Data 609: Final Project

Basic Portfolio Optimization

Aaron Grzasko

12/2/2017

Contents

Overview

We have nine, passively-managed fund options available for our retirement account:

ticker	style	name
VLCAX	large cap blend	Vanguard Large Cap Index Fund
VVIAX	large cap value	Vanguard Value Index Admiral Fund
VBR	small cap value	Vanguard Small Cap Value ETF
VGSLX	real estate	Vanguard REIT Index Admiral Fund
EFV	intl large value	iShares MSCI EAFE Value ETF
SCZ	intl small cap blend	iShares MSCI EAFE Small Cap ETF
VEMAX	emerging mkts large blend	Vanguard Emerging Markets Stock Index Admiral Fund
SHV	short term treasury	iShares Barclays Short Treasury Bond ETF
BWZ	intl short-term treasury	SPDR Barclays Capital Short-Term International Treasury Bond ETF

We wish to determine the optimal allocation weighting of each fund within your portfolio, given our risk appetite and other portfolio constraints.

In the following sections, we use a variety of mathematical tools to perform the following tasks:

- Calculate the global minimum variance portfolio using Lagrange multipliers
- Determine an optimal portfolio using Monte Carlo simulation with bootstrapping
- Assume normally distributed returns to find an optimal portfolio
- Use linear programming techniques to arrive an optimal allocation
- Employ a quadratic programming approach to determine appropriate portfolio weights

Load Libraries

```r
library(knitr)
library(quantmod)
library(iterpc)
library(DT)
library(ggplot2)
library(ggthemes)
library(tidyr)
library(dplyr)
library(ggrepel)
library(lpSolve)
library(corrplot)
library(quadprog)
```

Data Retrieval

Historical fund price data are readily accessible using functions in the **quantmod** package. In the script below, we retrieve daily price data from Yahoo! Finance using the `getSymbols()` function. Because we are interested in approximating fund returns, we adjust historical prices for fund dividend distributions using the `adjustOHLC()` function. Finally, we calculate monthly returns using `monthlyReturn()`.

```r
# fund symbols
symbols <- c("VLCAX","VVIAX", "VBR", "VGSLX", "EFV", "SCZ", "VEMAX", "SHV", "BWZ")

# set historical start and end dates: pull 8 eight years of data,
beg = '2009-10-30' # last trading day in Oct 2009
end = '2017-10-31'

# save time series daily for each fund as separate time series object
getSymbols(symbols, src = "yahoo", from = beg, to = end)

# calculate monthly return data
ret_matrix <- matrix(NA,nrow = 96, ncol=0)
for (i in symbols) {
    temp_col <- as.numeric(monthlyReturn(adjustOHLC(eval(parse(text=i)),use.Adjusted=TRUE))[2:97])
    ret_matrix <- cbind(ret_matrix,i=temp_col)
}
colnames(ret_matrix) <- symbols
```

Our data set comprises 864 unique data points: 96 monthly return figures for nine separate funds.

Note: For this exercise, we are pulling only eight years of historical data because one of the available fund options, BWZ, formed in 2009.

Graphical Exploration

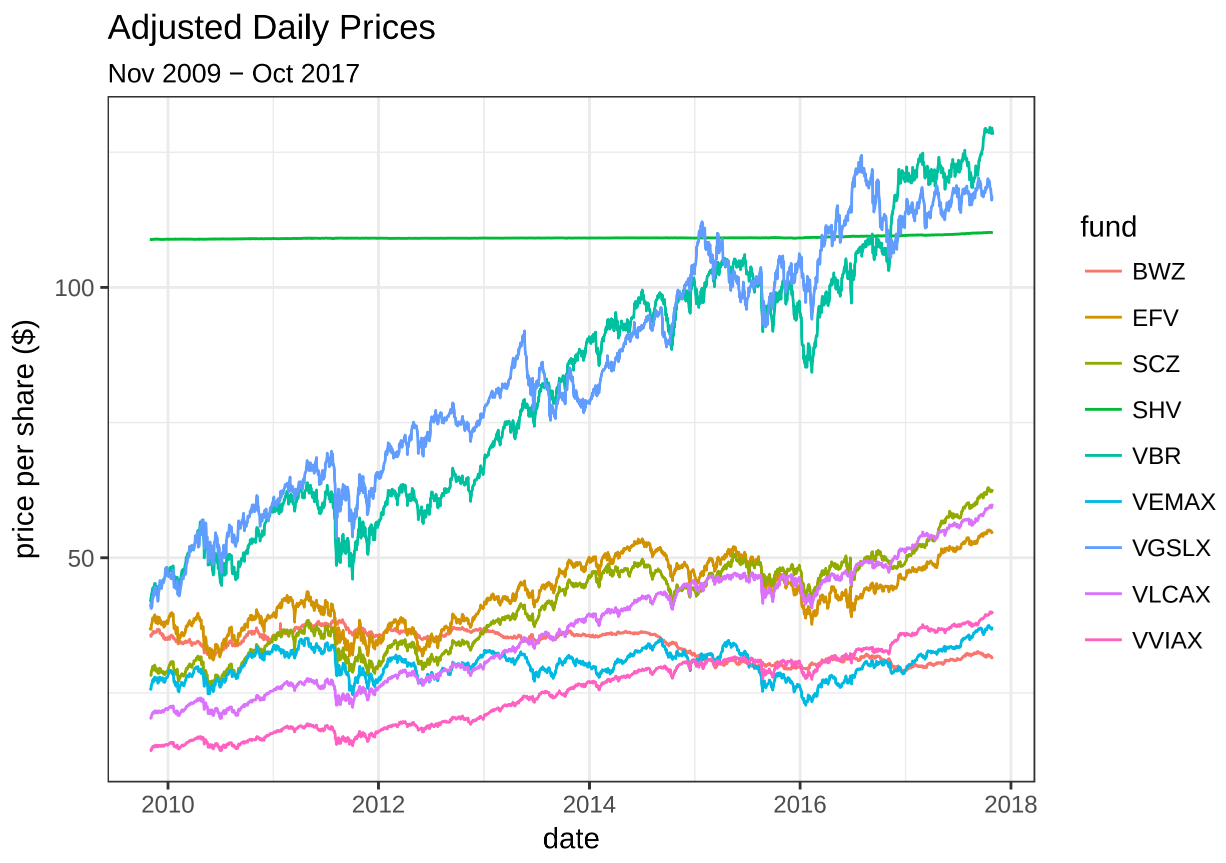
Price History

Using the price chart below, we make the following observations:

- Prices generally increased during the eight years examined, although we see significant short-term volatility.

- The small cap value (VBR) and real estate (VGSLX) funds experienced greater price appreciation vis-a-vis the other fund options.
- The price of US treasury fund (SHV) fund was very stable throughout the eight year period.

```
# get price history of each fund and store in df
price_df <- data.frame(date=as.Date(matrix(NA, nrow = 2014)))

for (i in symbols) {
    temp_col <- as.numeric(eval(parse(text=i))[,6])
    price_df[, i] <- temp_col
}

price_df[,'date'] <- as.Date(index(VVIAX))

price_df <- price_df %>% gather(fund, price,2:10) # long format

# plot price history
ggplot(price_df, aes(date,price, col=fund)) + geom_line() + theme_bw() +
    labs(title = 'Adjusted Daily Prices', y = "price per share ($)", subtitle = "Nov 2009 - Oct 2017")
```



If we believe past price movements are indicative of future trends and our primary concern is simply to maximize expected return, then the VBR and VGSLX funds may be good investment options. On the other hand, if our primary motive is to maintain stable value, then SHV could be a worthy option.

Return History

In this section, we present a series of monthly returns for each of the nine proposed fund options.

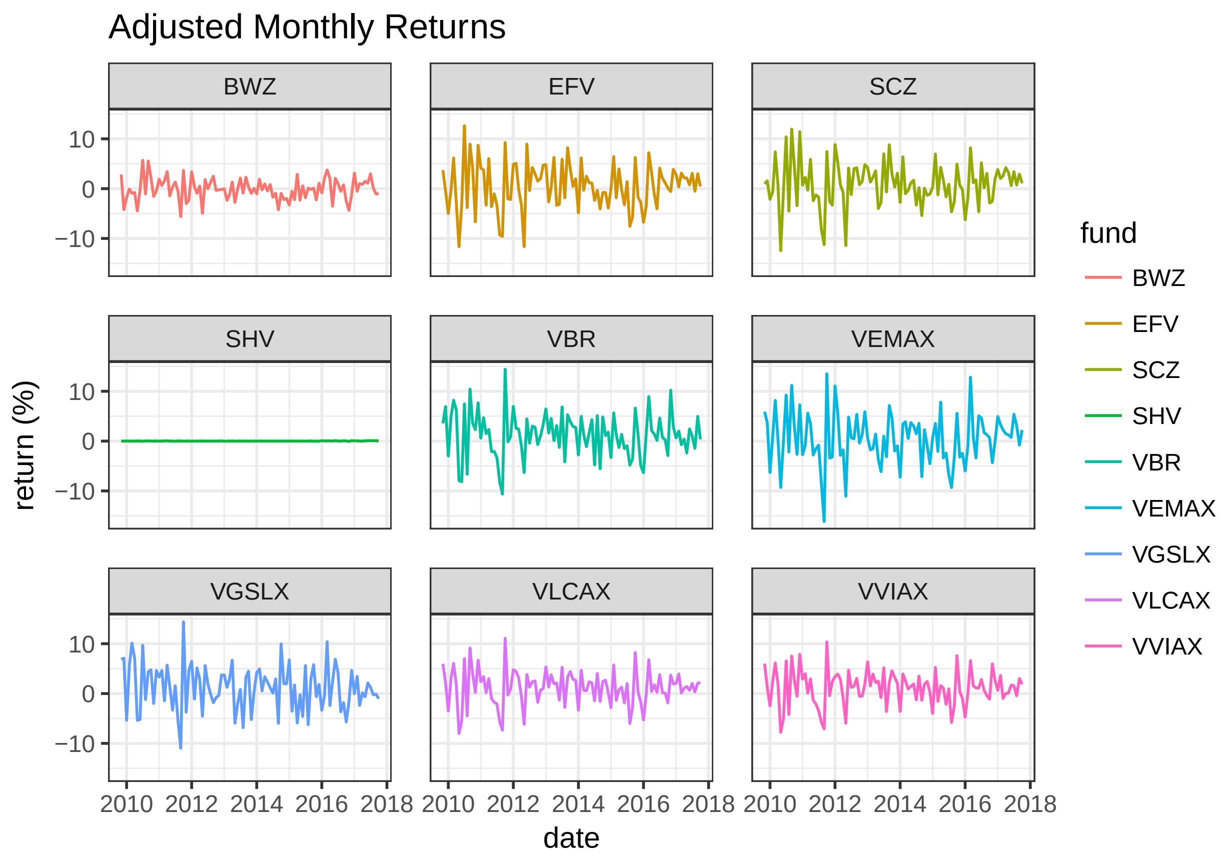
It's clear from these charts that the returns of the emerging markets fund (VEMAX) are more volatile than the the other potential options.

The two treasury funds, BWZ and SHV, appear to have stable returns–with the domestic fund, SHV, exhibiting virtually no volatility.

The other equity funds exhibit moderate return volatility during the period examined.

It's worth mentioning here that many asset models assume returns are independent, normally distributed random variables with constant mean and variance parameters. If returns are truly normally distributed, we would expect a historical return chart to resemble white noise, with no discernible pattern across time. In our charts, however, we notice heightened volatility in the 2011-2012 period that impact all equity funds in our hypothetical portfolio.

```
# monthly returns stored to df, long format
ret_df <- data.frame(date = seq(as.Date("2009/11/1"), as.Date("2017/10/1"), "months"), ret_matrix)
ret_df <- ret_df %>% gather(fund, return,2:10)

# plot
ggplot(ret_df, aes(date, return*100)) + geom_line(aes(col=fund)) +
  facet_wrap(~fund, ncol=3) + labs(y = "return (%)") +
  theme_bw() + theme(panel.spacing = unit(1, "lines")) +
  labs(title="Adjusted Monthly Returns")
```



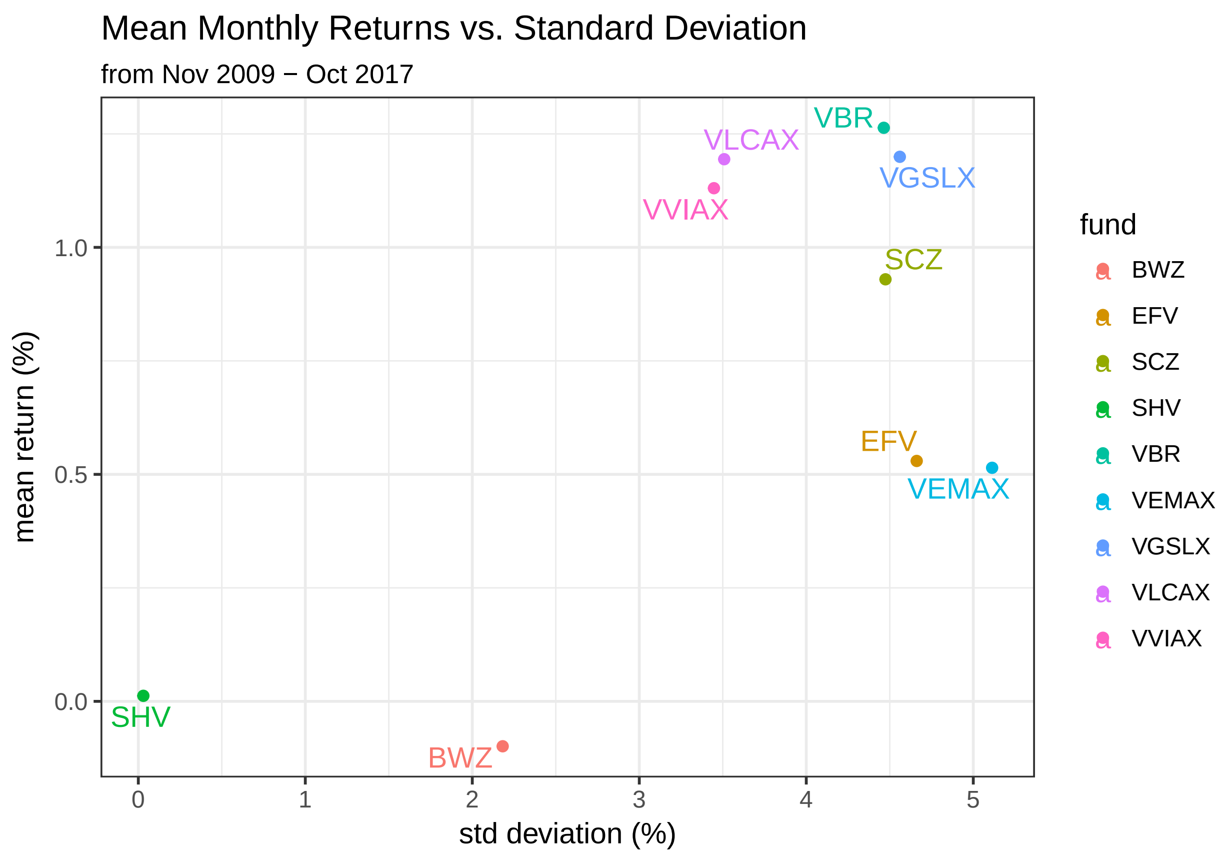
Risk vs. Reward

Later in our analysis, we will attempt to maximize expected portfolio return given a specified risk measure, such as standard deviation.

Before we begin examining characteristics of a portfolio of assets, let's examine the risk and return attributes of the individual components components under consideration.

We observe in the chart below that funds with high expected returns are generally associated with high volatility (i.e standard deviation).

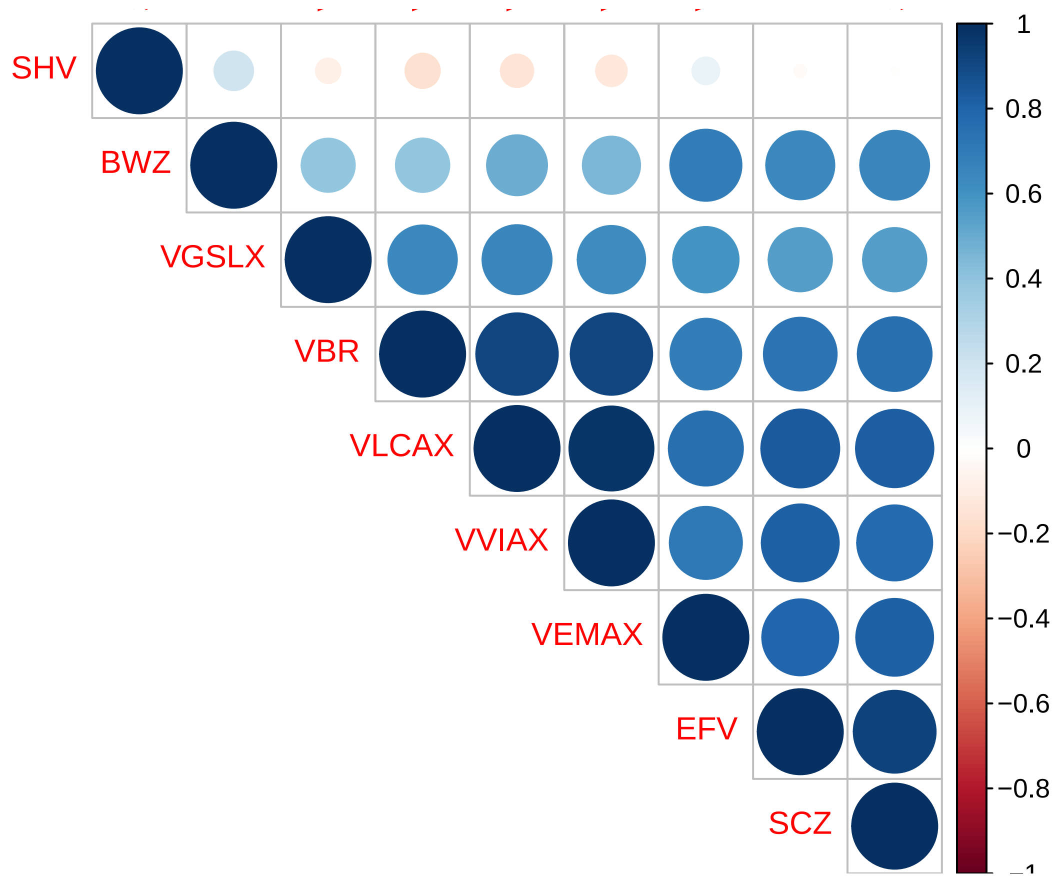
```r
# calculate means and standard deviation, and covariance of each fund
mean_vec <- apply(ret_matrix, 2, mean)
sd_vec <- apply(ret_matrix,2, sd)
cov_matrix <- cov(ret_matrix)

# risk reward df
rr_df <- data.frame(fund=names(mean_vec),return=mean_vec*100,risk=sd_vec*100, row.names=NULL )

# plot
ggplot(rr_df, aes(risk,return, col=fund)) + geom_point() + theme_bw() +
  geom_text_repel(mapping = aes(risk, return), data = rr_df, label = rr_df$fund) +
  labs(title = "Mean Monthly Returns vs. Standard Deviation",
       subtitle = "from Nov 2009 - Oct 2017", x="std deviation (%)", y= 'mean return (%)')
```



Fund Correlation

We conclude our graphical analysis by examining the pairwise correlations between our proposed fund options.

```r
# plot correlation matrix
corrplot(cor(ret_matrix, use="complete.obs", method='pearson'), type = "upper", order = "hclust")
```



Here are key takeaways from the correlation matrix:

- In general, the equity funds appear to be highly correlated with one another.
- The international treasury fund, BWZ, has moderate correlation with each of the equity funds, and low correlation with the U.S. Treasury fund.
- The domestic treasury fund, SHV, exhibits low correlation with all other funds.

Global Minimum Variance Portfolio

Suppose our primary goal is to minimize monthly portfolio variance, subject only to the constraint that the portfolio weights sum to one. This is nonlinear problem, as our objective function, variance, has a squared term. Fortunately, we can solve this problem analytically using the method of Lagrange multipliers.

Using formal notation, we state the problem as follows:

Let
$\sigma^2_{x,P}$ = portfolio variance
$\mathbf{x}$ = vector of portfolio weights
Σ = covariance matrix
λ = Lagrange multiplier

Objective:

$$\min_{\mathbf{x}} \sigma^2_{x,p} = \mathbf{x^T}\Sigma\mathbf{x}$$

such that

$$\mathbf{x^T 1} = 1$$

Set up the Lagrangian function as follows:

$$L(\mathbf{x}, \lambda) = \mathbf{x^T}\Sigma\mathbf{x} + \lambda(\mathbf{x^T 1} - 1)$$

Take the partial derivatives with respect to each asset, x_i and set equal to zero:

$$0 = \frac{\partial L}{\partial x_i} = 2x_i\sigma_i^2 + 2\sum_{i \neq j}\sigma_{ij} + \lambda$$

Also take the partial derivative with respect to λ and set equal to zero:

$$0 = \frac{\partial L}{\partial \lambda} = \mathbf{x^T 1} - 1$$

We now have 10 equations and 10 unknowns in the following form:

$$A\mathbf{z} = \mathbf{b}$$

where
$A = \begin{bmatrix} 2\Sigma & \mathbf{1} \\ \mathbf{1^T} & 0 \end{bmatrix}$, $\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix}$, and $\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$

Solve for $\mathbf{z}$:

$$\mathbf{z} = A^{-1}\mathbf{b}$$

Let's solve for $\mathbf{z}$ in R:

```r
# matrix A
A <- rbind(cbind(2 * cov_matrix, rep(1,nrow(cov_matrix))),c(rep(1, ncol(cov_matrix)),0))

# vector b
b <- c(rep(0, nrow(cov_matrix)),1)

# solve z
z <-solve(A) %*% b

# print porfolio weights and lambda parameter
z_df <- data.frame(matrix(sprintf("%.2f%%", z*100),1,10))
names(z_df) <- c(symbols, "lambda")
kable(z_df)
```

VLCAX	VVIAX	VBR	VGSLX	EFV	SCZ	VEMAX	SHV	BWZ	lambda
0.90%	-0.65%	0.14%	0.02%	0.11%	-0.11%	-0.22%	100.11%	-0.30%	-0.00%

Given our earlier graphical analysis, we are not surprised to see the vast majority of the portfolio being allocated to the the U.S Treasury fund (SHV), a fund with extremely low volatility. The solution to this exercise suggests that we should short four of the assets in the portfolio, and invest the proceeds from the short positions so that more than 100% of our original investment dollars are allocated to SHV.

Unfortunately, the proposed allocation is unrealistic, as we are investing for our retirement account (possibly a 401(k) or IRA account) where short selling most likely is prohibited.

Maximize Return Given Risk

Monte Carlo with Bootstrapping

We would now like to maximize expected portfolio return, given a specified maximum risk measure such as standard deviation.

Suppose we have no knowledge of constrained optimization techniques like linear programming. Assume also that we have limited knowledge of statistical distributions, such as the normal distribution.

We can can still attempt our optimize expected portfolio returns using Monte Carlo simulation and technique known as bootstrapping. Here is an outline of our procedure:

- Bootstrapping step: use sampling with replacement from the historical return data to generate 1,000 simulated return vectors. We simulate entire sets of historical monthly returns together, rather than attempting to simulate each asset independently. This is an important consideration because many of the funds' returns are highly correlated with each other.
- Use enumeration to generate all possible portfolio weightings.
  - This step can potentially be computationally expensive; so we make the simplifying assumption that our portfolio weighting percentages must be 0% or multiples of ten (e.g. 10%, 20%, etc.)
  - Given our desire to produce a diversified portfolio, we only enumerate over options where each asset has 40% or lower weighting.
- Calculate portfolio means and standard deviations for all possible portfolio weights.
- Return portfolio weights generating maximum expected portfolio return, given a specified upper bound on standard deviation.

Below is a summary of the optimization problem:

Let
$\mu_{p,x}$ = mean portfolio return
$\sigma_{p,x}$ = portfolio standard deviation
x_i = portfolio weight for asset i
S = constant, max allowed standard deviation

Objective:

$$\max_{\mathbf{x}} \mu_{p,x}$$

subject to

$$\sum_{i=1}^{N} x_i = 1$$

$$0 \leq x_i \leq 0.4 \text{ for all } i$$

$$100x_i \bmod 10 = 0 \text{ for all } i$$

$$\sigma_{p,x} \leq S$$

Now let's implement the procedure in R:

```r
# helper function: simulated returns for each asset using MC w/ bootstrapping
bs_sim <- function(num_sims, return_matrix) {

    return(return_matrix[sample(nrow(return_matrix),num_sims, replace=TRUE),])


}

# helper function:  enumerate over all possible portfolio weights
port_wts <- function(wt_vec, num_assets) {
  I <- iterpc(table(wt_vec), num_assets,replace=TRUE, ordered=TRUE)
  perms <- getall(I)
  row_sums <- apply(perms,1,sum)
  return(perms[row_sums == 1,]) # only return permutations where weights add to 1
}


# function to find max return given upper bound on sd
bs_max_ret <- function(sim_matrix, wt_matrix, max_sd) {
  sim_wt <- sim_matrix %*% t(wt_matrix)
  bs_mean_vec <- apply(sim_wt,2,mean)
   bs_sd_vec <- apply(sim_wt,2, sd)
  max_ret <- max(bs_mean_vec[bs_sd_vec <= max_sd])
  max_pos <- which(bs_mean_vec == max_ret & bs_sd_vec <= max_sd)
  wts <- as.numeric(wt_matrix[max_pos,])
  names(wts) <- symbols
  list(max_ret = max_ret, sd = bs_sd_vec[max_pos], weights=wts)

}
```

As an example, let's find the maximum expected return, given a maximum monthly standard deviation of 0.015:

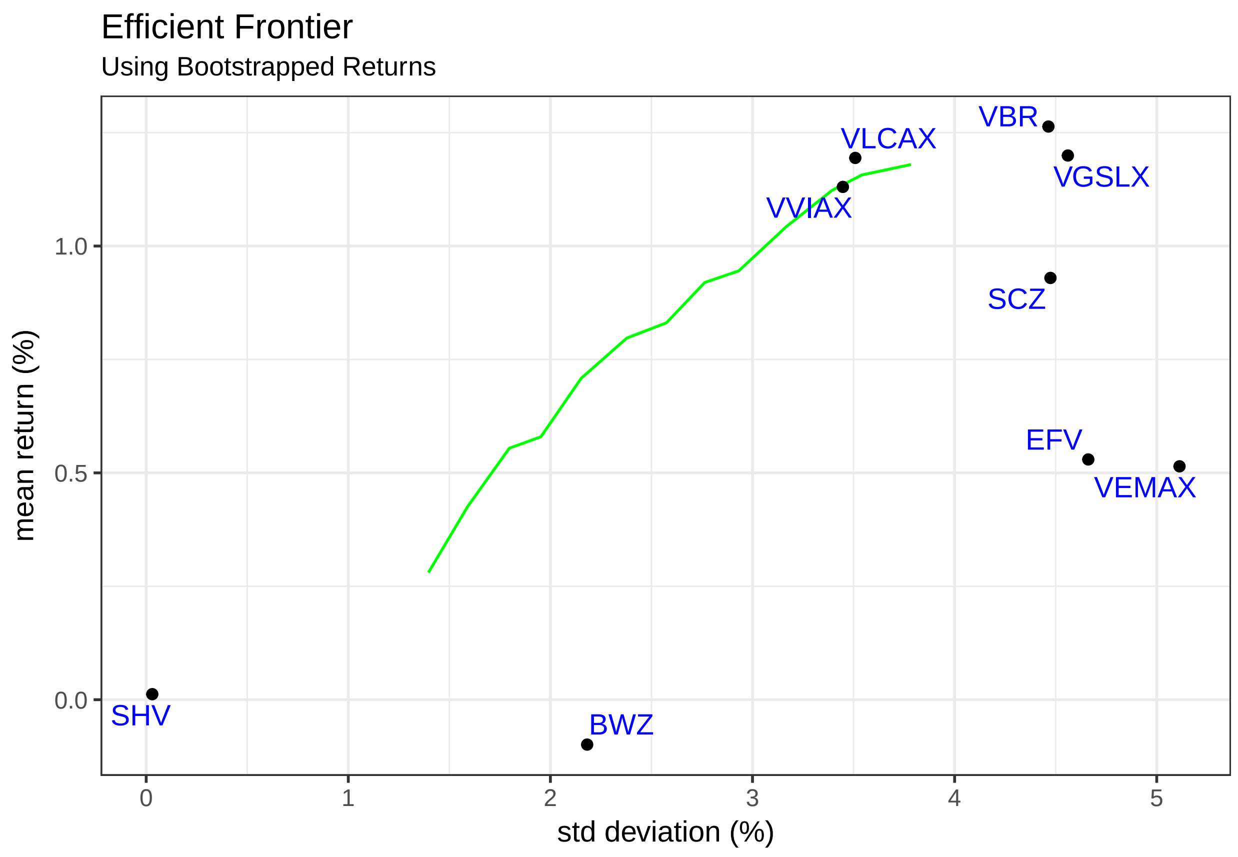
```r
# generate simulated asset returns
set.seed(1)
bs_sim_matrix <- bs_sim(1000,ret_matrix) # simulate individual asset returns
wt_matrix <- port_wts(c(0,0.1,0.2,0.3,0.4),9) # universe of potentail portfolio weights

bs_max_ret(bs_sim_matrix, wt_matrix, 0.015) # find max return
```

```
$max_ret
[1] 0.003017509

$sd
[1] 0.01471413

$weights
VLCAX VVIAX   VBR VGSLX   EFV   SCZ VEMAX   SHV   BWZ
  0.1   0.0   0.1   0.1   0.0   0.0   0.0   0.4   0.3
```

Given our low tolerance for monthly price volatility in the example above, it is not surprising that the optimal allocation requires 70% allocation to short-term treasury funds. Based on our simulated data, the best expected return we can achieve is 0.3% monthly, or 3.6% annually.

Finally, let's produce the efficient frontier using our Monte Carlo and bootstrapping procedure. The efficient frontier–see the green line in the chart below–provides the optimal return given a specified risk level.

```
# bootstrap helper function to loop through multiple risk objectives
eff_front_bs <- function(start, stop, by=0.01, sim=bs_sim_matrix, wt=wt_matrix) {
    df <- data.frame(risk=numeric(),return=numeric())
    for (i in seq(start,stop, by)) {
        results <- bs_max_ret(sim, wt, i)
        df <- rbind(df, data.frame(risk=results$sd*100,return=results$max_ret*100))

    }
    df
}

# calculate return and sd for given specified max risk
df <- eff_front_bs(0.014,0.04,0.002)

# plot
ggplot(df, aes(x=risk,y=return)) + geom_line(col="green") +
    geom_point(data = rr_df) +
    geom_text_repel(mapping = aes(risk, return), data = rr_df, label = rr_df$fund, col= "blue" ) + theme
    labs(title = "Efficient Frontier", subtitle = "Using Bootstrapped Returns",
         x="std deviation (%)", y = "mean return (%)")
```



Normally Distributed Returns

Let's say we want to solve the same optimization problem described in the Monte Carlo section, but now we assume returns are normally distributed.

Let
R_i = return for asset i
$\mathbf{R}$ = vector of returns, all assets $\mu_{p,x}$ = mean portfolio return $\sigma^2_{p,x}$ = portfolio variance
$\mathbf{x}$ = vector of portfolio weights
Σ = covariance matrix

$$R_i \sim iid\ N(\mu_i, \sigma_i^2)$$

$$cov(R_i, R_j) = \sigma_{ij}$$

The return of the portfolio is

$$\mu_{p,x} = E[\mathbf{x}'\mathbf{R}] = \mathbf{x}^{\mathbf{T}} E[\mathbf{R}] = \mathbf{x}^{\mathbf{T}} \mu$$

The variance of the weighted portfolio is:

$$\sigma^2_{p,x} = \mathbf{x}^{\mathbf{T}} \Sigma \mathbf{x}$$

Given a set of portfolio weights and the formulas above, we can calculate the portfolio mean and variance.

To calculate the optimal weights, we will use the brute-force enumeration procedure outlined in the previous section. We again will make the simplifying assumption that each fund's allocation percentage is between 0% and 40%, with only discrete increments of 10% possible.

Let's build a function to generate optimal portfolio allocations using normally distributed returns.

```r
# normal dist function to calculate maximum return, given max risk criteria
norm_max_ret <- function(ret_matrix, wt_matrix, max_sd){

  mean_vec <- apply(ret_matrix, 2, mean)
  cov_matrix <- cov(ret_matrix)


  p_mean_vec <- wt_matrix %*% mean_vec
  p_sd_vec <- numeric()
  for (i in 1: nrow(wt_matrix)){
      temp_sd <- sqrt(t(matrix(wt_matrix[i,])) %*% cov_matrix %*% wt_matrix[i,])
      p_sd_vec <- c(p_sd_vec, temp_sd)

  }

  max_ret <- max(p_mean_vec[p_sd_vec <= max_sd])
  max_pos <- which(p_mean_vec == max_ret & p_sd_vec <= max_sd)
  wts <- as.numeric(wt_matrix[max_pos,])
  names(wts) <- symbols
  list(max_ret = max_ret, sd = p_sd_vec[max_pos], weights=wts)

}
```

Here is the optimization output assuming a maximum standard deviation of 1.5%:

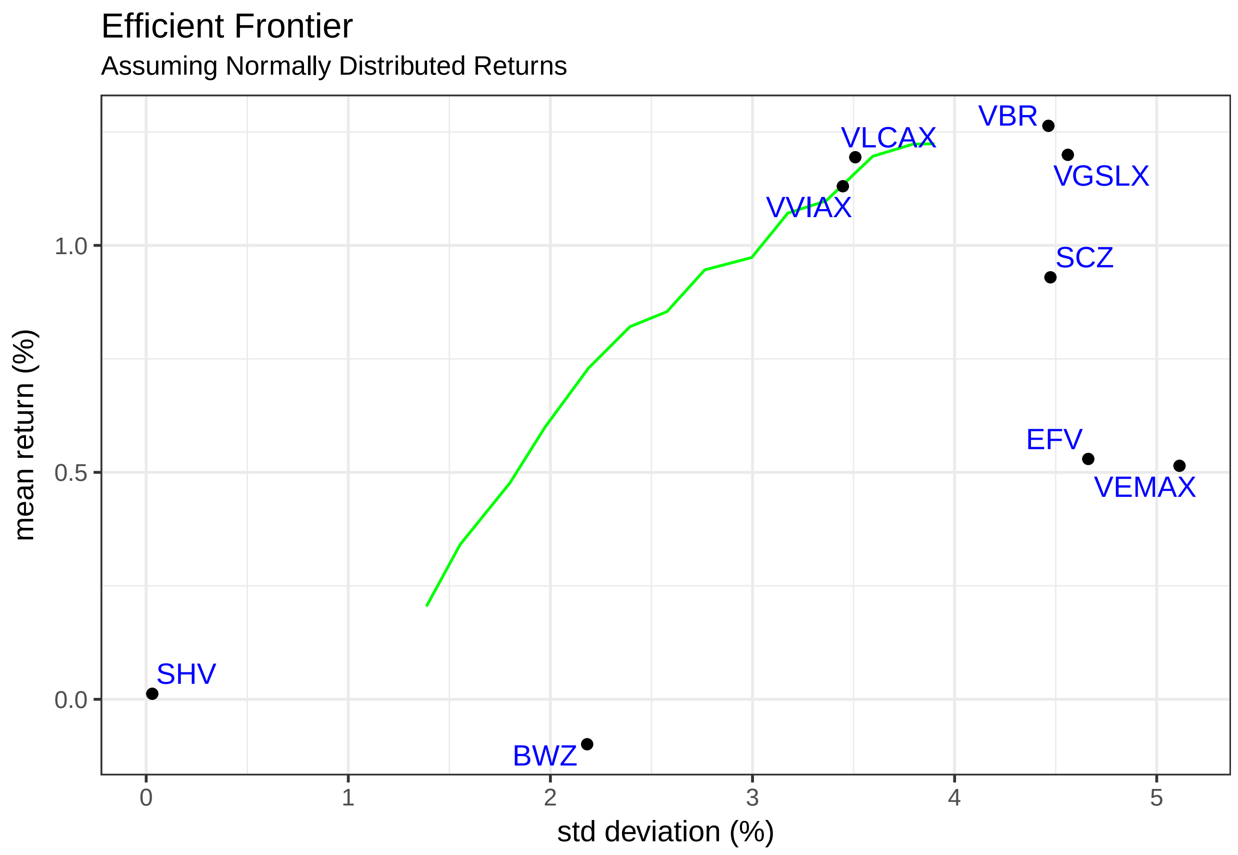
```r
norm_max_ret(ret_matrix,wt_matrix,0.015)
```

```
$max_ret
[1] 0.003339794

$sd
[1] 0.01497017

$weights
VLCAX VVIAX   VBR VGSLX   EFV   SCZ VEMAX   SHV   BWZ
  0.2   0.0   0.0   0.1   0.0   0.0   0.0   0.4   0.3
```

Using this procedure, we arrive at a solution similar to the Monte Carlo example result described earlier. The bond allocation is identical, but the equity allocation differs slightly.

We'll wrap up this section by plotting the efficient frontier using assuming normally distributed returns:

```r
# normal dist efficiient fronter helper function
eff_front_n <- function(start, stop, by=0.01,ret = ret_matrix, wt=wt_matrix) {
    df <- data.frame(risk=numeric(),return=numeric())
    for (i in seq(start,stop, by)) {
        results <- norm_max_ret(ret, wt, i)
        df <- rbind(df, data.frame(risk=results$sd*100,return=results$max_ret*100))

    }
    df
}


# calculate return and sd for multiple worst case sd scenarious
df <- eff_front_n(0.014,0.04,0.002)

# plot
ggplot(df, aes(x=risk,y=return)) + geom_line(col='green') +
    geom_point(data = rr_df) +
    geom_text_repel(mapping = aes(risk, return), data = rr_df, label = rr_df$fund, col= "blue" ) + theme
    labs(title = "Efficient Frontier", subtitle = "Assuming Normally Distributed Returns",
         x="std deviation (%)", y = "mean return (%)")
```



Linear Programming

Let's now take a more elegant approach to maximizing return given a set of risk constraints and other investment criteria.

Assume our technical expertise is limited to solving constrained optimization problems using standard linear programming techniques.

We need to modify our previous approaches where standard deviation was used in the problem constraint, as standard deviation is not a linear measure.

We begin by defining a constraint using mean absolute deviation (MAD):

$$\frac{1}{T}\sum_{t=1}^{T}\left|\sum_{i} x_i\big(R_i(t)-\mu_i\big)\right| \leq M \text{ where M is constant}$$

Although MAD is not linear, we can break out the inequality above into piecewise linear constraints:

$$\sum_{i} x_i\big(R_i(t)-\mu_{\mathbf{i}}\big) \leq y_t \text{ for all } t$$

$$\sum_{i} x_i\big(R_i(t)-\mu_{\mathbf{i}}\big) \geq -y_t \text{ for all } t$$

$$\frac{1}{T}\sum_{t=1}^{T} y_t \leq M \text{ where M is constant}$$

Here is the entire formulation of our optimization problem:

$$\max \mu_{p,x} = \sum_i x_i \mu_i$$

subject to

$$-y_t \leq \sum_i x_i \big(R_i(t) - \mu_{\mathbf{i}}\big) \leq y_t \text{ for all } t$$

$$\frac{1}{T}\sum_{t=1}^{T} y_t \leq M \text{ where M constant}$$

$$\sum_i x_i = 1$$

$$0 \leq x_i \leq 0.4, \text{ for all } i, \quad y_t \geq 0, \text{ for all } t$$

Note: Similar to previous sections, we're capping investment allocation in any given asset at 40%. However, we're now allowing the allocation to vary continuously between 0% and 40%.

Below we construct a function to solve the optimization problem using `lp()` from the **lpSolve** package. Our function takes the following inputs: risk aversion (i.e. maximum allowed MAD), a matrix with return data, and the maximum allowed allocation for any given asset. The function assumes that short-selling is not allowed.

```
# function to maximize return given upper bound on mean absolute deviation
opt_wt <- function(risk_aversion, return_matrix, max_wt) {


    num_pd <- nrow(return_matrix) # number of periods
    num_assets <- ncol(return_matrix) # number of assets under consideration

    # mean return vector for all aseets
    mean_vec <- matrix(apply(return_matrix,MARGIN=2,mean))

    # objective function:  maximuize return
    obj <- c(mean_vec, rep(0,num_pd))

    # initialize matrix, left hand side for constraints
    A <- matrix(,num_pd*2,num_pd + num_assets)

    #  left side of constraint: - y_t + sum [ x_j * (R_j(t) - mean(x_j))] <= 0; perform for all t
    for (i in 1:num_pd) {
        A[i,] <- c(return_matrix[i,] - t(matrix(mean_vec)), rep(0,num_pd))
        A[i, i + num_assets] <- -1
    }

    # left side of constraint: - y_t - sum [ x_j * (R_j(t) - mean(x_j))] <= 0; perform for all t
    for (i in (num_pd+1):(2*num_pd)) {
        A[i,] <- c(-(return_matrix[i-num_pd,] - t(matrix(mean_vec))), rep(0,num_pd))
```

```r
    A[i, i - num_pd + num_assets] <- -1
}

# left hand side constraint: all weights less than max_weight
for (i in 1:num_assets) {
    temp_row <- rep(0, num_assets + num_pd)
    temp_row[i] <- 1
    A <- rbind(A, temp_row)
}

# left side of constraint: avg y_t value less than risk aversion parameter
A <- rbind(A,c(rep(0,num_assets), rep(1, num_pd) / num_pd) )

# left side constraint: asset weights sum to 1
A <- rbind(A, c(rep(1,num_assets), rep(0, num_pd)))

# constraint, left hand side:  all weights greater than 0
for (i in 1:num_assets) {
    temp_row <- rep(0, num_assets + num_pd)
    temp_row[i] <- 1
    A <- rbind(A, temp_row)
}

# constraint, left hand side:  all y_t values are greater than 0
for (i in 1:num_pd) {
    temp_row <- rep(0, num_assets + num_pd)
    temp_row[i+num_assets] <- 1
    A <- rbind(A, temp_row)
}

# constraint operators
const_op <- c(rep("<=",num_pd*2 + num_assets + 1),"=",rep(">=",num_assets + num_pd))

# right side of constraint operator
b <- c(rep(0, num_pd*2), rep(max_wt, num_assets),risk_aversion, 1, rep(0, num_assets + num_pd))

# solve lp problem
mylp <- lp("max", obj, A, const_op, b)

# wt vector
wt <- mylp$solution[1:num_assets]
names(wt) <- symbols

# compute mean abs deviation, portfolio
p_mean <- t(matrix(wt)) %*% mean_vec

mad <- numeric()
for(i in 1: nrow(return_matrix)) {
  temp <- abs((t(matrix(return_matrix[i,])) %*% wt) - p_mean)
  mad <- c(mad, temp)
}
mad <- sum(mad) / length(mad)
```

```
    return(list(max_ret=mylp$objval,mad=mad, weights=round(wt,2)))

}
```

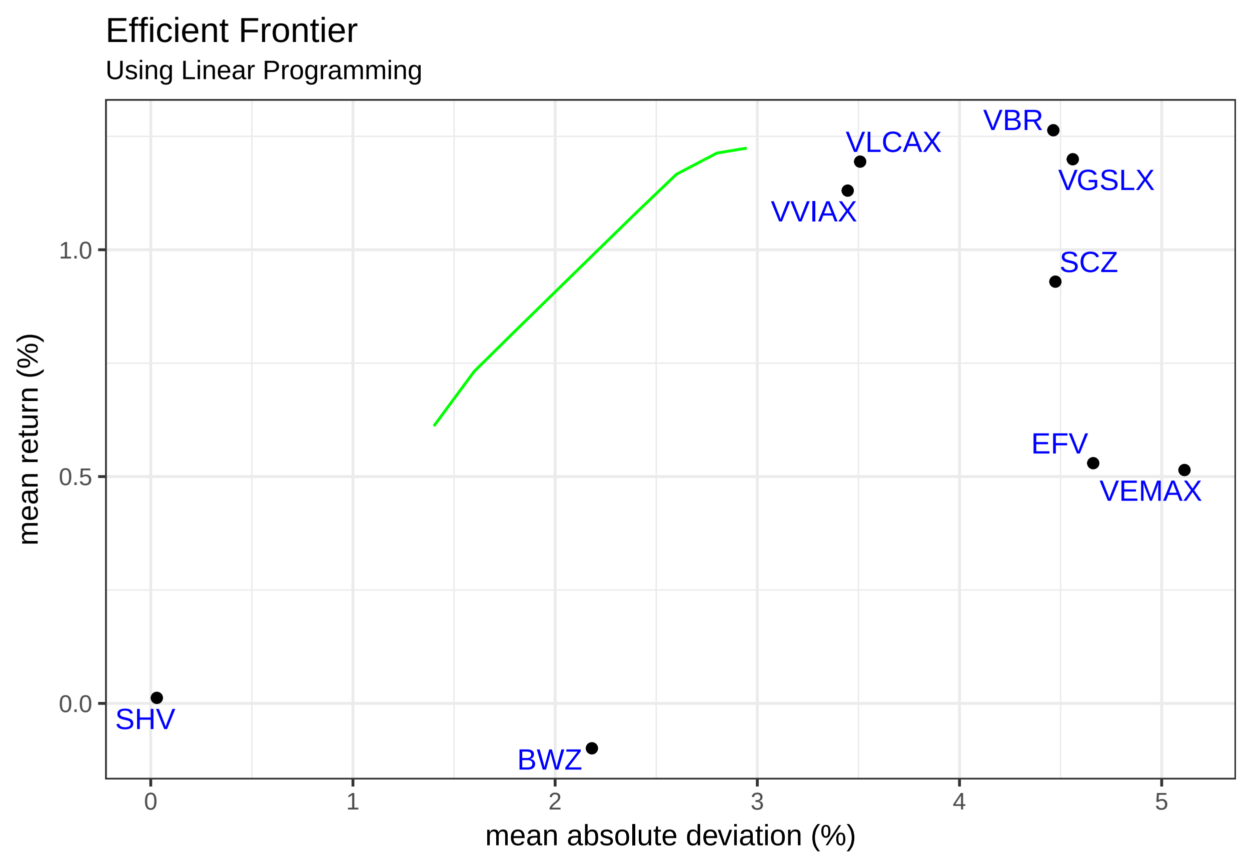
Here is example output from our function using a max MAD of 1.5%:

```
opt_wt(0.015, ret_matrix,0.4)
```

```
$max_ret
[1] 0.006768058

$mad
[1] 0.015

$weights
VLCAX VVIAX   VBR VGSLX   EFV   SCZ VEMAX   SHV   BWZ
 0.40  0.11  0.00  0.06  0.00  0.00  0.00  0.40  0.03
```

Now let's produce the efficient frontier, using MAD as our risk measure:

```
# lp efficient frontier helper function
eff_front_lp <- function(start, stop, by=0.01,ret = ret_matrix) {
    df <- data.frame(risk=numeric(),return=numeric())
    for (i in seq(start,stop, by)) {
        results <- opt_wt(i, ret, 0.4)
        df <- rbind(df, data.frame(risk=results$mad*100,return=results$max_ret*100))
    }
    df
}

# calculate return and mad for multiple scenarios
df <- eff_front_lp(0.014,0.06,0.002)

# plot
ggplot(df, aes(x=risk,y=return)) + geom_line(col='green') +
    geom_point(data = rr_df) +
    geom_text_repel(mapping = aes(risk, return), data = rr_df, label = rr_df$fund, col= "blue" ) +
    theme_bw() + labs(title = "Efficient Frontier", subtitle = "Using Linear Programming",
                      x="mean absolute deviation (%)", y = "mean return (%)")
```



Quadratic Programming

In our final optimization procedure, will will work with variance, a non-linear measure of dispersion.

Specifically, our objective will be to minimize an expression that considers both risk dispersion and portfolio return measures.

We will manually specify a constant parameter q that will be used to specify our level of risk aversion. If we set $q = 0$, then our goal will be to minimize variance. However, for large, positive values of q, our primary goal will be to maximize return.

Here is a formal statement of the optimization problem:

Define:
$q \geq 0$, constant

Objective:

$$\min\ \mathbf{x}^{\mathbf{T}} \Sigma \mathbf{x} - q \mu^{\mathbf{T}} \mathbf{x}$$

subject to

$$\sum_i x_i = 1$$

$$0 \leq x_i \leq 0.4, \text{ for all } i$$

Below we implement a function to calculate optimal portfolio weights given a risk parameter q, a return matrix, and a specified maximum allocation for any given asset.

Our function uses `solve.QP()` from the **quadprog** library.

```r
# function to optimize allocation using quadratic programming
qp_func <- function(q, return_matrix, max_wt) {
    Amat <- cbind(1, diag(ncol(return_matrix)), -diag(ncol(return_matrix)))
    bvec <- c(1, rep(0, ncol(return_matrix)),rep(-max_wt, ncol(return_matrix)))
    Dmat <-  2*cov(return_matrix)
    dvec <- matrix(apply(return_matrix,MARGIN=2,mean))

    qp_solve <- solve.QP(Dmat, dvec * q, Amat, bvec, meq=1, factorized=FALSE)

    wts <- qp_solve$solution
    names(wts) <- symbols
    p_returns <- c()
    for (i in 1:nrow(return_matrix)) {
        p_returns <- c(p_returns, t(matrix(wts)) %*% return_matrix[i,])
    }

    std_dev <- sd(p_returns)
    mean_ret <- mean(p_returns)

    list(max_ret=mean_ret, sd=std_dev, weights=round(wts,1))
}
```

Here is example output from our function, assuming $q = 0.5$

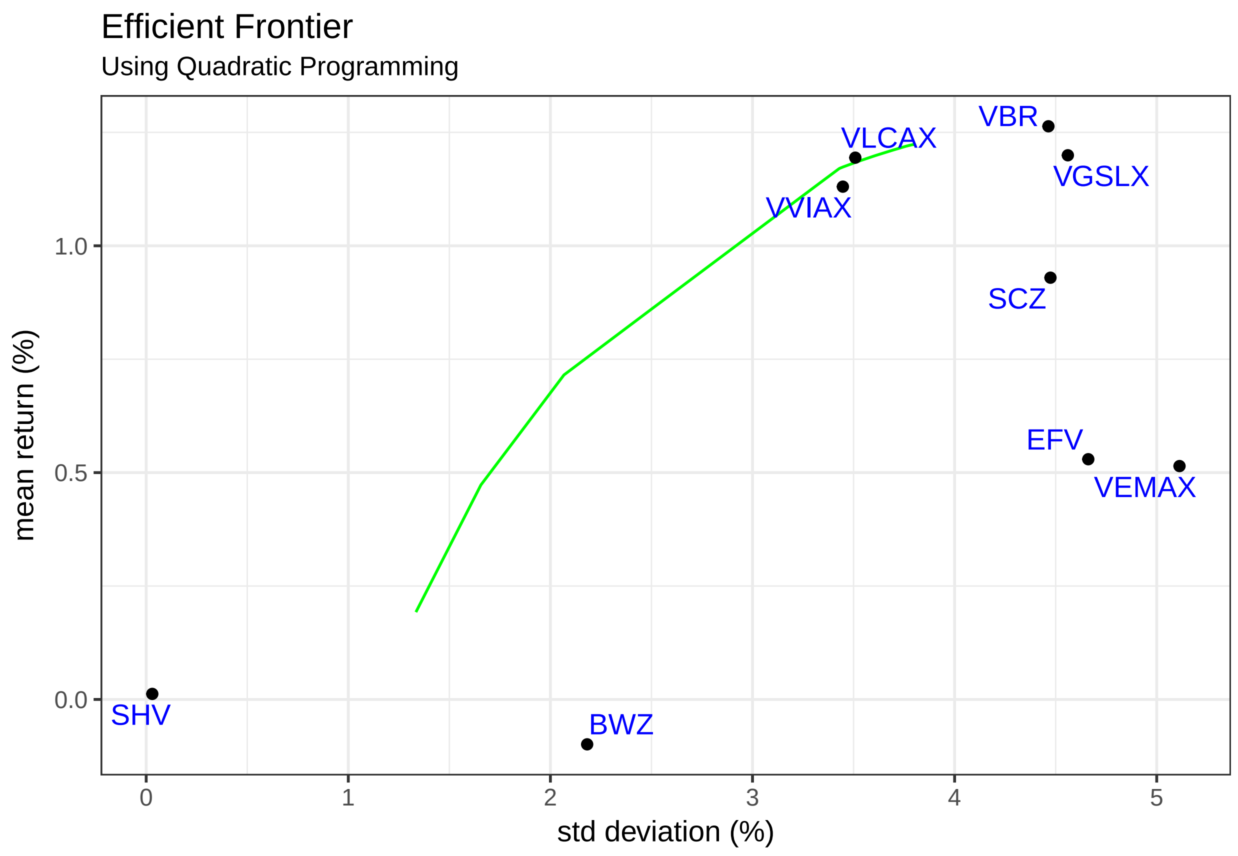
```r
qp_func(0.5, ret_matrix, 0.4)
```

```
$max_ret
[1] 0.01198807

$sd
[1] 0.03607641

$weights
VLCAX VVIAX   VBR VGSLX   EFV   SCZ VEMAX   SHV   BWZ 
  0.4   0.2   0.2   0.2   0.0   0.0   0.0   0.0   0.0 
```

Finally, here is the efficient frontier:

```r
# quadratic programm efficiient fronter helper function
eff_front_qp <- function(start, stop, by=0.01,ret = ret_matrix) {
    df <- data.frame(risk=numeric(),return=numeric())
    for (i in seq(start,stop, by)) {
        results <- qp_func(i, ret, 0.4)
        df <- rbind(df, data.frame(risk=results$sd*100,return=results$max_ret*100))
    }
    df
}

# calculate return and sd for multiple worst case sd scenarious
df <- eff_front_qp(0.00,5,0.05)

# plot
```

```
ggplot(df, aes(x=risk,y=return)) + geom_line(col='green') +
    geom_point(data = rr_df) +
    geom_text_repel(mapping = aes(risk, return), data = rr_df, label = rr_df$fund, col= "blue" ) + them
    labs(title = "Efficient Frontier", subtitle = "Using Quadratic Programming",
         x="std deviation (%)", y = "mean return (%)")
```



Closing

Investment science is a field that relies heavily on optimization procedures for making rational, portfolio allocation decisions. In this introductory analysis, we reviewed two basic portfolio optimization problems, and we tackled the latter problem using multiple approaches.

While our procedures were perhaps too simplistic for practical application, we have nevertheless provided a solid foundation for future study.

References

- Solin, Daniel R. "The SuperSmart Portfolio: Chapter and Verse." *The Smartest Portfolio You'll Ever Own: A Do-It-Yourself Breakthrough Strategy.* New York: Perigee, 2011. 79-83. Print.
  - Using author's suggested, diverse set of fund options comprising the "SuperSmart" portfolio
  - All options are low cost index funds or ETFs

- Customizing datatable appearance: https://rstudio.github.io/DT/

- Matrix Algebra and Portfolio Theory: https://faculty.washington.edu/ezivot/econ424/portfolioTheoryMatrix.pdf
  - Referenced for appropriate matrix and vector notation
  - Used specified procedures for finding global minimum variance portfolio
- Portfolio optimization using linear programming: http://orfe.princeton.edu/~rvdb/307/lectures/lec3_show.pdf
- Documentation on solve.qp(): https://www.rdocumentation.org/packages/quadprog/versions/1.5-5/topics/solve.QP
- Quadratic programming in R: http://www.wdiam.com/2012/06/10/mean-variance-portfolio-optimization-with-r-and-qua
- Overview of quadratic programming problem formulation: http://www.wdiam.com/b/wp-content/uploads/2012/06/quadratic_programming_markowitz_rvdb.pdf