Business Objectives and Complex Portfolio Optimization

Peter Carl
Brian Peterson
Kris Boudt

Overview

- Touch on challenges in portfolio optimization
- Introduce the *PortfolioAnalytics* package
  - Demonstrate main functions and outputs
  - Describe implementation and usage issues
- Work through some examples - some “simple,” some more complex

Portfolio Optimization Distilled

- Markowitz (1952) described an investor's objectives as:
  - maximizing some measure of gain while
  - minimizing some measure of risk.
- Many approaches follow Markowitz and use mean return and standard deviation of returns for “risk”.
- Real investors often have more complex objectives.
- R contains a multitude of methods for solving optimization problems, most are not specific to finance.

Which Optimizer Should I Use?

- Linear, quadradic or conical objectives are better addressed through a package like RMetrics' *fPortfolio*.
  - *fPortfolio* will be faster for those problems.
  - See *Portfolio Optimization with R/Rmetrics*, by Diethelm Würtz, Yohan Chalabi, William Chen, Andrew Ellis.
- Many business objectives do not fall into those categories...
  - ...and brute force solutions are often intractable
    - Unconstrained, our example has over 68 million possible solutions

Frustrations with Optimization

- Users familiar with classic optimization describe a variety of problems:
  - Too many objectives
  - Wrong weighting of objectives
  - Too many parameters (too many assets)
  - Weights float to zero
  - Hard to understand what it's doing, or when it's broken
  - Too few solutions
  - Unrealistic expectations
  - Wrong optimization method for the job
  - “Worthless results”
- Most would prefer to be approximately correct rather than precisely wrong

You want to do *what*?

- Construct a portfolio that:
  - maximizes return,
  - with per-asset conditional constraints,
  - with a specific univariate risk limit,
  - while minimizing component risk concentration,
  - and limiting drawdowns to a threshold value.

- Not a quadratic (or linear, or conical) problem any more.

About *PortfolioAnalytics*

- *PortfolioAnalytics* focuses on providing numerical solutions for portfolios with complex constraints and objective sets comprised of any R function.
- Unifies the interface into different numeric optimizers, while preserving the flexibility to define any kind of objective and constraints.
- Provides a framework for managing different sets of portfolio constraints for comparison through time
  - Min risk, Equal risk, Equal weight, Position limits...
  - Supports regular and flexible rebalancing
- Builds intuition about optimization through visualization

About *PortfolioAnalytics*

- Currently implements a front-end to two analytical solvers, Differential Evolution and Random Portfolios
- Available on R-forge in the *ReturnAnalytics* project
  - install.packages("PortfolioAnalytics", repos = "http://r-forge.r-project.org")
- Work in progress, use v >= 0.5 , rev >= 1674
- Functions are very compute intensive – even simple objectives may take a while (hours) to run on your netbook.
- Standard disclaimers apply: no warrantee, guarantees, etc.

About Random Portfolios

- At R/Finance 2009 and in multiple papers, Pat Burns describes using Random Portfolios to evaluate performance.
  - From a portfolio seed, generate random permutations that meet your constraints on the weights of each asset.
- Random Portfolio sampling can help provide insight into the goals and constraints of the optimization.
  - Aims to cover the 'edge case'(min/max) constraints almost completely, and evenly cover the 'interior' portfolios.
  - Useful for finding the search space for an optimizer.
  - Allows arbitrary number of samples.
  - Allows massively parallel execution.

About Differential Evolution

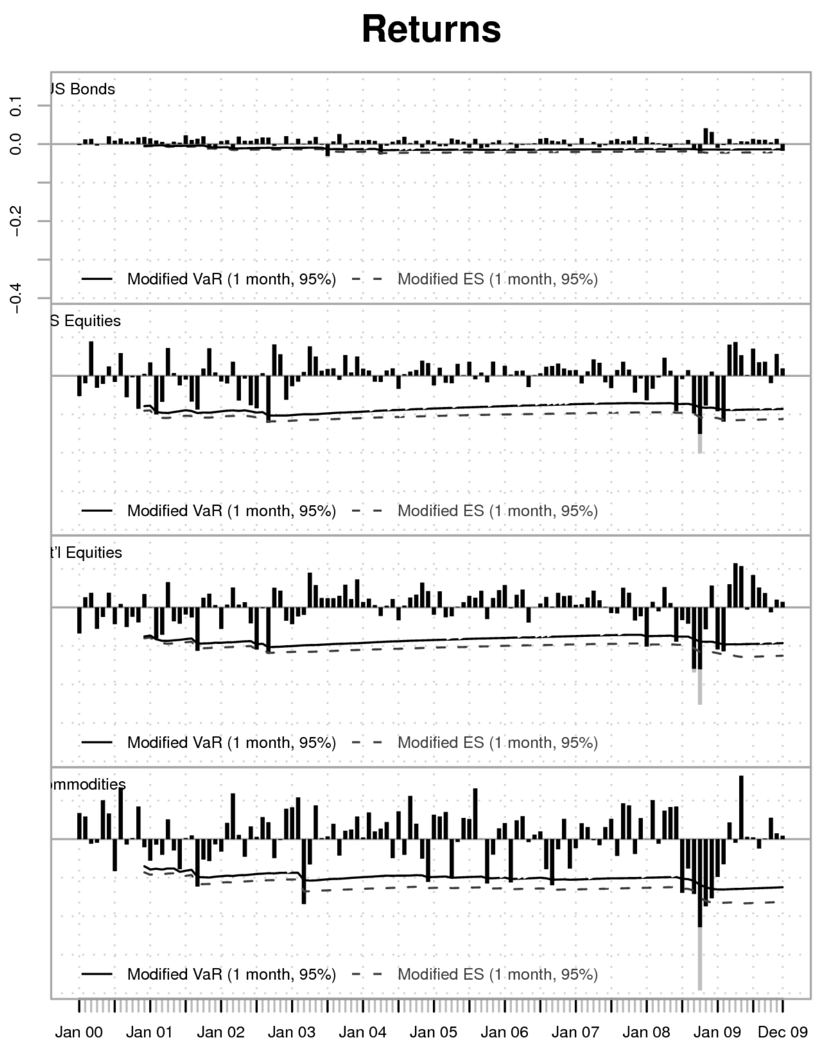
- Differential Evolution is a very powerful, elegant, population based stochastic function minimizer.
  - Continuous, evolutionary optimization.
  - Uses real-number parameters.
- Package *DEoptim* provides the algorithm in R.
  - Implementation of the algorithm distributed with the book:
    - *Differential Evolution - A Practical Approach to Global Optimization* by Price, K.V., Storn, R.M., Lampinen J.A, Springer-Verlag, 2005.
  - Thanks to R authors David Ardia and Katharine Mullen!

Load some packages, data

- Using returns for indexes representing four asset classes.
  - US Stocks, US Bonds, Int'l Stocks, Commodities.
  - 10 years of monthly total returns.

```
> library(PortfolioAnalytics)
> data(indexes)
> head(indexes)
           US Bonds US Equities Int'l Equities Commodities US Tbill Inflation
2000-01-31  -0.0026     -0.0529        -0.0677      0.0674   0.0044    0.0058
2000-02-29   0.0120     -0.0193         0.0264      0.0588   0.0046    0.0091
2000-03-31   0.0135      0.0891         0.0375     -0.0117   0.0048    0.0000
2000-04-30  -0.0038     -0.0310        -0.0553     -0.0092   0.0049    0.0011
2000-05-31  -0.0002     -0.0209        -0.0248      0.1007   0.0048    0.0057
2000-06-30   0.0199      0.0241         0.0379      0.0665   0.0049    0.0023
```

Asset Returns and Risk



- Note:
  - Huge discrepancy in risk and returns
  - Obvious co-kurtosis and outlier effects
  - Correlations increase markedly on negative shocks
- Generated with `charts.BarVaR`

Use an Equal Weight Benchmark

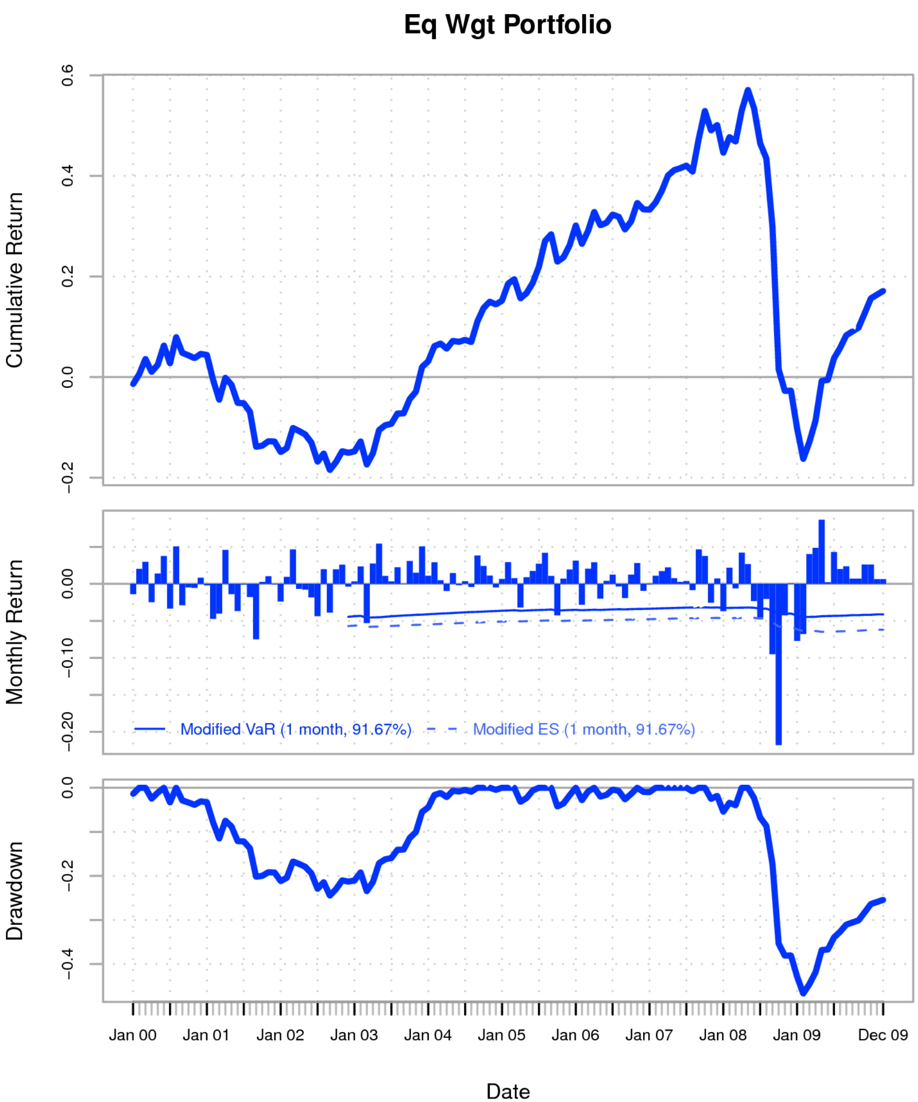
- Why an Equal Weight portfolio benchmark?
  - An equal weight portfolio provides a benchmark to evaluate the performance of an optimized portfolio against.
  - Each asset in the portfolio is purchased in the same quantity at the beginning of the period.
  - The portfolio is rebalanced back to equal weight at the beginning of each quarter.
  - Implies no information about return or risk.
- Helps answer questions a portfolio manager might ask:
  - Is the re-weighting adding or subtracting value?
  - Do we have a “useful” view of return and risk?

Calculate Equal Weight Benchmark

```
> dates=c(as.Date("1999-12-31"), time(indexes[endpoints(indexes,
  on="quarters")])
> weights = xts(matrix(rep(1/4,39*4), ncol=4), order.by=dates)
> colnames(weights)= colnames(indexes[,1:4])
> head(weights)
           US Bonds US Stocks Int'l Stocks Commodities
1999-12-31     0.25      0.25         0.25        0.25
2000-03-31     0.25      0.25         0.25        0.25
2000-06-30     0.25      0.25         0.25        0.25
2000-09-30     0.25      0.25         0.25        0.25
...
> EqWgt = Return.rebalancing(indexes[,1:4],weights)
> head(EqWgt)
           portfolio.returns
2000-01-31       -0.01395000
2000-02-29        0.02026640
2000-03-31        0.02976144
2000-04-30       -0.02482500
...
```

Monthly returns for a quarterly-rebalanced portfolio

About the Equal Weight Portfolio



- This was a difficult period for this long-only portfolio.
  - Annualized Return: 1.6%
  - Annualized Std Dev: 12.3%
  - Annualized Sharpe (R_f = 3%) : -0.1
  - Worst Drawdown: -47%

Example: Mean-CVaR Portfolio

- Although this is a “simple” case, the objectives are real:
  - Maximize the return per unit of risk taken.
  - Hold assets long-only, with positions limited by policy.
  - Remain fully allocated at all times.
  - Rebalance the portfolio quarterly.
  - Define risk as “downside risk” rather than just volatility.
  - Consider skewness and kurtosis.
- But even these “simple” portfolio objectives turn out to be complex to evaluate.
- This “base case” *could* be re-formulated in a conical solver.

Specify Constraints

- A 'constraints' object is simply a container that holds some key parameters we care about.

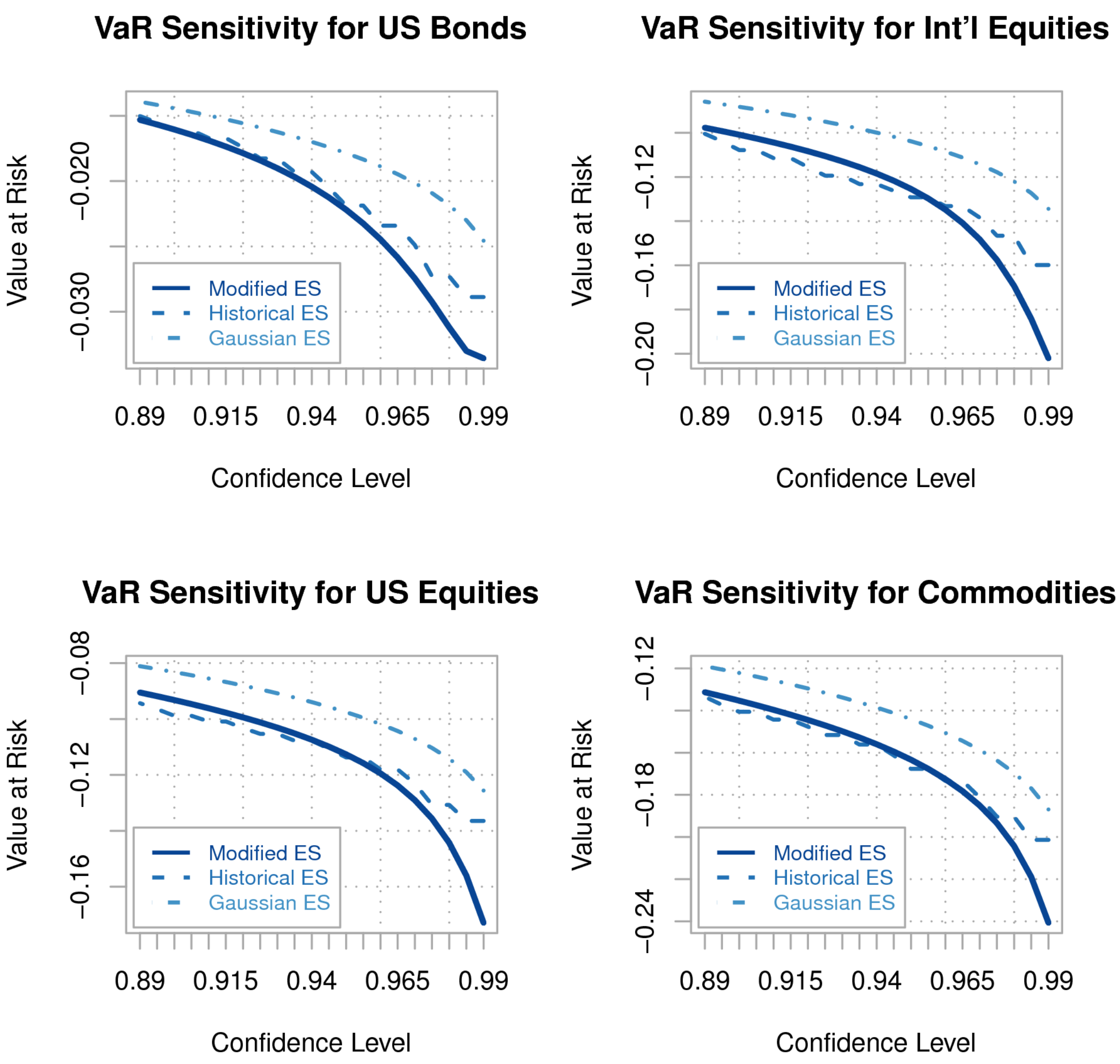
```
> aConstraintObj <- constraint(assets =
+ colnames(indexes[,1:4]),
+ min=0.05, # minimum position weight
+ max=c(0.85,0.5,0.5,0.3), # maximum position weight
+ min_sum=0.99, # minimum sum of weights approx. 1
+ max_sum=1.01, # maximum sum must also be 1 + epsilon
+ weight_seq = generatesequence()) # possible weights for
  random or brute force portfolios
```

- Constraints may be specified for each asset in the portfolio individually.
  - Here we specify “box constraints” for min.

Specify Objectives

- Any function can be specified as an objective.
  - In this case, we use modified Conditional Value-at-Risk (CVaR or ES) as a univariate measure of portfolio risk.
  - Unlike Value-at-Risk (VaR), CVaR has all the properties a risk measure should have to be coherent and is a convex function of the portfolio weights.
  - We assume our return series is skewed and/or has excess kurtosis, so we use Cornish-Fisher estimates (or “modified”) of CVaR instead.
    - Usually convex, but can break down or be non-convex at extremes.
  - See `?CVaR` in *PerformanceAnalytics*.

VaR Sensitivity



- 4-panel plot of VaR sensitivity from *PerformanceAnalytics*
- Modified CVaR (shown as ES) demonstrates a better fit for historical CVaR at lower confidence
- Breaks down at higher confidence levels

Specify Objectives

- Objectives are added to the constraint object. Here's a 'risk' objective:

```
> aConstraintObj <- add.objective(constraints =
+ aConstraintObj, # our constraint object
+ type="risk", # the kind of objective this is
+ name="CVaR", # the function to minimize
+ enabled=TRUE, # enable or disable the objective
+ arguments=list(p=(1-1/12), clean='boudt')
+ ) # parameters to pass to the CVaR function
```

- In this case, the `CVaR` function is portfolio-aware in that it takes returns and weights as arguments for evaluating each permutation.

Specify Objectives

- We need to pass the return series and the weighting vector for thousands of possible vectors, so we need to write a little wrapper to handle that:

```
> pamean <- function(n=12, R, weights, geometric=TRUE){
+ # Trailing 12 month returns, annualized
+ sum(Return.annualized(last(R,n),
+ geometric=geometric)*weights)
+ }
```

- Then we add the 'return' objective:

```
> aConstraintObj <- add.objective(constraints =
+ aConstraintObj, type="return", name="pamean",
+ enabled=TRUE, multiplier=-1,
+ arguments = list(n=12))
```

Adding Portfolio Functions

- What we just did with "`pamean`" was define a new, arbitrary function for use by the optimization.
- Our example function is an portfolio annualized mean return function, but it could be any function you've written.
- If you name the return series "*R*" and the weights vector "*weights*", the optimizer will populate these automatically.
- If your function has different arguments, you can specify them with the "*arguments*" parameter to `add.objective`.

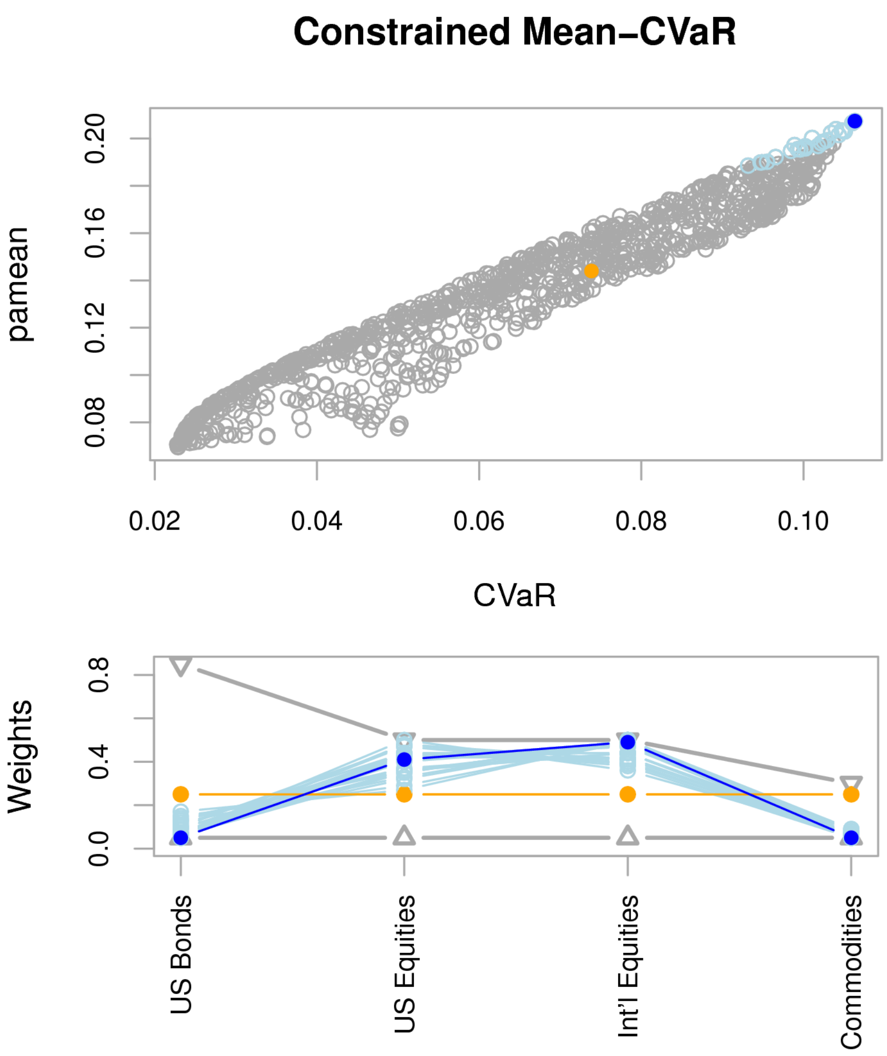
Specify Solver

- Generate sample portfolios for the most recent period:

```
> rndResult<-optimize.portfolio(R=indexes[,1:4],
+ constraints=aConstraintObj, # our constraints
+ optimize_method='random', # indicate solver to use
+ search_size=1000, # number of portfolios to generate
+ trace=TRUE, verbose=TRUE) # capture detail
```

- Our sample size should be about 1,000 portfolios.
  - For 4,050,000 possible combinations in the random portfolios with a step size of 1%, a 99% confidence and 2% error bands.
- Finds the optimal portfolio that maximizes the return per unit CVaR.

Mean-CVaR Results



- 1,000 unique, random portfolios within position constraints.
  - Orange is the equal-weight portfolio.
  - Blue is the optimal.
  - Light blue shows 25 “near optimal” portfolios
  - Weights are shown in the bottom panel.
- This is the default `plot` method for optimization

What Just Happened?

- The `optimize.portfolio` function manages the interface to the optimizer.
  - Instructs optimization backend to call `constrained_objective` for each target *w(eights)*.
- The `constrained_objective` function parses the constraints object and calls all the objective functions.
  - Applies penalty for failure to meet targets.
  - Summarizes the results in a single numerical output to be minimized.
- `optimize.portfolio.rebalancing` function manages the time loop and parallelization interface.

Mean-CVaR Through Time

- A few more parameters allow us to use the same constraint set through time:

```
> registerDoMC()
  # get out more cores,
  # this could be a different register* function

> rndResults<-optimize.portfolio.rebalancing(R=indexes[,1:4],
+ constraints=aConstraintObj, optimize_method='random',
+ search_size=1000, trace=TRUE, # all the same as prior
+ rebalance_on='quarters', # uses xts 'endpoints'
+ trailing_periods=NULL, # calculates from inception
+ training_period=36) # starts 3 years in
```

- Gives the optimal weights *each quarter* that maximize the return per unit CVaR (Minimum Risk).

Examine Results

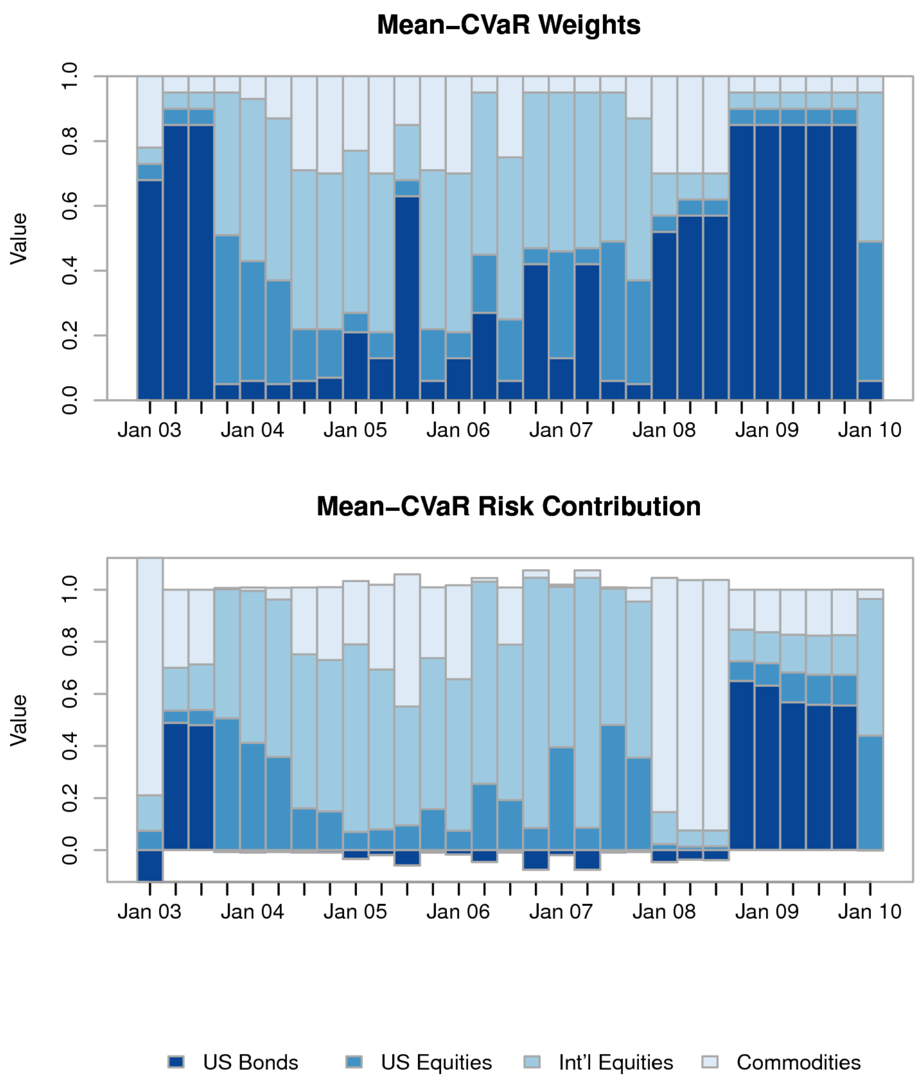
- Returns a list containing the optimal weights, some summary statistics, the function call, and optional trace information (turned off above).

```
> names(rndResults) # results organized by date
  [1] "2003-12-31" "2004-03-31" "2004-06-30" "2004-09-30"
  ...
> names(rndResults[[1]]) # look at the first slot
  [1] "weights"              "objective_measures" "call"
  [4] "constraints"          "data_summary"       "elapsed_time"
  [7] "end_t"
```

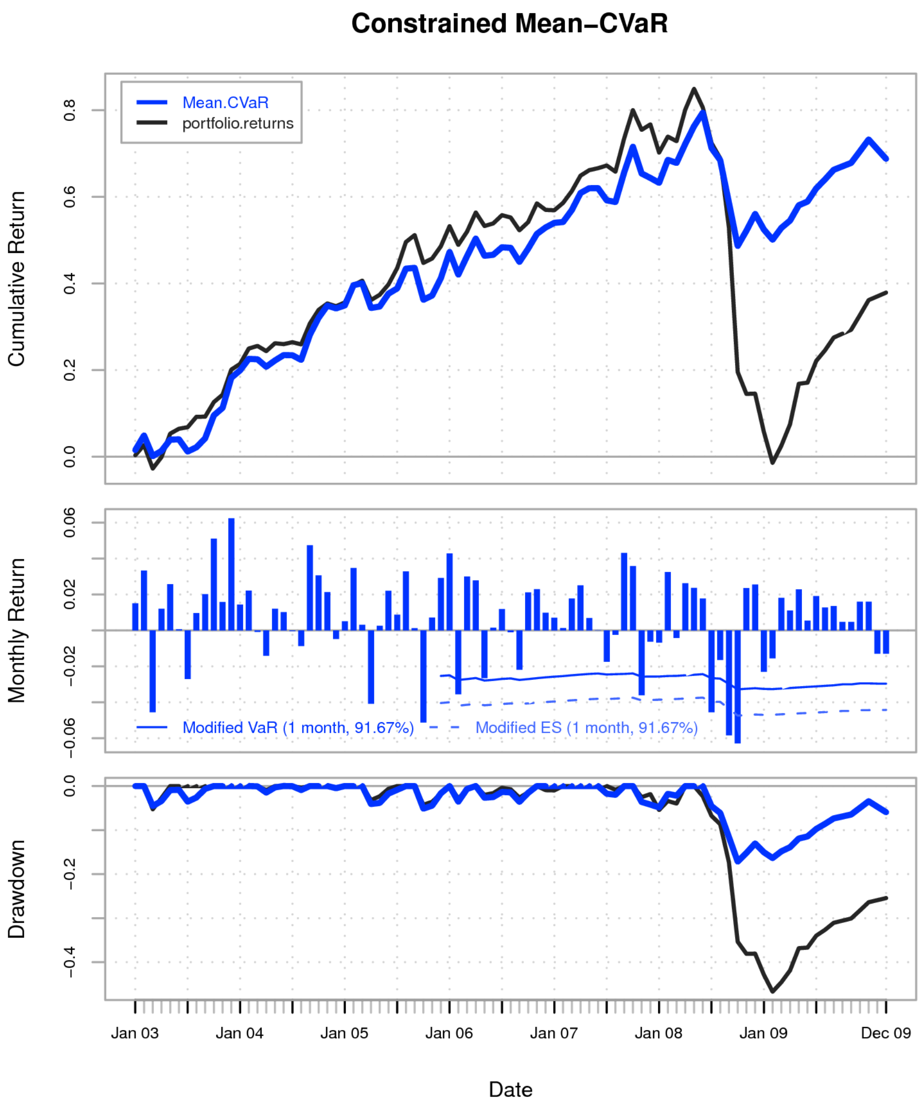
- `extractStats` function will pull out in-sample optimal portfolio statistics and weights for each rebalancing period
- Use `Return.portfolio` to calculate out of sample performance

Mean-CVaR Through Time

- Top panel shows weights through time.
- Second calculates contribution to portfolio CVaR through time.
  - Components sum to 100%
  - Diversifiers have contribution less than their portfolio weights, may have negative contributions



Mean-CVaR Through Time



- Controlling for risk improves performance...
- ... but lowers performance slightly during periods of stock out-performance

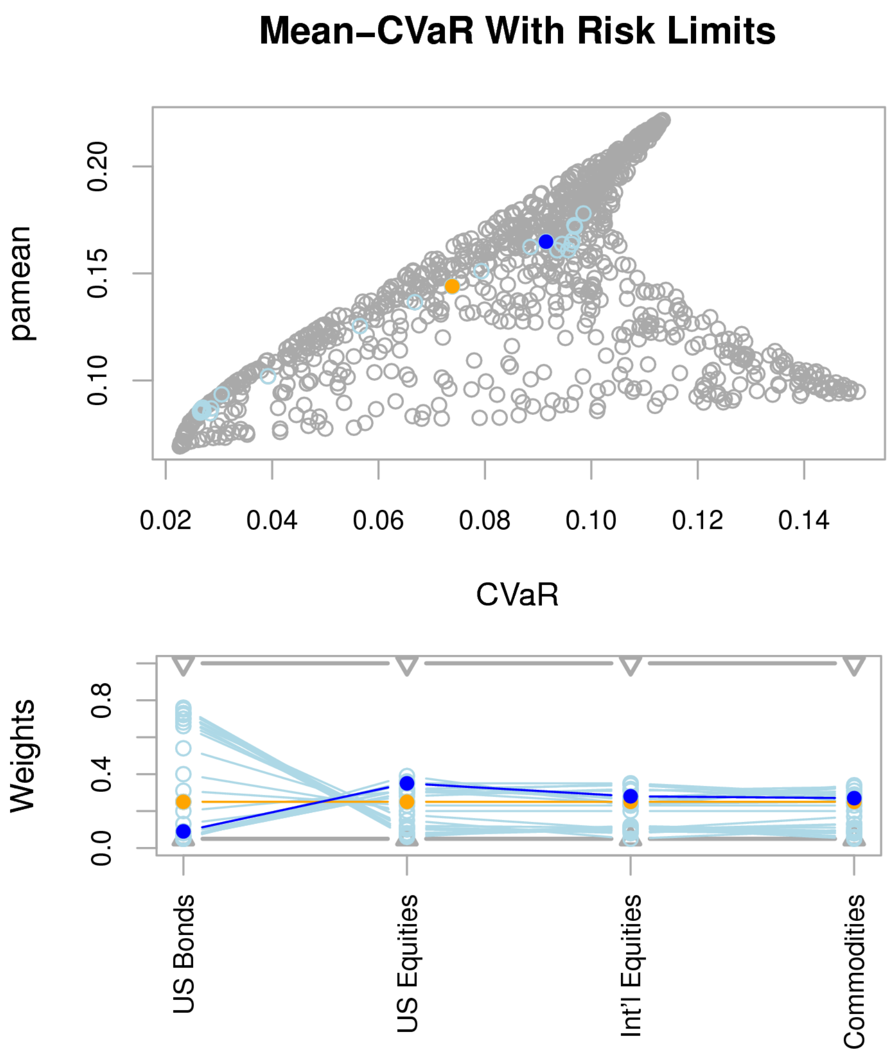
Example: Mean-CVaR w/ Risk Limit

- Add another portfolio objective:
  - No asset can contribute more than 40% to the portfolio risk.
- We remove the original position limits, then add a risk budget objective with component risk limits:

```
> aConstraintObj$max <-rep(1,4)
> names(aConstraintObj$max) <- names(aConstraintObj$min)
> aConstraintObj <- add.objective(aConstraintObj,
+ type="risk_budget", name="CVaR", enabled=TRUE,
+ min_prisk=-Inf, # no negative limit
+ max_prisk=.4, # 40% contribution limit
+ arguments = list(clean='boudt', method="modified",
> p=(1-1/12))) # arguements for CVaR function

> rndResult2<-optimize.portfolio(R=indexes[,1:4],
+ constraints=aConstraintObj, optimize_method='random',
+ search_size=1000, trace=TRUE, verbose=TRUE)
+ ) # same as previous
```

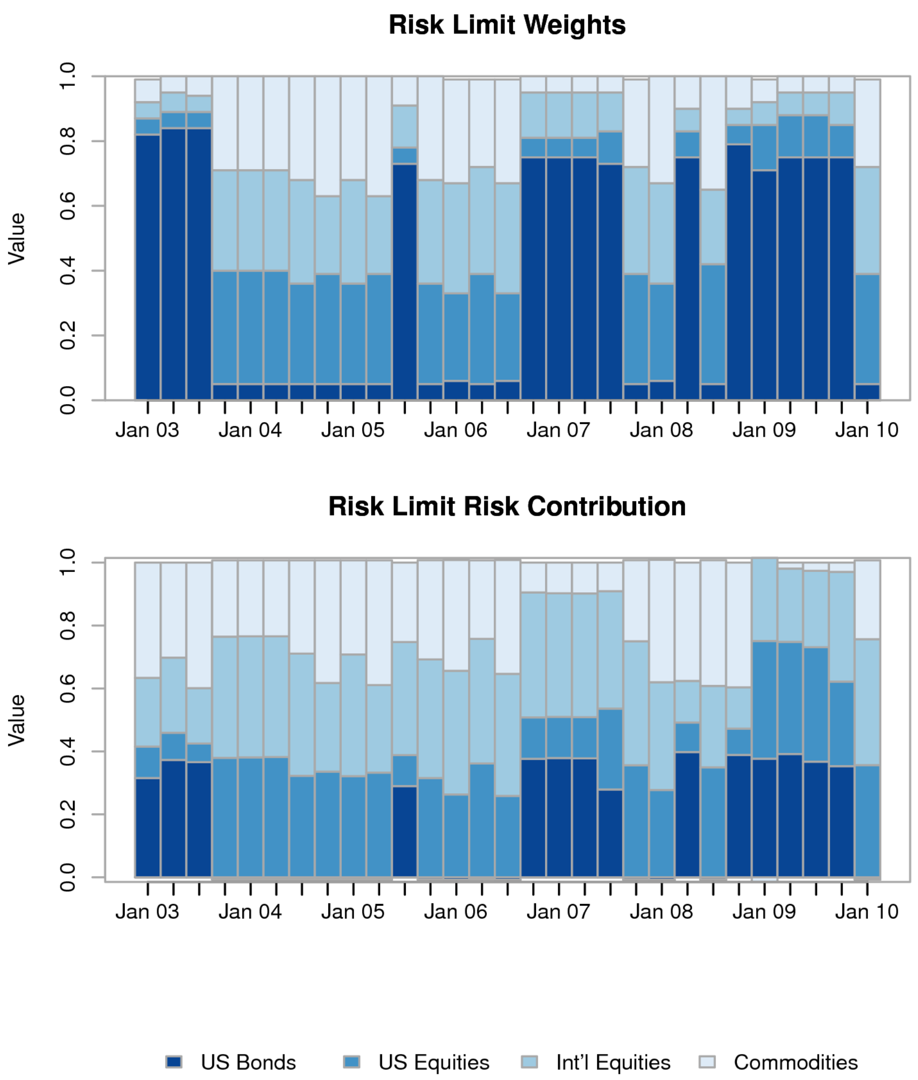
Mean-CVaR w/ Risk Limit Portfolio



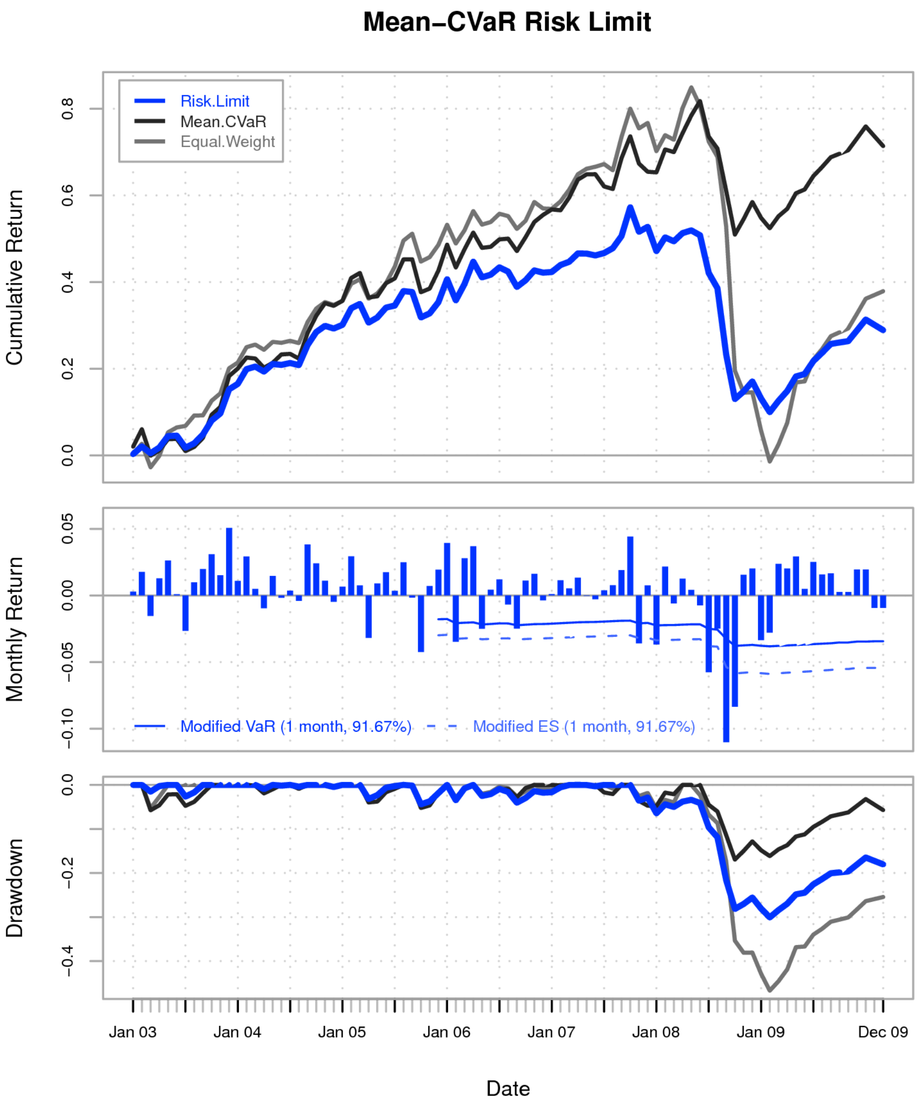
- Note:
  - Difference in shape of the feasible space
  - The optimal portfolio and its neighbors (near-optimal) are not on the outer hull
  - Weights for the bond can vary over a wide range because their contribution to risk is so low

Mean-CVaR Risk Limit Portfolio

- Bond allocations emerge when equity and commodity risk increases
- Large bond allocations still contribute little in terms of portfolio risk



Mean-CVaR Risk Limit Performance



- These risk limits appear to constrain the portfolio more than the position limits did
- Better downside performance than Equal Risk, but worse than Mean-CVaR

Example: Equal Risk Portfolio

- Actually, this is the minimum component risk contribution concentration portfolio...
  - But it's easier to say "Equal Risk."
- Why an Equal Risk portfolio?
  - Equal weight isn't necessarily balancing the portfolio risk
  - Equal risk looks to balance risk among the components of the portfolio.
  - Guard against estimation error on individual instruments (especially important on large real portfolios).
  - More likely to be "close" out of sample than traditional max/min objectives.

Specify the Equal Risk Constraints

- Build the constraint object:

```
> EqRiskConstr <- constraint(assets =
+ colnames(indexes[,1:4]), min = 0.05,
+ max = c(0.85,0.5,0.5,0.3), min_sum=1, max_sum=1,
+ weight_seq = generatesequence())
```

- Add a “risk budget” objective and a “return” objective:

```
> EqRiskConstr <- add.objective(EqRiskConstr,
+ type="risk_budget", name="CVaR",  enabled=TRUE,
+ min_concentration=TRUE, arguments = list(clean='boudt',
+ p=(1-1/12)))
> EqRiskConstr <- add.objective(constraints=EqRiskConstr,
+ type="return", name="pamean", enabled=TRUE, multiplier=0,
arguments = list(n=12))
```

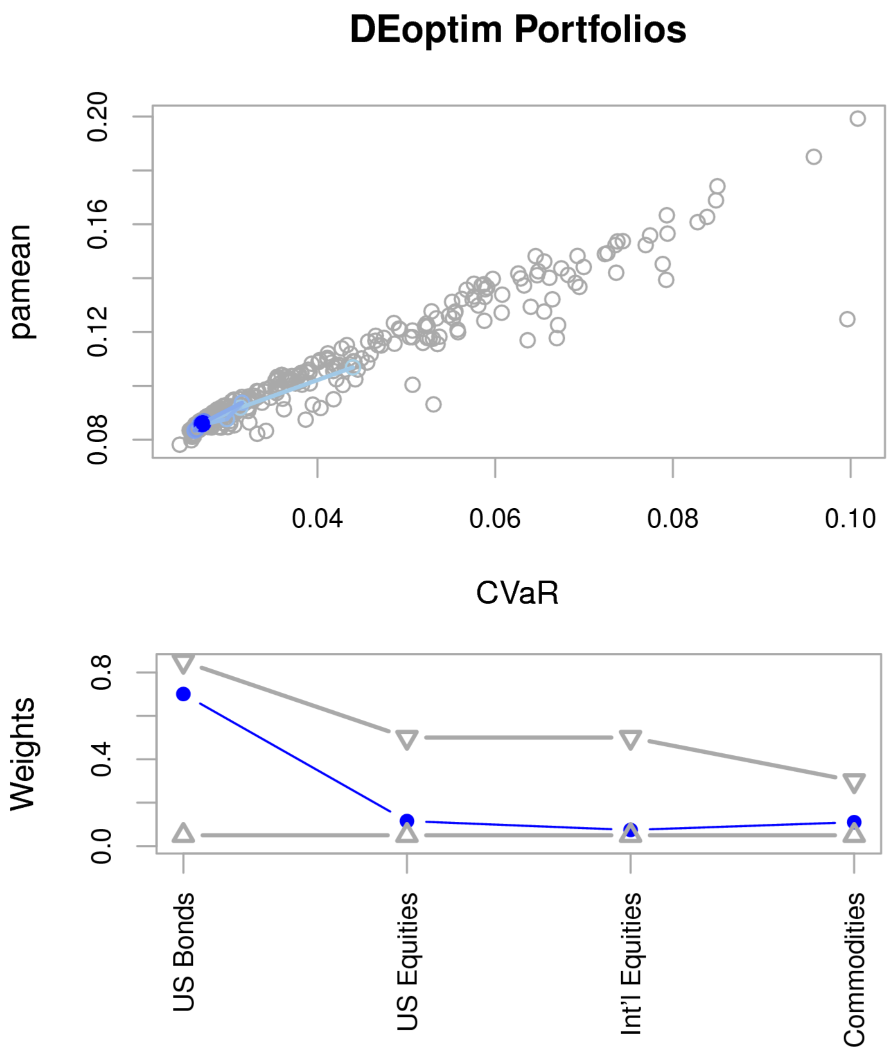
  - The zero multiplier in the return objective means that return will be calculated, but won't affect the optimization.

Use DEoptim

- Why DEoptim?
  - All numerical optimizations are a tradeoff between speed and accuracy
  - This space may well be non-convex in real portfolios
  - DEoptim will get more directed with each generation, rather than the uniform search of random portfolios
  - Allows more logical 'space' to be searched with the same number of trial portfolios for more complex objectives
- Specify DEoptim as the solver:

```
> EqRiskResultDE<-optimize.portfolio(R=indexes[,1:4],
+ constraints=EqRiskConstr, optimize_method='DEoptim',
+ search_size=2000, trace=TRUE, verbose=FALSE)
```

DEoptim Equal Risk Results



- DEoptim doesn't test many portfolios on the interior of the portfolio space
- Early generations search a wider space
- Later generations increasingly focus on the space that is near-optimal
- Random jumps are performed in every generation to avoid local minima

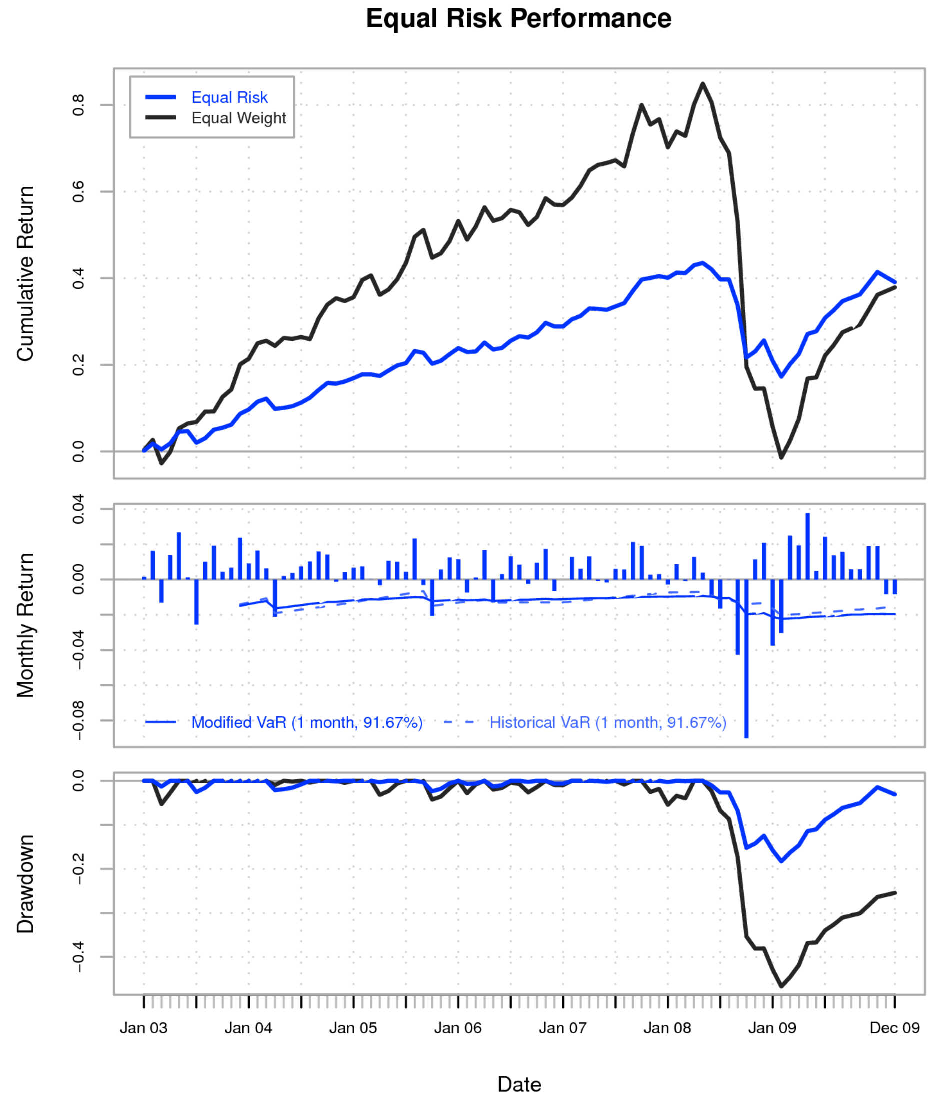
Run Optimizer Through Time

- Now we provide period information for rebalancing:

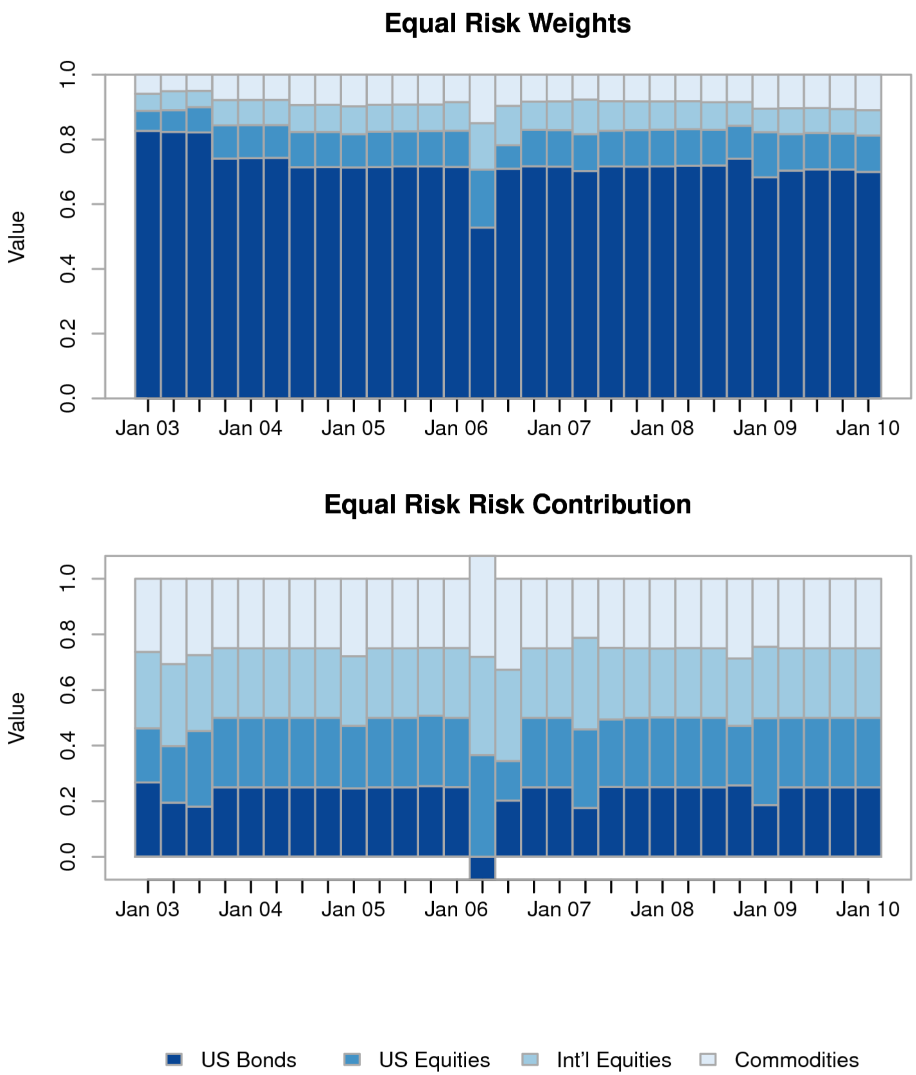
```
> EqRiskResultDERebal <-
+ optimize.portfolio.rebalancing(R=indexes[,1:4],
+ constraints = aConstraintObj, # our constraints object
+ optimize_method="DEoptim", # provide numeric sol'ns
+ trace=FALSE, # set verbosity for tracking
+ rebalance_on='quarters', # any xts 'endpoints'
+ trailing_periods=NULL, # calculation from inception
+ training_period=36, # starting period for calculation
+ search_size=3000) # parameter to Deoptim, increase?
```

DEoptim Equal Risk Results

- Equal Risk objective provides a smoother return
- ... that underperforms the equal weight portfolio in most periods
  - given that it has no view on returns and supresses weights in riskier assets
- ... but has a much smaller drawdown when things get ugly



Equal Risk Results

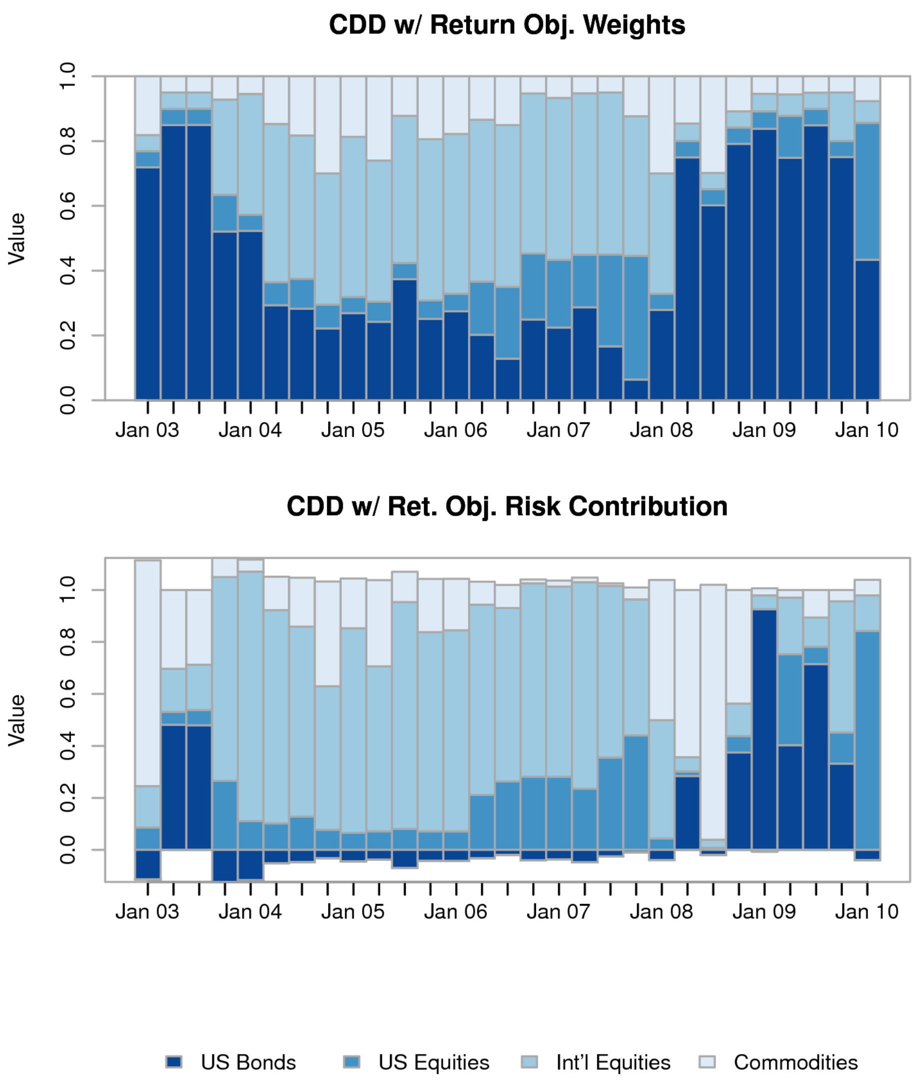


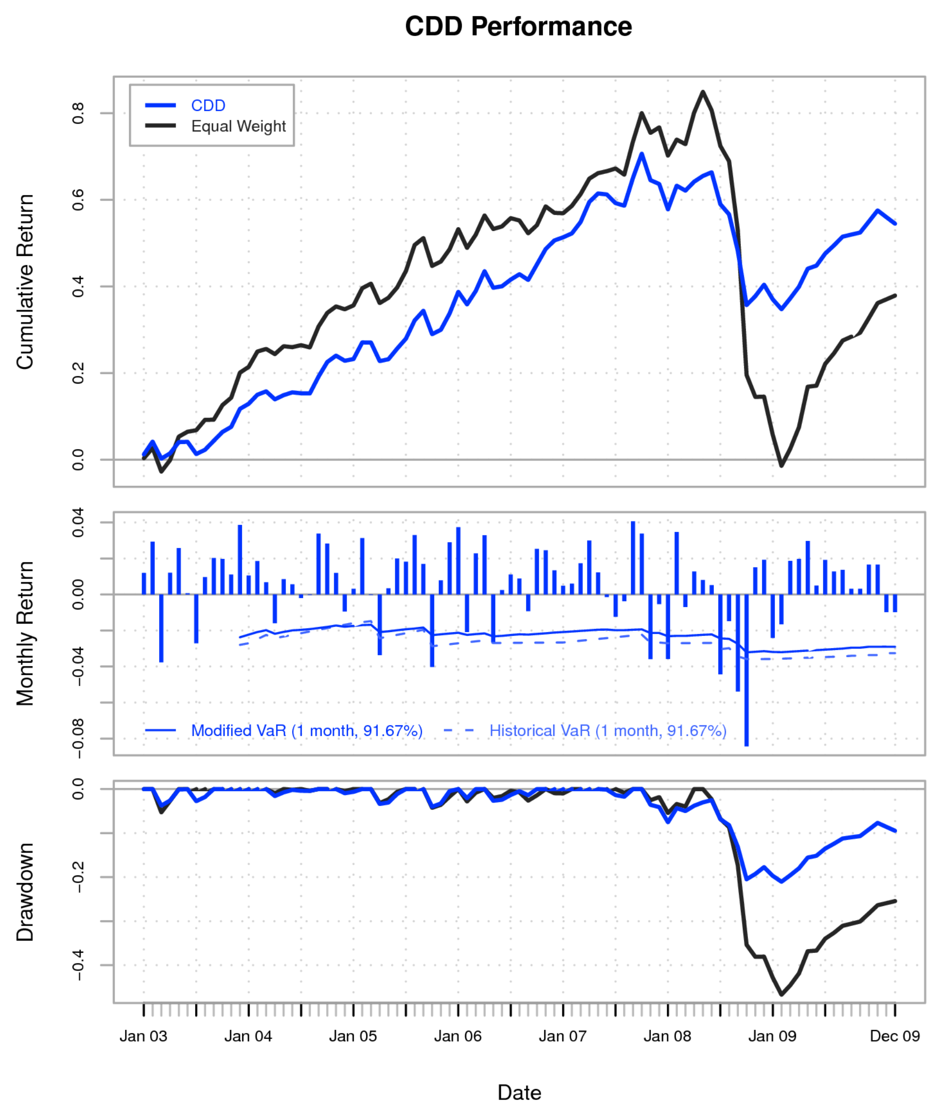
- We get a nearly-equal risk portfolio in almost all cases.
- 2Q2006 stands out as an exception
  - Maybe no feasible solution at the time?
  - Increase the search size?
  - Other DEoptim parameters?

Example: Mean-CDD

- Conditional Drawdown at Risk (CDD or CDaR) is the mean of the worst p% drawdowns proposed by Uryasev.
  - Another downside risk metric, but different than ES in that it does not assume independence of observations
  - Use the `name="CDD"` in `add.objective` to specify
- Qualitatively similar results to ES
  - Higher allocations to US Bonds and Int'l Stocks
- This fits in the broad class of "modified Sharpe" portfolio objectives

CDD with Return Objective





Using more iron...

- *PortfolioAnalytics* uses Revolution's *foreach* package to run multiple optimizations to get a set of optimal portfolios
- DEoptim may only find a near-optimal solution, does it matter, or is it close enough?
- Examining the results of multiple runs – toward the central limit theorem
- `optimize.portfolio.parallel` will run an arbitrary number of portfolio sets in parallel
  - Develop confidence bands around your optimal solution
  - Show where the optimizer makes tradeoffs between assets
  - Highlight where you need larger number of portfolios or generations

Roadmap

- Additional portfolio analysis functions
- More portfolio-aware risk/return functions
- Bi-directional as.* functions for portfolioSpec in *fPortfolio*
- More testing, documentation, and demo code
- CRAN release

Contributions and Collaboration are Encouraged!

Getting Your Objectives Right

- What are *your* business objectives?
  - Most literature uses objectives too simple to be realistic
  - Most software reinforces this
- Random Portfolios help you see the shape of the feasible space
  - The scatter chart shows the space covered by the portfolios that meet your constraints
- Rebalancing periodically and examining out of sample performance will help refine objectives
- DEoptim and parallel runs are valuable as things get more complex