Financial Computation: Stock Portfolio Optimization (constrOptim.nl())

Brian Clark

Fall 2018

Contents

1 Overview

This document is broken down into three parts. The first describes how to use R’s “quantmod” package to download stock data. This part is based on a vignette put together and maintained by Majeed Simaan who is a Lally School (RPI) PhD and is now an Assisant Professor of Finance at Stevens Institute of Technology. A link to his github page with the details of his vignette is **here**.

The second section uses the data from the first part to “manually” find and plot the efficient frontier using R’s “constrOptim.nl()” package. You will need the “alabama” package to use the function. Note that there are a few practical limitations for this package, but it is better suited for our portfolio optimization problem than the constrOptim() function. The main difference is that “constrOptim.nl()” accepts linear equality and inequality contraints. For more advanced nonlinear constraints you could use R’s “nloptr” package which is R’s interface with the general “nlopt” package which is compatible with many software packages such as Python. However, in the interest of time, we will restrict our focus to the more simplistic linear case using “coinstrOptim.nl().”

The final section uses some of R’s built-in portfolio optimization routines to find and plot the efficient frontier for some equity data.

2 Downloading Stock Data into R using “quantmod”"

This section is based on Majeed Simaan’s R vignette. Please refer to his original file as his vignette also contains instructions and examples of how to download and manage options data along with macroeconomic data hosted by the **Federal Reserve Bank of St. Louis**.

Prior to starting, you need to install the “quantmod” package. For this example, we download daily equity data for six stocks starting on January 1, 1980.

```r
rm(list=ls())

# install.packages("quantmod")
library("quantmod")

# Define some companies:
tics <- c("AAPL","GE","TWTR","BAC","RAD","PFE")
P.list <- lapply(tics, function(tic)
  get(getSymbols(tic, from = "1980-01-01")))

sapply(P.list,nrow)
```

```
## [1] 9536 9776 1236 9724 8521 9776
```

A few comments are in order for the above block of code. The tics object obviouly constains the ticker symbols you want to use. They correspond to the tickers on Yahoo Finance so if you want other companies you can search for the tickers **here**. We will stick to Majeed's example for now.

The "lapply" function is another way to write a for loop in R. For example, P.list could also be defined (equivalently) as follows:

```r
# The following block is equivalent to the lapply command::
P.list2 <- list(NA)
n <- length(tics)
for (i in 1:n){
  P.list2[[i]] <- get(getSymbols(tics[i], from = "1980-01-01"))
}

# ...is equivalent to...

# P.list <- lapply(tics, function(tic)
#   get(getSymbols(tic, from = "1980-01-01")))

# Printing the number of rows could also be accomplished in a loop:
for (i in 1:n){
  print(nrow(P.list2[[i]]))
}
```

```
## [1] 9536
## [1] 9776
## [1] 1236
## [1] 9724
## [1] 8521
## [1] 9776
```

```r
sapply(P.list,nrow)
```

```
## [1] 9536 9776 1236 9724 8521 9776
```

Note that each of the stocks has a different number of rows. This is because we are using real data and as such there will be missing values (for example, Twitter was not around in 1980!). Let's print the first and last few observations for each stock:

```r
for (i in 1:n){
  print(tics[i])
  print(head(P.list[[i]],4))
  print("-----------------------------------------")
```

```
}
```

```
## [1] "AAPL"
##            AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume
## 1980-12-12  0.513393  0.515625 0.513393   0.513393   117258400
## 1980-12-15  0.488839  0.488839 0.486607   0.486607    43971200
## 1980-12-16  0.453125  0.453125 0.450893   0.450893    26432000
## 1980-12-17  0.462054  0.464286 0.462054   0.462054    21610400
##            AAPL.Adjusted
## 1980-12-12      0.415317
## 1980-12-15      0.393649
## 1980-12-16      0.364757
## 1980-12-17      0.373786
## [1] "-------------------------------------------"
## [1] "GE"
##             GE.Open  GE.High   GE.Low GE.Close GE.Volume GE.Adjusted
## 1980-01-02 1.054688 1.057292 1.015625 1.015625   7147200    0.005260
## 1980-01-03 1.015625 1.031250 0.997396 1.028646   8832000    0.005328
## 1980-01-04 1.039063 1.065104 1.039063 1.062500   8227200    0.005503
## 1980-01-07 1.062500 1.114583 1.054688 1.098958  10113600    0.005692
## [1] "-------------------------------------------"
## [1] "TWTR"
##            TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
## 2013-11-07     45.10     50.09    44.00      44.90   117701600
## 2013-11-08     45.93     46.94    40.69      41.65    27925300
## 2013-11-11     40.50     43.00    39.40      42.90    16113900
## 2013-11-12     43.66     43.78    41.83      41.90     6316700
##            TWTR.Adjusted
## 2013-11-07         44.90
## 2013-11-08         41.65
## 2013-11-11         42.90
## 2013-11-12         41.90
## [1] "-------------------------------------------"
## [1] "BAC"
##            BAC.Open BAC.High BAC.Low BAC.Close BAC.Volume BAC.Adjusted
## 1980-03-17  1.40625 1.468750 1.40625  1.406250      57600     0.112083
## 1980-03-18  1.40625 1.406250 1.37500  1.390625     140000     0.110837
## 1980-03-19  1.40625 1.453125 1.40625  1.437500      88000     0.114574
## 1980-03-20  1.43750 1.437500 1.43750  1.437500      26400     0.114574
## [1] "-------------------------------------------"
## [1] "RAD"
##            RAD.Open RAD.High RAD.Low RAD.Close RAD.Volume RAD.Adjusted
## 1984-12-17  6.25000  6.40625 6.25000   6.40625     153600     2.548513
## 1984-12-18  6.40625  6.56250 6.34375   6.50000     628400     2.585808
## 1984-12-19  6.50000  6.53125 6.43750   6.46875     277600     2.573376
## 1984-12-20  6.50000  6.56250 6.46875   6.50000     268400     2.585808
## [1] "-------------------------------------------"
## [1] "PFE"
##            PFE.Open PFE.High  PFE.Low PFE.Close PFE.Volume PFE.Adjusted
## 1980-01-02 0.809896 0.809896 0.781250  0.781250    3216000     0.000712
## 1980-01-03 0.781250 0.789063 0.770833  0.781250    2846400     0.000712
## 1980-01-04 0.789063 0.809896 0.789063  0.809896    3316800     0.000738
## 1980-01-07 0.809896 0.820313 0.799479  0.809896    2184000     0.000738
## [1] "-------------------------------------------"
```

```r
for (i in 1:n){
  print(tics[i])
  print(tail(P.list[[i]],4))
  print("-----------------------------------------")
}
```

```
## [1] "AAPL"
##            AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume
## 2018-10-01    227.95    229.42   226.35     227.26    23600800
## 2018-10-02    227.25    230.00   226.63     229.28    24788200
## 2018-10-03    230.05    233.47   229.78     232.07    28654800
## 2018-10-04    230.78    232.35   226.73     227.99    31980700
##            AAPL.Adjusted
## 2018-10-01        227.26
## 2018-10-02        229.28
## 2018-10-03        232.07
## 2018-10-04        227.99
## [1] "-----------------------------------------"
## [1] "GE"
##            GE.Open GE.High GE.Low GE.Close GE.Volume GE.Adjusted
## 2018-10-01   13.02   13.07  11.94    12.09 308106800       12.09
## 2018-10-02   12.32   12.48  11.77    12.32 148705800       12.32
## 2018-10-03   12.34   12.63  12.28    12.48  82966100       12.48
## 2018-10-04   12.41   12.68  12.34    12.66  74874600       12.66
## [1] "-----------------------------------------"
## [1] "TWTR"
##            TWTR.Open TWTR.High TWTR.Low TWTR.Close TWTR.Volume
## 2018-10-01     28.51     28.70    28.00      28.31    20538900
## 2018-10-02     28.14     28.62    27.91      28.19    17714400
## 2018-10-03     28.38     29.12    28.25      29.01    19358700
## 2018-10-04     28.75     28.76    27.87      28.23    21112600
##            TWTR.Adjusted
## 2018-10-01         28.31
## 2018-10-02         28.19
## 2018-10-03         29.01
## 2018-10-04         28.23
## [1] "-----------------------------------------"
## [1] "BAC"
##            BAC.Open BAC.High BAC.Low BAC.Close BAC.Volume BAC.Adjusted
## 2018-10-01    29.68    29.94   29.54     29.65   53900900        29.65
## 2018-10-02    29.58    29.72   29.27     29.58   42940300        29.58
## 2018-10-03    29.81    30.18   29.72     30.00   61080300        30.00
## 2018-10-04    30.17    30.79   30.14     30.43   72280100        30.43
## [1] "-----------------------------------------"
## [1] "RAD"
##            RAD.Open RAD.High RAD.Low RAD.Close RAD.Volume RAD.Adjusted
## 2018-10-01     1.27     1.28    1.21      1.21   16145900         1.21
## 2018-10-02     1.21     1.22    1.15      1.15   21998700         1.15
## 2018-10-03     1.17     1.17    1.14      1.15   14013700         1.15
## 2018-10-04     1.15     1.21    1.15      1.15   19000000         1.15
## [1] "-----------------------------------------"
## [1] "PFE"
##            PFE.Open PFE.High PFE.Low PFE.Close PFE.Volume PFE.Adjusted
## 2018-10-01    44.03    44.52   43.91     44.27   16008100        44.27
```

```
## 2018-10-02    44.21    44.39   44.13     44.22   17236900         44.22
## 2018-10-03    44.28    44.84   44.28     44.81   22730200         44.81
## 2018-10-04    44.53    44.79   44.29     44.70   17989000         44.70
## [1] "-----------------------------------------"
```

While all of our stocks have different starting dates, they all have the same end date which is the most recent closing date (it will change depending on when you run the code).

Not all data is relevant for our puposes. What we are really interested in are the adjusted prices which are given in the last column of each list object. The adjusted prices are adjusted for stock splits and dividends. The idea is that the adjusted prices represent the true return on an investment in a given stock.

Therefore, the next step is to get all the adjusted prices into a single object:

```
# Get the adjusted prices into a single object
P.adj <- lapply(P.list, function(p) p[,6])

# Merge the elements of the list
P <- Reduce(merge,P.adj)
names(P) <- tics

head(P,10)
```

```
##              AAPL       GE TWTR BAC RAD      PFE
## 1980-01-02   NA 0.005260   NA  NA  NA 0.000712
## 1980-01-03   NA 0.005328   NA  NA  NA 0.000712
## 1980-01-04   NA 0.005503   NA  NA  NA 0.000738
## 1980-01-07   NA 0.005692   NA  NA  NA 0.000738
## 1980-01-08   NA 0.005894   NA  NA  NA 0.000773
## 1980-01-09   NA 0.005827   NA  NA  NA 0.000757
## 1980-01-10   NA 0.005854   NA  NA  NA 0.000747
## 1980-01-11   NA 0.005840   NA  NA  NA 0.000757
## 1980-01-14   NA 0.005840   NA  NA  NA 0.000757
## 1980-01-15   NA 0.005706   NA  NA  NA 0.000738
```

```
tail(P,10)
```

```
##              AAPL    GE  TWTR   BAC  RAD   PFE
## 2018-09-21 217.66 12.17 28.50 31.03 1.25 44.06
## 2018-09-24 220.79 11.74 28.60 30.74 1.24 43.93
## 2018-09-25 222.19 11.27 29.11 30.67 1.23 43.79
## 2018-09-26 220.42 11.39 29.01 30.13 1.28 43.68
## 2018-09-27 224.95 11.53 29.42 29.94 1.25 43.90
## 2018-09-28 225.74 11.29 28.46 29.46 1.28 44.07
## 2018-10-01 227.26 12.09 28.31 29.65 1.21 44.27
## 2018-10-02 229.28 12.32 28.19 29.58 1.15 44.22
## 2018-10-03 232.07 12.48 29.01 30.00 1.15 44.81
## 2018-10-04 227.99 12.66 28.23 30.43 1.15 44.70
```

Again, the P.adj "lapply" funciton could be written equivalently in a loop as follows:

```
# Get the adjusted prices into a single object
P.adj2 <- list(NA)
for (i in 1:n){
  P.adj2[[i]] <- P.list[[i]][,6]
}

P2 <- Reduce(merge,P.adj2)
```

```r
names(P2) <- tics

head(P2,10)
```

```
##              AAPL       GE TWTR BAC RAD      PFE
## 1980-01-02   NA 0.005260   NA  NA  NA 0.000712
## 1980-01-03   NA 0.005328   NA  NA  NA 0.000712
## 1980-01-04   NA 0.005503   NA  NA  NA 0.000738
## 1980-01-07   NA 0.005692   NA  NA  NA 0.000738
## 1980-01-08   NA 0.005894   NA  NA  NA 0.000773
## 1980-01-09   NA 0.005827   NA  NA  NA 0.000757
## 1980-01-10   NA 0.005854   NA  NA  NA 0.000747
## 1980-01-11   NA 0.005840   NA  NA  NA 0.000757
## 1980-01-14   NA 0.005840   NA  NA  NA 0.000757
## 1980-01-15   NA 0.005706   NA  NA  NA 0.000738
```

```r
tail(P2,10)
```

```
##              AAPL    GE  TWTR   BAC  RAD   PFE
## 2018-09-21 217.66 12.17 28.50 31.03 1.25 44.06
## 2018-09-24 220.79 11.74 28.60 30.74 1.24 43.93
## 2018-09-25 222.19 11.27 29.11 30.67 1.23 43.79
## 2018-09-26 220.42 11.39 29.01 30.13 1.28 43.68
## 2018-09-27 224.95 11.53 29.42 29.94 1.25 43.90
## 2018-09-28 225.74 11.29 28.46 29.46 1.28 44.07
## 2018-10-01 227.26 12.09 28.31 29.65 1.21 44.27
## 2018-10-02 229.28 12.32 28.19 29.58 1.15 44.22
## 2018-10-03 232.07 12.48 29.01 30.00 1.15 44.81
## 2018-10-04 227.99 12.66 28.23 30.43 1.15 44.70
```

Now we have a data set of equity prices which contains time series observations in along the rows and the stocks across the columns. In the next section, we will prepare the data to form a set of efficient portfolios.

3 The Efficient Frontier

The first step in forming efficient portfolios is to define the necessary parameters for the optimizaiton routines. In the most basic case, all we need is a vector of mean returns and a variance-covariance matrix of the returns.

To define the expected returns, we have several options. For one, we could use an asset pricing model such as the Capital Asset Pricing Model (CAPM) where the expected return is defined as follows:

$$R_{i,t} = R_{f,t} + \beta_{i,t}(R_{M,t} - R_{f,t}) \tag{1}$$

where, $R_{i,t}$ is the raw return for firm i at time t, $R_{f,t}$ is the risk free rate at time t (such as a short maturity Treasury), $R_{M,t}$ is the return on the overall market at time t. The market return can be proxied by a major index such as the S&P500.

In the CAPM, β is a measure of how an individual asset moves with the market. It can be estimated by a simple ordinary least squares (OLS) regression:

$$(R_{i,t} - R_{f,t}) = \beta_{i,t}(R_{M,t} - R_{f,t}) + \epsilon_{i,t} \tag{2}$$

As such, $\beta_{i,t}$ can also be expressed as follows:

$$\beta_{i,t} = \frac{Corr(R_{i,t} - R_{f,t}, R_{M,t} - R_{f,t})}{Var(R_{M,t} - R_{f,t})} \tag{3}$$

Intuitively, you can think about $\beta_{i,t}$ as the number of units of systematic risk inherent to asset i where a unit of risk is defined relative to the overall market risk ($Var(R_{M,t} - R_{f,t})$).

3.1 Define Mean and Covariance for the Optimization

However, for the purposes of this example, we will simplify the analysis and simply use the vector of mean returns over our sample period as out best estimate of the mean vector.

```
# Returns:
Rets <- P/lag(P) - 1

# Define parameters for the optimization:
# 1. The returns: retain the mean returns for all non-missing observations
R <- apply(Rets,2,function(x) mean(x,na.rm = TRUE))

# 2. The covariance matrix: run pairwise correlations
Sigma <- var(Rets, use="pairwise")
n <- length(R)

# Annualize data:
R <- (1 + R)^252 - 1
Sigma <- Sigma * 252


head(R,10)
```

```
##       AAPL         GE       TWTR        BAC        RAD        PFE
## 0.31596838 0.27597612 0.05822827 0.24681113 0.13906825 0.39992949
```

```
print(Sigma)
```

```
##            AAPL         GE        TWTR        BAC        RAD         PFE
## AAPL 0.21041200 0.04033009 0.022651962 0.04160559 0.03605614 0.026809059
## GE   0.04033009 0.08738836 0.014068513 0.05323977 0.03973695 0.032085231
## TWTR 0.02265196 0.01406851 0.297323854 0.02372492 0.01281202 0.009954609
## BAC  0.04160559 0.05323977 0.023724923 0.15142032 0.05626598 0.030875443
## RAD  0.03605614 0.03973695 0.012812018 0.05626598 0.30929330 0.026861208
## PFE  0.02680906 0.03208523 0.009954609 0.03087544 0.02686121 0.108223506
```

Note that both the mean vector and covariance matrix are annualized. The diagonal terms on Σ represent the variance of the individual stocks and the off-diagonal terms represent the covariances between r_i and r_j. As for the mean vector, the expected returns are all positive, although they are abnormally hig and unrealistic. However, they till be fine for this example. If you were to use the CAPM, you would get smaller (and in this case) more realistic returns.

3.2 Plot some Random Portfolios

In this step, we plot several random portfolios using random combinations of out six stocks. The only constraints we place on the analysis are $\sum_{i=1}^{n} w_i = 1$ and $w_i \geq 0$ for $i = 1..n$.

The first step is to define functions for the mean and variance of the portfolios:

$$R_{pf} = \mathbf{w}'\mathbf{R} \tag{4}$$

and

$$\sigma_{pf} = \mathbf{w}'\mathbf{\Sigma}\mathbf{w} \tag{5}$$

where w is a vector of the portfolio weights, R is the mean vector of expected returns, and Σ is the expected covariance matirx.

```
# Generate many random portfolios to plot:
R_pf <- function(w,R=R){
  r <- t(w) %*% R
  return(r)
}

S_pf <- function(w,Sigma=Sigma){
  s <- t(w) %*% Sigma %*% w
  return(s)
}
```

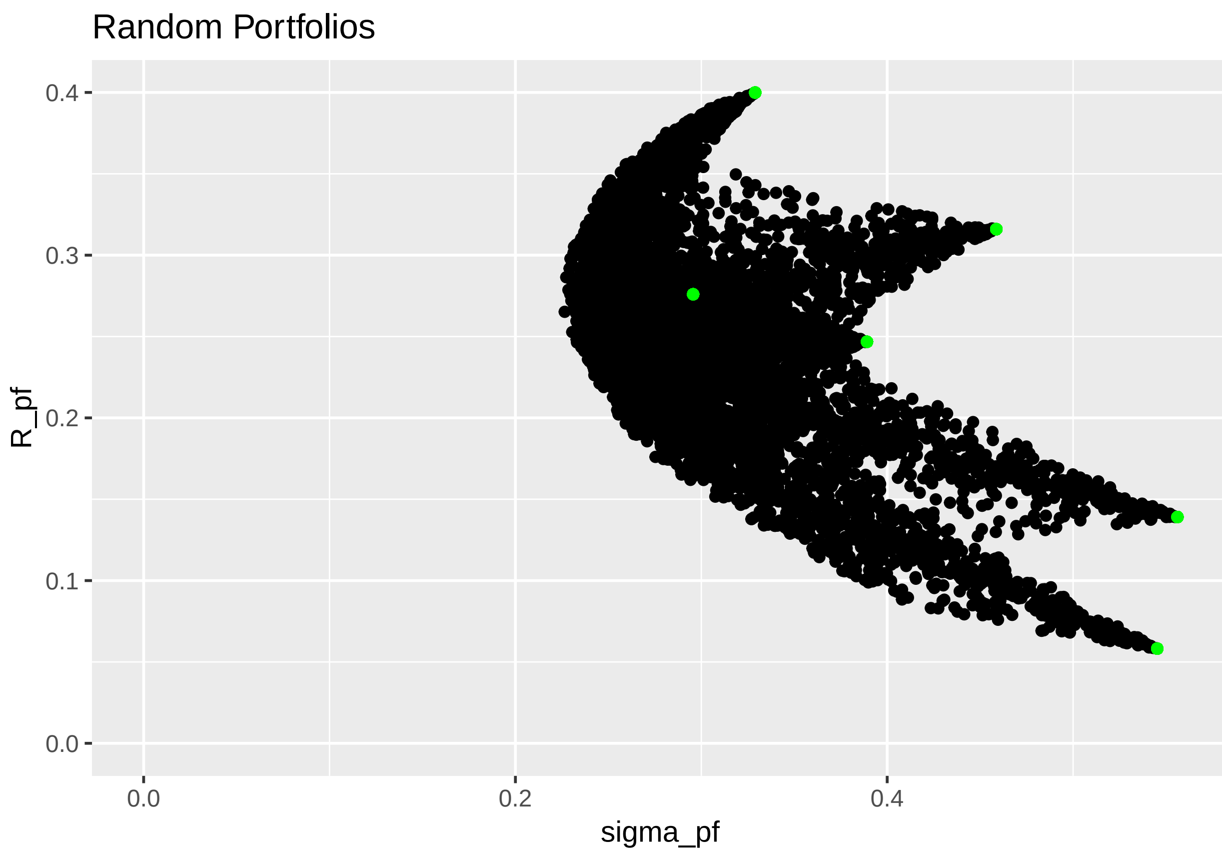
Next, we can define a function to generate a series of random portfolios of the $n = 6$ stocks.

```
randPfs <- function(R,Sigma){
  w <- rep(NA,length=n)
  w[1] <- runif(1)
  for (i in 2:n){
    w[i] <- runif(1,min=0,max=1 - sum(w,na.rm=TRUE))
  }
  w <- w / sum(w)
  w <- sample(w)
  R0 <- R_pf(w,R)
  S0 <- S_pf(w,Sigma)
  results <- list("R"=R0,"S"=S0,"w"=w)
  return(results)
}
```

The above function generates random weights that should be evenly spread across the possible combinations of portfolio weights. The next step is to call the function and generate a series of random portfolios to plot.

```
N <- n*1000
pfs <- as.data.frame(matrix(NA,ncol=2+n,nrow=N))
for (i in 1:N){
  tmp <- randPfs(R,Sigma)
  pfs[i,] <- matrix(c(tmp$R,sqrt(tmp$S),t(tmp$w)))
}

# The following block of code adds a single row to the pfs data frame
# that will be used later to generate a random starting value
# in the portfolio optimization routine.
x0 <- rep(0,length=n)
x0[which.max(R)] <- 1
pfs <- rbind(pfs,c(max(R),sqrt(S_pf(x0,Sigma)),x0))

# Plot the results:
library("ggplot2")
```

```r
p <- ggplot(data=pfs,aes(x=V2,y=V1)) +
  geom_point() +
  ylab(label="R_pf") +
  xlab(label="sigma_pf") +
  ggtitle("Random Portfolios")  +
  expand_limits(x=0, y=0)

# Plot the 100% portfolios:
for (i in 1:n){
  wtmp <- rep(0,length=n)
  wtmp[i]=1
  p <- p + geom_point(x=c(sqrt(S_pf(wtmp,Sigma))),y=c(R_pf(wtmp,R)),
                      color="green")
}
print(p)
```



The above plot shows $1,000n$ random portfolios of the n stocks. The green dots represnt the 100% portfolios where the investor puts all their wealth into a single stock.

3.3 The Efficient Frontier

The next step is to find a series of portoflios that either:

1. Maximize return for a given level of risk
2. Minimize risk for a given level of return

To do so, we will utilize R's *constrOptim.nl)* function. The function is a general optimization package that minimizes a function[1] and can handle *linear equality* and *inequality* constraints.[2]

The *constrOptim.nl()* function solves a general optimization problem that can be stated as follows:

$$\begin{gathered}\min_{\mathbf{x}} f(\mathbf{x}) \\ \text{S.T. } \mathbf{H_{in}} \geq \mathbf{0} \\ \mathbf{H_{eq}} = \mathbf{0}\end{gathered} \tag{6}$$

where H_{in} is a set of linear inequality constraints and H_{eq} is a set on linear equality constraints. The *constrOptim.nl()* function documentation can be found **here**. The function can call several different optimizaiton algorithms that require the gradient of the objective function. The only "no-derivative" algorithm is the well-known "Nelder-Mead" (simplex) algorithm, which is a very flexible and robust algorithm - although it can be somewhat slow to converge. Other include impementations of variations of Newton or conjugent gradient methods.

The first step of the analysis is to decide which optimization probelm we will solve (minimize risk or maximize return). In this example, we will minimize the portfolio variance subject to a wealth constraint, no shorting constraint and require a minimum level of return. In particuar, we will solve thfollowing problem:

$$\min_{\mathbf{w}} \mathbf{w}'\mathbf{\Sigma}\mathbf{w}$$

Subject to:

1. No shorting: $w_i \geq 0$
2. Wealth constraint: $\sum_{i=1}^{n} w_i = 1$
3. Minimum return: $\mathbf{w}'\mathbf{R} = R_0$

3.3.1 Defining the Feasible Region of Portfolios

In this example, we are restricted to solving problems with linear equality and inequality constraints. The first step is to define the region over which to solve for the efficient frontier. In particular, we want to find the feasible region, $[\underline{R_0}, \bar{R_0}]$, over which the above problem can be solved by changing the minimum return constraint.

In this setting, solving for $\bar{R_0}$ is straight-forward: simply allocate 100% of the wealth to the stock with the highest expected return. However, solving for $\underline{R_0}$ is more complicated. We will refer to this portfolio as the minimum variance portfolio and solve the following problem:

$$\min_{\mathbf{w}} \mathbf{w}'\mathbf{\Sigma}\mathbf{w}$$

Subject to:

1. No shorting: $w_i \geq 0$
2. Wealth constraint: $\sum_{i=1}^{n} w_i = 1$

[1] Although it can be ammended to maximize a function as well.

[2] A more flexible package that can handle non-linear equality and inequality contraints is *nloptr()* which is R's interface with the general *nlopt* package avilable across multiple platforms including Python, R, and Matlab.

The first step is to define the constraints. Note that the *constrOptim.nl()* function requires linear inequality constraints to be of the form:

$$\mathbf{Ax} - \mathbf{b} \geq 0$$

and the equality constraints to be of the form:

$$\mathbf{Ax} - \mathbf{b} = 0$$

The no shorting constraint ($w_i \geq 0$) can be written as follows:

$$\begin{aligned}
1w_1 + 0w_2 + 0w_3 + 0w_4 + 0w_5 + 0w_6 &\geq 0 \\
0w_1 + 1w_2 + 0w_3 + 0w_4 + 0w_5 + 0w_6 &\geq 0 \\
0w_1 + 0w_2 + 1w_3 + 0w_4 + 0w_5 + 0w_6 &\geq 0 \\
0w_1 + 0w_2 + 0w_3 + 1w_4 + 0w_5 + 0w_6 &\geq 0 \\
0w_1 + 0w_2 + 0w_3 + 0w_4 + 1w_5 + 0w_6 &\geq 0 \\
0w_1 + 0w_2 + 0w_3 + 0w_4 + 0w_5 + 1w_6 &\geq 0
\end{aligned}$$

The wealth constraint can written as:

$$w_1 + w_2 + ... + 1w_n - 1 = 0$$

constrOptim.nl() takes the contraints in the form $\mathbf{Ax} - \mathbf{b} \geq 0$. For the above set of n constraints:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and $\mathbf{b}$ is:

$$\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Similarly, the equality constraints can be written as follows:

$$\mathbf{A_{eq}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ r_1 & r_2 & r_3 & r_4 & r_5 & r_6 \end{bmatrix}$$

and

$$\mathbf{b_e q_{eq}} = \begin{bmatrix} 0 \\ \bar{R} \end{bmatrix}$$

Note that the last equality constraint could also be written as an inequality constraint but it will always be binding so it shoiuldn't make a difference. It is also not necessary for the first step in which we find the minimum variance portoflio.

To implement this problem using *constrOptim.nl()* in R, note that the arguments *hin* and *heq* correspond to $\mathbf{Ax} - \mathbf{b}$ and $\mathbf{A_{eq}x} - \mathbf{b}$, respectively.

First, set the the linear inequality and equality constraints as follows:

```
library("alabama")
```

```
## Loading required package: numDeriv
```

```
# ------Minimum Variance Portfolio--------
# Inequality
hin <- function(x){
  h <- x
  # # Or, alternatively:
  # b <- rep(0,length=length(x))
  # for (i in 1:j){
  #   h[j] <- x[j] - b[j]
  # }
  return(x)
}

# Equality
heq <- function(x){
  h <- sum(x) - 1
  return(h)
}
```

Next set the objective funtion:

```
# ------Objective Function: Minimize Variance-----
eval_f <- function(x){
  s <- t(x) %*% Sigma %*% x
  return(s)
}
```

With the constriants set, we can now solve for the minimum variance portfolio. For the example, we will set the following parameters that control the optimization (note that most of these paramters are specific to the problem at hand and can be better "tuned" for optimal performance):

```
# Set the starting valae to a feasible portfolio:
x0 <- rep(1/n,length=n)

# Solve the optimization problem:
tmp <- constrOptim.nl(par=x0, fn=eval_f,
                      heq=heq, hin=hin)
```

```
## Min(hin):  0.1666667 Max(abs(heq)):  0
## Outer iteration:  1
## Min(hin):  0.1666667 Max(abs(heq)):  0
## par:  0.166667 0.166667 0.166667 0.166667 0.166667 0.166667
## fval =   0.05828
##
## Outer iteration:  2
## Min(hin):  0.06234259 Max(abs(heq)):  0
## par:  0.0934572 0.281643 0.117004 0.0888629 0.0623426 0.264552
## fval =   0.04189
##
## Outer iteration:  3
```

```
## Min(hin):  0.06328317 Max(abs(heq)):  0
## par:  0.0984953 0.31796 0.12684 0.0905683 0.0632832 0.292782
## fval =   0.04979
##
## Outer iteration:  4
## Min(hin):  0.06370995 Max(abs(heq)):  0
## par:  0.0992667 0.3215 0.12803 0.0908713 0.0637099 0.295607
## fval =   0.0507
##
## Outer iteration:  5
## Min(hin):  0.06373212 Max(abs(heq)):  0
## par:  0.0993534 0.321807 0.12813 0.0909999 0.0637321 0.295876
## fval =   0.0508
##
## Outer iteration:  6
## Min(hin):  0.06375345 Max(abs(heq)):  0
## par:  0.0993555 0.32185 0.128146 0.0909926 0.0637535 0.295892
## fval =   0.05081
##
## Outer iteration:  7
## Min(hin):  0.06375438 Max(abs(heq)):  0
## par:  0.0993536 0.321867 0.128147 0.0909839 0.0637544 0.295892
## fval =   0.05081
##
## Outer iteration:  8
## Min(hin):  0.06375432 Max(abs(heq)):  0
## par:  0.0993526 0.321873 0.128147 0.0909804 0.0637543 0.295892
## fval =   0.05081
##
```

```r
# Print the return and risk of the estimated minimum variance portfolio:
Rmin <- tmp$par %*% R
print(Rmin)
```

```
##           [,1]
## [1,] 0.2773406
```

```r
Smin <- sqrt(S_pf(tmp$par,Sigma))
print(Smin)
```

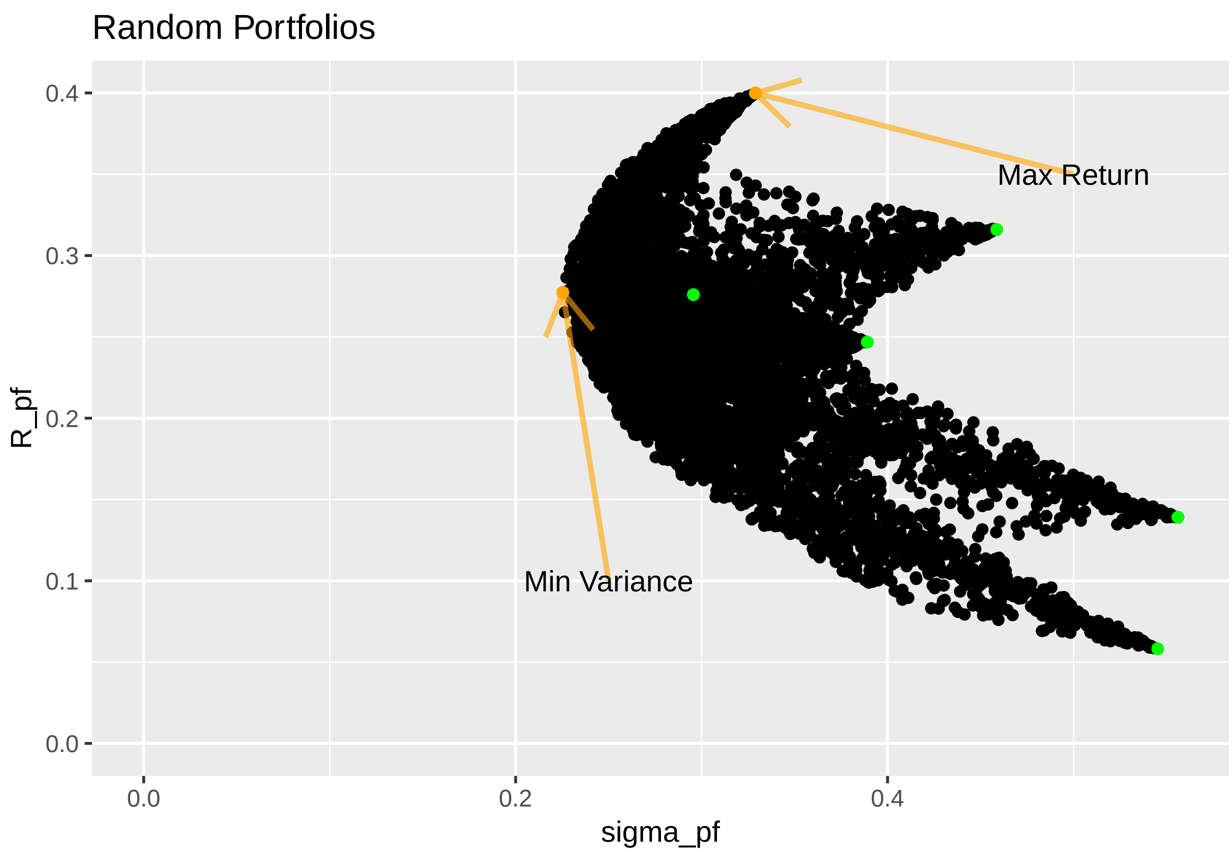
```
##           [,1]
## [1,] 0.2254052
```

```r
message(sprintf("Minimum variance portfolio:\n\tR = %5.4f\n\tSigma=%5.4f",Rmin,Smin))
```

```
## Minimum variance portfolio:
##  R = 0.2773
##  Sigma=0.2254
```

```r
# Plot the minimum variance portfolio (orange):
p <- p + geom_point(x=c(Smin),y=c(Rmin),color="orange") +
  annotate("segment", x = 0.25, xend = Smin,
                y = 0.1, yend = Rmin, colour = "orange",
                size=1, alpha=0.6, arrow=arrow()) +
  annotate("text", x = c(0.25), y = c(0.1),
             label = c("Min Variance") ,
             color="black", size=4 , angle=0)
```

```
#  The plot the maximim return protfolio:
wmax <- rep(0,length=n)
wmax[which.max(R)] <- 1
Rmax <- R_pf(wmax,R)
Smax <- sqrt(S_pf(wmax,Sigma))
p <- p + geom_point(x=c(Smax),y=c(Rmax),color="orange") +
  annotate("segment", x = 0.5, xend = Smax,
                  y = 0.35, yend = Rmax, colour = "orange",
                  size=1, alpha=0.6, arrow=arrow()) +
  annotate("text", x = c(0.5), y = c(0.35),
             label = c("Max Return") ,
             color="black", size=4 , angle=0)

print(p)
```



We finally have a feasible region of portoflios, $[\underline{R_0}, \bar{R_0}]$, over which we can solve for a nexus of efficient portfolios.

3.3.2 Solve and Plot the Efficient Frontier

The efficient frontier for this set of stocks ranges from the minimum variance to the maximum return protfolios. To solve for the efficint frontier, we recursively solve the following problem:

$$\min_{\mathbf{w}} \mathbf{w}'\mathbf{\Sigma}\mathbf{w}$$

Subject to:

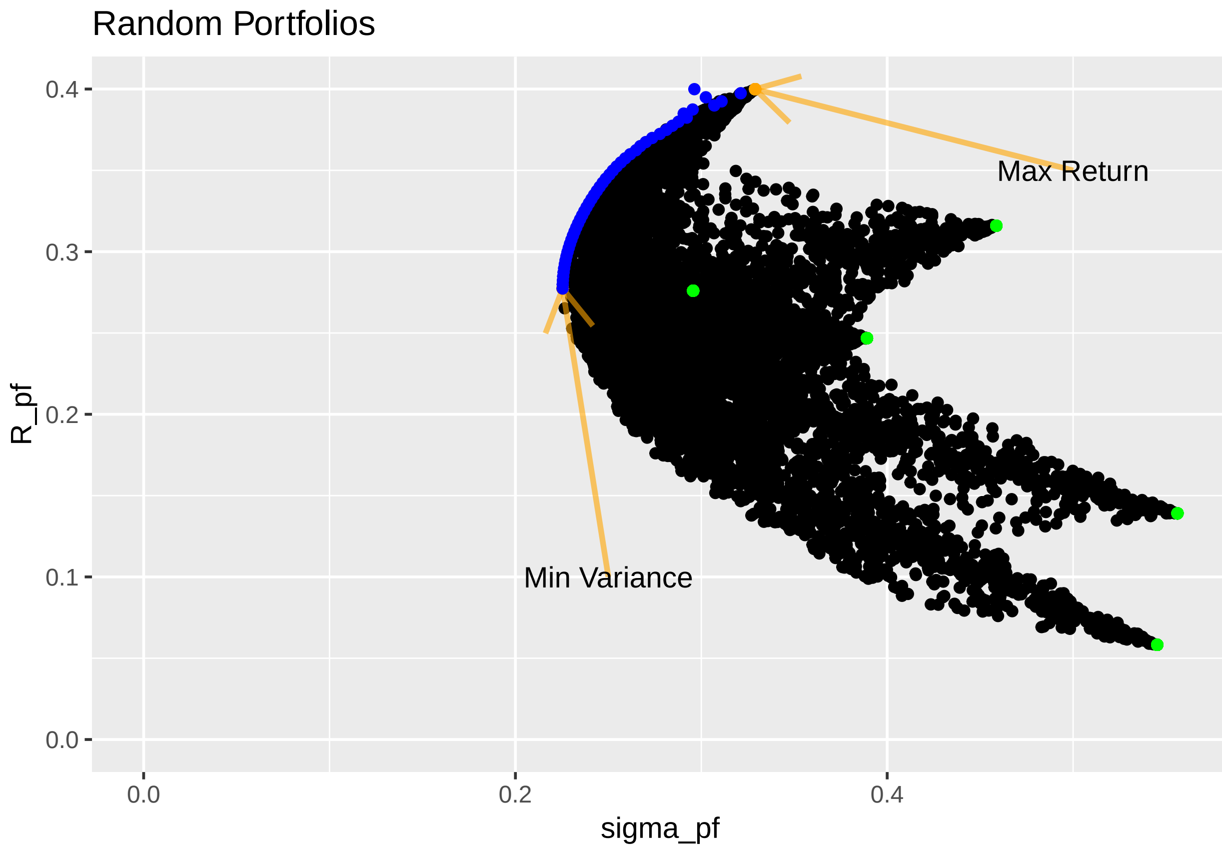
1. No shorting: $w_i \geq 0$
2. Wealth constraint: $\sum_{i=1}^{n} w_i = 1$
3. Minimum return: $\mathbf{w}'\mathbf{R} \geq R_i$

where R_i is a point on the interval $[\underline{R_0}, \bar{R_0}]$. Implement the loop in R:

```r
# --------------Efficient Frontier----------------
# Constraints wi >= 0
# Set this in the loop because it cahnges every time

# sum(w) = 1
heq <- function(x){
  h <- sum(x) - 1
  return(h)
}

# Set the number of portoflios along [Rmin,Rmax]
npf <- 50
soln <- as.data.frame(matrix(NA,ncol=2+n,nrow=npf))
Rconst <- seq(max(R),Rmin,length=npf)

# Objective function (minimize variance)
eval_f <- function(x){
  s <- (t(x) %*% Sigma %*% x)
  return(s)
}

# Gradient of the objective function (if you do not
# define it, a numerical approximation will be used)
eval_g <- function(x){
  g <- 0.5*t(x)%*%Sigma
  return(g)
}


# Loop through to solve:
outMat <- as.data.frame(matrix(NA,ncol=9,nrow=npf))
outWts <- as.data.frame(matrix(NA,ncol=(n+1),nrow=npf))

# Set a starting value to a feasible portfolio
# (Note that if the feasible region changes throughout the
# loop, then you will need to change x0 in the loop)
x0 <- rep(0,length=n)
x0[which.max(R)] <- 1

for (i in 1:npf){
  hin <- function(x){
    eps <- 1e-6
    h <- rep(NA,1)
    # Define no shorting constraints (the eps is to
    # mitigate the impact of rounding erros)
    for (j in 1:length(x)){
```

```r
      h[j] <- x[j] + eps
    }
    # Define the return constraint (Note - this is
    # the only element that changes in the loop)
    h[length(x)+1] <- t(x)%*%R - Rconst[i] + eps
    return(h)
  }
  # Solve the problem (the tryCatch() assures the
  # code will run if a single iteration returns an
  # error)
  tryCatch({
  tmp <- constrOptim.nl(par=x0, fn=eval_f,
                        heq=heq, hin=hin,
                        "control.outer"=list("trace"=FALSE),
                        "control.optim"=list("reltol"=1e-12,
                                             "trace"=0))
  },error=function(e){cat("ERROR :",conditionMessage(e), "\n")})

  # Save the resulting weight vector
  wtmp <- tmp$par

  # Save the results
  soln[i,2] <- sqrt(S_pf(wtmp,Sigma))
  soln[i,1] <- R_pf(wtmp,R)
  soln[i,c(3:(n+2))] <- wtmp
  message(sprintf("%d. %6.5f\t%6.5f\t%6.5f\t%d\t%6.5f\t%8.7f\t%d\t%d",
                  i,Rconst[i],soln[i,1],soln[i,2],
                  tmp$outer.iterations,sum(wtmp),
                  Rconst[i]-soln[i,1],
                  tmp$convergence,
                  tmp$counts[1]))
  outMat[i,] <- c(i,Rconst[i],soln[i,1],soln[i,2],
                  tmp$outer.iterations,sum(wtmp),
                  Rconst[i]-soln[i,1],
                  tmp$convergence,
                  tmp$counts[1])
  outWts[i,] <- c(i,tmp$par)

  # x0 <- wtmp
}
```

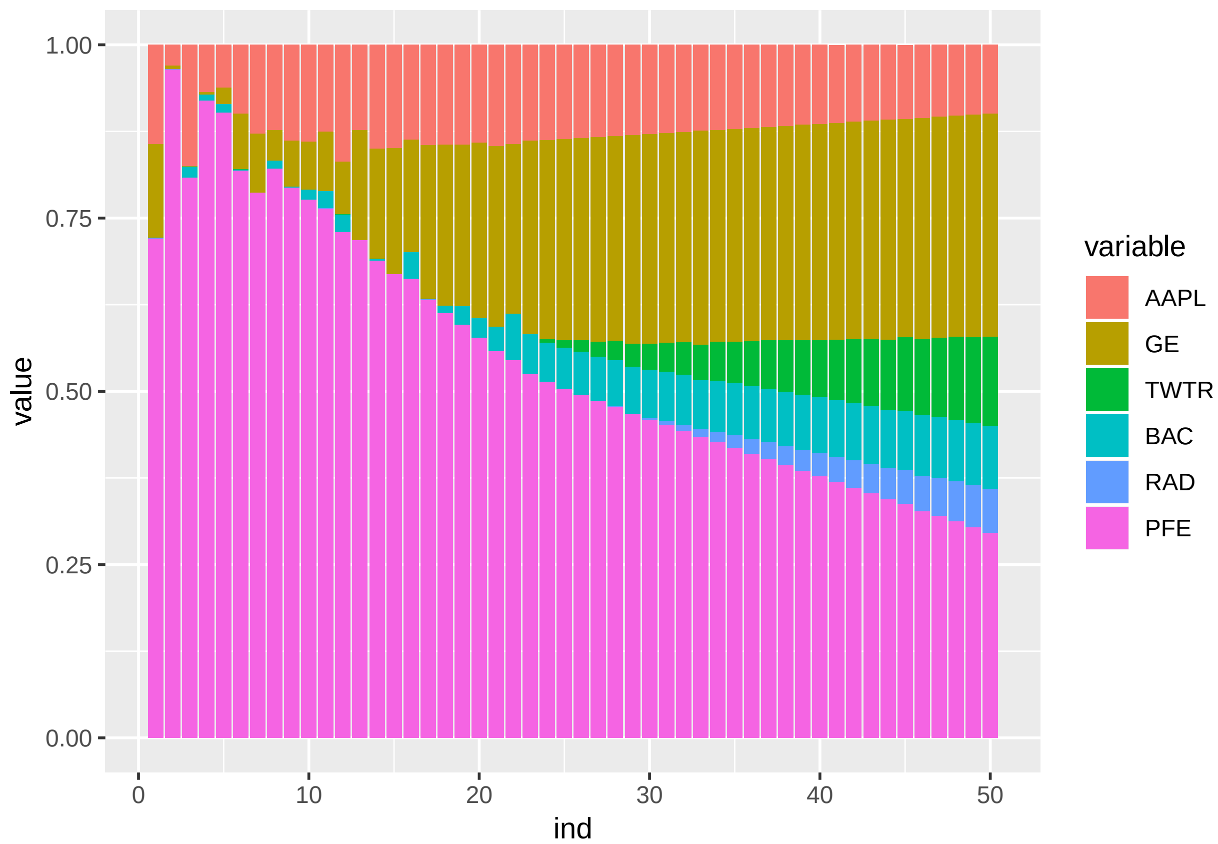
```r
# Add results to the plot:
p <- p + geom_point(data=soln, aes(x=soln[,2],y=soln[,1]),color="blue")
print(p)
```



In the above plot, the blue dots are the "efficent portfolios." Some of the "efficient" portfolios appear to be suboptimal to the random protfolios, that is an artifact of the optimization method we implemented. A more stringent optimization routine that adequatley handled equaltiy and perhaps non-linear constraints would likely yield better results.

A final step is to graph the portfolio composition across the frontier:

```
# Plot the bar chart:
names(outWts) <- c("i",tics)
library("reshape")
wts <- melt(cbind(outWts[,2:(n+1)], ind=outWts$i),
            id.vars = c('ind'))

wts$value[wts$value<0] <- 0
wbar <- ggplot(data = wts, aes(x = ind, y = value, fill = variable)) +
  geom_bar(position = "fill", stat="identity")
print(wbar)
```



3.4 Adding in a Risk Free Asset:

We could also include a risk free asset. In a more realistic setting you could download some historical data for a risk-free rate such as a short maturity US Treasury and use that data in the same way you did for the stock returns above to compute the mean vector and covaraince matrix. However, when computing the correlation between the equities and the risk free rate, the risk free rate is generally subtracted from the individual asset returns. As such, the correlation between the risk free asset and the individual assets will be equal to zero by definition. Since the asset is "risk-free" we can further assume that it has no volatility.

The ultimate result of this is that we can augment the mean return vector with the risk free rate and augment the covariance matrix with a outer column and row of zeros.

```
# Add a risk-free asset:
# 1. Return Vector
rf <- 0.02
R <- c(R,rf)
tics <- c(tics,"RF")
names(R) <- tics
print(R)
```

```
##       AAPL         GE       TWTR        BAC        RAD        PFE
## 0.31596838 0.27597612 0.05822827 0.24681113 0.13906825 0.39992949
##         RF
## 0.02000000
```

```
# 2. Covariance Matrix
Sigma <- rbind(Sigma,rep(0,length=n))
```

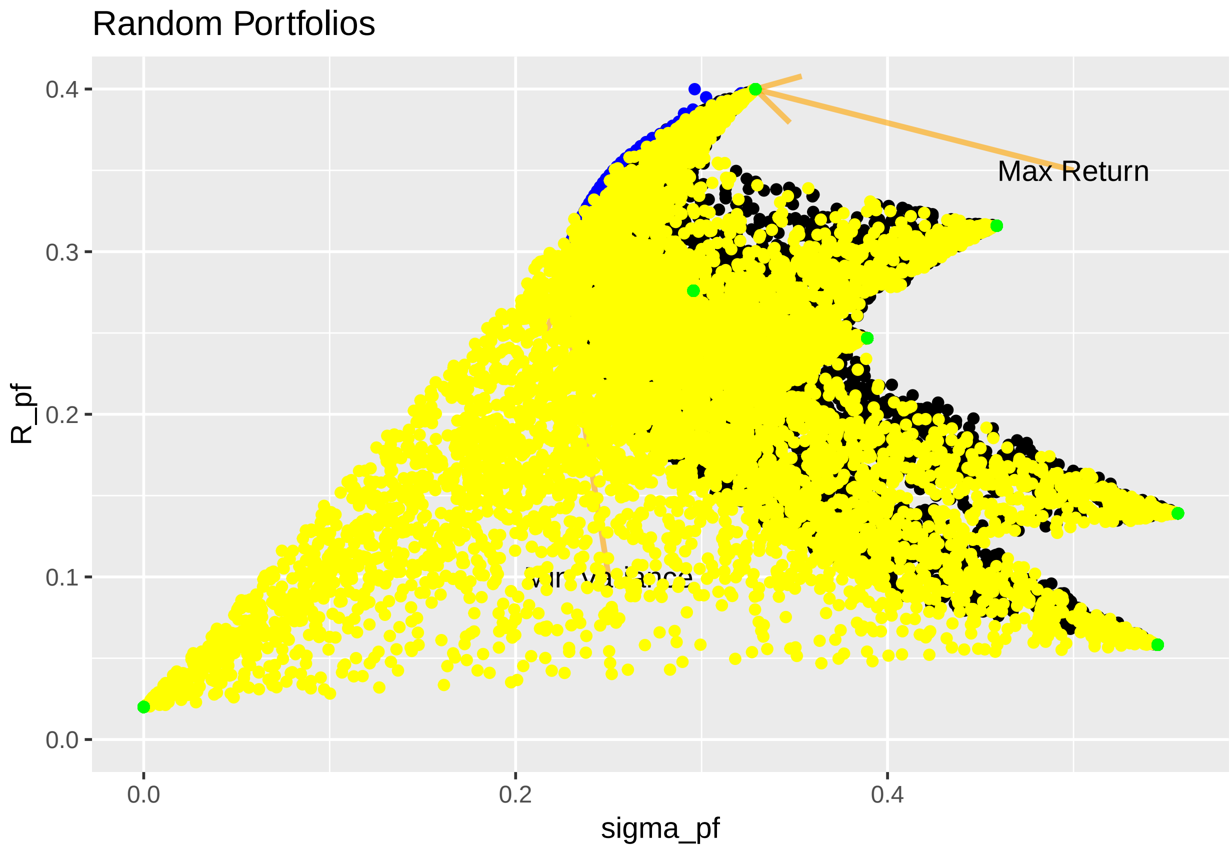
```r
Sigma <- cbind(Sigma,rep(0,length=(n+1)))
print(Sigma)
```

```
##            AAPL         GE        TWTR        BAC        RAD         PFE
## AAPL 0.21041200 0.04033009 0.022651962 0.04160559 0.03605614 0.026809059 0
## GE   0.04033009 0.08738836 0.014068513 0.05323977 0.03973695 0.032085231 0
## TWTR 0.02265196 0.01406851 0.297323854 0.02372492 0.01281202 0.009954609 0
## BAC  0.04160559 0.05323977 0.023724923 0.15142032 0.05626598 0.030875443 0
## RAD  0.03605614 0.03973695 0.012812018 0.05626598 0.30929330 0.026861208 0
## PFE  0.02680906 0.03208523 0.009954609 0.03087544 0.02686121 0.108223506 0
##      0.00000000 0.00000000 0.000000000 0.00000000 0.00000000 0.000000000 0
```

```r
n<-length(R)
```

With the updated R and Σ values, we can repeat the above analysis to plot the revised efficient frontier.

```r
N <- n*1000
pfs <- as.data.frame(matrix(NA,ncol=2+n,nrow=N))
for (i in 1:N){
  tmp <- randPfs(R,Sigma)
  pfs[i,] <- matrix(c(tmp$R,sqrt(tmp$S),t(tmp$w)))
}

# Plot the results:
library("ggplot2")
# p <- p + ggplot(data=pfs,aes(x=V2,y=V1)) +
#   geom_point(color="yellow") +
#   ylab(label="R_pf") +
#   xlab(label="sigma_pf") +
#   ggtitle("Random Portfolios")

p <- p +
  geom_point(data=pfs,aes(x=V2,y=V1),color="yellow") +
  ylab(label="R_pf") +
  xlab(label="sigma_pf") +
  ggtitle("Random Portfolios")

# Plot the 100% portfolios:
for (i in 1:n){
  wtmp <- rep(0,length=n)
  wtmp[i]=1
  p <- p + geom_point(x=c(sqrt(S_pf(wtmp,Sigma))),y=c(R_pf(wtmp,R)),
                      color="green")
}
print(p)
```



In the above plot, the yellow dots represent a set of random portfolios ranging from a minimum variance of zero to the same maximum return we had above (equal to the return on the highest yielding stock). Notice the straight light that is tangent to the previous efficient frontier.

Finally, we can solve for the neww efficient frontier with the risk free asset as follows.

First, set the the linear inequality constraints as follows:

```
# ------Set the Constraints--------
# Constraints wi >= 0
hin <- function(x){
  h <- x
  return(x)
}

# sum(w) = 1
heq <- function(x){
  h <- sum(x) - 1
  return(h)
}
```

Next set the objective funtion:

```
# ------Objective Function: Minimize Variance-----
eval_f <- function(x){
  s <- t(x) %*% Sigma %*% x
  return(s)
}
```

Before, we had to solve fo teh minimum variance portoflio. Now, we know that by definition it will be the case of 100% allocation to the risk free asset.

```
# Minimum variance portfolio:
Rmin <- rf
Smin <- 0
```

We can finally solve for the new efficient frontier.

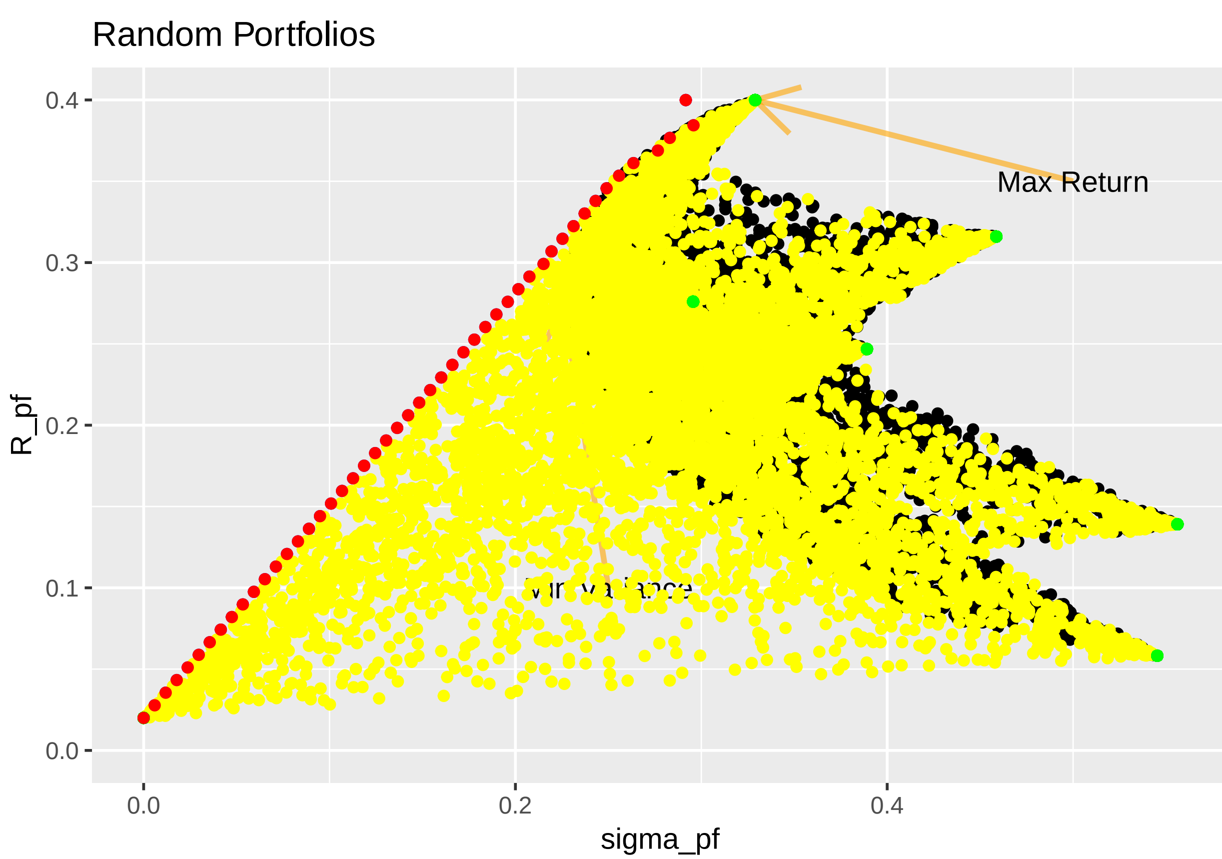
```
# --------------Efficient Frontier----------------

# Set the number of portoflios along [Rmin,Rmax]
npf <- 50
soln <- as.data.frame(matrix(NA,ncol=2+n,nrow=npf))
Rconst <- seq(max(R),Rmin,length=npf)

eval_f <- function(x){
  s <- (t(x) %*% Sigma %*% x)
  return(s)
}

eval_g <- function(x){
  g <- 0.5*t(x)%*%Sigma
  return(g)
}


# Loop through to solve:
outMat <- as.data.frame(matrix(NA,ncol=9,nrow=npf))
outWts <- as.data.frame(matrix(NA,ncol=(n+1),nrow=npf))

Rmax <- which.max(R)
x0 <- rep(0,length=n)
x0[which.max(R)] <- 1

for (i in 1:npf){
  hin <- function(x){
    eps <- 1e-6
    h <- rep(NA,1)
    for (j in 1:length(x)){
      h[j] <- x[j] + eps
    }
    h[length(x)+1] <- t(x)%*%R - Rconst[i] + eps
    return(h)
  }
  tryCatch({
  tmp <- constrOptim.nl(par=x0, fn=eval_f,
                        heq=heq, hin=hin,
                        "control.outer"=list("trace"=FALSE),
                        "control.optim"=list("reltol"=1e-12,
                                             "trace"=0))
  },error=function(e){cat("ERROR :",conditionMessage(e), "\n")})

  wtmp <- tmp$par
```

```
  soln[i,2] <- sqrt(S_pf(wtmp,Sigma))
  soln[i,1] <- R_pf(wtmp,R)
  soln[i,c(3:(n+2))] <- wtmp
  message(sprintf("%d. %6.5f\t%6.5f\t%6.5f\t%d\t%6.5f\t%8.7f\t%d\t%d",
                  i,Rconst[i],soln[i,1],soln[i,2],
                  tmp$outer.iterations,sum(wtmp),
                  Rconst[i]-soln[i,1],
                  tmp$convergence,
                  tmp$counts[1]))
  outMat[i,] <- c(i,Rconst[i],soln[i,1],soln[i,2],
                  tmp$outer.iterations,sum(wtmp),
                  Rconst[i]-soln[i,1],
                  tmp$convergence,
                  tmp$counts[1])
  outWts[i,] <- c(i,tmp$par)

  # x0 <- wtmp
}
```

```
## ERROR : initial value in 'vmmin' is not finite
```

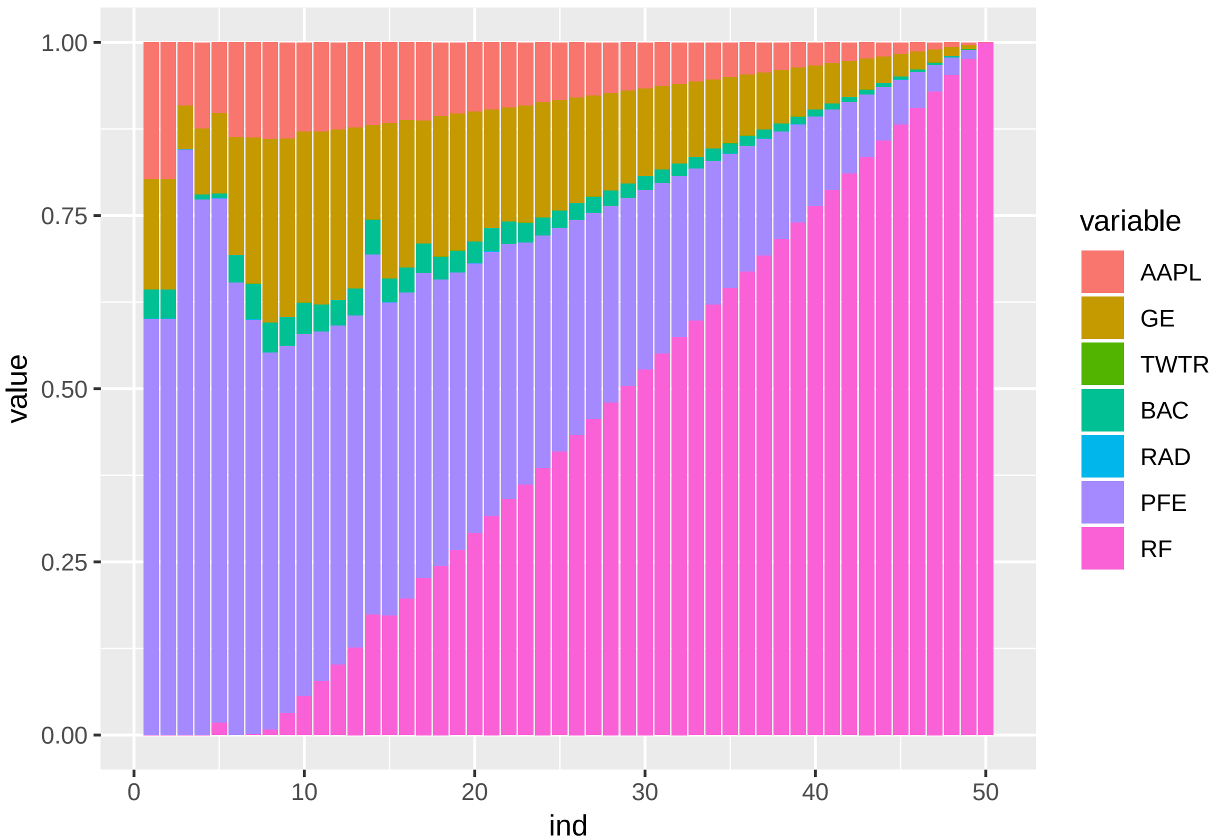
Plot:

```
# Add results to the plot:
p <- p + geom_point(data=soln, aes(x=soln[,2],y=soln[,1]),color="red")
print(p)
```



We can also plot the portfolio composition over time.

```
# Plot the bar chart:
names(outWts) <- c("i",tics)
library("reshape")
wts <- melt(cbind(outWts[,2:(n+1)], ind=outWts$i),
            id.vars = c('ind'))

wts$value[wts$value<0] <- 0
wbar <- ggplot(data = wts, aes(x = ind, y = value, fill = variable)) +
  geom_bar(position = "fill", stat="identity")
print(wbar)
```

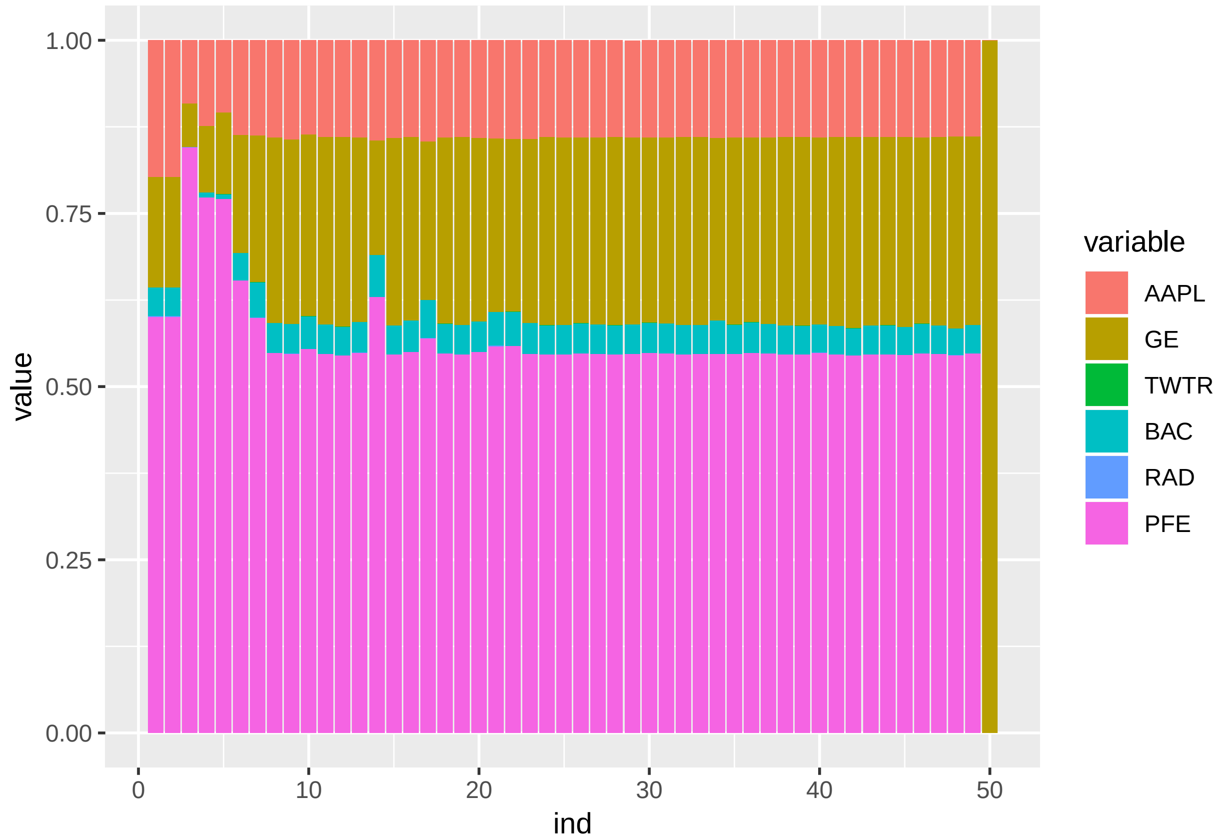


Notice that the new efficient frontier is the tangent portfolio. The point at which it intersects the old efficient frontier (the one with no risk free asset) is known as the market portfolio. In portfolio theory, the tangent portfolio is made up of linear combinations of the tangent portfolio with the risk free asset ranging from 100% allocation to the risk free asset to 100% allocation to the market portfolio. Notice how the weight n the risk-free asset decreases linearly as you move from left to right until you hit the market portoflio around portfolio "85." If shorting the risk free rate were allowed, the tangent line would continue indefinitely in a straight line with weights on the risk free asset becoming increasingly negative and weights in the market portfolio becoming more and more leveraged (i.e., weight > 100% of total wealth).

It is also interesting to repeat the above plot without the risk free asset to get the relative weights of the risky assets.

```
# Plot the bar chart:
names(outWts) <- c("i",tics)
library("reshape")
```

```
wts <- melt(cbind(outWts[,2:(n+1)], ind=outWts$i),
            id.vars = c('ind'))

wts$value[wts$value<0] <- 0
wts_noRF <- wts[!(wts$variable=="RF"),]
wbar <- ggplot(data = wts_noRF, aes(x = ind, y = value, fill = variable)) +
  geom_bar(position = "fill", stat="identity")
print(wbar)
```



If the optimization routine were "exact", we would see that the portoflio compsition without the risk free asset would be perfectly constant from zero to the point of the "market portfolio around portfolio"85" or so. The reason it is not in our case is because of some limitations in our optimizatin method (most notably because we are not using exact equality contraints.)

3.4.1 Relax the Shorting Constraint on the Risk Free Asset

If we relax the shorting constraint on the risk free asset, the tangency line will extend beyond what we saw in the previous example. All we need to do is elimiate the constraint $w_{rf} \geq 0$.

```
# --------------Efficient Frontier----------------
# ------Set the Constraints--------
# Constraints wi >= 0
# Set in loop

# sum(w) = 1
heq <- function(x){
```

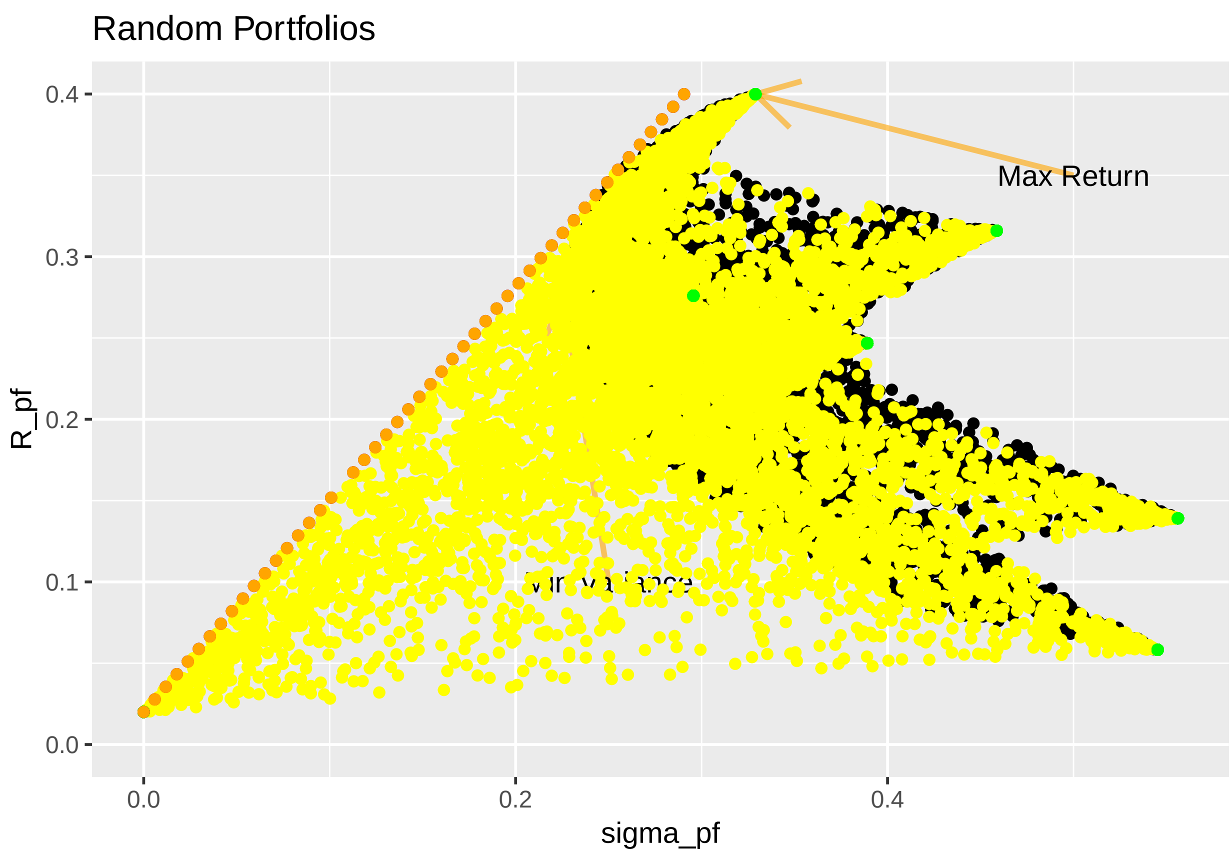
```r
  h <- sum(x) - 1
  return(h)
}

# Set the number of portoflios along [Rmin,Rmax]
npf <- 50
soln <- as.data.frame(matrix(NA,ncol=2+n,nrow=npf))
Rconst <- seq(max(R),Rmin,length=npf)

eval_f <- function(x){
  s <- (t(x) %*% Sigma %*% x)
  return(s)
}

eval_g <- function(x){
  g <- 0.5*t(x)%*%Sigma
  return(g)
}


# Loop through to solve:
outMat <- as.data.frame(matrix(NA,ncol=9,nrow=npf))
outWts <- as.data.frame(matrix(NA,ncol=(n+1),nrow=npf))

Rmax <- which.max(R)
x0 <- rep(0,length=n)
x0[which.max(R)] <- 1

for (i in 1:npf){
  hin <- function(x){
    eps <- 1e-6
    h <- rep(NA,1)
    for (j in 1:(length(x)-1)){
      h[j] <- x[j] + eps
    }
    h[length(x)] <- t(x)%*%R - Rconst[i] + eps
    return(h)
  }
  tryCatch({
  tmp <- constrOptim.nl(par=x0, fn=eval_f,
                        heq=heq, hin=hin,
                        "control.outer"=list("trace"=FALSE),
                        "control.optim"=list("reltol"=1e-12,
                                             "trace"=0))
  },error=function(e){cat("ERROR :",conditionMessage(e), "\n")})

  wtmp <- tmp$par

  soln[i,2] <- sqrt(S_pf(wtmp,Sigma))
  soln[i,1] <- R_pf(wtmp,R)
  soln[i,c(3:(n+2))] <- wtmp
  message(sprintf("%d. %6.5f\t%6.5f\t%6.5f\t%d\t%6.5f\t%8.7f\t%d\t%d",
                  i,Rconst[i],soln[i,1],soln[i,2],
```

```
                   tmp$outer.iterations,sum(wtmp),
                   Rconst[i]-soln[i,1],
                   tmp$convergence,
                   tmp$counts[1]))
  outMat[i,] <- c(i,Rconst[i],soln[i,1],soln[i,2],
                   tmp$outer.iterations,sum(wtmp),
                   Rconst[i]-soln[i,1],
                   tmp$convergence,
                   tmp$counts[1])
  outWts[i,] <- c(i,tmp$par)

  # x0 <- wtmp
}
```

```
## ERROR : initial value in 'vmmin' is not finite
```

Plot:

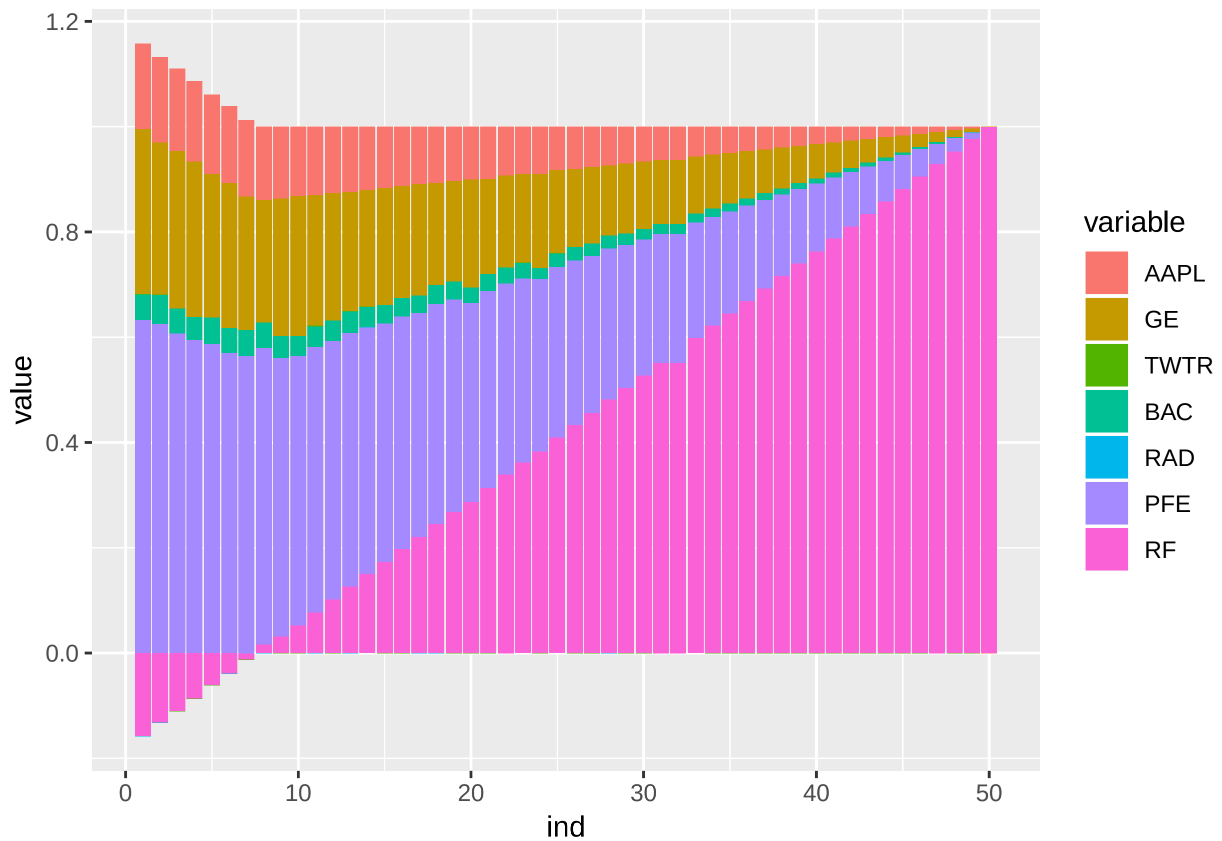
```
# Add results to the plot:
p <- p + geom_point(data=soln, aes(x=soln[,2],y=soln[,1]),color="orange")
print(p)
```



Again plot the portfolio composition both with and without the risk free asset.

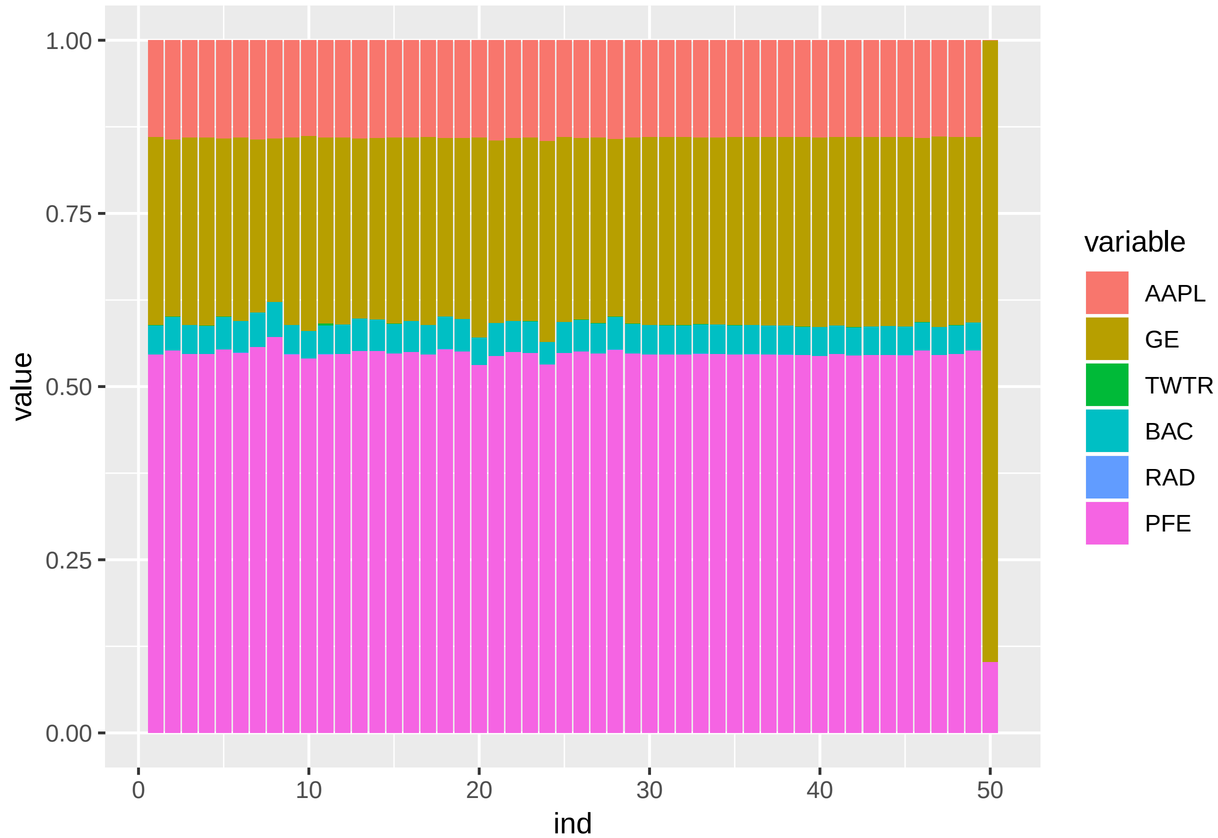
```
# Plot the bar chart:
names(outWts) <- c("i",tics)
library("reshape")
wts <- melt(cbind(outWts[,2:(n+1)], ind=outWts$i),
```

```r
             id.vars = c('ind'))

# wts$value[wts$value<0] <- 0
wbar <- ggplot(data = wts, aes(x = ind, y = value, fill = variable)) +
  geom_bar(stat="identity")
print(wbar)
```



Notice the negative weights on the far right. Also, notice how the weights on the risky assets sum to > 1 as shorting is allowed.

```r
# Plot the bar chart:
names(outWts) <- c("i",tics)
library("reshape")
wts <- melt(cbind(outWts[,2:(n+1)], ind=outWts$i),
            id.vars = c('ind'))

wts$value[wts$value<0] <- 0
wts_noRF <- wts[!(wts$variable=="RF"),]
wbar <- ggplot(data = wts_noRF, aes(x = ind, y = value, fill = variable)) +
  geom_bar(position = "fill", stat="identity")
print(wbar)
```



Again, we have the same issues with the tolerance of our optimization routine. If the optimization were more exact, then we would have exactly equivalent portolfios across the chart.

3.5 Maximizing Expected Utility

The last part of this document is to maximize quadratic utility. We define quadratic utility as follows:

$$U(x) = E[R_{pf}] - \frac{\lambda}{2}\sigma^2_{pf} \tag{7}$$

where λ is the risk aversion level.

If we want to maximize utility subject to the above constraints we simply change our objective function and solve.

```
# --------------Efficient Frontier----------------
# ------Set the Constraints--------
# Constraints wi >= 0
hin <- function(x){
  h <- x
  return(x)
}

# sum(w) = 1
heq <- function(x){
  h <- sum(x) - 1
```

```r
  return(h)
}
# Set a risk aversion level:
lambda <- seq(0,50,length=50)
soln <- as.data.frame(matrix(NA,ncol=2,nrow=length(lambda)))
x0 <- rep(1/n,length=n)
for (i in 1:length(lambda)){
  # Change the objective function
  eval_f <- function(x){
    s <- -t(x)%*%R + lambda[i]/2*(t(x) %*% Sigma %*% x)
    return(s)
  }


  # Loop through to solve:
  tryCatch({
  tmp <- constrOptim.nl(par=x0, fn=eval_f,
                        heq=heq, hin=hin,
                        "control.outer"=list("trace"=FALSE),
                        "control.optim"=list("reltol"=1e-12,
                                             "trace"=0))
  },error=function(e){cat("ERROR :",conditionMessage(e), "\n")})

  wtmp <- tmp$par
  soln[i,2] <- sqrt(S_pf(wtmp,Sigma))
  soln[i,1] <- R_pf(wtmp,R)
  message(sprintf("lambda = %3.1f: R = %5.4f, S = %5.4f",lambda[i],soln[i,1],soln[i,2]))
}

print(soln)
```

```
##            V1         V2
## 1  0.39992947 0.32897339
## 2  0.39992613 0.32896372
## 3  0.38547289 0.29648720
## 4  0.36639882 0.26894343
## 5  0.35590244 0.25763901
## 6  0.34870236 0.25140132
## 7  0.29919818 0.21353528
## 8  0.25930838 0.18302691
## 9  0.22939088 0.16014552
## 10 0.20612183 0.14234898
## 11 0.18750720 0.12811221
## 12 0.17227585 0.11646303
## 13 0.15958362 0.10675581
## 14 0.14884407 0.09854203
## 15 0.13963889 0.09150176
## 16 0.13166095 0.08540010
## 17 0.12468046 0.08006130
## 18 0.11852056 0.07535012
## 19 0.11304543 0.07116265
## 20 0.10814704 0.06741628
## 21 0.10373784 0.06404406
## 22 0.09974927 0.06099353
```

```
## 23 0.09612307 0.05822015
## 24 0.09281186 0.05568769
## 25 0.08977603 0.05336584
## 26 0.08698487 0.05123111
## 27 0.08440762 0.04925999
## 28 0.08202025 0.04743409
## 29 0.07980442 0.04573939
## 30 0.07774089 0.04416116
## 31 0.07581286 0.04268658
## 32 0.07401310 0.04131009
## 33 0.07232417 0.04001838
## 34 0.07073856 0.03880568
## 35 0.06924491 0.03766331
## 36 0.06783664 0.03658624
## 37 0.06650749 0.03556968
## 38 0.06525042 0.03460826
## 39 0.06405906 0.03369709
## 40 0.06292742 0.03283159
## 41 0.06185405 0.03201065
## 42 0.06083116 0.03122834
## 43 0.05985850 0.03048443
## 44 0.05893188 0.02977573
## 45 0.05804618 0.02909834
## 46 0.05719969 0.02845093
## 47 0.05639076 0.02783225
## 48 0.05561531 0.02723917
## 49 0.05487275 0.02667124
## 50 0.05416132 0.02612714
```

```
# Add results to the plot:
# p <- p + geom_point(aes(x=c(soln[,2]),y=c(soln[,1])),
#                      shape=9,color="black")
# print(p)
```

4 References