Portfolio Construction

FRE6871 & FRE7241, Fall 2022

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

October 30, 2022

draft: Asset Allocation

Asset allocation means dividing an investment portfolio among different asset classes, such as large company stocks, small company stocks, international stocks, bonds, commodities, cash, etc.

The goal of asset allocation is to diversify the sources of returns and to reduce risk, depending on the investor's risk tolerance, investment goals, and investment time horizon.

For example, an investor who needs to fund college for her children might put some of her investments into government bonds that mature when her children will need to pay for college.

1,600 years ago rabbi Isaac bar Aha proposed a simple heuristic method (rule of thumb) for asset allocation: "put a third in land, a third in merchandise, and a third in cash".

```
> library(PortfolioAnalytics)
> # Use ETF returns from package rutils
> library(rutils)
> portf_names <- c("VTI", "IEF", "DBC", "XLF",
+   "VNQ", "XLP", "XLV", "XLU", "XLB", "XLE")
> # Initial portfolio to equal weights
> portf_init <- rep(1/NROW(portf_names), NROW(portf_names))
> # Named vector
> names(portf_init) <- portf_names
> # Create portfolio object
> portf_init <- portfolio.spec(
+   assets=portf_init)
```

```
> # Add constraints
> portf_maxSR <- add.constraint(
+   portfolio=portf_init,  # Initial portfolio
+   type="weightsum",  # Constraint sum weights
+   min_sum=0.9, max_sum=1.1)
> # Add constraints
> portf_maxSR <- add.constraint(
+   portfolio=portf_maxSR,
+   type="long_only")  # box constraint min=0, max=1
> # Add objectives
> portf_maxSR <- add.objective(
+   portfolio=portf_maxSR,
+   type="return",  # Maximize mean return
+   name="mean")
> # Add objectives
> portf_maxSR <- add.objective(
+   portfolio=portf_maxSR,
+   type="risk",  # Minimize StdDev
+   name="StdDev")
```

draft: Portfolio Construction

Portfolio construction means determining the amounts to be invested different assets, such as specific stocks, bonds, commodities, etc.

Portfolio optimization is one approach to portfolio construction.

Heuristic Methods for Portfolio Construction

Victor DeMiguel and others have demonstrated that optimized portfolios perform poorly out-of-sample, and that simple heuristic methods can perform better than portfolio optimization.

```
> library(PortfolioAnalytics)
> # Use ETF returns from package rutils
> library(rutils)
> portf_names <- c("VTI", "IEF", "DBC", "XLF",
+   "VNQ", "XLP", "XLV", "XLU", "XLB", "XLE")
> # Initial portfolio to equal weights
> portf_init <- rep(1/NROW(portf_names), NROW(portf_names))
> # Named vector
> names(portf_init) <- portf_names
> # Create portfolio object
> portf_init <- portfolio.spec(
+   assets=portf_init)
```

```
> # Add constraints
> portf_maxSR <- add.constraint(
+   portfolio=portf_init,  # Initial portfolio
+   type="weightsum",  # Constraint sum weights
+   min_sum=0.9, max_sum=1.1)
> # Add constraints
> portf_maxSR <- add.constraint(
+   portfolio=portf_maxSR,
+   type="long_only")  # box constraint min=0, max=1
> # Add objectives
> portf_maxSR <- add.objective(
+   portfolio=portf_maxSR,
+   type="return",  # Maximize mean return
+   name="mean")
> # Add objectives
> portf_maxSR <- add.objective(
+   portfolio=portf_maxSR,
+   type="risk",  # Minimize StdDev
+   name="StdDev")
```

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:

$$\frac{d(\mathbf{v}^T \mathbb{1})}{d\mathbf{v}} = d_v[\mathbf{v}^T \mathbb{1}] = d_v[\mathbb{1}^T \mathbf{v}] = \mathbb{1}^T$$

$$d_v[\mathbf{v}^T \mathbf{w}] = d_v[\mathbf{w}^T \mathbf{v}] = \mathbf{w}^T$$

$$d_v[\mathbf{v}^T \mathbb{A}\, \mathbf{w}] = \mathbf{w}^T \mathbb{A}^T$$

$$d_v[\mathbf{v}^T \mathbb{A}\, \mathbf{v}] = \mathbf{v}^T \mathbb{A} + \mathbf{v}^T \mathbb{A}^T$$

Portfolio Weight Constraints

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

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

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

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

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

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

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

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

Box constraints limit the individual weights, for example: $0 \leq w_i \leq 1$.

Box constraints are often applied when constructing long-only portfolios, or when limiting the exposure to some stocks.

Maximum Return Portfolio Using Linear Programming

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

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

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

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

And a box constraint:

$$0 \leq w_i \leq 1$$

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

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

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

The *Minimum Variance* Portfolio Under *Linear* Constraints

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

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

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

Where λ is a *Lagrange multiplier*.

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

$$d_w[\mathbf{w}^T \mathbb{1}] = d_w[\mathbb{1}^T \mathbf{w}] = \mathbb{1}^T$$
$$d_w[\mathbf{w}^T \mathbf{r}] = d_w[\mathbf{r}^T \mathbf{w}] = \mathbf{r}^T$$
$$d_w[\mathbf{w}^T \mathbb{C}\,\mathbf{w}] = \mathbf{w}^T \mathbb{C} + \mathbf{w}^T \mathbb{C}^T$$

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

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

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

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

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

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

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

And finally the portfolio weights are then equal to:

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

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

Variance of the *Minimum Variance* Portfolio

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

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

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

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

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

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

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

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

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

The *Efficient Portfolios*

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

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

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

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

Where λ_1 and λ_2 are the *Lagrange multipliers*.

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

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

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

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

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

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

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

The above can be written in matrix notation as:

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

Or:

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

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

The *Lagrange multipliers* can be solved as:

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

The *Efficient Portfolio* Weights

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

$$\mathbf{w} = \frac{1}{2}(\lambda_1 \, \mathbb{C}^{-1}\mathbb{1} + \lambda_2 \, \mathbb{C}^{-1}\mathbf{r}) =$$
$$\frac{1}{2}\begin{bmatrix} \mathbb{C}^{-1}\mathbb{1} \\ \mathbb{C}^{-1}\mathbf{r} \end{bmatrix}^T \lambda = \begin{bmatrix} \mathbb{C}^{-1}\mathbb{1} \\ \mathbb{C}^{-1}\mathbf{r} \end{bmatrix}^T \mathbb{F}^{-1} \, u =$$
$$\frac{1}{ac - b^2}\begin{bmatrix} \mathbb{C}^{-1}\mathbb{1} \\ \mathbb{C}^{-1}\mathbf{r} \end{bmatrix}^T \begin{bmatrix} c & -b \\ -b & a \end{bmatrix} \begin{bmatrix} 1 \\ r_t \end{bmatrix} =$$
$$\frac{(c - br_t)\,\mathbb{C}^{-1}\mathbb{1} + (ar_t - b)\,\mathbb{C}^{-1}\mathbf{r}}{ac - b^2}$$

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

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

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

Variance of the *Efficient Portfolios*

The *efficient portfolio* variance is equal to:

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

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

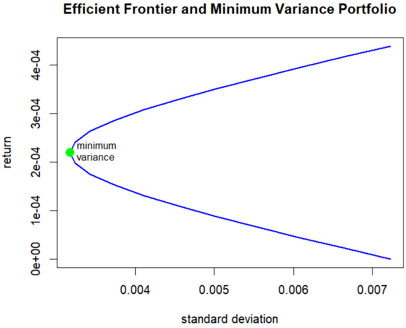
The vertex of the *parabola* is at $r_t = b/a = \mathbb{1}^T \mathbb{C}^{-1} \mathbf{r} / \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$ and $\sigma^2 = 1/c = 1/\mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$.

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

The *Efficient Frontier*

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

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



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

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 variance σ^2 by the target return r_t:

$$\frac{d\sigma^2}{dr_t} = 2\sigma \frac{d\sigma}{dr_t} = \frac{2ar_t - 2b}{ac - b^2}$$

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

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

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

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

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

$$r_t - \frac{ar_t^2 - 2br_t + c}{ac - b^2} \frac{ac - b^2}{ar_t - b} =$$

$$r_t - \frac{ar_t^2 - 2br_t + c}{ar_t - b} = \frac{br_t - c}{ar_t - b}$$

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

The *Capital Market Line*

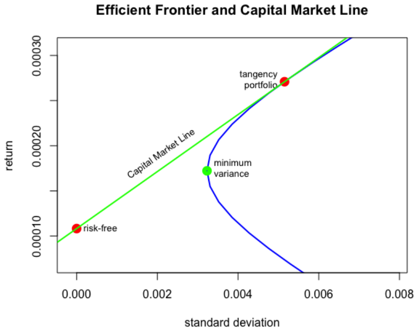
The *Capital Market Line* (CML) is the tangent line connecting the risk-free point ($\sigma = 0, r = r_f$) with a single tangent point on the *efficient frontier*.

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




```
> # Plot tangent portfolio
> points(x=stdev, y=retarget, col="red", lwd=6)
> text(x=stdev, y=retarget, labels="tangency\nportfolio", pos=2, ce
> # Plot risk-free point
> points(x=0, y=riskf, col="red", lwd=6)
> text(x=0, y=riskf, labels="risk-free", pos=4, cex=0.8)
> # Plot tangent line
> abline(a=riskf, b=sharper, lwd=2, col="green")
> rangev <- par("usr")
> 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 *Market Portfolio*

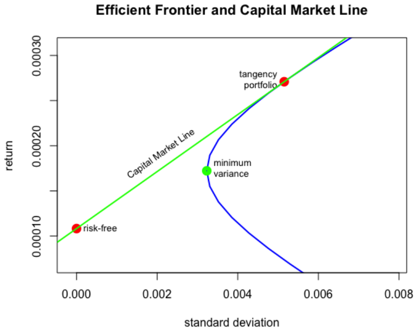
The *market portfolio* is the *tangency portfolio* corresponding to the given *risk-free* rate.

The points on the *Capital Market Line* represent portfolios consisting of the *market portfolio* and the *risk-free* asset.

The *CML* portfolios above the tangent point are levered with respect to the *market portfolio* through borrowing at the *risk-free* rate r_f.

The *CML* portfolios below the tangent point are delevered with respect to the *market portfolio* through investing at the *risk-free* rate r_f.

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

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

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

We then get:

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

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

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

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

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

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

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

The mean portfolio return can be written as:

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

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

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

Returns and Variance of Maximum *Sharpe* Portfolio

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

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

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

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

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

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

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

The *Sharpe* ratio is equal to:

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

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

Optimal Portfolios Under Zero Correlation

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

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

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

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

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

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

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

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

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

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

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

draft: Portfolio Optimization Using Principal Components

First apply Principal Component Analysis and then perform portfolio optimization using the principal components.

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

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

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

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

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

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

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

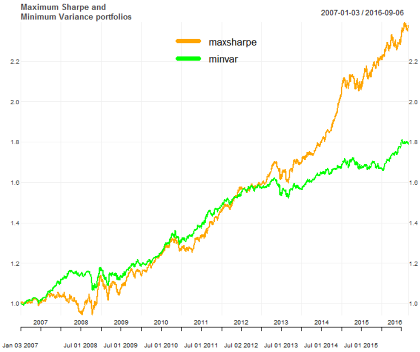
The maximum *Sharpe* portfolio weights are proportional to the ratio of excess returns divided by the individual variances:

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

Maximum *Sharpe* and *Minimum Variance* Performance

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

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



The *Efficient Frontier* and *Capital Market Line*

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

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

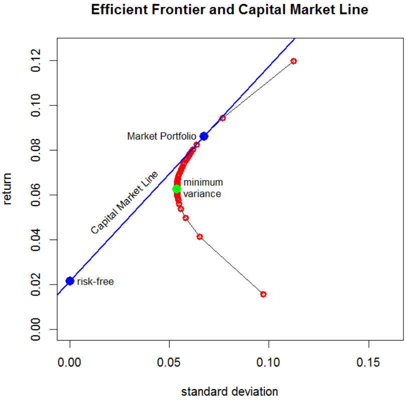
The *Efficient Frontier* is the set of *efficient portfolios*, that have the lowest risk (standard deviation) for the given level of return.

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

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

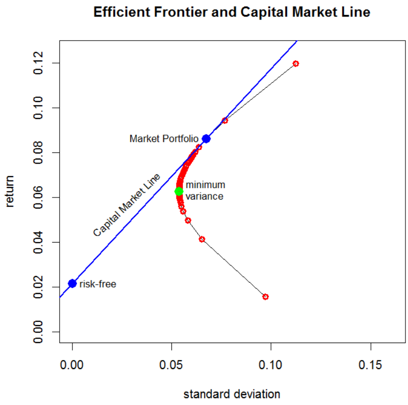
The maximum *Sharpe* portfolios are also called *tangency* portfolios, since they are the tangent point on the *Efficient Frontier*.

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



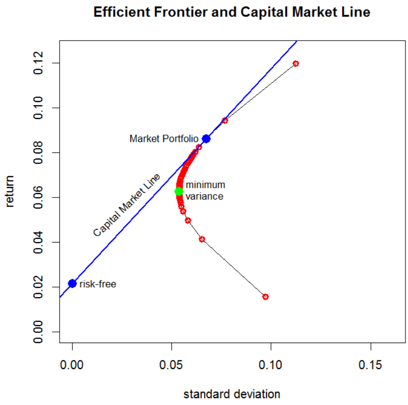
Plotting *Efficient Frontier* and Maximum *Sharpe* Portfolios

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



Plotting the *Capital Market Line*

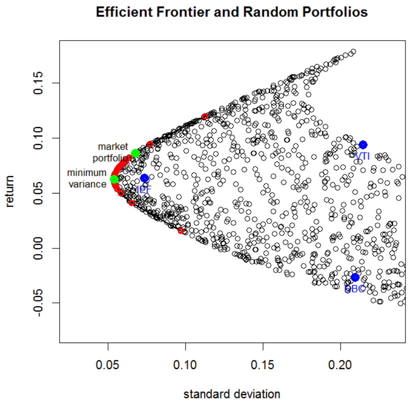
```
> # Plot minimum variance portfolio
> points(x=stdev, y=retsp, col="green", lwd=6)
> text(stdev, retsp, labels="minimum \nvariance",
+      pos=4, cex=0.8)
> # Draw Capital Market Line
> sortv <- sort(effront[, 1])
> riskf <- sortv[findInterval(x=0.5*retsp, vec=sortv)]
> points(x=0, y=riskf, col="blue",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



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

Plotting Random Portfolios

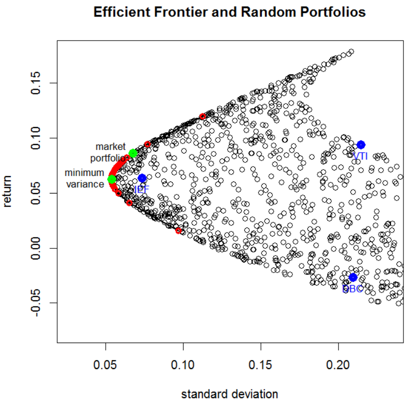
```
> # Calculate random portfolios
> nportf <- 1000
> randportf <- sapply(1:nportf, function(it) {
+   weightv <- runif(nstocks-1, min=-0.25, max=1.0)
+   weightv <- c(weightv, 1-sum(weightv))
+   # Portfolio return and standard deviation
+   c(return=252*sum(weightv*retsm),
+     stddev=sqrt(252*drop(weightv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




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

draft: Plotting Random Portfolios Without Using Covariance Matrix

```
> # Vector of symbol names
> symbolv <- c("VTI", "IEF", "DBC")
> nstocks <- NROW(symbolv)
> # Calculate random portfolios
> nportf <- 1000
> randportf <- sapply(1:nportf, function(it) {
+   weightv <- runif(nstocks, min=0, max=10)
+   weightv <- weightv/sum(weightv)
+   retsp <- rutils::etfenv$returns[, symbolv] %*% weightv
+   100*c(ret=mean(retsp), sd=sd(retsp))
+ })  # end sapply
> # Plot scatterplot of random portfolios
> x11(width=6, height=5)
> plot(x=randportf[2, ], y=randportf[1, ], xlim=c(0, max(randportf[2
+      main="Random portfolios",
+      ylim=c(min(0, min(randportf[1, ])), max(randportf[1, ])),
+      xlab=rownames(randportf)[2], ylab=rownames(randportf)[1])
```




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

draft: Efficient Frontier for Two-asset Portfolios

The covariance matrix for two assets is equal to:

$$\mathbb{C} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}$$

Where σ_{12} is the covariance of returns between the two assets, The excess returns of a two-asset portfolio are equal to:

$$R = w\mu_1 + (1 - w)\mu_2$$

Solving for the weight **w**:

$$w = (R - \mu_2)/(\mu_1 - \mu_2)$$

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

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

If the correlations of returns are equal to zero, then The *minimum variance* portfolio weights are proportional to the inverse of the individual variances:

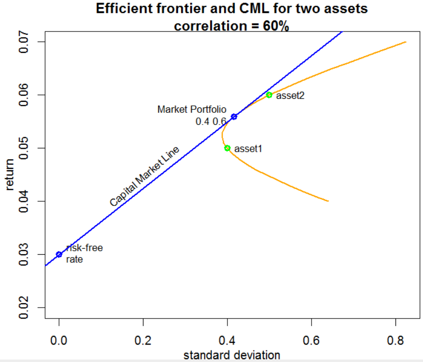
$$w_i = \frac{1}{\sigma_i^2 \sum_{i=1}^n \sigma_i^{-2}}$$

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

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

Plotting Efficient Frontier for Two-asset Portfolios

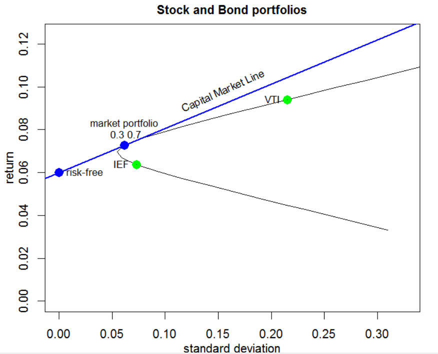
```
> riskf <- 0.03
> retsp <- c(asset1=0.05, asset2=0.06)
> stdevs <- c(asset1=0.4, asset2=0.5)
> corrp <- 0.6
> covmat <- matrix(c(1, corrp, corrp, 1), nc=2)
> covmat <- t(t(stdevs*covmat)*stdevs)
> weightv <- seq(from=(-1), to=2, length.out=31)
> weightv <- cbind(weightv, 1-weightv)
> retsp <- weightv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




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

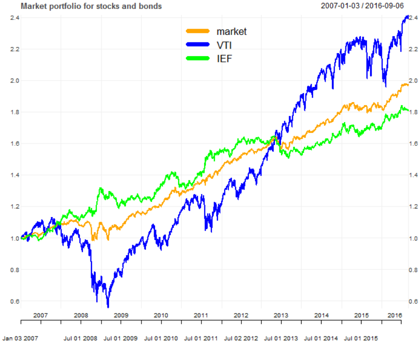
```
> # Vector of symbol names
> symbolv <- c("VTI", "IEF")
> # Matrix of portfolio weights
> weightv <- seq(from=(-1), to=2, length.out=31)
> weightv <- cbind(weightv, 1-weightv)
> # Calculate portfolio returns and volatilities
> retsp <- rutils::etfenv$returns[, symbolv]
> retsp <- retsp %*% t(weightv)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




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

Performance of Market Portfolio for Stocks and Bonds

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



Conditional Value at Risk (*CVaR*)

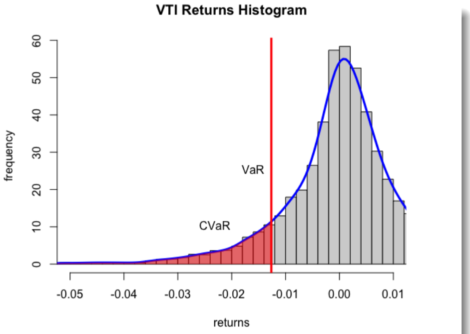
The *Conditional Value at Risk* (*CVaR*) is equal to the average of the *VaR* for confidence levels less than a given confidence level α:

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

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

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

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




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

CVaR Portfolio Weights Using Linear Programming

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

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

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

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

And a box constraint:

$$0 \leq w_i \leq 1$$

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

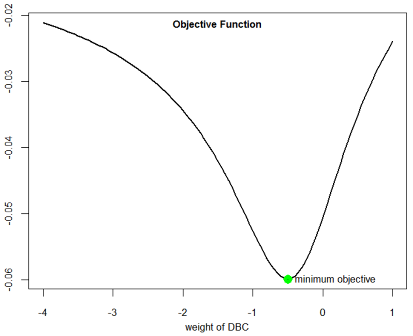
```
> library(rutils)  # Load rutils
> library(Rglpk)
> # Vector of symbol names and returns
> symbolv <- c("VTI", "IEF", "DBC")
> nstocks <- NROW(symbolv)
> retsp <- na.omit(rutils::etfenv$returns[, symbolv])
> retsm <- colMeans(retsp)
> confl <- 0.05
> rmin <- 0 ; wmin <- 0 ; wmax <- 1
> weightsum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

Sharpe Ratio Objective Function

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

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

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




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

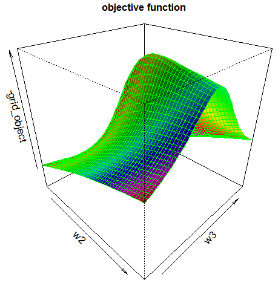
Perspective Plot of Portfolio Objective Function

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

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

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

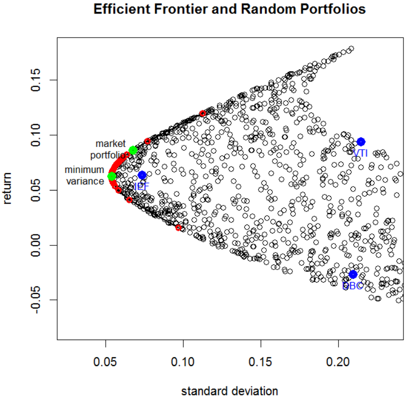
```
> # Vectorize function with respect to all weights
> objvec <- Vectorize(
+   FUN=function(w1, w2, w3) objfun(c(w1, w2, w3)),
+   vectorize.args=c("w2", "w3"))  # end Vectorize
> # Calculate objective on 2-d (w2 x w3) parameter grid
> w2 <- seq(-3, 7, length=50)
> w3 <- seq(-5, 5, length=50)
> grid_object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




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

draft: Multi-dimensional Portfolio Optimization

```
> # Vector of initial portfolio weights equal to 1
> weightv <- rep(1, nstocks)
> names(weightv) <- symbolv
> # Objective function equal to standard deviation of returns
> objfun <- function(weightv) {
+   retsp <- retsp %*% weightv
+   sd(retsp)/sum(weightv)
+ }  # end objfun
> # objfun() for equal weight portfolio
> objfun(weightv)
> objfun(2*weightv)
> # Perform portfolio optimization
> optiml <- optim(par=weightv,
+           fn=objfun,
+           method="L-BFGS-B",
+           upper=rep(10, nstocks),
+           lower=rep(-10, nstocks))
> # Rescale the optimal weights
> weightv <- optiml$par/sum(optiml$par)
> # Minimum variance portfolio returns
> retsoptim <- xts(x=retsp %*% weightv,
+             order.by=zoo::index(retsp))
> chart_Series(x=exp(cumsum(retsoptim)), name="minvar portfolio")
> # Add green point for minimum variance portfolio
> optim_sd <- 100*sd(retsoptim)
> optim_ret <- 100*mean(retsoptim)
> points(x=optim_sd, y=optim_ret, col="green", lwd=6)
> text(x=optim_sd, y=optim_ret, labels="minvar", pos=2, cex=0.8)
>
>
> # Objective function equal to minus Sharpe ratio
> riskf <- 0.03
> objfun <- function(weightv) {
+   retsp <- 100*rutils::etfenv$returns[, names(weightv)] %*% weightv / sum(weightv)
+   -mean(retsp-riskf)/sd(retsp)
+ }  # end objfun
> # Perform portfolio optimization
```



Multi-dimensional Portfolio Optimization

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

The argument `par` are the initial parameter values.

The argument `fn` is the objective function to be minimized.

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

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

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

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

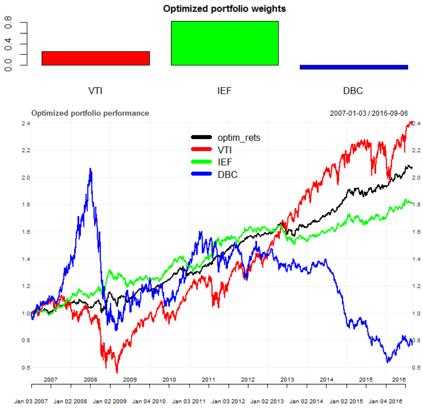
`optim()` returns a list containing the location of the minimum and the objective function value.

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

Optimized Portfolio Performance

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

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



draft: Mean-Variance Portfolio Optimization

The mean-variance objective function is designed to maximize portfolio returns and minimize their variance:

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

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

The mean-variance optimal portfolio is defined as

$$\theta_{MLE} = \arg\max_{\theta} \mathcal{L}(\theta|x) \mathbf{w}^T \mathbb{C}\, \mathbf{w}$$

Where the sum of portfolio weights $\mathbf{w}$ is constrained to equal 1: $\mathbf{w}^T \mathbb{1} = \sum_{i=1}^{n} w_i = 1$.

Legacy stuff below:
A Linear Regression model with p explanatory variables $\{x_j\}$, is defined by the formula:

$$z_i = \alpha + \sum_{j=1}^{k} \beta_j x_{i,j} + \varepsilon_i$$

Or in vector notation:

$$z = \alpha + \beta x + \varepsilon$$

The response variable z and the p explanatory variables $\{x_j\}$ each contain $\mathtt{n}$ observations.

The response variable z is a vector of length $\mathtt{n}$, and the explanatory variable x is a (n, p)-dimensional matrix.

The OLS estimate for α is given by:

$$\alpha = z^T \mathbb{1} - \beta x^T \mathbb{1}$$

If the variables are de-meaned, then the OLS estimate for β is given by equating the RSS derivative to zero:

$$RSS_\beta = -2(z - \beta x)^T x = 0$$
$$x^T z - \beta x^T x = 0$$
$$\beta = (x^T x)^{-1} x^T z$$

The matrix $x^T x$ is the covariance matrix of the matrix x.

The covariance matrix $x^T x$ is invertible if the columns of x are linearly independent.

The matrix $(x^T x)^{-1} x^T$ is known as the *Moore-Penrose pseudo-inverse* of the matrix x.

In the special case when the inverse matrix x^{-1} does exist, then the *pseudo-inverse* matrix simplifies to the inverse: $(x^T x)^{-1} x^T = x^{-1} (x^T)^{-1} x^T = x^{-1}$

Package *quadprog* for Quadratic Programming

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

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

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

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

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

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

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

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

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

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

Portfolio Optimization Using Package *quadprog*

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

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

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

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

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

$$0 \leq w_i \leq 1$$

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

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

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

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

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

Package *DEoptim* for Global Optimization

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

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

The first generation of solutions is selected randomly.

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

The best solutions are selected for creating the next generation.

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

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

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

Portfolio Optimization Using Package *Deoptim*

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

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

Portfolio Optimization Using *Shrinkage*

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

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

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

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

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

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

draft: Portfolio Optimization Packages in R

The following R packages provide functions for portfolio optimization:

- package *PortfolioAnalytics*: relies on packages *xts*, *ROI*, and *DEoptim*,
- package *parma*: relies on packages *xts*, *Rglpk*, and *quadprog*,
- package *fPortfolio* from the *Rmetrics* suite: relies on packages *tseries*, *Rglpk*, and *quadprog*,

These portfolio optimization packages call generic optimization functions written in compiled C++

```
> # Portfolio optimization
```

Package *PortfolioAnalytics*

The package *PortfolioAnalytics* contains functions and data sets for portfolio optimization.

The function data() loads external data or listv data sets in a package.

```
> library(PortfolioAnalytics)  # load package "PortfolioAnalytics"
> # get documentation for package "PortfolioAnalytics"
> packageDescription("PortfolioAnalytics")  # get short description
>
> help(package="PortfolioAnalytics")  # load help page
>
> data(package="PortfolioAnalytics")  # list all datasets in "PortfolioAnalytics"
>
> ls("package:PortfolioAnalytics")  # list all objects in "PortfolioAnalytics"
>
> detach("package:PortfolioAnalytics")  # remove PortfolioAnalytics from search p
```

Portfolio Definition

Portfolios are defined by a named vector of asset weights, and portfolio constraints and objectives.

`portfolio.spec` creates a portfolio object that contains asset weights, constraints, and objectives.

`add.constraint` adds or updates constraints on of the portfolio object.

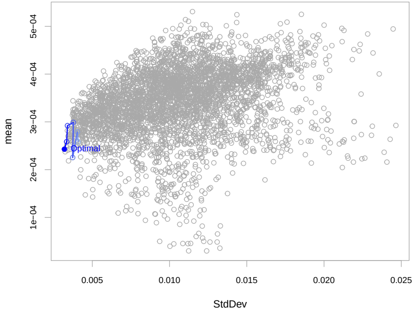
`add.objective` adds or updates risk/return objectives of the portfolio object.

```
> library(PortfolioAnalytics)
> # Use ETF returns from package rutils
> library(rutils)
> portf_names <- c("VTI", "IEF", "DBC", "XLF",
+   "VNQ", "XLP", "XLV", "XLU", "XLB", "XLE")
> # Initial portfolio to equal weights
> portf_init <- rep(1/NROW(portf_names), NROW(portf_names))
> # named vector
> names(portf_init) <- portf_names
> # Create portfolio object
> portf_init <- portfolio.spec(assets=portf_init)
```

```
> # Add constraints
> portf_maxSR <- add.constraint(
+   portfolio=portf_init,  # Initial portfolio
+   type="weightsum",  # Constraint sum weights
+   min_sum=0.9, max_sum=1.1)
> # Add constraints
> portf_maxSR <- add.constraint(
+   portfolio=portf_maxSR,
+   type="long_only")  # box constraint min=0, max=1
> # Add objectives
> portf_maxSR <- add.objective(
+   portfolio=portf_maxSR,
+   type="return",  # Maximize mean return
+   name="mean")
> # Add objectives
> portf_maxSR <- add.objective(
+   portfolio=portf_maxSR,
+   type="risk",  # Minimize StdDev
+   name="StdDev")
```

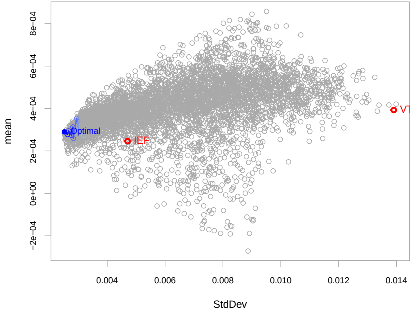
Portfolio Optimization

```
> # Perform optimization of weights
> maxSR_DEOpt <- optimize.portfolio(
+   R=rutils::etfenv$returns[, portf_names],  # Specify returns
+   portfolio=portf_maxSR,  # Specify portfolio
+   optimize_method="DEoptim", # Use DEoptim
+   maxSR=TRUE,  # Maximize Sharpe
+   trace=TRUE, traceDE=0)
> # Plot optimization
> chart.RiskReward(maxSR_DEOpt,
+   risk.col="stddev",
+   return.col="mean")
> maxSR_DEOpt$weights
> maxSR_DEOpt$objective_measures$mean[1]
> maxSR_DEOpt$objective_measures$StdDev[[1]]
```



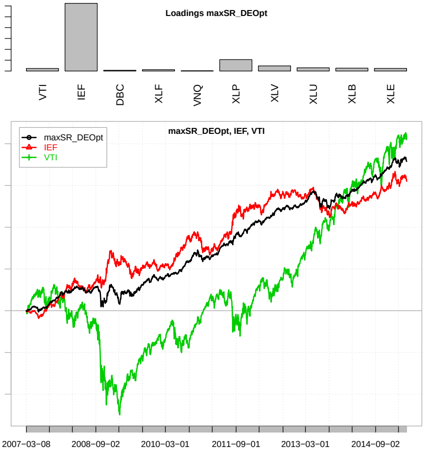
Portfolio Optimization Scatterplot

```
> # Plot optimization
> chart.RiskReward(maxSR_DEOpt,
+   risk.col="StdDev",
+   return.col="mean")
>
> # Plot risk/ret points in portfolio scatterplot
> risk_ret_points <- function(rets=rutils::etfenv$returns,
+   risk=c("sd", "ETL"), symbolv=c("VTI", "IEF")) {
+   risk <- match.arg(risk)  # Match to arg list
+   if (risk=="ETL") {
+     stopifnot(
+ "package:PerformanceAnalytics" %in% search() ||
+ require("PerformanceAnalytics", quietly=TRUE))
+   }  # end if
+   risk <- match.fun(risk)  # Match to function
+   risk_ret <- t(sapply(rets[, symbolv],
+      function(xtsv)
+  c(ret=mean(xtsv), risk=abs(risk(xtsv)))))
+   points(x=risk_ret[, "risk"], y=risk_ret[, "ret"],
+    col="red", lwd=3)
+   text(x=risk_ret[, "risk"], y=risk_ret[, "ret"],
+  labels=rownames(risk_ret), col="red",
+  lwd=2, pos=4)
+ }  # end risk_ret_points
>
> risk_ret_points()
```



Optimized Sharpe Portfolio

```
> plot_portf <- function(portfolio,
+       rets_data=rutils::etfenv$returns) {
+   weightv <- portfolio$weights
+   portf_names <- names(weightv)
+   # Calculate xts of portfolio
+   portf_max <- xts(
+     rets_data[, portf_names] %*% weightv,
+     order.by=zoo::index(rets_data))
+   colnames(portf_max) <-
+     deparse(substitute(portfolio))
+   graph_params <- par(oma=c(1, 0, 1, 0),
+     mgp=c(2, 1, 0), mar=c(2, 1, 2, 1),
+     cex.lab=0.8, cex.axis=1.0,
+     cex.main=0.8, cex.sub=0.5)
+   layout(matrix(c(1,2), 2),
+     widths=c(1,1), heights=c(1,3))
+   barplot(weightv, names.arg=portf_names,
+     las=3, ylab="", xlab="Symbol", main="")
+   title(main=paste("Loadings",
+           colnames(portf_max)), line=(-1))
+   chart.CumReturns(
+     cbind(portf_max, rets_data[, c("IEF", "VTI")]),
+     lwd=2, ylab="", legend.loc="topleft", main="")
+   title(main=paste0(colnames(portf_max),
+               ", IEF, VTI"), line=(-1))
+   par(graph_params)  # restore original parameters
+   invisible(portf_max)
+ }  # end plot_portf
> maxSR_DEOpt_xts <- plot_portf(portfolio=maxSR_DEOpt)
```



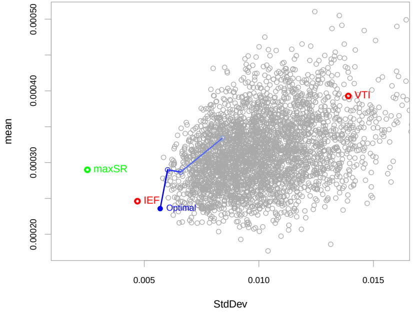
Portfolio Leverage Constraints

The leverage constraint applies to the sum of absolute weights.

```
> # Add leverage constraint abs(weightsum)
> portf_maxSRN <- add.constraint(
+   portfolio=portf_init, type="leverage",
+   min_sum=0.9, max_sum=1.1)
> # Add box constraint long/short
> portf_maxSRN <- add.constraint(
+   portfolio=portf_maxSRN,
+   type="box", min=-0.2, max=0.2)
>
> # Add objectives
> portf_maxSRN <- add.objective(
+   portfolio=portf_maxSRN,
+   type="return",  # Maximize mean return
+   name="mean")
> # Add objectives
> portf_maxSRN <- add.objective(
+   portfolio=portf_maxSRN,
+   type="risk",  # Minimize StdDev
+   name="StdDev")
```

Portfolio Leverage Constraint Optimization

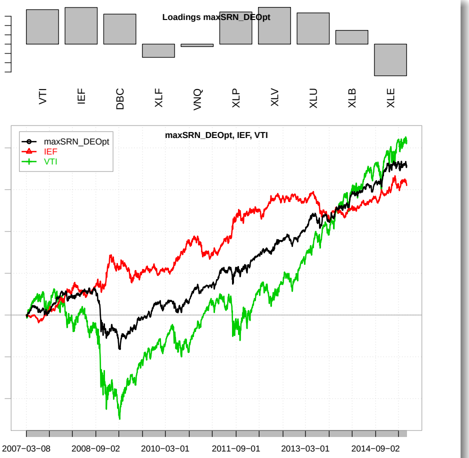
```
> # Perform optimization of weights
> maxSRN_DEOpt <- optimize.portfolio(
+   R=rutils::etfenv$returns[, portf_names],  # Specify returns
+   portfolio=portf_maxSRN,  # Specify portfolio
+   optimize_method="DEoptim", # Use DEoptim
+   maxSR=TRUE,  # Maximize Sharpe
+   trace=TRUE, traceDE=0)
> # Plot optimization
> chart.RiskReward(maxSRN_DEOpt,
+   risk.col="StdDev",
+   return.col="mean",
+   xlim=c(
+     maxSR_DEOpt$objective_measures$StdDev[[1]]-0.001,
+     0.016))
>   points(x=maxSR_DEOpt$objective_measures$StdDev[[1]],
+    y=maxSR_DEOpt$objective_measures$mean[1],
+    col="green", lwd=3)
>   text(x=maxSR_DEOpt$objective_measures$StdDev[[1]],
+    y=maxSR_DEOpt$objective_measures$mean[1],
+  labels="maxSR", col="green",
+  lwd=2, pos=4)
> # Plot risk/ret points in portfolio scatterplot
> risk_ret_points()
```




```
> maxSRN_DEOpt$weights
> maxSRN_DEOpt$objective_measures$mean[1]
> maxSRN_DEOpt$objective_measures$StdDev[[1]]
```

Optimized Leverage Constraint Portfolio

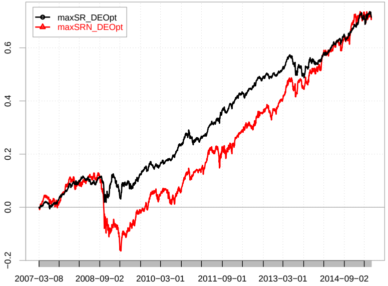
```
> maxSRN_DEOpt_xts <- plot_portf(portfolio=maxSRN_DEOpt)
```



Sharpe Portfolios CumReturns Plots

`chart.CumReturns()` plots the cumulative returns of a time series of returns.

```
> chart.CumReturns(
+   cbind(maxSR_DEOpt_xts, maxSRN_DEOpt_xts),
+   lwd=2, ylab="",
+   legend.loc="topleft", main="")
> rbind(maxSR_DEOpt$weights, maxSRN_DEOpt$weights)
> c(maxSR_DEOpt$objective_measures$mean,
+ maxSRN_DEOpt$objective_measures$mean)
> c(maxSR_DEOpt$objective_measures$StdDev[[1]],
+ maxSRN_DEOpt$objective_measures$StdDev[[1]])
```



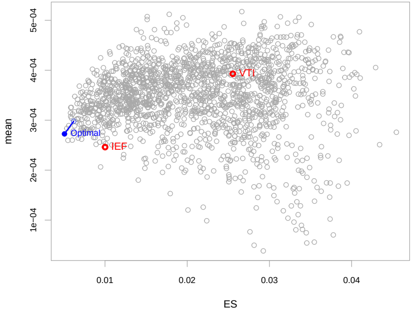
STARR Portfolio Constraints

The objective constraint applies to risk or return.

```
> # Add constraints
> portf_maxSTARR <- add.constraint(
+   portfolio=portf_init,  # Initial portfolio
+   type="weightsum",  # Constraint sum weights
+   min_sum=0.9, max_sum=1.1)
> # Add constraints
> portf_maxSTARR <- add.constraint(
+   portfolio=portf_maxSTARR,
+   type="long_only")  # box constraint min=0, max=1
> # Add objectives
> portf_maxSTARR <- add.objective(
+   portfolio=portf_maxSTARR,
+   type="return",  # Maximize mean return
+   name="mean")
> # Add objectives
> portf_maxSTARR <- add.objective(
+   portfolio=portf_maxSTARR,
+   type="risk",  # Minimize Expected Shortfall
+   name="ES")
```

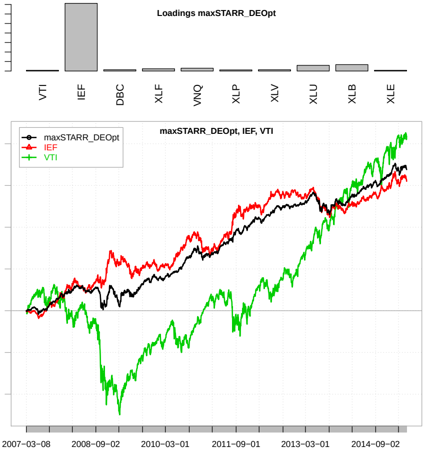
STARR Optimization

```
> # Perform optimization of weights
> maxSTARR_DEOpt <- optimize.portfolio(
+   R=rutils::etfenv$returns[, portf_names],  # Specify returns
+   portfolio=portf_maxSTARR,  # Specify portfolio
+   optimize_method="DEoptim", # Use DEoptim
+   maxSTARR=TRUE,  # Maximize STARR
+   trace=TRUE, traceDE=0)
>
> # Plot optimization
> chart.RiskReward(maxSTARR_DEOpt,
+   risk.col="ES",
+   return.col="mean")
> # Plot risk/ret points in portfolio scatterplot
> risk_ret_points(risk="ETL")
> maxSTARR_DEOpt$weights
> maxSTARR_DEOpt$objective_measures$mean[1]
> maxSTARR_DEOpt$objective_measures$ES[[1]]
```



Optimized STARR Portfolio

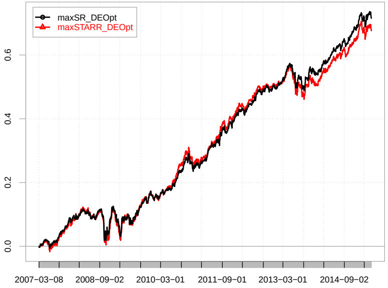
```
> maxSTARR_DEOpt_xts <-
+   plot_portf(portfolio=maxSTARR_DEOpt)
```



Sharpe STARR CumReturns Plots

`chart.CumReturns()` plots the cumulative returns of a time series of returns.

```
> chart.CumReturns(
+   cbind(maxSR_DEOpt_xts, maxSTARR_DEOpt_xts),
+   lwd=2, ylab="",
+   legend.loc="topleft", main="")
> rbind(maxSR_DEOpt$weights, maxSTARR_DEOpt$weights)
> c(maxSR_DEOpt$objective_measures$mean,
+ maxSTARR_DEOpt$objective_measures$mean)
> c(maxSR_DEOpt$objective_measures$StdDev[[1]],
+ maxSTARR_DEOpt$objective_measures$ES[[1]])
```



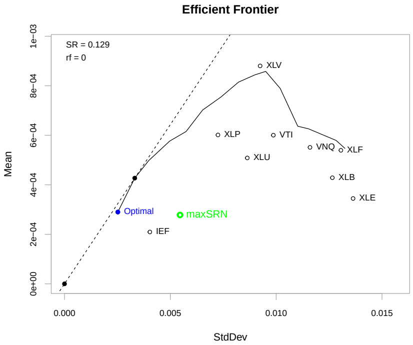
The Efficient Frontier and Capital Market Line

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

The Capital Market Line (CML) is the line drawn from the risk-free asset to the tangent point on the Efficient Frontier.

The tangent point on the *Efficient Frontier* is the *Market Portfolio*.

```
> # Plot the efficient frontier
> chart.EfficientFrontier(maxSR_DEOpt,
+           match.col="StdDev",
+           n.portfolios=15, type="l")
> points(x=maxSRN_DEOpt$objective_measures$StdDev[[1]],
+    y=maxSRN_DEOpt$objective_measures$mean[1],
+    col="green", lwd=3)
> text(x=maxSRN_DEOpt$objective_measures$StdDev[[1]],
+    y=maxSRN_DEOpt$objective_measures$mean[1],
+  labels="maxSRN", col="green",
+  lwd=2, pos=4)
```



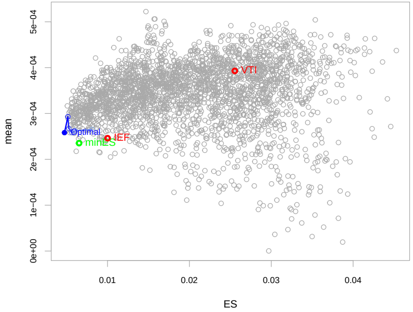
minES Portfolio Constraints

The objective constraint applies to risk or return.

```
> # Add constraints
> portf_minES <- add.constraint(
+   portfolio=portf_init,  # Initial portfolio
+   type="weightsum",  # Constraint sum weights
+   min_sum=0.9, max_sum=1.1)
> # Add constraints
> portf_minES <- add.constraint(
+   portfolio=portf_minES,
+   type="long_only")  # box constraint min=0, max=1
> # Add objectives
> portf_minES <- add.objective(
+   portfolio=portf_minES,
+   type="risk",  # Minimize ES
+   name="ES")
```

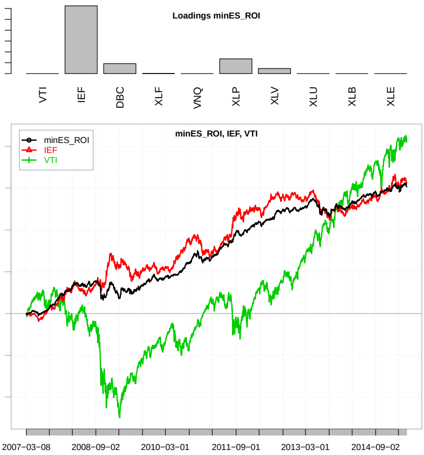
minES Optimization

```
> # Perform optimization of weights
> minESROI <- optimize.portfolio(
+   R=rutils::etfenv$returns[, portf_names],  # Specify returns
+   portfolio=portf_minES,  # Specify portfolio
+   optimize_method="ROI", # Use ROI
+   trace=TRUE, traceDE=0)
>
> # Plot optimization
> chart.RiskReward(maxSTARR_DEOpt,
+   risk.col="ES",
+   return.col="mean")
>   points(x=minESROI$objective_measures$ES[[1]],
+    y=mean(minESROI_xts),
+    col="green", lwd=3)
>   text(x=minESROI$objective_measures$ES[[1]],
+    y=mean(minESROI_xts),
+  labels="minES", col="green",
+  lwd=2, pos=4)
> # Plot risk/ret points in portfolio scatterplot
> risk_ret_points(risk="ETL")
> minESROI$weights
> minESROI$objective_measures$ES[[1]]
```



Optimized minES Portfolio

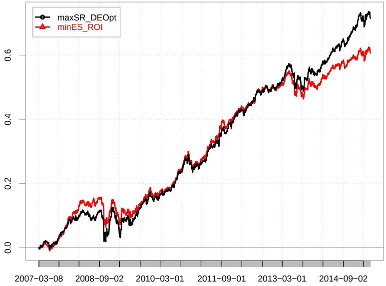
```
> minESROI_xts <-
+   plot_portf(portfolio=minESROI)
```



Sharpe minES CumReturns Plots

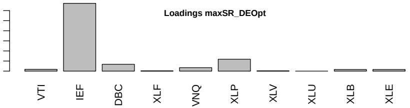
chart.CumReturns() plots the cumulative returns of a time series of returns.

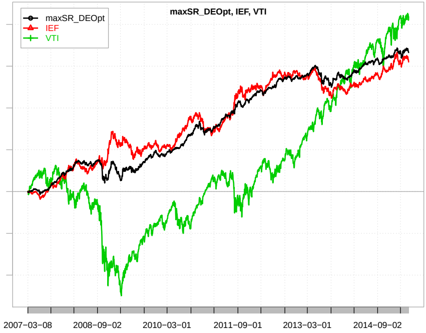
```
> chart.CumReturns(
+   cbind(maxSR_DEOpt_xts, minESROI_xts),
+   lwd=2, ylab="",
+   legend.loc="topleft", main="")
> rbind(maxSR_DEOpt$weights, minESROI$weights)
> c(maxSR_DEOpt$objective_measures$mean,
+ minESROI$objective_measures$mean)
> c(maxSR_DEOpt$objective_measures$StdDev[[1]],
+ minESROI$objective_measures$ES[[1]])
```



Out-of-sample Portfolios

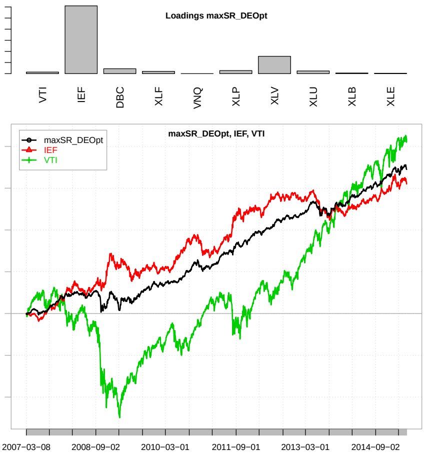
```
> # Perform optimization of weights
> maxSR_DEOpt <- optimize.portfolio(
+   R=rutils::etfenv$returns["/2011", portf_names],
+   portfolio=portf_maxSR,  # Specify portfolio
+   optimize_method="DEoptim", # Use DEoptim
+   maxSR=TRUE,  # Maximize Sharpe
+   trace=TRUE, traceDE=0)
> weights1h <- maxSR_DEOpt$weights
>
> # Plot optimization
> maxSR_DEOpt_xts <-
+   plot_portf(portfolio=maxSR_DEOpt)
```





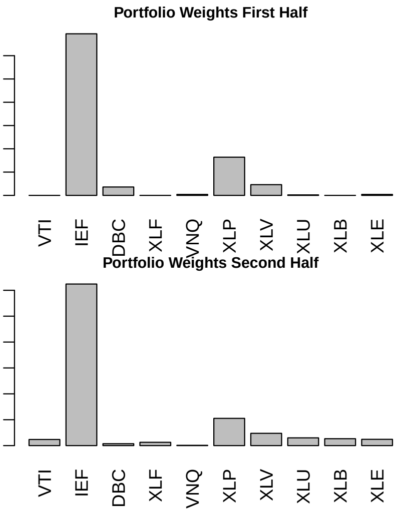
Out-of-sample Portfolios (cont.)

```
> # Perform optimization of weights
> maxSR_DEOpt <- optimize.portfolio(
+   R=rutils::etfenv$returns["2011/", portf_names],
+   portfolio=portf_maxSR,  # Specify portfolio
+   optimize_method="DEoptim", # Use DEoptim
+   maxSR=TRUE,  # Maximize Sharpe
+   trace=TRUE, traceDE=0)
> weights2h <- maxSR_DEOpt$weights
>
> # Plot optimization
> maxSR_DEOpt_xts <-
+   plot_portf(portfolio=maxSR_DEOpt)
```



Out-of-sample Portfolio Weights

```
> weights1h
> weights2h
> weights1h - weights2h
> barplot(weights1h,
+   names.arg=names(weights1h),
+   las=3, ylab="", xlab="",
+   main="Portfolio Weights First Half")
> barplot(weights2h,
+   names.arg=names(weights2h),
+   las=3, ylab="", xlab="",
+   main="Portfolio Weights Second Half")
```



Simulating Single-period Defaults

Consider a portfolio of credit assets (bonds or loans) over a single period of time.

At the end of the period, some of the assets default, while the rest don't.

The default probabilities are equal to p_i.

Individual defaults can be simulated by comparing the probabilities p_i with the uniform random numbers u_i.

Default occurs if u_i is less than the default probability p_i:

$$u_i < p_i$$

Simulations in R can be accelerated by pre-computing a vector of random numbers, instead of generatng them one at a time in a loop.

Vectors of random numbers allow using *vectorized* functions, instead of inefficient (slow) `for()` loops.

```
> # Calculate random default probabilities
> set.seed(1121)
> nassets <- 100
> defprobs <- runif(nassets, max=0.2)
> mean(defprobs)
> # Simulate number of defaults
> unifv <- runif(nassets)
> sum(unifv < defprobs)
> # Simulate average number of defaults using for() loop (inefficie
> nsimu <- 1000
> set.seed(1121)
> defaultv <- numeric(nsimu)
> for (i in 1:nsimu) {  # Perform loop
+   unifv <- runif(nassets)
+   defaultv[i] <- sum(unifv < defprobs)
+ }  # end for
> # Calculate average number of defaults
> mean(defaultv)
> # Simulate using vectorized functions (efficient way)
> set.seed(1121)
> unifm <- matrix(runif(nsimu*nassets), ncol=nsimu)
> defaultv <- colSums(unifm < defprobs)
> mean(defaultv)
> # Plot the distribution of defaults
> x11(width=6, height=5)
> plot(density(defaultv), main="Distribution of Defaults",
+      xlab="number of defaults", ylab="frequqncy")
> abline(v=mean(defaultv), lwd=3, col="red")
```

Asset Values and Default Thresholds

Defaults can also be simulated using normally distributed variables a_i called *asset values*, instead of the uniformly distributed variables u_i.

The asset values a_i are the *quantiles* corresponding to the uniform variables u_i: $a_i = \Phi^{-1}(u_i)$ (where $\Phi()$ is the cumulative *Standard Normal* distribution).

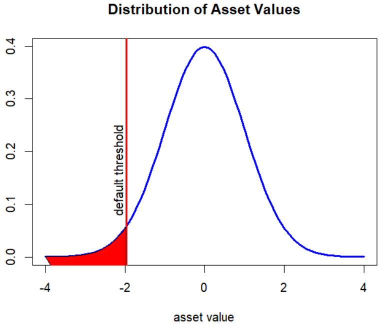
Similarly, the default probabilities p_i are also transformed into *default thresholds* t_i, which are the *quantiles*: $t_i = \Phi^{-1}(p_i)$.

Before, default occurred if u_i was less than the default probability p_i: $u_i < p_i$.

Now, default occurs if the *asset value* a_i is less than the *default threshold* t_i: $a_i < t_i$.

The asset values a_i are mathematical variables which can be negative, so they are not actual company asset values.

```
> # Calculate default thresholds and asset values
> defthresh <- qnorm(defprobs)
> assets <- qnorm(unifm)
> # Simulate defaults
> defaultv <- colSums(assets < defthresh)
> mean(defaultv)
```




```
> # Plot Standard Normal distribution
> x11(width=6, height=5)
> xlim <- 4; defthresh <- qnorm(0.025)
> curve(expr=dnorm(x), type="l", xlim=c(-xlim, xlim),
+ xlab="asset value", ylab="", lwd=3,
+ col="blue", main="Distribution of Asset Values")
> abline(v=defthresh, col="red", lwd=3)
> text(x=defthresh-0.1, y=0.15, labels="default threshold",
+  lwd=2, srt=90, pos=3)
> # Plot polygon area
> xvar <- seq(-xlim, xlim, length=100)
> yvar <- dnorm(xvar)
> intail <- ((xvar >= (-xlim)) & (xvar <= defthresh))
> polygon(c(xlim, xvar[intail], defthresh),
+   c(-1, yvar[intail], -1), col="red")
```

Vasicek Model of Correlated Asset Values

So far, the asset values are independent from each other, but in reality default events are correlated.

The *Vasicek* model introduces correlation between the asset values a_i.

Under the *Vasicek* single factor model, the asset value a_i is equal to the sum of a *systematic* factor s, plus an *idiosyncratic* factor z_i:

$$a_i = \sqrt{\rho}\, s + \sqrt{1 - \rho}\, z_i$$

Where ρ is the correlation between asset values.

The variables s, z_i, and a_i all follow the *Standard Normal* distribution $\phi(0, 1)$.

The *Vasicek* model resembles the *CAPM* model, with the asset value equal to the sum of a *systematic* factor plus an *idiosyncratic* factor.

The Bank for International Settlements (BIS) uses the *Vasicek* model as part of its regulatory capital requirements for bank credit risk:
http://bis2information.org/content/Vasicek_model
https://www.bis.org/bcbs/basel3.htm
https://www.bis.org/bcbs/irbriskweight.pdf

```
> # Define correlation parameters
> rho <- 0.2
> rho_sqrt <- sqrt(rho) ; rho_sqrtm <- sqrt(1-rho)
> nassets <- 5 ; nsimu <- 10000
> # Calculate vector of systematic and idiosyncratic factors
> sysv <- rnorm(nsimu)
> idiosyncv <- rnorm(nsimu*nassets)
> # Simulate asset values using vectorized functions (efficient way)
> assets <- rho_sqrt*sysv + rho_sqrtm*idiosyncv
> dim(assets) <- c(nsimu, nassets)
> # Asset values are standard normally distributed
> apply(assets, MARGIN=2, function(x) c(mean=mean(x), sd=sd(x)))
> # Calculate correlations between asset values
> cor(assets)
> # Simulate asset values using for() loop (inefficient way)
> # Allocate matrix of assets
> assets <- matrix(nr=nsimu, nc=nassets)
> # Simulate asset values using for() loop
> for (i in 1:nsimu) {  # Perform loop
+   assets[i, ] <- rho_sqrt*sysv[i] + rho_sqrtm*rnorm(nassets)
+ }  # end for
> cor(assets)
> # benchmark the speed of the two methods
> library(microbenchmark)
> summary(microbenchmark(
+   forloop={for (i in 1:nsimu) {
+     rho_sqrt*sysv[i] + rho_sqrtm*rnorm(nassets)}},
+   vectorized={rho_sqrt*sysv + rho_sqrtm*rnorm(nsimu*nassets)},
+   times=10))[, c(1, 4, 5)]
```

Vasicek Model of Correlated Defaults

Under the *Vasicek* model, default occurs if the *asset value* a_i is less than the *default threshold* t_i:

$$a_i = \sqrt{\rho} s + \sqrt{1 - \rho} z_i$$

$$a_i < t_i$$

The *systematic* factor s may be considered to represent the state of the macro economy, with positive values representing an economic expansion, and negative values representing an economic recession.

When the value of the *systematic* factor s is positive, then the asset values will all tend to be bigger as well, which will produce fewer defaults.

But when the *systematic* factor is negative, then the asset values will tend to be smaller, which will produce more defaults.

This way the *Vasicek* model introduces a correlation among defaults.

```
> # Calculate random default probabilities
> nassets <- 5
> defprobs <- runif(nassets, max=0.2)
> mean(defprobs)
> # Calculate default thresholds
> defthresh <- qnorm(defprobs)
> # Calculate number of defaults using vectorized functions (efficie
> # Calculate vector of number of defaults
> rowMeans(t(assets) < defthresh)
> defprobs
> # Calculate number of defaults using for() loop (inefficient way)
> # Allocate matrix of defaultm
> defaultm <- matrix(nr=nsimu, nc=nassets)
> # Simulate asset values using for() loop
> for (i in 1:nsimu) {  # Perform loop
+   defaultm[i, ] <- (assets[i, ] < defthresh)
+ }  # end for
> colSums(defaultm) / nsimu
> defprobs
> # Calculate correlations between defaults
> cor(defaultm)
```

Asset Correlation and Default Correlation

Default correlation is defined as the correlation between the `Boolean` vectors of default events.

The *Vasicek* model introduces correlation among default events, through the correlation of *asset values*.

If *asset values* have a positive correlation, then the defaults among credits are clustered together, and if one credit defaults then the other credits are more likely to default as well.

Empirical studies have found that the asset correlation ρ can vary between 5% to 20%, depending on the default risk.

Credits with higher default risk tend to also have higher asset correlation, since they are more sensitive to the economic conditions.

Default correlations are usually much lower than the corresponding asset correlations.

```
> # Define default probabilities
> nassets <- 2
> defprob <- 0.2
> defthresh <- qnorm(defprob)
> # Define correlation parameters
> rho <- 0.2
> rho_sqrt <- sqrt(rho) ; rho_sqrtm <- sqrt(1-rho)
> # Calculate vector of systematic factors
> nsimu <- 1000
> sysv <- rnorm(nsimu)
> # Simulate asset values using vectorized functions
> assets <- rho_sqrt*sysv + rho_sqrtm*rnorm(nsimu*nassets)
> dim(assets) <- c(nsimu, nassets)
> # Calculate number of defaults using vectorized functions
> defaultm <- t(t(assets) < defthresh)
> # Calculate correlations between defaults
> cor(defaultm)
> # Calculate average number of defaults and compare to defprob
> colSums(defaultm) / nsimu
> defprob
```

Cumulative Defaults Under the Vasicek Model

A formula for the default distribution under the Vasicek Model can be derived under the simplifying assumptions that the number of assets is very large and that they all have the same default probabilities $p_i = p$.

In that case the single default threshold is equal to $t = \Phi^{-1}(p)$.

If the systematic factor s is fixed, then the *asset value* a_i follows the *Normal* distribution with mean equal to $\sqrt{\rho}s$ and standard deviation equal to $\sqrt{1-\rho}$:

$$a_i = \sqrt{\rho}s + \sqrt{1-\rho}z_i$$

The conditional default probability $p(s)$, given the systematic factor s, is equal to:

$$p(s) = \Phi(\frac{t - \sqrt{\rho}s}{\sqrt{1-\rho}})$$

Since the systematic factor s is fixed, then the defaults are all independent with the same default probability $p(s)$.

Because the number of assets is very large, the percentage x of the portfolio that defaults, is equal to the conditional default probability $x = p(s)$.

We can invert the formula $x = \Phi(\frac{t-\sqrt{\rho}s}{\sqrt{1-\rho}})$ to obtain the systematic factor s:

$$s = \frac{\sqrt{1-\rho}\,\Phi^{-1}(x) - t}{\sqrt{\rho}}$$

Since the systematic factor s follows the *Standard Normal* distribution, then the portfolio cumulative default probability $P(x)$ is equal to:

$$P(x) = \Phi(\frac{\sqrt{1-\rho}\,\Phi^{-1}(x) - t}{\sqrt{\rho}})$$

Cumulative Default Distribution And Correlation

The cumulative portfolio default probability $P(x)$:

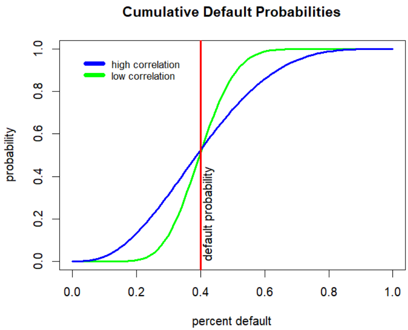
$$P(x) = \Phi\left(\frac{\sqrt{1-\rho}\,\Phi^{-1}(x) - t}{\sqrt{\rho}}\right)$$

Depends on the correlation parameter ρ.

If the correlation ρ is very low (close to 0) then the percentage x of the portfolio defaults is always very close to the default probability p, and the cumulative default probability curve is steep close to the expected value of p.

If the correlation ρ is very high (close to 1) then the percentage x of the portfolio defaults has a very wide dispersion around the default probability p, and the cumulative default probability curve is flat close to the expected value of p.

This is because with high correlation, the assets will tend to all default together or not default.




```
> # Define cumulative default distribution function
> cumdefdistr <- function(x, defthresh=(-2), rho=0.2)
+   pnorm((sqrt(1-rho)*qnorm(x) - defthresh)/sqrt(rho))
> cumdefdistr(x=0.2, defthresh=qnorm(defprob), rho=rho)
> # Plot cumulative default distribution function
> defprob <- 0.4; defthresh <- qnorm(defprob)
> curve(expr=cumdefdistr(x, defthresh=defthresh, rho=0.05),
+ xlim=c(0, 0.999), lwd=3, xlab="percent default", ylab="probabili
+ col="green", main="Cumulative Default Probabilities")
```

```
> # Plot default distribution with higher correlation
> curve(expr=cumdefdistr(x, defthresh=defthresh, rho=0.2),
+ xlim=c(0, 0.999), add=TRUE, lwd=3, col="blue", main="")
> # Add legend
> legend(x="topleft",
+  legend=c("high correlation", "low correlation"),
+  title=NULL, inset=0.05, cex=0.8, bg="white",
+  bty="n", lwd=6, lty=1, col=c("blue", "green"))
> # Add unconditional default probability
> abline(v=defprob, col="red", lwd=3)
> text(x=defprob, y=0.0, labels="default probability",
+  lwd=2, srt=90, pos=4)
```

Distribution of Defaults Under the Vasicek Model

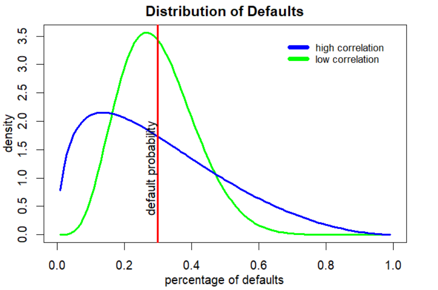
The probability density $f(x)$ of portfolio defaults is equal to the derivative of the cumulative default distribution $P(x)$:

$$f(x) = \frac{\sqrt{1-\rho}}{\sqrt{\rho}} \exp(-\frac{1}{2\rho}(\sqrt{1-\rho}\,\Phi^{-1}(x) - t)^2 + \frac{1}{2}\Phi^{-1}(x)^2)$$

If the correlation ρ is very low (close to 0) then the probability density $f(x)$ is centered around the default probability p.

If the correlation ρ is very high (close to 1) then the probability density $f(x)$ is wide, with significant probability of large portfolio defaults and also small portfolio defaults.

```
> # Define default probability density function
> defdistr <- function(x, defthresh=(-2), rho=0.2)
+   sqrt((1-rho)/rho)*exp(-(sqrt(1-rho)*qnorm(x) -
+   defthresh)^2/(2*rho) + qnorm(x)^2/2)
> # Define parameters
> rho <- 0.2 ; rho_sqrt <- sqrt(rho) ; rho_sqrtm <- sqrt(1-rho)
> defprob <- 0.3; defthresh <- qnorm(defprob)
> defdistr(0.03, defthresh=defthresh, rho=rho)
> # Plot probability distribution of defaults
> curve(expr=defdistr(x, defthresh=defthresh, rho=0.1),
+ xlim=c(0, 1.0), lwd=3,
+ xlab="Default percentage", ylab="Density",
+ col="green", main="Distribution of Defaults")
```




```
> # Plot default distribution with higher correlation
> curve(expr=defdistr(x, defthresh=defthresh, rho=0.3),
+ xlab="default percentage", ylab="",
+ add=TRUE, lwd=3, col="blue", main="")
> # Add legend
> legend(x="topright",
+  legend=c("high correlation", "low correlation"),
+  title=NULL, inset=0.05, cex=0.8, bg="white",
+  bty="n", lwd=6, lty=1, col=c("blue", "green"))
> # Add unconditional default probability
> abline(v=defprob, col="red", lwd=3)
> text(x=defprob, y=2, labels="default probability",
+  lwd=2, srt=90, pos=2)
```

Distribution of Defaults Under Extreme Correlations

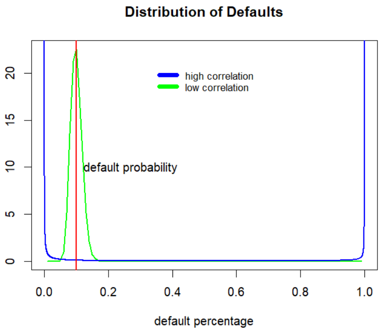
If the correlation ρ is close to 0, then the asset values a_i are independent from each other, and defaults are also independent, so that the percentage of portfolio defaults is very close to the default probability p.

In that case, the probability density of portfolio defaults is very narrow and is centered on the default probability p.

If the correlation ρ is close to 1, then the asset values a_i are almost the same, and defaults occur at the same time, so that the percentage of portfolio defaults is either 0 or 1.

In that case, the probability density of portfolio defaults becomes *bimodal*, with two peaks around *zero* and 1.

```
> # Plot default distribution with low correlation
> curve(expr=defdistr(x, defthresh=defthresh, rho=0.01),
+ xlab="default percentage", ylab="", lwd=2,
+ col="green", main="Distribution of Defaults")
> # Plot default distribution with high correlation
> curve(expr=defdistr(x, defthresh=defthresh, rho=0.99),
+ xlab="percentage of defaults", ylab="density",
+ add=TRUE, lwd=2, n=10001, col="blue", main="")
```




```
> # Add legend
> legend(x="top",
+  legend=c("high correlation", "low correlation"),
+  title=NULL, inset=0.1, cex=0.8, bg="white",
+  bty="n", lwd=6, lty=1, col=c("blue", "green"))
> # Add unconditional default probability
> abline(v=0.1, col="red", lwd=2)
> text(x=0.1, y=10, lwd=2, pos=4,
+  labels="default probability")
```

Numerical Integration of Functions

The function `integrate()` performs numerical integration of a function of a single variable, i.e. it calculates a definite integral over an integration interval.

Additional parameters can be passed to the integrated function through the dots `"..."` argument of the function `integrate()`.

The function `integrate()` accepts the integration limits `-Inf` and `Inf` equal to minus and plus infinity.

```
> # Get help for integrate()
> ?integrate
> # Calculate slowly converging integral
> func <- function(x) {1/((x+1)*sqrt(x))}
> integrate(func, lower=0, upper=10)
> integrate(func, lower=0, upper=Inf)
> # Integrate function with parameter lambda
> func <- function(x, lambda=1) {
+   exp(-x*lambda)
+ }  # end func
> integrate(func, lower=0, upper=Inf)
> integrate(func, lower=0, upper=Inf, lambda=2)
> # Cumulative probability over normal distribution
> pnorm(-2)
> integrate(dnorm, low=2, up=Inf)
> str(dnorm)
> pnorm(-1)
> integrate(dnorm, low=2, up=Inf, mean=1)
> # Expected value over normal distribution
> integrate(function(x) x*dnorm(x), low=2, up=Inf)
```

Portfolio Loss Distribution

The expected loss (EL) of a credit portfolio is equal to the sum of the default probabilities p_i multiplied by the loss given default LGD (aka the *loss severity* - equal to 1 minus the *recovery rate*):

$$EL = \sum_{i=1}^{n} p_i LGD_i$$

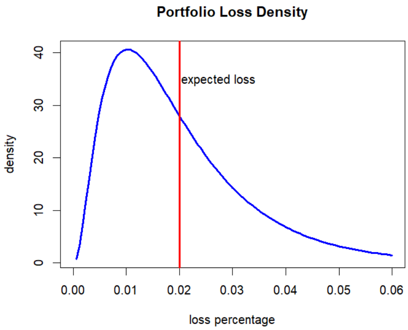
Then the *cumulative loss distribution* is equal to:

$$P(x) = \Phi\Big(\frac{\sqrt{1-\rho}\,\Phi^{-1}(\frac{x}{LGD}) - t}{\sqrt{\rho}}\Big)$$

And the *default distribution* is the derivative, and is equal to:

$$f(x) = \frac{\sqrt{1-\rho}}{LGD\sqrt{\rho}} \exp\Big(-\frac{1}{2\rho}\big(\sqrt{1-\rho}\Phi^{-1}(\frac{x}{LGD}) - t\big)^2 + \frac{1}{2}\Phi^{-1}(\frac{x}{LGD})\Big)^2$$

```
> # Vasicek model parameters
> rho <- 0.1; lgd <- 0.4
> defprob <- 0.05; defthresh <- qnorm(defprob)
> # Define Vasicek cumulative loss distribution
> cumlossdistr <- function(x, defthresh=(-2), rho=0.2, lgd=0.4)
+   pnorm((sqrt(1-rho)*qnorm(x/lgd) - defthresh)/sqrt(rho))
> # Define Vasicek loss distribution function
> lossdistr <- function(x, defthresh=(-2), rho=0.2, lgd=0.4)
+   sqrt((1-rho)/rho)*exp(-(sqrt(1-rho)*qnorm(x/lgd) - defthresh)^2/(2*rho) + qnorm(x/lgd)^2/2)/lgd
```




```
> # Plot probability distribution of losses
> x11(width=6, height=5)
> curve(expr=lossdistr(x, defthresh=defthresh, rho=rho),
+ cex.main=1.8, cex.lab=1.8, cex.axis=1.5,
+ type="l", xlim=c(0, 0.06),
+ xlab="loss percentage", ylab="density", lwd=3,
+ col="blue", main="Portfolio Loss Density")
> # Add line for expected loss
> abline(v=lgd*defprob, col="red", lwd=3)
> text(x=lgd*defprob-0.001, y=35, labels="expected loss", lwd=3, pos
```

Collateralized Debt Obligations (*CDOs*)

Collateralized Debt Obligations (cash *CDOs*) are securities (bonds) collateralized by other debt assets.

The *CDO* assets can be debt instruments like bonds, loans, and mortgages.

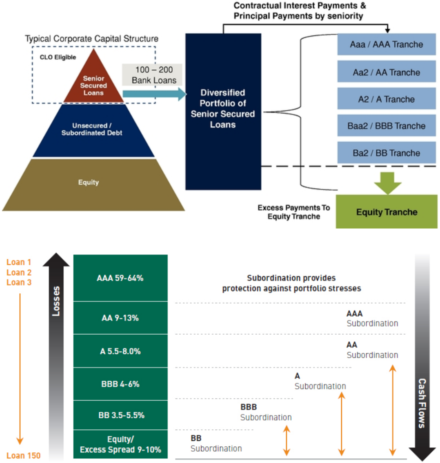
The *CDO* liabilities are *CDO* tranches, which receive cashflows from the *CDO* assets, and are exposed to their defaults.

CDO tranches have an attachment point (subordination, i.e. the percentage of asset default losses at which the tranche starts absorbing those losses), and a detachment point when the tranche is wiped out (suffers 100% losses).

The *equity tranche* is the most junior tranche, and is the first to absorb default losses.

The *mezzanine tranches* are senior to the *equity tranche* and absorb losses ony after the *equity tranche* is wiped out.

The *senior tranche* is the most senior tranche, and is the last to absorb losses.



CDO Tranche Losses

Single-tranche (synthetic) *CDOs* are credit default swaps which reference credit portfolios.

The expected loss *EL* on a *CDO* tranche is:

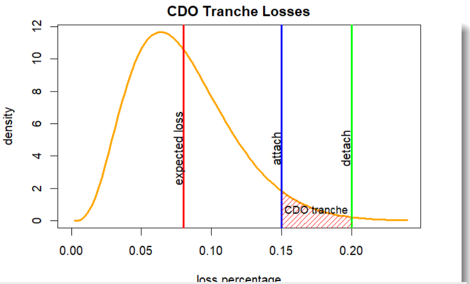
$$EL = \frac{1}{d-a} \int_a^d (x-a)\, f(x)\, \mathrm{d}x + \int_d^{LGD} f(x)\, \mathrm{d}x$$

Where $f(x)$ is the density of portfolio losses, and a and d are the tranche attachment (subordination) and detachment points.

The difference $(d-a)$ is the tranche *thickness*, so that *EL* is the expected loss as a percentage of the tranche notional.

A single-tranche *CDO* can be thought of as a short option spread on the asset defaults, struck at the attachment and detachment points.

```
> # Define Vasicek cumulative loss distribution
> cumlossdistr <- function(x, defthresh=(-2), rho=0.2, lgd=0.4)
+   pnorm((sqrt(1-rho)*qnorm(x/lgd) - defthresh)/sqrt(rho))
> # Define Vasicek loss distribution function
> # (vectorized version with error handling for x)
> lossdistr <- function(x, defthresh=(-2), rho=0.1, lgd=0.4) {
+   qnormv <- ifelse(x/lgd < 0.999, qnorm(x/lgd), 3.1)
+   sqrt((1-rho)/rho)*exp(-(sqrt(1-rho)*qnormv - defthresh)^2/(2*rl
+ }  # end lossdistr
```



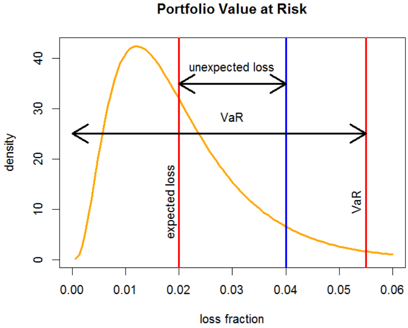

```
> defprob <- 0.2; defthresh <- qnorm(defprob)
> rho <- 0.1; lgd <- 0.4
> attachp <- 0.15; detachp <- 0.2
> # Expected tranche loss is sum of two terms
> tranchel <-
+   # Loss between attachp and detachp
+   integrate(function(x, attachp) (x-attachp)*lossdistr(x,
+ defthresh=defthresh, rho=rho, lgd=lgd),
+ low=attachp, up=detachp, attachp=attachp)$value / (detachp-attachp
+   # Loss in excess of detachp
+   (1-cumlossdistr(x=detachp, defthresh=defthresh, rho=rho, lgd=lgd
> # Plot probability distribution of losses
> curve(expr=lossdistr(x, defthresh=defthresh, rho=rho),
+ cex.main=1.8, cex.lab=1.8, cex.axis=1.5,
+ type="l", xlim=c(0, 3*lgd*defprob),
+ xlab="loss percentage", ylab="density", lwd=3,
+ col="orange", main="CDO Tranche Losses")
> # Add line for expected loss
> abline(v=lgd*defprob, col="red", lwd=3)
> text(x=lgd*defprob-0.001, y=4, labels="expected loss",
+ lwd=2, srt=90, pos=3, cex=1.8)
```

Portfolio Value at Risk

Value at Risk (*VaR*) measures extreme portfolio loss (but not the worst possible loss), defined as the *quantile* of the loss distribution, corresponding to a given confidence level α.

A loss exceeding the *EL* is called the Unexpected Loss (*UL*), and can be calculated from the *portfolio loss distribution*.

```
> # Add lines for unexpected loss
> abline(v=0.04, col="blue", lwd=3)
> arrows(x0=0.02, y0=35, x1=0.04, y1=35, code=3, lwd=3, cex=0.5)
> text(x=0.03, y=36, labels="unexpected loss", lwd=2, pos=3)
> # Add lines for VaR
> abline(v=0.055, col="red", lwd=3)
> arrows(x0=0.0, y0=25, x1=0.055, y1=25, code=3, lwd=3, cex=0.5)
> text(x=0.03, y=26, labels="VaR", lwd=2, pos=3)
> text(x=0.055-0.001, y=10, labels="VaR", lwd=2, srt=90, pos=3)
```

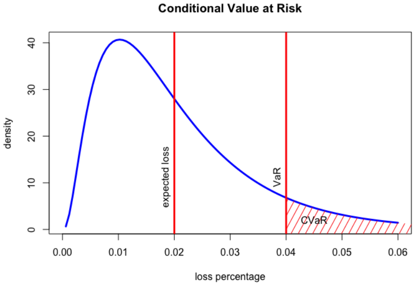


Conditional Value at Risk

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

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

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




```
> varisk <- 0.04; varmax <- 4*lgd*defprob
> # Calculate CVaR
> cvar <- integrate(function(x) x*lossdistr(x, defthresh=defthresh,
+   low=varisk, up=lgd)$value
> cvar <- cvar/integrate(lossdistr, low=varisk, up=lgd, defthresh=de
> # Plot probability distribution of losses
> curve(expr=lossdistr(x, defthresh=defthresh, rho=rho),
+ type="l", xlim=c(0, 0.06),
+ xlab="loss percentage", ylab="density", lwd=3,
+ col="blue", main="Conditional Value at Risk")
> # Add line for expected loss
> abline(v=lgd*defprob, col="red", lwd=3)
> text(x=lgd*defprob-0.001, y=10, labels="expected loss", lwd=2, s
```

```
> # Add lines for VaR
> abline(v=varisk, col="red", lwd=3)
> text(x=varisk-0.001, y=10, labels="VaR",
+  lwd=2, srt=90, pos=3)
> # Add shading for CVaR
> vars <- seq(varisk, varmax, length=100)
> densv <- sapply(vars, lossdistr,
+   defthresh=defthresh, rho=rho)
> # Draw shaded polygon
> polygon(c(varisk, vars, varmax), density=20,
+   c(-1, densv, -1), col="red", border=NA)
> text(x=varisk+0.005, y=0, labels="CVaR", lwd=2, pos=3)
```

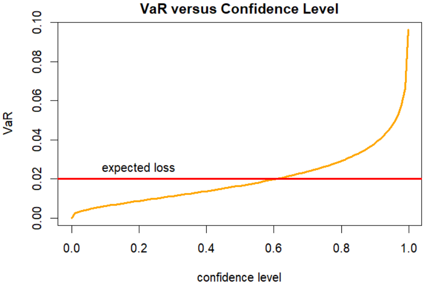
Value at Risk Under the Vasicek Model

Value at Risk (VaR) measures extreme portfolio loss (but not the worst possible loss), defined as the *quantile* of the loss distribution, corresponding to a given confidence level α.

The *quantile* of the loss distribution (the VaR), for a given a confidence level α, is given by the inverse of the cumulative loss distribution:

$$VaR(\alpha) = LGD \cdot \Phi\left(\frac{\sqrt{\rho}\Phi^{-1}(\alpha) + t}{\sqrt{1-\rho}}\right)$$

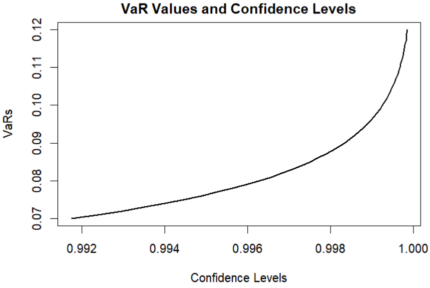
```
> # VaR (quantile of the loss distribution)
> varfun <- function(x, defthresh=qnorm(0.1), rho=0.1, lgd=0.4)
+   lgd*pnorm((sqrt(rho)*qnorm(x) + defthresh)/sqrt(1-rho))
> varfun(x=0.99, defthresh=defthresh, rho=rho, lgd=lgd)
> # Plot VaR
> curve(expr=varfun(x, defthresh=defthresh, rho=rho, lgd=lgd),
+ type="l", xlim=c(0, 0.999), xlab="confidence level", ylab="VaR", lwd=3,
+ col="orange", main="VaR versus Confidence Level")
> # Add line for expected loss
> abline(h=lgd*defprob, col="red", lwd=3)
> text(x=0.2, y=lgd*defprob, labels="expected loss", lwd=2, pos=3)
```



Value at Risk and Confidence Levels

The confidence levels of *VaR* values can also be calculated by integrating over the tail of the loss density function.

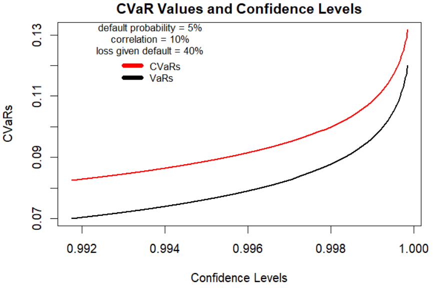
```
> # Integrate lossdistr() over full range
> integrate(lossdistr, low=0.0, up=lgd,
+     defthresh=defthresh, rho=rho, lgd=lgd)
> # Calculate expected losses using lossdistr()
> integrate(function(x) x*lossdistr(x, defthresh=defthresh, rho=rho,
+     low=0.0, up=lgd)
> # Calculate confidence levels corresponding to VaR values
> vars <- seq(0.07, 0.12, 0.001)
> confls <- sapply(vars, function(varisk) {
+   integrate(lossdistr, low=varisk, up=lgd, defthresh=defthresh, rh
+ })  # end sapply
> confls <- cbind(as.numeric(t(confls)[, 1]), vars)
> colnames(confls) <- c("levels", "VaRs")
> # Calculate 95% confidence level VaR value
> confls[match(TRUE, confls[, "levels"] < 0.05), "VaRs"]
> plot(x=1-confls[, "levels"],
+      y=confls[, "VaRs"], lwd=2,
+      xlab="confidence level", ylab="VaRs",
+      t="l", main="VaR Values and Confidence Levels")
```



Conditional Value at Risk Under the Vasicek Model

The *CVaR* values can be calculated by integrating over the tail of the loss density function.

```
> # Calculate CVaR values
> cvars <- sapply(vars, function(varisk) {
+   integrate(function(x) x*lossdistr(x, defthresh=defthresh, rho=rl
+       low=varisk, up=lgd)})  # end sapply
> confls <- cbind(confls, as.numeric(t(cvars)[, 1]))
> colnames(confls)[3] <- "CVaRs"
> # Divide CVaR by confidence level
> confls[, "CVaRs"] <- confls[, "CVaRs"]/confls[, "levels"]
> # Calculate 95% confidence level CVaR value
> confls[match(TRUE, confls[, "levels"] < 0.05), "CVaRs"]
> # Plot CVaRs
> plot(x=1-confls[, "levels"], y=confls[, "CVaRs"],
+      t="l", col="red", lwd=2,
+      ylim=range(confls[, c("VaRs", "CVaRs")]),
+      xlab="confidence level", ylab="CVaRs",
+      main="CVaR Values and Confidence Levels")
```




```
> # Add VaRs
> lines(x=1-confls[, "levels"], y=confls[, "VaRs"], lwd=2)
> # Add legend
> legend(x="topleft", legend=c("CVaRs", "VaRs"),
+  title="default probability = 5%
+ correlation = 10%
+ loss given default = 40%",
+  inset=0.1, cex=0.8, bg="white", bty="n",
+  lwd=6, lty=1, col=c("red", "black"))
```

Simulating Portfolio Losses Under the Vasicek Model

If the default probabilities p_i are not all the same, then there's no formula for the *portfolio loss distribution* under the Vasicek Model.

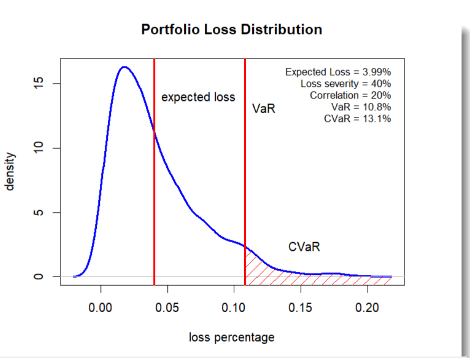
In that case the portfolio losses and *VaR* must be simulated.

```
> # Define model parameters
> nassets <- 300; nsimu <- 1000; lgd <- 0.4
> # Define correlation parameters
> rho <- 0.2; rho_sqrt <- sqrt(rho); rho_sqrtm <- sqrt(1-rho)
> # Calculate default probabilities and thresholds
> set.seed(1121)
> defprobs <- runif(nassets, max=0.2)
> defthresh <- qnorm(defprobs)
> # Simulate losses under Vasicek model
> sysv <- rnorm(nsimu)
> assets <- matrix(rnorm(nsimu*nassets), ncol=nsimu)
> assets <- t(rho_sqrt*sysv + t(rho_sqrtm*assets))
> losses <- lgd*colSums(assets < defthresh)/nassets
```

VaR and *CVaR* Under the Vasicek Model

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 VaR from confidence level
> confl <- 0.95
> varisk <- quantile(losses, confl)
> # Calculate the CVaR as the mean losses in excess of VaR
> cvar <- mean(losses[losses > varisk])
> # Plot the density of portfolio losses
> x11(width=6, height=5)
> densv <- density(losses, from=0)
> plot(densv, xlab="loss percentage", ylab="density",
+      cex.main=1.8, cex.lab=1.8, cex.axis=1.5,
+      lwd=3, col="blue", main="Portfolio Loss Distribution")
> # Add vertical line for expected loss
> exploss <- lgd*mean(defprobs)
> abline(v=exploss, col="red", lwd=3)
> xmax <- max(densv$x); ymax <- max(densv$y)
> text(x=exploss, y=(6*ymax/7), labels="expected loss",
+      lwd=2, pos=4, cex=1.8)
> # Add vertical line for VaR
> abline(v=varisk, col="red", lwd=3)
> text(x=varisk, y=4*ymax/5, labels="VaR", lwd=2, pos=4, cex=1.8)
```

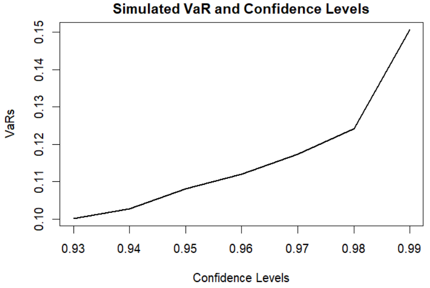



```
> # Draw shaded polygon for CVaR
> intail <- (densv$x > varisk)
> xvar <- c(min(densv$x[intail]), densv$x[intail], max(densv$x))
> polygon(xvar, c(-1, densv$y[intail], -1), col="red", border=NA, de
> # Add text for CVaR
> text(x=5*varisk/4, y=(ymax/7), labels="CVaR", lwd=2, pos=4, cex=1.
> # Add text with data
> text(xmax, ymax, labels=paste0(
+  "Expected Loss = ", format(100*exploss, digits=3), "%", "\n",
+  "Loss severity = ", format(100*lgd, digits=3), "%", "\n",
+  "Correlation = ", format(100*rho, digits=3), "%", "\n",
+  "VaR = ", format(100*varisk, digits=3), "%", "\n",
+  "CVaR = ", format(100*cvar, digits=3), "%"),
+      adj=c(1, 1), cex=1.8, lwd=2)
```

Simulating *VaR* Under the Vasicek Model

The *VaR* can be calculated from the simulated portfolio losses using the function `quantile()`.

The function `quantile()` calculates the sample quantiles. It uses interpolation to improve the accuracy. Information about the different interpolation methods can be found by typing `?quantile`.



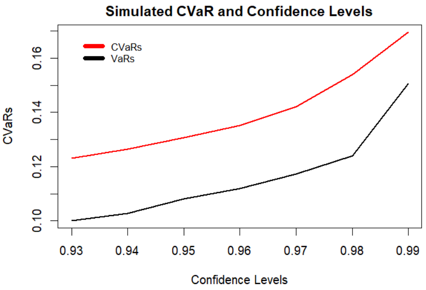

```
> # Calculate VaRs from confidence levels
> confls <- seq(0.93, 0.99, 0.01)
> vars <- quantile(losses, probs=confls)
> plot(x=confls, y=vars, t="l", lwd=2,
+      xlab="confidence level", ylab="VaRs",
+      main="Simulated VaR and Confidence Levels")
```

Simulating *CVaR* Under the Vasicek Model

The *CVaR* can be calculated from the frequency of tail losses in excess of the *VaR*.

The function table() calculates the frequency distribution of categorical data.

```
> # Calculate CVaRs
> cvars <- sapply(vars, function(varisk) {
+   mean(losses[losses >= varisk])
+ })  # end sapply
> cvars <- cbind(cvars, vars)
> # Alternative CVaR calculation using frequency table
> # first calculate frequency table of losses
> # tablev <- table(losses)/nsimu
> # Calculate CVaRs from frequency table
> # cvars <- sapply(vars, function(varisk) {
> #   tailrisk <- tablev[names(tablev) > varisk]
> #   tailrisk %*% as.numeric(names(tailrisk)) / sum(tailrisk)
> # })  # end sapply
> # Plot CVaRs
> plot(x=confls, y=cvars[, "cvars"],
+      t="l", col="red", lwd=2,
+      ylim=range(cvars),
+      xlab="confidence level", ylab="CVaRs",
+      main="Simulated CVaR and Confidence Levels")
```




```
> # Add VaRs
> lines(x=confls, y=cvars[, "vars"], lwd=2)
> # Add legend
> legend(x="topleft", legend=c("CVaRs", "VaRs"), bty="n",
+  title=NULL, inset=0.05, cex=0.8, bg="white",
+  lwd=6, lty=1, col=c("red", "black"))
```

Function for Simulating *VaR* Under the Vasicek Model

The function calc_var() simulates default losses under the *Vasicek* model, for a vector of confidence levels, and calculates a vector of *VaR* and *CVaR* values.

```
> calcvar <- function(defthresh, # Default thresholds
+                lgd=0.6, # loss given default
+                rho_sqrt, rho_sqrtm, # asset correlation
+                nsimu=1000, # number of simulations
+                confls=seq(0.93, 0.99, 0.01) # Confidence levels
+                ) {
+   # Define model parameters
+   nassets <- NROW(defthresh)
+   # Simulate losses under Vasicek model
+   sysv <- rnorm(nsimu)
+   assets <- matrix(rnorm(nsimu*nassets), ncol=nsimu)
+   assets <- t(rho_sqrt*sysv + t(rho_sqrtm*assets))
+   losses <- lgd*colSums(assets < defthresh)/nassets
+   # Calculate VaRs and CVaRs
+   vars <- quantile(losses, probs=confls)
+   cvars <- sapply(vars, function(varisk) {
+     mean(losses[losses >= varisk])
+   })  # end sapply
+   names(vars) <- confls
+   names(cvars) <- confls
+   c(vars, cvars)
+ }  # end calcvar
```

Standard Errors of *VaR* Using Bootstrap Simulation

The values of *VaR* and *CVaR* produced by the function `calc_var()` are subject to uncertainty because they're calculated from a simulation.

We can calculate the standard errors of *VaR* and *CVaR* by running the function `calc_var()` many times and repeating the simulation in a loop.

This bootstrap will only capture the uncertainty due to the finite number of trials in the simulation, but not due to the uncertainty of model parameters.

```
> # Define model parameters
> nassets <- 300; nsimu <- 1000; lgd <- 0.4
> rho <- 0.2; rho_sqrt <- sqrt(rho); rho_sqrtm <- sqrt(1-rho)
> # Calculate default probabilities and thresholds
> set.seed(1121)
> defprobs <- runif(nassets, max=0.2)
> defthresh <- qnorm(defprobs)
> # Define number of bootstrap simulations
> nboot <- 500
> # Perform bootstrap of calcvar
> set.seed(1121)
> bootd <- sapply(rep(lgd, nboot), calcvar,
+   defthresh=defthresh,
+   rho_sqrt=rho_sqrt, rho_sqrtm=rho_sqrtm,
+   nsimu=nsimu, confls=confls)  # end sapply
> bootd <- t(bootd)
> # Calculate vectors of standard errors of VaR and CVaR from bootd
> stderr_var <- apply(bootd[, 1:7], MARGIN=2,
+     function(x) c(mean=mean(x), sd=sd(x)))
> stderr_cvar <- apply(bootd[, 8:14], MARGIN=2,
+     function(x) c(mean=mean(x), sd=sd(x)))
> # Scale the standard errors of VaRs and CVaRs
> stderr_var[2, ] <- stderr_var[2, ]/stderr_var[1, ]
> stderr_cvar[2, ] <- stderr_cvar[2, ]/stderr_cvar[1, ]
```

Standard Errors of *VaR* at High Confidence Levels

The standard errors of *VaR* and *CVaR* are inversely proportional to square root of the number of loss events in the simulation that exceed the *VaR*.

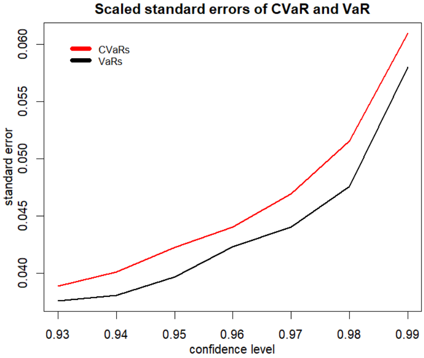
So the greater the number of loss events, the smaller the standard errors, and vice versa.

But as the confidence level increases, the *VaR* also increases, and the number of loss events decreases, causing larger standard errors.

So the as the confidence level increases, the standard errors of *VaR* and *CVaR* also increase.

The *scaled* (relative) standard errors of *VaR* and *CVaR* also increase with the confidence level, making them much less reliable at very high confidence levels.

The standard error of *CVaR* is even greater than that of *VaR*.

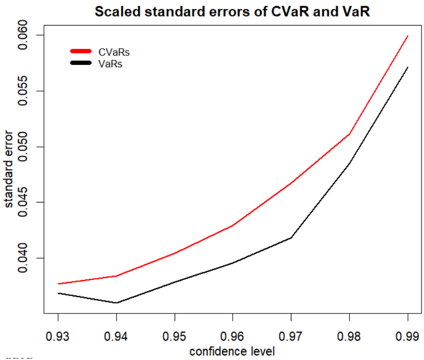



```
> # Plot the standard errors of VaRs and CVaRs
> x11(width=6, height=5)
> par(mar=c(3, 3, 2, 1), oma=c(0, 0, 0, 0), mgp=c(2, 1, 0))
> plot(x=colnames(stderr_cvar), y=stderr_cvar[2, ],
+   t="l", col="red", lwd=2,
+   ylim=range(c(stderr_var[2, ], stderr_cvar[2, ])),
+   xlab="confidence level", ylab="standard error",
+   main="Scaled standard errors of CVaR and VaR")
> lines(x=colnames(stderr_var), y=stderr_var[2, ], lwd=2)
> legend(x="topleft", legend=c("CVaRs", "VaRs"), bty="n",
+  title=NULL, inset=0.05, cex=0.8, bg="white",
+  lwd=6, lty=1, col=c("red", "black"))
```

Standard Errors of *VaR* Using Parallel Bootstrap

The *scaled* standard errors of *VaR* and *CVaR* increase with the confidence level, making them much less reliable at very high confidence levels.

```
> library(parallel)  # load package parallel
> ncores <- detectCores() - 1  # number of cores
> cluster <- makeCluster(ncores)  # Initialize compute cluster
> # Perform bootstrap of calcvar for Windows
> clusterSetRNGStream(cluster, 1121)
> bootd <- parLapply(cluster, rep(lgd, nboot),
+   fun=calcvar, defthresh=defthresh,
+   rho_sqrt=rho_sqrt, rho_sqrtm=rho_sqrtm,
+   nsimu=nsimu, confls=confls)  # end parLapply
> # Bootstrap under Mac-OSX or Linux
> bootd <- mclapply(rep(lgd, nboot),
+   FUN=calcvar, defthresh=defthresh,
+   rho_sqrt=rho_sqrt, rho_sqrtm=rho_sqrtm,
+   nsimu=nsimu, confls=confls)  # end mclapply
> bootd <- rutils::do_call(rbind, bootd)
> stopCluster(cluster)  # Stop R processes over cluster
> # Calculate vectors of standard errors of VaR and CVaR from bootd
> stderr_var <- apply(bootd[, 1:7], MARGIN=2,
+     function(x) c(mean=mean(x), sd=sd(x)))
> stderr_cvar <- apply(bootd[, 8:14], MARGIN=2,
+     function(x) c(mean=mean(x), sd=sd(x)))
> # Scale the standard errors of VaRs and CVaRs
> stderr_vars <- stderr_var[2, ]/stderr_var[1, ]
> stderr_cvars <- stderr_cvar[2, ]/stderr_cvar[1, ]
```




```
> # Plot the standard errors of VaRs and CVaRs
> x11(width=6, height=5)
> plot(x=colnames(stderr_cvar),
+   y=stderr_cvars, t="l", col="red", lwd=2,
+   ylim=range(c(stderr_vars, stderr_cvars)),
+   xlab="confidence level", ylab="standard error",
+   main="Scaled standard errors of CVaR and VaR")
> lines(x=colnames(stderr_var), y=stderr_vars, lwd=2)
> legend(x="topleft", legend=c("CVaRs", "VaRs"), bty="n",
+  title=NULL, inset=0.05, cex=0.8, bg="white",
+  lwd=6, lty=1, col=c("red", "black"))
```

Vasicek Model With Uncertain Default Probabilities

The previous bootstrap only captured the uncertainty due to the finite simulation trials, but not due to the uncertainty of model parameters, such as the default probabilities and correlations.

The below function calc_var() can simulate the *Vasicek* model with uncertain default probabilities.

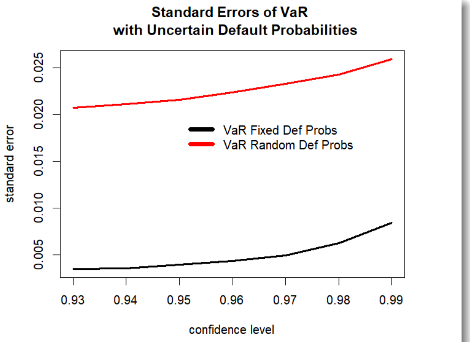
```
> calcvar <- function(defprobs, # Default probabilities
+                lgd=0.6, # loss given default
+                rho_sqrt, rho_sqrtm, # asset correlation
+                nsimu=1000, # number of simulations
+                confls=seq(0.93, 0.99, 0.01) # Confidence levels
+                ) {
+   # Calculate random default thresholds
+   defthresh <- qnorm(runif(1, min=0.5, max=1.5)*defprobs)
+   # Simulate losses under Vasicek model
+   nassets <- NROW(defprobs)
+   sysv <- rnorm(nsimu)
+   assets <- matrix(rnorm(nsimu*nassets), ncol=nsimu)
+   assets <- t(rho_sqrt*sysv + t(rho_sqrtm*assets))
+   losses <- lgd*colSums(assets < defthresh)/nassets
+   # Calculate VaRs and CVaRs
+   vars <- quantile(losses, probs=confls)
+   cvars <- sapply(vars, function(varisk) {
+     mean(losses[losses >= varisk])
+   })  # end sapply
+   names(vars) <- confls
+   names(cvars) <- confls
+   c(vars, cvars)
+ }  # end calcvar
```

Standard Errors Due to Uncertain Default Probabilities

The greatest contribution to the standard errors of *VaR* and *CVaR* is from the uncertainty of model parameters, such as the default probabilities, correlations, and loss severities.

For example, a 50% uncertainty in the default probabilities can produce a 20% uncertainty of the *VaR*.

```
> library(parallel)  # load package parallel
> ncores <- detectCores() - 1  # number of cores
> cluster <- makeCluster(ncores)  # Initialize compute cluster
> # Perform bootstrap of calcvar for Windows
> clusterSetRNGStream(cluster, 1121)
> bootd <- parLapply(cluster, rep(lgd, nboot),
+   fun=calcvar, defprobs=defprobs,
+   rho_sqrt=rho_sqrt, rho_sqrtm=rho_sqrtm,
+   nsimu=nsimu, confls=confls)  # end parLapply
> # Bootstrap under Mac-OSX or Linux
> bootd <- mclapply(rep(lgd, nboot),
+   FUN=calcvar, defprobs=defprobs,
+   rho_sqrt=rho_sqrt, rho_sqrtm=rho_sqrtm,
+   nsimu=nsimu, confls=confls)  # end mclapply
> bootd <- rutils::do_call(rbind, bootd)
> stopCluster(cluster)  # Stop R processes over cluster
> # Calculate vectors of standard errors of VaR and CVaR from boot
> stderr_var_param <- apply(bootd[, 1:7], MARGIN=2,
+     function(x) c(mean=mean(x), sd=sd(x)))
> stderr_cvar_param <- apply(bootd[, 8:14], MARGIN=2,
+     function(x) c(mean=mean(x), sd=sd(x)))
```

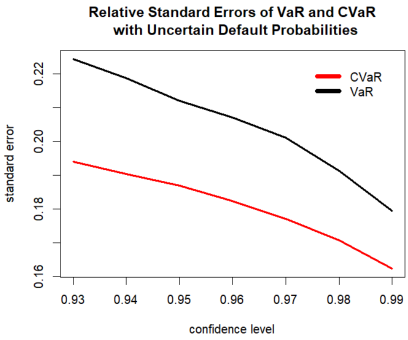



```
> # Plot the standard errors of VaRs under uncertain default probab
> x11(width=6, height=5)
> plot(x=colnames(stderr_var),
+   y=stderr_var[2, ], t="l", lwd=3,
+   ylim=range(c(stderr_var[2, ], stderr_var_param[2, ])),
+   xlab="confidence level", ylab="standard error",
+   main="Standard Errors of VaR
+   with Uncertain Default Probabilities")
> lines(x=colnames(stderr_var), y=stderr_var_param[2, ],
+ col="red", lwd=3)
> legend(x=0.95, y=0.02, bty="n",
+  legend=c("VaR Fixed Def Probs", "VaR Random Def Probs"),
+  title=NULL, inset=0.05, cex=1.0, bg="white",
+  lwd=6, lty=1, col=c("black", "red"))
```

Relative Errors Due to Uncertain Default Probabilities

The *scaled* (relative) standard errors of *VaR* and *CVaR* under uncertain default probabilities decrease with higher confidence level, because the standard errors are less dependent on the confidence level and don't increase as fast as the *VaR* does.

```
> # Scale the standard errors of VaRs and CVaRs
> stderr_vars <- stderr_var_param[2, ]/
+   stderr_var_param[1, ]
> stderr_cvars <- stderr_cvar_param[2, ]/
+   stderr_cvar_param[1, ]
```




```
> # Plot the standard errors of VaRs and CVaRs
> x11(width=6, height=5)
> plot(x=colnames(stderr_cvar_param),
+   y=stderr_cvars, t="l", col="red", lwd=3,
+   ylim=range(c(stderr_vars, stderr_cvars)),
+   xlab="confidence level", ylab="standard error",
+   main="Relative Standard Errors of VaR and CVaR
+   with Uncertain Default Probabilities")
> lines(x=names(stderr_vars), y=stderr_vars, lwd=3)
> legend(x="topright", legend=c("CVaR", "VaR"), bty="n",
+  title=NULL, inset=0.05, cex=1.0, bg="white",
+  lwd=6, lty=1, col=c("red", "black"))
```

Model Risk of Credit Portfolio Models

Credit portfolio models are subject to very significant *model risk* due to the uncertainties of model parameters, such as the default probabilities, correlations, and loss severities.

Model risk is the risk of incorrect model predictions due to incorrect model specification, and due to incorrect model parameters.

Jon Danielsson at the London School of Economics (LSE) has studied the model risk of *VaR* and *CVaR* in: Why Risk is So Hard to Measure, and in Model Risk of Risk Models.

Jon Danielsson has pointed out that there's not enough historical data to be able to accurately calculate the credit model parameters.

Jon Danielsson and Chen Zhou have demonstrated that accurately estimating *CVaR* at 5% confidence would require decades of price history, something that simply doesn't exist for many assets.

draft: Simulating the Vasicek Model Using Importance Sampling

The simulation estimates of *VaR* and *CVaR* have large standard errors because default events are rare, so the number of events which contribute to their estimates is therefore small.

The *variance* of an estimate produced by simulation decreases with the number of events which contribute to the estimate: $\sigma^2 \propto \frac{1}{n}$.

Importance sampling with probability tilting can be applied to reduce the standard errors of *VaR* and *CVaR*.

The exponential probability tilting can be applied to the *systematic* factor s:

$$\Phi(s, \lambda) = \exp(s\lambda - \lambda^2/2) \cdot \Phi(s, \lambda = 0)$$

Where λ is the tilt parameter.

The simulation outputs are then multiplied by the weights to compensate for the probability tilting:

$$w_x = \exp(-x\lambda + \lambda^2/2)$$

```
> # Define model parameters
> nassets <- 300; nsimu <- 1000; lgd <- 0.4
> # Define correlation parameters
> rho <- 0.2; rho_sqrt <- sqrt(rho); rho_sqrtm <- sqrt(1-rho)
> # Calculate default probabilities and thresholds
> set.seed(1121)
> defprobs <- runif(nassets, max=0.2)
> defthresh <- qnorm(defprobs)
> # Calculate vector of systematic factors
> sysv <- rnorm(nsimu)
> # Calculate vector of idiosyncratic factors
> idiosyncv <-
+   matrix(rnorm(nsimu*nassets), ncol=nsimu)
> # Simulate losses under Vasicek model
> assets <-
+   t(rho_sqrt*sysv + t(rho_sqrtm*idiosyncv))
> losses <-
+   lgd*colSums(assets < defthresh)/nassets
> # Calculate VaRs
> confls <- seq(0.93, 0.99, 0.01)
> vars <- quantile(losses, probs=confls)
>
> # Importance sampling losses
> lambda <- 3
> assets <-
+   t(rho_sqrt*sysv + t(rho_sqrtm*idiosyncv))
>
> cond_thresh <- outer(rho_sqrtm*defthresh, -rho_sqrt*sysv, FUN="+")
>
> cond_probs <- pnorm(cond_thresh)
>
> tilt_probs <- lambda*cond_probs/(1 + cond_probs*(lambda - 1))
>
> weightv <- (1 + tilt_probs*(lambda - 1))/lambda
>
> tilt_thresh <- qnorm(tilt_probs)
>
> losses <-
```

draft: Standard Errors of *VaR* Using Bootstrap Simulation

The values of *VaR* and *CVaR* produced by the function `calc_var()` are subject to uncertainty because they're calculated from a simulation.

We can calculate the standard errors of *VaR* and *CVaR* by running the function `calc_var()` many times and repeating the simulation in a loop.

This bootstrap will only capture the uncertainty due to the finite number of trials in the simulation, but not due to the uncertainty of model parameters.

```
> # Define model parameters
> nassets <- 300; nsimu <- 1000; lgd <- 0.4
> rho <- 0.2; rho_sqrt <- sqrt(rho); rho_sqrtm <- sqrt(1-rho)
> # Calculate default probabilities and thresholds
> set.seed(1121)
> defprobs <- runif(nassets, max=0.2)
> defthresh <- qnorm(defprobs)
> # Define number of bootstrap simulations
> nboot <- 500
> # Perform bootstrap of calcvar
> set.seed(1121)
> bootd <- sapply(rep(lgd, nboot),
+   calcvar,
+   defthresh=defthresh,
+   rho_sqrt=rho_sqrt, rho_sqrtm=rho_sqrtm,
+   nsimu=nsimu, confls=confls)  # end sapply
> bootd <- t(bootd)
> # Calculate vectors of standard errors of VaR and CVaR from bootd
> stderr_var <- apply(bootd[, 1:7], MARGIN=2,
+     function(x) c(mean=mean(x), sd=sd(x)))
> stderr_cvar <- apply(bootd[, 8:14], MARGIN=2,
+     function(x) c(mean=mean(x), sd=sd(x)))
> # Scale the standard errors of VaRs and CVaRs
> stderr_var[2, ] <- stderr_var[2, ]/stderr_var[1, ]
> stderr_cvar[2, ] <- stderr_cvar[2, ]/stderr_cvar[1, ]
```