```
# portfolio.r
#
# Functions for portfolio analysis
# to be used in Introduction to Computational Finance & Financial Econometrics
# last updated: November 7, 2000 by Eric Zivot
#               Oct 15, 2003 by Tim Hesterberg
#               November 18, 2003 by Eric Zivot
#               November 9, 2004 by Eric Zivot
#                           November 9, 2008 by Eric Zivot
#               August 11, 2011 by Eric Zivot
#
# Functions:
#       1. efficient.portfolio                  compute minimum variance portfolio
#                                                                       subject to target
return
#       2. globalMin.portfolio                  compute global minimum variance portfolio
#       3. tangency.portfolio                     compute tangency portfolio
#       4. efficient.frontier                     compute Markowitz bullet
#       5. getPortfolio                                     create portfolio object
#

getPortfolio <-
function(er, cov.mat, weights)
{
        # contruct portfolio object
        #
        # inputs:
        # er                              N x 1 vector of expected returns
        # cov.mat                 N x N covariance matrix of returns
        # weights                       N x 1 vector of portfolio weights
        #
        # 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
        #
        call <- match.call()

        #
        # check for valid inputs
        #
        asset.names <- names(er)
        weights <- as.vector(weights)
        names(weights) = names(er)
        er <- as.vector(er)                                  # assign names if none exist
        if(length(er) != length(weights))
                stop("dimensions of er and weights do not match")
        cov.mat <- as.matrix(cov.mat)
        if(length(er) != nrow(cov.mat))
                stop("dimensions of er and cov.mat do not match")
        if(any(diag(chol(cov.mat)) <= 0))
                stop("Covariance matrix not positive definite")

        #
        # create portfolio
        #
        er.port <- crossprod(er,weights)
        sd.port <- sqrt(weights %*% cov.mat %*% weights)
        ans <- list("call" = call,
```

```
              "er" = as.vector(er.port),
              "sd" = as.vector(sd.port),
              "weights" = weights)
        class(ans) <- "portfolio"
        ans
}

efficient.portfolio <-
function(er, cov.mat, target.return)
{
  # compute minimum variance portfolio subject to target return
  #
  # inputs:
  # er                                    N x 1 vector of expected returns
  # cov.mat                     N x N covariance matrix of returns
  # target.return         scalar, target expected return
  #
  # 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
  #
  call <- match.call()

  #
  # check for valid inputs
  #
  asset.names <- names(er)
  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")
  # remark: could use generalized inverse if cov.mat is positive semidefinite

  #
  # compute efficient portfolio
  #
  ones <- rep(1, length(er))
  top <- cbind(2*cov.mat, er, ones)
  bot <- cbind(rbind(er, ones), matrix(0,2,2))
  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) <- asset.names

  #
  # compute portfolio expected returns and variance
  #
  er.port <- crossprod(er,w)
  sd.port <- sqrt(w %*% cov.mat %*% w)
  ans <- list("call" = call,
              "er" = as.vector(er.port),
              "sd" = as.vector(sd.port),
              "weights" = w)
  class(ans) <- "portfolio"
  ans
}
```

```
globalMin.portfolio <-
function(er, cov.mat)
{
  # Compute global minimum variance portfolio
  #
  # inputs:
  # er                          N x 1 vector of expected returns
  # cov.mat             N x N return covariance matrix
  #
  # 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
  call <- match.call()

  #
  # check for valid inputs
  #
  asset.names <- names(er)
  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")
  # remark: could use generalized inverse if cov.mat is positive semi-definite

  #
  # compute global minimum portfolio
  #
  cov.mat.inv <- solve(cov.mat)
  one.vec <- rep(1,length(er))
#  w.gmin <- cov.mat.inv %*% one.vec/as.vector(one.vec %*% cov.mat.inv %*% one.vec)
  w.gmin <- rowSums(cov.mat.inv) / sum(cov.mat.inv)
  w.gmin <- as.vector(w.gmin)
  names(w.gmin) <- asset.names
  er.gmin <- crossprod(w.gmin,er)
  sd.gmin <- sqrt(t(w.gmin) %*% cov.mat %*% w.gmin)
  gmin.port <- list("call" = call,
                    "er" = as.vector(er.gmin),
                    "sd" = as.vector(sd.gmin),
                    "weights" = w.gmin)
  class(gmin.port) <- "portfolio"
  gmin.port
}


tangency.portfolio <-
function(er,cov.mat,risk.free)
{
  # compute tangency portfolio
  #
  # inputs:
  # er                             N x 1 vector of expected returns
  # cov.mat                N x N return covariance matrix
  # risk.free            scalar, risk-free rate
  #
  # output is portfolio object with the following elements
  # call                           captures function call
```

```
  # er                              portfolio expected return
  # sd                              portfolio standard deviation
  # weights               N x 1 vector of portfolio weights
  call <- match.call()

  #
  # check for valid inputs
  #
  asset.names <- names(er)
  if(risk.free < 0)
    stop("Risk-free rate must be positive")
  er <- as.vector(er)
  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")
  # remark: could use generalized inverse if cov.mat is positive semi-definite

  #
  # compute global minimum variance portfolio
  #
  gmin.port <- globalMin.portfolio(er,cov.mat)
  if(gmin.port$er < risk.free)
    stop("Risk-free rate greater than avg return on global minimum variance portfolio")

  #
  # compute tangency portfolio
  #
  cov.mat.inv <- solve(cov.mat)
  w.t <- cov.mat.inv %*% (er - risk.free) # tangency portfolio
  w.t <- as.vector(w.t/sum(w.t))        # normalize weights
  names(w.t) <- asset.names
  er.t <- crossprod(w.t,er)
  sd.t <- sqrt(t(w.t) %*% cov.mat %*% w.t)
  tan.port <- list("call" = call,
                   "er" = as.vector(er.t),
                   "sd" = as.vector(sd.t),
                   "weights" = w.t)
  class(tan.port) <- "portfolio"
  tan.port
}

efficient.frontier <-
function(er, cov.mat, nport=20, alpha.min=-0.5, alpha.max=1.5)
{
  # Compute efficient frontier with no short-sales constraints
  #
  # inputs:
  # er                      N x 1 vector of expected returns
  # cov.mat       N x N return covariance matrix
  # nport                   scalar, number of efficient portfolios to compute
  #
  # output is a Markowitz object with the following elements
  # call                    captures function call
  # er                      nport x 1 vector of expected returns on efficient porfolios
  # sd                      nport x 1 vector of std deviations on efficient portfolios
  # weights     nport x N matrix of weights on efficient portfolios
  call <- match.call()

  #
```

```
  # check for valid inputs
  #
  asset.names <- names(er)
  er <- as.vector(er)
  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")

  #
  # create portfolio names
  #
  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 <- globalMin.portfolio(er,cov.mat)
  w.gmin <- port.gmin$weights

  #
  # compute efficient frontier as convex combinations of two efficient portfolios
  # 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
  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,asset.names)

  #
  # summarize results
  #
  ans <- list("call" = call,
              "er" = er.e,
              "sd" = sd.e,
              "weights" = we.mat)
  class(ans) <- "Markowitz"
  ans
}

#
# print method for portfolio object
print.portfolio <- function(x, ...)
{
```

```
  cat("Call:\n")
  print(x$call, ...)
  cat("\nPortfolio expected return:    ", format(x$er, ...), "\n")
  cat("Portfolio standard deviation: ", format(x$sd, ...), "\n")
  cat("Portfolio weights:\n")
  print(round(x$weights,4), ...)
  invisible(x)
}

#
# summary method for portfolio object
summary.portfolio <- function(object, risk.free=NULL, ...)
# risk.free                     risk-free rate. If not null then
#                               compute and print Sharpe ratio
#
{
  cat("Call:\n")
  print(object$call)
  cat("\nPortfolio expected return:    ", format(object$er, ...), "\n")
  cat(  "Portfolio standard deviation: ", format(object$sd, ...), "\n")
  if(!is.null(risk.free)) {
    SharpeRatio <- (object$er - risk.free)/object$sd
    cat("Portfolio Sharpe Ratio:       ", format(SharpeRatio), "\n")
  }
  cat("Portfolio weights:\n")
  print(round(object$weights,4), ...)
  invisible(object)
}
# hard-coded 4 digits; prefer to let user control, via ... or other argument

#
# plot method for portfolio object
plot.portfolio <- function(object, ...)
{
  asset.names <- names(object$weights)
  barplot(object$weights, names=asset.names,
          xlab="Assets", ylab="Weight", main="Portfolio Weights", ...)
  invisible()
}

#
# print method for Markowitz object
print.Markowitz <- function(x, ...)
{
  cat("Call:\n")
  print(x$call)
  xx <- rbind(x$er,x$sd)
  dimnames(xx)[[1]] <- c("ER","SD")
  cat("\nFrontier portfolios' expected returns and standard deviations\n")
  print(round(xx,4), ...)
  invisible(x)
}
# hard-coded 4, should let user control

#
# summary method for Markowitz object
summary.Markowitz <- function(object, risk.free=NULL)
{
  call <- object$call
  asset.names <- colnames(object$weights)
  port.names <- rownames(object$weights)
```

```
  if(!is.null(risk.free)) {
    # compute efficient portfolios with a risk-free asset
    nport <- length(object$er)
    sd.max <- object$sd[1]
    sd.e <- seq(from=0,to=sd.max,length=nport)
    names(sd.e) <- port.names

    #
    # get original er and cov.mat data from call
    er <- eval(object$call$er)
    cov.mat <- eval(object$call$cov.mat)

    #
    # compute tangency portfolio
    tan.port <- tangency.portfolio(er,cov.mat,risk.free)
    x.t <- sd.e/tan.port$sd             # weights in tangency port
    rf <- 1 - x.t                       # weights in t-bills
    er.e <- risk.free + x.t*(tan.port$er - risk.free)
    names(er.e) <- port.names
    we.mat <- x.t %o% tan.port$weights  # rows are efficient portfolios
    dimnames(we.mat) <- list(port.names, asset.names)
    we.mat <- cbind(rf,we.mat)
  }
  else {
    er.e <- object$er
    sd.e <- object$sd
    we.mat <- object$weights
  }
  ans <- list("call" = call,
              "er"=er.e,
              "sd"=sd.e,
              "weights"=we.mat)
  class(ans) <- "summary.Markowitz"
  ans
}

print.summary.Markowitz <- function(x, ...)
{
        xx <- rbind(x$er,x$sd)
        port.names <- names(x$er)
        asset.names <- colnames(x$weights)
        dimnames(xx)[[1]] <- c("ER","SD")
        cat("Frontier portfolios' expected returns and standard deviations\n")
        print(round(xx,4), ...)
        cat("\nPortfolio weights:\n")
        print(round(x$weights,4), ...)
        invisible(x)
}
# hard-coded 4, should let user control

#
# plot efficient frontier
#
# things to add: plot original assets with names
# tangency portfolio
# global min portfolio
# risk free asset and line connecting rf to tangency portfolio
#
plot.Markowitz <- function(object, plot.assets=FALSE, ...)
# plot.assets           logical. If true then plot asset sd and er
{
```

```
  if (!plot.assets) {
     y.lim=c(0,max(object$er))
     x.lim=c(0,max(object$sd))
     plot(object$sd,object$er,type="b",xlim=x.lim, ylim=y.lim,
          xlab="Portfolio SD", ylab="Portfolio ER",
          main="Efficient Frontier", ...)
     }
  else {
          call = object$call
          mu.vals = eval(call$er)
          sd.vals = sqrt( diag( eval(call$cov.mat) ) )
          y.lim = range(c(0,mu.vals,object$er))
          x.lim = range(c(0,sd.vals,object$sd))
          plot(object$sd,object$er,type="b", xlim=x.lim, ylim=y.lim,
          xlab="Portfolio SD", ylab="Portfolio ER",
          main="Efficient Frontier", ...)
        text(sd.vals, mu.vals, labels=names(mu.vals))
  }
  invisible()
}
```