FRE7241 Algorithmic Portfolio Management

Lecture#6, Spring 2025

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

April 29, 2025

NYU | TANDON SCHOOL OF ENGINEERING

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*:

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

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 ETF 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 **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
> retarg <- 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*retarg))
> # Calculate the efficient portfolio weights
> weightv <- 0.5*drop(covinv %*% cbind(unitv, retm) %*% lagm)
> # Calculate constraints
> all.equal(1, sum(weightv))
> all.equal(retarg, 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 $\sigma^2_{mv} = 1/c_{11} = 1/\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$.

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 $\mathbb{F}$.

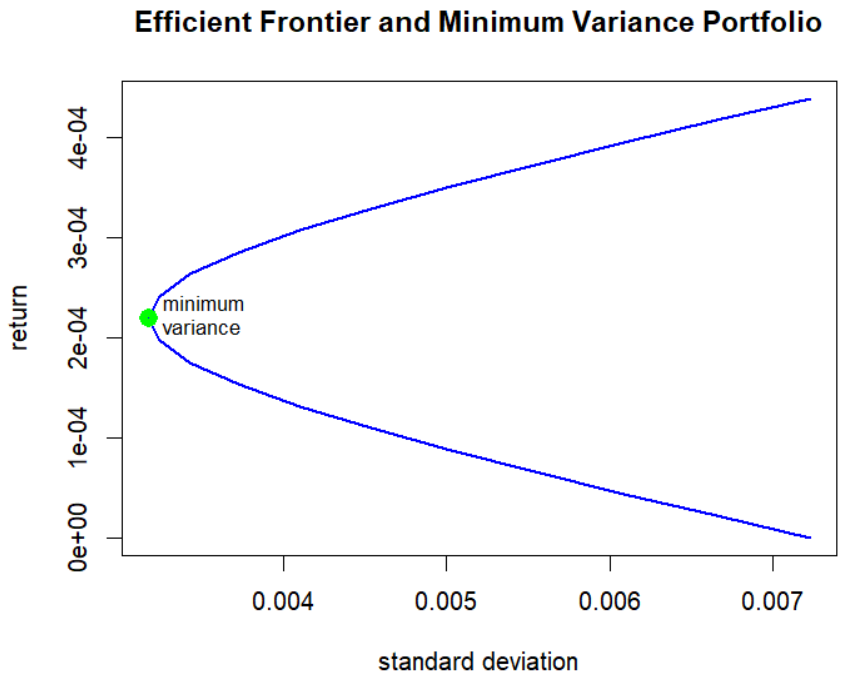
```
> # Calculate the efficient portfolio returns
> reteff <- drop(retp %*% weightv)
> reteffm <- mean(reteff)
> all.equal(reteffm, retarg)
> # Calculate the efficient portfolio variance in three ways
> uu <- c(1, retarg)
> finv <- solve(fmat)
> detf <- (c11*crr-cr1^2)  # det(fmat)
> all.equal(var(reteff),
+   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
> retargv <- retmvm*(1+seq(from=(-1), to=1, by=0.1))
> stdevs <- sapply(retargv, function(rett) {
+   uu <- c(1, rett)
+   sqrt(drop(t(uu) %*% finv %*% uu))
+ })  # end sapply
```




```
> # Plot the efficient frontier
> plot(x=stdevs, y=retargv, 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, retarg)
> 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*retarg-cr1)
> # Calculate the risk-free rate as intercept of the tangent line
> raterf <- retarg - sharper*stdeveff
> # Calculate the risk-free rate from target return
> all.equal(raterf,
+   (retarg*cr1-crr)/(retarg*c11-cr1))
```

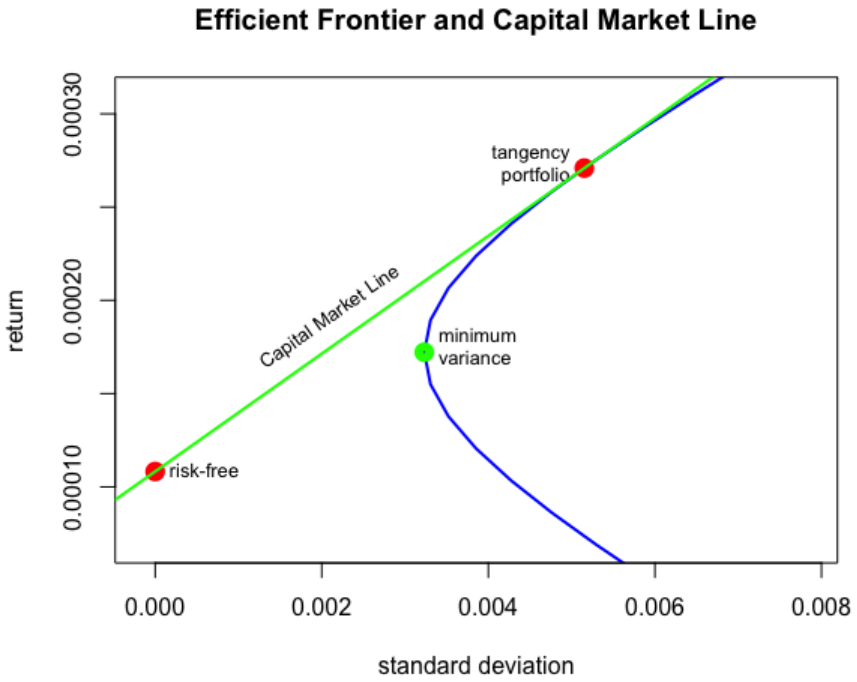
The *Capital Market Line*

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
> aspectr <- 1.0*max(stdevs)/diff(range(retargv)) # Aspect ratio
> plot(x=stdevs, y=retargv, t="l", col="blue", lwd=2, asp=aspectr,
+      xlim=c(0.4, 0.6)*max(stdevs), ylim=c(0.2, 0.9)*max(retargv),
+      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=retarg, col="red", lwd=6)
> text(x=stdeveff, y=retarg, labels="tangency\nportfolio", pos=2, ce
> # Plot the risk-free point
> points(x=0, y=raterf, col="red", lwd=6)
> text(x=0, y=raterf, labels="risk-free", pos=4, cex=0.8)
> # Plot the tangent line
> abline(a=raterf, b=sharper, lwd=2, col="green")
> text(x=0.6*stdev, y=0.8*retarg,
+      labels="Capital Market Line", pos=2, cex=0.8,
+      srt=180/pi*atan(aspectr*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).

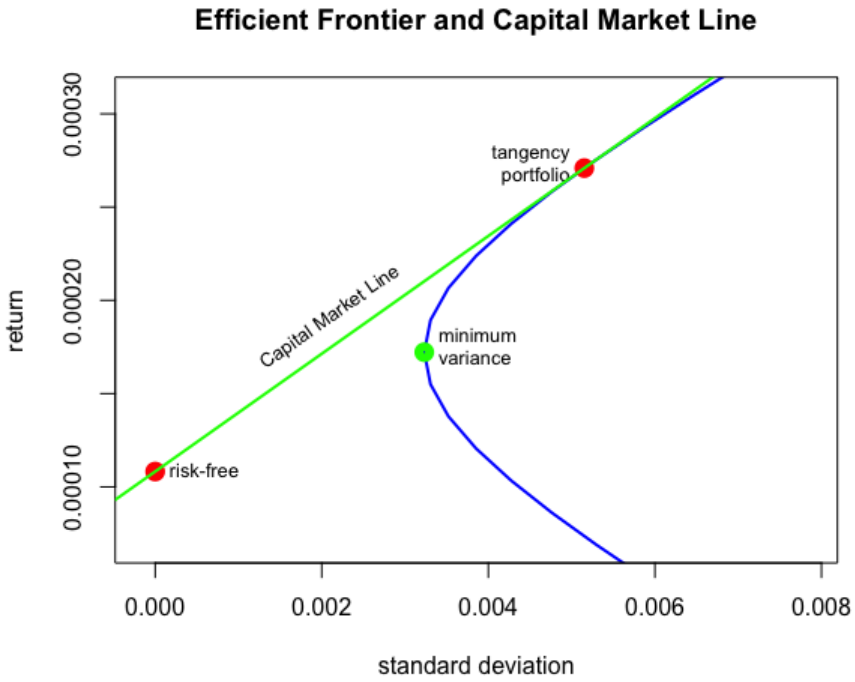
The *Capital Market Line* represents delevered and levered portfolios, consisting of the *tangency portfolio* combined with the *risk-free* asset (bond).

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:

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

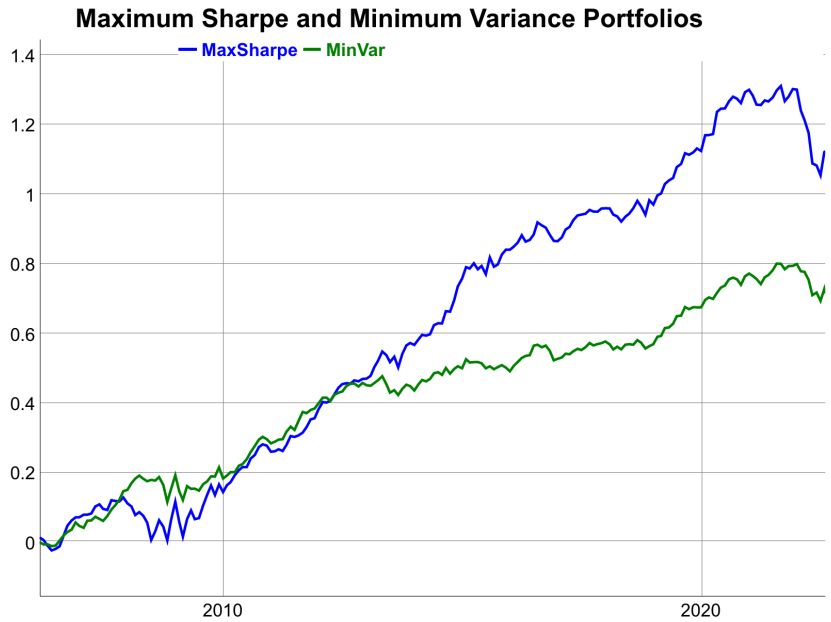
```
> # Calculate the mean excess returns
> raterf <- retarg - sharper*stdeveff
> retx <- (retm - raterf)
> # Calculate the efficient portfolio weights
> weightv <- 0.5*drop(covinv %*% cbind(unitv, retm) %*% lagm)
> # Calculate the maximum Sharpe weights
> weightms <- drop(covinv %*% retx)/sum(covinv %*% retx)
> all.equal(weightv, weightms)
> # Calculate the maximum Sharpe mean return in two ways
> all.equal(sum(retm*weightv), (cr1*raterf-crr)/(c11*raterf-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(weightv) %*% covmat %*% weightv,
+   (t(retx) %*% covinv %*% retx)/sum(covinv %*% retx)^2,
+   (c11*retarg^2-2*cr1*retarg+crr)/detf)
> # Calculate the maximum Sharpe ratio
> sqrt(252)*sum(weightv*retx)/
+   sqrt(drop(t(weightv) %*% covmat %*% weightv))
> # Calculate the stock Sharpe ratios
> sqrt(252)*sapply((retp - raterf), 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)
+   (mean(x)-raterf)/c(Sharpe=sd(x), Sortino=sd(x[x<0])))
> # Plot the log wealth
> endd <- rutils::calc_endpoints(wealthv, 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)
```



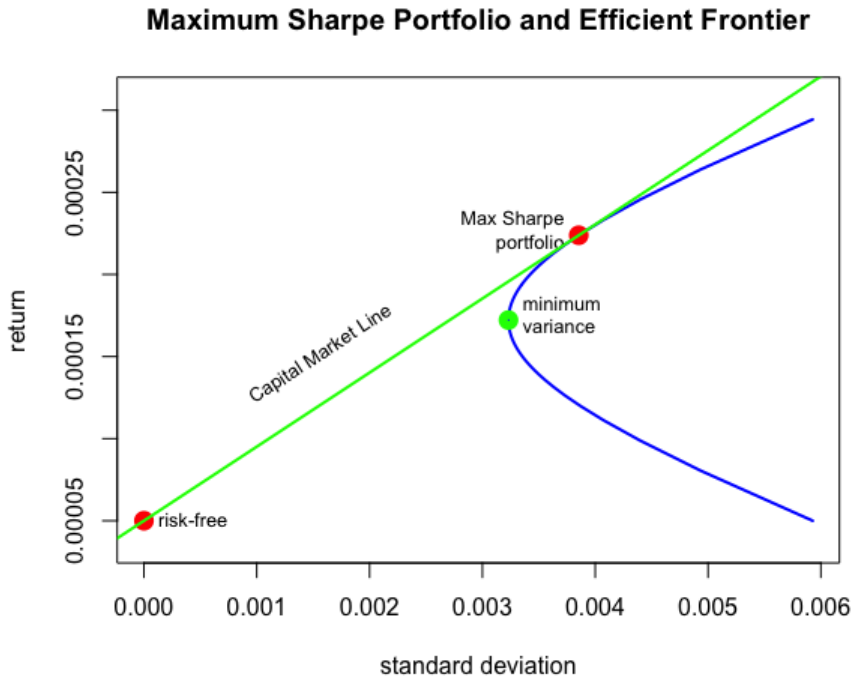
The *Maximum Sharpe* Portfolios and the *Efficient Frontier*

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)
> raterfv <- retmvm*seq(from=1.3, to=20, by=0.1)
> raterfv <- c(raterfv, retmvm*seq(from=(-20), to=0.7, by=0.1))
> effront <- sapply(raterfv, function(raterf) {
+   # Calculate the maximum Sharpe mean return
+   reteffm <- (cr1*raterf-crr)/(c11*raterf-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", ]
> aspectr <- 0.6*max(stdevs)/diff(range(reteffv)) # Aspect ratio
> plot(x=stdevs, y=reteffv, t="l", col="blue", lwd=2, asp=aspectr,
+   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, cex
```




```
> # Calculate the maximum Sharpe return and standard deviation
> raterf <- min(reteffv)
> retmax <- (cr1*raterf-crr)/(c11*raterf-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=raterf, col="red", lwd=6)
> text(x=0, y=raterf, labels="risk-free", pos=4, cex=0.8)
> # Plot the tangent line
> sharper <- (stdevmax*detf)/(c11*retmax-cr1)
> abline(a=raterf, 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(aspectr*sharper))
```

Efficient Portfolios and Their Tangent Lines

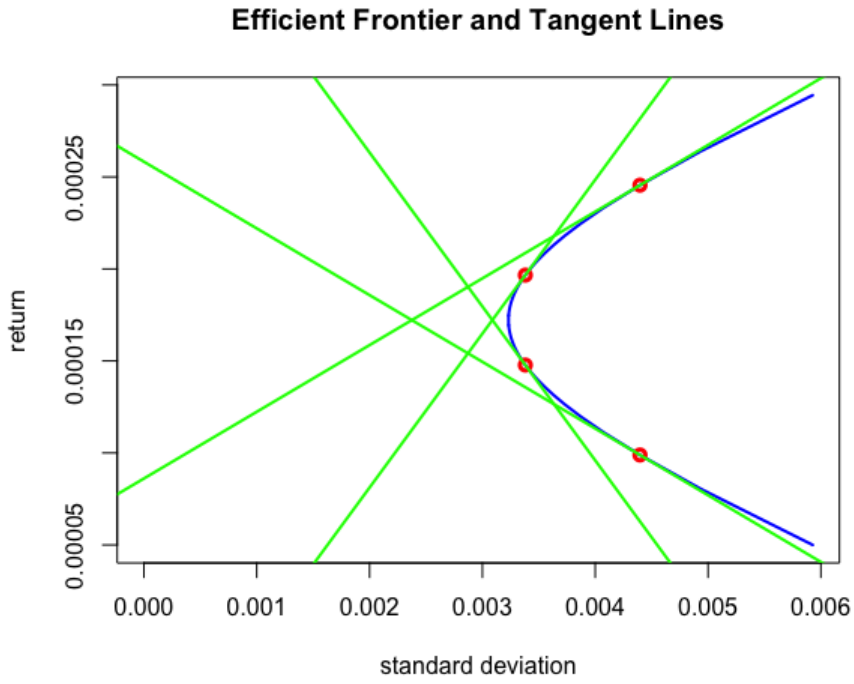
The *efficient frontier* consists of all the *maximum Sharpe* portfolios corresponding to different values of the risk-free rate.

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

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

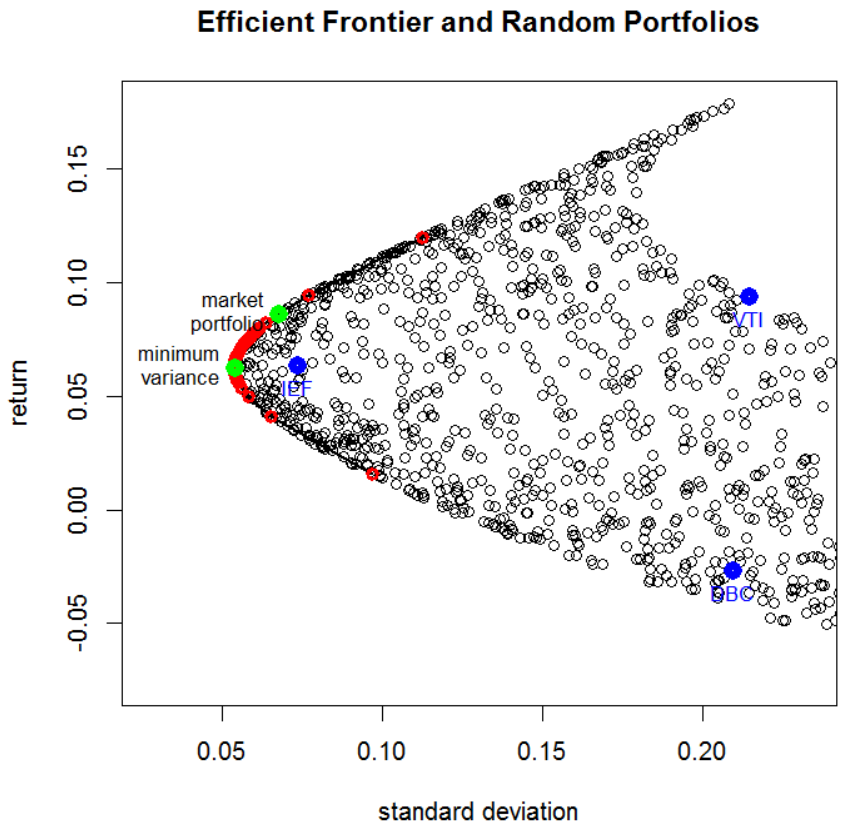



```
> # Plot the efficient frontier
> reteffv <- effront["return", ]
> stdevs <- effront["stdev", ]
> plot(x=stdevs, y=reteffv, t="l", col="blue", lwd=2,
+   xlim=c(0.0, max(stdevs)),
+   main="Efficient Frontier and Tangent Lines",
+   xlab="standard deviation", ylab="return")
```

```
> # 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
+   raterf <- reteffm - sharper*stdev
+   # Plot the tangent portfolio
+   points(x=stdev, y=reteffm, col="red", lwd=3)
+   # Plot the tangent line
+   abline(a=raterf, 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(weightv))
+   # 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)
```

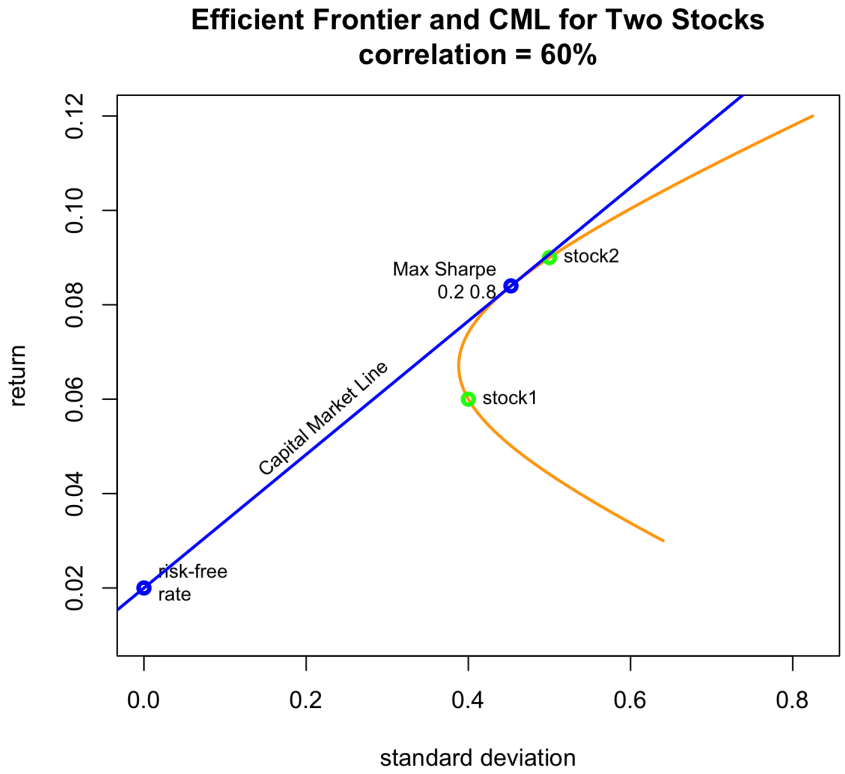



```
> # Plot individual assets
> points(x=sqrt(252*diag(covmat)),
+  y=252*retm, col="blue", lwd=6)
> text(x=sqrt(252*diag(covmat)), y=252*retm,
+      labels=names(retm),
+      col="blue", pos=1, cex=0.8)
```

Efficient Frontier for Two-stock Portfolio

The *efficient frontier* for a two-stock portfolio illustrates its dependence on the correlation and the risk-free rate.

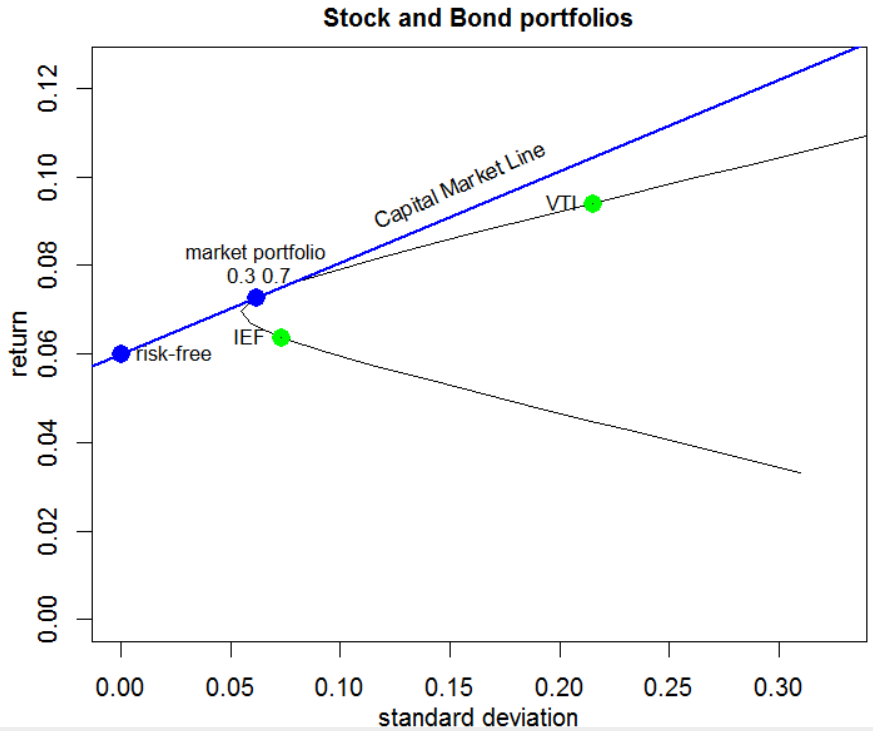
```
> # Define the parameters
> raterf <- 0.02 # Risk-free rate
> retp <- c(stock1=0.06, stock2=0.09) # Returns
> stdevs <- c(stock1=0.4, stock2=0.5) # Standard deviations
> corrp <- 0.6 # Correlation
> covmat <- matrix(c(1, corrp, corrp, 1), nc=2) # Covariance matrix
> covmat <- t(t(stdevs*covmat)*stdevs)
> weightv <- seq(from=(-1), to=2, length.out=71) # Weights
> weightv <- cbind(weightv, 1-weightv)
> retport <- weightv %*% retp # Portfolio returns
> portfsd <- sqrt(rowSums(weightv*(weightv %*% covmat))) # Portfolio
> sharper <- (retport-raterf)/portfsd # Portfolio Sharpe ratios
> # Plot the efficient frontier
> # x11(widthp <- 6, heightp <- 5) # Windows
> dev.new(widthp <- 6, heightp <- 5, noRStudioGD=TRUE) # Mac
> plot(portfsd, retport, t="l",
+  main=paste0("Efficient Frontier and CML for Two Stocks\ncorrelat
+  xlab="standard deviation", ylab="return",
+  lwd=2, col="orange", xlim=c(0, max(portfsd)), ylim=c(0.01, max(re
> # Add the maximum Sharpe portfolio
> whichmax <- which.max(sharper)
> sharpem <- max(sharper) # Maximum Sharpe ratio
> retmax <- retport[whichmax]
> sdeff <- portfsd[whichmax]
> weightm <- round(weightv[whichmax], 2)
> points(sdeff, retmax, col="blue", lwd=3)
> text(x=sdeff, y=retmax, labels=paste(c("Max Sharpe\n",
+   structure(c(weightm, (1-weightm)), names=c("stock1", "stock2")
+   pos=2, cex=0.8)
```




```
> # Plot individual stocks
> 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=raterf, col="blue", lwd=3)
> text(0, raterf, labels="risk-free\nrate", pos=4, cex=0.8)
> abline(a=raterf, b=sharpem, lwd=2, col="blue")
> rangev <- par("usr")
> text(sdeff/2, (retmax+raterf)/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

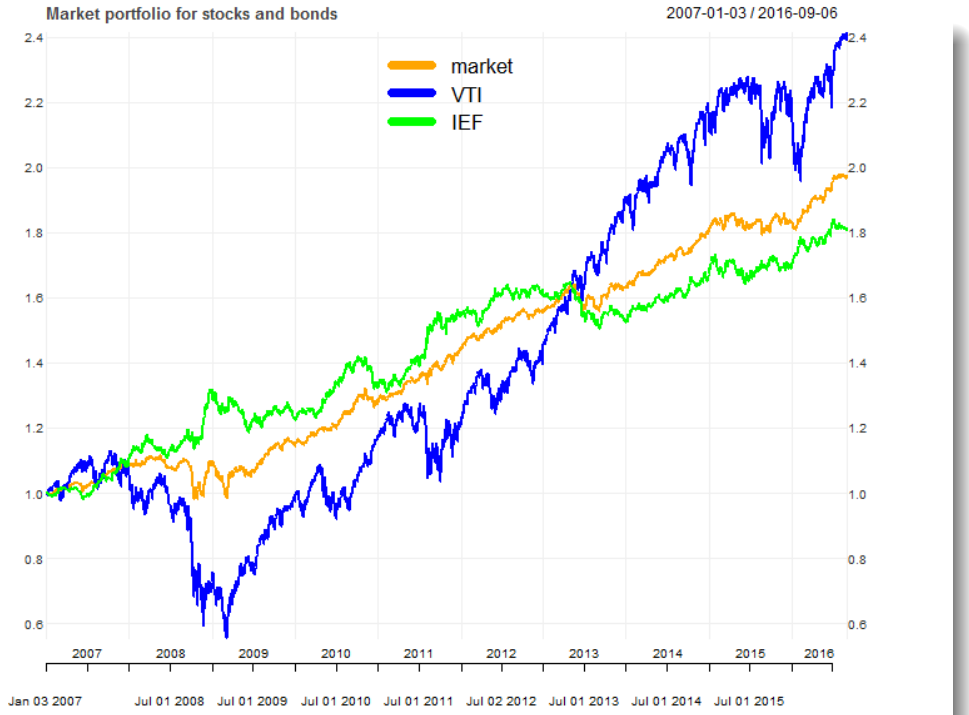
```
> # Vector of symbol names
> symbolv <- c("VTI", "IEF")
> # Matrix of portfolio weights
> weightv <- seq(from=(-1), to=2, length.out=31)
> weightv <- cbind(weightv, 1-weightv)
> # Calculate portfolio returns and volatilities
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> retport <- retp %*% t(weightv)
> portfv <- cbind(252*colMeans(retport),
+   sqrt(252)*matrixStats::colSds(retport))
> colnames(portfv) <- c("returns", "stdev")
> raterf <- 0.06
> portfv <- cbind(portfv,
+   (portfv[, "returns"]-raterf)/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 stocks
> retm <- 252*sapply(retport, mean)
> stdevs <- sqrt(252)*sapply(retport, sd)
> points(stdevs, retm, col="green", lwd=6)
> text(stdevs, retm, labels=names(retport), pos=2, cex=0.8)
> # Add point at risk-free rate and draw Capital Market Line
> points(x=0, y=raterf, col="blue", lwd=6)
> text(0, raterf, labels="risk-free", pos=4, cex=0.8)
> abline(a=raterf, 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

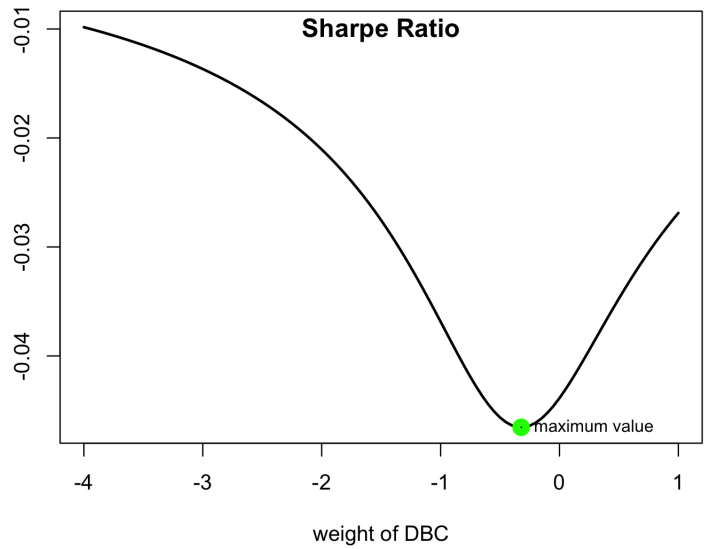
```
> # Calculate cumulative returns of VTI and IEF
> retsoptim <- lapply(retp, function(retp) exp(cumsum(retp)))
> retsoptim <- rutils::do_call(cbind, retsoptim)
> # Calculate the efficient portfolio returns
> retsoptim <- cbind(exp(cumsum(retp %*%
+     c(weightv[whichmax], 1-weightv[whichmax]))),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



Sharpe Ratio Objective Function

The function `optimize()` performs *one-dimensional* optimization over a single independent variable, in the specified interval.

```
> # Calculate daily ETF percentage returns
> symbolv <- c("VTI", "IEF", "DBC")
> nstocks <- NROW(symbolv)
> retp <- na.omit(rutils::etfenv$returns[, symbolv])
> nrows <- NROW(retp)
> # Create initial vector of portfolio weights
> weightv <- rep(1, NROW(symbolv))
> names(weightv) <- symbolv
> # Objective equal to minus Sharpe ratio
> objfun <- function(weightv, retp) {
+   retportf <- retp %*% weightv
+   stdev <- sd(retportf)
+   if (stdev == 0)
+     return(0)
+   else
+     return(-mean(retportf)/stdev)
+ }  # end objfun
> # Objective for equal weight portfolio
> objfun(weightv, retp=retp)
> optiml <- unlist(optimize(f=function(weightv)
+     objfun(c(1, 1, weightv), retp=retp),
+   interval=c(-10, 10)))
> # Vectorize objective function with respect to third weight
> objvec <- function(weightv) sapply(weightv,
+   function(weightv) objfun(c(1, 1, weightv), retp=retp))
> # Or
> objvec <- Vectorize(FUN=function(weightv)
+     objfun(c(1, 1, weightv), retp=retp),
+   vectorize.args="weightv")  # end Vectorize
> objvec(1)
> objvec(1:3)
```



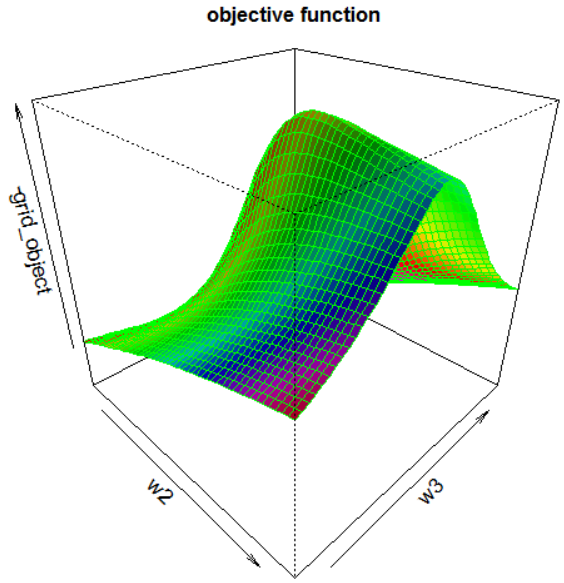

```
> # Plot objective function with respect to third weight
> curve(expr=objvec, type="l", xlim=c(-4.0, 1.0),
+ xlab=paste("weight of", names(weightv[3])),
+ ylab="", lwd=2)
> title(main="Sharpe Ratio", line=(-1))  # Add title
> points(x=optiml[1], y=optiml[2], col="green", lwd=6)
> text(x=optiml[1],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

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 two weights
> objvec <- Vectorize(
+   FUN=function(w1, w2, w3) objfun(c(w1, w2, w3), retp),
+   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)
> gridm <- outer(w2, w3, FUN=objvec, w1=1)
> rownames(gridm) <- round(w2, 2)
> colnames(gridm) <- round(w3, 2)
> # Perspective plot of objective function
> persp(w2, w3, -gridm,
+ 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=-gridm, 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)
> # Render the 3d surface plot of function
> rgl::rglwidget(elementId="plot3drgl", width=1000, height=1000)
```

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.

The parameter `control=list(factr=1e5)` determines the precision of the `L-BFGS-B` method. A smaller `factr` value produces results with a higher precision, but it requires more iterations and a longer optimization. A larger `factr` value achieves a faster solution but less precise results.

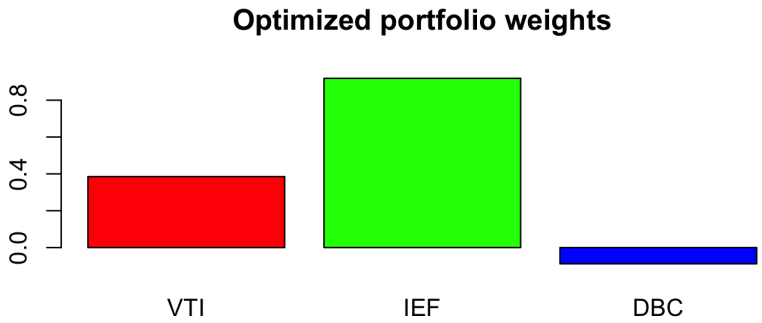
`optim()` returns a list containing the location of the minimum and the objective function value.

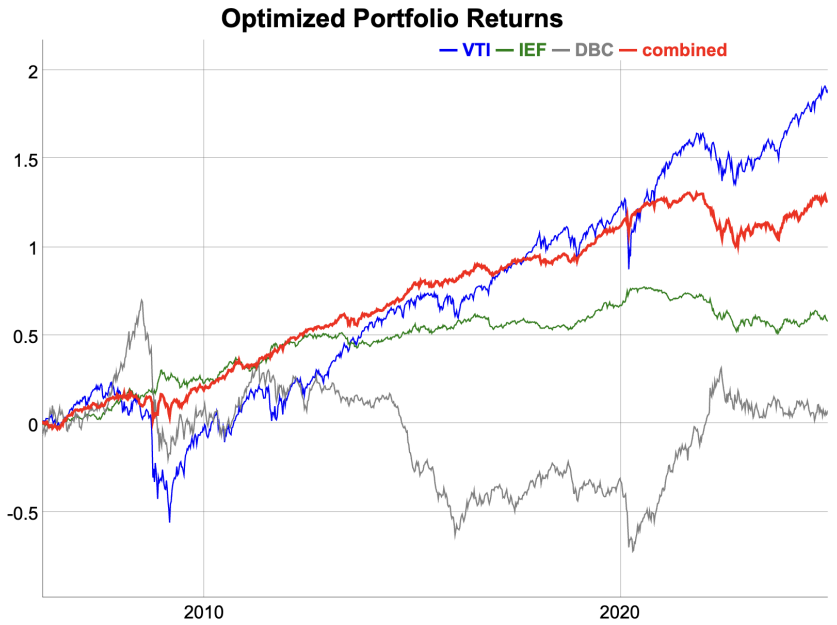
```
> # Create initial vector of portfolio weights
> weightv <- rep(1, NROW(symbolv))
> names(weightv) <- symbolv
> # Optimization to find weights with maximum Sharpe ratio
> optiml <- optim(par=weightv,
+           fn=objfun,
+           retp=retp,
+           method="L-BFGS-B",
+           control=list(factr=1e5),
+           upper=c(10, 10, 10),
+           lower=c(-10, -10, -10))
> # Optimal parameters
> weightv <- optiml$par
> weightv <- weightv/sqrt(sum(weightv^2))
> weightv
> # Optimal Sharpe ratio
> -objfun(weightv, retp)
> # Calculate the weights from the inverse covariance matrix
> weightv <- drop(solve(cov(retp)) %*% sapply(retp, mean))
> weightv <- weightv/sqrt(sum(weightv^2))
> weightv
```

Optimized Portfolio Returns

The optimized portfolio has both long and short positions, and has a larger *Sharpe* ratio than the individual assets.

```
> # barplot of optimal portfolio weights
> barplot(weightv, col=c("red", "green", "blue"),
+   main="Optimized portfolio weights")
> # Calculate the cumulative wealth of the optimized portfolio
> wealthv <- cbind(retp %*% weightv, retp)
> colnames(wealthv)[1] <- "combined"
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   (mean(x)-raterf)/c(Sharpe=sd(x), Sortino=sd(x[x<0])))
> # Plot the log wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> colv <- colnames(wealthv)
> colr <- c("red", "blue", "green", "grey")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Optimized Portfolio Returns") %>%
+   dyOptions(colors=colr, strokeWidth=1) %>%
+   dySeries(name=colv[1], col="red", strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```





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(vecv, param=25){
+   sum(vecv^2 - param*cos(vecv))
+ }  # end rastrigin
> vecv <- c(pi/6, pi/6)
> rastrigin(vecv=vecv)
> library(DEoptim)
> # Optimize rastrigin using DEoptim
> optiml <- DEoptim::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.

With the parameter `control=list(parallelType=1)`, `DEoptim::DEoptim()` performs the optimization using parallel computing.

With the parameter `control=list(itermax=100)`, `DEoptim::DEoptim()` performs at most 100 iterations.

To achieve a good precision, `itermax` should be set to a much larger value, such as 10000. 100 iterations.

```
> # 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
> names(weightv) <- colnames(retp)
> weightv <- weightv/sqrt(sum(weightv^2))
> weightv
```

Portfolio Optimization With *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 penalty term in *ridge* regularization is the sum of squares of the weights: $\sum_{i=1}^{n} w_i^2$. The *ridge* penalty shrinks (reduces) the weights together, without shrinking some of them to zero.

The penalty term in *Lasso* regularization is the sum of the absolute values of the weights: $\sum_{i=1}^{n} |w_i|$. The *Lasso* penalty shrinks the less important weights to zero, similar to dimension reduction.

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 λ parameter controls the amount of shrinkage, and α controls the balance between the *ridge* and *Lasso* penalties.

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

Optimal Stock Portfolio Weights

The portfolio weights obtained from inverting the covariance matrix are the same as the weights obtained from portfolio optimization.

```
> # Load stock returns
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_returns.RDat
> datev <- zoo::index(na.omit(retstock$GOOGL))
> retp <- retstock[datev] # Subset the returns to GOOGL
> # Remove the stocks with any NA values
> numna <- sapply(retp, function(x) sum(is.na(x)))
> retp <- retp[, numna==0]
> # Select 100 random stocks
> retp <- retp[, sample(NCOL(retp), 100)]
> symbolv <- colnames(retp)
> datev <- zoo::index(retp)
> retis <- retp["/2014"] # In-sample returns
> raterf <- 0.03/252
> retx <- (retis - raterf) # Excess returns
> # Calculate the maximum Sharpe weights in-sample interval
> colmeanv <- colMeans(retx, na.rm=TRUE)
> covmat <- cov(retx, use="pairwise.complete.obs")
> invreg <- MASS::ginv(covmat)
> wmaxs <- drop(invreg %*% colmeanv)
> names(wmaxs) <- symbolv
```

```
> # Calculate the weights using optimization without shrinkage
> lambdaf <- 0.0 ; alphaf <- 1.0
> optiml <- optim(par=wmaxs,
+           fn=objfun,
+           retp=retx,
+           lambdaf=lambdaf,
+           alphaf=alphaf,
+           method="L-BFGS-B",
+           control=list(factr=1e5),
+           upper=c(10, 10, 10),
+           lower=c(-10, -10, -10))
> weighto <- optiml$par
> names(weighto) <- symbolv
> all.equal(weighto, wmaxs)
```

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 the 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:

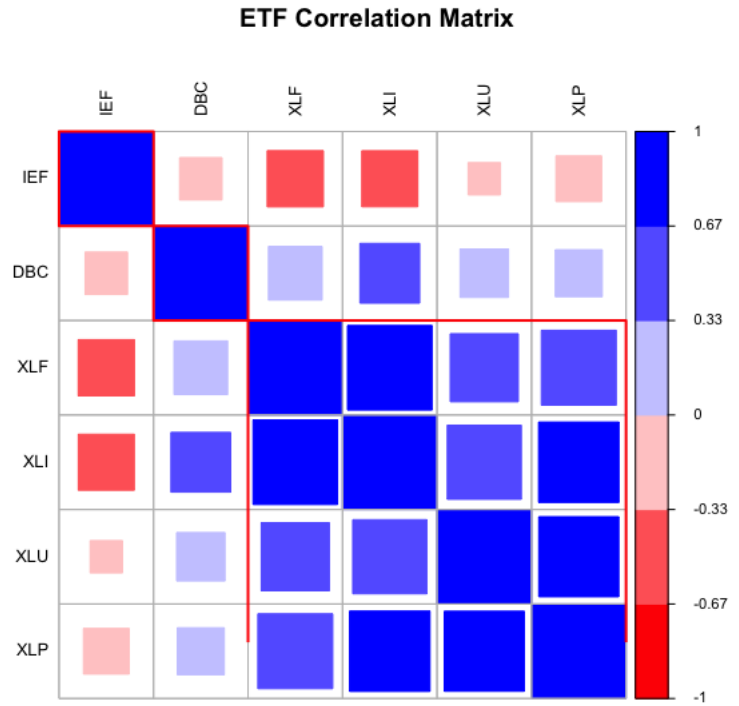
$$w_i \propto \frac{\mu_i}{\sigma_i^2}$$

Covariance Matrix of ETF Returns

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

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

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




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

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

Principal Component Vectors

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

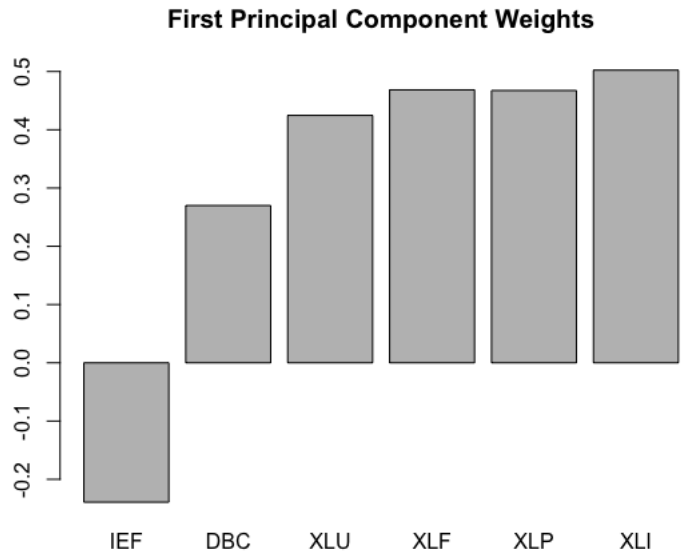
$$\mathbf{pc}_j = \sum_{i=1}^{k} w_{ij}\, \mathbf{r}_i$$

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

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

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

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




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

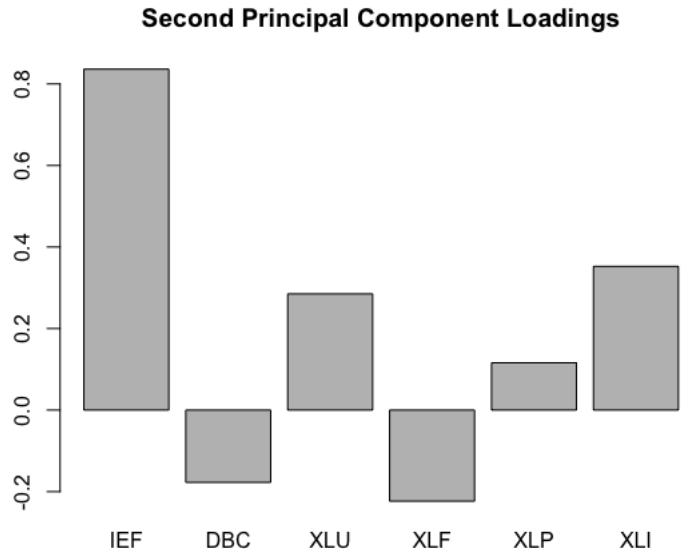
```
> # Find weights with maximum variance
> optim1 <- optim(par=weightv,
+   fn=objfun,
+   retp=retp,
+   method="L-BFGS-B",
+   upper=rep(10.0, nweights),
+   lower=rep(-10.0, nweights))
> # Optimal weights and maximum variance
> weights1 <- optim1$par
> -objfun(weights1, retp)
> # Plot first principal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

Higher Order Principal Components

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

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

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




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

Eigenvalues of the Correlation Matrix

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

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

Where λ is a *Lagrange multiplier*.

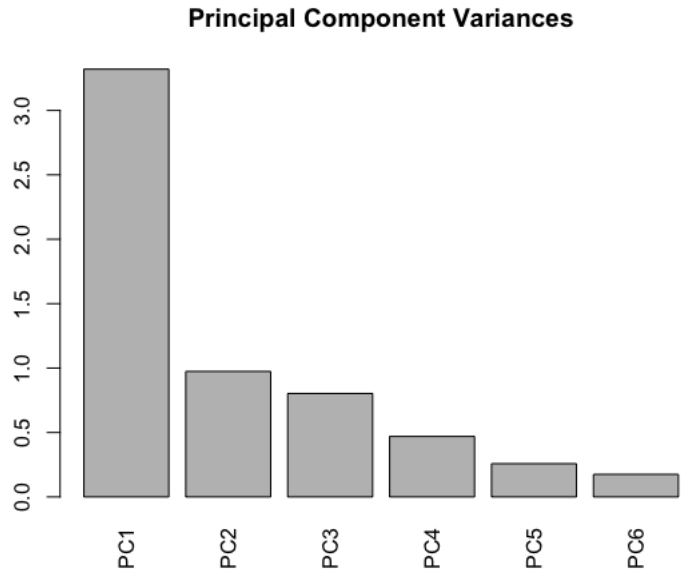
The maximum variance portfolio weights can be found by differentiating $\mathcal{L}$ with respect to $\mathbf{w}$ and setting it to zero:

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

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

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

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




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

Principal Component Analysis Versus *Eigen Decomposition*

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

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

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

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

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

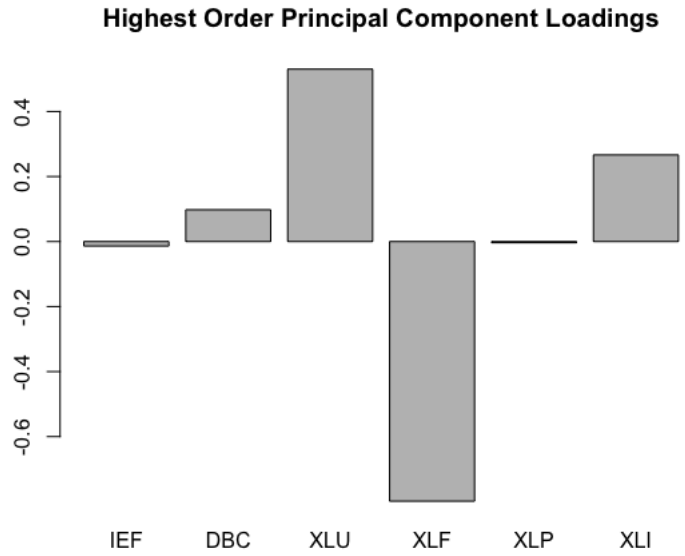
```
> # Calculate the eigen decomposition of the correlation matrix
> eigend <- eigen(cormat)
> # Perform PCA with scaling
> pcad <- prcomp(retp, scale=TRUE)
> # Compare outputs
> all.equal(eigend$values, pcad$sdev^2)
> all.equal(abs(eigend$vectors), abs(pcad$rotation),
+     check.attributes=FALSE)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

Minimum Variance Portfolio

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

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

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




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

Principal Component Analysis of ETF Returns

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

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

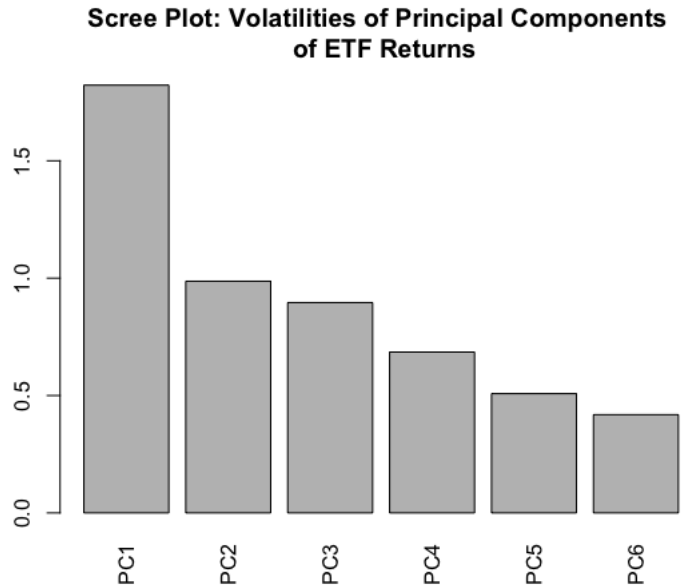
If the input time series are not scaled, then *PCA* analysis is equivalent to the *eigen decomposition* of the covariance matrix, and if they are scaled, then *PCA* analysis is equivalent to the *eigen decomposition* of the correlation matrix.

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

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

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

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



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

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

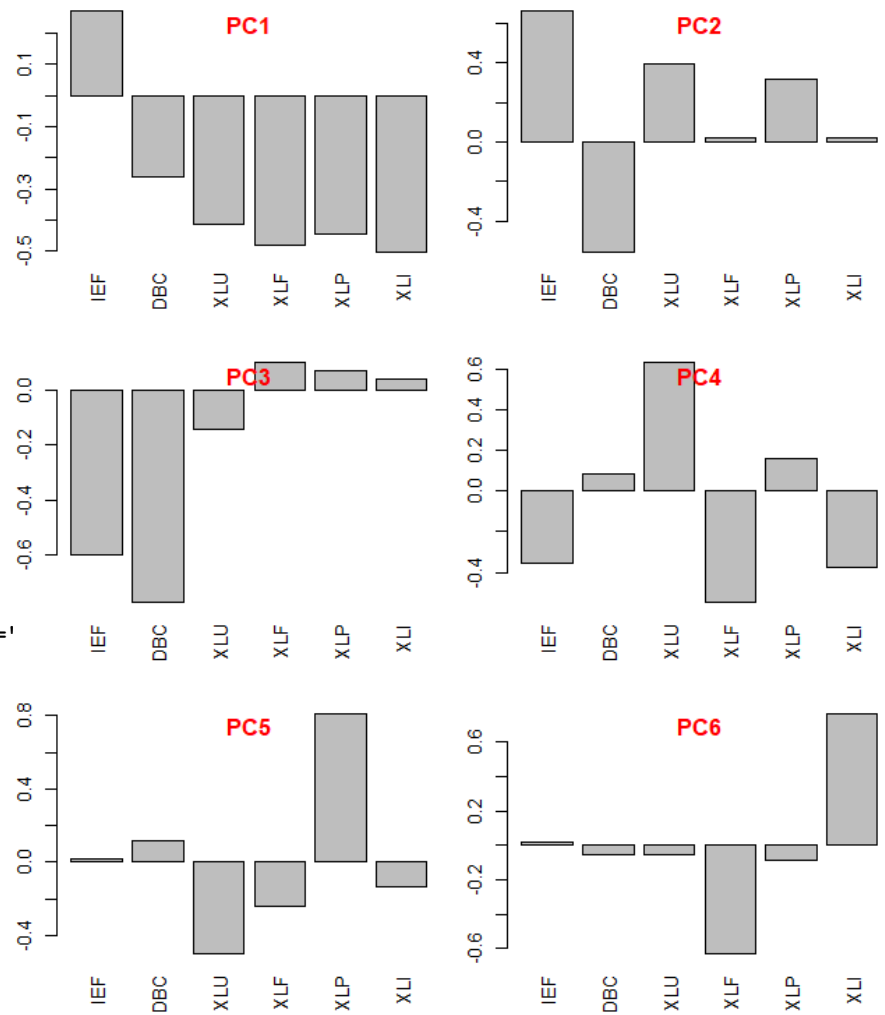
Principal Component Loadings (Weights)

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

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

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

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



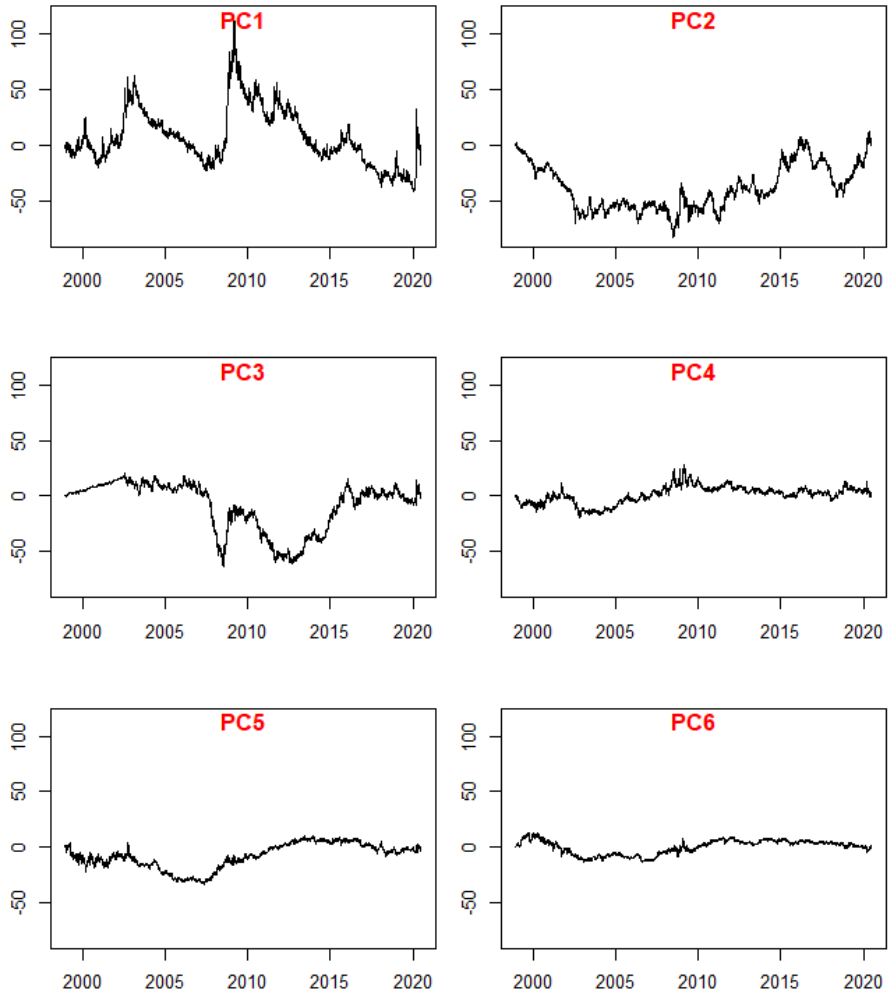
Principal Component Time Series

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

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

Higher order *principal components* are gradually less volatile.

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



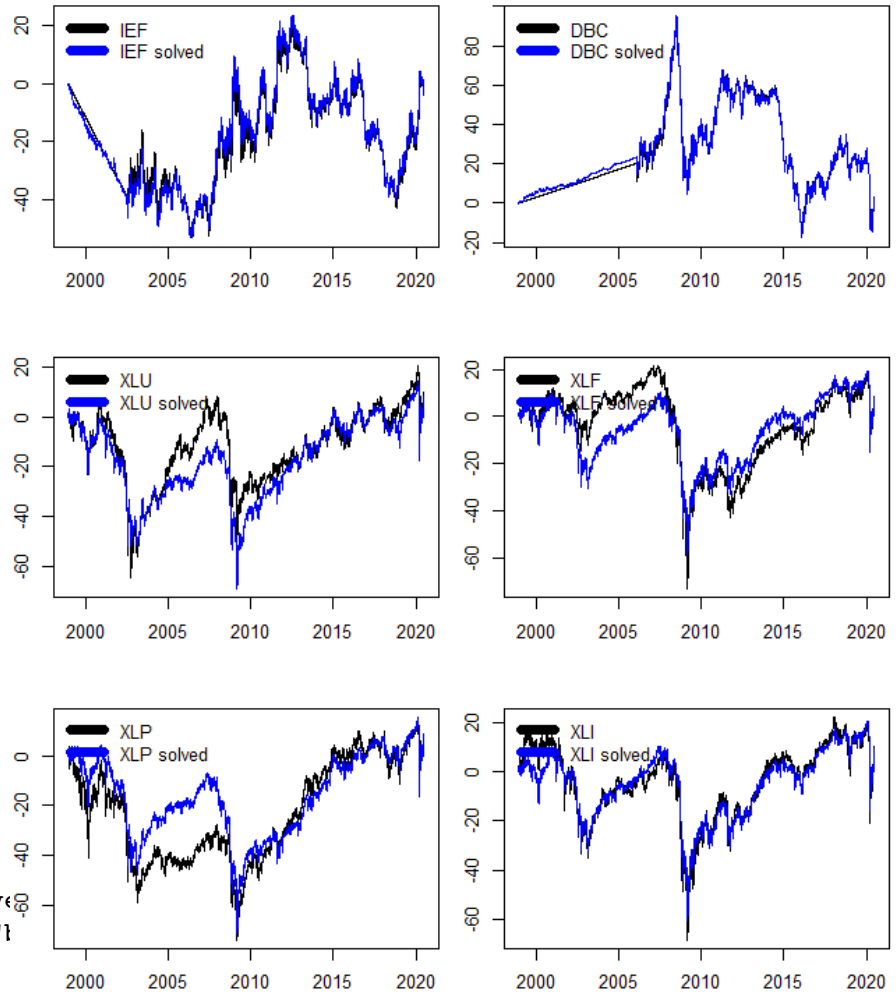
Dimension Reduction Using Principal Component Analysis

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

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

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

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



Condition Number of Correlation Matrices

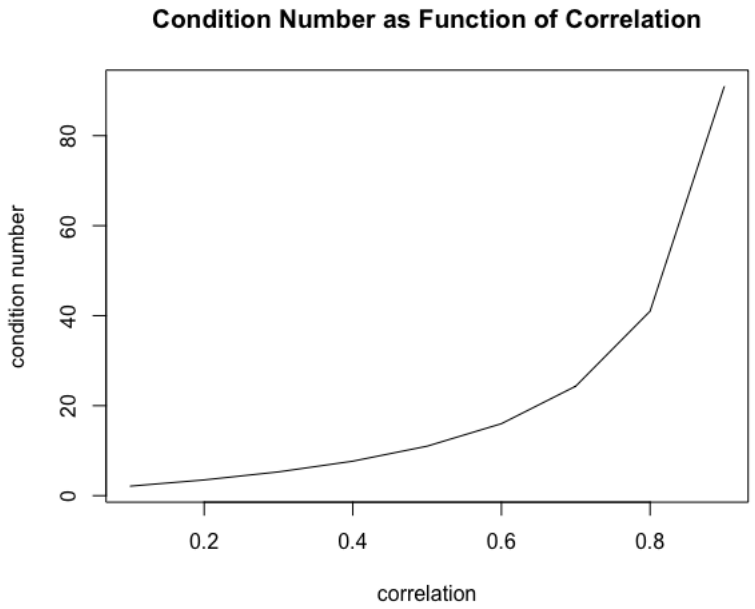
The condition number κ of a correlation matrix is equal to the ratio of its largest eigenvalue divided by the smallest:

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

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

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

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




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

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

Condition Number for Small Number of Observations

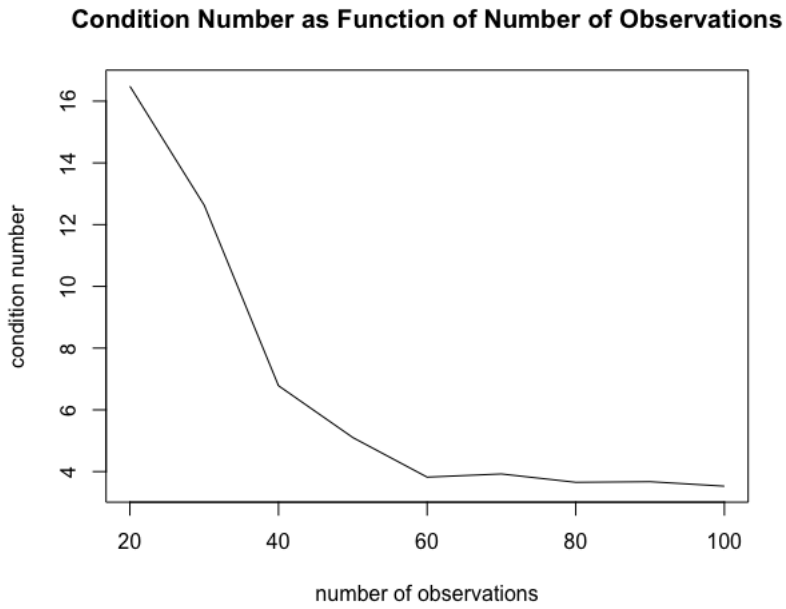
The condition number also depends on the number of observations.

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

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

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

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




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

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

The Correlations of Stock Returns

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

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

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

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

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

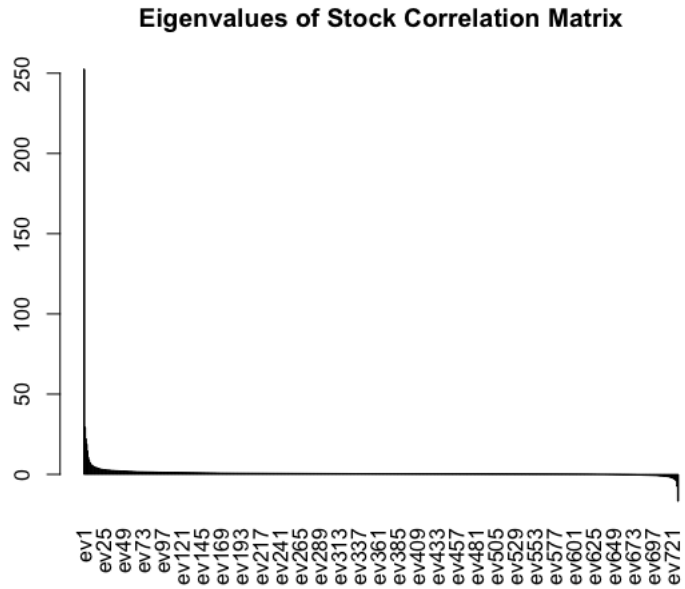
Principal Component Analysis of Stock Returns

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

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

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

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




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

Principal Component Analysis of Low and High Volatility Stocks

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

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

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

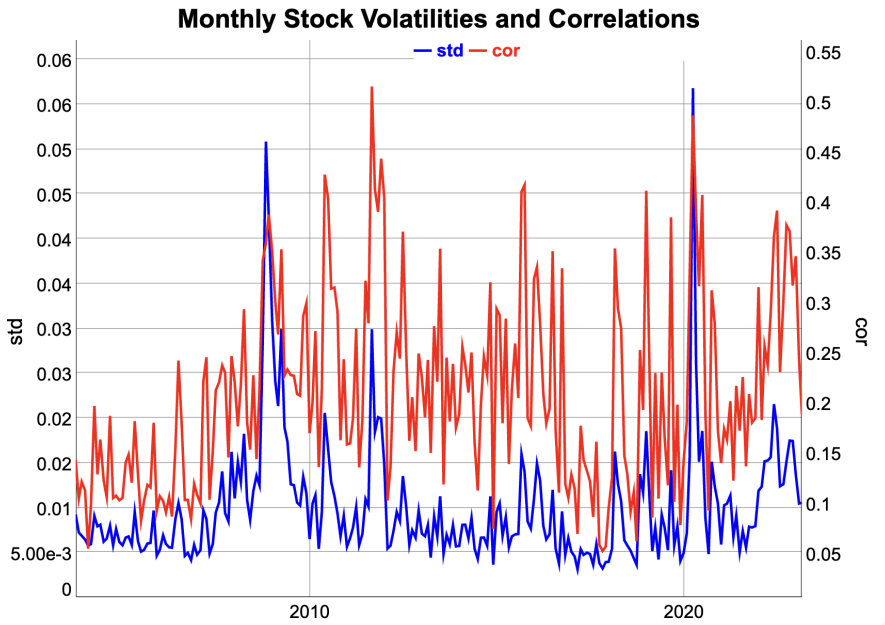
```
> # Calculate the stock variance
> varv <- sapply(retp, var, na.rm=TRUE)
> # Calculate the returns of low and high volatility stocks
> nstocks <- NCOL(retp)
> medianv <- median(varv)
> retlow <- retp[, varv <= medianv]
> rethigh <- retp[, varv > medianv]
> # Calculate the correlations of low volatility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

Stock Correlations in Periods of Low and High Volatility

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

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

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




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

Principal Component Analysis in Periods of Low and High Volatility

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

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

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

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

Trailing Correlations of Stock Returns

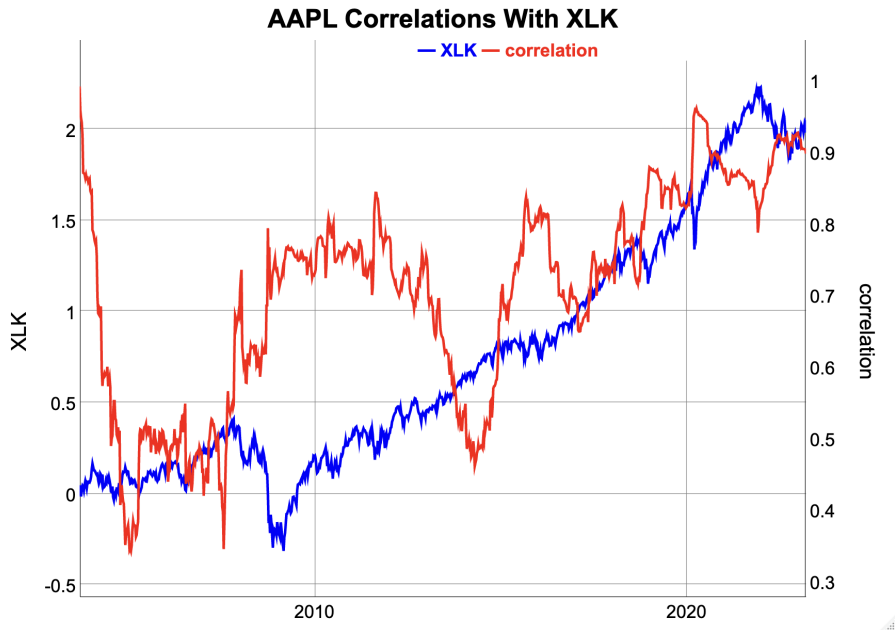
The trailing covariance can be updated using *online* recursive formulas with the decay factor λ:

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

The parameter λ determines the rate of decay of the weight of past returns. If λ is close to 1 then the decay is weak and past returns have a greater weight, and the trailing mean values have a stronger dependence on past returns. This is equivalent to a long look-back interval. And vice versa if λ is close to 0.

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

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



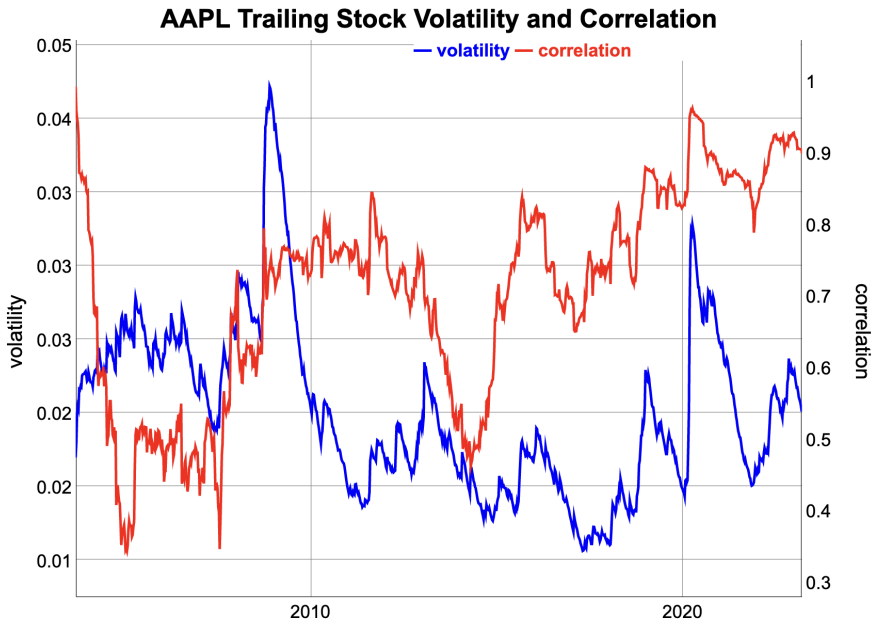

```
> # Plot dygraph of XLK returns and AAPL correlations
> datav <- cbind(cumsum(retp$XLK), correlv)
> colnames(datav)[2] <- "correlation"
> colv <- colnames(datav)
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> dygraphs::dygraph(datav[endd], main="AAPL Correlations With XLK")
+   dyAxis("y",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

Trailing Stock Correlations and Volatilities

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

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

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



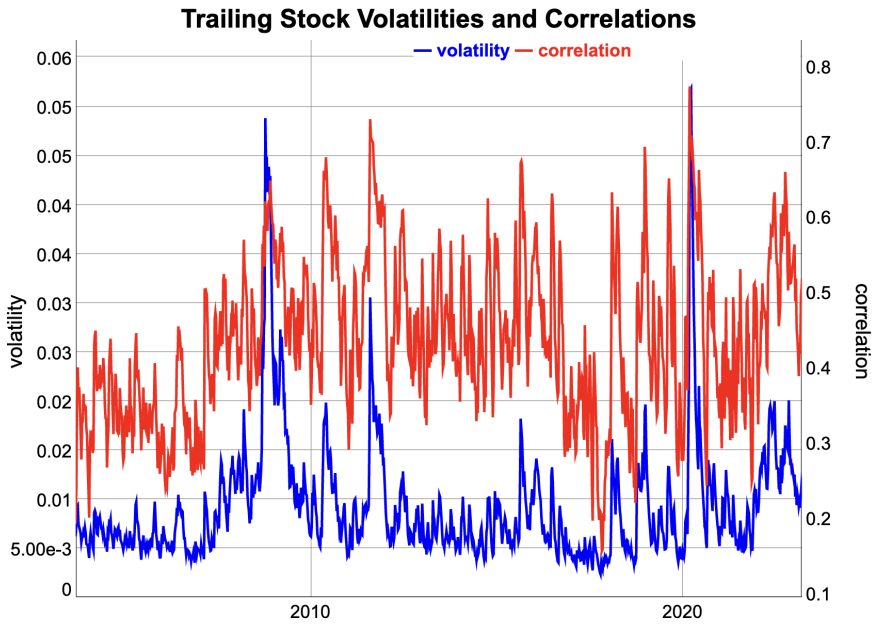

```
> # Scatterplot of trailing stock volatilities and correlations
> volv <- sqrt(covarv[, 2])
> plot(x=volv[endd], y=correlv[endd, ], pch=1, col="blue",
+  xlab="AAPL volatility", ylab="Correlation",
+  main="Trailing Volatilities and Correlations of AAPL vs XLK")
> # Interactive scatterplot of trailing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

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

Stock Portfolio Correlations and Volatilities

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

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




```
> # Calculate the portfolio returns
> retvti <- na.omit(rutils::etfenv$returns$VTI)
> colnames(retvti) <- "VTI"
> datev <- zoo::index(retvti)
> retp <- retstock100
> retp[is.na(retp)] <- 0
> retp <- retp[datev]
> nrows <- NROW(retp)
> nstocks <- NCOL(retp)
> head(retp[, 1:5])
> # Calculate the average trailing portfolio correlations
> lambdaf <- 0.9
> correlv <- sapply(retp, function(retp) {
+   covarv <- HighFreq::run_covar(cbind(retvti, retp), lambdaf)
+   covarv[, 1, drop=FALSE]/sqrt(covarv[, 2]*covarv[, 3])
+ })  # end sapply
> correlv[is.na(correlv)] <- 0
> correlp <- rowMeans(correlv)
> # Scatterplot of trailing stock volatilities and correlations
> volvti <- sqrt(HighFreq::run_var(retvti, lambdaf)[, 2])
> endd <- rutils::calc_endpoints(retvti, interval="weeks")
> plot(x=volvti[endd], y=correlp[endd],
+  xlab="volatility", ylab="correlation",
+  main="Trailing Stock Volatilities and Correlations")
```

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

Homework Assignment

Required

Study all the lecture slides in `FRE7241_Lecture_6.pdf`, and run all the code in `FRE7241_Lecture_6.R`

Recommended

- Read about *estimator shrinkage*:
  Aswani Regression Shrinkage Bias Variance Tradeoff.pdf
  Blei Regression Lasso Shrinkage Bias Variance Tradeoff.pdf
- Read about *optimization methods*:
  Bolker Optimization Methods.pdf
  Yollin Optimization.pdf
  DEoptim Introduction.pdf
  Ardia DEoptim Portfolio Optimization.pdf
  Boudt DEoptim Portfolio Optimization.pdf
  Boudt DEoptim Large Portfolio Optimization.pdf
  Mullen Package DEoptim.pdf
- Read about *momentum*:
  Bouchaud Momentum Mean Reversion Equity Returns.pdf