COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

Introduction to Trading Systems

Guy Yollin

Applied Mathematics
University of Washington

Outline

Outline

Lecture references

- TradeAnalytics project page on R-forge:
  `http://r-forge.r-project.org/projects/blotter/`
  - documents and demos for:
    - blotter package (specifically the demo script `longtrend.R`)[†]
    - quantstrat package

- R-SIG-FINANCE:
  `https://stat.ethz.ch/mailman/listinfo/r-sig-finance`

[†] demos are located in the directory: `.../R-3.1.0/library/blotter/demo`

Quantitative analysis package hierarchy

Application Area	R Package
Performance metrics and graphs	**PerformanceAnalytics** - Tools for performance and risk analysis
Portfolio optimization and quantitative trading strategies	**PortfolioAnalytics** - Portfolio analysis and optimization
	quantstrat – Rules-based trading system development
	blotter – Trading system accounting infrastructure
Data access and financial charting	**quantmod** - Quantitative financial modeling framework
	TTR - Technical trading rules
Time series objects	**xts** - Extensible time series
	zoo - Ordered observation

About blotter and quantstrat

- Provides support for multi-asset class and multi-currency portfolios for backtesting and other financial research. **Still in heavy development**.
- The software is in an beta stage
  - some things are not completely implemented (or documented)
  - some things invariably have errors
  - some implementations will change in the future
- Software has been in development for a number of years
  - blotter: Dec-2008
  - quantstrat: Feb-2010
- Software is used everyday by working professions in asset management

The blotter package

Description

Transaction infrastructure for defining instruments, transactions, portfolios and accounts for trading systems and simulation. Provides portfolio support for multi-asset class and multi-currency portfolios. Still in heavy development.

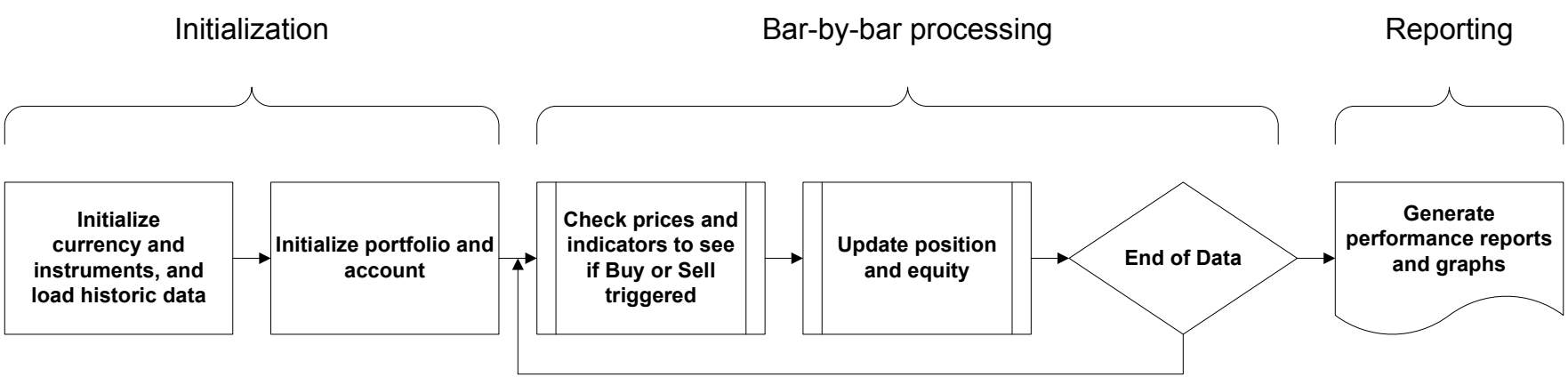
Key features

- supports portfolios of multiple assets
- supports accounts of multiple portfolios
- supports P&L calculation and roll-up across instruments and portfolios (i.e. `blotter` does low-level trading system accounting)

Authors

- Peter Carl
- Brian Peterson

Basic strategy backtesting workflow for blotter



Key blotter functions

Initialization	
initPortf	initializes a portfolio object
initAcct	initializes an account object
Processing	
addTxn	add transactions to a portfolio
updatePortf	calculate P&L for each symbol for each period
updateAcct	calculate equity from portfolio data
updateEndEq	update ending equity for an account
getEndEq	retrieves the most recent value of the capital account
getPosQty	gets position at Date

Key blotter functions

Analysis	
chart.Posn	chart market data, position size, and cumulative P&L
chart.ME	chart Maximum Adverse/Favorable Excursion
PortfReturns	calculate portfolio instrument returns
getAccount	get an account object from the .blotter environment
getPortfolio	get a portfolio object from the .blotter environment
getTxns	retrieve transactions from a portfolio
tradeStats	calculate trade statistics
perTradeStats	calculate flat to flat per-trade statistics

Outline

Loading the blotter package

```
library(blotter)
search()
```

```
##  [1] ".GlobalEnv"                    "package:blotter"
##  [3] "package:PerformanceAnalytics" "package:FinancialInstrument"
##  [5] "package:quantmod"              "package:methods"
##  [7] "package:TTR"                   "package:Defaults"
##  [9] "package:xts"                   "package:zoo"
## [11] "package:stringr"               "package:tools"
## [13] "package:knitr"                 "package:stats"
## [15] "package:graphics"              "package:grDevices"
## [17] "package:utils"                 "package:datasets"
## [19] "Autoloads"                     "package:base"
```

Loading `blotter` causes these other libraries to be loaded automatically:

- PerformanceAnalytics
- FinancialInstrument
- quantmod
- TTR
- Defaults
- xts
- zoo

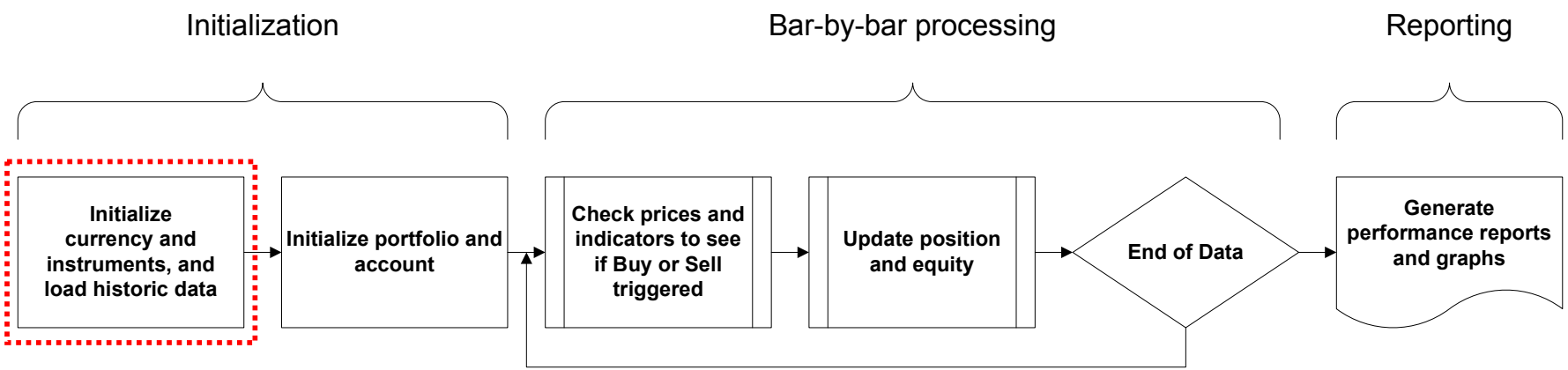
The .blotter and .instrument environment

The `blotter` package creates an environment named `.blotter` for private storage of portfolio and account objects

```
ls()

## [1] "filename"

ls(all=T)

## [1] ".blotter" "filename"
```

The `FinancialInstrument` package will also create an environment called `.instrument` for private storage of defined instruments (e.g. currency, stock, future, etc.)

Blotter backtesting: Step 1



The FinancialInstrument package

The FinancialInstrument package provides an infrastructure for defining meta-data and relationships for financial instruments.

Key functions:

- currency
- stock
- bond
- option
- future
- fund
- exchange_rate

Author:

- Brian Peterson
- Peter Carl

Instrument class constructors

```
args(currency)

## function (primary_id, identifiers = NULL, assign_i = TRUE, ...)
## NULL

args(stock)

## function (primary_id, currency = NULL, multiplier = 1, tick_size = 0.01,
##     identifiers = NULL, assign_i = TRUE, overwrite = TRUE, ...)
## NULL
```

Main arguments:

primary_id	character string providing a unique ID for the instrument
currency	string describing the currency ID
multiplier	numeric multiplier (multiplier $\times$ price $=$ notional values)
tick_size	tick increment of the instrument price

- Note: all `currency` instruments must be defined before instruments of other types can be defined

Initialize a currency and a stock instrument

```
currency("USD")

## [1] "USD"

stock("SPY",currency="USD",multiplier=1)

## [1] "SPY"

ls(all=T)

## [1] ".blotter" "filename"

ls(envir=FinancialInstrument:::.instrument)

## [1] "SPY" "USD"
```

- Instrument objects are stored in the .instrument environment of FinancialInstrument

Initialize a currency and a stock instrument

```
get("USD",envir=FinancialInstrument:::.instrument)

## primary_id :"USD"
## currency   :"USD"
## multiplier :1
## tick_size  :0.01
## identifiers: list()
## type       :"currency"

get("SPY",envir=FinancialInstrument:::.instrument)

## primary_id :"SPY"
## currency   :"USD"
## multiplier :1
## tick_size  :0.01
## identifiers: list()
## type       :"stock"
```

Fetch historic data

```
# system settings
initDate <- '1997-12-31'
startDate <- '1998-01-01'
endDate <-  '2014-06-30'
initEq <- 1e6
```

```
Sys.setenv(TZ="UTC")
```

```
getSymbols('SPY', from=startDate, to=endDate, index.class="POSIXct", adjust=T)
```

- Must set timezone
- Must use a POSIX time-date index class

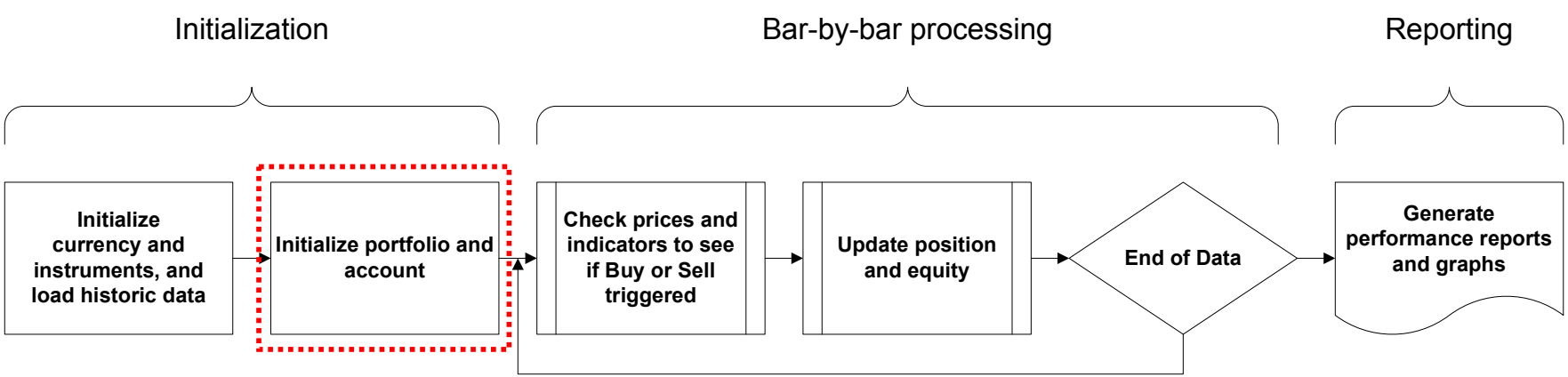
Convert data to monthly

```
SPY=to.monthly(SPY, indexAt='endof', drop.time=FALSE)
SPY$SMA10m <- SMA(Cl(SPY), 10)
tail(SPY)
```

```
##            SPY.Open SPY.High SPY.Low SPY.Close SPY.Volume SPY.Adjusted  SMA10m
## 2014-01-31  182.298  183.249 175.263   176.551 2553999871       176.55 167.526
## 2014-02-28  176.343  185.439 172.122   184.587 2416274469       184.59 170.407
## 2014-03-31  182.962  188.119 182.070   186.118 2573871110       186.12 173.073
## 2014-04-30  186.725  188.795 180.445   187.412 2357143058       187.41 176.082
## 2014-05-30  187.322  191.881 185.123   191.761 1789628418       191.76 178.712
## 2014-06-30  192.030  196.500 191.055   195.720 1679907858       195.72 182.235
```

- Note conversion from daily data to monthly data using the last trading day of the month (`xts` functionality)
- In `to.monthly`, you must use `'endof'` and you must set `drop.time=FALSE`

Blotter backtesting: Step 2



The initPortf function

The initPortf function constructs and initializes a portfolio object, which is used to contain transactions, positions, and aggregate level values.

```
args(initPortf)

## function (name = "default", symbols, initPosQty = 0, initDate = "1950-01-01",
##     currency = "USD", ...)
## NULL
```

Main arguments:

name	name for the resulting portfolio object
symbols	list of symbols to be included in the portfolio
initPosQty	initial position quantity
initDate	date for initial account equity and position (prior to the first close price)
currency	currency identifier

The initAcct function

The initAcct function constructs the data container used to store calculated account values such as aggregated P&L, equity, etc.

```
args(initAcct)

## function (name = "default", portfolios, initDate = "1950-01-01",
##     initEq = 0, currency = "USD", ...)
## NULL
```

Main arguments:

name	name for the resulting account object
portfolios	vector of strings naming portfolios included in this account
initDate	date for initial account equity and position (prior to the first close price)
initEq	initial account equity
currency	currency identifier

Initialize portfolio and account

```
b.strategy <- "bFaber"
initPortf(b.strategy, 'SPY', initDate=initDate)

## [1] "bFaber"

initAcct(b.strategy, portfolios=b.strategy, initDate=initDate, initEq=initEq)

## [1] "bFaber"

initDate

## [1] "1997-12-31"

first(SPY)

##            SPY.Open SPY.High SPY.Low SPY.Close SPY.Volume SPY.Adjusted SMA10m
## 1998-01-30  72.8349   74.519 68.0446   73.5834  139725624        73.59     NA
```

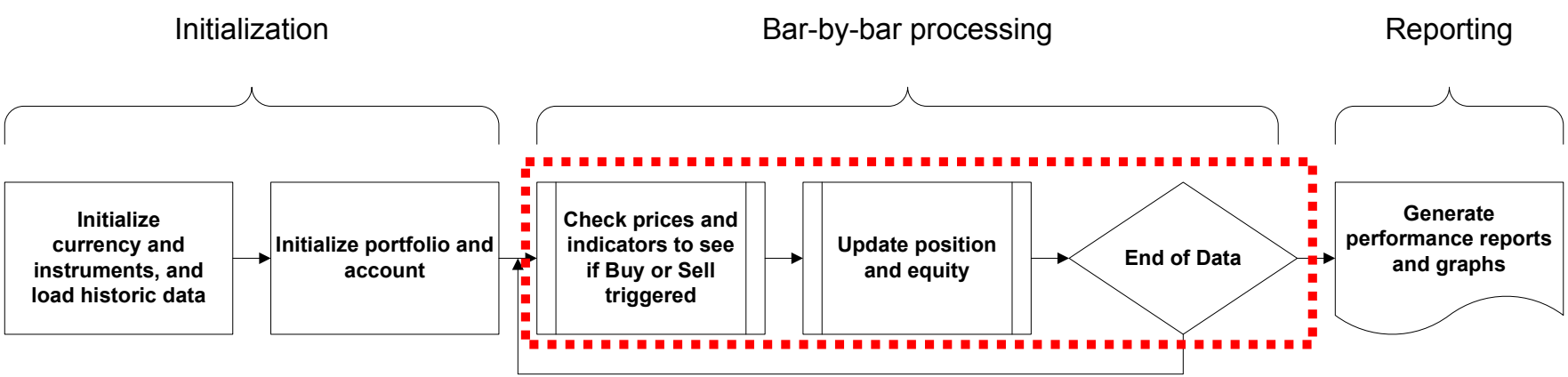
- Note that `initDate` is prior to the start of the data

The .blotter and .instrument environment

```
ls()

## [1] "b.strategy" "endDate"    "filename"   "initDate"   "initEq"     "SPY"
## [7] "startDate"

ls(.blotter)

## [1] "account.bFaber"   "portfolio.bFaber"

ls(envir=FinancialInstrument:::.instrument)

## [1] "SPY" "USD"
```

- various objects (including the historic price xts object) stored in the global environment
- portfolio and account objects stored in `.blotter` environment
- currency and trading instrument objects stored in the `.instrument` environment

Blotter backtesting: Step 3



Faber tactical asset allocation system

Buy-Sell rules:

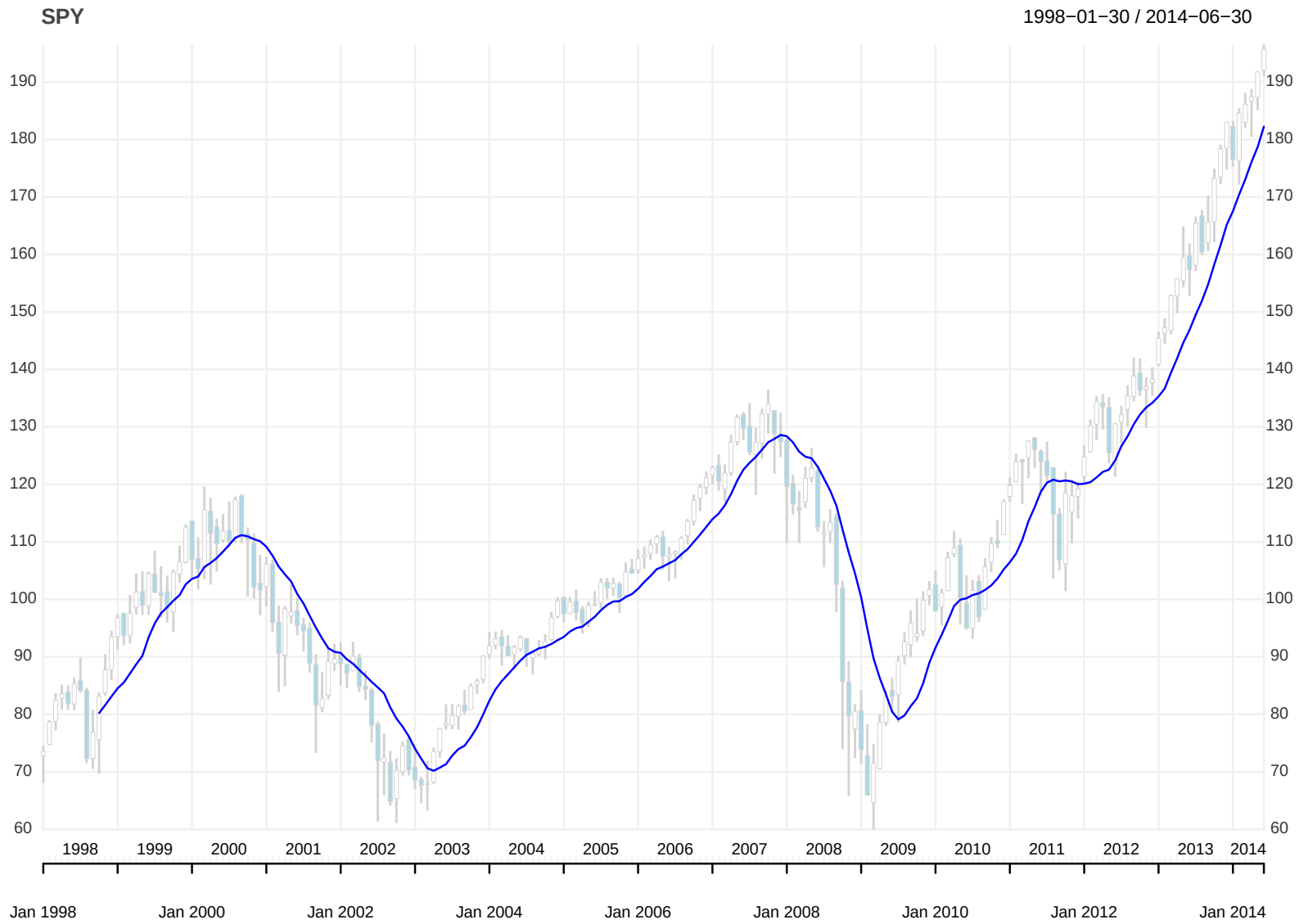
- buy when monthly price $>$ 10-month SMA
- sell and move to cash when monthly price $<$ 10-month SMA

Notes:

- all entry and exit prices are on the day of the signal at the close
- all data series are total return series including dividends, updated monthly
- commissions and slippage are excluded

A Quantitative Approach to Tactical Asset Allocation by Mebane T. Faber, The Journal of Wealth Management, Spring 2007

Monthly SPY and 10-month SMA



Plot monthly SPY and 10-month SMA

```
# create custom theme
myTheme<-chart_theme()
myTheme$col$dn.col<-'lightblue'
myTheme$col$dn.border <- 'lightgray'
myTheme$col$up.border <- 'lightgray'

# plot OHLC series
chart_Series(
  x=SPY,
  theme=myTheme,
  name="SPY",
  TA="add_SMA(n=10,col=4)"
  )
```

Apply trading logic

```
for( i in 1:nrow(SPY) )
{
  # update values for this date
  CurrentDate <- time(SPY)[i]
  equity = getEndEq(b.strategy, CurrentDate)
  ClosePrice <- as.numeric(Cl(SPY[i,]))
  Posn <- getPosQty(b.strategy, Symbol='SPY', Date=CurrentDate)
  UnitSize = as.numeric(trunc(equity/ClosePrice))
  MA <- as.numeric(SPY[i,'SMA10m'])
  # change market position if necessary
  if( !is.na(MA) ) # if the moving average has begun
  {
    if( Posn == 0 ) { # No position, test to go Long
      if( ClosePrice > MA ) {
        # enter long position
        addTxn(b.strategy, Symbol='SPY', TxnDate=CurrentDate,
          TxnPrice=ClosePrice, TxnQty = UnitSize , TxnFees=0) }
    } else { # Have a position, so check exit
      if( ClosePrice < MA ) {
        # exit position
        addTxn(b.strategy, Symbol='SPY', TxnDate=CurrentDate,
          TxnPrice=ClosePrice, TxnQty = -Posn , TxnFees=0)
      } else {
        if( i==nrow(SPY) ) # exit on last day
          addTxn(b.strategy, Symbol='SPY', TxnDate=CurrentDate,
            TxnPrice=ClosePrice, TxnQty = -Posn , TxnFees=0)
      }
    }
  }
  updatePortf(b.strategy,Dates=CurrentDate)
  updateAcct(b.strategy,Dates=CurrentDate)
  updateEndEq(b.strategy,CurrentDate)
} # End dates loop
```

Transactions

```
getTxns(Portfolio=b.strategy, Symbol="SPY")
```

```
##            Txn.Qty  Txn.Price Txn.Fees   Txn.Value Txn.Avg.Cost Net.Txn.Realized.PL
## 1997-12-31       0   0.000000        0        0.00     0.000000               0.000
## 1998-10-30   12030  83.125042        0   999994.25    83.125042               0.000
## 1999-09-30  -12030  98.425927        0 -1184063.91    98.425927          184069.653
## 1999-10-29   11305 104.732831        0  1184004.66   104.732831               0.000
## 2000-09-29  -11305 110.880542        0 -1253504.53   110.880542           69499.870
## 2002-03-28   13916  90.076820        0  1253509.03    90.076820               0.000
## 2002-04-30  -13916  84.838332        0 -1180610.23    84.838332          -72898.796
## 2003-04-30   16058  73.524300        0  1180653.20    73.524300               0.000
## 2004-08-31  -16058  90.615126        0 -1455097.69    90.615126          274444.488
## 2004-09-30   15898  91.524627        0  1455058.52    91.524627               0.000
## 2007-12-31  -15898 127.365627        0 -2024858.74   127.365627          569800.219
## 2009-06-30   24372  83.080687        0  2024842.51    83.080687               0.000
## 2010-06-30  -24372  95.050379        0 -2316567.83    95.050379          291725.326
## 2010-07-30   22814 101.542388        0  2316588.03   101.542388               0.000
## 2010-08-31  -22814  96.974960        0 -2212386.74    96.974960         -104201.294
## 2010-09-30   20939 105.659523        0  2212404.75   105.659523               0.000
## 2011-08-31  -20939 114.804639        0 -2403894.34   114.804639          191489.585
## 2012-01-31   19265 124.777069        0  2403830.23   124.777069               0.000
## 2014-06-30  -19265 195.720000        0 -3770545.80   195.720000         1366715.571
```

The updatePortf function

The updatePortf function goes through each symbol and calculates the PL for each period prices are available

```
args(updatePortf)

## function (Portfolio, Symbols = NULL, Dates = NULL, Prices = NULL,
##     Interval = Interval, ...)
## NULL
```

Main arguments:

Portfolio	portfolio object containing transactions
Symbols	character vector of symbols
Dates	dates for calculation (must appear in the price stream)
Prices	xts object of prices (with timestamps) to mark the book on
Interval	character string defining interval

The updateAcct function

The updateAcct function performs the equity account calculations from the portfolio data and corresponding close prices

```
args(updateAcct)

## function (name = "default", Dates = NULL)
## NULL
```

Main arguments:

name name of account

Dates dates for calculation

- requires that updatePortf has already been run

The updateEndEq function

The updateEndEq calculates End.Eq and Net.Performance

```
args(updateEndEq)

## function (Account, Dates = NULL)
## NULL
```

Main arguments:

Account name of account

Dates dates for calculation

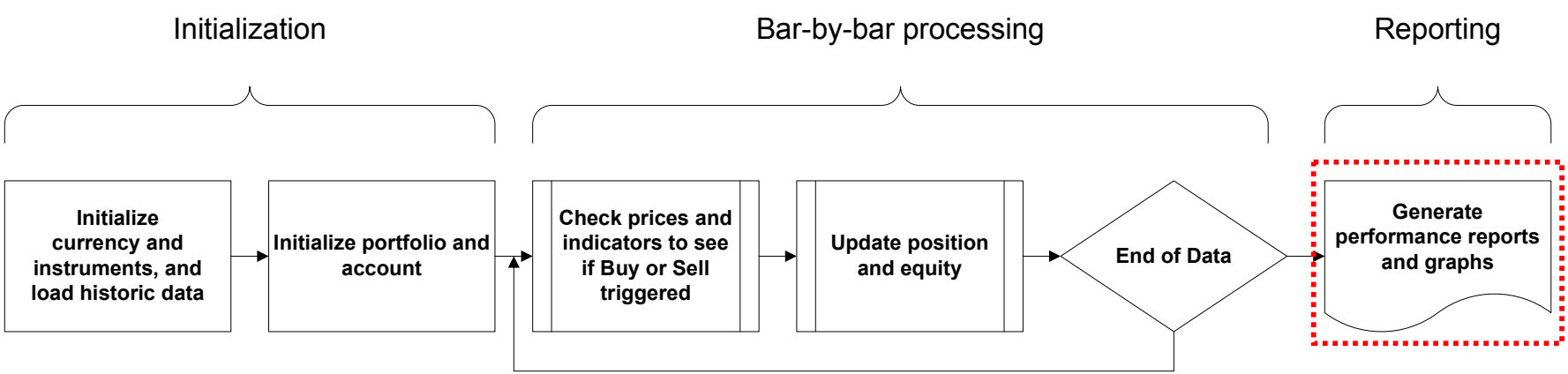
- requires that updateAcct has already been run

Data integrity check

```
checkBlotterUpdate <- function(port.st,account.st,verbose=TRUE)
{
  ok <- TRUE
  p <- getPortfolio(port.st)
  a <- getAccount(account.st)
  syms <- names(p$symbols)
  port.tot <- sum(sapply(syms,FUN = function(x) eval(parse(
    text=paste("sum(p$symbols",x,"posPL.USD$Net.Trading.PL)",sep="$")))))
  port.sum.tot <- sum(p$summary$Net.Trading.PL)
  if( !isTRUE(all.equal(port.tot,port.sum.tot)) ) {
    ok <- FALSE
    if( verbose )
      print("portfolio P&L doesn't match sum of symbols P&L")
  }
  initEq <- as.numeric(first(a$summary$End.Eq))
  endEq <- as.numeric(last(a$summary$End.Eq))
  if( !isTRUE(all.equal(port.tot,endEq-initEq)) ) {
    ok <- FALSE
    if( verbose )
      print("portfolio P&L doesn't match account P&L")
  }
  if( sum(duplicated(index(p$summary))) ) {
    ok <- FALSE
    if( verbose )
      print("duplicate timestamps in portfolio summary")
  }
  if( sum(duplicated(index(a$summary))) ) {
    ok <- FALSE
    if( verbose )
      print("duplicate timestamps in account summary")
  }
  return(ok)
}
checkBlotterUpdate(b.strategy,b.strategy)


## [1] TRUE
```

Outline

Blotter backtesting: Step 4

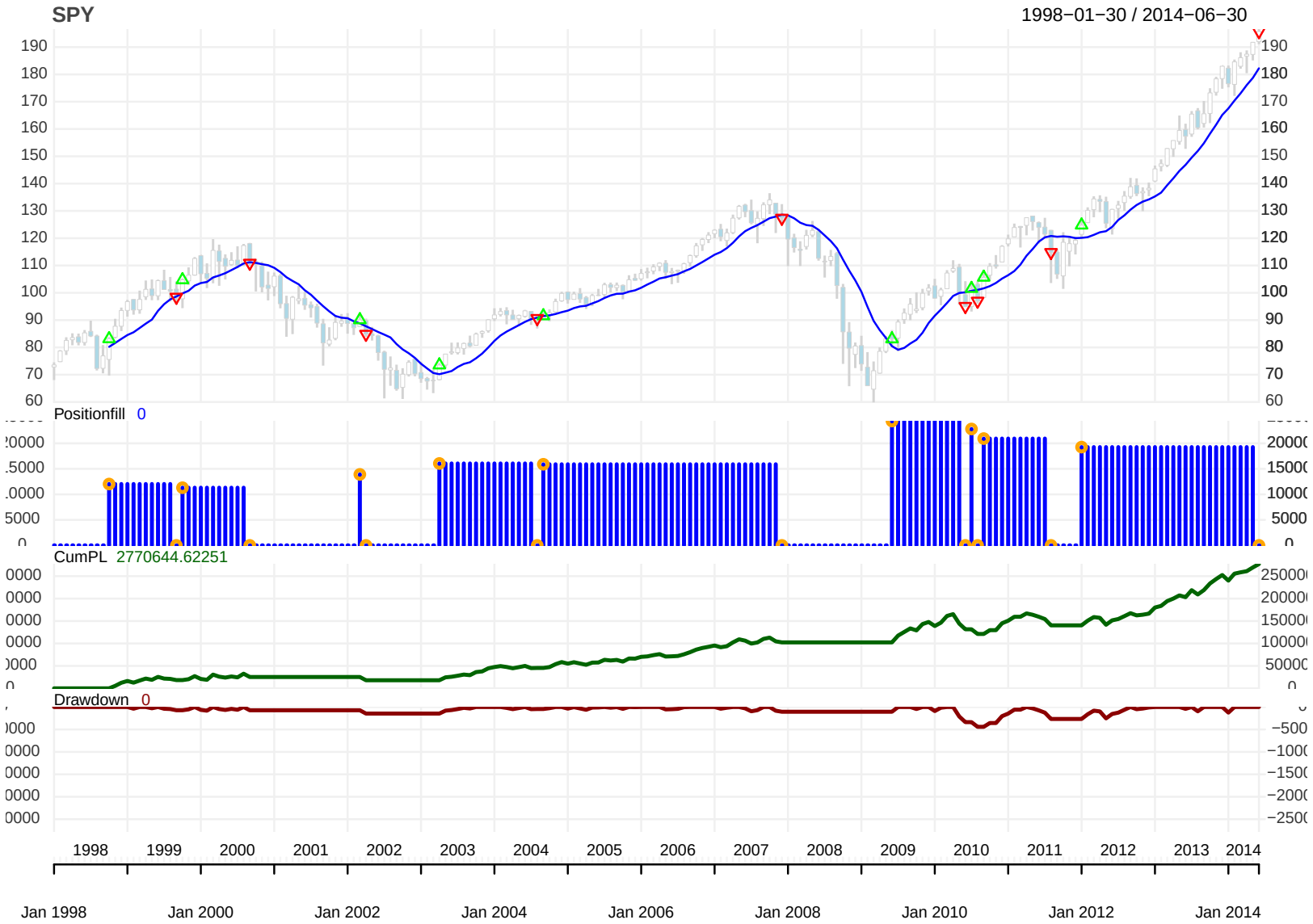


Performance plot

The function `chart.Posn` charts trades against market data, position through time, and cumulative P&L

```
args(chart.Posn)

## function (Portfolio, Symbol, Dates = NULL, ..., TA = NULL)
## NULL

chart.Posn(b.strategy, Symbol = 'SPY',theme=myTheme,
  TA='add_SMA(n=10,col=4, on=1)')
```

Performance plot



Trade statistics

The function `tradeStats` calculates trade-level statistics on a symbol or symbols within a portfolio or portfolios.

```
tstats <- tradeStats(Portfolio=b.strategy)
```

Portfolio	bFaber	**Avg.Win.Trade**	421106.39
Symbol	SPY	**Med.Win.Trade**	274444.49
Num.Txns	18	**Avg.Losing.Trade**	−88550.045
Num.Trades	9	**Med.Losing.Trade**	−88550.045
Net.Trading.PL	2770644.6	**Avg.Daily.PL**	395806.37
Avg.Trade.PL	307849.4	**Med.Daily.PL**	274444.49
Med.Trade.PL	191489.58	**Std.Dev.Daily.PL**	471451.55
Largest.Winner	1366715.6	**Ann.Sharpe**	13.327417
Largest.Loser	−104201.29	**Max.Drawdown**	−441461.8
Gross.Profits	2947744.7	**Profit.To.Max.Draw**	6.2760687
Gross.Losses	−177100.09	**Avg.WinLoss.Ratio**	4.7555751
Std.Dev.Trade.PL	446040.66	**Med.WinLoss.Ratio**	3.0993151
Percent.Positive	77.777778	**Max.Equity**	2770644.6
Percent.Negative	22.222222	**Min.Equity**	0
Profit.Factor	16.644513	**End.Equity**	2770644.6

Compute trade statistics

```
# trade related
tab.trades <- cbind(
  c("Trades","Win Percent","Loss Percent","W/L Ratio"),
  c(tstats[,"Num.Trades"],tstats[,c("Percent.Positive","Percent.Negative")],
  tstats[,"Percent.Positive"]/tstats[,"Percent.Negative"]))

# profit related
tab.profit <- cbind(
  c("Net Profit","Gross Profits","Gross Losses","Profit Factor"),
  c(tstats[,c("Net.Trading.PL","Gross.Profits","Gross.Losses",
    "Profit.Factor")]))

# averages
tab.wins <- cbind(
  c("Avg Trade","Avg Win","Avg Loss","Avg W/L Ratio"),
  c(tstats[,c("Avg.Trade.PL","Avg.Win.Trade","Avg.Losing.Trade",
    "Avg.WinLoss.Ratio")]))

trade.stats.tab <- data.frame(tab.trades,tab.profit,tab.wins)
```

Trade statistics

Trades	9.00	Net Profit	2770644.62	Avg Trade	307849.40
Win Percent	77.78	Gross Profits	2947744.71	Avg Win	421106.39
Loss Percent	22.22	Gross Losses	-177100.09	Avg Loss	-88550.04
W/L Ratio	3.50	Profit Factor	16.64	Avg W/L Ratio	4.76

Per-trade statistics

The function `perTradeStats` calculates flat to flat per-trade statistics.

```
pts <- perTradeStats(Portfolio=b.strategy)
```

Start	End	Init.Pos	Max.Pos	Num.Txns	Max.Notional.Cost	Net.Trading.PL
1998−10−30	1999−09−30	12030	12030	2	999994.25	184069.653
1999−10−29	2000−09−29	11305	11305	2	1184004.66	69499.870
2002−03−28	2002−04−30	13916	13916	2	1253509.03	−72898.796
2003−04−30	2004−08−31	16058	16058	2	1180653.20	274444.488
2004−09−30	2007−12−31	15898	15898	2	1455058.52	569800.219
2009−06−30	2010−06−30	24372	24372	2	2024842.51	291725.326
2010−07−30	2010−08−31	22814	22814	2	2316588.03	−104201.294
2010−09−30	2011−08−31	20939	20939	2	2212404.75	191489.585
2012−01−31	2014−06−30	19265	19265	2	2403830.23	1366715.571

Per-trade statistics

- *Maximum adverse excursion* (MAE) is the largest loss that a trade suffers while it is open
- *Maximum favorable excursion* (MFE) is the peak profit that a trade achieves while it is open

MAE	MFE	Pct.Net.Trading.PL	Pct.MAE	Pct.MFE	tick.Net.Trading.PL	tick.MAE	tick.MFE
0	256403.0442	0.1841	0	0.2564	1530.0886	0	2131.3636
0	142275.3455	0.0587	0	0.1202	614.7711	0	1258.517
−72898.796	0	−0.0582	−0.0582	0	−523.8488	−523.8488	0
0	319232.8431	0.2325	0	0.2704	1709.0826	0	1987.9988
0	675378.6827	0.3916	0	0.4642	3584.1	0	4248.199
0	628985.8349	0.1441	0	0.3106	1196.9692	0	2580.7723
−104201.2937	0	−0.045	−0.045	0	−456.7428	−456.7428	0
0	457742.4122	0.0866	0	0.2069	914.5116	0	2186.0758
0	1366715.5708	0.5686	0	0.5686	7094.2931	0	7094.2931

Outline

The PerformanceAnalytics package

Description

The PerformanceAnalytics package is a collection of econometric functions for performance and risk analysis

Key features

- extensive collection of performance charts
- extensive collection of performance metrics and ratios
- extensive collection of risk metrics
- support for building tables of metrics

Authors

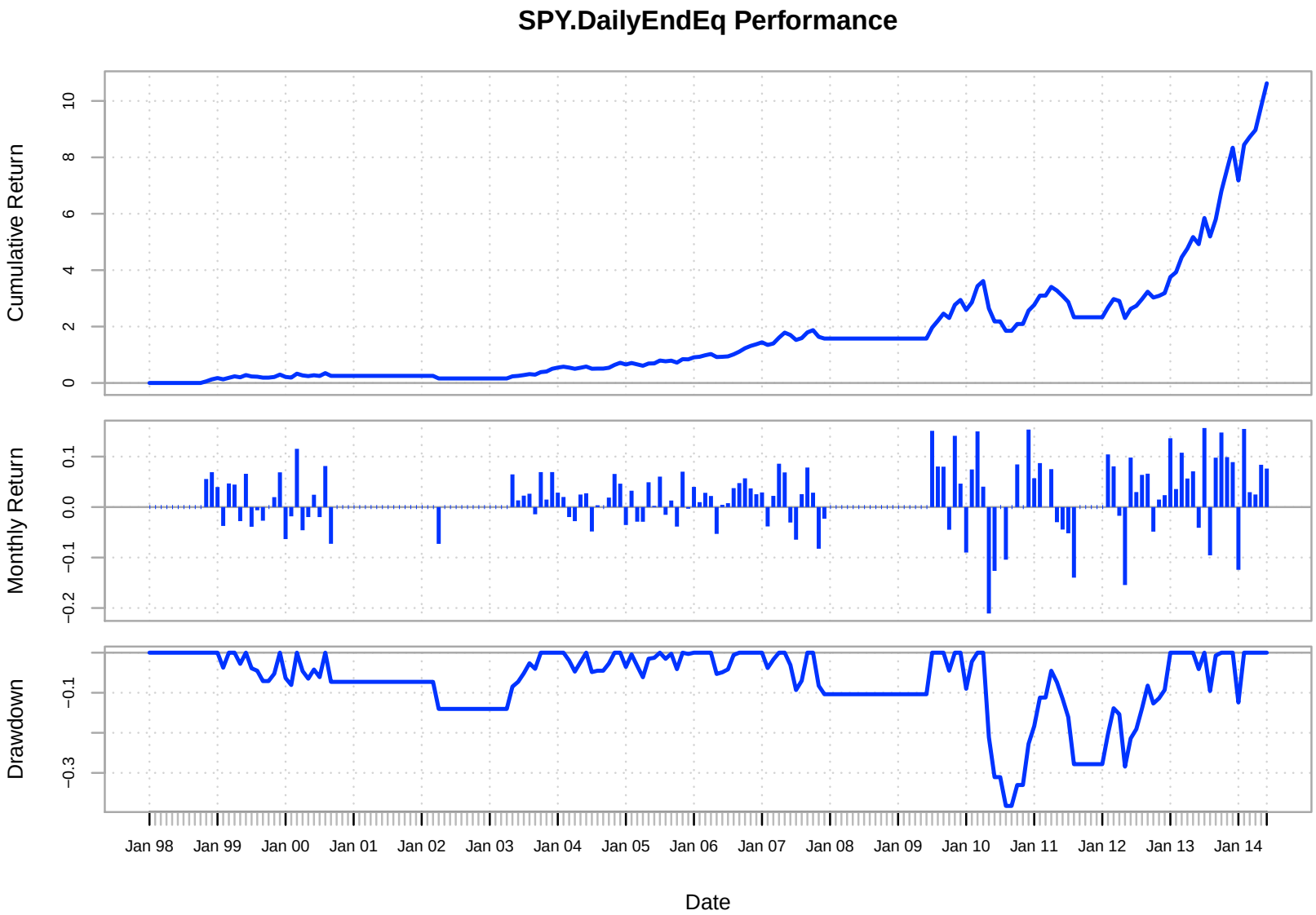
- Peter Carl
- Brian Peterson

Plot cumulative return and drawdown

```
library(PerformanceAnalytics)
rets <- PortfReturns(Account=b.strategy)
rownames(rets) <- NULL
tail(rets)

##             SPY.DailyEndEq
## 2014-01-31   -0.124268590
## 2014-02-28    0.154810793
## 2014-03-31    0.029499476
## 2014-04-30    0.024925064
## 2014-05-30    0.083786563
## 2014-06-30    0.076267786

charts.PerformanceSummary(rets,colorset = bluefocus)
```

- clear `rownames` to avoid timeBased/xtsible error when using certain functions with `table.Arbitrary` (e.g. `Return.annualized`)

Cumulative return and drawdown



The table.Arbitrary function

The table.Arbitrary creates a table of statistics from a passed vector of functions and vector of labels

```
args(table.Arbitrary)

## function (R, metrics = c("mean", "sd"), metricsNames = c("Average Return",
##     "Standard Deviation"), ...)
## NULL
```

Main arguments:

R	time series object (e.g. xts, zoo) of asset returns
metrics	list of functions to apply
metricsNames	column names for each function

Return value:

an data.frame object

Compute performance statistics

```
tab.perf <- table.Arbitrary(rets,
  metrics=c(
    "Return.cumulative",
    "Return.annualized",
    "SharpeRatio.annualized",
    "CalmarRatio"),
  metricsNames=c(
    "Cumulative Return",
    "Annualized Return",
    "Annualized Sharpe Ratio",
    "Calmar Ratio"))
tab.perf
```

```
##                          SPY.DailyEndEq
## Cumulative Return          10.62576116
## Annualized Return           0.16030187
## Annualized Sharpe Ratio     0.83497353
## Calmar Ratio                0.41915213
```

Compute risk statistics

```
tab.risk <- table.Arbitrary(rets,
  metrics=c(
    "StdDev.annualized",
    "maxDrawdown",
    "VaR",
    "ES"),
  metricsNames=c(
    "Annualized StdDev",
    "Max DrawDown",
    "Value-at-Risk",
    "Conditional VaR"))
tab.risk
```

```
##                    SPY.DailyEndEq
## Annualized StdDev     0.191984378
## Max DrawDown          0.382443180
## Value-at-Risk        -0.077230807
## Conditional VaR      -0.120257234
```

Performance and risk statistics

```
performance.stats.tab <- data.frame(
  rownames(tab.perf),tab.perf[,1],
  rownames(tab.risk),tab.risk[,1])
```

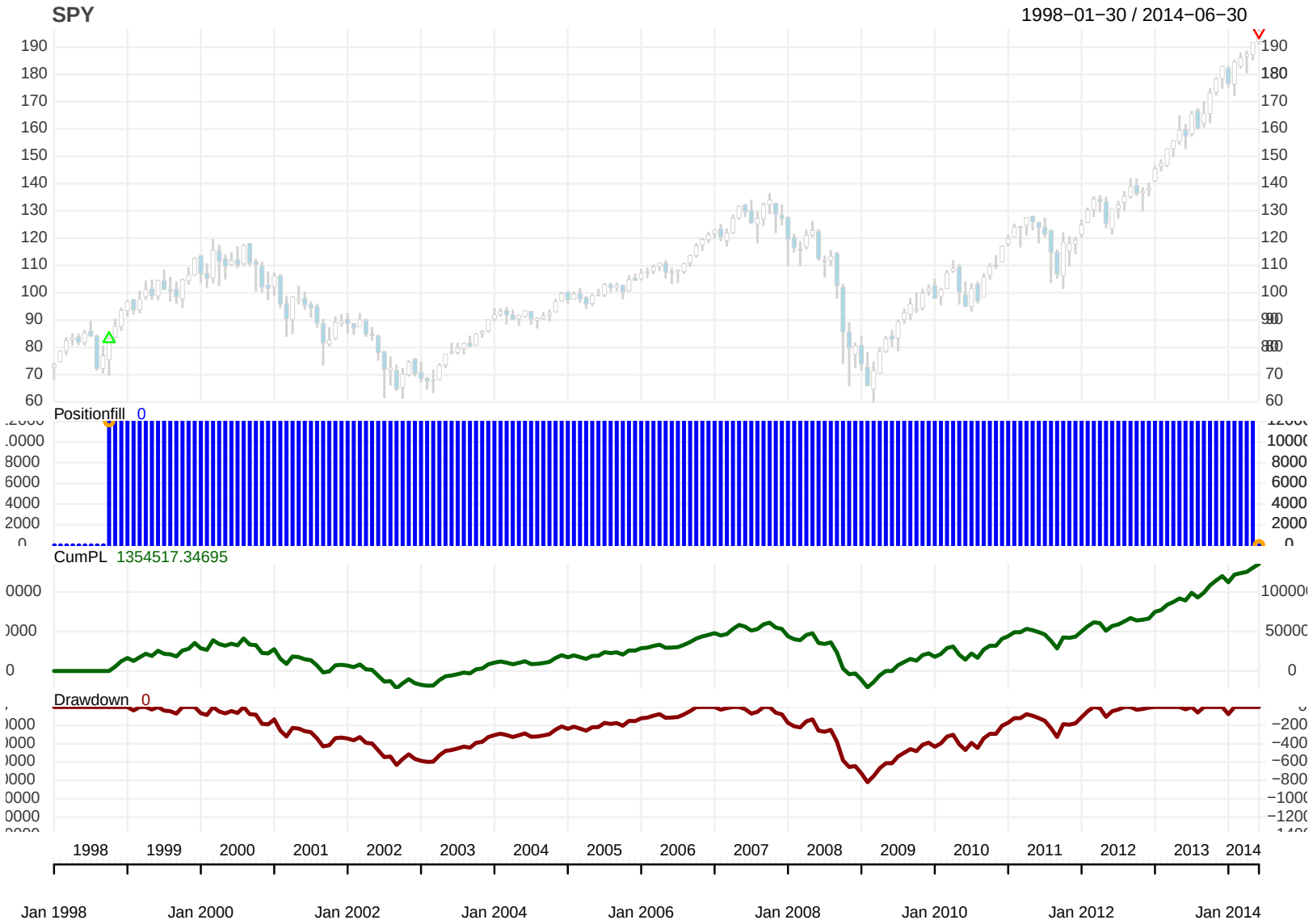
Cumulative Return	10.626	Annualized StdDev	0.192
Annualized Return	0.160	Max DrawDown	0.382
Annualized Sharpe Ratio	0.835	Value-at-Risk	-0.077
Calmar Ratio	0.419	Conditional VaR	-0.120

Outline

Buy and hold strategy

```
# remove objects to allow re-runs
suppressWarnings(try(rm(list=c("account.buyHold","portfolio.buyHold"),pos=.blotter),
# initialize portfolio and account
initPortf("buyHold", 'SPY', initDate=initDate)
initAcct("buyHold", portfolios="buyHold",
  initDate=initDate, initEq=initEq)
# place an entry order
CurrentDate <- time(getTxns(Portfolio=b.strategy, Symbol="SPY"))[2]
equity = getEndEq("buyHold", CurrentDate)
ClosePrice <- as.numeric(Cl(SPY[CurrentDate,]))
UnitSize = as.numeric(trunc(equity/ClosePrice))
addTxn("buyHold", Symbol='SPY', TxnDate=CurrentDate, TxnPrice=ClosePrice,
  TxnQty = UnitSize , TxnFees=0)
# place an exit order
LastDate <- last(time(SPY))
LastPrice <- as.numeric(Cl(SPY[LastDate,]))
addTxn("buyHold", Symbol='SPY', TxnDate=LastDate, TxnPrice=LastPrice,
  TxnQty = -UnitSize , TxnFees=0)
# update portfolio and account
updatePortf(Portfolio="buyHold")
updateAcct(name="buyHold")
updateEndEq(Account="buyHold")
```

Buy and hold performance



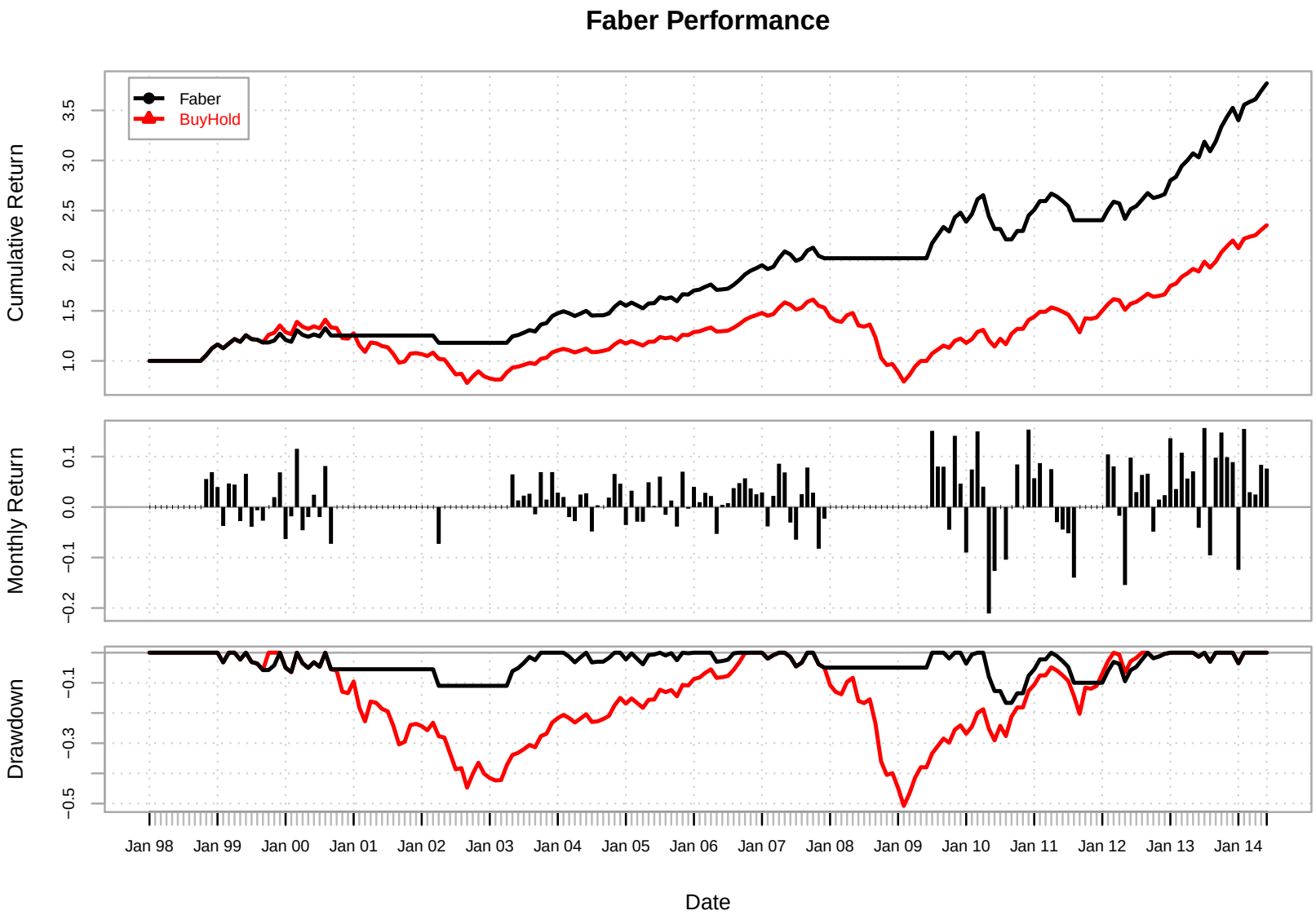
Faber versus buy-and-hold performance

```
rets.bh <- PortfReturns(Account="buyHold")
returns <- cbind(rets,rets.bh)
colnames(returns) <- c("Faber","BuyHold")
returns["2011"]
```

```
##                    Faber         BuyHold
## 2011-01-31  0.05709661682  0.03280349111
## 2011-02-28  0.08710644273  0.05004491647
## 2011-03-31  0.00031166763  0.00017906116
## 2011-04-29  0.07515476880  0.04317836901
## 2011-05-31 -0.02994447819 -0.01720388140
## 2011-06-30 -0.04453965524 -0.02558919015
## 2011-07-29 -0.05192506175 -0.02983229824
## 2011-08-31 -0.13984363220 -0.08034380321
## 2011-09-30  0.00000000000 -0.09587698499
## 2011-10-31  0.00000000000  0.14027840789
## 2011-11-30  0.00000000000 -0.00579287352
## 2011-12-30  0.00000000000  0.01483361108
```

```
charts.PerformanceSummary(returns, geometric=FALSE, wealth.index=TRUE)
```

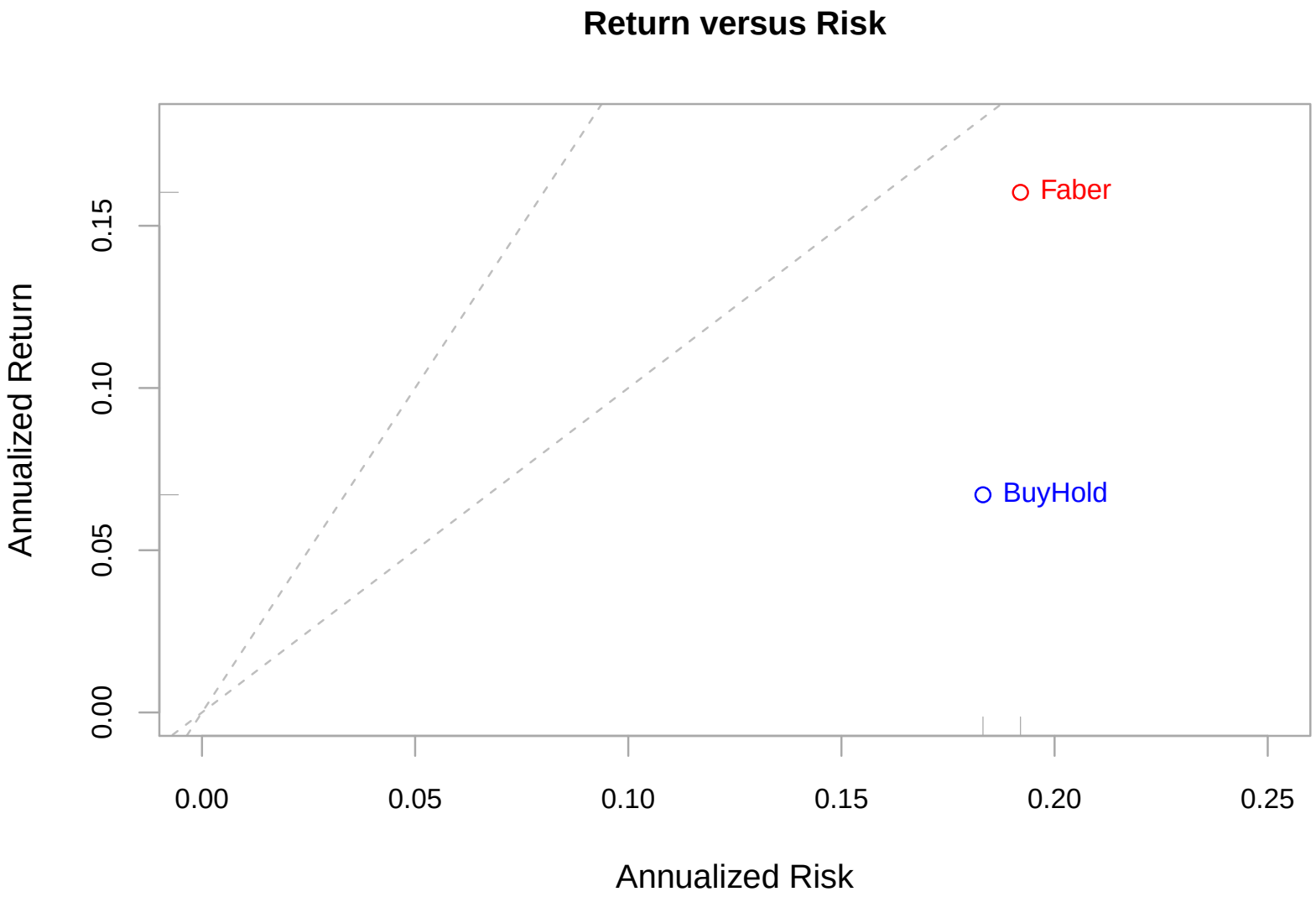
Faber versus buy-and-hold performance



Return and risk comparison

```
table.AnnualizedReturns(returns)

##                                Faber BuyHold
## Annualized Return             0.1603  0.0671
## Annualized Std Dev            0.1920  0.1832
## Annualized Sharpe (Rf=0%) 0.8350  0.3662

chart.RiskReturnScatter(returns, Rf = 0, add.sharpe = c(1, 2), xlim=c(0,0.25),
  main = "Return versus Risk", colorset = c("red","blue"))
```

Return and risk comparison



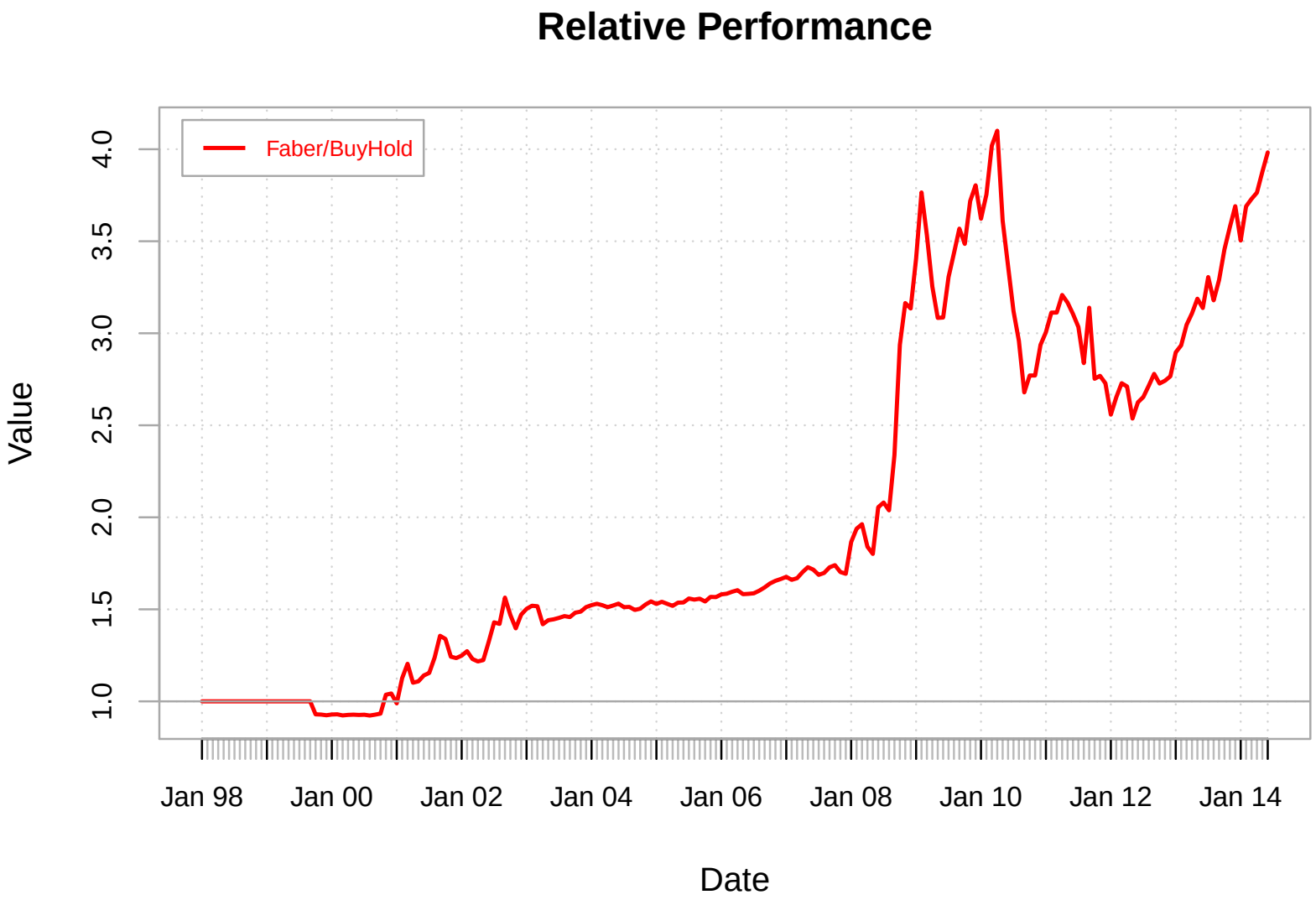
Return stats and relative performance

```
table.Stats(returns)
```

```
##                     Faber  BuyHold
## Observations    198.0000 198.0000
## NAs               0.0000   0.0000
## Minimum          -0.2109  -0.2040
## Quartile 1        0.0000  -0.0217
## Median            0.0000   0.0104
## Arithmetic Mean   0.0140   0.0068
## Geometric Mean    0.0125   0.0054
## Quartile 3        0.0435   0.0454
## Maximum           0.1566   0.1403
## SE Mean           0.0039   0.0038
## LCL Mean (0.95)   0.0062  -0.0006
## UCL Mean (0.95)   0.0218   0.0143
## Variance          0.0031   0.0028
## Stdev             0.0554   0.0529
## Skewness         -0.1562  -0.5721
## Kurtosis          1.9125   0.7583
```

```
chart.RelativePerformance(returns[,1],returns[,2],
  colorset = c("red","blue"), lwd = 2, legend.loc = "topleft")
```

Return stats and relative performance

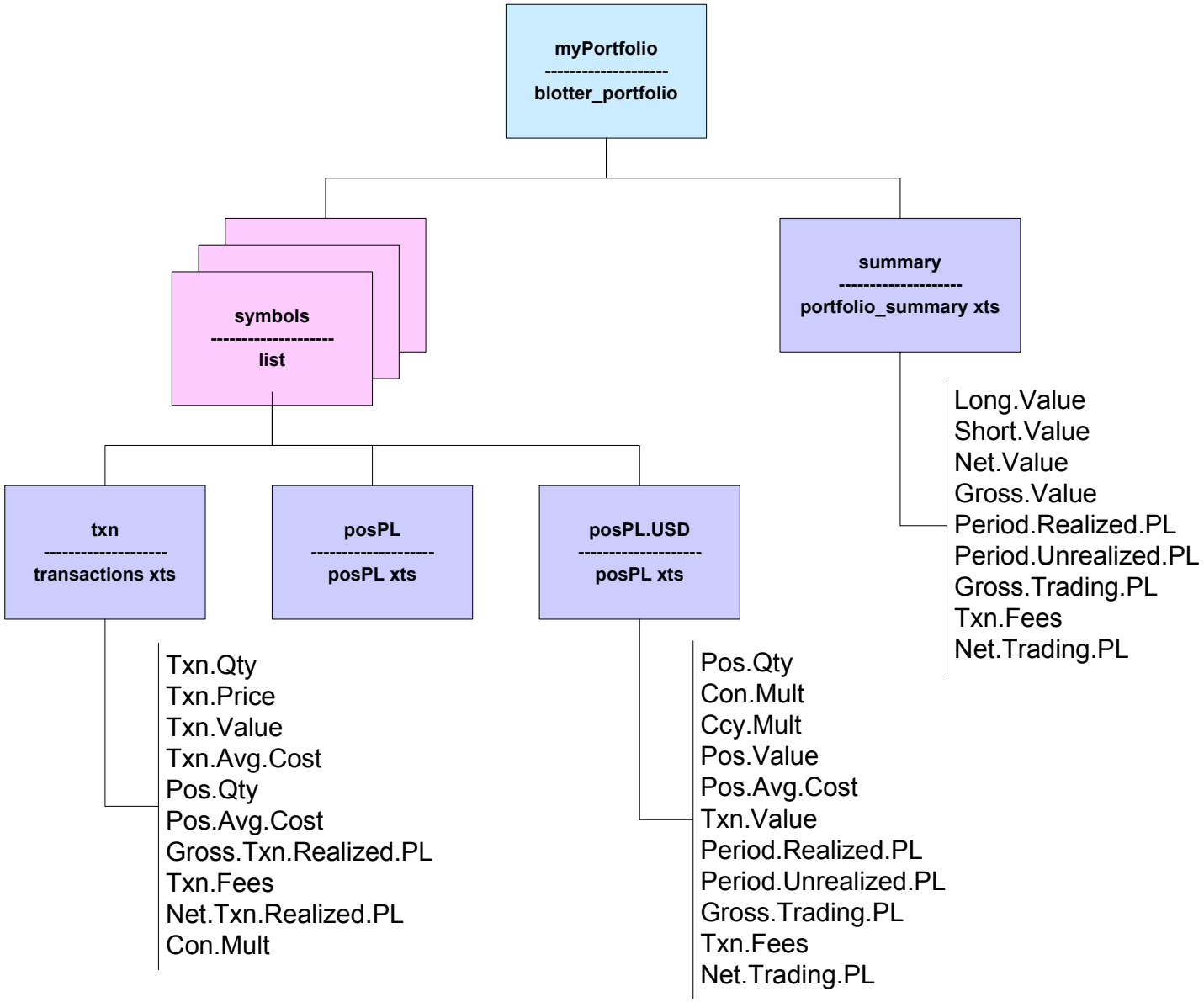


Outline

The blotter *portfolio* object

```
thePortfolio = getPortfolio(b.strategy)
names(thePortfolio)

## [1] "summary" "symbols"

names(thePortfolio$symbols)

## [1] "SPY"

names(thePortfolio$symbols$SPY)

## [1] "txn"       "posPL.USD" "posPL"

names(thePortfolio$summary)

## [1] "Long.Value"       "Short.Value"      "Net.Value"        "Gross.Value"
## [5] "Realized.PL"      "Unrealized.PL"    "Gross.Trading.PL" "Txn.Fees"
## [9] "Net.Trading.PL"
```

The blotter_portfolio object



Transactions in the blotter_portfolio object

```
thePortfolio$symbols$SPY$txn[1:12,]
```

```
##            Txn.Qty  Txn.Price   Txn.Value Txn.Avg.Cost Pos.Qty Pos.Avg.Cost Gross.Txn.Realized.PL
## 1997-12-31       0   0.000000        0.00     0.000000       0     0.000000                 0.000
## 1998-10-30   12030  83.125042   999994.25    83.125042   12030    83.125042                 0.000
## 1999-09-30  -12030  98.425927 -1184063.91    98.425927       0     0.000000            184069.653
## 1999-10-29   11305 104.732831  1184004.66   104.732831   11305   104.732831                 0.000
## 2000-09-29  -11305 110.880542 -1253504.53   110.880542       0     0.000000             69499.870
## 2002-03-28   13916  90.076820  1253509.03    90.076820   13916    90.076820                 0.000
## 2002-04-30  -13916  84.838332 -1180610.23    84.838332       0     0.000000            -72898.796
## 2003-04-30   16058  73.524300  1180653.20    73.524300   16058    73.524300                 0.000
## 2004-08-31  -16058  90.615126 -1455097.69    90.615126       0     0.000000            274444.488
## 2004-09-30   15898  91.524627  1455058.52    91.524627   15898    91.524627                 0.000
## 2007-12-31  -15898 127.365627 -2024858.74   127.365627       0     0.000000            569800.219
## 2009-06-30   24372  83.080687  2024842.51    83.080687   24372    83.080687                 0.000
##            Txn.Fees Net.Txn.Realized.PL Con.Mult
## 1997-12-31        0                0.000        0
## 1998-10-30        0                0.000        1
## 1999-09-30        0           184069.653        1
## 1999-10-29        0                0.000        1
## 2000-09-29        0            69499.870        1
## 2002-03-28        0                0.000        1
## 2002-04-30        0           -72898.796        1
## 2003-04-30        0                0.000        1
## 2004-08-31        0           274444.488        1
## 2004-09-30        0                0.000        1
## 2007-12-31        0           569800.219        1
## 2009-06-30        0                0.000        1
```

Lattice graphics

The *lattice* add-on package is an implementation of Trellis graphics for R

- Lattice is a powerful and elegant high-level data visualization system with an emphasis on multivariate data
- It is designed to meet most typical graphics needs with minimal tuning, but can also be easily extended to handle most nonstandard requirements
- Trellis Graphics were originally developed for S and S-PLUS at the Bell Labs by R. Becker and W. Cleveland

The xyplot function

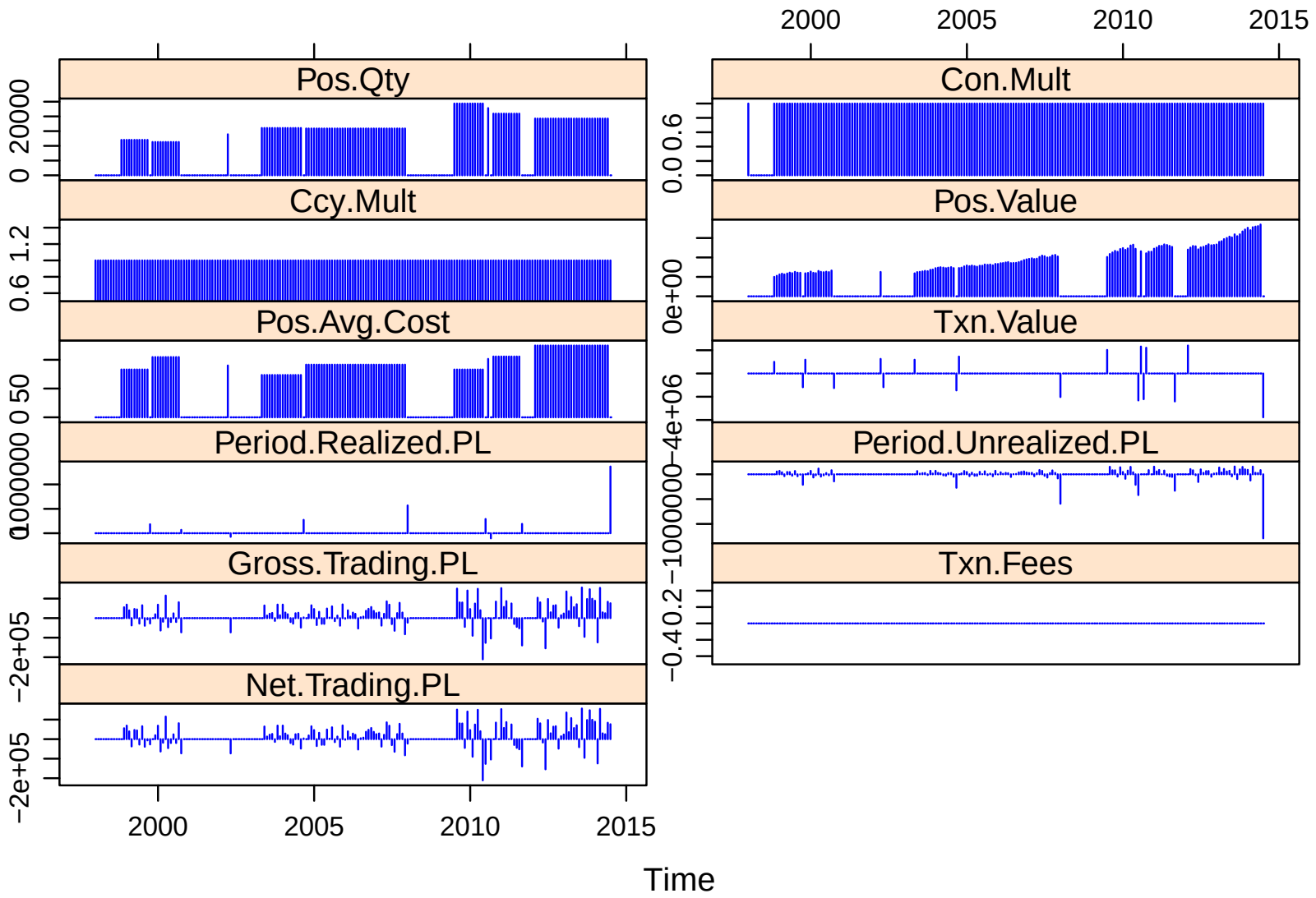
The function `xyplot` produces bivariate scatterplots or time-series plots

```
library(lattice)
xyplot(thePortfolio$symbols$SPY$posPL.USD,type="h",col=4)
```

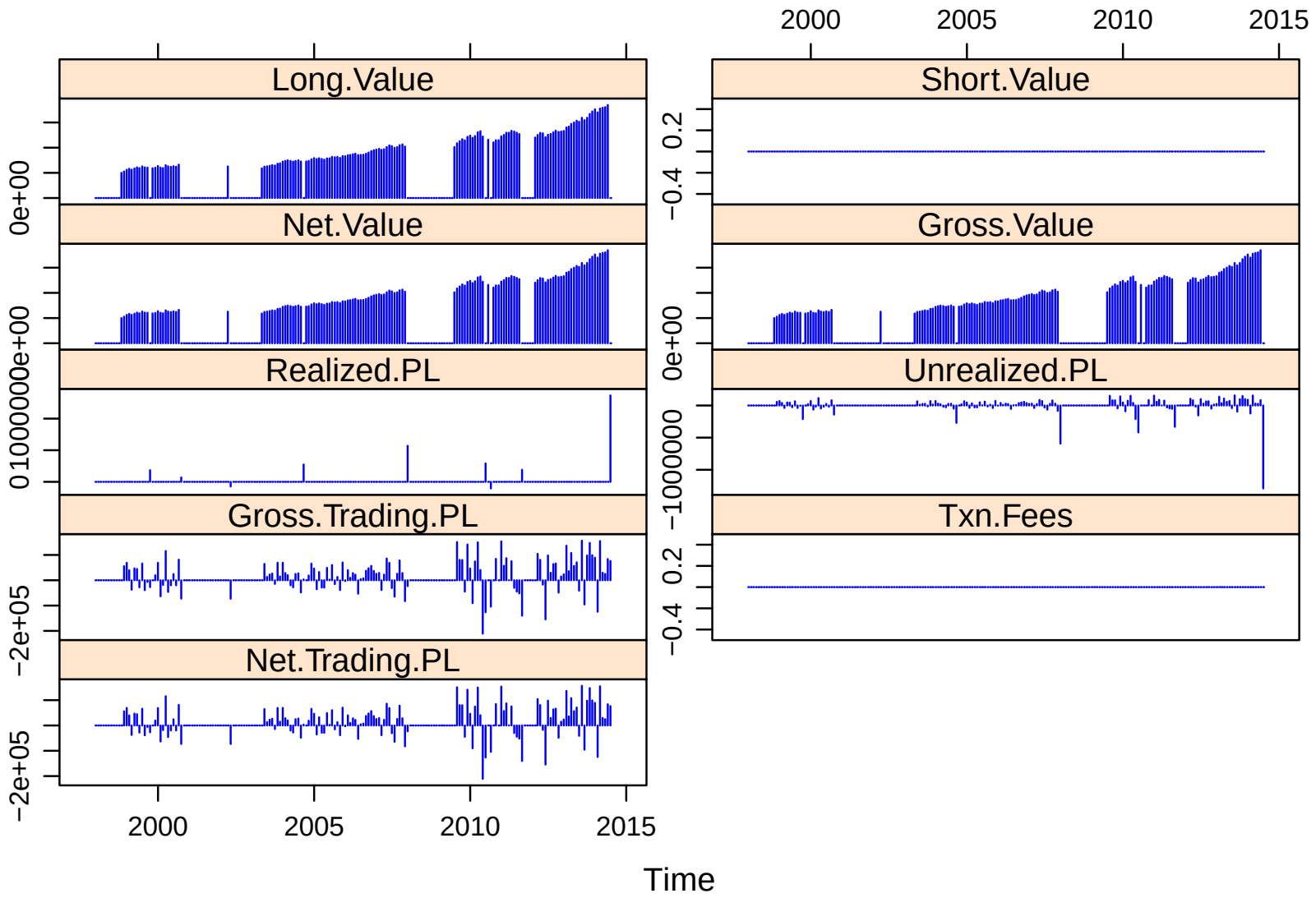
```
xyplot(thePortfolio$summary,type="h",col=4)
```

- *Lattice graphics* are very good for multi-panel plots

Plot of instrument P&L



Plot of portfolio summary time series object



The str function

The str function compactly displays the internal structure of an R object

```
args(str)

## function (object, ...)
## NULL
```

Main arguments:

object the R object to be inspected

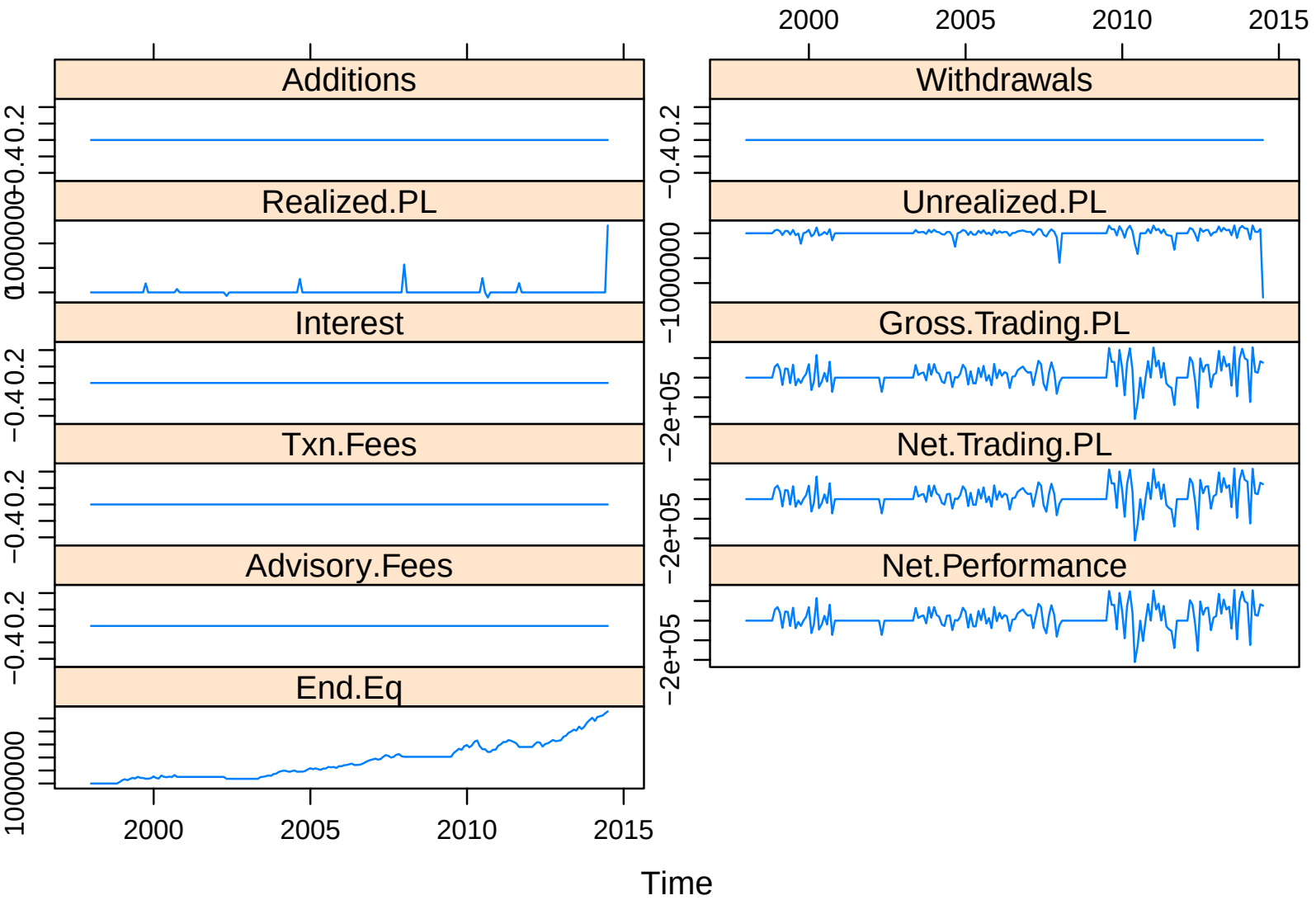
blotter_portfolio object before applyStrategy

```
str(thePortfolio)


## List of 2
##  $ summary:An 'xts' object on 1997-12-31/2014-06-30 containing:
##   Data: num [1:199, 1:9] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:9] "Long.Value" "Short.Value" "Net.Value" "Gross.Value" ...
##   Indexed by objects of class: [POSIXct,POSIXt] TZ:
##   xts Attributes:
##  NULL
##  $ symbols:List of 1
##   ..$ SPY:List of 3
##   .. ..$ txn      :An 'xts' object on 1997-12-31/2014-06-30 containing:
##   Data: num [1:19, 1:10] 0 12030 -12030 11305 -11305 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:10] "Txn.Qty" "Txn.Price" "Txn.Value" "Txn.Avg.Cost" ...
##   Indexed by objects of class: [POSIXct,POSIXt] TZ:
##   xts Attributes:
##  NULL
##   .. ..$ posPL.USD:An 'xts' object on 1997-12-31/2014-06-30 containing:
##   Data: num [1:199, 1:11] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:11] "Pos.Qty" "Con.Mult" "Ccy.Mult" "Pos.Value" ...
##   Indexed by objects of class: [POSIXct,POSIXt] TZ:
##   xts Attributes:
##  NULL
##   .. ..$ posPL    :An 'xts' object on 1997-12-31/2014-06-30 containing:
##   Data: num [1:199, 1:11] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:11] "Pos.Qty" "Con.Mult" "Ccy.Mult" "Pos.Value" ...
##   Indexed by objects of class: [POSIXct,POSIXt] TZ:
##   xts Attributes:
##  NULL
##  - attr(*, "class")= chr [1:2] "blotter_portfolio" "portfolio"
##  - attr(*, "currency")= chr "USD"
##  - attr(*, "initDate")= chr "1997-12-31"
```

The blotter *account* object

```
theAccount = getAccount(b.strategy)
names(theAccount)

## [1] "portfolios"  "summary"     "Additions"   "Withdrawals" "Interest"

names(theAccount$portfolios)

## [1] "bFaber"

names(theAccount$portfolios$bFaber)

## [1] "Long.Value"       "Short.Value"      "Net.Value"        "Gross.Value"
## [5] "Realized.PL"      "Unrealized.PL"    "Gross.Trading.PL" "Txn.Fees"
## [9] "Net.Trading.PL"

names(theAccount$summary)

##  [1] "Additions"        "Withdrawals"      "Realized.PL"      "Unrealized.PL"
##  [5] "Interest"         "Gross.Trading.PL" "Txn.Fees"         "Net.Trading.PL"
##  [9] "Advisory.Fees"    "Net.Performance"  "End.Eq"

xyplot(theAccount$summary)
```

Plot of account summary time series object



COMPUTATIONAL FINANCE & RISK MANAGEMENT

UNIVERSITY *of* WASHINGTON

Department of Applied Mathematics

`http://depts.washington.edu/compfin`