R 中 portfolio 包的比较

蓝海

R 中比较常见的资产优化组合相关的包有 fPortfolio, parma 和 portfolioanalytics。下边我们通过几个例子来看看它们各自的特性，以及存在的问题。我们考虑资产组合中比较基本的三个问题：

- 给定风险，计算期望收益最大的组合
- 给定期望收益，计算风险最小组合
- 计算市场有效边际。

具体算例为

- 三资产模型
- 参数为：$\mu_1 = \mu_2 = 8\%, \mu_3 = 5\%, \sigma_1 = 20\%, \sigma_2 = 21\%, \sigma_3 = 10\%$ and $\rho_{i,j} = 80\%, i \neq j$
- 问题：
- Q1：给定风险 $\sigma_p = 15\%$，寻找期望收益最大组合。
- Q2：给定收益 $r_p = 7\%$，寻找风险最小的组合。
- Q3：计算市场有效边际。

任何一个资产优化组合包都应该提供以上三个基本的功能，除此之外，应该考虑对更实际、更复杂问题的支持或者拓展的可能。

fPortfolio

安装命令

```
install.packages('fPortfolio')
```

测算 Q1（可以做空）

问题简单，接口的调用也很简洁。唯一需要注意的是大多数组合优化包在设计时都是从输入投资回报的原始数据开始的。对于我们这样直接给出期望收益和方差的问题，反而需要产生一些伪回报的随机数，以满足函数借口的调用检查。

```
require('fPortfolio')
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
```

```
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting<-my_mean_var_setting(NULL,NULL)
mySpec <-portfolioSpec()
setTargetRisk(mySpec)<-0.15
setSolver(mySpec)<-'solveRshortExact'
port<-maxreturnPortfolio(setting,mySpec,constraints = 'Short')
print(port)
```

```
##
## Title:
##  MV Return Maximized Efficient Portfolio
##  Estimator:         covEstimator
##  Solver:            solveRshortExact
##  Optimize:          maxReturn
##  Constraints:       Short
##
## Portfolio Weights:
##     A1     A2     A3
## 0.3833 0.2020 0.4147
##
## Covariance Risk Budgets:
##     A1     A2     A3
## 0.4898 0.2582 0.2520
##
## Target Return and Risk:
## [1] 0.0676
##
## Description:
##  Tue May 16 19:41:00 2017 by user:
```

这可能是完成 Q1 问题最为简洁的代码。

测算 Q1（不可做空）

可是 fPortfolio 的设计者在构建该软件包的时候，是试图从时序数据开始处理的。对于这样简单的直接给出期望回报和方差的例子在计算的时候反而问题很多。大多数问题都是涉及数据封装。尽管从理论上模型只需要依赖于给出的期望回报和方差，可是在实际的程序中，尤其是一些独立的优化方法调用以及风险度量的模块里头不太合理的访问了原始的时序数据，从而形成了封装不严谨的问题。比如我们的例子中对于期望收益和风险的计算就是不正确的。

```
require('fPortfolio')
require('Rsocp')
require('MASS')

my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting<-my_mean_var_setting(NULL,NULL)
faked_data<-mvrnorm(100, setting$mu, setting$Sigma)
colnames(faked_data)<-c('A1','A2','A3')
mySpec <-portfolioSpec()
setTargetRisk(mySpec)<-0.15
setSolver(mySpec)<-'solveRsocp'
setEstimator(mySpec)<-'my_mean_var_setting'
myData<-portfolioData(as.timeSeries(faked_data),mySpec)
port<-maxreturnPortfolio(myData,mySpec,constraints = 'Short')
print(port)
```

尽管我们为了使用 fportfolio 而人为生成了伪时序数据，但是奇怪的是后续在 Rsocp 包中又试图将作者自定义的类 timeSeries 转化为 R 原生数据类型 vector 从而估计出一个合理的缩放比例进行数值计算的精度控制。这是典型的不合理使用数据封装的例子，达成同样的目的，其实可以使用 Sigma 数据中的最大数。

R 包 Rsocp 是老式 c 程序包 socp 的 R 接口拓展。该程序解决如下问题。

$$\min f^t x$$
$$s.t. ||A_i^t x + b_i|| = c_i^t x + d_i$$

对应的接口函数.rsocpArguments 处理参数的转换与准备问题。我们可以使用如下方式

```
debug(maxreturnPortfolio)
```

查看具体问题出现的地方。发现在函数.rsocpArguments 中出现一处问题：访问 eqsumW[2, -1] 用以生成约束 $\sum w_i = 1$。这种硬访问的方式很容易出现前后处理不一致的情况。果然在准备 eqsumW 的函数中，原本设计的函数是针对最小方差的问题设计。因此在 eqsumW 中第一行是表述目标收益的约束，第二行是进行资产权重的约束，比如全部投资约束则要求总权重为 1。而在最大收益的模型中，第一项目的约束是不存在的。这里有两种可以的处理方法：一是保持一个约束存在，目标值为 NA，我们在.rsocpArguments 中只是访问第二行。如果这样处理的话，我们需要在准备的 eqsumW 的函数中保留那些包含 NA 的约束，而系统似乎有意的以 NA 作为自由变量的约束表达方式。所以我们决定不采用这种方式，还是在.rsocpArguments 函数中，将 eqsumW[2, -1] 改为 eqsumW[1, -1]。验证的时候可以用

```
trace('.rsocpArguments',edit=T)
```

进行修改验证，只是需要注意这种修改不是永久的。验证通过后，我们需要通过下载 fPortfolio 的源文件，修改后重新安装本地版本的包。下载地址https://cloud.r-project.org/。具体步骤为：

1. 下载并解压 fPortfolio
2. 打开解压后子目录 R 下文件 solve-Rsocp.R
3. 替换 eqsumW[2,-1] 为 eqsumW[2,-1]，同时替换 eqsumW[2,1] 为 eqsumW[1,1]
4. 修改 package 目录下的 DESCRIPTION 文件，以显示我们修改了一个改进版本
5. 在上级目录下运行命令 R CMD build fPortfolio，生成了 fPortfolio_3011.82.tar.gz
6. 安装修改打包后的 package，命令为 R CMD INSTALL fPortfolio_3011.82.tar.gz （注意大小写敏感）

```
require('fPortfolio')
require('Rsocp')
require('MASS')

my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
```

```
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting<-my_mean_var_setting(NULL,NULL)
faked_data<-mvrnorm(100, setting$mu, setting$Sigma)
colnames(faked_data)<-c('A1','A2','A3')
mySpec <-portfolioSpec()
setTargetRisk(mySpec)<-0.15
setSolver(mySpec)<-'solveRsocp'
setEstimator(mySpec)<-'my_mean_var_setting'
myData<-portfolioData(as.timeSeries(faked_data),mySpec)
port<-maxreturnPortfolio(myData,mySpec,constraints = 'Short')
print(port)
```

```
##
## Title:
##  MV Return Maximized Efficient Portfolio
##  Estimator:         my_mean_var_setting
##  Solver:            solveRsocp
##  Optimize:          maxReturn
##  Constraints:       Short
##
## Portfolio Weights:
##     A1     A2     A3
## 0.3833 0.2020 0.4147
##
## Covariance Risk Budgets:
##     A1     A2     A3
## 0.4955 0.2654 0.2391
##
## Target Returns and Risks:
##   mean     mu    Cov  Sigma   CVaR    VaR
## 0.0694 0.0676 0.1581 0.1500 0.2742 0.2188
##
## Description:
##  Tue May 16 19:41:01 2017 by user:
```

重复运行之前不能正常运行的代码，在我们修改错误后得到如上的结果。

测算 Q2（可以做空）

对于可以做空的情况，因为在理论上存在 close-form 的解，所以使用 solveRshortExact 作为 solver 是最为高效的做饭。同时需要注意虽然在理论上使用 solveRquadprog 也应该能够求解，但是由于接口设计的欠缺，实际上是不能使用的。

```
require('fPortfolio')
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting<-my_mean_var_setting(NULL,NULL)
mySpec <-portfolioSpec()
setTargetReturn(mySpec)<-0.07
setSolver(mySpec)<-'solveRshortExact'
port<-efficientPortfolio(setting,mySpec,constraints = 'Short')
print(port)
```

```
##
## Title:
##  MV Efficient Portfolio
##  Estimator:         covEstimator
##  Solver:            solveRshortExact
##  Optimize:          minRisk
##  Constraints:       Short
##
## Portfolio Weights:
##     A1     A2     A3
## 0.4300 0.2367 0.3333
##
## Covariance Risk Budgets:
##     A1     A2     A3
## 0.5232 0.2880 0.1888
```

```
##
## Target Return and Risk:
## [1] 0.07
##
## Description:
##  Tue May 16 19:41:01 2017 by user:
```

测算 Q2（不许做空）

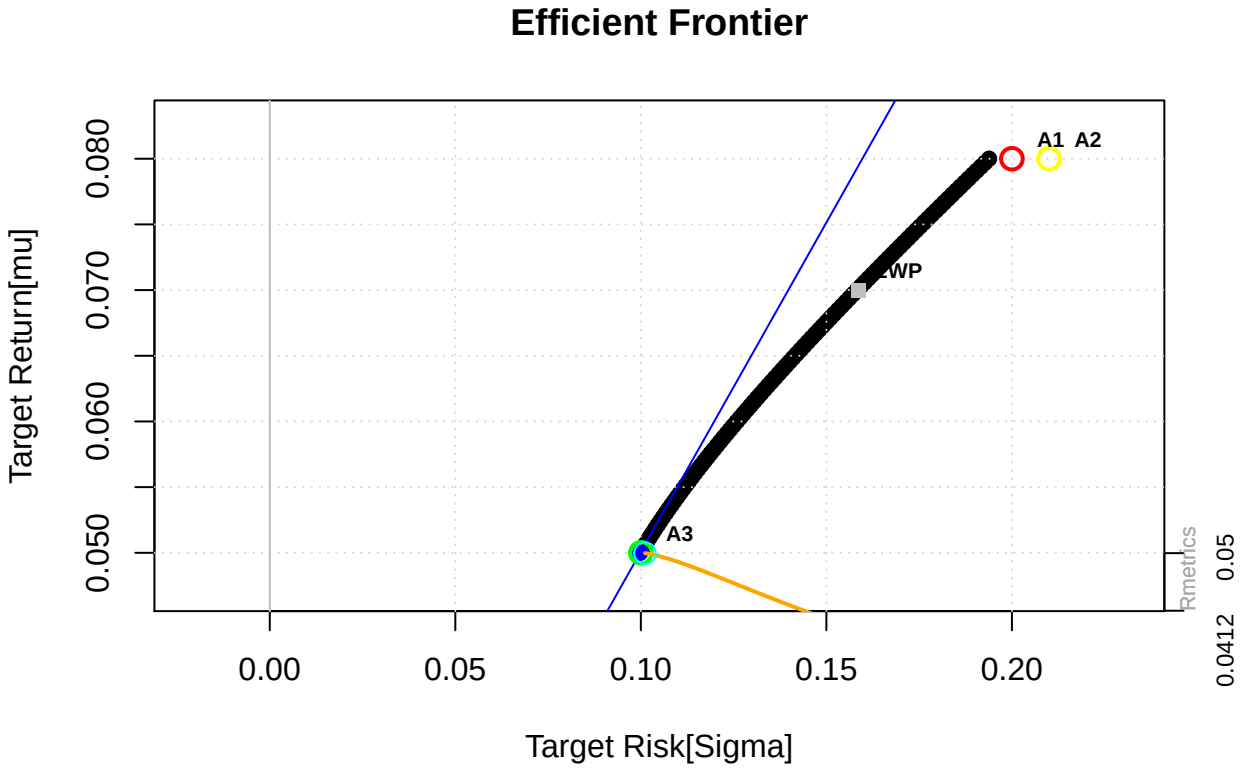
```
require('fPortfolio')
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting<-my_mean_var_setting(NULL,NULL)
mySpec <-portfolioSpec()
setTargetReturn(mySpec)<-0.07
setSolver(mySpec)<-'solveRquadprog'
port<-efficientPortfolio(setting,mySpec,constraints = 'LongOnly')
print(port)
```

```
##
## Title:
##  MV Efficient Portfolio
##  Estimator:         covEstimator
##  Solver:            solveRquadprog
##  Optimize:          minRisk
##  Constraints:       LongOnly
##
## Portfolio Weights:
##     A1     A2     A3
```

```
## 0.4300 0.2367 0.3333
##
## Covariance Risk Budgets:
##     A1     A2     A3
## 0.5232 0.2880 0.1888
##
## Target Return and Risk:
## [1] 0.07
##
## Description:
##  Tue May 16 19:41:01 2017 by user:
```

测算 Q3（可以做空）

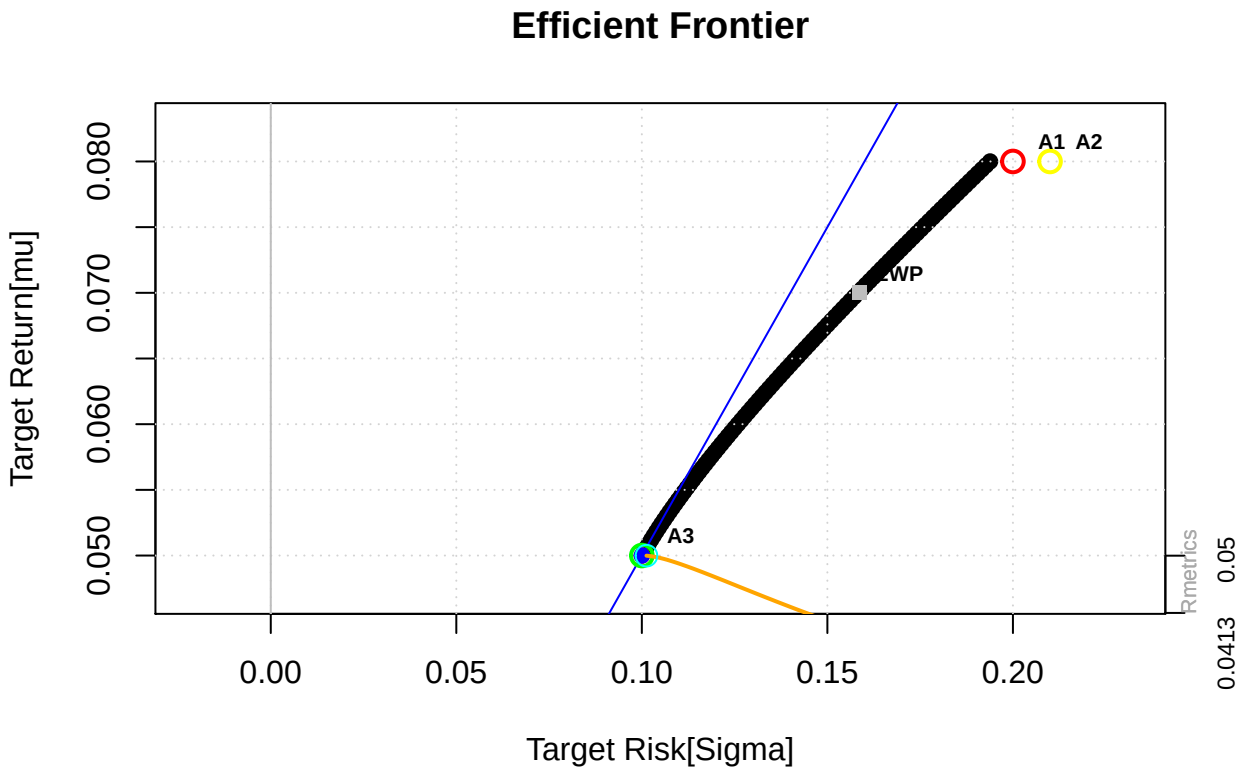
```
require('MASS')
require('fPortfolio')
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting<-my_mean_var_setting(NULL,NULL)
faked_data<-mvrnorm(100, setting$mu, setting$Sigma)
colnames(faked_data)<-c('A1','A2','A3')
mySpec <-portfolioSpec()
setNFrontierPoints(mySpec)<-100
setEstimator(mySpec)<-'my_mean_var_setting'
setSolver(mySpec)<-'solveRshortExact'
myData<-portfolioData(as.timeSeries(faked_data),mySpec)
shortFrontier<-portfolioFrontier(myData,mySpec,constraints = "Short")
tailoredFrontierPlot(shortFrontier,return='mu',risk='Sigma',mText='MP Portfolio - Shorting Allowed
```



测算 Q3（不许做空）

```
require('MASS')
require('fPortfolio')
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting<-my_mean_var_setting(NULL,NULL)
faked_data<-mvrnorm(100, setting$mu, setting$Sigma)
colnames(faked_data)<-c('A1','A2','A3')
mySpec <-portfolioSpec()
```

```
setNFrontierPoints(mySpec)<-100
setEstimator(mySpec)<-'my_mean_var_setting'
myData<-portfolioData(as.timeSeries(faked_data),mySpec)
longFrontier<-portfolioFrontier(myData,mySpec,constraints = "LongOnly")
tailoredFrontierPlot(longFrontier,return='mu',risk='Sigma',mText='MP Portfolio - Long Only')
```



关于 fPortfolio 的总结

这是一个比较广泛使用投资组合包，其特点是

- 接口简单
- 包括线性、整数、二次规划等确定性算法，可以基于 Var, CVaR 等计算
- 不包括随机算法，因而对于非凸规划问题没有现成的 solver
- 包括了丰富的基于历史数据的参数估计方法，流行的管制方法等
- 对投资组合有回测功能
- 扩展容易

其缺点是

- 因为要包含太多的 solver 使得接口设计混乱，封装失败，从而错误极多。建议使用熟悉的套路或者先验证后再使用不熟悉的功能

Parma

安装命令

```
install.packages('parma')
```

测算 Q1（可以做空）

因为对应问题的描述，都统一的封装在 parmaspec 的生成函数中，所以该函数的调用十分复杂。通过

```
args(parmaspec)
```

我们可以看到函数的具体接口。详细一些的帮助，可以通过一下命令查看。

```
help(parmaspec)
```

相对于 fPortfolio 改进的地方是，在问题描述成功生成后，能够使用什么样的 solver 来解决问题，可以通过 show(mySpec) 来查看。比如我们这里就只能够使用 SOCP 来计算。

```
require('parma')
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting <- my_mean_var_setting()
mySpec<-parmaspec(S=setting$Sigma,forecast = setting$mu, risk='EV',riskType = 'maxreward',riskB=0.
#show(mySpec)
sol = parmasolve(mySpec,type='SOCP')
weights(sol)
```

```
##        A1        A2        A3
## 0.3832635 0.2020196 0.4147269
```

```
parmarisk(sol)
```

```
##     EV
```

```
## 0.0225
```

```
parmareward(sol)
```

```
## [1] 0.06755899
```

注意此处 risk 的描述用的是 EV——方差，而我们通常设定的是均方差 15%。因而计算控制的 EV 为其平方 2.25%。

测算 Q1（不许做空）

```
require('parma')
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting <- my_mean_var_setting()
mySpec<-parmaspec(S=setting$Sigma,forecast = setting$mu, risk='EV',riskType = 'maxreward',riskB=0.
#show(mySpec)
sol = parmasolve(mySpec,type='SOCP')
weights(sol)
```

```
##        A1        A2        A3
## 0.3832635 0.2020196 0.4147269
```

```
parmarisk(sol)
```

```
##     EV
## 0.0225
```

```
parmareward(sol)
```

```
## [1] 0.06755899
```

测算 Q2 （可以做空）

```
require('parma')
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting <- my_mean_var_setting()
mySpec<-parmaspec(S=setting$Sigma,forecast = setting$mu, risk='EV',riskType = 'minrisk',target=0.0
show(mySpec)
```

```
##
## +---------------------------------+
## |       PARMA Specification       |
## +---------------------------------+
## No.Assets        : 3
## Problem          : QP,SOCP
## Input            : Covariance
## Risk Measure : EV
## Objective        : minrisk
```

```
sol_socp = parmasolve(mySpec,type='SOCP')
weights(sol_socp)
```

```
##        A1        A2        A3
## 0.4298733 0.2367533 0.3333833
```

```
parmarisk(sol_socp)
```

```
##         EV
## 0.02505223
```

```
parmareward(sol_socp)
```

```
## [1] 0.0699993
```

```
sol_qp = parmasolve(mySpec,type='QP')
weights(sol_qp)
```

```
##        A1        A2        A3
## 0.4299803 0.2366864 0.3333333
```

```
parmarisk(sol_qp)
```

```
##         EV
## 0.02505325
```

```
parmareward(sol_qp)
```

```
## [1] 0.07
```

测算 Q2 （不许做空）

```
require('parma')
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,Sigma=Sigma)
}
setting <- my_mean_var_setting()
mySpec<-parmaspec(S=setting$Sigma,forecast = setting$mu, risk='EV',riskType = 'minrisk',target=0.0
show(mySpec)
```

```
##
## +---------------------------------+
## |       PARMA Specification       |
## +---------------------------------+
## No.Assets        : 3
## Problem          : QP,SOCP
## Input            : Covariance
## Risk Measure : EV
```

```
## Objective        : minrisk
```

```
sol_socp = parmasolve(mySpec,type='SOCP')
weights(sol_socp)
```

```
##        A1        A2        A3
## 0.4298733 0.2367533 0.3333833
```

```
parmarisk(sol_socp)
```

```
##         EV
## 0.02505223
```

```
parmareward(sol_socp)
```

```
## [1] 0.0699993
```

```
sol_qp = parmasolve(mySpec,type='QP')
weights(sol_qp)
```

```
##        A1        A2        A3
## 0.4299803 0.2366864 0.3333333
```

```
parmarisk(sol_qp)
```

```
##         EV
## 0.02505325
```

```
parmareward(sol_qp)
```

```
## [1] 0.07
```

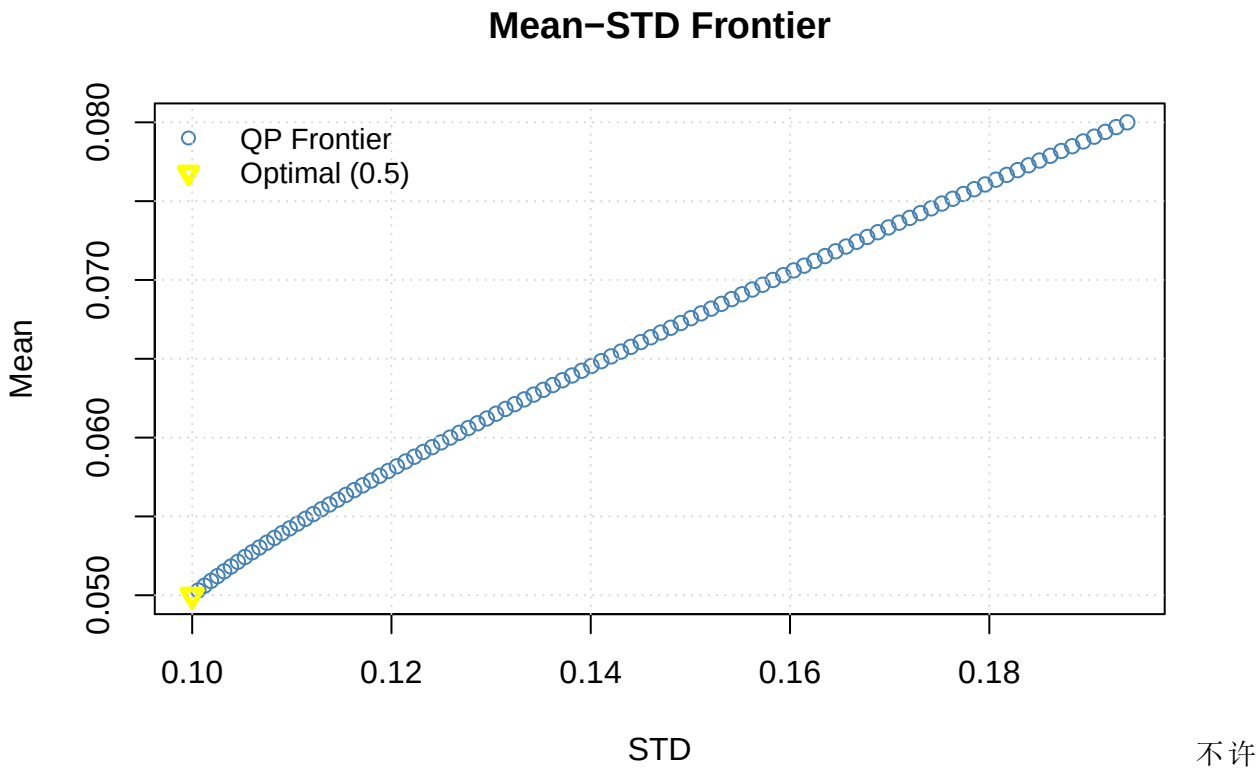
测算 Q3 （可以做空）

parma 包的一个好处是可以方便的使用并行计算。此处我们使用 parallel 包利用普遍存在的多核心 CPU 加快计算。

```
require('parallel')
require('parma')
cl = makeForkCluster(16)
mySpec<-parmaspec(S=setting$Sigma,forecast = setting$mu, risk='EV',riskType = 'minrisk',target=0.0
front_qp = parmafrontier(mySpec,n.points = 100,type='QP',cluster=cl)
#show(front_qp)
```

我们可以进一步的绘画出市场的最有边界来。

```
r1=sqrt(front_qp[,'EV'])
rw1=front_qp[,'reward']
mySpec2<-parmaspec(S=setting$Sigma,forecast = setting$mu, risk='EV',riskType = 'optimal',target=0.
#show(mySpec2)
opt_qp=parmasolve(mySpec2,type='QP')
#show(opt_qp)
plot(r1, rw1, type = 'p', col = 'steelblue', xlab = 'STD', ylab = 'Mean',
    main = 'Mean-STD Frontier')
points(sqrt(parmarisk(opt_qp)), parmareward(opt_qp), col = 'yellow', pch = 6, lwd = 3)
legend('topleft', c('QP Frontier',  paste('Optimal (',
    round(parmareward(opt_qp)/sqrt(parmarisk(opt_qp)), 3), ')', sep = '')), col = c('steelblue', '
    lty = c(0,  0), bty = 'n', cex = 0.9)
grid()
```



不许做空的情形相似，在此不在列出。

parma 总结

作为一个资产配置的计算包，parma 相对于之前的 fPortfolio 要友好很多。其主要特点是

- 逻辑结构简单，提示信息丰富
- 支持算法丰富

- 网站http://unstarched.net/r-examples/parma/custom-constraints/提供了很好的使用说明，并且有很多给予 parma 的实用的例子

缺点是

- 接口参数多，参数设置复杂
- 不支持投资组合回测
- 没有封装市场有效边界的绘画

PortfolioAnalytics

这个包是我测试的 3 个包中帮助文档最为全面的。在此我们显示一下，如何实用 R 自带的帮助系统来学习一个新的包的功能。在安装和载入包完成之后，我们使用如下命令可以得到关于包的一个初步介绍。

```
require('PortfolioAnalytics')
```

```
## Loading required package: PortfolioAnalytics
## Loading required package: zoo
##
## Attaching package: 'zoo'
## The following object is masked from 'package:timeSeries':
##
##     time<-
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
## Loading required package: xts
## Loading required package: foreach
## Loading required package: PerformanceAnalytics
##
## Attaching package: 'PerformanceAnalytics'
## The following objects are masked from 'package:timeDate':
##
##     kurtosis, skewness
## The following object is masked from 'package:graphics':
##
```

```
##     legend
```

```
help('PortfolioAnalytics')
```

成熟的 R 包，往往除了参考文档之外，还会提供若干个 vignette，它们描述了场景化的应用，往往比参考文档更有说明力。

```
vignette(all=F)
vignette('portfolio_vignette')
```

参照该文档我们可以进行针对 PortfolioAnalytics 的测试。但是，我们不得不加一句评论，PortfolioAnalytics 包与 RPortfolio 包类似，其设计初衷是面向历史回报数据的。我们此处采用的教学用的 “玩具” 问题，在求解上反而繁琐不少。

测试 Q1

```
require('MASS')
require('PortfolioAnalytics')
require('xts')

mySpec<-portfolio.spec(assets=c('A1','A2','A3'))
mySpec<-add.constraint(portfolio=mySpec,type='weight_sum',min_sum=1,max_sum=1)
mySpec<-add.objective(portfolio = mySpec,type='return',name='mean')
mySpec<-add.objective(portfolio = mySpec,type='risk',name='StdDev',target=0.15)
my_mean_var_setting<-function(x,spec)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,sigma=Sigma,m3=0,m4=0)
}
setting<-my_mean_var_setting(NULL,NULL)
faked_data<-mvrnorm(100, setting$mu, setting$sigma)
colnames(faked_data)<-c('A1','A2','A3')
faked_data<-ts(faked_data,start=c(2011,1),frequency=12)
rp<-random_portfolios(mySpec,10000,rp_method = 'simplex')
```

```
opt<-optimize.portfolio(faked_data,mySpec,optimize_method = 'random',rp=rp,trace=T)
```

尽管程序已经足够复杂，实际上任然没有能够解决我们的问题，而且即便是能够解决，随机算法得到的结果也不是理想的。所以可以说对于 Q1 类玩具问题，PortfolioAnalytics 是无能为力的。

测试 Q2

```
require('MASS')
require('PortfolioAnalytics')
require('ROI')
require('foreach')
require('iterators')
require('ROI.plugin.quadprog')
require('ROI.plugin.glpk')

mySpec<-portfolio.spec(assets=c('A1','A2','A3'))
mySpec<-add.constraint(portfolio=mySpec,type='weight_sum',min_sum=1,max_sum=1)
mySpec<-add.constraint(portfolio=mySpec,type='return',return_target=0.07)
mySpec<-add.objective(portfolio = mySpec,type='risk',name='var')
my_mean_var_setting<-function(R)
{
  mu<-c(0.08,0.08,0.05)
  names(mu)<-c('A1','A2','A3')
  Sigma<-matrix(0.8,3,3)
  Sigma[1,1]<-1
  Sigma[2,2]<-1
  Sigma[3,3]<-1
  var<-c(0.2,0.21,0.1)
  Sigma<-t(var*Sigma)*var
  list(mu=mu,sigma=Sigma,m3=0,m4=0)
}
setting<-my_mean_var_setting(NULL)
faked_data<-mvrnorm(100, setting$mu, setting$sigma)
colnames(faked_data)<-c('A1','A2','A3')
faked_data<-ts(faked_data,start=c(2011,1),frequency=12)

opt<-optimize.portfolio(faked_data,mySpec,optimize_method = 'ROI',monentFUN='my_mean_var_setting')
print(opt)
```

总结

我并没有完全完成这次的测试, 因为发现 PortfolioAnalytics 实际上不是为了解决这样的 “玩具” 问题而设计的, 强制的使用它解决这样的问题费事而无聊。综合计算能力, 接口设计和扩展性。我觉得 parma 是一个不错的选择。RPortfolio 需要慎重使用。而 parma 和 PortfolioAnalytics 则是面向科研和实际的生产力工具。后者则更是具备了回测的能力。parma 不具备回测能力, 需要结合 R 中做交易回测的包 blotter 来工作。