Discussion of Upcoming and Desired Design and Coding Decisions in PortfolioAnalytics (2012)

Brian G. Peterson
brian@braverock.com

April 18, 2015

Abstract

PortfolioAnalytics has grown significantly since it was initially written, and will continue to grow and change, especially with the increased involvement of additional portfolio managers and with work on the next version of Doug Martin et. al.'s *Modern Portfolio Optimization*(Scherer and Martin, 2005)

This document lays out Some current state information, some design musings, and some things that we desire to address.

It is my hope that interested readers and users will give us feedback, and possibly even code contributions, to make some of this functionality a reality.

Contents

1 Introduction

Difference between constraints and objectives

Renaming constraints object, separating constraints and objectives, collections of objectives multi-objective optimization

Specifying constraints

Relaxing constraints penalized versus direct

2 Current State

Our overall goal with Portfolioanalytics is to allow the use of many different portfolio solvers to solve the same portfolio problem specification.

2.1 Constraints Current State

On the constraints front, this includes support for box constraints, inequality constraints, turnover, and full investment.

Box Constraints box constraints (min/max) are supported for all optimization engines, this is a basic feature of any optimization solver in R. It is worth noting that *most* optimization solvers in R support *only* box constraints.

Box constraints are of course necessary, but not sufficient, and a portfolio that satisfies the box constraints may very well not satisfy any number of other constraints on the combinations of weights.

Full Investment Constraint a full investment constraint or min_sum, max_sum constraint is supported for all solvers inside the constrained_objective function using either a normalization method or a penalty method The normalization method simply scales the supplied parameters to either the minimum or the maximum if the sum is outside the allowed bounds. This has the advantage of being fast, but the disadvantages of decreasing the number of portfolios that will be tested that have assets that use your max/min box constraints, because of the scaling effects. You'll get close, but very few tested portfolios, perhaps none, will have assets that exactly hit the max or min box constrained weights.

The penalty method simply uses the absolute value of the difference for the sum of the weight vector from your min_sum or max_sum (depending on which was violated) as part of the penalized objective output that the solver is working on. This has the advantage that you are not doing transformations on the supplied weight vector inside the constrained_objective function, but the disadvantage that it may take the solver a long time to find a population that meets the objectives.

For the ROI solvers, min_sum and max_sum are supported as linear constraints on the sum of the weights vector.

Linear Inequality Constraints individual linear inequality constraints are currently supported only for the ROI solvers, because ROI supports these types of constraints for at least Rglpk and quadprog solvers, and apparently also several other solvers that directly support these types of constraints, such as ipop. These constraints can indicate that specific assets must be larger or smaller than other assets in the portfolio.

Group Inequality Constraints linear group inequality constraints are currently supported only for the ROI solvers. These constraints can separate the portfolio assets into groups, and apply basic inequalities ">","<",">=", "<=", etc to them so that the sum of the weights in the group must satisfy the constraint versus another group or groups.

Turnover turnover is currently supported as an objective that can be added to your overall portfolio objectives, and is handled inside constrained objective. I think that it could also be handled as a true constraint, which I'll discuss in a bit.

3 Improving the Current State

3.1 Modularize Constraints

Today, the box constraints are included in the `constraints` constructor. It would be better to have a portfolio specification object that included multiple sections: constraints, objectives, assets. (this is how the object is organized, but that's not obvious to the user). I think that we should have an `add.constraint` function like `add.objective` that would add specific types of constraints:

1. box constraints

2. asset inequality constraints

3. group inequality constraints

4. turnover constraint

5. full investment or leverage constratint

3.2 Creating a mapping function

`DEoptim` contains the ability to use a *mapping function* to manage constraints, but there are no generalized mapping functions available. The mapping function takes in a vector of optimization parameters(weights), tests it for feasibility, and either transforms the vector to a vector of feasible weights or provides a penalty term to add to the objective value. For PortfolioAnalytics, I think that we can write a constraint mapping function that could do the trick, even with general optimization solvers that use only box constraints. The mapping function should have the following features:

methods: the methods should be able to be turned on and off, and applied in different orders. For some constraint mapping, it will be important to do things in a particular order. Also, for solvers that support certain types of constraints directly, it will be important to use the solver's features, and not use the corresponding mapping functionality.

layering: `ROI` constains function `rbind.L_constraint`, which combines the various linear inequality constraints into a single model. We should examine this and see if we can make use of it, or something like it, to create a consolidated inequality constraint mapping capability

relocatable: for some solvers such as `DEoptim`, the solver can use the mapping function to only evaluate the objective for portfolios that meet the constraints. For other solvers, where only box constraints are supported, we will need to either penalize or transform (see discussion above in 2) weights vector later in the process, inside the `constrained_objective` function.

relax constraints: we need the ability to relax infeasible constraints, either via the penalty method (just find the closest) or when transforming the weights vector to a feasible set. see also 3.3 for a discussion of adaptive penalties.

The mapping function could easily be incorporated directly into random portfolios, and some code might even move from random portfolios into the mapping function. DEoptim can call the

mapping function directly when generating a population. For other solvers, we'll need to read the constraint specification, determine what types of constraints need to be applied, and utilize any solver-specific functionality to support as many of them as possible. For any remaining constraints that the solver cannot apply directly, we can call the mapping function to either penalize or transfrom the weights on the remaining methods inside `constrained_objective`.

3.3 Penalty Functions

The current state uses a user-defined fixed penalty function for each objective, multiplying the exceedence or difference of the objective from a target against a fixed penalty. This requires the user to understand how penalty terms influence optimization, and to modify their terms as required.

The multiple papers since 2003 or so suggest that an adaptive penalty that changes with the number of iterations and the variability of the answers for the objective can significantly improve convergence. [add references]. As we re-do the constraints, we should consider applying the penalty there for small feasible constraint areas (e.g. after trying several hundred times to get a random draw that fits, use adaptively penalized contraints to get as close as possible), but even more importantly perhaps we should support an adaptive constraint in the objectives.

Several different methods have been proposed. In the case of constraints, penalties should probably be relaxed as more infeasible solutions are found, as the feasible space is likely to be small. In the sase of objectives, arguably the opposte is true, where penalties should increase as the number of iterations increases, to speed convergence to a single solution, hopefully at or near the global minima.

4 Bibliography

References

Boudt, K., P. Carl, and B. G. Peterson (2012). PortfolioAnalytics: Portfolio analysis, including numeric methods for optimization of portfolios. R package version 0.8.2.

Scherer, B. and D. Martin (2005). *Modern Portfolio Optimization*. Springer.