Introduction to PortfolioAnalytics

Ross Bennett

April 18, 2015

Abstract

The purpose of this vignette is to demonstrate the new interface in PortfolioAnalytics to specify a portfolio object, add constraints and objectis, and run optimizations.

Contents

1 Getting Started

1.1 Load Packages

Load the necessary packages.

```
> library(PortfolioAnalytics)
```

1.2 Data

The edhec data set from the PerformanceAnalytics package will be used as example data.

```
> data(edhec)
> # Use the first 4 columns in edhec for a returns object
> returns <- edhec[, 1:4]
> colnames(returns) <- c("CA", "CTAG", "DS", "EM")
> print(head(returns, 5))
```

```
               CA    CTAG      DS      EM
1997-01-31 0.0119  0.0393  0.0178  0.0791
1997-02-28 0.0123  0.0298  0.0122  0.0525
1997-03-31 0.0078 -0.0021 -0.0012 -0.0120
1997-04-30 0.0086 -0.0170  0.0030  0.0119
1997-05-31 0.0156 -0.0015  0.0233  0.0315
```

```
> # Get a character vector of the fund names
> fund.names <- colnames(returns)
```

2 Creating the Portfolio Object

The portfolio object is instantiated with the `portfolio.spec` function. The main argument to `portfolio.spec` is assets, this is a required argument. The assets argument can be a scalar value for the number of assets, a character vector of fund names, or a named vector of initial weights. If initial weights are not specified, an equal weight portfolio will be assumed.

The `pspec` object is an S3 object of class "portfolio". When first created, the portfolio object has an element named `assets` with the initial weights, an element named `category_labels`, an element named `weight_seq` with sequence of weights if specified, an empty constraints list and an empty objectives list.

```
> # Specify a portfolio object by passing a character vector for the
> # assets argument.
> pspec <- portfolio.spec(assets=fund.names)
> print.default(pspec)

$assets
  CA CTAG   DS   EM
0.25 0.25 0.25 0.25


$category_labels
NULL


$weight_seq
NULL


$constraints
```

```
list()

$objectives
list()

$call
portfolio.spec(assets = fund.names)

attr(,"class")
[1] "portfolio.spec" "portfolio"
```

3 Adding Constraints to the Portfolio Object

Adding constraints to the portfolio object is done with `add.constraint`. The `add.constraint` function is the main interface for adding and/or updating constraints to the portfolio object. This function allows the user to specify the portfolio to add the constraints to, the type of constraints, arguments for the constraint, and whether or not to enable the constraint (`enabled=TRUE` is the default). If updating an existing constraint, the indexnum argument can be specified.

3.1 Sum of Weights Constraint

The `weight_sum` constraint specifies the constraint on the sum of the weights. Aliases for the `weight_sum` constraint type include `weight` and `leverage`. Here we add a constraint that the weights must sum to 1, or the full investment constraint.

```
> # Add the full investment constraint that specifies the weights must sum to 1.
> pspec <- add.constraint(portfolio=pspec,
+                         type="weight_sum",
+                         min_sum=1,
+                         max_sum=1)
```

There are two special cases for the leverage constraint:

1. The sum of the weights equal 1, i.e. the full investment constraint. The full investment constraint can be specified with `type="full_investment"`. This automatically sets `min_sum=1` and `max_sum=1`.

2. The sum of the weights equal 0, i.e. the dollar neutral or active constraint. This constraint can be specified with `type="dollar_neutral"` or `type="active"`.

```
> # The full investment constraint can also be specified with type="full_investment"
> # pspec <- add.constraint(portfolio=pspec, type="full_investment")
>
> # Another common constraint is that portfolio weights sum to 0.
> # This can be specified any of the following ways
> # pspec <- add.constraint(portfolio=pspec, type="weight_sum",
> #                         min_sum=0,
> #                         max_sum=0)
> # pspec <- add.constraint(portfolio=pspec, type="dollar_neutral")
> # pspec <- add.constraint(portfolio=pspec, type="active")
```

3.2 Box Constraint

Box constraints allows the user to specify upper and lower bounds on the weights of the assets. Here we add box constraints for the asset weights so that the minimum weight of any asset must be greater than or equal to 0.05 and the maximum weight of any asset must be less than or equal to 0.4. The values for min and max can be passed in as scalars or vectors. If min and max are scalars, the values for min and max will be replicated as vectors to the length of assets. If min and max are not specified, a minimum weight of 0 and maximum weight of 1 are assumed. Note that min and max can be specified as vectors with different weights for linear inequality constraints.

```
> # Add box constraints
> pspec <- add.constraint(portfolio=pspec,
+                         type="box",
+                         min=0.05,
+                         max=0.4)
>
> # min and max can also be specified per asset
> # pspec <- add.constraint(portfolio=pspec,
> #                         type="box",
> #                         min=c(0.05, 0, 0.08, 0.1),
> #                         max=c(0.4, 0.3, 0.7, 0.55))
>
> # A special case of box constraints is long only where min=0 and max=1
> # The default action is long only if min and max are not specified
> # pspec <- add.constraint(portfolio=pspec, type="box")
```

```
> # pspec <- add.constraint(portfolio=pspec, type="long_only")
```

3.3 Group Constraint

Group constraints allow the user to specify the the sum of weights by group. Group constraints are currently supported by the ROI, DEoptim, and random portfolio solvers. The following code groups the assets such that the first 3 assets are grouped together labeled GroupA and the fourth asset is in its own group labeled GroupB. The `group_min` argument specifies that the sum of the weights in GroupA must be greater than or equal to 0.1 and the sum of the weights in GroupB must be greater than or equal to 0.15. The `group_max` argument specifies that the sum of the weights in GroupA must be less than or equal to 0.85 and the sum of the weights in GroupB must be less than or equal to 0.55.The `group_labels` argument is optional and is useful if groups is not a named list for labeling groups in terms of market capitalization, sector, etc.

```
> # Add group constraints
> pspec <- add.constraint(portfolio=pspec, type="group",
+                         groups=list(groupA=c(1, 2, 3),
+                                     grouB=4),
+                         group_min=c(0.1, 0.15),
+                         group_max=c(0.85, 0.55))
```

3.4 Position Limit Constraint

The position limit constraint allows the user to specify limits on the number of assets with non-zero, long, or short positions. The ROI solver interfaces to the Rglpk package (i.e. using the glpk plugin) for solving maximizing return and ETL/ES/cVaR objectives. The Rglpk package supports integer programming and thus supports position limit constraints for the `max_pos` argument. The quadprog package does not support integer programming, and therefore `max_pos` is not supported for the ROI solver using the quadprog plugin. Note that `max_pos_long` and `max_pos_short` are not supported for either ROI solver. All position limit constraints are fully supported for DEoptim and random solvers.

```
> # Add position limit constraint such that we have a maximum number of three assets with non-zero
> pspec <- add.constraint(portfolio=pspec, type="position_limit", max_pos=3)
>
> # Can also specify maximum number of long positions and short positions
> # pspec <- add.constraint(portfolio=pspec, type="position_limit", max_pos_long=3, max_pos_short=
```

3.5 Diversification Constraint

The diversification constraint allows the user to target diversification. Diversification is defined as $diversification = \sum_{i=1}^{N} w_i^2$ for N assets. The diversification constraint is implemented for the global optimizers by applying a penalty if the diversification value is more than 5% away from `div_target`. Note that diversification as a constraint is not supported for the ROI solvers, it is only supported for the global numeric solvers.

```
> pspec <- add.constraint(portfolio=pspec, type="diversification", div_target=0.7)
```

3.6 Turnover Constraint

A target turnover can be specified as a constraint. The turnover is calculated from a set of initial weights. The initial weights can be specified, by default they are the initial weights in the portfolio object. The turnover constraint is implemented for the global optimizers by applying a penalty if the turnover value is more than 5% away from `turnover_target`. Note that the turnover constraint is not currently supported for quadratic utility and minimum variance problems using the ROI solver.

```
> pspec <- add.constraint(portfolio=pspec, type="turnover", turnover_target=0.2)
```

3.7 Target Return Constraint

The target return constraint allows the user to specify a target mean return.

```
> pspec <- add.constraint(portfolio=pspec, type="return", return_target=0.007)
```

3.8 Factor Exposure Constraint

The factor exposure constraint allows the user to set upper and lower bounds on exposures to risk factors. The exposures can be passed in as a vector or matrix. Here we specify a vector for `B` with arbitrary values, e.g. betas of the assets, with a market risk exposure range of 0.6 to 0.9.

```
> pspec <- add.constraint(portfolio=pspec, type="factor_exposure",
+                         B=c(-0.08, 0.37, 0.79, 1.43),
+                         lower=0.6, upper=0.9)
```

3.9 Transaction Cost Constraint

The transaction cost constraint allows the user to specify proportional transaction costs. Proportional transaction cost constraints can be implemented for quadratic utility and minimum variance

problems using the ROI solver. Transaction costs are supported as a penalty for the global numeric solvers. Here we add the transaction cost contraint with the proportional transaction cost value of 1%.

```
> pspec <- add.constraint(portfolio=pspec, type="transaction_cost", ptc=0.01)
```

The print method for the portfolio object shows a concise view of the portfolio and the constraints that have been added.

```
> print(pspec)
```

```
***************************************************
PortfolioAnalytics Portfolio Specification
***************************************************

Call:
portfolio.spec(assets = fund.names)

Number of assets: 4
Asset Names
[1] "CA"   "CTAG" "DS"   "EM"

Constraints
Enabled constraint types
                - weight_sum
                - box
                - group
                - position_limit
                - diversification
                - turnover
                - return
                - factor_exposure
                - transaction_cost
```

The summary method gives a more detailed view of the constraints.

```
> summary(pspec)
```

```
$assets
  CA CTAG   DS   EM
```

```
0.25 0.25 0.25 0.25

$enabled_constraints
$enabled_constraints[[1]]
An object containing 6 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[2]]
An object containing 5 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[3]]
An object containing 7 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[4]]
An object containing 4 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[5]]
An object containing 4 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[6]]
An object containing 4 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[7]]
An object containing 4 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[8]]
An object containing 6 constraints.
Some constraints are of type nonlinear.
```

```
$enabled_constraints[[9]]
An object containing 4 constraints.
Some constraints are of type nonlinear.


$disabled_constraints
list()

$enabled_objectives
list()

$disabled_objectives
list()

attr(,"class")
[1] "summary.portfolio"
```

This demonstrates adding constraints to the portfolio object. As an alternative to adding constraints directly to the portfolio object, constraints can be specified as separate objects.

3.10 Specifying Constraints as Separate Objects

The following examples will demonstrate how to specify constraints as separate objects for all constraints types.

```
> # full investment constraint
> weight_constr <- weight_sum_constraint(min_sum=1, max_sum=1)
> # box constraint
> box_constr <- box_constraint(assets=pspec$assets, min=0, max=1)
> # group constraint
> group_constr <- group_constraint(assets=pspec$assets,
+                                  groups=list(c(1, 2, 3),
+                                              4),
+                                  group_min=c(0.1, 0.15),
+                                  group_max=c(0.85, 0.55),
+                                  group_labels=c("GroupA", "GroupB"))
> # position limit constraint
```

```
> poslimit_constr <- position_limit_constraint(assets=pspec$assets, max_pos=3)
> # diversification constraint
> div_constr <- diversification_constraint(div_target=0.7)
> # turnover constraint
> to_constr <- turnover_constraint(turnover_target=0.2)
> # target return constraint
> ret_constr <- return_constraint(return_target=0.007)
> # factor exposure constraint
> exp_constr <- factor_exposure_constraint(assets=pspec$assets,
+                                          B=c(-0.08, 0.37, 0.79, 1.43),
+                                          lower=0.6, upper=0.9)
> # transaction cost constraint
> ptc_constr <- transaction_cost_constraint(assets=pspec$assets, ptc=0.01)
```

4 Adding Objectives

Objectives can be added to the portfolio object with `add.objective`. The `add.objective` function is the main function for adding and/or updating business objectives to the portfolio object. This function allows the user to specify the `portfolio` to add the objectives to, the `type` (currently 'return', 'risk', 'risk_budget', or 'weight_concentration'), `name` of the objective function, `arguments` to the objective function, and whether or not to `enable` the objective. If updating an existing constraint, the `indexnum` argument can be specified.

4.1 Portfolio Risk Objective

The portfolio risk objective allows the user to specify a risk function to minimize Here we add a risk objective to minimize portfolio expected tail loss with a confidence level of 0.95. Other default arguments to the function can be passed in as a named list to arguments. Note that the name of the function must correspond to a function in R. Many functions are available in the `PerformanceAnalytics` package or a user defined function.

```
> pspec <- add.objective(portfolio=pspec,
+                        type='risk',
+                        name='ETL',
+                        arguments=list(p=0.95))
```

4.2 Portfolio Return Objective

The return objective allows the user to specify a return function to maximize. Here we add a return objective to maximize the portfolio mean return.

```
> pspec <- add.objective(portfolio=pspec,
+                        type='return',
+                        name='mean')
```

4.3 Portfolio Risk Budget Objective

The portfolio risk objective allows the user to specify constraints to minimize component contribution (i.e. equal risk contribution) or specify upper and lower bounds on percentage risk contribution. Here we specify that no asset can contribute more than 30% to total portfolio risk. See the risk budget optimization vignette for more detailed examples of portfolio optimizations with risk budgets.

```
> pspec <- add.objective(portfolio=pspec, type="risk_budget", name="ETL",
+                        arguments=list(p=0.95), max_prisk=0.3)
>
> # for an equal risk contribution portfolio, set min_concentration=TRUE
> # pspec <- add.objective(portfolio=pspec, type="risk_budget", name="ETL",
> #                          arguments=list(p=0.95), min_concentration=TRUE)
```

4.4 Portfolio Weight Concentration Objective

The weight concentration objective allows the user to specify an objective to minimize concentration as measured by the Herfindahl-Hirschman Index. For otpimization problems solved with the global numeric optimizers, the portfolio HHI value is penalized using `conc_aversion` value as the multiplier.

For quadratic utility problems with weight concentration as an objective using the ROI solver, this is implemented as a penalty to the objective function. The objective function is implemented as follows:

$$\underset{\boldsymbol{w}}{\text{maximize}} \boldsymbol{w}'\boldsymbol{\mu} - \frac{\lambda}{2}(\boldsymbol{w}'\boldsymbol{\Sigma}\boldsymbol{w} + \lambda_{hhi} * HHI) \tag{1}$$

$$\tag{2}$$

Where μ is the estimated mean asset returns, λ is the risk aversion parameter, $lambda_{hhi}$ is the concentration aversion parameter, HHI is the portfolio HHI, $\mathbf{\Sigma}$ is the estimated covariance matrix of asset returns and $\boldsymbol{w}$ is the set of weights.

Here we add a weight concentration objective for the overall portfolio HHI.

```
> pspec <- add.objective(portfolio=pspec, type="weight_concentration",
+                        name="HHI", conc_aversion=0.1)
```

The weight concentration aversion parameter by groups can also be specified. Here we add a weight concentration objective specifying groups and concentration aversion parameters by group.

```
> pspec <- add.objective(portfolio=pspec, type="weight_concentration",
+                        name="HHI",
+                        conc_aversion=c(0.03, 0.06),
+                        conc_groups=list(c(1, 2),
+                                         c(3, 4)))
```

The print method for the portfolio object will now show all the constraints and objectives that have been added.

```
> print(pspec)

***************************************************
PortfolioAnalytics Portfolio Specification
***************************************************

Call:
portfolio.spec(assets = fund.names)

Number of assets: 4
Asset Names
[1] "CA"   "CTAG" "DS"   "EM"

Constraints
Enabled constraint types
                - weight_sum
                - box
                - group
                - position_limit
```

```
                - diversification
                - turnover
                - return
                - factor_exposure
                - transaction_cost

Objectives:
Enabled objective names
                - ETL
                - mean
                - ETL
                - HHI
                - HHI
```

The `summary` function gives a more detailed view.

```
> summary(pspec)

$assets
  CA CTAG   DS   EM
0.25 0.25 0.25 0.25

$enabled_constraints
$enabled_constraints[[1]]
An object containing 6 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[2]]
An object containing 5 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[3]]
An object containing 7 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[4]]
An object containing 4 constraints.
```

```
Some constraints are of type nonlinear.

$enabled_constraints[[5]]
An object containing 4 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[6]]
An object containing 4 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[7]]
An object containing 4 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[8]]
An object containing 6 constraints.
Some constraints are of type nonlinear.

$enabled_constraints[[9]]
An object containing 4 constraints.
Some constraints are of type nonlinear.


$disabled_constraints
list()

$enabled_objectives
$enabled_objectives[[1]]
$name
[1] "ETL"

$target
NULL

$arguments
```

```
$arguments$p
[1] 0.95

$arguments$portfolio_method
[1] "single"



$enabled
[1] TRUE

$multiplier
[1] 1

$call
add.objective(portfolio = pspec, type = "risk", name = "ETL",
    arguments = list(p = 0.95))

attr(,"class")
[1] "portfolio_risk_objective" "objective"

$enabled_objectives[[2]]
$name
[1] "mean"

$target
NULL

$arguments
list()

$enabled
[1] TRUE

$multiplier
[1] -1
```

```
$call
add.objective(portfolio = pspec, type = "return", name = "mean")

attr(,"class")
[1] "return_objective" "objective"

$enabled_objectives[[3]]
$name
[1] "ETL"

$target
NULL

$arguments
$arguments$p
[1] 0.95

$arguments$portfolio_method
[1] "component"



$enabled
[1] TRUE

$multiplier
[1] 1

$max_prisk
  CA CTAG   DS   EM
 0.3  0.3  0.3  0.3

$min_concentration
[1] FALSE
```

```
$min_difference
[1] FALSE

$call
add.objective(portfolio = pspec, type = "risk_budget", name = "ETL",
    arguments = list(p = 0.95), max_prisk = 0.3)

attr(,"class")
[1] "risk_budget_objective" "objective"

$enabled_objectives[[4]]
$name
[1] "HHI"

$target
NULL

$arguments
list()

$enabled
[1] TRUE

$multiplier
[1] 1

$conc_aversion
[1] 0.1

$call
add.objective(portfolio = pspec, type = "weight_concentration",
    name = "HHI", conc_aversion = 0.1)

attr(,"class")
[1] "weight_concentration_objective" "objective"
```

```
$enabled_objectives[[5]]
$name
[1] "HHI"

$target
NULL

$arguments
$arguments$groups
$arguments$groups[[1]]
[1] 1 2

$arguments$groups[[2]]
[1] 3 4




$enabled
[1] TRUE

$multiplier
[1] 1

$conc_aversion
[1] 0.03 0.06

$conc_groups
$conc_groups[[1]]
[1] 1 2

$conc_groups[[2]]
[1] 3 4
```

```
$call
add.objective(portfolio = pspec, type = "weight_concentration",
    name = "HHI", conc_aversion = c(0.03, 0.06), conc_groups = list(c(1,
        2), c(3, 4)))

attr(,"class")
[1] "weight_concentration_objective" "objective"



$disabled_objectives
list()

attr(,"class")
[1] "summary.portfolio"
```

5 Solvers

The PortfolioAnalytics package currently supports random portfolios, DEoptim, pso, GenSA, and ROI as back ends. Note that some of the QP/LP problems are solved directly with Rglpk and quadprog. The solver can be specified with the `optimize_method` argument in `optimize.portfolio` and `optimize.portfolio.rebalancing`.

5.1 DEoptim

PortfolioAnalytics uses the `DEoptim` function from the R package `DEoptim`. Differential evolution is a stochastic global optimization algorithm. See `?DEoptim` and the references contained therein for more information. See also Large scale portfolio optimization with DEoptim.

5.2 Random Portfolios

PortfolioAnalytics has three methods to generate random portfolios.

1. The 'sample' method to generate random portfolios is based on an idea by Pat Burns. This is the most flexible method, but also the slowest, and can generate portfolios to satisfy leverage, box, group, and position limit constraints.

2. The 'simplex' method to generate random portfolios is based on a paper by W. T. Shaw. The simplex method is useful to generate random portfolios with the full investment constraint,

where the sum of the weights is equal to 1, and min box constraints. Values for `min_sum` and `max_sum` of the leverage constraint will be ignored, the sum of weights will equal 1. All other constraints such as the box constraint max, group and position limit constraints will be handled by elimination. If the constraints are very restrictive, this may result in very few feasible portfolios remaining. Another key point to note is that the solution may not be along the vertexes depending on the objective. For example, a risk budget objective will likely place the portfolio somewhere on the interior.

3. The 'grid' method to generate random portfolios is based on the `gridSearch` function in package `NMOF`. The grid search method only satisfies the `min` and `max` box constraints. The `min_sum` and `max_sum` leverage constraint will likely be violated and the weights in the random portfolios should be normalized. Normalization may cause the box constraints to be violated and will be penalized in `constrained_objective`.

The following plots illustrate the various methods to generate random portfolios.

```
> R <- edhec[, 1:4]
> # set up simple portfolio with leverage and box constraints
> pspec <- portfolio.spec(assets=colnames(R))
> pspec <- add.constraint(portfolio=pspec, type="leverage",
+                         min_sum=0.99, max_sum=1.01)
> pspec <- add.constraint(portfolio=pspec, type="box", min=0, max=1)
> # generate random portfolios using the 3 methods
> rp1 <- random_portfolios(portfolio=pspec, permutations=5000,
+                          rp_method='sample')
> rp2 <- random_portfolios(portfolio=pspec, permutations=5000,
+                          rp_method='simplex')
> rp3 <- random_portfolios(portfolio=pspec, permutations=5000,
+                          rp_method='grid')
> # show feasible portfolios in mean-StdDev space
> tmp1.mean <- apply(rp1, 1, function(x) mean(R %*% x))
> tmp1.StdDev <- apply(rp1, 1, function(x) StdDev(R=R, weights=x))
> tmp2.mean <- apply(rp2, 1, function(x) mean(R %*% x))
> tmp2.StdDev <- apply(rp2, 1, function(x) StdDev(R=R, weights=x))
> tmp3.mean <- apply(rp3, 1, function(x) mean(R %*% x))
> tmp3.StdDev <- apply(rp3, 1, function(x) StdDev(R=R, weights=x))
> # plot feasible portfolios
```

```
> plot(x=tmp1.StdDev, y=tmp1.mean, col="gray", main="Random Portfolio Methods",
+      ylab="mean", xlab="StdDev")
> points(x=tmp2.StdDev, y=tmp2.mean, col="red", pch=2)
> points(x=tmp3.StdDev, y=tmp3.mean, col="lightgreen", pch=5)
> legend("bottomright", legend=c("sample", "simplex", "grid"),
+        col=c("gray", "red", "lightgreen"),
+        pch=c(1, 2, 5), bty="n")
```

Figure 1 shows the feasible space using the different random portfolio methods. The 'sample' method has relatively even coverage of the feasible space. The 'simplex' method also has relatively even coverage of the space, but it is also more concentrated around the assets. The 'grid' method is pushed to the interior of the space due to the normalization.

The `fev` argument controls the face-edge-vertex biasing. Higher values for `fev` will result in the weights vector more concentrated on a single asset. This can be seen in the following charts.

```
> fev <- 0:5
> par(mfrow=c(2, 3))
> for(i in 1:length(fev)){
+   rp <- rp_simplex(portfolio=pspec, permutations=2000, fev=fev[i])
+   tmp.mean <- apply(rp, 1, function(x) mean(R %*% x))
+   tmp.StdDev <- apply(rp, 1, function(x) StdDev(R=R, weights=x))
+   plot(x=tmp.StdDev, y=tmp.mean, main=paste("FEV =", fev[i]),
+        ylab="mean", xlab="StdDev", col=rgb(0, 0, 100, 50, maxColorValue=255))
+ }
> par(mfrow=c(1,1))
```

Figure 2 shows the feasible space varying the fev values.

The `fev` argument can be passed in as a vector for more control over the coverage of the feasible space. The default value is `fev=0:5`.

```
> par(mfrow=c(1, 2))
> # simplex
> rp_simplex <- random_portfolios(portfolio=pspec, permutations=2000,
+                                 rp_method='simplex')
> tmp.mean <- apply(rp_simplex, 1, function(x) mean(R %*% x))
> tmp.StdDev <- apply(rp_simplex, 1, function(x) StdDev(R=R, weights=x))
> plot(x=tmp.StdDev, y=tmp.mean, main="rp_method=simplex fev=0:5",
```

```
+      ylab="mean", xlab="StdDev", col=rgb(0, 0, 100, 50, maxColorValue=255))
> #sample
> rp_sample <- random_portfolios(portfolio=pspec, permutations=2000,
+                                rp_method='sample')
> tmp.mean <- apply(rp_sample, 1, function(x) mean(R %*% x))
> tmp.StdDev <- apply(rp_sample, 1, function(x) StdDev(R=R, weights=x))
> plot(x=tmp.StdDev, y=tmp.mean, main="rp_method=sample",
+      ylab="mean", xlab="StdDev", col=rgb(0, 0, 100, 50, maxColorValue=255))
> par(mfrow=c(1,1))
```

5.3 pso

PortfolioAnalytics uses the `psoptim` function from the R package `pso`. Particle swarm optimization is a heuristic optimization algorithm. See `?psoptim` and the references contained therein for more information.

5.4 GenSA

PortfolioAnalytics uses the `GenSA` function from the R package `GenSA`. Generalized simmulated annealing is generic probabilistic heuristic optimization algorithm. See `?GenSA` and the references contained therein for more information.

5.5 ROI

The `ROI` package serves as an interface to the `Rglpk` package and the `quadprog` package to solve linear and quadratic programming problems. The interface to the `ROI` package solves a limited type of convex optimization problems:

1. Maxmimize portfolio return subject leverage, box, group, position limit, target mean return, and/or factor exposure constraints on weights.
2. Minimize portfolio variance subject to leverage, box, group, turnover, and/or factor exposure constraints (otherwise known as global minimum variance portfolio).
3. Minimize portfolio variance subject to leverage, box, group, and/or factor exposure constraints and a desired portfolio return.
4. Maximize quadratic utility subject to leverage, box, group, target mean return, turnover, and/or factor exposure constraints and risk aversion parameter. (The risk aversion parameter is passed into `optimize.portfolio` as an added argument to the `portfolio` object).

5. Minimize ETL subject to leverage, box, group, position limit, target mean return, and/or factor exposure constraints and target portfolio return.

6 Optimization

The previous sections demonstrated how to specify a portfolio object, add constraints, add objectives, and the solvers available. This section will demonstrate run the optimizations via `optimize.portfolio`. Only a small number of examples will be shown here, see the demos for several more examples.

6.1 Initial Portfolio Object

```
> library(DEoptim)
> library(ROI)
> require(ROI.plugin.glpk)
> require(ROI.plugin.quadprog)
> data(edhec)
> R <- edhec[, 1:6]
> colnames(R) <- c("CA", "CTAG", "DS", "EM", "EQMN", "ED")
> funds <- colnames(R)
> # Create an initial portfolio object with leverage and box constraints
> init <- portfolio.spec(assets=funds)
> init <- add.constraint(portfolio=init, type="leverage",
+                        min_sum=0.99, max_sum=1.01)
> init <- add.constraint(portfolio=init, type="box", min=0.05, max=0.65)
```

6.2 Maximize mean return with ROI

Add an objective to maximize mean return.

```
> maxret <- add.objective(portfolio=init, type="return", name="mean")
```

Run the optimization.

```
> opt_maxret <- optimize.portfolio(R=R, portfolio=maxret,
+                                  optimize_method="ROI",
+                                  trace=TRUE)
> print(opt_maxret)
```

```
**********************************
PortfolioAnalytics Optimization
**********************************

Call:
optimize.portfolio(R = R, portfolio = maxret, optimize_method = "ROI",
    trace = TRUE)

Optimal Weights:
  CA CTAG   DS   EM EQMN   ED
0.05 0.05 0.16 0.65 0.05 0.05

Objective Measure:
    mean
0.007959
```

Chart the weights and optimal portfolio in risk-return space. The weights and a risk-reward scatter plot can be plotted separately as shown below with the `chart.Weights` and `chart.RiskReward` functions. The `plot` function will plot the weights and risk-reward scatter together.

```
> plot(opt_maxret, risk.col="StdDev", return.col="mean",
+      main="Maximum Return Optimization", chart.assets=TRUE,
+      xlim=c(0, 0.05), ylim=c(0,0.0085))
```

6.3 Minimize variance with ROI

Add an objective to minimize portfolio variance.

```
> minvar <- add.objective(portfolio=init, type="risk", name="var")
```

Run the optimization. Note that although 'var' is the risk metric, 'StdDev' is returned as an objective measure.

```
> opt_minvar <- optimize.portfolio(R=R, portfolio=minvar,
+                                  optimize_method="ROI", trace=TRUE)
> print(opt_minvar)

**********************************
PortfolioAnalytics Optimization
```

```
**********************************

Call:
optimize.portfolio(R = R, portfolio = minvar, optimize_method = "ROI",
    trace = TRUE)

Optimal Weights:
  CA CTAG   DS   EM EQMN   ED
0.05 0.14 0.05 0.05 0.65 0.05

Objective Measure:
 StdDev
0.01005
```

Chart the weights and optimal portfolio in risk-return space.

```
> plot(opt_minvar, risk.col="StdDev", return.col="mean",
+      main="Minimum Variance Optimization", chart.assets=TRUE,
+      xlim=c(0, 0.05), ylim=c(0,0.0085))
```

6.4 Maximize quadratic utility with ROI

Add mean and var objectives for quadratic utility. Note that the risk aversion parameter for quadratic utility is specifed in the objective as shown below.

```
> qu <- add.objective(portfolio=init, type="return", name="mean")
> qu <- add.objective(portfolio=qu, type="risk", name="var", risk_aversion=0.25)
```

Run the optimization.

```
> opt_qu <- optimize.portfolio(R=R, portfolio=qu,
+                              optimize_method="ROI",
+                              trace=TRUE)
> print(opt_qu)

**********************************
PortfolioAnalytics Optimization
**********************************
```

```
Call:
optimize.portfolio(R = R, portfolio = qu, optimize_method = "ROI",
    trace = TRUE)

Optimal Weights:
    CA   CTAG     DS     EM   EQMN     ED
0.0500 0.0500 0.2714 0.5386 0.0500 0.0500

Objective Measure:
    mean
0.007926



 StdDev
0.02663
```

```
> plot(opt_qu, risk.col="StdDev", return.col="mean",
+      main="Quadratic Utility Optimization", chart.assets=TRUE,
+      xlim=c(0, 0.05), ylim=c(0, 0.0085))
```

6.5 Minimize expected tail loss with ROI

Add ETL objective.

```
> etl <- add.objective(portfolio=init, type="risk", name="ETL")
```

Run the optimization.

```
> opt_etl <- optimize.portfolio(R=R, portfolio=etl,
+                               optimize_method="ROI",
+                               trace=TRUE)
> print(opt_etl)

**********************************
PortfolioAnalytics Optimization
**********************************

Call:
```

```
optimize.portfolio(R = R, portfolio = etl, optimize_method = "ROI",
    trace = TRUE)

Optimal Weights:
    CA   CTAG     DS     EM   EQMN     ED
0.0500 0.2968 0.0500 0.0500 0.4932 0.0500

Objective Measure:
    ETL
0.01967
```

```
> plot(opt_etl, risk.col="ES", return.col="mean",
+      main="ETL Optimization", chart.assets=TRUE,
+      xlim=c(0, 0.14), ylim=c(0,0.0085))
```

6.6 Maximize mean return per unit ETL with random portfolios

Add mean and ETL objectives.

```
> meanETL <- add.objective(portfolio=init, type="return", name="mean")
> meanETL <- add.objective(portfolio=meanETL, type="risk", name="ETL",
+                          arguments=list(p=0.95))
```

Run the optimization. The default random portfolio method is 'sample'.

```
> opt_meanETL <- optimize.portfolio(R=R, portfolio=meanETL,
+                                   optimize_method="random",
+                                   trace=TRUE, search_size=2000)
> print(opt_meanETL)

***********************************
PortfolioAnalytics Optimization
***********************************

Call:
optimize.portfolio(R = R, portfolio = meanETL, optimize_method = "random",
    search_size = 2000, trace = TRUE)
```

```
Optimal Weights:
   CA  CTAG    DS    EM  EQMN    ED
0.062 0.366 0.100 0.070 0.344 0.052

Objective Measures:
    mean
0.006606



    ETL
0.02035
```

The optimization was run with `trace=TRUE` so that iterations and other output from random portfolios is stored in the `opt_meanETL` object. The `extractStats` function can be used to get a matrix of the weights and objective measures at each iteration.

```
> stats_meanETL <- extractStats(opt_meanETL)
> dim(stats_meanETL)

[1] 1999    9

> head(stats_meanETL)

                     mean         ETL         out      w.CA    w.CTAG      w.DS
.rnd.portf.1 0.007120395 0.04511960 0.03799921 0.1666667 0.1666667 0.1666667
.rnd.portf.2 0.006908307 0.06177758 0.05486928 0.3680000 0.0680000 0.0500000
.rnd.portf.3 0.007315600 0.04713311 0.03981751 0.1420000 0.1200000 0.1540000
.rnd.portf.4 0.007243608 0.03989320 0.03264960 0.1840000 0.2160000 0.3800000
.rnd.portf.5 0.006931571 0.04593206 0.03900049 0.1580000 0.1220000 0.0580000
.rnd.portf.6 0.007262338 0.06624113 0.05897879 0.2800000 0.0560000 0.2020000
                   w.EM    w.EQMN      w.ED
.rnd.portf.1 0.1666667 0.1666667 0.1666667
.rnd.portf.2 0.1660000 0.1540000 0.1860000
.rnd.portf.3 0.0500000 0.0780000 0.4620000
.rnd.portf.4 0.0500000 0.0700000 0.1060000
.rnd.portf.5 0.2560000 0.3520000 0.0580000
.rnd.portf.6 0.2540000 0.1500000 0.0660000
```

Chart the optimal weights and optimal portfolio in risk-return space. Because the optimization was run with `trace=TRUE`, the chart of the optimal portfolio also includes the trace portfolios of the optimization. This is usefule to visualize the feasible space of the portfolios. The 'neighbor' portfolios relative to the optimal portfolio weights can be included the chart of the optimal weights.

```
> plot(opt_meanETL, risk.col="ETL", return.col="mean",
+      main="mean-ETL Optimization", neighbors=25)
```

Calculate and plot the portfolio component ETL contribution.

```
> pct_contrib <- ES(R=R, p=0.95, portfolio_method="component",
+                   weights=extractWeights(opt_meanETL))
> barplot(pct_contrib$pct_contrib_MES, cex.names=0.8, las=3, col="lightblue")
```

This figure shows that the Equity Market Nuetral strategy has greater than 50% risk contribution. A risk budget objective can be added to limit risk contribution percentage to 40%.

6.7 Maximize mean return per unit ETL with ETL risk budgets

Add objectives to maximize mean return per unit ETL with 40% limit ETL risk budgets.

```
> # change the box constraints to long only
> init$constraints[[2]]$min <- rep(0, 6)
> init$constraints[[2]]$max <- rep(1, 6)
> rb_meanETL <- add.objective(portfolio=init, type="return", name="mean")
> rb_meanETL <- add.objective(portfolio=rb_meanETL, type="risk", name="ETL",
+                             arguments=list(p=0.95))
> rb_meanETL <- add.objective(portfolio=rb_meanETL, type="risk_budget",
+                             name="ETL", max_prisk=0.4, arguments=list(p=0.95))
```

Run the optimization. Set `traceDE=5` so that every fifth iteration is printed. The default is to print every iteration.

```
> opt_rb_meanETL <- optimize.portfolio(R=R, portfolio=rb_meanETL,
+                                      optimize_method="DEoptim",
+                                      search_size=2000,
+                                      trace=TRUE, traceDE=5)

Iteration: 5 bestvalit: 0.014832 bestmemit:    0.000000    0.316000    0.186000    0.044000    0.2
Iteration: 10 bestvalit: 0.014832 bestmemit:    0.000000    0.316000    0.186000    0.044000    0.
```

```
Iteration: 15 bestvalit: 0.012812 bestmemit:    0.038000    0.331371    0.118000    0.048000    0.
Iteration: 20 bestvalit: 0.012812 bestmemit:    0.038000    0.331371    0.118000    0.048000    0.
[1] 0.03800000 0.33137121 0.11800000 0.04800000 0.41600000 0.04279574

> print(opt_rb_meanETL)

**********************************
PortfolioAnalytics Optimization
**********************************

Call:
optimize.portfolio(R = R, portfolio = rb_meanETL, optimize_method = "DEoptim",
    search_size = 2000, trace = TRUE, traceDE = 5)

Optimal Weights:
    CA   CTAG     DS     EM   EQMN     ED
0.0380 0.3314 0.1180 0.0480 0.4160 0.0428

Objective Measures:
    mean
0.006552



    ETL
0.01936

contribution :
      CA     CTAG       DS       EM     EQMN       ED
0.001305 0.004959 0.002620 0.001920 0.007555 0.001005

pct_contrib_MES :
     CA    CTAG      DS      EM    EQMN      ED
0.06739 0.25610 0.13529 0.09914 0.39016 0.05192

> plot(opt_rb_meanETL, risk.col="ETL", return.col="mean",
+      main="Risk Budget mean-ETL Optimization",
+      xlim=c(0,0.12), ylim=c(0.005,0.009))
```

Chart the contribution to risk in percentage terms.

```
> plot.new()
> chart.RiskBudget(opt_rb_meanETL, risk.type="percentage", neighbors=25)
```

6.8 Maximize mean return per unit ETL with ETL equal contribution to risk

Add objective to maximize mean return per unit ETL with ETL equal contribution to risk.

```
> eq_meanETL <- add.objective(portfolio=init, type="return", name="mean")
> eq_meanETL <- add.objective(portfolio=eq_meanETL, type="risk", name="ETL",
+                             arguments=list(p=0.95))
> eq_meanETL <- add.objective(portfolio=eq_meanETL, type="risk_budget",
+                             name="ETL", min_concentration=TRUE,
+                             arguments=list(p=0.95))
```

Run the optimization. Set `traceDE=5` so that every fifth iteration is printed. The default is to print every iteration.

```
> opt_eq_meanETL <- optimize.portfolio(R=R, portfolio=eq_meanETL,
+                                      optimize_method="DEoptim",
+                                      search_size=2000,
+                                      trace=TRUE, traceDE=5)

Iteration: 5 bestvalit: 4.191443 bestmemit:    0.098000    0.386000    0.050000    0.070000    0.2
Iteration: 10 bestvalit: 4.191443 bestmemit:    0.098000    0.386000    0.050000    0.070000    0.
[1] 0.098 0.386 0.050 0.070 0.276 0.120

> print(opt_eq_meanETL)

**********************************
PortfolioAnalytics Optimization
**********************************

Call:
optimize.portfolio(R = R, portfolio = eq_meanETL, optimize_method = "DEoptim",
    search_size = 2000, trace = TRUE, traceDE = 5)
```

```
Optimal Weights:
   CA  CTAG    DS    EM  EQMN    ED
0.098 0.386 0.050 0.070 0.276 0.120

Objective Measures:
    mean
0.006679



   ETL
0.0217

contribution :
      CA     CTAG       DS       EM     EQMN       ED
0.003829 0.007165 0.001095 0.003120 0.003520 0.002971

pct_contrib_MES :
     CA    CTAG      DS      EM    EQMN      ED
0.17644 0.33018 0.05046 0.14378 0.16223 0.13690
```

Chart the optimal weights and optimal portfolio in risk-return space.

```
> plot.new()
> plot(opt_eq_meanETL, risk.col="ETL", return.col="mean",
+      main="Risk Budget mean-ETL Optimization",
+      xlim=c(0,0.12), ylim=c(0.005,0.009))
```

Chart the contribution to risk in percentage terms. It is clear in this chart that the optimization results in a near equal risk contribution portfolio.

```
> plot.new()
> chart.RiskBudget(opt_eq_meanETL, risk.type="percentage", neighbors=25)
```

The `opt_meanETL`, `opt_rb_meanETL`, and `opt_eq_meanETL` optimizations are similar and can be easily compared.

opt_meanETL Objective to maximize mean return per unit ETL. The constraints are full investment and box constraints such that the minimum weight of any asset is 0.05 and maximum weight of any asset is 0.65.

opt_rb_meanETL Objective to maximize mean return per unit ETL with risk budget objective to limit maximum percent risk 40%. The constraints are full investment and long only constraints.

opt_eq_meanETL Objective to maximize mean return per unit ETL with equal contribution to risk. The constraints are full investment and long only constraints.

Combine the optimizations for easy comparison.

```
> opt_combine <- combine.optimizations(list(meanETL=opt_meanETL,
+                                           rbmeanETL=opt_rb_meanETL,
+                                           eqmeanETL=opt_eq_meanETL))
> # View the weights and objective measures of each optimization
> extractWeights(opt_combine)

              CA      CTAG    DS    EM  EQMN         ED
meanETL   0.062 0.3660000 0.100 0.070 0.344 0.05200000
rbmeanETL 0.038 0.3313712 0.118 0.048 0.416 0.04279574
eqmeanETL 0.098 0.3860000 0.050 0.070 0.276 0.12000000

> obj_combine <- extractObjectiveMeasures(opt_combine)

> chart.Weights(opt_combine, plot.type="bar", legend.loc="topleft", ylim=c(0, 1))
```

Chart the optimal portfolios of each optimization in risk-return space.

```
> plot.new()
> chart.RiskReward(opt_combine, risk.col="ETL", return.col="mean",
+                  main="ETL Optimization Comparison", xlim=c(0.018, 0.024),
+                  ylim=c(0.005, 0.008))
```

Calculate the STARR of each optimization

```
> STARR <- obj_combine[, "mean"] / obj_combine[, "ETL"]
> barplot(STARR, col="blue", cex.names=0.8, cex.axis=0.8,
+         las=3, main="STARR", ylim=c(0,1))

> plot.new()
> chart.RiskBudget(opt_combine, match.col="ETL", risk.type="percent",
+                  ylim=c(0,1), legend.loc="topright")
```