Quantitative Strategy Research using R

WISERCLUB 贾茹
2016-12-06

目录

书单

量化投资基础入门资料（转自易金超学长）

叙事故事类：
1，解读量化投资：西蒙斯用公式打败市场的故事
2，打开量化投资的黑箱 Inside the black box, Rishi Narang
3，打开高频交易的黑箱 All about high frequency trading，迈克尔.德宾
4，宽客人生：从物理学家到数量金融大师的传奇， 德曼Derman
5，宽客：华尔街顶级数量金融大师的另类人生， 斯科特•帕特森
备注：这些书基本上是故事类书籍，读起来非常快，可以对量化投资有一个大概的了解，包括量化历史，量化策略以及量化门派等。

理性普及类：
6，通向财务自由之路(原书第2版•珍藏版)，范 K.撒普 (Van K.Tharp)
7，交易圣经:系统交易赢利要诀，布伦特•奔富 (Brent Penfold)
备注：这两本书应该来说对于理性认识投资，特别是形成系统性交易思维还是挺有帮助的。

专业基础类(尽可能在学校就完成训练)：
8, 统计与金融，Statistics and Finanace, by David Ruppert
9, Option, future and other derivatives, by John Hull
10, The Econometrics of Financial Markets, by John Y. Campbell, Andrew Lo , A. Craig MacKinlay
11, Asset Pricing, by John H. Cochrane
备注：8基本上对于不是金融专业的人来说，非常好的普及知识，9也是。
10应该是很多做ALPHA以及事件驱动的好书，11是ALPHA的基础书。

量化投资研究之工具篇

量化投资技能树：数学，计算机，金融学。此处重点说说计算机方面需要的技能储备。

在我司做量化研究经常用到的工具

R packages

- 结构化数据 data.table
- 非结构化数据 rlist
- 文本数据与正则表达式 stringr
- 时间序列 xts zoo lubridate
- 可视化 ggplot2
- 高性能 Rcpp
- 并行计算 parallel, foreach
- 数据库相关 mongolite, jsonlite, RSQLite, rredis, DBI
- 网络爬虫 rvest, RCurl

数据库

- 关系型数据库 MySQL, SQLite

- 非关系型数据库 mongoDB, Redis

最基本的增删改查操作；以及通过R连接、操作数据库

版本控制工具 git

- 自己本地和远程的版本控制
- 团队协作

R

- 至少要刷过一遍 Advanced R。是否听说过以下概念？environment, functional progreamming, evaluation, expression, ... 不要求了如指掌，但至少听说过。
- 会调试代码，代码有问题能自己独立解决。
- 善用google。遇到问题能通过google快速找到解决方案。
- 了解术语，能简洁而准确地描述自己遇到的问题，能与同事进行高质量的交流。

如果你是自己做量化研究。。。

数据来源

wind，新浪，雅虎API等

研究、回测平台

在线回测平台：优矿(uqer)、米筐(RiceQuant)、聚宽(JoinQuant)

- 中国版Quantopian。
- 使用python
- 数据都是现成的，常用指标都自动计算好了，无需自己造轮子。
- 活跃的社区。
- 但据说也有一些坑：比如性能问题。

传送门：

- 优矿 https://uqer.io/home/
- 米筐 https://www.ricequant.com/
- 聚宽 https://www.joinquant.com/

国内程序化交易软件

国泰安Quantrader, wind量化平台, 国信Tradestation, 交易开拓者tradeblazer ...

R语言量化包：quantstrat

http://timtrice.github.io/backtesting-strategies/index.html

量化策略研究之思想篇

数学，计算机只是量化投资的实现工具，投资逻辑才是灵魂。

常见的策略有哪些?

- 股票策略：多因子模型，量化选股， T+0策略
- 期货策略：趋势类，反转类，套利类
- 期权策略

策略的来源?

经典策略（创新的基础是模仿），前沿paper（吸收新鲜思想），卖方研报（找灵感），自己顿悟。。。

实战：一个经典CTA策略

策略逻辑

趋势追踪策略。以低成本反复试探，试图捕捉大趋势。

- 价格突破前30天最高价开多、价格突破前30天最低价开空。
- 1%移动止损。

这里有两个参数：`N = 30`，`lossrate = 0.01`

回测信息

- 回测品种：螺纹钢连续
- 回测时间段： 2010-01-01 ~ 2016-10-21
- 采用日度数据

数据问题，没有处理主力合约换月，而是简单地拼接起来当做一个合约，实际中应该考虑移仓问题。简单起见，不考虑仓位调整问题，买卖都是一手，最多持有一手仓位

写策略

这里不使用任何回测框架，全部自己写

基本思路很简单，就是写一个for循环，依次扫过每一个bar, 判断止损条件，开仓信号，在每个bar结束时更新账户状态，同时记录每笔交易的信息。

回测结果主要是两个表：

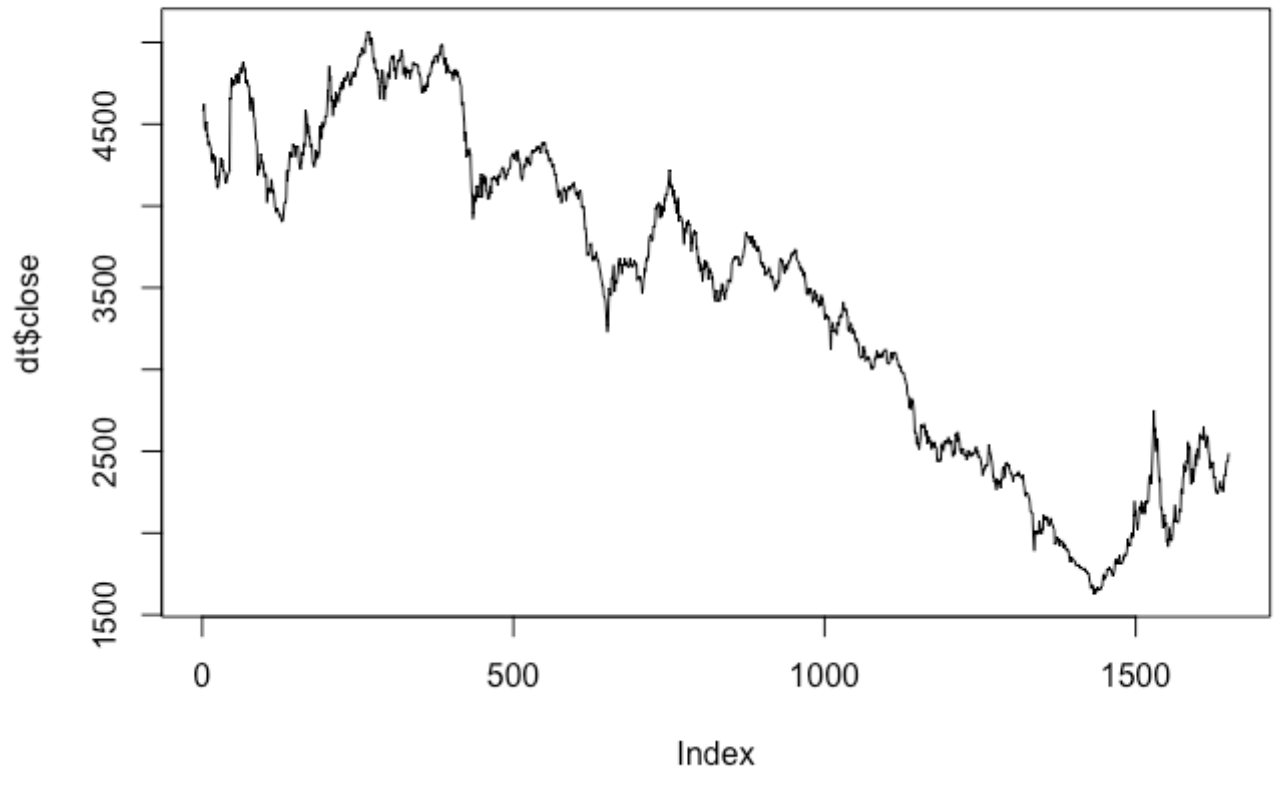
1. 逐日的账户状态（每天的：持仓状态，已平仓利润，持仓盈亏，手续费，保证金，其他策略signal等
2. 逐笔交易明细（每笔交易的：入场时间，入场价格，离场时间，离场价格，多or空

根据这两个表可以做更进一步的分析。

```
rm(list = ls())
library(data.table)
library(rlist)
library(pipeR)
library(ggplot2)

load("data/rb_dominant_daily.rda")
head(dt)
```

```
##        date   code open high  low close  volume
## 1: 20100104 rb1005 4523 4610 4523  4579 1809996
## 2: 20100105 rb1005 4592 4606 4573  4589 1320756
## 3: 20100106 rb1005 4587 4635 4582  4622 1481228
## 4: 20100107 rb1005 4641 4650 4389  4489 2196160
## 5: 20100108 rb1005 4505 4509 4412  4462 1520076
## 6: 20100111 rb1005 4495 4517 4476  4516 1130570
```

```
#View(dt)  # 看一下数据结构
plot(dt$close, type = "l")  # 有几波超级大趋势可以抓住
```

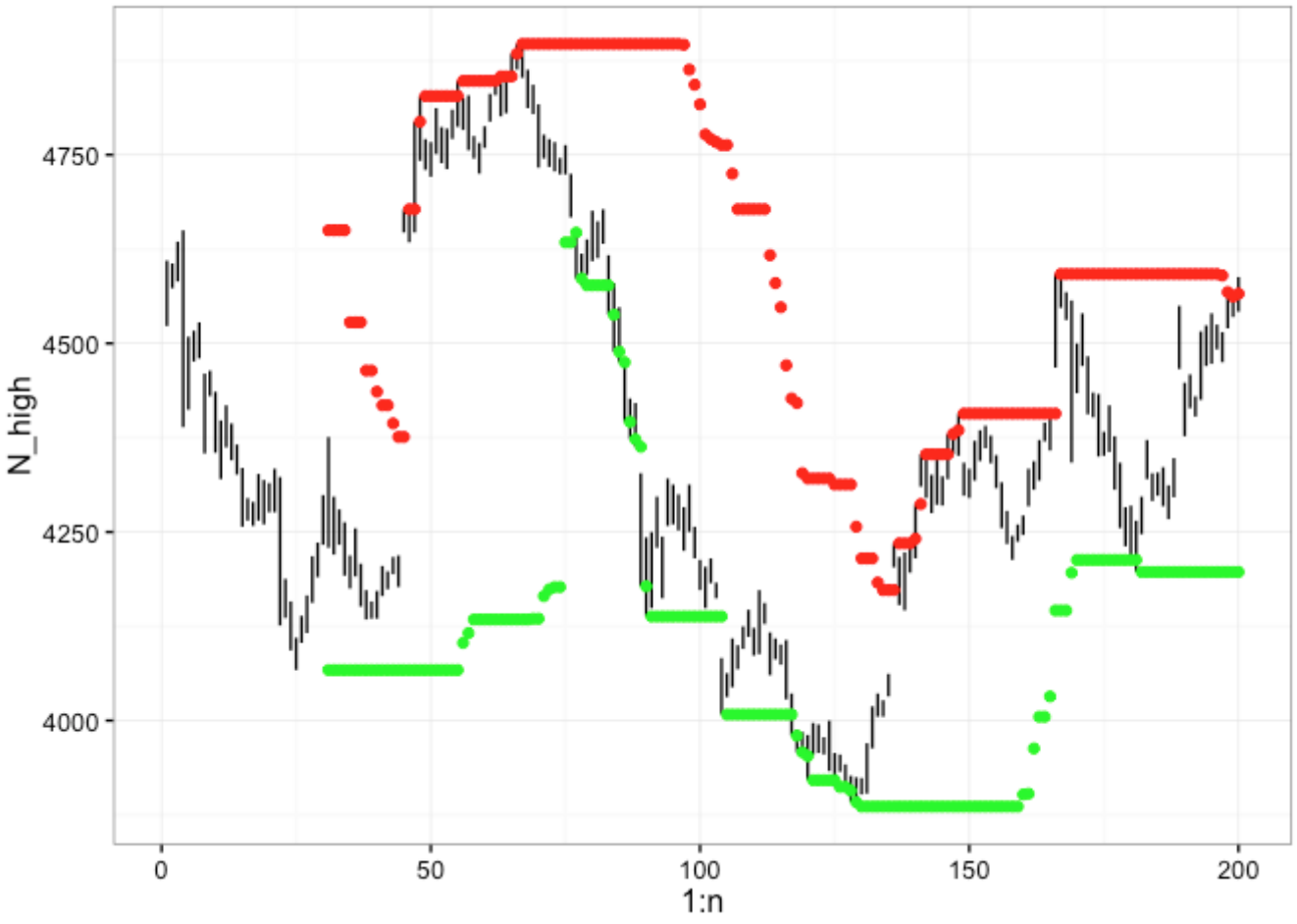



```
# 参数取值
N = 30
lossrate = .03

# 手续费 & 滑点
fee.rate <- 8e-5  #手续费单边万分之0.8
slippage <- 2 * 1  # 滑点2个tick * rb最小变动价位1
vm <- 10  # rb合约乘数为10。 volume_multiple
```

注意：

```
螺纹钢标准合约请看 http://www.shfe.com.cn/products/rb/standard/194.html
报价单位 4579元/吨
合约乘数 10吨/手
那么每张合约价值 = 45790元
每买卖一手合约手续费为 45790元 * 8e-5 = 3.6632 元

"2个滑点"意思是：模拟成交时的价格为理论价格向不利方向变动2个最小变动价位
1. 实际成交情况不如回测时那么理想。
2. 冲击成本，尤其是大资金量。
```

```
# 生成N日最高价、N日最低价 N_high, N_low。
# 这一步也可以放到for循环里面，但从代码效率方面考虑把它写在for循环外面
# 这种情况一定要注意：不要用到未来函数。 shift(...)
dt[, `:=`(
  N_high = shift(TTR::runMax(high, n = N), n = 1, type = "lag"),
  N_low = shift(TTR::runMin(low, n = N), n = 1, type = "lag")
)]

#看一下
n = 200   # 数据太多，看不清，只看前n个
p <- ggplot(dt[1:n], aes(x = 1:n))+
  geom_linerange(aes(ymin = low, ymax = high)) +   # high low
  geom_point(aes(y = N_high), color = "red", na.rm = TRUE) +
  geom_point(aes(y = N_low), color = "green", na.rm = TRUE) +
  theme_bw()
p
```




```
# 下面开始回测。。。

# 初始化状态变量
ptr <- 1L  # ptr: pointer “指针”  for ( ptr in 1:nrow(dt) ) {...backtest code...}
position <- 0L  # 当前持有的仓位。只有三个取值：{-1L, 0L, 1L}，不考虑仓位调整问题。
stop_price <- NA_real_   # 移动止损价。没有仓位时为NA_real_
rec_stop_price <- NA_real_   # **记录**移动止损价。
closed_profit <- 0.0   # 已平仓利润（累计）
position_profit <- 0.0   # 当前持仓浮动盈亏
fee <- 0.0   # 手续费（累计）


out <- list()  # 逐日账户状态，list of list，for循环每循环一次，后面append一个list
trades <- list()  # 逐笔交易记录，list of list。
# Two util functions
bar <- function(w){
  dt[[w]][ptr]
}
pre <- function(w, n = 1){
  dt[[w]][(ptr - n - 1):(ptr - 1)]
}
# eg1. bar("close") returns close price of this bar
# eg2. pre("close") returns the previous close price
# eg3. pre("high", 30) returns the high prices of latest 30 days

# 循环
for(ptr in 1:nrow(dt))   # ptr: pointer
{
  # PART 0. 数据准备
  high <- bar("high")  # 当前bar最高价
  low <- bar("low")    # 当前bar最低价
  sig_long <- high > bar("N_high")  # 开多信号 bool
  sig_short <- low < bar("N_low")   # 开空信号 bool

  if (is.na(sig_short) | is.na(sig_long)) next  # 跳过前N个

  # PART 1. 检查止损，更新止损点

  if (position != 0L) {  # 如果持有仓位。。。
    stopifnot(!is.na(stop_price)) # stop_price 不能是缺失值，否则程序应停止报错

    # 是否触及止损点：两个bool变量
    stop_long <- position == 1L & low < stop_price  # bool 多头时是否触及止损
    stop_short <- position == -1L & high > stop_price  # bool 空头时是否触及止损

    if (stop_long | stop_short){
      # 如果触及止损点。为了简洁两种情况写在一起了，也可以分开写。

      # 平掉现有仓位
      leave_price <- stop_price - position * slippage  # long: - short: +
      closed_profit <- closed_profit + position * (leave_price - enter_price) * vm
      fee <- fee + leave_price * vm * fee.rate
```

```
    position_profit <- 0.0

    # 添加交易记录
    trade_out <- list(
      enter_date = enter_date,  # 入场时间（开仓时会定义这个变量）
      enter_price = enter_price,    # 入场价格（开仓时会定义这个变量）
      leave_date = bar("date"),  # 离场时间：就是当前bar的date值
      leave_price = leave_price,  # 离场价格：刚刚算好的。
      side = position,   # 买卖方向：多还是空
      commission = leave_price * vm * fee.rate + enter_price * vm * fee.rate #佣金
    )
    trades <- list.append(trades, trade_out)

    # 重置状态变量
    position <- 0L
    rec_stop_price <- stop_price
    stop_price <- NA_real_
    enter_price <- NA_real_
    enter_date <- NA_real_
    rm(trade_out)

  } else { # 如果没有触及止损点。。。
    rec_stop_price <- stop_price
    # 更新移动止损点
    if (position == 1L){
      stop_price <- max(stop_price, high * (1 - lossrate))
    } else if (position == -1L){
      stop_price <- min(stop_price, low * (1 + lossrate))
    } else {
      stop(102)
    }
  } # End if(stop_long | stop_short)
} # End if(position == 0L)

# PART 2. 处理开仓信号
if (position == 0L) {
  # 情况1：没有任何仓位
  if (sig_long & !sig_short) {
    # 情况1.1：开多
    enter_price <- max(bar("N_high"), bar("open")) + slippage  # 记录入场价格
    enter_date <- bar("date")  # 记录入场时间
    stop_price <- enter_price * (1 - lossrate)  # 设好止损价
    rec_stop_price <- stop_price
    position <- 1L   # position 切换成 1L
    fee <- fee + enter_price * vm * fee.rate  # 累加手续费
  } else if (sig_short & !sig_long) {
    # 情况1.2：开空
    enter_price <- min(bar("N_low"), bar("open")) - slippage
    enter_date <- bar("date")
    stop_price <- enter_price * (1 + lossrate)
    rec_stop_price <- stop_price
    position <- -1L   # position 切换成 -1L
    fee <- fee + enter_price * vm * fee.rate
  } else if (sig_long & sig_short) {
    # 情况1.3：(极其少见)多空信号都出现了
    # you may add some message ...
  } else {
    # 情况1.4：既没有开多信号，也没有开空信号。
    # pass
  }

} else if (position == 1L) {
  # 情况2：持有多仓
  # 持有多仓的情况下出现了开空信号：平掉现有的仓位，再反向开仓
  # pass
} else if (position == -1L){
  # 情况3：持有空仓
  # 持有空仓的情况下出现了开多信号：平掉现有的仓位，再反向开仓
  # pass
} else {
  stop(101)
}

# PART 3. 每个bar结束时，保存信息至out变量
position_profit <- ifelse(position == 0L,
                          0.0,
                          position * (bar("close") - enter_price) * vm
                          ) # 计算持仓浮动盈亏
bar_out <- list(
```

```
    date = bar("date"),
    position = position,   # bar结束时 持仓情况 %in% {1L, 0L, -1L}
    closed_profit = closed_profit,   # 已平仓利润
    position_profit = position_profit,  # 持仓盈亏
    close = bar("close"),
    market_value = bar("close") * vm,   # 合约价值
    margin = abs(position) * bar("close") * vm * 0.05,  # 保证金占用
    N_high = bar("N_high"),
    N_low = bar("N_low"),
    low = low,
    high = high,
    stop_price = rec_stop_price  # 如果有持仓的话，记录移动止损价。
  )

  out <- list.append(out, bar_out)
  rec_stop_price <- NA_real_
}
```

生成两个data.table: `out_dt` & `trade_dt`

- out_dt: 每天结束时的账户状态（已平仓利润，头寸，浮盈浮亏，保证金占用，其他信息）
- trade_dt: 每笔交易明细（入场时间，入场价格，离场时间，离场价格，多空）

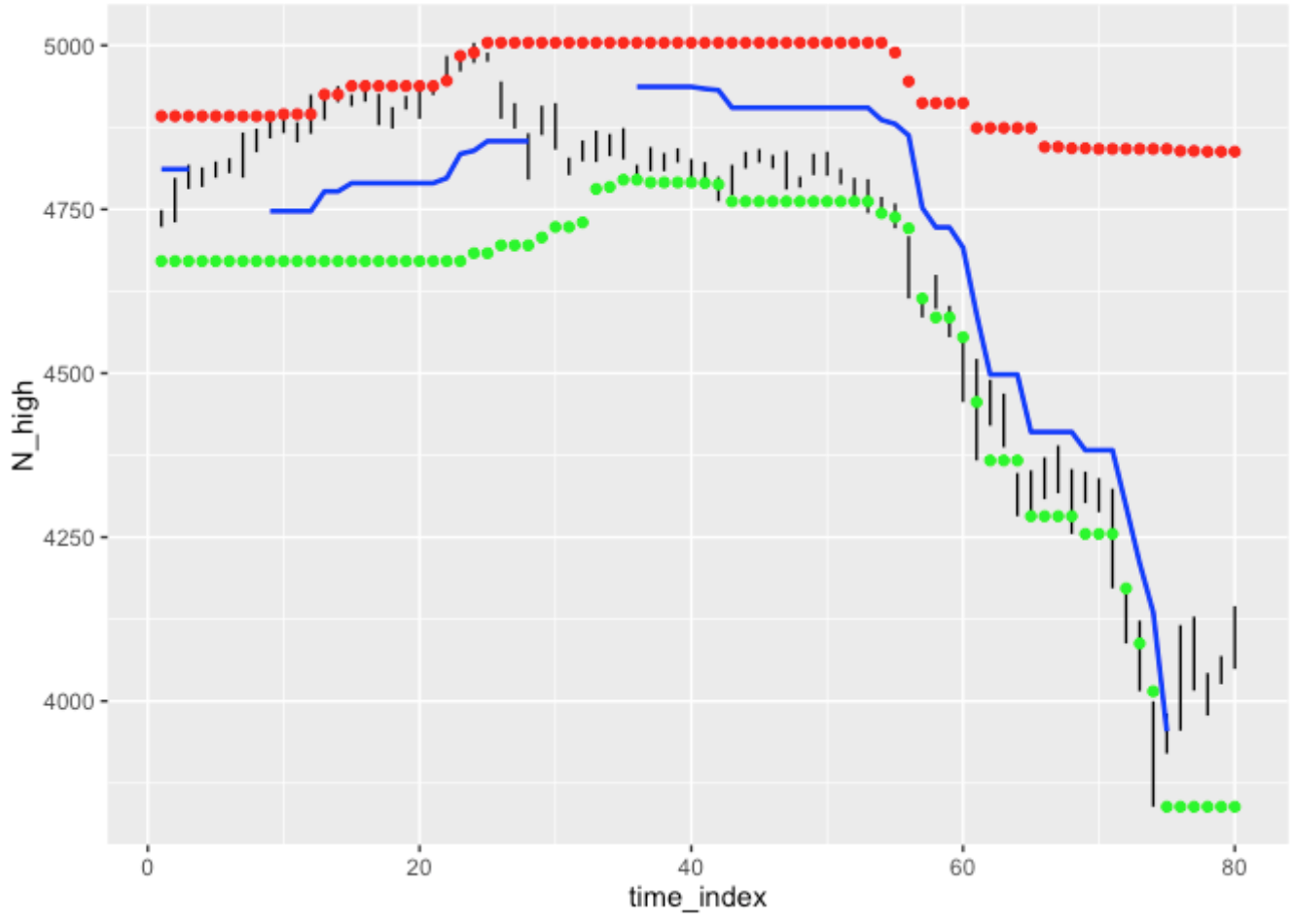
```
out_dt <- list.stack(out, data.table = TRUE)
out_dt[, net_profit := closed_profit + position_profit - fee]  # 净利润
#View(out_dt)

trades_dt <- list.stack(trades, data.table = TRUE)
trades_dt[, profit := side * (leave_price - enter_price) - commission]
#View(trades_dt)
```

画一个交易的图像看一下

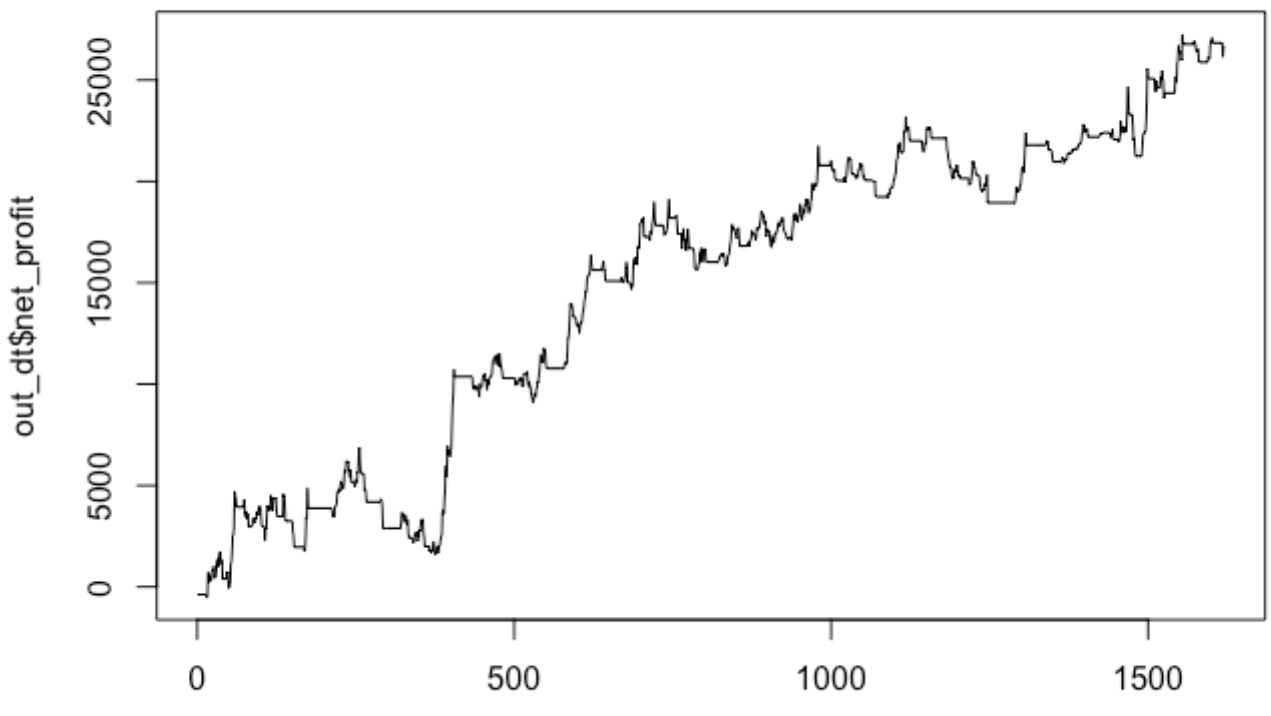
```
# 找出利润最大的一次交易
trades_dt[which.max(trades_dt$profit), .(enter_date, leave_date)]
```

```
##    enter_date leave_date
## 1:   20110819   20111021
```

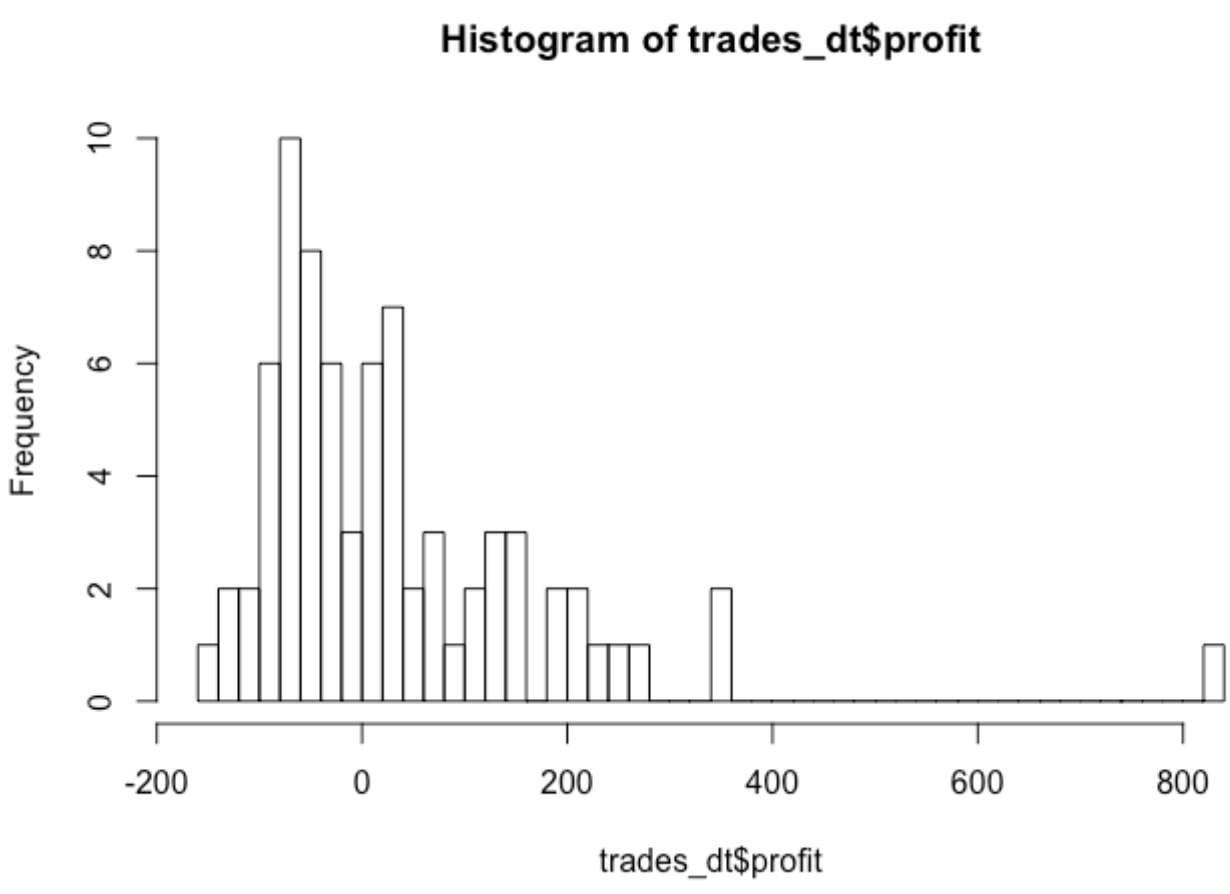
```
# 画图看一下这次交易的情况
et_day <- 20110701
lv_day <- 20111030
# et_day <- 20160101
# lv_day <- 20161231

dtplot <- copy(out_dt[date >= et_day & date <= lv_day])
dtplot[, time_index := 1:.N]
pe <- ggplot(dtplot, aes(x = time_index)) +
  geom_linerange(aes(ymin = low, ymax = high)) +
  geom_point(aes(y = N_high), color = "red") +
  geom_point(aes(y = N_low), color = "green") +
  geom_line(aes(y = stop_price), color = "blue", size = .9, na.rm = TRUE)
pe
```




```
# 资金曲线
plot(out_dt$net_profit, type = "l")
```




```
# 每笔交易利润分布：符合趋势追踪类策略的特点
hist(trades_dt$profit, breaks = 50)
```



计算策略评价指标，自动生成报告

评价一个策略好不好，有很多指标:

- 最大回撤 & 最大回撤周期：买入产品后最糟糕的情况，是非常重要的一个风险指标。越低越好。
- 年化收益率
- Sharpe ratio: 单位风险所获得的超额回报率。越高越好
- 最大连续亏损次数，最大单周/单月亏损幅度。同理，估计策略最坏的情况。
- 胜率，盈亏比
- ...

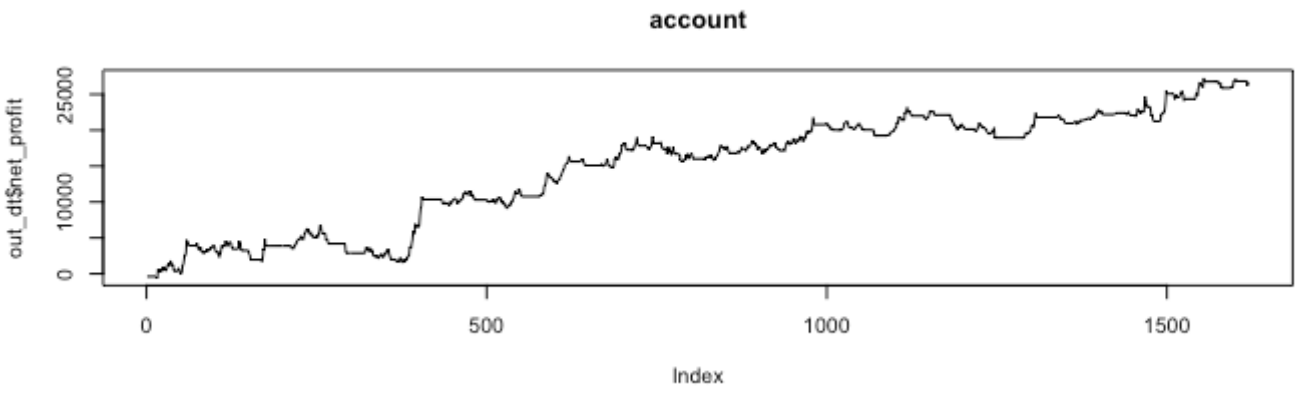
这里算几个最常见的指标:

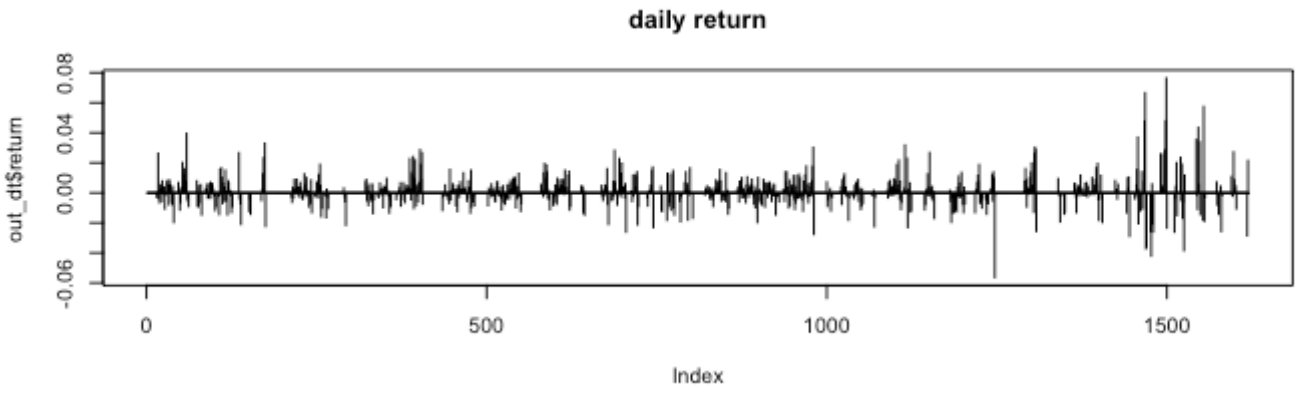
```
# 计算每日收益率：（不加杠杆，return = 当日净收益/合约市值的60日移动平均）
out_dt[, profit := c(0, diff(net_profit))]
out_dt[, market_value_MA60 := zoo::na.fill(TTR::SMA(market_value, n = 60), fill = "extend")]
out_dt[, return := profit / market_value_MA60]

calc_annual_return <- function(x){  # 根据日度收益率计算年化收益率
  prod(1 + x) ^(250 / length(x)) - 1
}
calc_sharpe_ratio <- function(x){   # 根据日度收益率计算 sharpe ratio
  calc_annual_return(x) / (sqrt(250) * sd(x))
}
# 年化收益率
annual_return <- calc_annual_return(out_dt$return)
# 夏普比
sharpe_ratio <- calc_sharpe_ratio(out_dt$return)
# 最大回撤
out_dt[, cum_profit := cumsum(profit)]
out_dt[, cummax_cum_profit := cummax(cum_profit)]
out_dt[, drawdown := cum_profit - cummax_cum_profit]
maxdrawdown <- min(out_dt$drawdown)
maxdrawdown_idx <- which.min(out_dt$drawdown)
maxdrawdown_per <-
  out_dt[maxdrawdown_idx, drawdown] /
  out_dt[maxdrawdown_idx, cummax_cum_profit]
# 胜率
win_prob <- mean(trades_dt$profit > 0)
# 平均盈亏比
win_loss_ratio <-
  (sum(trades_dt[profit > 0, profit]) / trades_dt[profit > 0, .N]) /
  (sum(trades_dt[profit <= 0, -profit]) / trades_dt[profit <= 0, .N])


indicators <- sprintf("
年化收益率: %s
Sharpe_Ratio: %f
```

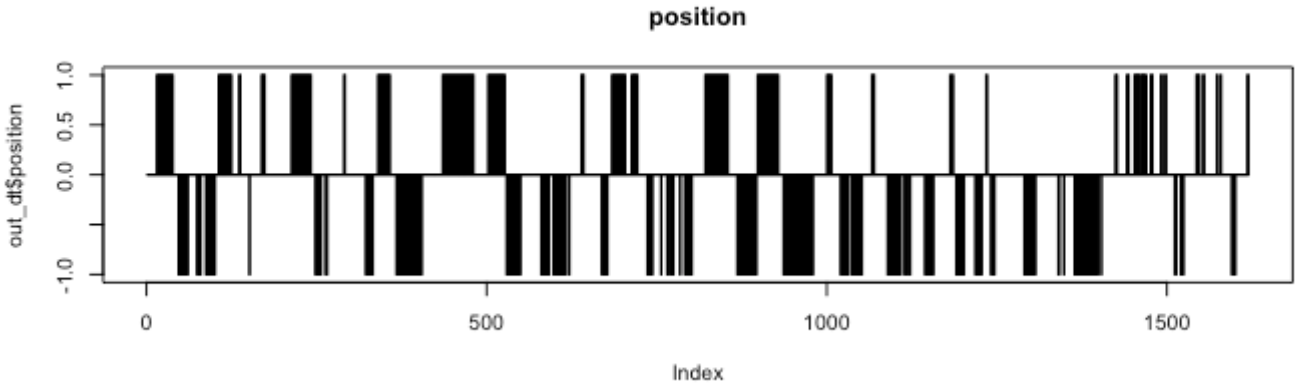
```
最大资金回撤：%f（百分比：%s）
胜率：%s
平均盈亏比：%f
",
scales::percent(annual_return),
sharpe_ratio,
maxdrawdown, scales::percent(maxdrawdown_per),
scales::percent(win_prob),
win_loss_ratio
)
cat(indicators)
```

```
##
## 年化收益率：11.9%
## Sharpe_Ratio: 0.919974
## 最大资金回撤：-5254.200000（百分比：-72.5%）
## 胜率：49.3%
## 平均盈亏比：2.005567
```

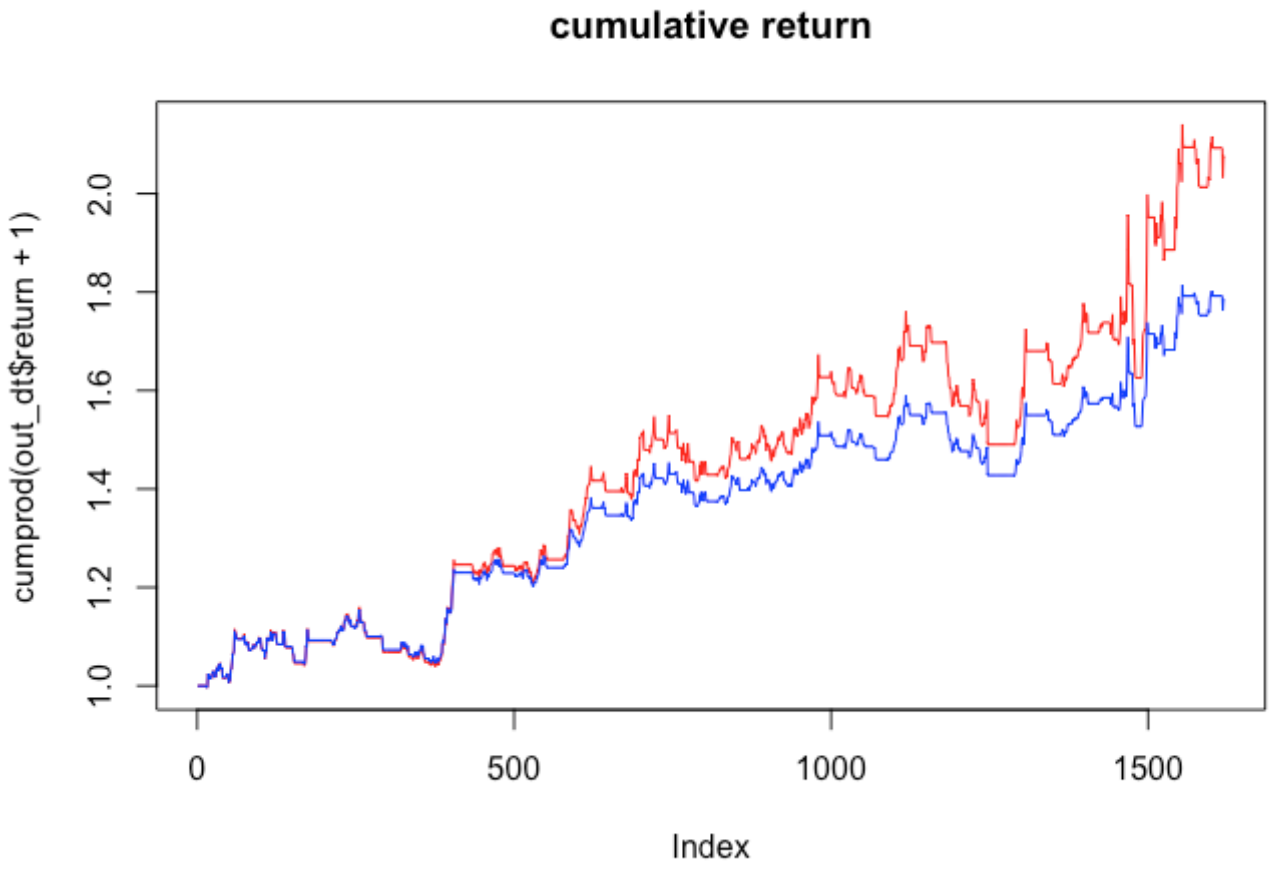
```
par(mfrow = c(3, 1))
plot(out_dt$net_profit, type = "l", main = "account")  # 账户资金曲线
plot(out_dt$return, type = "h", main = "daily return")  # 日度收益率
plot(out_dt$position, type = "h", main = "position")  # 每日postion
```








```
par(mfrow = c(1, 1))
plot(cumprod(out_dt$return + 1), type = "l", col = "red", main = "cumulative return")  # 累计收益率 乘
lines(cumsum(out_dt$return) + 1, type = "l", col = "blue")  # 累计收益率 加
```



不写for循环，快速得到净值曲线的方法

经@刘天宇提醒，有一种快速得到净值曲线的办法：向量化生成每日的买卖信号（1， 0， -1），然后和每日收益率相乘，再cumprod，直接就能得到净值曲线。方法简单直观且速度非常快。

参数优化

策略往往带有参数，我们经常要对一个策略做参数寻优。最简单的方法就是用不同的参数跑策略，比较各个参数下的结果。本例中只有两个参数，评价标准就简单粗暴的采用最终收益。

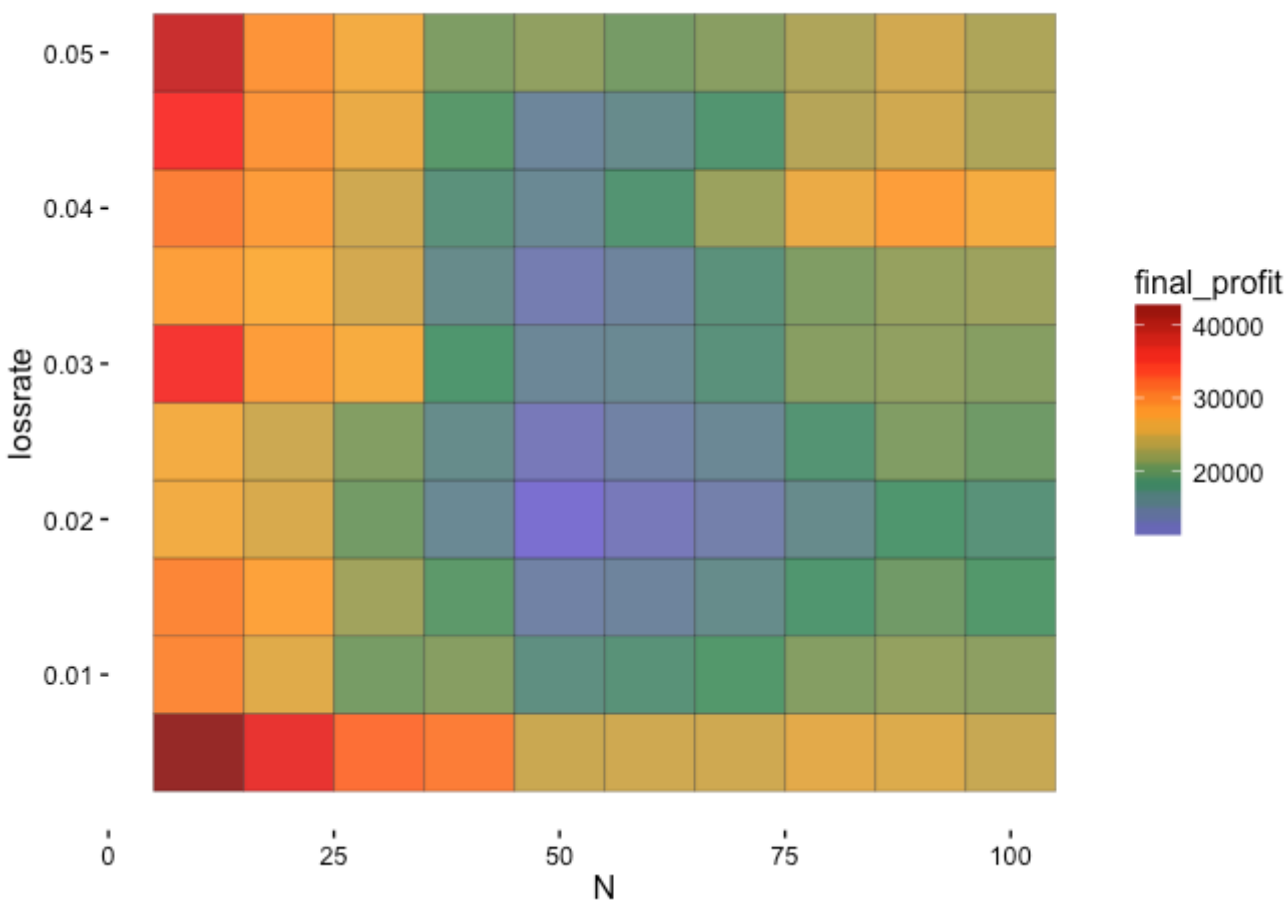
参数优化计算量往往较大，需要了解一些并行计算的知识。

```
rm(list = ls())
# 全局变量
load("data/rb_dominant_daily.rda")
fee.rate <- 8e-5
slippage <- 2 * 1
vm <- 10
gvars <- ls()  # global variables. 收集全局变量，后面并行计算要用。
source("RunBackTest.R")  # 加载 RunBackTest() 函数

# 参数集合
N_set <- seq(10, 100, 10)  # 参数N：10, 20, 30, ..., 100
lossrate_set <- seq(.5, 5, .5) / 100  # 移动止损参数：1%, 1.5%, 2%, ... 5%
para_lst <-
  expand.grid(N = N_set, lossrate = lossrate_set) %>>%
  list.parse()   # 参数集合

#str(para_lst)
#result <- RunBackTest(para = para_lst[[1]])
#result <- RunBackTest(para = para_lst[[100]])

library(foreach)
library(doParallel)

file.create("log.txt")  # 日志文件
detectCores()  # 看一下你的电脑有几个核
cl <- makeCluster(2, outfile = "log.txt")  # 使用2个核，并指定outfile
clusterExport(cl, varlist = gvars) # 将全局变量分发到各个核，在每个核上加载必要的包
clusterEvalQ(cl, expr = {
  library(data.table)
  library(rlist)
  library(pipeR)
})
# 开始并行计算
results <- foreach(x = para_lst) %dopar%
{
  tryCatch(
```

```
    {
      cat(sprintf("Start Running [N = %s, lossrate = %s] ... ", x$N, x$lossrate),
          file = "log.txt", append = TRUE)
      rst <- RunBackTest(x)  # 回测一个参数
      cat(sprintf("final_profit: %s \n", rst$final_profit), file = "log.txt", append = TRUE)
      return(rst)
    },
    error = function(e) {
      error_into <- sprintf("When running [ N = %s, lossrate = %s ] Raised Error : %s \n", x$N, x$K, x$ATR_N, e)
        cat(error_into, file = "log.txt", append = TRUE)
        return(error_into)
    }
  ) # End tryCatch
}
stopCluster(cl)  # 计算完毕，停掉cluster

results_dt <- rbindlist(results)
head(results_dt)

save(results_dt, file = "data/result_dt.rda")
```

画出二维热力图：

```
load("data/result_dt.rda")

# 可视化
p <- ggplot(data = results_dt, aes(x = N, y = lossrate, fill = final_profit)) +
  geom_tile(color = "black", alpha = .9) +
  scale_fill_gradientn(colors = c("slateblue", "seagreen", "orange", "red", "darkred")) +
  theme_classic()
p
```



注意点

关于并行计算

R中常见的并行计算包： `parallel, snow, doparallel, foreach, ...`

我的几点经验：

- 大规模的计算一定要有容错机制，你不可能考虑到所有的情况。
- 算好一个，保存一个，最好再花点时间写日志文件，一是防止停电之类的突发状况，二是方便程序运行时实时监控，早发现潜在的问题。
- 负载均衡问题。最好用自动安排负载均衡的包或函数，比如 `foreach`，`parallel::parLapplyLB()`
- Windows 与 Mac/Linux 环境下有一点区别。

关于性能

- 因为计算量较大，所以需要我们在写策略的时候就注意性能问题，尽量使用高性能的包，例如`data.table`
- 性能分析/可视化包`profvis`，Rstudio中也集成了profiling功能。
- 有时候还要用到Rcpp，甚至是直接在cpp中写。

招聘广告一则

深圳凌云至善科技有限公司
【IT开发/运维工程师】（全职，若干）
工作职责：
1）软件系统维护与测试；
2）数据库管理；
3）服务器与硬件设施运维。
职位资格要求：
1）重点高校毕业，本科及以上学历，计算机相关专业；
2）熟练掌握 C# 和 C/C++ 并有相关项目开发和测试经验；
3）熟悉 MongoDB 和 Redis 数据库的管理和维护，并具有较好的数据处理能力；
4）具有 Linux 本地和远程服务器管理和维护经验；
5）有较强的责任心、严谨的工作态度和较好的沟通能力。
【量化研究员】（全职，若干）
工作职责：
1）市场及交易数据处理、建模与分析；
2）策略开发、回测、评估与分析；
3）组合绩效分析与优化；
4）其他专题研究。
职位资格要求：
1）重点高校毕业，硕士及以上学历（技术性专业、数学、物理、计算机、金融工程、统计学等）；
2）熟悉中国证券、期货、期权等市场规则，通过证券从业或者基金从业资格考试，具有国内外对冲基金或定量分析经验者优先；
3）具备良好的编程与数据处理能力，能够熟练使用R/Python/Matlab等数据分析工具中的一种，具备C++/C#等通用编程经验者优先；
4）具备操作关系型数据库和MongoDB/Redis数据库经验者优先；
5）具备学术研究、数学/统计建模经验者优先；
6）具备快速学习的能力，重视细节的品质，以及较好的表达能力。
欢迎加入凌云至善，待遇从优，根据条件税前工资区间在10000-30000之间，具体面谈，工作地点深圳。
由于精力有限，请投简历之前看清楚岗位要求，非诚勿扰。
有意者请将简历发送至hr@lyzsfund.com。