R/Rmetrics Solver Factory for Portfolio Optimization and Design

Diethelm Würtz and Yohan Chalabi

Institute for Theoretical Physics ETH Zurich
Curriculum for Computational Science ETH Zurich

May 25, 2011

Mathematical Programming Formulation

Portfolio Optimization Problems can be formulated in different forms with several types of constraints:

Minimum Risk Objective

Minimize Any Risk + Transaction Costs
subject to: Return $\geq$ a given level
Any other user defined constraints

Maximum Return Objective

Maximize Return – Transaction Costs
subject to: Any Risk Measure $\leq$ a given level.
Any other user defined constraints

Maximum Risk-Adjusted Return

Maximize Utility = Return – λ*Risk – TransCosts
where λ is a risk aversion
subject to: Any user defined constraints

Constraints

- Simple Bounds on individual Assets
- Grouping Assets by Linear Constraints
- Risk Budget Constraints are usually Quadratic Constraints
- Nonlinear Constraints as Drawdowns, Reward/Risk Ratios,
- Turnover Constaints, . . .
- Mixed Integer Constraints adding Buy-in and Cardinality Constraints, …

Examples

Some Typical Portfolios

CVaR Rockafeller Uryasev
LP Problem

$$\max_{w_i, e_s, VaR} \; VaR - \left(\frac{1}{m} \sum_{s=1}^{m} e_s \right) / \alpha$$

$$\sum_{i=1}^{n} w_i \mu_i \geq \bar{\mu}$$

$$e_s \geq VaR - \sum_{i=1} w_i r_{i,s}$$

$$e_s \geq 0$$

$$e_s = \max \left[0, VaR - \sum_{i=1}^{n} w_i r_{i,s} \right]$$

$$CVaR = VaR - \left(\frac{1}{m} \sum_{s=1}^{m} e_s \right) / \alpha$$

Mean-Variance Markowitz
QP Problem

$$\min_{w} \; w^T \hat{\Sigma} w$$

$$s.t.$$

$$w^T \hat{\mu} = \bar{r}$$

$$w^T 1 = 1$$

$$w > 0$$

Tail Risk Diversification
NLP Problem

$$\min \; w^\top \hat{\Sigma} w$$

$$s.\ t.$$

$$w^\top \hat{\mu} = \bar{r}$$

$$w^\top 1 = 1$$

$$\mathcal{L}_i^{lower} \leq \frac{w_i}{\lambda} \frac{d\lambda}{dw_i} \leq \mathcal{L}_i^{upper}$$

$$\ldots$$

$$\lambda_{lower} = \lim_{u \to 0} \left[\Pr \left(Y \leq F_Y^{-1}(u) \mid X \leq F_X^{-1}(u) \right) \right]$$

$$= \lim_{u \to 0} \left[\frac{C(u,u)}{u} \right]$$

$$l_i \delta_i \leq w_i \leq u_i \delta_i$$ Buy-in Constraints

$$\sum \delta_i \leq k$$ Cardinality Constraints

R Mathematical Programming

Solver Overview

General Optimization Overview:

R Task View: Optimization

Major R Contributed Solvers

NLP Rsolnp on CRAN, Rdonlp2 on R-forge
QP quadprog, kernlab [loqo] on CRAN
SOCP Rsocp on R-forge
[MI]LP Rglpk, Rsymphony on CRAN

Simple R/AMPL Interface / Coin-OR / NEOS

RmetricsPortfolio (www.rmetrics.org)
Common Function Wrappers: fPortfolio (CRAN)

Infrastructures:

ROI Infrastructure on R-forge – Stefan Theussl et al.
R-AMPL Infrastructure (unpublished) - Roland Hocheiter
R-NEOS API on CRAN – Bernhard Pfaff

R Mathematical Programming

What we need in Portfolio Optimization

Wee need LP, QP, NLP Solver, in combination with MI Programming to solve the following optimization problem:

$$\min_x \; f(x)$$

$$s.t.$$

$$b_{li} \le x_i \le b_{ui}$$

$$A_{eq}x = a_e$$

$$a_l \le A_{in}x \le a_u$$

$$h_i(x) = h_e$$

$$g_l \le g_i(x) \le g_u$$

$$l_i\delta_i \le w_i \le u_i\delta_i$$

$$\sum \delta_i \le k$$

R Contributed Solvers

NLP Nonlinear Programming

Three contributed R packages offer nonlinear mathematical programming solvers, Rmetrics has added a NLP convenience wrapper.

- `donlp2()` - a NLP solver from the `Rdonlp2` package
- `solnp()` - a NLP solver from the `Rsolnp` package
- `nlminb2()` - a NLP solver from the `Rnlminb` package

- `donlp2NLP()` - a donlp2 NLP wrapper function
- `solnpNLP()` - a solnp NLP wrapper function
- `nlminb2NLP()` - a nlminb2 NLP wrapper function

donlp2 solves:

$$\min_x \ f(x)$$
$$s.t.$$
$$a_{li} \le x_i \le a_{ui}$$
$$b_l \le Ax \le b_u$$
$$c_{li} \le c_i(x) \le c_{ui}$$

solnp solves:

$$\min_x \ f(x)$$
$$s.t.$$
$$b_{li} \le x_i \le b_{ui}$$
$$g_i(x) = 0$$
$$h_{li} \le h_i(x) \le h_{ui}$$

nlminb2 solves:

$$\min_x \ f(x)$$
$$s.t.$$
$$h_i(x) = 0, \quad i = 1...p$$
$$g_i(x) \le 0, \quad i = 1...m$$

Covenience Wrapper:

$$\min_x \ f(x)$$
$$s.t.$$
$$b_{li} \le x_i \le b_{ui}$$
$$A_{eq}x = a_e$$
$$a_l \le A_{in}x \le a_u$$
$$h_i(x) = h_e$$
$$g_l \le g_i(x) \le g_u$$

R Contributed Solvers

QP Programming

Two contributed R packages offer quadratic mathematical programming solvers, Rmetrics has added a QP convenience wrapper

- `solve.QP()` - solver from the `quadprog` package
- `ipop()` - solver from the `kernlab` package
- `quadprogQP()` - solve.QP QP wrapper function
- `ipopQP()` - ipop QP wrapper function

solve.QP solves:

$$\min_x \; -d'x + \frac{1}{2}x'Dx$$
$$s.t.$$
$$A \bowtie b_0$$

ipop solves:

$$\min_x \; c'x + \frac{1}{2}x'Hx$$
$$s.t.$$
$$l \le x \le u$$
$$b \le Ax \le b + r$$

Convenience Wrapper:

$$\min_x \; d'x + \frac{1}{2}x'Dx$$
$$s.t.$$
$$b_l \le x_i \le b_u$$
$$A_{eq}x = a_e$$
$$a_l \le A_{in}x \le a_u$$

R Contributed Solver

LP Programming

Two contributed R packages offer linear mathematical programming solvers, no convenience wrapper is required.

- `Rglpk_solve_LP()` - LP solver from the `Rglpk` package
- `Rsymphony_solve_LP()` - LP solver from the `Rsymphony` package

Rglpk_solve_LP and Rsymphony_solve_LP solve:

$$\min_x \sum_i v_i x_i$$

$$s.t.$$

$$Ax \bowtie b$$

$$l_i \delta_i \leq w_i \leq u_i \delta_i$$

$$\sum \delta_i \leq k$$

R / AMPL Interface

Solver Interface and AMPL Workflow

AMPL is an algebraic modeling language for solving linear and nonlinear optimization problems in discrete and/or continuous variables.

How to get AMPL:

Get 60-day trial or student edition from www.ampl.com

Supported Solvers:

Open Source: Coin OR hosts more than 30 projects, including LP, QP, NLP, SOCP, and Mixed Integer Programming MIPS. www.coin-or.org/projects/

Commercial Solvers: Include cplex, donlp2, knitro, loqo, minos, snopt, amongst others.

R / AMPL Workflow:

1. Write an AMPL model file
2. Write an AMPL data file
3. Write an AMPL run file

Execute the AMPL run file

Read the results from the output file

R / AMPL Interface

Model, Data & Run Files

Rmetrics provides constuctor functions to generate the model, the data, and the run files from inside R in a highly automated way. (Next)Package fPortfolio

ampl.mod

```
param nAssets;
param mu{1..nAssets};
param Cov{1..nAssets, 1..nAssets};
param targetReturn;
var x{1..nAssets} >= 0;
minimize Risk: sum {i in 1..nAssets} sum{j in 1..nAssets} x[i]*Cov[i,j]*x[j];
subject to Return: sum{i in 1..nAssets} mu[i]*x[i] = targetReturn;
subject to fullInvestment: sum{i in 1..nAssets} x[i] = 1;
```

ampl.dat

```
param nAssets := 6 ;
param targetReturn := 0.043077 ;
param mu := 1 4.0663e-05 2 0.084175 3 0.023894 4 0.0055315 5 0.059052 6 0.085768 ;
param Cov :  1 2 3 4 5 6 :=
1   0.015900  -0.012741  0.0017994  0.0098039  -0.015888  -0.013238
2  -0.012741   0.58461    0.030336   -0.014075   0.4116  0.29840
3   0.0017994  0.030336 0.085138  0.00092505  0.024816  0.015549
4   0.0098039  -0.014075  0.00092505  0.014951  -0.023322  -0.017247
5  -0.015888  0.4116  0.024816  -0.023322  0.53503  0.36481
6  -0.013238  0.29840  0.015549 -0.017247 0.36481 0.32312 ;
```

ampl.run

```
option solver cplex;
model ampl.mod;
data ampl.dat;
solve;
display x > ampl.txt;
exit;
```

Coin OR Solvers

Using Coin-OR Solvers

"The Computational Infrastructure for Operations Research, COIN-OR, is an initiative to spur the development of open-source software for the operations research community."
www.coin-or.org

Binary files from the COIN-OR project can be obtained
projects.coin-or.org/CoinBinary

Projects include in Coin All:

Solver:	Description:
bonmin	NL Mixed Integer Programming
cbc	LP branch-and-cut solver
clp	LP simplex solver
couenne	branch-and-bound NL MI Programming
ipopt	IP general large-scale NL optimization
symphony	Linear Mixed Integer Programming

R / AMPL Interface

Using Coin-OR Solvers

To use the Coin-OR solvers we have just to modify the name to the desired solver.

ampl.mod

```
param nAssets;
param mu{1..nAssets};
param Cov{1..nAssets, 1..nAssets};
param targetReturn;
var x{1..nAssets} >= 0;
minimize Risk: sum {i in 1..nAssets} sum{j in 1..nAssets} x[i]*Cov[i,j]*x[j];
subject to Return: sum{i in 1..nAssets} mu[i]*x[i] = targetReturn;
subject to fullInvestment: sum{i in 1..nAssets} x[i] = 1;
```

ampl.dat

```
param nAssets := 6 ;
param targetReturn := 0.043077 ;
param mu := 1 4.0663e-05 2 0.084175 3 0.023894 4 0.0055315 5 0.059052 6 0.085768 ;
param Cov :  1 2 3 4 5 6 :=
1   0.015900  -0.012741  0.0017994  0.0098039  -0.015888  -0.013238
2  -0.012741   0.58461    0.030336   -0.014075   0.4116  0.29840
3   0.0017994  0.030336 0.085138  0.00092505  0.024816  0.015549
4   0.0098039  -0.014075  0.00092505  0.014951  -0.023322  -0.017247
5  -0.015888  0.4116  0.024816  -0.023322  0.53503  0.36481
6  -0.013238  0.29840  0.015549 -0.017247 0.36481 0.32312 ;
```

ampl.run

```
option solver ipopt;
model ampl.mod;
data ampl.dat;
solve;
display x > ampl.txt;
exit;
```

Kestrel / AMPL Interface

NEOS Server

"The Kestrel client/server is one way of sending your optimization job to be solved via the NEOS Server from within your modeling environment and receiving results that can be interpreted directly by your modeler".

http://www-neos.mcs.anl.gov/neos/kestrel.html

Binary files for the Kestrel Client can be obtained

http://www-neos.mcs.anl.gov/neos/Installation.html

R / Kestrel / AMPL Interface

Using Kestrel-NEOS Interface

To use the Kestrel-NEOS Interface we have just to modify again the name to the desired solver. [Note, the NEOS Server is heavily overloaded]

ampl.mod

```
param nAssets;
param mu{1..nAssets};
param Cov{1..nAssets, 1..nAssets};
param targetReturn;
var x{1..nAssets} >= 0;
minimize Risk: sum {i in 1..nAssets} sum{j in 1..nAssets} x[i]*Cov[i,j]*x[j];
subject to Return: sum{i in 1..nAssets} mu[i]*x[i] = targetReturn;
subject to fullInvestment: sum{i in 1..nAssets} x[i] = 1;
```

ampl.dat

```
param nAssets := 6 ;
param targetReturn := 0.043077 ;
param mu := 1 4.0663e-05 2 0.084175 3 0.023894 4 0.0055315 5 0.059052 6 0.085768 ;
param Cov :  1 2 3 4 5 6 :=
1   0.015900  -0.012741  0.0017994  0.0098039  -0.015888  -0.013238
2  -0.012741   0.58461    0.030336   -0.014075   0.4116  0.29840
3   0.0017994  0.030336 0.085138  0.00092505  0.024816  0.015549
4   0.0098039  -0.014075  0.00092505  0.014951  -0.023322  -0.017247
5  -0.015888  0.4116  0.024816  -0.023322  0.53503  0.36481
6  -0.013238  0.29840  0.015549 -0.017247 0.36481 0.32312 ;
```

ampl.run

```
option solver kestrel;
option kestrel_options 'solver=mosek';
model ampl.mod;
data ampl.dat;
solve;
display x > ampl.txt;
exit;
```

R / neosr Interface

NEOS Server

The NEOS API provides another way of sending your job to the NEOS Server.
We can use the contributed R package neosr written by Bernhard Pfaff.

```
# AMPL Model Specifications:
model <- paste(
    "param nAssets;",
    "param mu{1..nAssets};",
    "param Cov{1..nAssets, 1..nAssets};",
    "param targetReturn;",
    "var weights{1..nAssets} >= 0;",
    "minimize Risk: sum {i in 1..nAssets} sum{j in 1..nAssets} weights[i]*Cov[i,j]*weights[j];",
    "subject to Return: sum{i in 1..nAssets} mu[i]*weights[i] = targetReturn;",
    "subject to fullInvestment: sum{i in 1..nAssets} weights[i] = 1;",
    sep = "\n")

# AMPL Data Specifications:
data <- paste(
    "param nAssets := 6 ;",
    "param targetReturn := 0.043077 ;",
    "param mu := 1 4.0663e-05 2 0.084175 3 0.023894 4 0.0055315 5 0.059052 6 0.085768 ;",
    "param Cov : 1 2 3 4 5 6 :=",
    "1  0.015900  -0.012741 0.0017994  0.0098039  -0.015888 -0.013238",
    "2 -0.012741   0.58461  0.030336  -0.014075    0.4116    0.29840",
    "3  0.0017994  0.030336 0.085138   0.00092505  0.024816  0.015549",
    "4  0.0098039 -0.014075 0.00092505 0.014951   -0.023322 -0.017247",
    "5 -0.015888   0.4116   0.024816  -0.023322    0.53503   0.36481",
    "6 -0.013238   0.29840  0.015549  -0.017247    0.36481   0.32312 ;",
    sep = "\n”)
```

R / neosr Interface, Cont.

NEOS Server

```
# AMPL Run Specifications:

run <- paste(
    "solve ;",
    "display weights ;",
    "exit;",
    sep = "\n")


# Create and Submit Job:

# mod, data and run Specifications:
amplSpec <- list(model=model, data=data, commands=run, comments="NEOS")

# Solver Specification:
solverTemplate <- NgetSolverTemplate(category="nco", solvername="MINOS", inputMethod ="AMPL")

# Create XML strings:
xmls <- CreateXmlString(neosxml=solverTemplate, cdatalist=amplSpec)

# Submit Job:
submittedJob <- NsubmitJob(xmlstring=xmls, user="rneos", interface="", id=0)

# Get Final Results:
NgetFinalResults(obj=submittedJob, convert=TRUE)
```

Putting all Together

Simple Optimization Layer– Towards a High Level Interface

Putting all Together:

```
Arguments:
                    start objective lower upper linCons funCons amplCons            control ...
                                          =0    =1                                  =list()
                           class()                 list()  list()   list()             list()

LP Solver:
glpkLP      default       .   numeric     x     x    yes       .        .       glpkLPControl    .
symphonyLP        .       .   numeric     x     x    yes       .        .   symphonyLPControl    .
amplLP            .       .   numeric     x     x    yes       .        .       amplLPControl    .

QP Solver:
quadprogQP  default       .      list     x     x    yes       .        .   quadprogQPControl    .
ipoptQP           .       .      list     x     x    yes       .        .      ipoptQPControl    .
amplQP            .       .      list     x     x    yes       .        .       amplQPControl    .

NLP Solver:
solnpNLP    default     yes  function     x     x    yes     yes        .     solnpNLPControl    .
donlp2NLP         .     yes  function     x     x    yes     yes        .    donlp2NLPControl    .
nlminb2NLP        .     yes  function     x     x    yes     yes        .   nlminb2NLPControl    .
amplNLP           .  length character     x     x      .       .      yes      amplNLPControl  yes
```

Experimental High Level Portfolio Solver Interface:

```
solveRmetrics(start, objective, lower, upper, linCons, funCons, amplCons, control)
```

Thank You

Diethelm Würtz, ETH Zürich
wuertz@phys.ethz.ch