FRE7241 Algorithmic Portfolio Management

Lecture#7, Spring 2018

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

March 24, 2018

NYU | TANDON SCHOOL OF ENGINEERING

The *Minimum Variance* Portfolio

If $\mathbb{C}$ is equal to the covariance matrix of returns, then the portfolio variance is equal to:

$$w^T \mathbb{C}\, w$$

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

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

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

Where λ is a *Lagrange multiplier*,

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

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

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

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

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

By setting the derivative to zero we find w equal to:

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

By multiplying the above from the left by $\mathbb{1}^T$, and using $w^T \mathbb{1} = 1$, 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:

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

Variance of the *Minimum Variance* Portfolio

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

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

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 i
> cov_mat <- cov(re_turns)
> cov_inv <- solve(a=cov_mat)
> u_nit <- rep(1, NCOL(cov_mat))
> # minimum variance weights with constraint
> # weight_s <- solve(a=cov_mat, b=u_nit)
> weight_s <- cov_inv %*% u_nit
> weight_s <- weight_s / drop(t(u_nit) %*% weigh
> # minimum variance
> t(weight_s) %*% cov_mat %*% weight_s
> 1/(t(u_nit) %*% cov_inv %*% u_nit)
```

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 w_i is equal to 1: $w^T \mathbb{1} = \sum_{i=1}^{n} w_i = 1$, and the mean portfolio return is equal to the target return r_t: $w^T \mathbb{r} = \sum_{i=1}^{n} w_i r_i = r_t$,

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

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

Where λ_1 and λ_2 are the *Lagrange multipliers*,

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

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

By setting the derivative to zero we obtain the *efficient portfolio* weights w,

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

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

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

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

The above can be written in matrix notation as:

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

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 $a = \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$, $b = \mathbb{1}^T \mathbb{C}^{-1} \mathbb{r}$, $c = \mathbb{r}^T \mathbb{C}^{-1} \mathbb{r}$, $\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$, $u = \begin{bmatrix} 1 \\ r_t \end{bmatrix}$, 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 w can now be solved as:

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

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

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

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

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
> mean_rets <- colMeans(re_turns)
> # products of inverse with mean returns and un
> f_mat <- matrix(c(
+   t(u_nit) %*% cov_inv %*% u_nit,
+   t(u_nit) %*% cov_inv %*% mean_rets,
+   t(mean_rets) %*% cov_inv %*% u_nit,
+   t(mean_rets) %*% cov_inv %*% mean_rets), nc=
> # solve for the Lagrange multipliers
> multipli_ers <-
+   solve(a=f_mat, b=c(2, 2*re_turn))
> # calculate weights
> weight_s <- drop(0.5*cov_inv %*%
+   cbind(u_nit, mean_rets) %*% multipli_ers)
> # calculate constraints
> all.equal(1, sum(weight_s))
> all.equal(re_turn, sum(mean_rets*weight_s))
```

Variance of the *Efficient Portfolios*

The *efficient portfolio* variance is equal to:

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

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

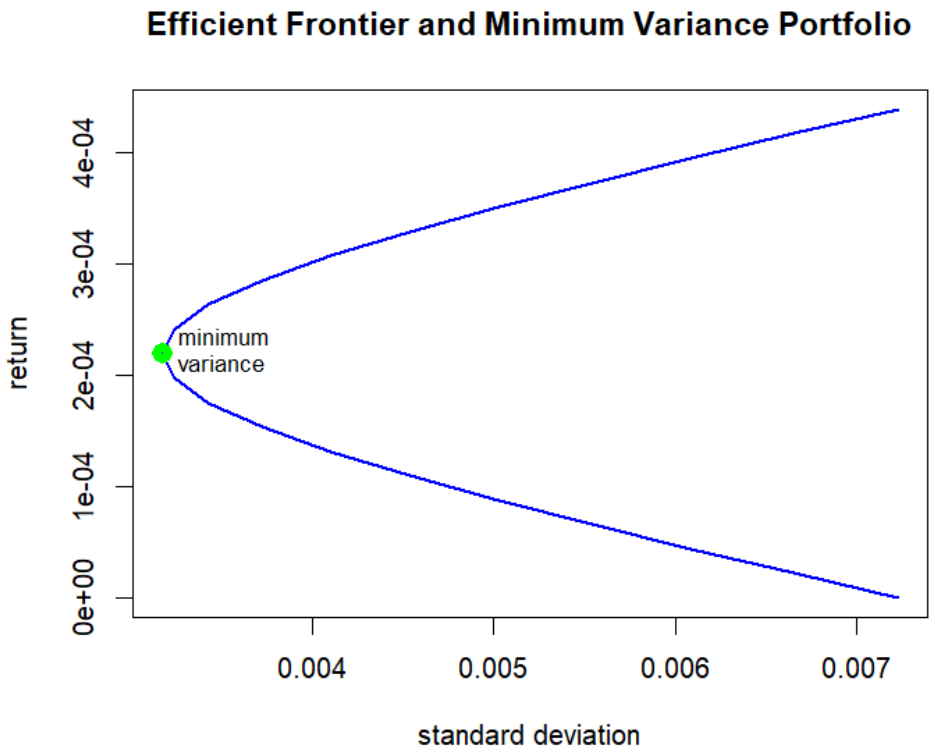
The vertex of the *parabola* is at $r_t = \mathbb{1}^T \mathbb{C}^{-1} \mathbb{r} / \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$ and $\sigma^2 = 1 / \mathbb{1}^T \mathbb{C}^{-1} \mathbb{1}$,

```
> # calculate portfolio return and standard devi
> portf_rets <- drop(re_turns %*% weight_s)
> c(return=mean(portf_rets), sd=sd(portf_rets))
> all.equal(mean(portf_rets), re_turn)
> # calculate portfolio variance
> uu <- c(1, re_turn)
> f_inv <- solve(f_mat)
> all.equal(var(portf_rets), drop(t(uu) %*% f_in
> # calculate vertex of variance parabola
> weight_s <- drop(cov_inv %*% u_nit /
+   drop(t(u_nit) %*% cov_inv %*% u_nit))
> portf_rets <- drop(re_turns %*% weight_s)
> v_rets <-
+   drop(t(u_nit) %*% cov_inv %*% mean_rets /
+   t(u_nit) %*% cov_inv %*% u_nit)
> all.equal(mean(portf_rets), v_rets)
> var_min <-
+   drop(1/t(u_nit) %*% cov_inv %*% u_nit)
> all.equal(var(portf_rets), var_min)
```

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
> target_s <- v_rets*(1+seq(from=-1, to=1, by=0.
> eff_front <- sapply(target_s, function(re_turn
+   uu <- c(1, re_turn)
+   sqrt(drop(t(uu) %*% f_inv %*% uu))
+ })  # end sapply
> # plot efficient frontier
> x11(width=6, height=5)
> plot(x=eff_front, y=target_s, t="l", col="blue
+      main="Efficient Frontier and Minimum Vari
+      xlab="standard deviation", ylab="return")
> points(x=sqrt(var_min), y=v_rets, col="green",
> text(x=sqrt(var_min), y=v_rets, labels="minimu
+      pos=4, cex=0.8)
```



Maximum *Sharpe* Portfolio Weights

The *Sharpe* ratio is defined as the ratio of excess returns divided by the portfolio standard deviation:

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

Where $\mu = r - r_{rf}$ is the vector of excess returns (returns in excess of the risk-free rate), w is the vector of portfolio weights, and $\sigma = \sqrt{w^T \mathbb{C}\, w}$, 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:

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

We then get:

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

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

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

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

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

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

Returns and Variance of Maximum *Sharpe* Portfolio

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

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

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:

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

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 re_turns
> risk_free <- 0.03/252
> ex_cess <- re_turns - risk_free
> # calculate covariance and inverse matrix
> cov_mat <- cov(re_turns)
> u_nit <- rep(1, NCOL(cov_mat))
> cov_inv <- solve(a=cov_mat)
> # calculate mean excess returns
> ex_cess <- sapply(ex_cess, mean)
> # weights of maximum Sharpe portfolio
> # weight_s <- solve(a=cov_mat, b=re_turns)
> weight_s <- cov_inv %*% ex_cess
> weight_s <- weight_s/drop(t(u_nit) %*% weight_
> # Sharpe ratios
> sqrt(252)*sum(weight_s * ex_cess) /
+   sqrt(drop(weight_s %*% cov_mat %*% weight_s)
> sapply(re_turns - risk_free,
+   function(x) sqrt(252)*mean(x)/sd(x))
> weights_maxsharpe <- weight_s
```

The *Efficient Frontier* and *Capital Market Line*

The maximum *Sharpe* portfolio weights depend on the value of the risk-free rate r_{rf},

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

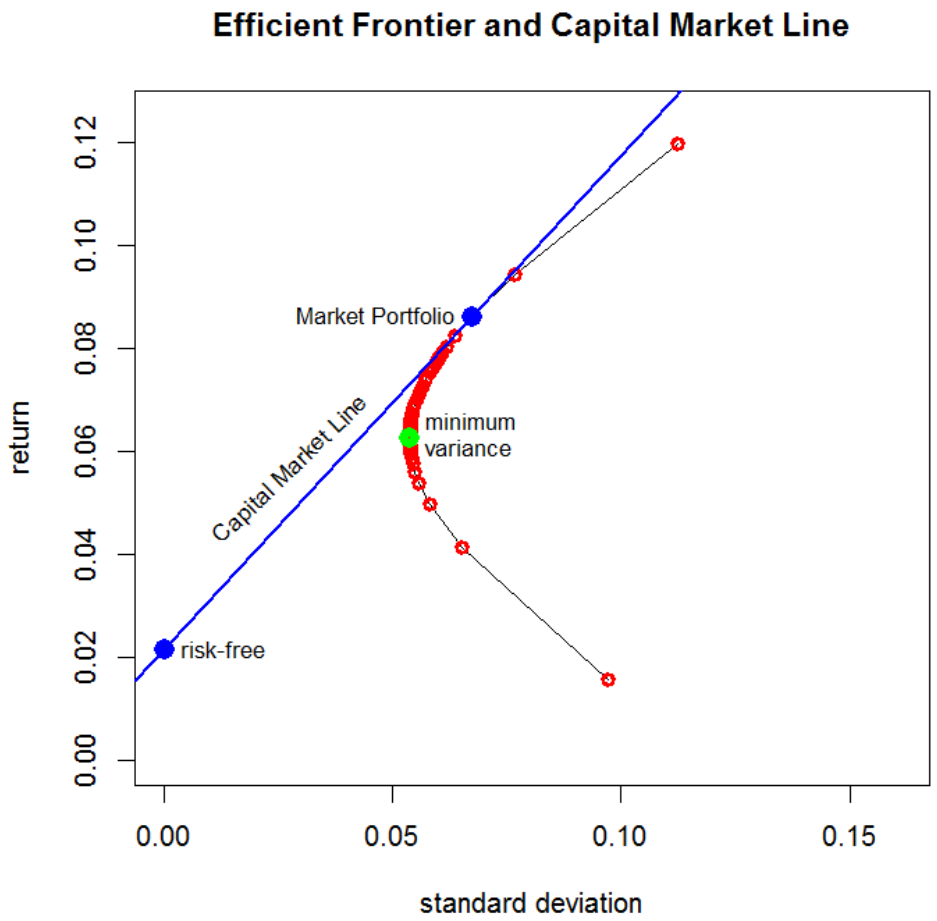
The *Efficient Frontier* is the set of *efficient portfolios*, that have the lowest risk (standard deviation) for the given level of return,

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_{rf},

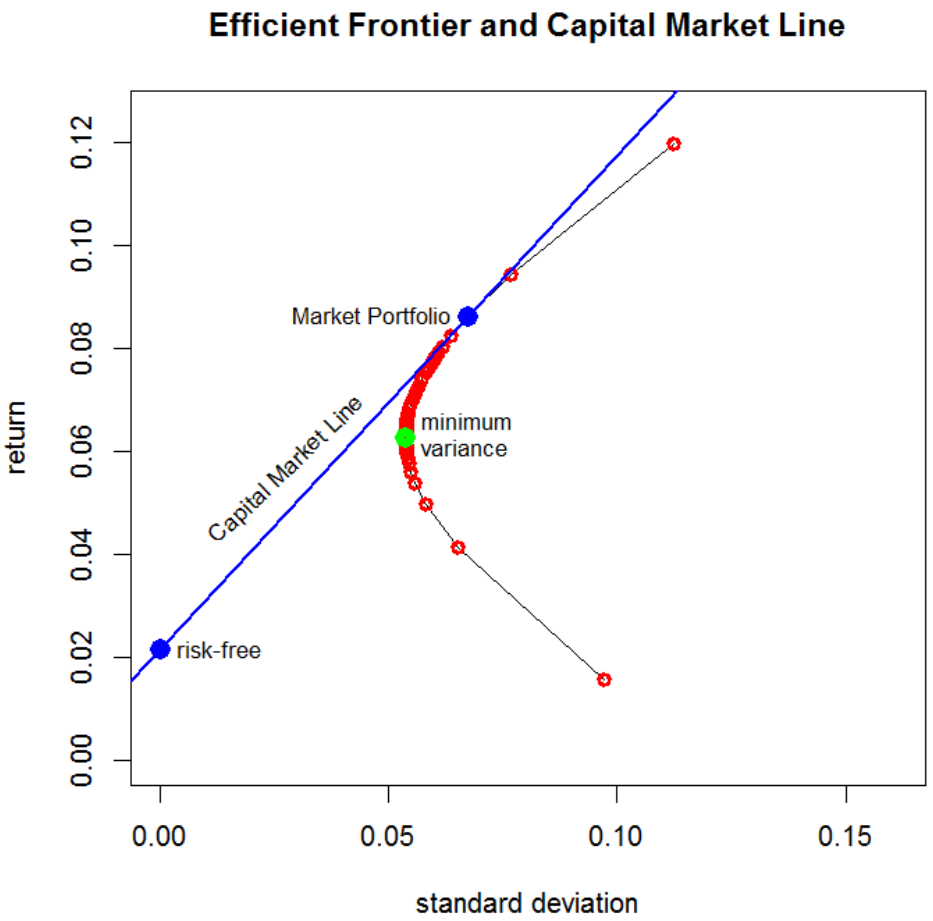
The maximum *Sharpe* portfolios are also called *tangency* portfolios, since they are the tangency point on the *Efficient Frontier*,

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



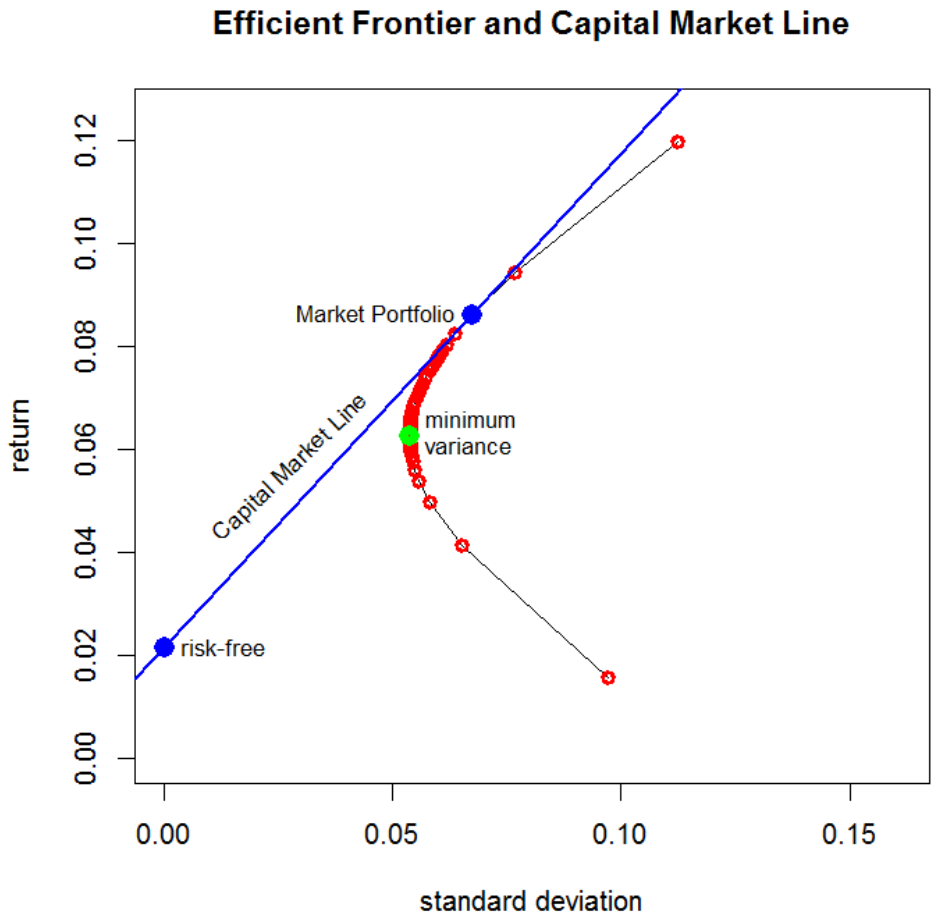
Plotting *Efficient Frontier* and Maximum *Sharpe* Portfolios

```
> # calculate minimum variance weights
> weight_s <- cov_inv %*% u_nit
> weight_s <- weight_s / drop(t(u_nit) %*% weigh
> # minimum standard deviation and return
> std_dev <- sqrt(252*drop(weight_s %*% cov_mat
> min_ret <- 252*sum(weight_s * mean_rets)
> # calculate maximum Sharpe portfolios
> risk_free <- (min_ret * seq(-10, 10, by=0.1)^3
> eff_front <- sapply(risk_free, function(risk_f
+   weight_s <- cov_inv %*% (mean_rets - risk_fr
+   weight_s <- weight_s/drop(t(u_nit) %*% weigh
+   # portfolio return and standard deviation
+   c(return=252*sum(weight_s * mean_rets),
+     stddev=sqrt(252*drop(weight_s %*% cov_mat
+ })  # end sapply
> eff_front <- cbind(252*risk_free, t(eff_front)
> colnames(eff_front)[1] <- "risk-free"
> eff_front <- eff_front[is.finite(eff_front[, "
> eff_front <- eff_front[order(eff_front[, "retu
> # plot maximum Sharpe portfolios
> plot(x=eff_front[, "stddev"],
+      y=eff_front[, "return"], t="l",
+      xlim=c(0.0*std_dev, 3.0*std_dev),
+      ylim=c(0.0*min_ret, 2.0*min_ret),
+      main="Efficient Frontier and Capital Market Line",
+      xlab="standard deviation", ylab="return")
> points(x=eff_front[, "stddev"], y=eff_front[, "return"],
+  col="red", lwd=3)
```



Plotting the *Capital Market Line*

```
> # plot minimum variance portfolio
> points(x=std_dev, y=min_ret, col="green", lwd=
> text(std_dev, min_ret, labels="minimum \nvaria
+      pos=4, cex=0.8)
> # draw Capital Market Line
> sor_ted <- sort(eff_front[, 1])
> risk_free <-
+   sor_ted[findInterval(x=0.5*min_ret, vec=sor_
> points(x=0, y=risk_free, col="blue", lwd=6)
> text(x=0, y=risk_free, labels="risk-free",
+      pos=4, cex=0.8)
> in_dex <- match(risk_free, eff_front[, 1])
> points(x=eff_front[in_dex, "stddev"],
+  y=eff_front[in_dex, "return"],
+  col="blue", lwd=6)
> text(x=eff_front[in_dex, "stddev"],
+      y=eff_front[in_dex, "return"],
+      labels="market portfolio",
+      pos=2, cex=0.8)
> sharp_e <- (eff_front[in_dex, "return"]-risk_f
+   eff_front[in_dex, "stddev"]
> abline(a=risk_free, b=sharp_e, col="blue", lwd
> text(x=0.7*eff_front[in_dex, "stddev"],
+      y=0.7*eff_front[in_dex, "return"]+0.01,
+      labels="Capital Market Line", pos=2, cex=0.8,
+      srt=45*atan(sharp_e*hei_ght/wid_th)/(0.25*pi))
```

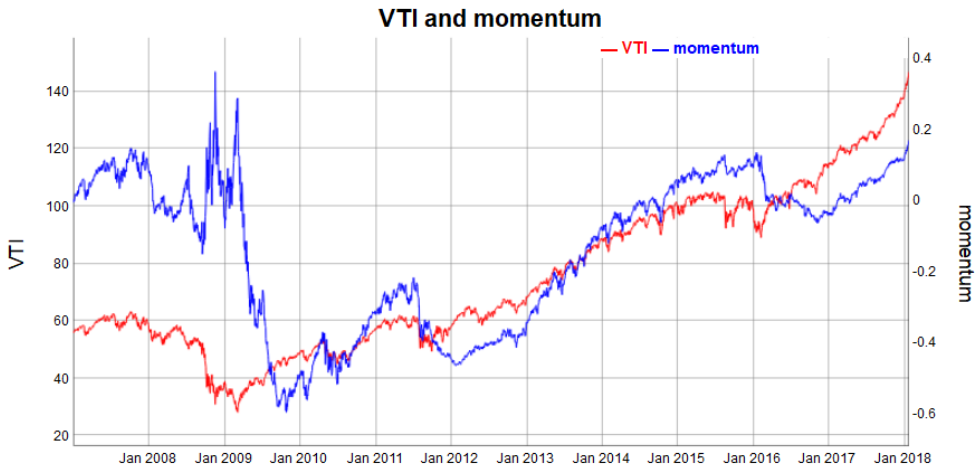


The *Capital Market Line* represents delevered and levered portfolios, consisting of the *market* portfolio combined with the *risk-free* rate,

Momentum Strategy for *S&P500* Stock Portfolio

A very simple *momentum* strategy for the *S&P500*, is to go long constituents with positive recent performance, and short constituents with negative performance,

This *momentum* strategy does not perform well and suffers from *momentum crashes* when the market rebounds sharply from a recent lows,




```
> # calculate rolling variance of S&P500 portfol
> wid_th <- 252
> vari_ance <- roll::roll_var(re_turns, width=wi
> vari_ance <- zoo::na.locf(vari_ance)
> vari_ance[is.na(vari_ance)] <- 0
> # calculate rolling Sharpe of S&P500 portfoli
> returns_width <- rutils::diff_it(price_s, lagg
> weight_s <- returns_width/sqrt(wid_th*vari_an
> weight_s[vari_ance==0] <- 0
> weight_s[1:wid_th, ] <- 1
> weight_s[is.na(weight_s)] <- 0
> weight_s <- weight_s/rowSums(abs(weight_s))/pi
> weight_s[is.na(weight_s)] <- 0
> weight_s <- rutils::lag_it(weight_s)
> sum(is.na(weight_s))
> # calculate portfolio profits and losses
> pnl_s <- rowSums(weight_s*re_turns)
```

```
> # Calculate transaction costs
> bid_offer <- 0.001
> cost_s <- 0.5*bid_offer*price_s*abs(rutils::di
> cost_s <- rowSums(cost_s)
> pnl_s <- (pnl_s - cost_s)
> pnl_s <- cumsum(pnl_s)
> pnl_s <- xts(pnl_s, order.by=index(price_s))
> pnl_s <- cbind(rutils::env_etf$VTI[, 4], pnl_s
> pnl_s <- na.omit(pnl_s)
> colnames(pnl_s) <- c("VTI", "momentum")
> col_names <- colnames(pnl_s)
> # plot momentum and VTI
> dygraphs::dygraph(pnl_s, main=paste(col_names,
+   dyAxis("y", label=col_names[1], independentT
+   dyAxis("y2", label=col_names[2], independent
+   dySeries(col_names[2], axis="y2", col=c("red
```

Rolling Portfolio Optimization Strategy for *S&P500*

```
> library(HighFreq)
> load("C:/Develop/R/lecture_slides/data/sp500_p
> n_col <- NCOL(price_s)
> # define end_points
> end_points <- rutils::calc_endpoints(price_s,
> end_points <- end_points[end_points>50]
> len_gth <- NROW(end_points)
> look_back <- 12
> start_points <- c(rep_len(1, look_back-1), end
> # scale price_s
> date_s <- index(price_s)
> price_s <- t(t(price_s) / as.numeric(price_s[1
> sum(is.na(price_s))
> in_dex <- xts(rowSums(price_s)/n_col, date_s)
> re_turns <- diff_it(price_s)
> # compile backtest function
> Rcpp::sourceCpp(file="C:/Develop/R/lecture_sl
> # run backtest function
> al_pha <- 0.01
> max_eigen <- 2
> strat_rets_arma <- roll_portf(re_turns,
+   re_turns,
+   start_points-1,
+   end_points-1,
+   al_pha=al_pha,
+   max_eigen=max_eigen)
```


```
> # plot strategy
> strat_rets_arma <- cumsum(strat_rets_arma)
> strat_rets_arma <- xts(strat_rets_arma, date_s
> library(dygraphs)
> strat_rets_arma <- cbind(strat_rets_arma, in_d
> col_names <- c("Strategy", "Index")
> colnames(strat_rets_arma) <- col_names
> dygraphs::dygraph(strat_rets_arma, main=paste(
+   dyAxis("y", label=col_names[1], independentT
+   dyAxis("y2", label=col_names[2], independent
+   dySeries(col_names[2], axis="y2", col=c("red
```

Range Volatility Estimators of OHLC Time Series

Range volatility estimators utilize the high and low prices, and therefore have lower standard error than the standard *close-to-close* estimator,

The *Garman-Klass* estimator uses the *low-to-high* price range, but it underestimates volatility because it doesn't account for *close-to-open* price jumps:

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(0.5\log(\frac{H_i}{L_i})^2 - (2\log 2 - 1)\log(\frac{C_i}{O_i})^2)$$

The *Yang-Zhang* estimator is the most efficient (has the lowest standard error) among unbiased estimators, and also accounts for *close-to-open* price jumps:

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\log(\frac{O_i}{C_{i-1}}) - \bar{r}_{co})^2 +$$

$$0.134(\log(\frac{C_i}{O_i}) - \bar{r}_{oc})^2 +$$

$$\frac{0.866}{n}\sum_{i=1}^{n}(\log(\frac{H_i}{O_i})\log(\frac{H_i}{C_i}) + \log(\frac{L_i}{O_i})\log(\frac{L_i}{C_i}))$$

```
> # daily SPY volatility from minutely prices us
> library(TTR)
> sqrt((6.5*60)*mean(na.omit(
+   TTR::volatility(SPY, N=1,
+           calc="yang.zhang"))^2))
> # SPY volatility using package HighFreq
> 60*sqrt((6.5*60)*agg_regate(oh_lc=SPY,
+     weight_ed=FALSE, mo_ment="run_variance",
+     calc_method="yang_zhang"))
```

Theoretically, the *Yang-Zhang* (*YZ*) and *Garman-Klass-Yang-Zhang* (*GKYZ*) range variance estimators are unbiased and have up to seven times smaller standard errors than the standard close-to-close estimator,

But in practice, prices are not observed continuously, so the price range is underestimated, and so is the variance when using the *YZ* and *GKYZ* range estimators,

Therefore in practice the *YZ* and *GKYZ* range estimators underestimate volatility,

In addition, their standard errors are reduced less than by the theoretical amount, for the same reason,

Standard Errors of Volatility Estimators Using Bootstrap

The standard errors of estimators can be calculated using a *bootstrap* simulation,

The *bootstrap* procedure generates new data by randomly sampling with replacement from the observed data set,

The *bootstrapped* data is then used to re-calculate the estimator many times, producing a vector of values,

The *bootstrapped* estimator values can then be used to calculate the probability distribution of the estimator and its standard error,

Bootstrapping doesn't provide accurate estimates for estimators that are sensitive to the ordering and correlations in the data,

```
> # standard errors of TTR variance estimators u
> boot_strap <- sapply(1:1e2, function(x) {
+ # create random OHLC
+   oh_lc <- HighFreq::random_ohlc()
+ # calculate variance estimate
+   c(var=var(oh_lc[, 4]),
+     yang_zhang=HighFreq::calc_variance(
+ oh_lc, calc_method="yang_zhang", sca_le=FALSE)
+ })  # end sapply
> # analyze bootstrapped variance
> boot_strap <- t(boot_strap)
> head(boot_strap)
> colMeans(boot_strap)
> apply(boot_strap, MARGIN=2, sd) /
+   colMeans(boot_strap)
```

Homework Assignment

No homework!