I regularly use monte carlo engines to answer questions. First, they are really flexible in their ability to model non-normal distributions and assumptions. Second, you can incorporate any constraints you want which may be outside the scope of a non-linear optimization function.

At any rate, this is how to use R and a Monte Carlo technique to optimize an investment portfolio. I am here using the mean-variance method, though I don't use that regularly in practice.

Let's first set the expected returns, volatility, and correlations for each asset (I've labeled them A, B, and C).

```
## Expected Returns
A.r = 0.08
B.r = 0.03
C.r = 0.12

## Expected St Dev
A.s = 0.12
B.s = 0.03
C.s = 0.22

## Build Correlation vectors
A.col = c( 1.00, -0.05, 0.15 )  ## Pattern is correlation of A to A then A to B then A to C
B.col = c( -0.05, 1.00, -0.25 ) ## Same, except starting with B,A then B,B then B,C
C.col = c( 0.15, -0.25, 1.00 )  ## etc.

## Build Covariance Vectors
A.covcol = c( A.s*A.s, A.s*B.s, A.s*C.s ) ## This is covar without correlation
B.covcol = c( B.s*A.s, B.s*B.s, B.s*C.s ) ## LOL, I typed "B.s" a lot.
C.covcol = c( C.s*A.s, C.s*B.s, C.s*C.s )

A.cov = A.covcol*A.col
B.cov = B.covcol*B.col
C.cov = C.covcol*C.col
```

Obviously, the above is very simplistic. Normally, you'd have an entire model dedicated to your capital market expectations which would serve as inputs. But, alas, this is for instructional purposes only.…

Next, we need to build the relevant functions for our optimization. We care about portfolio return and portfolio standard deviation, so let's build those as functions of asset weights. Custom functions, for those newbies out there, are built using the **`function`**`( x1, x2, … ){ put your formula here }` structure.

```
## Build relevant functions
port.rf = function( wA, wB, wC ){ # Portfolio return function
        wA*A.r + wB*B.r + wC*C.r
}

port.sdf = function( wA, wB, wC ){ # Portfolio standard deviation function
        w.vec = c(wA, wB, wC)
        sqrt(
                wA * sum( w.vec * A.cov ) +
                wB * sum( w.vec * B.cov ) +
                wC * sum( w.vec * C.cov )
        )
}
```

We now have the two functions to generate portfolio risk and return given the weights. The weights, of course, will be generated from random distributions. In essence, the approach is to throw a ton of possible weights at the functions, then find which ones deliver the best risk/return profile.

Here is the challenge, you cannot simply generate random numbers between 0 and 1 because they will not sum to 1 every time. Given three assets, they could sum to 3 or almost 0. So, we need to build a "normalizer" to make the actual weights sum to 1 (or, if you want to include leverage, >1).

```
## Build MC structure
trials = 100000           # How many portfolios to generate?
a = runif( trials, 0, 1 ) # Generate random numbers between 0 and 1.
```

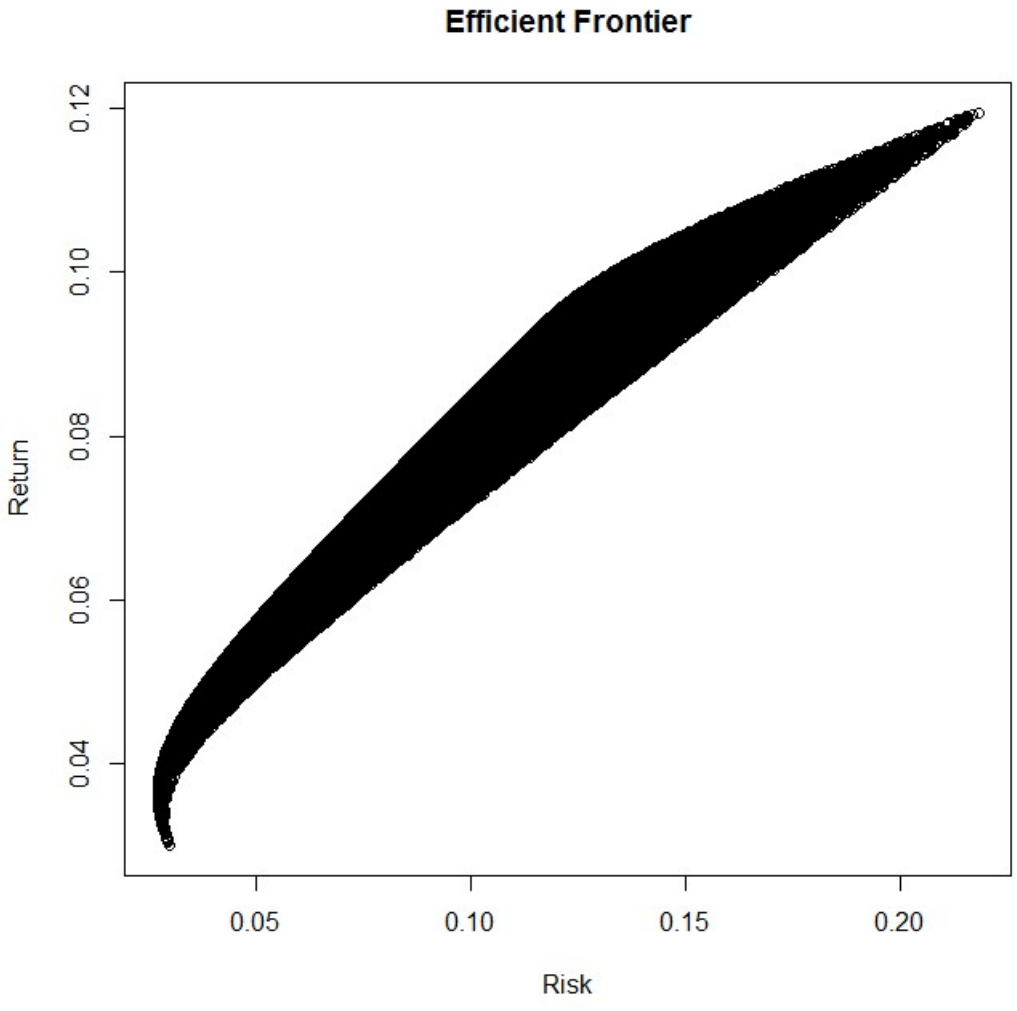
```
b = runif( trials, 0, 1 )
c = runif( trials, 0, 1 )

nmlzer = a + b + c

wA = a/nmlzer   # This forces weights to sum to 1.
wB = b/nmlzer
wC = c/nmlzer
```

We now have 100,000 possible portfolios to consider! Next we need to iterate through each possible portfolio to generate it's risk return characteristics.

```
## Return each portfolio's risk, return
sd = 1                  # Prime the variable
for( i in 1:trials ){   # Iterate through each trial to return sd
        sd[i] = port.sdf( wA[i], wB[i], wC[i] )
}

r = 1
for( i in 1:trials ){   # Iterate through each trial to return portfolio return
        r[i] = port.rf( wA[i], wB[i], wC[i] )
}
```

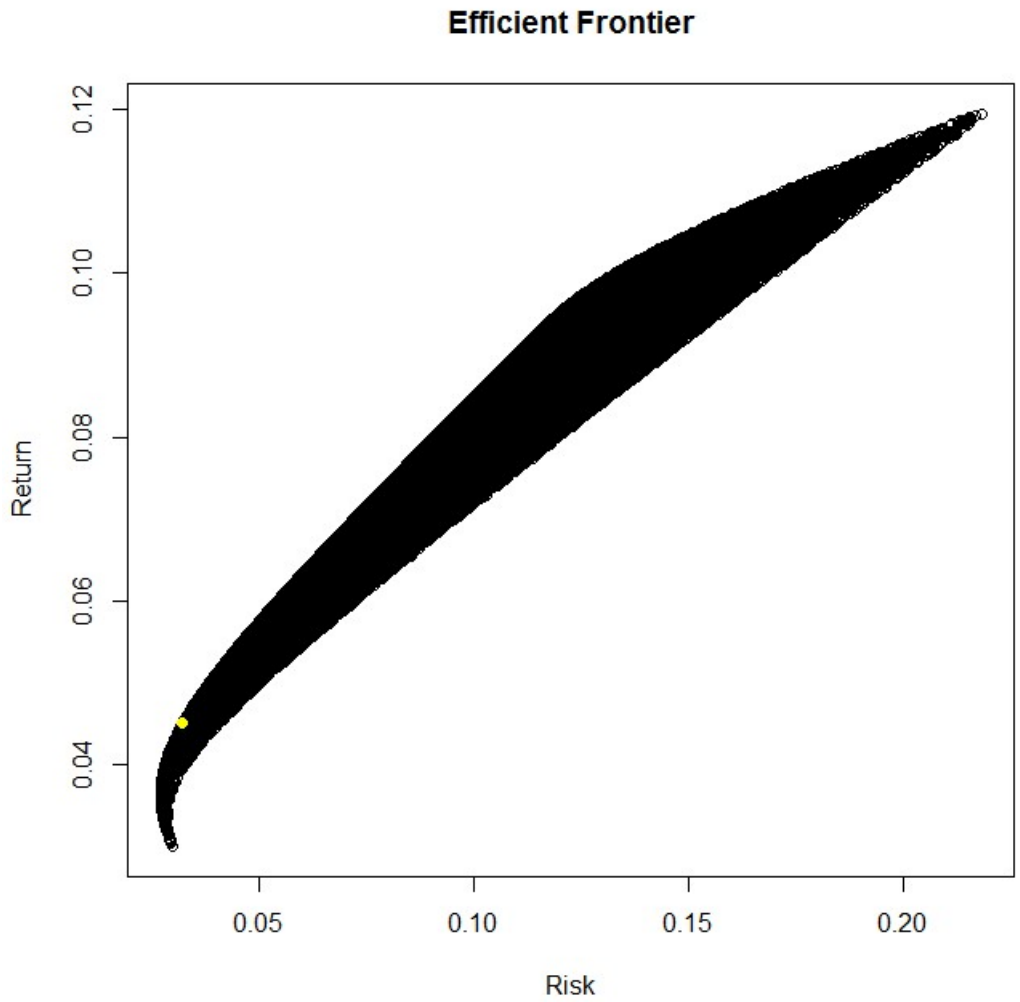
Great! Now we have the data we need. Let's take a look at our result.

```
## Visualize Efficient Frontier
plot( sd, r,
        main="Efficient Frontier",
        xlab="Risk",
        ylab="Return" )
```



That looks promising! Given the range of portfolios, we can now ask our optimizer questions. For example, which portfolio has the highest Sharpe ratio?

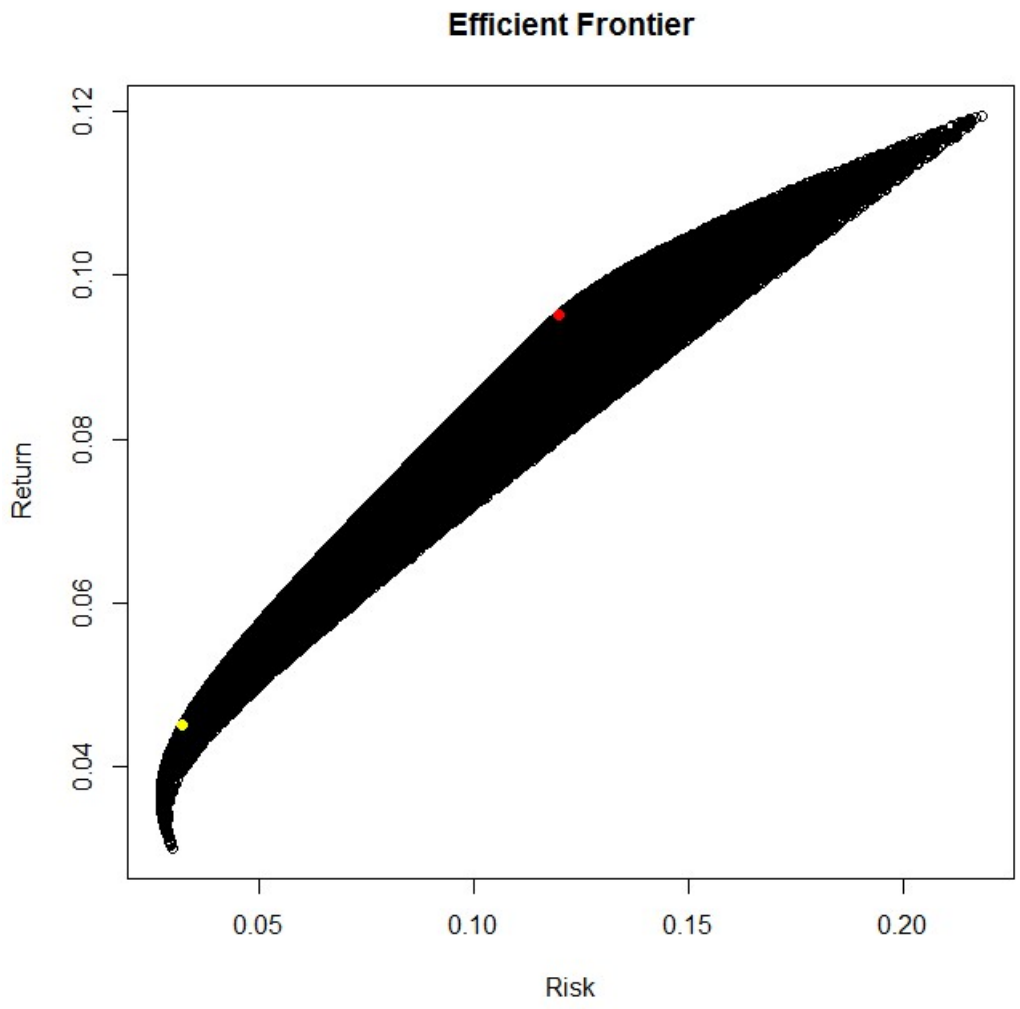
```
## Highest Sharpe ratio
risk.free = 0.015
sharpe.f = function( sd, r ){ (r - risk.free)/sd }# Function to generate Sharpe ratio

sharpe = 1
for( i in 1:trials ){                                                    # Iterate
through each portfolio
        sharpe[i] = sharpe.f( sd[i], r[i] )
}

index = which( sharpe == max(sharpe) ) # Return portfolio number with highest sharpe
sd.maxSR = sd[index]                                                     # Max
Sharpe ratio portfolio, risk
r.maxSR = r[index]                                                                #
Max Sharpe ratio portfolio, return

## Plot Max SR portfolio on Efficient Frontier
points( sd.maxSR, r.maxSR, col="yellow", pch=19 ) # Make this portfolio a yellow dot on our
EF
```



Or, perhaps you have a risk budget of 12% standard deviation. Which portfolio gives the highest return for that level of risk?

```
## Max return for 12% risk
index2 = which( r == max( r[ which( sd <= 0.12 ) ] )
)                                                  # Selects the maximum r from only
the                                                       # rs that are available with a sd
less than                                                                 # or equal
```

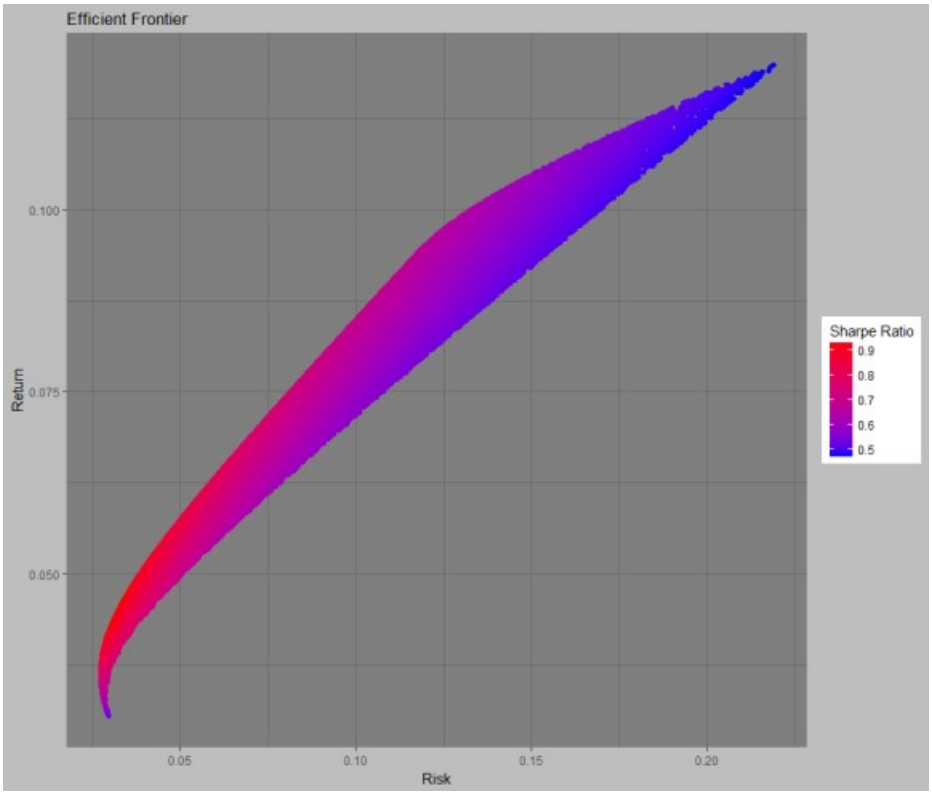
```
to 0.12. Then returns the portfolio number
sd.max12 = sd[ index2 ]  # SD for that portfolio number
r.max12 = r[ index2 ]    # R for that portfolio number
points( sd.max12, r.max12, col="red", pch=19 )  # Visualize on frontier.
```



Of course, to find the weights of that portfolio, you simply call the weight number using the index variable we just created. You can even create a pie chart with that information.

```
(my.portfolio.weights = c( wA[ index2 ], wB[ index2 ], wC[ index2 ] ) )
labels = c( "A", "B", "C" )
dev.new()
pie( my.portfolio.weights,
       labels = labels,
       main = "Optimal Portfolio for 12% Risk" )
```

So there you go! That is a simple way to optimize a portfolio in R. Of course, you can make a really cool efficient frontier in R, too. This one is the same data, but the color of the dots changes with respect to the Sharpe Ratio. But, that is a visualization lesson for a different day.


Efficient Frontier
Return
0.100
0.075
0.050
0.05
0.10
0.15
0.20
Risk
Sharpe Ratio
0.9
0.8
0.7
0.6
0.5