weights to the future returns.

```
> # Define monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> endd <- endd[endd > (nstocks+1)]
> npts <- NROW(endd)
> look_back <- 3
> startp <- c(rep_len(0, look_back), endd[1:(npts-look_back)])
> # Perform loop over end points
> pnls <- lapply(1:(npts-1), function(tday) {
+     # Calculate the portfolio weights
+     retis <- retx[startp[tday]:endd[tday], ]
+     covmat <- cov(retis, use="pairwise.complete.obs")
+     covmat[is.na(covmat)] <- 0
+     invreg <- MASS::ginv(covmat)
+     colm <- colMeans(retis, na.rm=TRUE)
+     colm[is.na(colm)] <- 0
+     weightv <- invreg %*% colm
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the in-sample portfolio returns
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     # Calculate the out-of-sample portfolio returns
+     retos <- retp[(endd[tday]+1):endd[tday+1], ]
+     pnlos <- HighFreq::mult_mat(weightv, retos)
+     pnlos <- rowMeans(pnlos, na.rm=TRUE)
+     pnlos <- pnlos*0.01/sd(pnlos)
+     xts::xts(pnlos, zoo::index(retos))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
```

Rolling Portfolio Strategy Performance

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

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



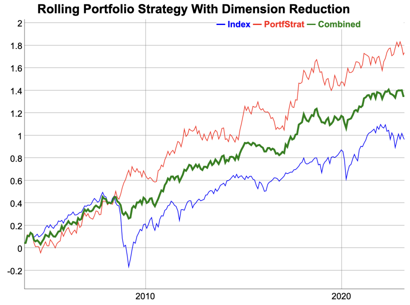

```
> # Dygraph cumulative wealth
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Monthly ETF Rollin
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Rolling Portfolio Strategy With Dimension Reduction

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

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

```
> # Perform loop over end points
> dimax <- 9
> pnls <- lapply(1:(npts-1), function(tday) {
+     # Calculate the portfolio weights
+     retis <- retx[startp[tday]:endd[tday], ]
+     covmat <- cov(retis, use="pairwise.complete.obs")
+     covmat[is.na(covmat)] <- 0
+     eigend <- eigen(covmat)
+     eigenvec <- eigend$vectors
+     eigenval <- eigend$values
+     invred <- eigenvec[, 1:dimax] %*%
+ (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
+     colm <- colMeans(retis, na.rm=TRUE)
+     colm[is.na(colm)] <- 0
+     weightv <- invred %*% colm
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the in-sample portfolio returns
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     # Calculate the out-of-sample portfolio returns
+     retos <- retp[(endd[tday]+1):endd[tday+1], ]
+     pnlos <- HighFreq::mult_mat(weightv, retos)
+     pnlos <- rowMeans(pnlos, na.rm=TRUE)
+     pnlos <- pnlos*0.01/sd(pnlos)
+     xts::xts(pnlos, zoo::index(retos))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls)
```



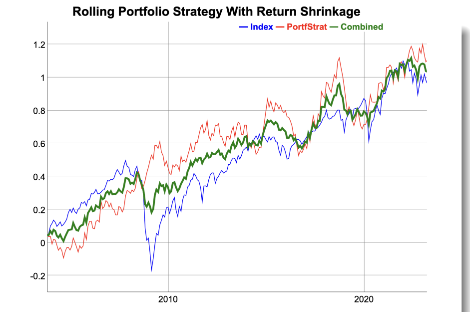

```
> # Calculate the Sharpe and Sortino ratios
> pnls <- pnls*sd(indeks)/sd(pnls)
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "PortfStrat", "Combined")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Dygraph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

Rolling Portfolio Strategy With Return Shrinkage

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

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

```
> alpha <- 0.7 # Return shrinkage intensity
> # Perform loop over end points
> pnls <- lapply(1:(npts-1), function(tday) {
+     # Shrink the in-sample returns to their mean
+     retis <- retx[startp[tday]:endd[tday], ]
+     rowm <- rowMeans(retis, na.rm=TRUE)
+     rowm[is.na(rowm)] <- 0
+     retis <- (1-alpha)*retis + alpha*rowm
+     # Calculate the portfolio weights
+     covmat <- cov(retis, use="pairwise.complete.obs")
+     covmat[is.na(covmat)] <- 0
+     eigend <- eigen(covmat)
+     eigenvec <- eigend$vectors
+     eigenval <- eigend$values
+     invred <- eigenvec[, 1:dimax] %*%
+ (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
+     colm <- colMeans(retis, na.rm=TRUE)
+     colm[is.na(colm)] <- 0
+     weightv <- invred %*% colm
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the in-sample portfolio returns
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     # Calculate the out-of-sample portfolio returns
+     retos <- retp[(endd[tday]+1):endd[tday+1], ]
+     pnlos <- HighFreq::mult_mat(weightv, retos)
+     pnlos <- rowMeans(pnlos, na.rm=TRUE)
+     pnlos <- pnlos*0.01/sd(pnlos)
+     xts::xts(pnlos, zoo::index(retos))
+ })  # end lapply
> pnls <- do.call(rbind, pnls)
```




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

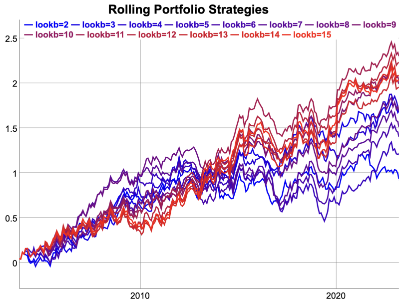
Function for Rolling Portfolio Optimization Strategy

```
> # Define backtest functional for rolling portfolio strategy
> roll_portf <- function(retx, # Excess returns
+                  retp, # Stock returns
+                  endd, # End points
+                  look_back=12, # Look-back interval
+                  dimax=3, # Dimension reduction parameter
+                  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(1:(npts-1), function(tday) {
+     retis <- retx[startp[tday]:endd[tday], ]
+     # Shrink the in-sample returns to their mean
+     if (alpha > 0) {
+ rowm <- rowMeans(retis, na.rm=TRUE)
+ rowm[is.na(rowm)] <- 0
+ retis <- (1-alpha)*retis + alpha*rowm
+     } # end if
+     # Calculate the portfolio weights
+     covmat <- cov(retis, use="pairwise.complete.obs")
+     covmat[is.na(covmat)] <- 0
+     eigend <- eigen(covmat)
+     eigenvec <- eigend$vectors
+     eigenval <- eigend$values
+     invred <- eigenvec[, 1:dimax] %*% (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
+     colm <- colMeans(retis, na.rm=TRUE)
+     colm[is.na(colm)] <- 0
+     weightv <- invred %*% colm
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the in-sample portfolio returns
+     pnlis <- HighFreq::mult_mat(weightv, retis)
+     pnlis <- rowMeans(pnlis, na.rm=TRUE)
+     # Calculate the out-of-sample portfolio returns
+     retos <- retp[(endd[tday]+1):endd[tday+1], ]
+     pnlos <- HighFreq::mult_mat(weightv, retos)
+     pnlos <- rowMeans(pnlos, na.rm=TRUE)
+     pnlos <- pnlos*0.01/sd(pnlos)
```

Rolling Portfolio Optimization With Different Look-backs

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

```
> # Simulate a monthly ETF portfolio strategy
> pnls <- roll_portf(retx=retx, retp=retp, endd=endd,
+   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,
+   retp=retp, retx=retx, endd=endd, dimax=dimax)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lookb=", look_backs)
> pnlsums <- sapply(pnls, sum)
> look_back <- look_backs[which.max(pnlsums)]
```

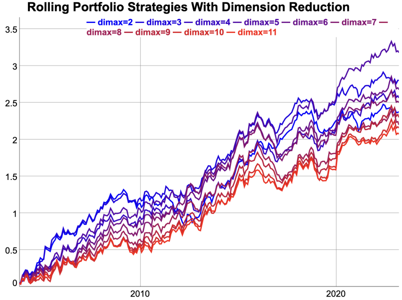



```
> # Plot dygraph of monthly ETF portfolio strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd], main="Rolling Portfolio Stra
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=600)
> # Plot EWMA strategies using 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
> dimaxs <- 2:11
> pnls <- lapply(dimaxs, roll_portf, retx=retx,
+   retp=retp, endd=endd, look_back=look_back)
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("dimax=", dimaxs)
> pnlsums <- sapply(pnls, sum)
> dimax <- dimaxs[which.max(pnlsums)]
```



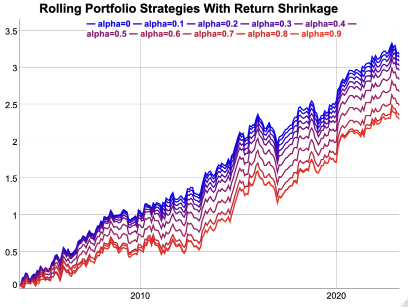

```
> # Plot dygraph of monthly ETF portfolio strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies With Dimension Reduction") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # 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, retx=retx,
+   retp=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)]
```



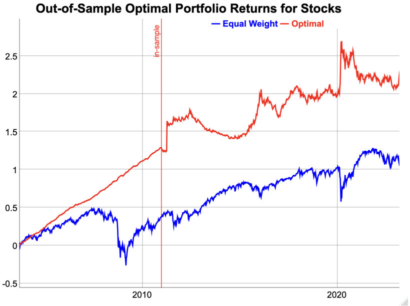

```
> # Plot dygraph of monthly ETF portfolio strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Rolling Portfolio Strategies With Return Shrinkage") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
> # Plot 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 *overfit* in the *in-sample* interval.

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

```
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RData
> # Overwrite NA values in returns
> retp <- returns
> nstocks <- NCOL(retp)
> retp[is.na(retp)] <- 0
> sum(is.na(retp))
> datev <- zoo::index(retp)
> riskf <- 0.03/252
> retx <- (retp - riskf)
> retis <- retp["/2010"]
> retos <- retp["2011/"]
> # Maximum Sharpe weights in-sample interval
> covmat <- cov(retis, use="pairwise.complete.obs")
> invreg <- MASS::ginv(covmat)
> weightv <- invreg %*% colMeans(retx["/2010"], na.rm=TRUE)
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retp)
> # Calculate portfolio returns
> pnlis <- (retis %*% weightv)
> pnlos <- (retos %*% weightv)
> indeks <- xts::xts(rowMeans(retp), datev)
> # Combine in-sample and out-of-sample returns
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(indeks)/sd(pnls)
> wealthv <- cbind(indeks, pnls)
> colnames(wealthv) <- c("Equal Weight", "Optimal")
```



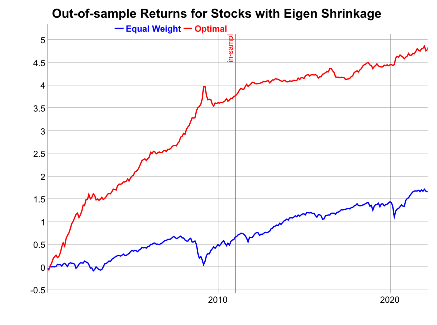

```
> # Calculate the in-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[index(retis)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[index(retos)],
+   function(x) c(Sharpe=mean(x)/sd(x),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

Portfolio Strategy for Stocks with Dimension Reduction

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

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

```
> # Calculate reduced inverse of covariance matrix
> dimax <- 3
> eigend <- eigen(cov(retis, use="pairwise.complete.obs"))
> eigenvec <- eigend$vectors
> eigenval <- eigend$values
> invred <- eigenvec[, 1:dimax] %*%
+   (t(eigenvec[, 1:dimax]) / eigenval[1:dimax])
> # Calculate portfolio weights
> weightv <- invred %*% colMeans(retx["/2010"], na.rm=TRUE)
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> names(weightv) <- colnames(retp)
> # Calculate portfolio returns
> pnlis <- (retis %*% weightv)
> pnlos <- (retos %*% weightv)
```




```
> # Combine in-sample and out-of-sample returns
> pnls <- c(pnlis, pnlos)
> pnls <- pnls*sd(indeks)/sd(pnls)
> wealthv <- cbind(indeks, pnls)
> colnames(wealthv) <- c("Equal Weight", "Optimal")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv[index(retos)],
+   function(x) c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot of cumulative portfolio returns
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Out-of-Sample Returns for Stocks with Dimension Reduction"
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyEvent(zoo::index(last(retis[, 1])), label="in-sample", strokeP
+   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 $\bar{r}$:

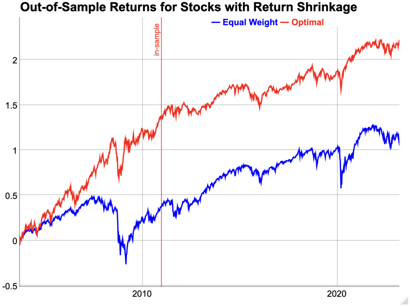
$$r_i' = (1 - \alpha)\, r_i + \alpha\, \bar{r}$$

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 <- invred %*% colMeans(retxis, na.rm=TRUE)
> weightv <- drop(weightv/sqrt(sum(weightv^2)))
> # Calculate portfolio returns
> pnlis <- (retis %*% weightv)
> pnlos <- (retos %*% weightv)
```




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

Fast Covariance Matrix Inverse Using *RcppArmadillo*

RcppArmadillo can be used to quickly calculate the reduced 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)
> # Reduced inverse of covariance matrix
> dimax <- 3
> eigend <- eigen(covmat)
> invred <- eigend$vectors[, 1:dimax] %*%
+   (t(eigend$vectors[, 1:dimax]) / eigend$values[1:dimax])
> # Reduced inverse using RcppArmadillo
> invarma <- calc_inv(covmat, dimax)
> all.equal(invred, invarma)
> # Microbenchmark RcppArmadillo code
> library(microbenchmark)
> summary(microbenchmark(
+   rcode={eigend <- eigen(covmat)
+     eigend$vectors[, 1:dimax] %*%
+ (t(eigend$vectors[, 1:dimax]) / eigend$values[1:dimax])
+   },
+   rcpp=calc_inv(covmat, dimax),
+   times=10))[, c(1, 4, 5)]  # end microbenchmark summary
```

```
arma::mat calc_inv(const arma::mat& 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 reduced inverse using SVD decomposi

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

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

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

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

  }  // end if

}  // end calc_inv
```

Portfolio Optimization Using *RcppArmadillo*

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

```
arma::vec calc_weights(const arma::mat& returns, // Asset returns
                       Rcpp::List controlv) { // List of portfolio optimization parameters

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

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

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

Strategy Backtesting Using *RcppArmadillo*

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

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

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

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

  // Return the strategy pnls
  return pnls;

} // end back_test
```

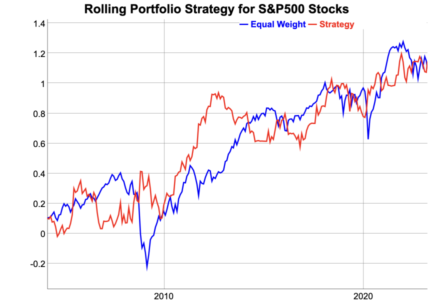
Rolling Portfolio Strategy for S&P500 Stocks

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[is.na(retp)] <- 0
> retx <- (retp - riskf)
> nstocks <- NCOL(retp) ; datev <- zoo::index(retp)
> # Define monthly end points
> endd <- rutils::calc_endpoints(retp, interval="months")
> endd <- endd[endd > (nstocks+1)]
> npts <- NROW(endd) ; look_back <- 12
> startp <- c(rep_len(0, look_back), endd[1:(npts-look_back)])
> # Perform loop over end points - takes long
> pnls <- lapply(1:(npts-1), function(tday) {
+     # Subset the excess returns
+     retis <- retx[startp[tday]:endd[tday], ]
+     invreg <- MASS::ginv(cov(retis, use="pairwise.complete.obs"))
+     # Calculate the maximum Sharpe ratio portfolio weights
+     weightv <- invreg %*% colMeans(retis, na.rm=TRUE)
+     weightv <- drop(weightv/sqrt(sum(weightv^2)))
+     # Calculate the out-of-sample portfolio returns
+     retos <- retp[(endd[tday]+1):endd[tday+1], ]
+     xts::xts(retos %*% weightv, zoo::index(retos))
+ })  # end lapply
> pnls <- rutils::do_call(rbind, pnls)
```

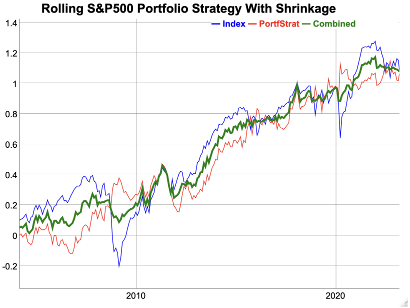



```
> # Calculate returns of equal weight portfolio
> indeks <- xts::xts(rowMeans(retp), datev)
> pnls <- rbind(indeks[paste0("/", start(pnls)-1)], pnls*sd(indeks)/
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> colnames(wealthv) <- c("Equal Weight", "Strategy")
> sqrt(252)*sapply(wealthv, 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 Strategy for S&P500 Stocks") %>%
+   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=dimax,
> # Perform backtest in Rcpp
> pnls <- HighFreq::back_test(retx=retx, retp=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", "PortfStrat", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling S&P500 Portfolio Strategy With Shrinkage") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

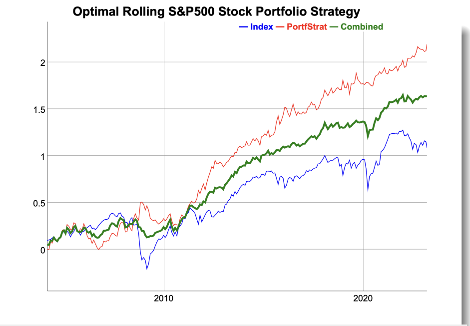
Optimal Dimension Reduction And Shrinkage Parameters

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=15, which means relatively weak dimension reduction.

The best return shrinkage parameter for this portfolio of stocks is equal to $\alpha = 0.71$, 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) {
+   controlv <- HighFreq::param_portf(method="maxsharpe",
+       dimax=dimax, alpha=alpha)
+   HighFreq::back_test(retx=retx, retp=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 parameters
> dimaxs <- seq(from=3, to=40, by=2)
> pnls <- lapply(dimaxs, function(dimax) {
+   controlv <- HighFreq::param_portf(method="maxsharpe",
+     dimax=dimax, alpha=alpha)
+   HighFreq::back_test(retx=retx, retp=retp,
+     startp=startp, endd=endd, controlv=controlv)
+ })  # end lapply
> profilev <- sapply(pnls, sum)
```



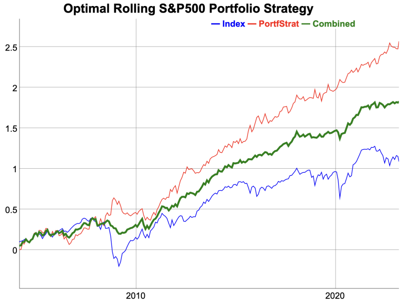

```
> plot(x=dimaxs, y=profilev, t="l", xlab="dimax", ylab="pnl",
+   main="Strategy PnL as Function of dimax")
> whichmax <- which.max(profilev)
> dimax <- dimaxs[whichmax]
> pnls <- pnls[[whichmax]]
> pnls <- pnls*sd(indeks)/sd(pnls)
> # Plot cumulative strategy returns
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "PortfStrat", "Combined")
> # Calculate the out-of-sample Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Rolling S&P500 Stock Portfolio Strategy") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   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=11 months, which roughly agrees with the research literature on momentum strategies.

```
> # Create list of model parameters
> controlv <- HighFreq::param_portf(method="maxsharpe",
+       dimax=dimax, alpha=alpha)
> # Perform backtest over look-backs
> look_backs <- seq(from=5, to=16, 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(retx=retx, retp=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)
```




```
> # Calculate the out-of-sample Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls, (pnls+indeks)/2)
> colnames(wealthv) <- c("Index", "PortfStrat", "Combined")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Dygraph the cumulative wealth
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimal Rolling S&P500 Portfolio Strategy") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

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:

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

The *Minimum Variance* Portfolio

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 $\mathbb{C}$.

Returns and Variance of the *Minimum Variance* Portfolio

The stock 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 daily returns of the *minimum variance* portfolio are equal to:

$$\mathbf{r}_{mv} = \frac{\mathbf{r}^T \mathbb{C}^{-1} \mathbb{1}}{\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}} = \frac{\mathbf{r}^T \mathbb{C}^{-1} \mathbb{1}}{c_{11}}$$

Where $\mathbf{r}$ are the daily stock returns, and $c_{11} = \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$.

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

$$\sigma^2_{mv} = \mathbf{w}^T \mathbb{C} \, \mathbf{w} = \frac{\mathbb{1}^T \mathbb{C}^{-1} \mathbb{C} \, \mathbb{C}^{-1} \mathbb{1}}{(\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1})^2} = \frac{1}{\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}} = \frac{1}{c_{11}}$$

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.

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

```
> # Calculate daily stock returns
> symbolv <- c("VTI", "IEF", "DBC")
> nstocks <- NROW(symbolv)
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> # Calculate covariance matrix of returns and its inverse
> covmat <- cov(retp)
> covinv <- solve(a=covmat)
> unitv <- rep(1, nstocks)
> # Calculate the minimum variance weights
> c11 <- drop(t(unitv) %*% covinv %*% unitv)
> weightmv <- drop(covinv %*% unitv/c11)
> # Calculate the daily minvar portfolio returns in two ways
> retmv <- (retp %*% weightmv)
> all.equal(retmv, (retp %*% covinv %*% unitv)/c11)
> # Calculate the minimum variance in three ways
> all.equal(var(retmv),
+   t(weightmv) %*% covmat %*% weightmv,
+   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: $\mathbf{w}^T \bar{\mathbf{r}} = \sum_{i=1}^{n} w_i \bar{r}_i = r_t$.

Where $\bar{\mathbf{r}}$ are the mean stock returns.

The stock 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:

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

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

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

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

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

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

The above can be written in matrix notation as:

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

Or:

$$\begin{bmatrix} c_{11} & c_{r1} \\ c_{r1} & c_{rr} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \mathbb{F}\lambda = 2 \begin{bmatrix} 1 \\ r_t \end{bmatrix} = 2u$$

With $c_{11} = \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$, $c_{r1} = \mathbb{1}^T \mathbb{C}^{-1} \bar{\mathbf{r}}$, $c_{rr} = \bar{\mathbf{r}}^T \mathbb{C}^{-1} \bar{\mathbf{r}}$, $\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$, $u = \begin{bmatrix} 1 \\ r_t \end{bmatrix}$, and $\mathbb{F} = u^T \mathbb{C}^{-1} u = \begin{bmatrix} c_{11} & c_{r1} \\ c_{r1} & c_{rr} \end{bmatrix}$.

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:

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

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

$$\frac{1}{\det \mathbb{F}}\begin{bmatrix} \mathbb{C}^{-1}\mathbb{1} \\ \mathbb{C}^{-1}\bar{\mathbf{r}} \end{bmatrix}^T \begin{bmatrix} c_{rr} & -c_{r1} \\ -c_{r1} & c_{11} \end{bmatrix} \begin{bmatrix} 1 \\ r_t \end{bmatrix} =$$

$$\frac{(c_{rr} - c_{r1} r_t)\,\mathbb{C}^{-1}\mathbb{1} + (c_{11} r_t - c_{r1})\,\mathbb{C}^{-1}\bar{\mathbf{r}}}{\det \mathbb{F}}$$

With $c_{11} = \mathbb{1}^T\mathbb{C}^{-1}\mathbb{1}$, $c_{r1} = \mathbb{1}^T\mathbb{C}^{-1}\bar{\mathbf{r}}$, $c_{rr} = \bar{\mathbf{r}}^T\mathbb{C}^{-1}\bar{\mathbf{r}}$. And $\det \mathbb{F} = c_{11} c_{rr} - c_{r1}^2$ is the determinant of the matrix $\mathbb{F}$.

The above formula shows that the efficient portfolio weights are a linear function of the target return.

Therefore 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
> retm <- colMeans(retp)
> # Specify the target return
> retarget <- 1.5*mean(retp)
> # Products of inverse with mean returns and unit vector
> c11 <- drop(t(unitv) %*% covinv %*% unitv)
> cr1 <- drop(t(unitv) %*% covinv %*% retm)
> crr <- drop(t(retm) %*% covinv %*% retm)
> fmat <- matrix(c(c11, cr1, cr1, crr), nc=2)
> # Solve for the Lagrange multipliers
> lagm <- solve(a=fmat, b=c(2, 2*retarget))
> # Calculate the efficient portfolio weights
> weightv <- 0.5*drop(covinv %*% cbind(unitv, retm) %*% lagm)
> # Calculate constraints
> all.equal(1, sum(weights))
> all.equal(retarget, sum(retm*weightv))
```

Variance of the *Efficient Portfolios*

The *efficient portfolio* variance is equal to:

$$\sigma^2 = \mathbf{w}^T \mathbb{C}\, \mathbf{w} = \frac{1}{4} \lambda^T \mathbb{F}\, \lambda = u^T \mathbb{F}^{-1}\, u =$$

$$\frac{1}{\det \mathbb{F}} \begin{bmatrix} 1 \\ r_t \end{bmatrix}^T \begin{bmatrix} c_{rr} & -c_{r1} \\ -c_{r1} & c_{11} \end{bmatrix} \begin{bmatrix} 1 \\ r_t \end{bmatrix} =$$

$$\frac{c_{11} r_t^2 - 2 c_{r1} r_t + c_{rr}}{\det \mathbb{F}}$$

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 the minimum variance portfolio: $r_{mv} = c_{r1}/c_{11} = \mathbb{1}^T \mathbb{C}^{-1} \bar{\mathbf{r}} / \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$ and $\sigma_{mv}^2 = 1/c_{11} = 1/\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$.

The *efficient portfolio* variance can be expressed in terms of the difference $\Delta_r = r_t - r_{mv}$ as:

$$\sigma^2 = \frac{\Delta_r^2 + \det \mathbb{F}}{c_{11} \det \mathbb{F}}$$

So that if $\Delta_r = 0$ then $\sigma^2 = 1/c_{11}$.

Where $\det \mathbb{F} = c_{11} c_{rr} - c_{r1}^2$ is the determinant of the matrix $\mathbb{F}$.

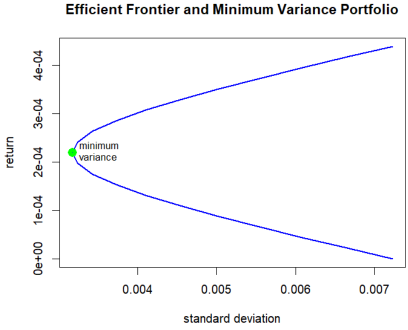
```
> # Calculate the efficient portfolio returns
> reteff <- drop(retp %*% weightv)
> reteffm <- mean(reteff)
> all.equal(reteffm, retarget)
> # Calculate the efficient portfolio variance in three ways
> uu <- c(1, retarget)
> finv <- solve(fmat)
> detf <- (c11*crr-cr1^2)  # det(fmat)
> all.equal(var(reteff),
+   drop(t(uu) %*% finv %*% uu),
+   (c11*reteffm^2-2*cr1*reteffm+crr)/detf)
```

The *Efficient Frontier*

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

The *efficient frontier* is the plot of the target returns r_t and the standard deviations of the *efficient portfolios*, which is a *hyperbola*.

```
> # Calculate the daily and mean minvar portfolio returns
> c11 <- drop(t(unitv) %*% covinv %*% unitv)
> weightv <- drop(covinv %*% unitv/c11)
> retmv <- (retp %*% weightv)
> retmvm <- sum(weightv*retm)
> # Calculate the minimum variance
> varmv <- 1/c11
> stdevmv <- sqrt(varmv)
> # Calculate efficient frontier from target returns
> targetv <- retmvm*(1+seq(from=(-1), to=1, by=0.1))
> stdevs <- sapply(targetv, function(rett) {
+   uu <- c(1, rett)
+   sqrt(drop(t(uu) %*% finv %*% uu))
+ })  # end sapply
```




```
> # Plot the efficient frontier
> plot(x=stdevs, y=targetv, t="l", col="blue", lwd=2,
+      main="Efficient Frontier and Minimum Variance Portfolio",
+      xlab="standard deviation", ylab="return")
> points(x=stdevmv, y=retmvm, col="green", lwd=6)
> text(x=stdevmv, y=retmvm, labels="minimum \nvariance",
+      pos=4, cex=0.8)
```

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

The *tangent* line connects the risk-free point $(\sigma = 0, r = r_f)$ with a single tangent point on the *efficient frontier*.

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 efficient portfolio variance σ^2 by the target return r_t:

$$\frac{d\sigma^2}{dr_t} = 2\sigma \frac{d\sigma}{dr_t} = \frac{2c_{11} r_t - 2c_{r1}}{\det \mathbb{F}}$$

$$\frac{d\sigma}{dr_t} = \frac{c_{11} r_t - c_{r1}}{\sigma \, \det \mathbb{F}}$$

$$\beta = \frac{\sigma \, \det \mathbb{F}}{c_{11} r_t - c_{r1}}$$

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:

$$r_f = r_t - \sigma \, \beta = r_t - \frac{\sigma^2 \, \det \mathbb{F}}{c_{11} r_t - c_{r1}} =$$

$$r_t - \frac{c_{11} r_t^2 - 2c_{r1} r_t + c_{rr}}{\det \mathbb{F}} \frac{\det \mathbb{F}}{c_{11} r_t - c_{r1}} =$$

$$r_t - \frac{c_{11} r_t^2 - 2c_{r1} r_t + c_{rr}}{c_{11} r_t - c_{r1}} = \frac{c_{r1} r_t - c_{rr}}{c_{11} r_t - c_{r1}}$$

```
> # Calculate standard deviation of efficient portfolio
> uu <- c(1, retarget)
> stdeveff <- sqrt(drop(t(uu) %*% finv %*% uu))
> # Calculate the slope of the tangent line
> detf <- (c11*crr-cr1^2)  # det(fmat)
> sharper <- (stdeveff*detf)/(c11*retarget-cr1)
> # Calculate the risk-free rate as intercept of the tangent line
> riskf <- retarget - sharper*stdeveff
> # Calculate the risk-free rate from target return
> all.equal(riskf,
+   (retarget*cr1-crr)/(retarget*c11-cr1))
```

The *Capital Market Line*

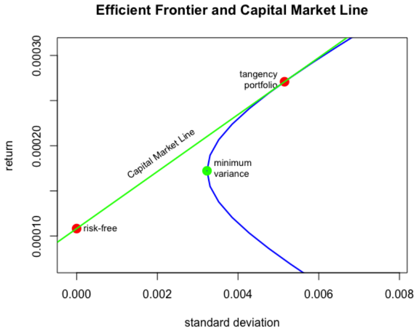
The *Capital Market Line* (CML) is the tangent line connecting the risk-free point ($\sigma = 0, r = r_f$) with a single tangent point on the *efficient frontier*.

The *tangency portfolio* is the *efficient portfolio* at the tangent point corresponding to the given *risk-free* rate.

Each value of the *risk-free* rate r_f corresponds to a unique *tangency portfolio*.

For a given *risk-free* rate r_f, the *tangency portfolio* has the highest *Sharpe ratio* among all the *efficient portfolios*.

```
> # Plot efficient frontier
> aspratio <- 1.0*max(stdevs)/diff(range(targetv))
> plot(x=stdevs, y=targetv, t="l", col="blue", lwd=2, asp=aspratio,
+      xlim=c(0.4, 0.6)*max(stdevs), ylim=c(0.2, 0.9)*max(targetv),
+      main="Efficient Frontier and Capital Market Line",
+      xlab="standard deviation", ylab="return")
> # Plot the minimum variance portfolio
> points(x=stdevmv, y=retmvm, col="green", lwd=6)
> text(x=stdevmv, y=retmvm, labels="minimum \nvariance",
+      pos=4, cex=0.8)
```




```
> # Plot the tangent portfolio
> points(x=stdeveff, y=retarget, col="red", lwd=6)
> text(x=stdeveff, y=retarget, labels="tangency\nportfolio", pos=2,
> # Plot the 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 the tangent line
> abline(a=riskf, b=sharper, lwd=2, col="green")
> text(x=0.6*stdev, y=0.8*retarget,
+      labels="Capital Market Line", pos=2, cex=0.8,
+      srt=180/pi*atan(aspratio*sharper))
```

The *Capital Market Line* Portfolios

The points on the *Capital Market Line* represent portfolios consisting of the *tangency portfolio* and the *risk-free* asset (bond).

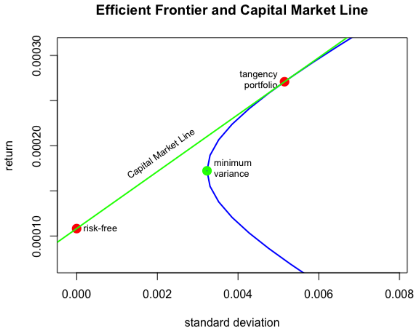
The *Capital Market Line* represents delevered and levered portfolios, consisting of the *tangency portfolio* combined with the *risk-free* asset (bond).

The *CML* portfolios have weights proportional to the tangency portfolio weights.

The *CML* portfolios above the tangent point are levered with respect to the *tangency portfolio* through borrowing at the *risk-free* rate r_f. Their weights are equal to the tangency portfolio weights multiplied by a factor greater than 1.

The *CML* portfolios below the tangent point are delevered with respect to the *tangency portfolio* through investing at the *risk-free* rate r_f. Their weights are equal to the tangency portfolio weights multiplied by a factor less than 1.

All the *CML* portfolios have the same *Sharpe ratio*.



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 $\mu = \bar{\mathbf{r}} - r_f$ is the vector of mean 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 mean 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: $\bar{\mathbf{r}}^T \mathbf{w} = \sum_{i=1}^{n} w_i r_i = r_t$.

The mean return of the *maximum Sharpe* portfolio is equal to:

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

$$\frac{\bar{\mathbf{r}}^T \mathbb{C}^{-1} \mathbb{1}\, r_f - \bar{\mathbf{r}}^T \mathbb{C}^{-1} \mathbb{1}\, \bar{\mathbf{r}}}{\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}\, r_f - \bar{\mathbf{r}}^T \mathbb{C}^{-1} \mathbb{1}} = \frac{c_{r1}\, r_f - c_{rr}}{c_{11}\, r_f - c_{r1}}$$

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

Returns and Variance of the *Maximum Sharpe* Portfolio

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

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

The mean return of the *maximum Sharpe* portfolio is equal to:

$$r_t = \bar{\mathbf{r}}^T\mathbf{w} = \frac{\bar{\mathbf{r}}^T\mathbb{C}^{-1}\mu}{\mathbb{1}^T\mathbb{C}^{-1}\mu} = \frac{c_{r1}\, r_f - c_{rr}}{c_{11}\, r_f - c_{r1}}$$

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

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

$$\frac{(\bar{\mathbf{r}} - r_f)^T\mathbb{C}^{-1}(\bar{\mathbf{r}} - r_f)}{(\mathbb{1}^T\mathbb{C}^{-1}(\bar{\mathbf{r}} - r_f))^2} = \frac{c_{11}r_t^2 - 2c_{r1}r_t + c_{rr}}{\det\mathbb{F}}$$

The above formula expresses the *maximum Sharpe* portfolio variance as a function of its mean return r_t.

The *maximum Sharpe* ratio is equal to:

$$SR = \frac{\mathbf{w}^T\mu}{\sigma} = \frac{\mu^T\mathbb{C}^{-1}\mu}{\mathbb{1}^T\mathbb{C}^{-1}\mu} / \frac{\sqrt{\mu^T\mathbb{C}^{-1}\mu}}{\mathbb{1}^T\mathbb{C}^{-1}\mu} =$$

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

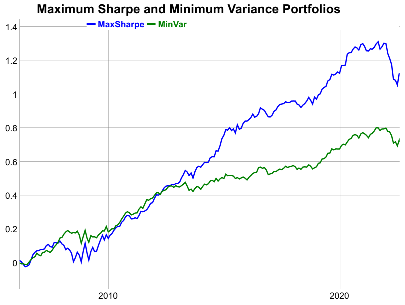
```
> # Calculate the mean excess returns
> riskf <- retarget - sharper*stdeveff
> retx <- (retm - riskf)
> # Calculate the efficient portfolio weights
> weightv <- 0.5*drop(covinv %*% cbind(unitv, retm) %*% lagm)
> # Calculate the maximum Sharpe weights
> weightms <- drop(covinv %*% retx)/sum(covinv %*% retx)
> all.equal(weightv, weightms)
> # Calculate the maximum Sharpe mean return in two ways
> all.equal(sum(retm*weightv),
+   (cr1*riskf-crr)/(c11*riskf-cr1))
> # Calculate the maximum Sharpe daily returns
> retd <- (retp %*% weightms)
> # Calculate the maximum Sharpe variance in four ways
> detf <- (c11*crr-cr1^2)  # det(fmat)
> all.equal(var(retd),
+   t(weights) %*% covmat %*% weightv,
+   (t(retx) %*% covinv %*% retx)/sum(covinv %*% retx)^2,
+   (c11*retarget^2-2*cr1*retarget+crr)/detf)
> # Calculate the maximum Sharpe ratio
> sqrt(252)*sum(weightv*retx)/
+   sqrt(drop(t(weights) %*% covmat %*% weightv))
> # Calculate the stock Sharpe ratios
> sqrt(252)*sapply((retp - riskf), function(x) mean(x)/sd(x))
```

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.

The *maximum Sharpe* portfolio has both a higher Sharpe ratio and higher absolute returns.

```
> # Calculate optimal portfolio returns
> wealthv <- cbind(retp %*% weightms, retp %*% weightmv)
> wealthv <- xts::xts(wealthv, zoo::index(retp))
> colnames(wealthv) <- c("MaxSharpe", "MinVar")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv,
+   function(x) c(Sharpe=(mean(x)-riskf)/sd(x), Sortino=(mean(x)-ris
> # Plot the log wealth
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Maximum Sharpe and Minimum Variance Portfolios") %>%
+   dyOptions(colors=c("blue", "green"), strokeWidth=2) %>%
+   dyLegend(show="always", width=500)
```



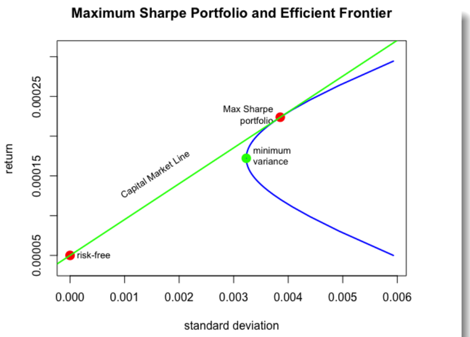
The *Maximum Sharpe* Portfolios and the *Efficient Frontier*

The *maximum Sharpe* portfolios are *efficient portfolios*, so they form the *efficient frontier*.

A *market portfolio* is the portfolio of all the available assets, with weights proportional to their market capitalizations.

The *maximum Sharpe* portfolio is sometimes considered to be the *market portfolio*, because it's the optimal portfolio for the given value of the risk-free rate r_f.

```
> # Calculate the maximum Sharpe portfolios for different risk-free
> detf <- (c11*crr-cr1^2)  # det(fmat)
> riskfv <- retmvm*seq(from=1.3, to=20, by=0.1)
> riskfv <- c(riskfv, retmvm*seq(from=(-20), to=0.7, by=0.1))
> effront <- sapply(riskfv, function(riskf) {
+   # Calculate the maximum Sharpe mean return
+   reteffm <- (cr1*riskf-crr)/(c11*riskf-cr1)
+   # Calculate the maximum Sharpe standard deviation
+   stdev <- sqrt((c11*reteffm^2-2*cr1*reteffm+crr)/detf)
+   c(return=reteffm, stdev=stdev)
+ })  # end sapply
> effront <- effront[, order(effront["return", ])]
> # Plot the efficient frontier
> reteffv <- effront["return", ]
> stdevs <- effront["stdev", ]
> aspratio <- 0.6*max(stdevs)/diff(range(reteffv))
> plot(x=stdevs, y=reteffv, t="l", col="blue", lwd=2, asp=aspratio
+   main="Maximum Sharpe Portfolio and Efficient Frontier",
+   xlim=c(0.0, max(stdevs)), xlab="standard deviation", ylab="retu
> # Plot the minimum variance portfolio
> points(x=stdevmv, y=retmvm, col="green", lwd=6)
> text(x=stdevmv, y=retmvm, labels="minimum \nvariance", pos=4, ce
```




```
> # Calculate the maximum Sharpe return and standard deviation
> riskf <- min(reteffv)
> retmax <- (cr1*riskf-crr)/(c11*riskf-cr1)
> stdevmax <- sqrt((c11*retmax^2-2*cr1*retmax+crr)/detf)
> # Plot the maximum Sharpe portfolio
> points(x=stdevmax, y=retmax, col="red", lwd=6)
> text(x=stdevmax, y=retmax, labels="Max Sharpe\nportfolio", pos=2,
> # Plot the 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 the tangent line
> sharper <- (stdevmax*detf)/(c11*retmax-cr1)
> abline(a=riskf, b=sharper, lwd=2, col="green")
> text(x=0.6*stdevmax, y=0.8*retmax, labels="Capital Market Line",
+      pos=2, cex=0.8, srt=180/pi*atan(aspratio*sharper))
```

Efficient Portfolios and Their Tangent Lines

The *efficient frontier* consists of all the *maximum Sharpe* portfolios corresponding to different values of the risk-free rate.

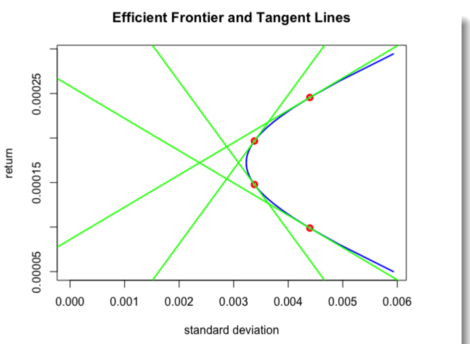
The target return can be expressed as a function of the risk-free rate as:

$$r_t = \frac{c_{r1} \, r_f - c_{rr}}{c_{11} \, r_f - c_{r1}}$$

If $r_f \rightarrow \pm\infty$ then $r_t \rightarrow r_{mv} = c_{r1}/c_{11}$.

But if the risk-free rate tends to the mean returns of the minimum variance portfolio: $r_f \rightarrow r_{mv} = c_{r1}/c_{11}$, then $r_t \rightarrow \pm\infty$, which means that there is no efficient portfolio corresponding to the risk-free rate equal to the mean returns of the minimum variance portfolio: $r_f = r_{mv} = c_{r1}/c_{11}$.

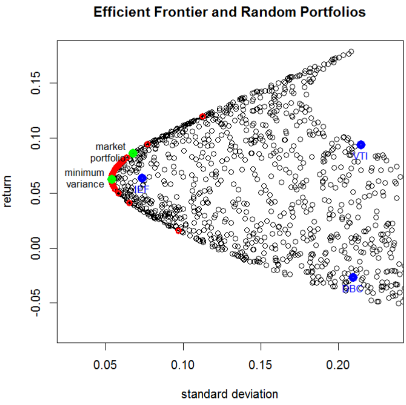
```
> # Plot the efficient frontier
> reteffv <- effront["return", ]
> stdevs <- effront["stdev", ]
> plot(x=stdevs, y=reteffv, t="l", col="blue", lwd=2,
+   xlim=c(0.0, max(stdevs)),
+   main="Efficient Frontier and Tangent Lines",
+   xlab="standard deviation", ylab="return")
```




```
> # Calculate vector of mean returns
> reteffv <- min(reteffv) + diff(range(reteffv))*c(0.2, 0.4, 0.6, 0.
> # Plot the tangent lines
> for (reteffm in reteffv) {
+   # Calculate the maximum Sharpe standard deviation
+   stdev <- sqrt((c11*reteffm^2-2*cr1*reteffm+crr)/detf)
+   # Calculate the slope of the tangent line
+   sharper <- (stdev*detf)/(c11*reteffm-cr1)
+   # Calculate the risk-free rate as intercept of the tangent line
+   riskf <- reteffm - sharper*stdev
+   # Plot the tangent portfolio
+   points(x=stdev, y=reteffm, col="red", lwd=3)
+   # Plot the tangent line
+   abline(a=riskf, b=sharper, lwd=2, col="green")
+ } # end for
```

Random Portfolios

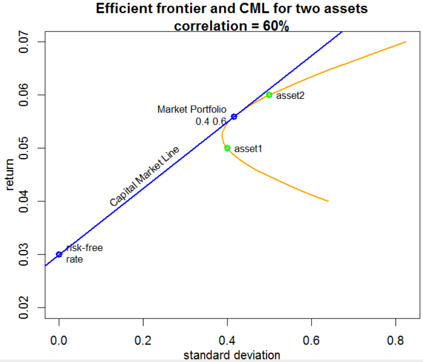
```
> # Calculate random portfolios
> nportf <- 1000
> randportf <- sapply(1:nportf, function(it) {
+   weightv <- runif(nstocks-1, min=-0.25, max=1.0)
+   weightv <- c(weightv, 1-sum(weights))
+   # Portfolio returns and standard deviation
+   c(return=252*sum(weightv*retm),
+     stdev=sqrt(252*drop(weightv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




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

Plotting Efficient Frontier for Two-asset Portfolios

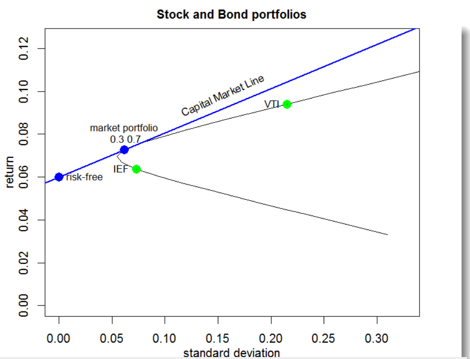
```
> riskf <- 0.03
> retp <- c(asset1=0.05, asset2=0.06)
> stdevs <- c(asset1=0.4, asset2=0.5)
> corrp <- 0.6
> covmat <- matrix(c(1, corrp, corrp, 1), nc=2)
> covmat <- t(t(stdevs*covmat)*stdevs)
> weightv <- seq(from=(-1), to=2, length.out=31)
> weightv <- cbind(weightv, 1-weightv)
> retp <- weightv %*% retp
> portfsd <- sqrt(rowSums(weightv*(weightv %*% covmat)))
> sharper <- (retp-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, retp, 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(retp)))
> # Add efficient portfolio (maximum Sharpe ratio portfolio)
> points(portfsd[whichmax], retp[whichmax],
+  col="blue", lwd=3)
> text(x=portfsd[whichmax], y=retp[whichmax],
+      labels=paste(c("efficient portfolio\n",
+  structure(c(weightv[whichmax], 1-weightv[whichmax]),
+          names=names(retp))), collapse=" "),
+      pos=2, cex=0.8)
```




```
> # Plot individual assets
> points(stdevs, retp, col="green", lwd=3)
> text(stdevs, retp, labels=names(retp), 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")
> rangev <- par("usr")
> text(portfsd[whichmax]/2, (retp[whichmax]+riskf)/2,
+      labels="Capital Market Line", cex=0.8, , pos=3,
+      srt=45*atan(sharpem*(rangev[2]-rangev[1])/
+              (rangev[4]-rangev[3])*
+              heightp/widthp)/(0.25*pi))
```

Efficient Frontier of Stock and Bond Portfolios

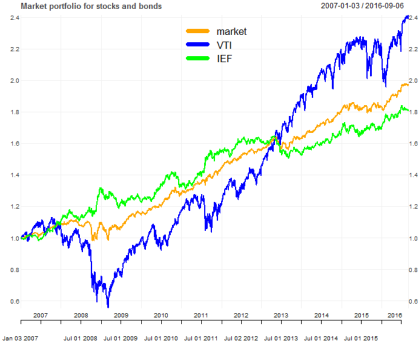
```
> # Vector of symbol names
> symbolv <- c("VTI", "IEF")
> # Matrix of portfolio weights
> weightv <- seq(from=(-1), to=2, length.out=31)
> weightv <- cbind(weightv, 1-weightv)
> # Calculate portfolio returns and volatilities
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> retp <- retp %*% t(weights)
> portfv <- cbind(252*colMeans(retp),
+   sqrt(252)*matrixStats::colSds(retp))
> colnames(portfv) <- c("returns", "stdev")
> riskf <- 0.06
> portfv <- cbind(portfv,
+   (portfv[, "returns"]-riskf)/portfv[, "stdev"])
> colnames(portfv)[3] <- "Sharpe"
> whichmax <- which.max(portfv[, "Sharpe"])
> sharpem <- portfv[whichmax, "Sharpe"]
> plot(x=portfv[, "stdev"], y=portfv[, "returns"],
+      main="Stock and Bond portfolios", t="l",
+      xlim=c(0, 0.7*max(portfv[, "stdev"])), ylim=c(0, max(portfv[
+      xlab="standard deviation", ylab="return")
> # Add blue point for efficient portfolio
> points(x=portfv[whichmax, "stdev"], y=portfv[whichmax, "returns"]
> text(x=portfv[whichmax, "stdev"], y=portfv[whichmax, "returns"],
+      labels=paste(c("efficient portfolio\n",
+   structure(c(weightv[whichmax, 1], weightv[whichmax, 2]), names=
+      pos=3, cex=0.8)
```




```
> # Plot individual assets
> retm <- 252*sapply(retp, mean)
> stdevs <- sqrt(252)*sapply(retp, sd)
> points(stdevs, retm, col="green", lwd=6)
> text(stdevs, retm, labels=names(retp), 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)
> rangev <- par("usr")
> text(max(portfv[, "stdev"])/3, 0.75*max(portfv[, "returns"]),
+      labels="Capital Market Line", cex=0.8, , pos=3,
+      srt=45*atan(sharpem*(rangev[2]-rangev[1])/
+              (rangev[4]-rangev[3])*
+              heightp/widthp)/(0.25*pi))
```

Performance of Efficient Portfolio for Stocks and Bonds

```
> # Calculate cumulative returns of VTI and IEF
> retsoptim <- lapply(retp,
+   function(retp) exp(cumsum(retp)))
> retsoptim <- rutils::do_call(cbind, retsoptim)
> # Calculate the efficient portfolio returns
> retsoptim <- cbind(exp(cumsum(retp %*%
+     c(weightv[whichmax], 1-weightv[whichmax]))),
+   retsoptim)
> colnames(retsoptim)[1] <- "efficient"
> # Plot efficient 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="Efficient 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")
```



Maximum Return Portfolio Using Linear Programming

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

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

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 the objective vector - the mean returns
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> objvec <- colMeans(retp)
> # Specify matrix of linear constraint coefficients
> coeffm <- matrix(c(rep(1, nstocks), 1, 1, 0),
+            nc=nstocks, byrow=TRUE)
> # Specify the logical constraint operators
> logop <- c("==", "<=")
> # Specify the vector of constraints
> consv <- c(1, 0)
> # Specify box constraints (-1, 1) (default is c(0, Inf))
> boxc <- list(lower=list(ind=1:nstocks, val=rep(-1, nstocks)),
+        upper=list(ind=1:nstocks, val=rep(1, nstocks)))
> # Perform optimization
> optiml <- Rglpk::Rglpk_solve_LP(
+   obj=objvec,
+   mat=coeffm,
+   dir=logop,
+   rhs=consv,
+   bounds=boxc,
+   max=TRUE)
> all.equal(optiml$optimum, sum(objvec*optiml$solution))
> optiml$solution
> coeffm %*% optiml$solution
```

Conditional Value at Risk (*CVaR*)

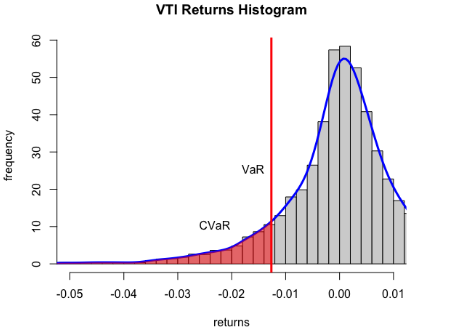
The *Conditional Value at Risk* (*CVaR*) is equal to the average of the *VaR* for confidence levels less than a given confidence level α:

$$\mathrm{CVaR} = \frac{1}{\alpha} \int_0^\alpha \mathrm{VaR}(p)\, \mathrm{d}p$$

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.

```
> # Calculate the VTI percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> confl <- 0.1
> varisk <- quantile(retp, confl)
> cvar <- mean(retp[retp < varisk])
> # Or
> sortv <- sort(as.numeric(retp))
> varind <- round(confl*NROW(retp))
> varisk <- sortv[varind]
> cvar <- mean(sortv[1:varind])
> # Plot histogram of VTI returns
> varmin <- (-0.05)
> histp <- hist(retp, 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(retp, 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 stock 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,

$$w_{min} = \operatorname*{arg\,max}_{w} \Big[\sum_{i=1}^{n} w_i \, b_i \Big]$$

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.

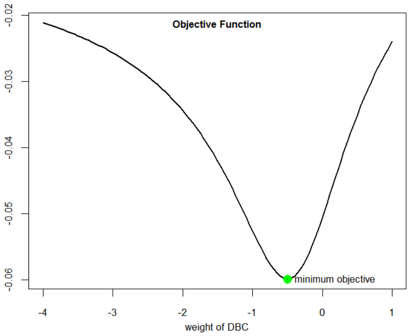
```
> library(rutils)  # Load rutils
> library(Rglpk)
> # Vector of symbol names and returns
> symbolv <- c("VTI", "IEF", "DBC")
> nstocks <- NROW(symbolv)
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> retm <- colMeans(retp)
> confl <- 0.05
> rmin <- 0 ; wmin <- 0 ; wmax <- 1
> weightsum <-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

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

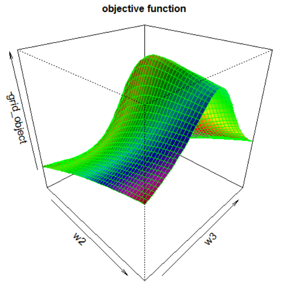
Perspective Plot of Portfolio Objective Function

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.

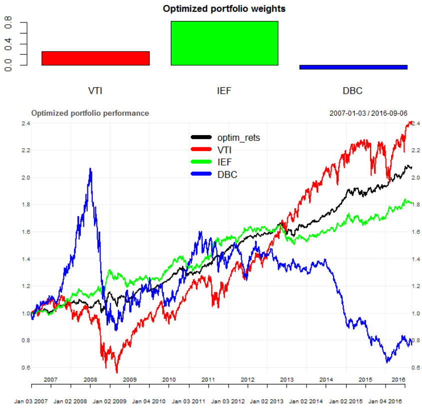
`optim()` returns a list containing the location of the minimum and the objective function value.

```
> # Optimization to find weights with maximum Sharpe ratio
> optiml <- optim(par=weightv,
+              fn=objfun,
+              retp=retp,
+              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$par, col=c("red", "green", "blue"),
+   main="Optimized portfolio weights")
> # Calculate cumulative returns of VTI, IEF, DBC
> retc <- lapply(retp,
+   function(retp) exp(cumsum(retp)))
> retc <- rutils::do_call(cbind, retc)
> # Calculate optimal portfolio returns with VTI, IEF, DBC
> retsoptim <- cbind(
+   exp(cumsum(retp %*% optiml$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=1.0,
+    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(retp %*% optiml$par, retp),
+   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")
> nstocks <- NROW(symbolv)
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> # Calculate the covariance matrix
> covmat <- cov(retp)
> # 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*retp, 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
> symbolv <- c("VTI", "IEF", "DBC")
> nstocks <- NROW(symbolv)
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> # Objective equal to minus Sharpe ratio
> objfun <- function(weightv, retp) {
+   retp <- retp %*% weightv
+   if (sd(retp) == 0)
+     return(0)
+   else
+     -return(mean(retp)/sd(retp))
+ }  # end objfun
> # Perform optimization using DEoptim
> optiml <- DEoptim::DEoptim(fn=objfun,
+   upper=rep(10, NCOL(retp)),
+   lower=rep(-10, NCOL(retp)),
+   retp=retp,
+   control=list(trace=FALSE, itermax=100, parallelType=1))
> weightv <- optiml$optim$bestmem/sum(abs(optiml$optim$bestmem))
> names(weights) <- colnames(retp)
```

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

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:

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

The portfolio weights are proportional to the *Kelly ratios* - the excess returns divided by the variances:

$$w_i \propto \frac{\mu_i}{\sigma_i^2}$$

Homework Assignment

No homework!