Coding in R: Generate Half Gaussian Table, Optimal Markowitz portfolio

Salpawuni, A.S

14.06.2021

Problem 2

```
# 1. phi function ----------------------------------------------------------
phi <- function(x = NULL){
  if(is.null(x)) stop("Stop! Null vector not allowed!") # test for non-null vector

  x = scale(x) # obtain standardization of vector x
  out <- 1/sqrt(2 * pi) * exp(-x^2/2) # probability density of vector x
  return(as.vector(out))
}
# output vector of densities
# e.g. phi(x = seq(100)) outputs 100 densities

# 2. quantile vector, integrate to obtain phi vector values ---------------

quant = seq(0, 5.5, by = 1/100) # for half-table values
f = function(x) 1/sqrt(2 * pi) * exp(-x^2/2) # integrand for Phi(x) value
probs <- sapply(quant,
                FUN = function(x) integrate(f, lower = 0, upper = x)$value)

# check if probs and pnorm() agree to 4 decimal places
identical(round(probs, 4), round(pnorm(q = quant)-0.5, 4))
```

```
## [1] TRUE
```

```
# matrix form (subset first 400 to draw half-table)
half_table <- as.data.frame(matrix(probs[1:400], ncol = 10, byrow = TRUE))
rownames(half_table) = sprintf('%0.1f', seq(0, 3.9, by = 1/10))
colnames(half_table) = sprintf('%0.2f', seq(0, 0.09, by = 1/100))
# output Gaussian table (half-table)
knitr::kable(half_table,
             digits = 4,
             caption = "Standard normal table: $P(0 < Z < z )$")
```

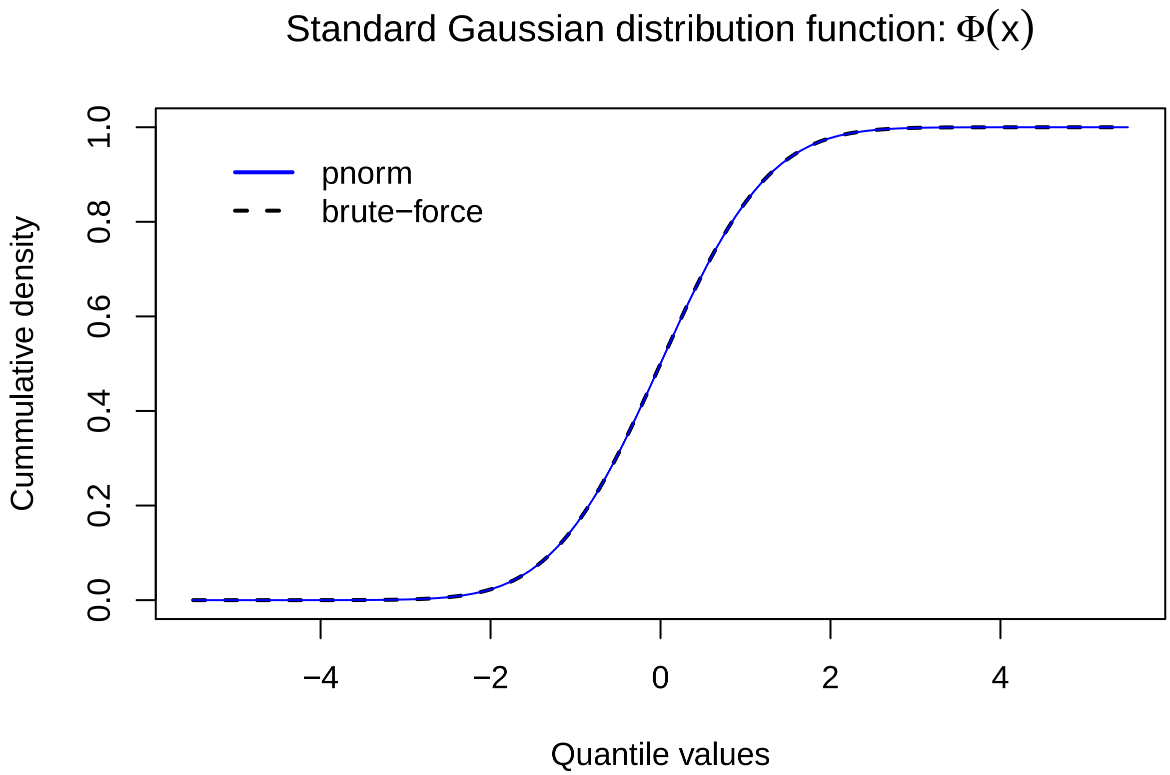
Table 1: Standard normal table: $P(0 < Z < z)$

	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.0000	0.0040	0.0080	0.0120	0.0160	0.0199	0.0239	0.0279	0.0319	0.0359
0.1	0.0398	0.0438	0.0478	0.0517	0.0557	0.0596	0.0636	0.0675	0.0714	0.0753
0.2	0.0793	0.0832	0.0871	0.0910	0.0948	0.0987	0.1026	0.1064	0.1103	0.1141
0.3	0.1179	0.1217	0.1255	0.1293	0.1331	0.1368	0.1406	0.1443	0.1480	0.1517
0.4	0.1554	0.1591	0.1628	0.1664	0.1700	0.1736	0.1772	0.1808	0.1844	0.1879
0.5	0.1915	0.1950	0.1985	0.2019	0.2054	0.2088	0.2123	0.2157	0.2190	0.2224
0.6	0.2257	0.2291	0.2324	0.2357	0.2389	0.2422	0.2454	0.2486	0.2517	0.2549
0.7	0.2580	0.2611	0.2642	0.2673	0.2704	0.2734	0.2764	0.2794	0.2823	0.2852
0.8	0.2881	0.2910	0.2939	0.2967	0.2995	0.3023	0.3051	0.3078	0.3106	0.3133
0.9	0.3159	0.3186	0.3212	0.3238	0.3264	0.3289	0.3315	0.3340	0.3365	0.3389
1.0	0.3413	0.3438	0.3461	0.3485	0.3508	0.3531	0.3554	0.3577	0.3599	0.3621

	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
1.1	0.3643	0.3665	0.3686	0.3708	0.3729	0.3749	0.3770	0.3790	0.3810	0.3830
1.2	0.3849	0.3869	0.3888	0.3907	0.3925	0.3944	0.3962	0.3980	0.3997	0.4015
1.3	0.4032	0.4049	0.4066	0.4082	0.4099	0.4115	0.4131	0.4147	0.4162	0.4177
1.4	0.4192	0.4207	0.4222	0.4236	0.4251	0.4265	0.4279	0.4292	0.4306	0.4319
1.5	0.4332	0.4345	0.4357	0.4370	0.4382	0.4394	0.4406	0.4418	0.4429	0.4441
1.6	0.4452	0.4463	0.4474	0.4484	0.4495	0.4505	0.4515	0.4525	0.4535	0.4545
1.7	0.4554	0.4564	0.4573	0.4582	0.4591	0.4599	0.4608	0.4616	0.4625	0.4633
1.8	0.4641	0.4649	0.4656	0.4664	0.4671	0.4678	0.4686	0.4693	0.4699	0.4706
1.9	0.4713	0.4719	0.4726	0.4732	0.4738	0.4744	0.4750	0.4756	0.4761	0.4767
2.0	0.4772	0.4778	0.4783	0.4788	0.4793	0.4798	0.4803	0.4808	0.4812	0.4817
2.1	0.4821	0.4826	0.4830	0.4834	0.4838	0.4842	0.4846	0.4850	0.4854	0.4857
2.2	0.4861	0.4864	0.4868	0.4871	0.4875	0.4878	0.4881	0.4884	0.4887	0.4890
2.3	0.4893	0.4896	0.4898	0.4901	0.4904	0.4906	0.4909	0.4911	0.4913	0.4916
2.4	0.4918	0.4920	0.4922	0.4925	0.4927	0.4929	0.4931	0.4932	0.4934	0.4936
2.5	0.4938	0.4940	0.4941	0.4943	0.4945	0.4946	0.4948	0.4949	0.4951	0.4952
2.6	0.4953	0.4955	0.4956	0.4957	0.4959	0.4960	0.4961	0.4962	0.4963	0.4964
2.7	0.4965	0.4966	0.4967	0.4968	0.4969	0.4970	0.4971	0.4972	0.4973	0.4974
2.8	0.4974	0.4975	0.4976	0.4977	0.4977	0.4978	0.4979	0.4979	0.4980	0.4981
2.9	0.4981	0.4982	0.4982	0.4983	0.4984	0.4984	0.4985	0.4985	0.4986	0.4986
3.0	0.4987	0.4987	0.4987	0.4988	0.4988	0.4989	0.4989	0.4989	0.4990	0.4990
3.1	0.4990	0.4991	0.4991	0.4991	0.4992	0.4992	0.4992	0.4992	0.4993	0.4993
3.2	0.4993	0.4993	0.4994	0.4994	0.4994	0.4994	0.4994	0.4995	0.4995	0.4995
3.3	0.4995	0.4995	0.4995	0.4996	0.4996	0.4996	0.4996	0.4996	0.4996	0.4997
3.4	0.4997	0.4997	0.4997	0.4997	0.4997	0.4997	0.4997	0.4997	0.4997	0.4998
3.5	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998	0.4998
3.6	0.4998	0.4998	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999
3.7	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999
3.8	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999	0.4999
3.9	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000

```r
# 3. Distribution curve (full table plot) ----------------------------------
x = c(-rev(quant), quant)
y = sapply(x,
           FUN = function(x) integrate(f, lower = -5.5, upper = x)$value)

plot(x, y, type = 'l',
     xlab = "Quantile values",
     ylab = "Cummulative density",
     lwd = 2, lty = 2,
     main = expression(
       paste("Standard Gaussian distribution function:")~Phi(x))
     # xlim = c(-5, 5), ylim = c(0, 1), axes = TRUE,
     )
curve(expr = pnorm(x), add = TRUE, col = 'blue', lty = 1.5)
legend('topleft',
       inset = 0.05,
       legend = c("pnorm", "brute-force"),
       bty = "n", # Removes the legend box
       lty = c(1, 2),
       col = c('blue', 'black'),
       lwd = 2)
```



Problem 3

```
options (digits=4, width=70)

library("PerformanceAnalytics")
library("tseries")
library("zoo")

# DATA -------------------------------------------------------------------
##########################################################
### TASK 1: Retrieving 10 stock prices (Yahoo S&P500)
##########################################################
tickers <- c("AAPL", "MSFT", "AMZN", "GOOGL", "BRK-B",
             "JPM", "JNJ", "NVDA", "BAC", "PFE")
lst <- list() # To hold adjusted monthly returns

retrieve_stock <- function(){
    for(i in tickers){
      lst[[i]] <- get.hist.quote(instrument = i,
                                          start = "2010-01-01",
                                          end = "2020-01-01",
                                          quote = "AdjClose",
                                          provider = "yahoo",
                                          origin = "2000-09-01",
                                          compression = "m",
                                          retclass = "zoo")
      }
    return(lst)
  }
stock_prices = retrieve_stock() # Stock prices from S&P500 (Yahoo)
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
## time series ends   2019-12-01
## time series ends   2019-12-01
## time series ends   2019-12-01
## time series ends   2019-12-01
## time series ends   2019-12-01
## time series ends   2019-12-01
## time series ends   2019-12-01
## time series ends   2019-12-01
## time series ends   2019-12-01
## time series ends   2019-12-01
```

```r
sapply(stock_prices, length) # Check lengths of downloaded RETURNs (i.e 120)
```

```
##  AAPL  MSFT  AMZN GOOGL BRK-B   JPM   JNJ  NVDA   BAC   PFE
##   120   120   120   120   120   120   120   120   120   120
```

```r
# Transformation and Summary Statistics -----------------------------------
########################################################
### TASK 2: mean, variance-covariance and weights
########################################################
N = length(tickers) # number of securities (10)
ones <- matrix(data = 1, nrow = N) # matrix of ones

descriptive_stat <- function(){
  lr <- lapply(X = stock_prices, FUN = function(x) diff(log(x)))
  mat <- sapply(lr, unclass) # obtain matrix of 10 securities

  ln = nrow(mat) # off by one after differencing

  # random weights
  wts <- runif(n = N) # random weights (not scaled)
  wts <- wts/sum(wts) # scaled weights (NB: this sums to One!)
  names(wts) <- tickers

  mu <- 1/ln * t(mat) %*% matrix(1, nrow = ln) # mean vector
  mu <- as.vector(mu)
  names(mu) <- tickers

  mu_mat <- matrix(data = mu, ncol = N, nrow = ln, byrow = TRUE) # mean matrix

  vcv <- 1/(ln - 1) * t((mat - mu_mat)) %*% (mat - mu_mat) # var-cov
  vcv <- as.matrix(vcv)
  dimnames(vcv) <- list(tickers, tickers)
  class(vcv) <- "matrix"

  list(mu_cap = mu, cov_mat = vcv, weights = wts) # mean, variance, weights
}
descript <- descriptive_stat() # Get and print values from descriptive_stat()
descript$mu_cap
```

```
##     AAPL     MSFT     AMZN    GOOGL    BRK-B      JPM      JNJ
## 0.021089 0.016483 0.022607 0.013608 0.009129 0.012748 0.009606
```

```
##     NVDA      BAC      PFE
## 0.023609 0.007925 0.009329
```

```r
descript$cov_mat
```

```
##            AAPL      MSFT      AMZN     GOOGL     BRK-B       JPM
## AAPL  5.332e-03 0.0017387 0.0019689 0.0016484 0.0005565 0.0013656
## MSFT  1.739e-03 0.0036007 0.0019377 0.0018938 0.0006564 0.0021819
## AMZN  1.969e-03 0.0019377 0.0062797 0.0024272 0.0007627 0.0016718
## GOOGL 1.648e-03 0.0018938 0.0024272 0.0041605 0.0004818 0.0016069
## BRK-B 5.565e-04 0.0006564 0.0007627 0.0004818 0.0016122 0.0014506
## JPM   1.366e-03 0.0021819 0.0016718 0.0016069 0.0014506 0.0048291
## JNJ   5.647e-04 0.0008304 0.0010026 0.0005800 0.0008434 0.0009828
## NVDA  3.671e-03 0.0028929 0.0032514 0.0024704 0.0016160 0.0029807
## BAC   1.849e-03 0.0026145 0.0020101 0.0016732 0.0016923 0.0053808
## PFE   8.117e-05 0.0005848 0.0012071 0.0007677 0.0006772 0.0012951
##             JNJ      NVDA      BAC       PFE
## AAPL  0.0005647 0.0036713 0.001849 8.117e-05
## MSFT  0.0008304 0.0028929 0.002615 5.848e-04
## AMZN  0.0010026 0.0032514 0.002010 1.207e-03
## GOOGL 0.0005800 0.0024704 0.001673 7.677e-04
## BRK-B 0.0008434 0.0016160 0.001692 6.772e-04
## JPM   0.0009828 0.0029807 0.005381 1.295e-03
## JNJ   0.0016453 0.0010763 0.001002 1.049e-03
## NVDA  0.0010763 0.0138160 0.004080 8.588e-04
## BAC   0.0010024 0.0040798 0.008562 1.284e-03
## PFE   0.0010487 0.0008588 0.001284 2.186e-03
```

```r
descript$weights
```

```
##     AAPL     MSFT     AMZN    GOOGL    BRK-B      JPM      JNJ
## 0.202692 0.067247 0.181160 0.100225 0.190775 0.049756 0.076445
##     NVDA      BAC      PFE
## 0.004249 0.004731 0.122720
```

```r
# Get Portfolio -----------------------------------------------------------
#########################################################
### Task 3: Create a portfolio object
#########################################################
get_portfolio <-function(er, cov_mat, weights){

  ### Note the following
  # er: expected returns vector (N x 1)
  # cov_mat: variance-covariance matrix of returns (N x N)
  # weights: random weights vector, summing to 1 (N x 1)

  # Assess validity of inputs
  weights <- as.vector(weights); names(weights) <- tickers
  er <- as.vector(er)                    # assign names if none exist
  if(length(er) != length(weights))
    stop("Stop! Wrong dimensions: er and weights do not match")
  cov_mat <- as.matrix(cov_mat)
  if(length(er) != nrow(cov_mat))
    stop("Stop! Wrong dimensions: er and cov_mat do not match")
  if(any(diag(chol(cov_mat)) <= 0)) # check for positive definiteness
    stop("Covariance matrix not positive definite")

  # create portfolio
  er.port <- crossprod(er, weights)
  sd.port <- sqrt(weights %*% cov_mat %*% weights)
  # results
```

```r
  list(g.er = as.vector(er.port),
       g.sd = as.vector(sd.port),
       weights = weights
       )
}
# Get values from get_portfolio()
get_port <- get_portfolio(er = descript$mu,
                          cov_mat = descript$cov_mat,
                          weights = descript$weights)
# portfolio with random weights
cat("Portfolio expected return: ", get_port$g.er)
```

```
## Portfolio expected return:  0.01524
```

```r
cat("\nPortfolio standard deviation: ", get_port$g.sd)
```

```
##
## Portfolio standard deviation:  0.03938
```

```r
get_port$weights
```

```
##     AAPL     MSFT     AMZN    GOOGL    BRK-B      JPM      JNJ
## 0.202692 0.067247 0.181160 0.100225 0.190775 0.049756 0.076445
##     NVDA      BAC      PFE
## 0.004249 0.004731 0.122720
```

```r
# Efficient portfolio (Resolution 1) --------------------------------------
#########################################################
### Task 4: Compute minimum variance portfolio
#########################################################

# NB: in this section, target return, c* = mu_cap for the ith return

efficient_portfolio <- function(er, cov_mat, target_return){
  ### Note the following
  # er: expected returns vector (N x 1)
  # cov_mat: variance-covariance matrix of returns (N x N)
  # weights: random weights vector, summing to 1 (N x 1)
  # target.return: targeted return (scalar)

  #
  # output is portfolio object with the following elements
  # call                    original function call
  # er                      portfolio expected return
  # sd                      portfolio standard deviation
  # weights             N x 1 vector of portfolio weights

  ## inputs
  er <- as.vector(er)                   # assign names if none exist
  cov_mat <- as.matrix(cov_mat)
  if(length(er) != nrow(cov_mat))
    stop("invalid inputs")
  if(any(diag(chol(cov_mat)) <= 0))
    stop("Covariance matrix not positive definite")


  # compute efficient portfolio
  # forming system of equation

  ones <- rep(1, length(er))
```

```r
  top <- cbind(2*cov_mat, er, ones)
  bot <- cbind(rbind(t(er), t(ones)), matrix(0,2,2)) # edited by me (no error)
  A <- rbind(top, bot)
  b.target <- as.matrix(c(rep(0, length(er)), target_return, 1))
  x <- solve(A, b.target)
  w <- x[1:length(er)]
  names(w) <- tickers

  #
  # compute portfolio expected returns and variance
  #
  er.port <- crossprod(er,w)
  sd.port <- sqrt(w %*% cov_mat %*% w)

  # output
  list(exp_ret_port = as.vector(er.port),
       sd_port = as.vector(sd.port),
       weights_ef = w)
}
efficient_portfolio(er = descript$mu_cap,
                    cov_mat = descript$cov_mat,
                    target_return = 0.0211) ### mu_cap for AAPL is 0.0211
```

```
## $exp_ret_port
## [1] 0.0211
##
## $sd_port
## [1] 0.05075
##
## $weights_ef
##     AAPL     MSFT     AMZN    GOOGL    BRK-B      JPM      JNJ
##  0.33510  0.28451  0.28213 -0.11812  0.17346  0.37663 -0.07312
##     NVDA      BAC      PFE
##  0.04852 -0.44763  0.13852
```

```r
# To compute for efficient portfolios for all tickers, run the code below
# sapply(1:N, FUN = function(x){
#  efficient_portfolio(er = descrip$mu_cap,
#                      cov_mat = desc_vals$cov_mat,
#                      target_return = descript$mu_cap[x])})


# Lagrangian method ---------------------------------------------------------
#########################################################
### Task 5: Lagrangian method (lambda1 and lambda2)
#########################################################
markowitz = function(mu,cov_mat,er) {
  A = t(ones) %*% solve(cov_mat) %*% mu
  B = t(mu) %*% solve(cov_mat) %*% mu
  C = t(ones) %*% solve(cov_mat) %*% ones
  D = B*C - A^2
  lam1 = (C*er-A)/D
  lam2 = (B-A*er)/D
  wts = lam1[1]*(solve(cov_mat) %*% mu) + lam2[1]*(solve(cov_mat) %*% ones)
  g = (B[1]*(solve(cov_mat) %*% ones) - A[1]*(solve(cov_mat) %*% mu))/D[1]
  h = (C[1]*(solve(cov_mat) %*% mu) - A[1]*(solve(cov_mat) %*% ones))/D[1]
  wts = g + h*er
}
# output
```

```r
# Optimal portfolio weights: Assuming expected returns = c*, eg. AAPL
(markowitz(mu = descript$mu_cap, cov_mat = descript$cov_mat, er = 0.0211))
```

```
##           [,1]
## AAPL   0.33510
## MSFT   0.28451
## AMZN   0.28213
## GOOGL -0.11812
## BRK-B  0.17346
## JPM    0.37663
## JNJ   -0.07312
## NVDA   0.04852
## BAC   -0.44763
## PFE    0.13852
```

```r
# Global Minimum variance (Resolution 2) ----------------------------------
########################################################
### Task 6: Global minimum variance portfolio
########################################################
global_min_portfolio <- function(er, cov_mat){
  #
  cov_mat_inv <- solve(cov_mat)
  one_vec <- rep(1,length(er))
  w_gmin <- rowSums(cov_mat_inv) / sum(cov_mat_inv)
  w_gmin <- as.vector(w_gmin)
  names(w_gmin) <- tickers
  er_gmin <- crossprod(w_gmin, er)
  sd_gmin <- sqrt(t(w_gmin) %*% cov_mat %*% w_gmin)

  # output
  list(er_gmin = as.vector(er_gmin),
       sd_gmin = as.vector(sd_gmin),
       w_gmin = w_gmin)
}
glmin <- global_min_portfolio(er = descript$mu_cap, cov_mat = descript$cov_mat)

# Global minimum variance
cat("Portfolio expected return (global): ", glmin$er_gmin)
```

```
## Portfolio expected return (global):  0.01059
```

```r
cat("Portfolio standard deviation (global): ", glmin$sd_gmin)
```

```
## Portfolio standard deviation (global):  0.03062
```

```r
glmin$w_gmin
```

```
##     AAPL     MSFT     AMZN    GOOGL    BRK-B      JPM      JNJ
##  0.11713  0.12599 -0.03203  0.09569  0.44496 -0.07798  0.17196
##     NVDA      BAC      PFE
## -0.05228 -0.03597  0.24254
```

```r
# Compute efficient frontier ----------------------------------------------
########################################################
### Task 7: Compute Markowitz bullet
########################################################
efficient_frontier <- function(er, cov_mat,
                               nport = N, alpha_min = -0.5,
                               alpha_max = 1.5){

  # create portfolio names
```

```r
  port_names <- rep("port", nport)
  ns <- seq(1, nport)
  port_names <- paste(port_names, ns)

  # compute global minimum variance portfolio
  #
  cov_mat_inv <- solve(cov_mat)
  one_vec <- rep(1, length(er))
  port_gmin <- global_min_portfolio(er, cov_mat)
  w_gmin <- port_gmin$w_gmin


  # compute efficient frontier as convex combinations of two efficient ports
  # 1st efficient port: global min var portfolio
  # 2nd efficient port: min var port with ER = max of ER for all assets
  er.max <- max(er)
  port.max <- efficient_portfolio(er, cov_mat, er.max)
  w.max <- port.max$weights_ef
  a <- seq(from = alpha_min, to = alpha_max,length = nport)# convex combinations
  we.mat <- a %o% w_gmin + (1-a) %o% w.max  # rows are efficient portfolios
  er.e <- we.mat %*% er                       # expected returns of efficient portfolios
  er.e <- as.vector(er.e)
  names(er.e) <- port_names
  cov.e <- we.mat %*% cov_mat %*% t(we.mat) # cov mat of efficient portfolios
  sd.e <- sqrt(diag(cov.e))                 # std devs of efficient portfolios
  sd.e <- as.vector(sd.e)
  names(sd.e) <- port_names
  dimnames(we.mat) <- list(port_names, tickers)

  #
  # summarize results
  list(markowitz_er = er.e,
       markowitz_sd = sd.e,
       markowitz_weights = we.mat)
}
efficient_frontier(er = descript$mu_cap,
                   cov_mat = descript$cov_mat,
                   nport = N,
                   alpha_min = -0.5,
                   alpha_max = 1.5)
```

```
## $markowitz_er
##   port 1   port 2   port 3   port 4   port 5   port 6   port 7
## 0.030120 0.027226 0.024333 0.021439 0.018545 0.015652 0.012758
##   port 8   port 9  port 10
## 0.009864 0.006971 0.004077
##
## $markowitz_sd
##  port 1  port 2  port 3  port 4  port 5  port 6  port 7  port 8
## 0.08118 0.07099 0.06113 0.05179 0.04331 0.03630 0.03174 0.03075
##  port 9 port 10
## 0.03364 0.03957
##
## $markowitz_weights
##            AAPL    MSFT    AMZN     GOOGL    BRK-B     JPM
## port 1  0.52213 0.42054  0.55170 -0.301584 -0.05950  0.76671
## port 2  0.46213 0.37691  0.46522 -0.242729  0.01524  0.64157
## port 3  0.40213 0.33327  0.37875 -0.183874  0.08997  0.51643
## port 4  0.34213 0.28963  0.29227 -0.125019  0.16471  0.39129
```

```
## port 5   0.28213 0.24599  0.20579 -0.066164  0.23944  0.26615
## port 6   0.22213 0.20235  0.11931 -0.007308  0.31417  0.14101
## port 7   0.16213 0.15871  0.03283  0.051547  0.38891  0.01587
## port 8   0.10213 0.11508 -0.05365  0.110402  0.46364 -0.10927
## port 9   0.04213 0.07144 -0.14013  0.169257  0.53838 -0.23441
## port 10 -0.01787 0.02780 -0.22661  0.228112  0.61311 -0.35955
##              JNJ      NVDA       BAC     PFE
## port 1  -0.28341  0.135002 -0.800866 0.04926
## port 2  -0.21595  0.107257 -0.687548 0.07789
## port 3  -0.14848  0.079513 -0.574230 0.10653
## port 4  -0.08102  0.051768 -0.460912 0.13516
## port 5  -0.01356  0.024023 -0.347594 0.16379
## port 6   0.05390 -0.003722 -0.234275 0.19243
## port 7   0.12136 -0.031467 -0.120957 0.22106
## port 8   0.18882 -0.059212 -0.007639 0.24970
## port 9   0.25628 -0.086957  0.105679 0.27833
## port 10  0.32375 -0.114702  0.218997 0.30696
```

```
#######################  THE END  #############################
```