Getting started with quantstrat

`https://github.com/gyollin/quantstrat-tutorial.git`

Guy Yollin

www.r-programming.org

Legal Disclaimer

- This presentation is for informational purposes only
- This presentation should not be construed as a solicitation or offering of investment services
- The presentation does not intend to provide investment advice
- The information in this presentation should not be construed as an offer to buy or sell, or the solicitation of an offer to buy or sell any security, or as a recommendation or advice about the purchase or sale of any security
- The presenter(s) shall not shall be liable for any errors or inaccuracies in the information presented
- There are no warranties, expressed or implied, as to accuracy, completeness, or results obtained from any information presented

INVESTING ALWAYS INVOLVES RISK

Guy Yollin

- Professional Experience
  - Data Science roles
    - Milliman
    - r-programming.org
    - Insightful Corporation
  - Hedge Fund roles
    - Rotella Capital Management
    - J.E. Moody, LLC
  - Software Engineering roles
    - Electro Scientific Industries, Vision Products Division
    - Cybernetic Systems & Automation, Inc.
  - Academic roles
    - University of Washington
    - Oregon Graduate Institute
- Education
  - Oregon Graduate Institute, Computational Finance
  - Drexel University, Electrical Engineering
- Contact Info
  - gyollin@r-programming.org
  - http://www.linkedin.com/in/guyyollin

Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
   - Position limits
   - User-supplied order sizing function
6. Stop orders
7. Parameter optimization

Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
6. Stop orders
7. Parameter optimization

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

The zoo package

The zoo package provides an infrastructure for regularly-spaced and irregularly-space time series

Key functions:

zoo	create a zoo time series object
merge	merges time series (automatically handles of time alignment)
aggregate	create coarser resolution time series with summary statistics
rollapply	calculate rolling window statistics
read.zoo	read a text file into a zoo time series object

Authors:

- Achim Zeileis
- Gabor Grothendieck

The xts package

The `xts` package extends the zoo time series class with fine-grained time indexes, interoperability with other R time series classes, and user defined attributes

Key functions:

xts	create an xts time series object
align.time	align time series to a coarser resolution
to.period	convert time series data to an OHLC series
[.xts	subset time series

Authors:

- Jeffrey Ryan
- Josh Ulrich

The TTR package

The `TTR` package is a comprehensive collection of technical analysis indicators for R

Key features:

- moving averages
- oscillators
- price channels
- trend indicators

Author:

- Joshua Ulrich

The quantmod package

The `quantmod` package for R is designed to assist the quantitative trader in the development, testing, and deployment of statistically based trading models.

Key functions:

getSymbols load or download price data

- Yahoo Finance / Google Finance
- FRED
- Oanda
- csv, RData
- MySQL, SQLite

chartSeries charting tool to create standard financial charts

Author:

- Jeffrey Ryan

Install trading system development packages

```
#
# install these packages from CRAN (or r-forge)
#
install.packages("xts")
install.packages("PerformanceAnalytics")
install.packages("quantmod")
install.packages("TTR")
#
# Install these package from r-forge
#
install.packages("FinancialInstrument", repos = "http://R-Forge.R-project.org")
install.packages("blotter", repos = "http://R-Forge.R-project.org")
install.packages("quantstrat", repos = "http://R-Forge.R-project.org")
```

- R-Forge packages can be installed by setting the repos argument to `http://R-Forge.R-project.org`

Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
6. Stop orders
7. Parameter optimization

Time series data

Time series

A *time series* is a sequence of *ordered* data points measured at specific points in time

Time series object

A time series object in R is a *compound data structure* that includes a data matrix as well as a vector of associated time stamps

class	package	overview
`ts`	stats	regularly spaced time series
`mts`	stats	multiple regularly spaced time series
`zoo`	zoo	reg/irreg and arbitrary time stamp classes
`xts`	xts	an extension of the zoo class

Components of an xts object

An `xts` object is composed of 3 components:

Index a Data class or Time-Date class used for the time-stamp of observations

Matrix the time series observations (univariate or multivariate)

- can be numeric, character, logical, etc. but must be homogeneous

Attr hidden attributes and user attributes

- class of the index
- format of the index
- time zone

The getSymbols function

The getSymbols function loads (downloads) historic price data

Usage:

```
getSymbols(Symbols = NULL, env = parent.frame(), src = "yahoo",
  auto.assign = getOption('getSymbols.auto.assign',TRUE), ...)
```

Main arguments:

Symbols	a vector of ticker symbols
env	where to create objects. Setting env=NULL is equal to auto.assign=FALSE
src	source of data (yahoo)
auto.assign	should results be returned or loaded to env
...	additional parameters

Return value:

an object of type return.class depending on env and auto.assign

The getSymbols.yahoo function

The `getSymbols.yahoo` function downloads historic price data from `finance.yahoo.com`

Usage:

```
getSymbols.yahoo(Symbols, env, return.class = 'xts', index.class  = 'Date',
  from = "2007-01-01", to = Sys.Date(), ...)
```

Main arguments:

return.class	class of returned object
index.class	class of returned object index (xts only)
...	additional parameters

Return value:

an object of type `return.class` depending on `env` and `auto.assign`

The getSymbols function

```
library(quantmod)
ls()
```

```
## character(0)
```

```
getSymbols("^GSPC")
```

```
ls()
```

```
## [1] "GSPC"
```

```
class(GSPC)
```

```
## [1] "xts" "zoo"
```

```
class(index(GSPC))
```

```
## [1] "Date"
```

```
dim(GSPC)
```

```
## [1] 2343    6
```

- By default, the symbol was *auto-assigned* to the parent environment

The getSymbols function

```
tail(GSPC,4)
```

```
##            GSPC.Open GSPC.High  GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
## 2016-04-19 2096.0500 2104.0500 2091.6799  2100.8000  3896830000     2100.8000
## 2016-04-20 2101.5200 2111.0500 2096.3201  2102.3999  4184880000     2102.3999
## 2016-04-21 2102.0901 2103.7800 2088.5200  2091.4800  4175290000     2091.4800
## 2016-04-22 2091.4900 2094.3201 2081.2000  2091.5801  3790580000     2091.5801
```

```
tail(Cl(GSPC),4)
```

```
##            GSPC.Close
## 2016-04-19  2100.8000
## 2016-04-20  2102.3999
## 2016-04-21  2091.4800
## 2016-04-22  2091.5801
```

```
tail(Ad(GSPC),4)
```

```
##            GSPC.Adjusted
## 2016-04-19     2100.8000
## 2016-04-20     2102.3999
## 2016-04-21     2091.4800
## 2016-04-22     2091.5801
```

- Note that the symbol is prepended to columns names of the xts object; use extractor functions to access column data (e.g. `Cl(GSPC)`)

quantmod extractor functions

The quantmod package includes a number of functions to extract specific series from an xts object of market data:

Function	Description
Op(x)	Get Open
Hi(x)	Get High
Lo(x)	Get Low
Cl(x)	Get Close
Vo(x)	Get Volume
Ad(x)	Get Adjusted Close
HLC(x)	Get High, Low, and Close
OHLC(x)	Get Open, High, Low, and Close

The chartSeries function

The chartSeries function creates financial charts

Usage:

```
chartSeries(x, type = c("auto", "candlesticks", "matchsticks",
    "bars", "line"), subset = NULL, show.grid = TRUE, name = NULL,
    time.scale = NULL, log.scale = FALSE, TA = "addVo()", TAsep = ";",
    line.type = "l", bar.type = "ohlc", theme = chartTheme("black"),
    layout = NA, major.ticks = "auto", minor.ticks = TRUE, yrange = NULL,
    plot = TRUE, up.col, dn.col, color.vol = TRUE, multi.col = FALSE, ...)
```

Main arguments:

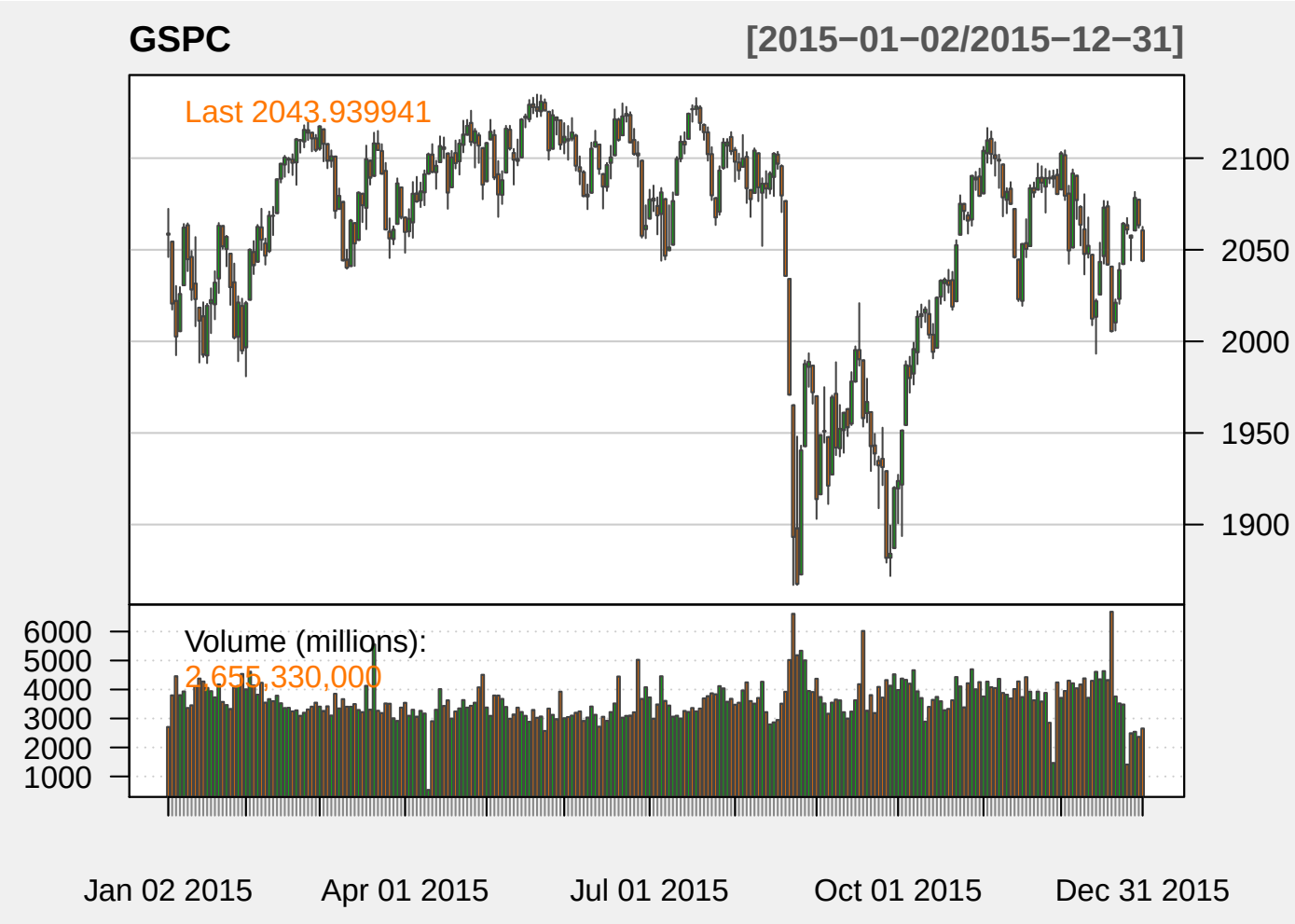
x	an OHLC object
type	style of chart to draw
theme	a chart.theme object
subset	xts style date subsetting argument
TA	a vector of technical indicators and params

Return value:

a chob object

The chartSeries function

```
chartSeries(GSPC,subset="2015",theme="white")
```



Customize a chartSeries plot

```
whiteTheme <- chartTheme("white")
names(whiteTheme)
```

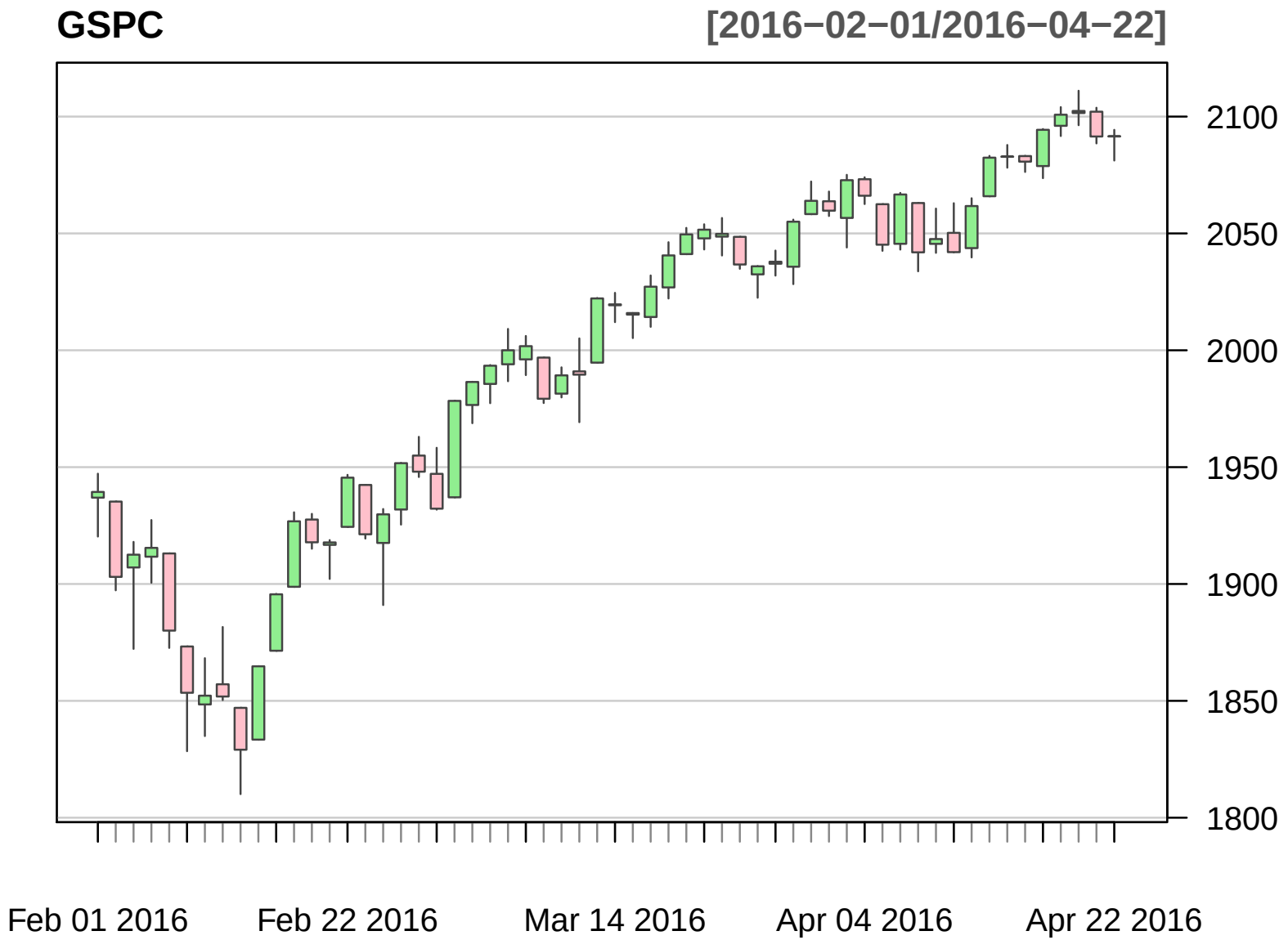
```
##  [1] "fg.col"       "bg.col"       "grid.col"     "border"       "minor.tick"
##  [6] "major.tick"   "up.col"       "dn.col"       "dn.up.col"    "up.up.col"
## [11] "dn.dn.col"    "up.dn.col"    "up.border"    "dn.border"    "dn.up.border"
## [16] "up.up.border" "dn.dn.border" "up.dn.border" "main.col"     "sub.col"
## [21] "area"         "fill"         "Expiry"       "theme.name"
```

```
whiteTheme$bg.col <- "white"
whiteTheme$dn.col <- "pink"
whiteTheme$up.col <- "lightgreen"
whiteTheme$border <- "lightgray"
x <- chartSeries(GSPC,subset="last 3 months",theme=whiteTheme,TA=NULL)
class(x)
```

```
## [1] "chob"
## attr(,"package")
## [1] "quantmod"
```

- subset to last 3 months
- totally white background
- no volume sub-graph (`TA=NULL`)

A chartSeries plot



Date class

A `Date` object is stored internally as the number of days since 1970-01-01

```
myStr <- "7/4/2014"
class(myStr)
```

```
## [1] "character"
```

```
args(getS3method("as.Date","character"))
```

```
## function (x, format, ...)
## NULL
```

```
myDate <- as.Date(myStr,format="%m/%d/%Y")
myDate
```

```
## [1] "2014-07-04"
```

```
class(myDate)
```

```
## [1] "Date"
```

```
as.numeric(myDate)
```

```
## [1] 16255
```

Date format string for as.Date and format.Date

%Y Year with century
%y Year without century (00-99)
%m Month as decimal number (01-12)
%d Day of the month as decimal number (01-31)

```
format(myDate,"%m/%d/%Y")

## [1] "07/04/2014"

format(myDate,"%m/%d/%y")

## [1] "07/04/14"

format(myDate,"%Y%m%d")

## [1] "20140704"
```

- For comprehensive list of date/time conversion specifications, see help for `strptime` function

Time-Date formatting strings

%a	Abbreviated weekday name in the current locale
%A	Full weekday name in the current locale
%b	Abbreviated month name in the current locale
%B	Full month name in the current locale
%c	Date and time. Locale-specific on output
%C	Century (00–99): the integer part of the year divided by 100.
%d	Day of the month as decimal number (01–31).
%D	Date format such as %m/%d/%y: ISO C99 says it should be that exact format.
%e	Day of the month as decimal number (1–31)
%F	Equivalent to %Y-%m-%d (the ISO 8601 date format)
%g	The last two digits of the week-based year (see %V)
%G	The week-based year (see %V) as a decimal number
%h	Equivalent to %b
%H	Hours as decimal number (00–23)
%I	Hours as decimal number (01–12)
%j	Day of year as decimal number (001–366)
%m	Month as decimal number (01–12)
%M	Minute as decimal number (00–59)

Time-Date formatting strings (continued)

%n	Newline on output, arbitrary whitespace on input
%p	AM/PM indicator in the locale. Used in conjunction with %I and not with %H
%r	The 12-hour clock time (using the locale's AM or PM)
%R	Equivalent to %H:%M
%S	Second as decimal number (00–61)
%T	Equivalent to %H:%M:%S
%u	Weekday as a decimal number (1–7, Monday is 1)
%U	Week of the year as decimal number (00–53) using Sunday as the first day 1 of the week
%V	Week of the year as decimal number (00–53) as defined in ISO 8601
%w	Weekday as decimal number (0–6, Sunday is 0)
%W	Week of the year as decimal number (00–53) using Monday as the first day of week
%x	Date. Locale-specific on output, "%y/%m/%d" on input
%X	Time. Locale-specific on output, "%H:%M:%S" on input
%y	Year without century (00–99)
%Y	Year with century
%z	Signed offset in hours and minutes from UTC, so -0800 is 8 hours behind UTC
%Z	(Output only.) Time zone abbreviation as a character string (empty if not available)

Date-Time classes

- A `POSIXct` object is a Date-Time object internally stored as the number of seconds since 1970-01-01
- A `POSIXlt` object is a Date-Time object internally stored as 9 calendar and time components

```
d <- Sys.time()
class(d)
```

```
## [1] "POSIXct" "POSIXt"
```

```
unclass(d)
```

```
## [1] 1461375401
```

```
sapply(unclass(as.POSIXlt(d)), function(x) x)
```

```
##                sec                min               hour               mday
## "40.9073739051819"               "36"               "18"               "22"
##                mon               year               wday               yday
##                "3"              "116"                "5"              "112"
##              isdst               zone             gmtoff
##                "1"              "PDT"          "-25200"
```

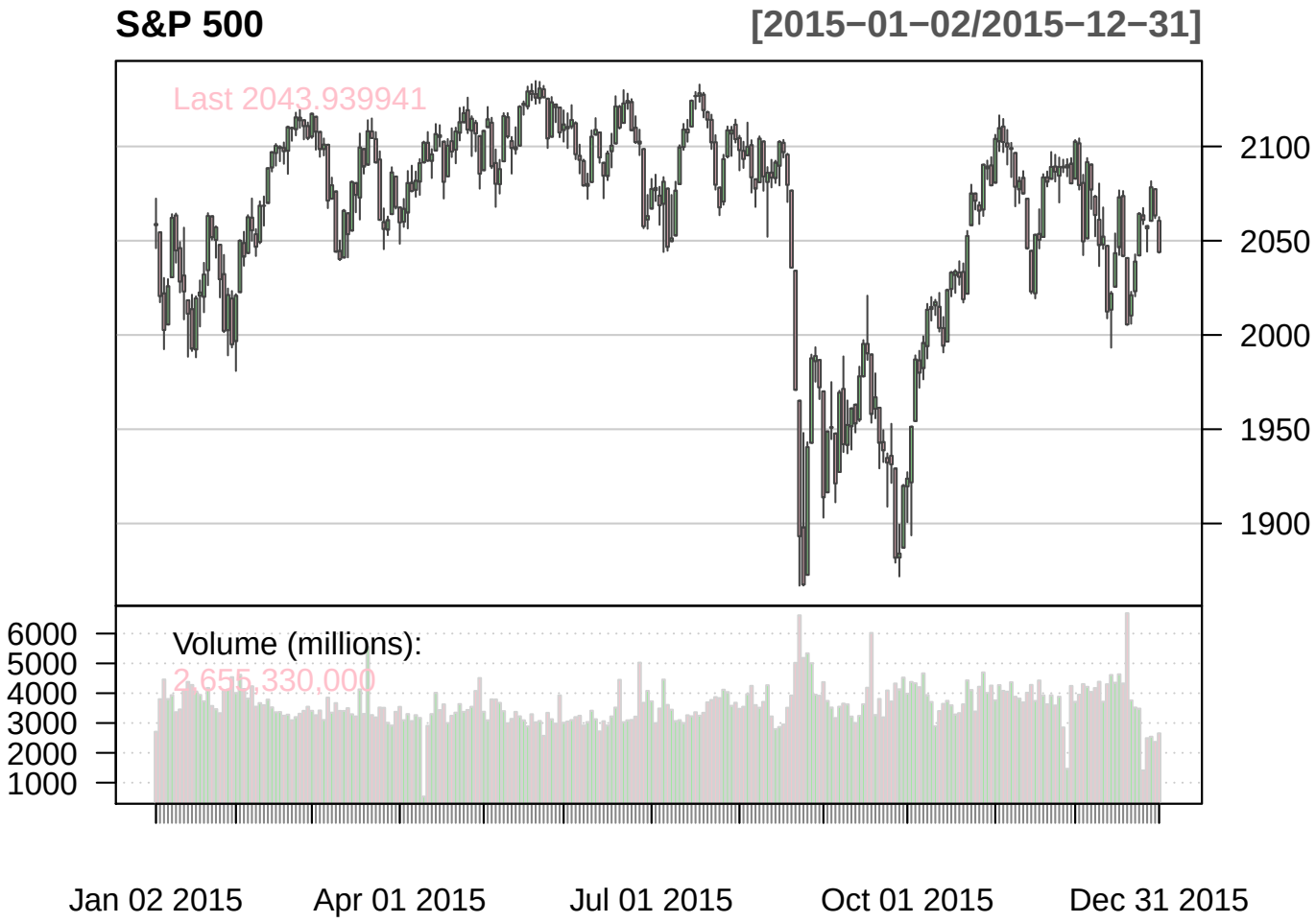
xts subsetting

xts time series objects can be easily subset:

- Date and time organized from most significant to least significant
  - CCYY-MM-DD HH:MM:SS[.s]
- Separators can be omitted
  - CCYYMMDDHHMMSS
- Intervals can be designated with the "/" or "::"
  - 2010/2011
  - 2011-04::2011-07
  - ::Sys.time()
  - 2000::

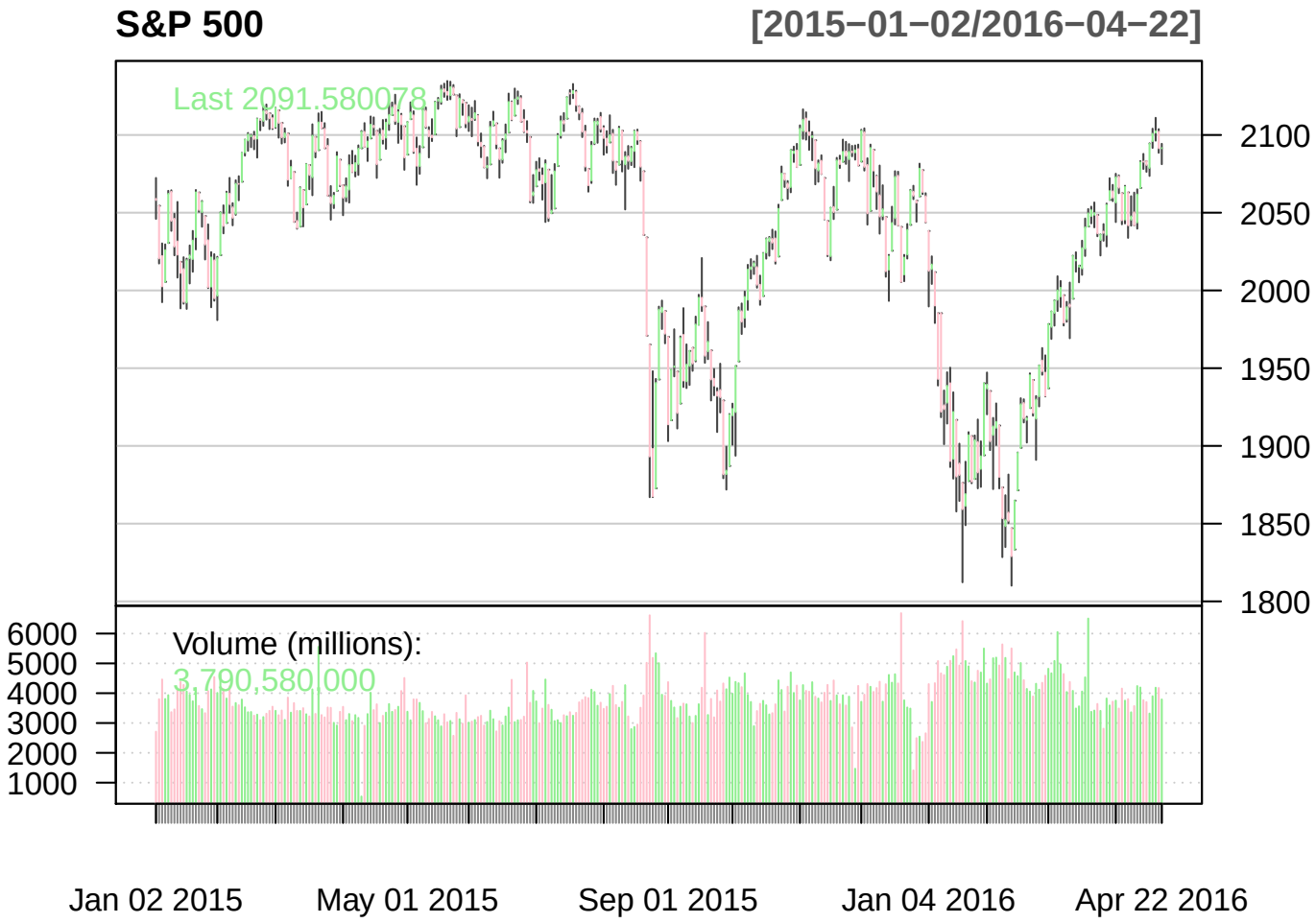
xts subsetting

```
chartSeries(GSPC["2015"],theme=whiteTheme,name="S&P 500")
```



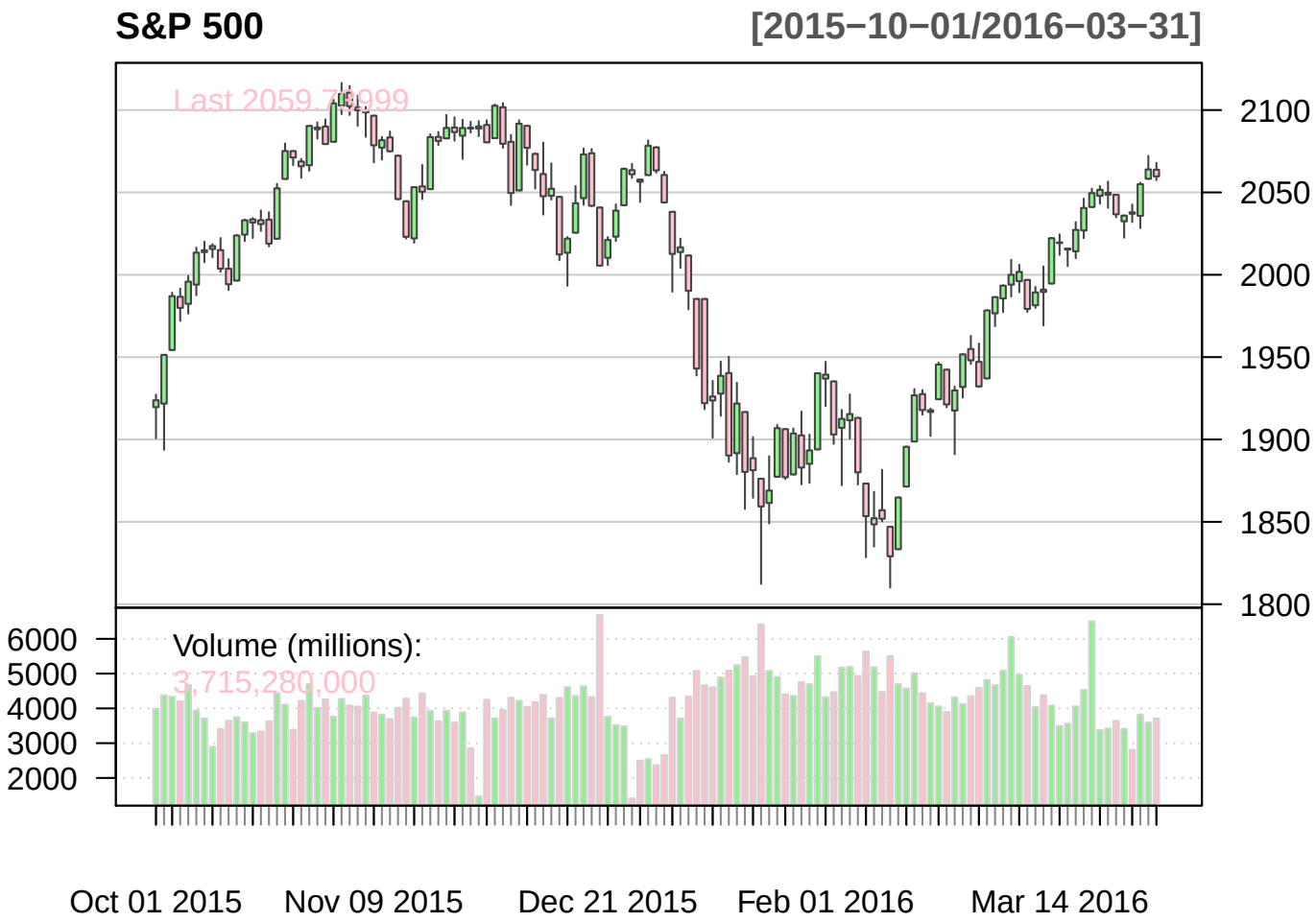
xts subsetting

```
chartSeries(GSPC["2015/2016"],theme=whiteTheme,name="S&P 500")
```



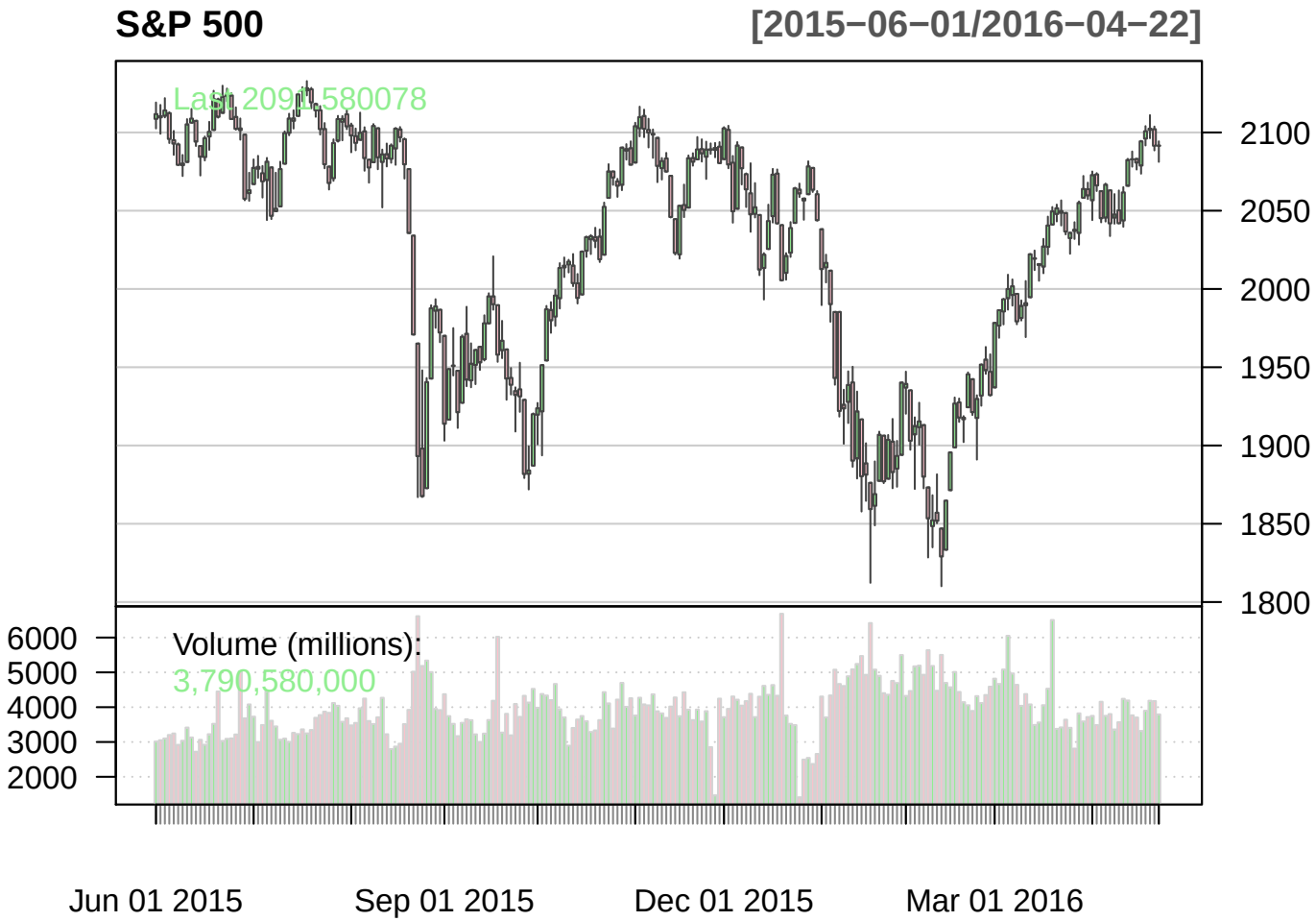
xts subsetting

```
chartSeries(GSPC["2015-10::2016-03"],theme=whiteTheme,name="S&P 500")
```



xts subsetting

```
chartSeries(GSPC["201506::"],theme=whiteTheme,name="S&P 500")
```



Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
6. Stop orders
7. Parameter optimization

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

The quantstrat package

Description

quantstrat provides a generic infrastructure to model and backtest signal-based quantitative strategies. It is a high-level abstraction layer (built on xts, FinancialInstrument, blotter, etc.) that allows you to build and test strategies in very few lines of code.

Key features

- Supports strategies which include indicators, signals, and rules
- Allows strategies to be applied to multi-asset portfolios
- Supports market, limit, stoplimit, and stoptrailing order types
- Supports order sizing and parameter optimization

Authors

- Peter Carl, Brian Peterson
- Joshua Ulrich, Jan Humme

ETF Portfolio

In the following examples, we'll use a 9-asset portfolio composed of the 9 Select Sector SPDRs that divide the S&P 500 into nine sector index funds:

Symbol	Sector
XLY	Consumer Discretionary
XLP	Consumer Staples
XLE	Energy
XLF	Financial
XLV	Health Care
XLI	Industrial
XLB	Materials
XLK	Technology
XLU	Utilities

Download data

```
library(PerformanceAnalytics)
library(quantmod)
library(lattice)
startDate <- '2010-01-01'   # start of data
endDate <-  '2015-05-01'    # end of data
Sys.setenv(TZ="UTC")        # set time zone
symbols = c("XLF", "XLP", "XLE", "XLY", "XLV", "XLI", "XLB", "XLK", "XLU")
```

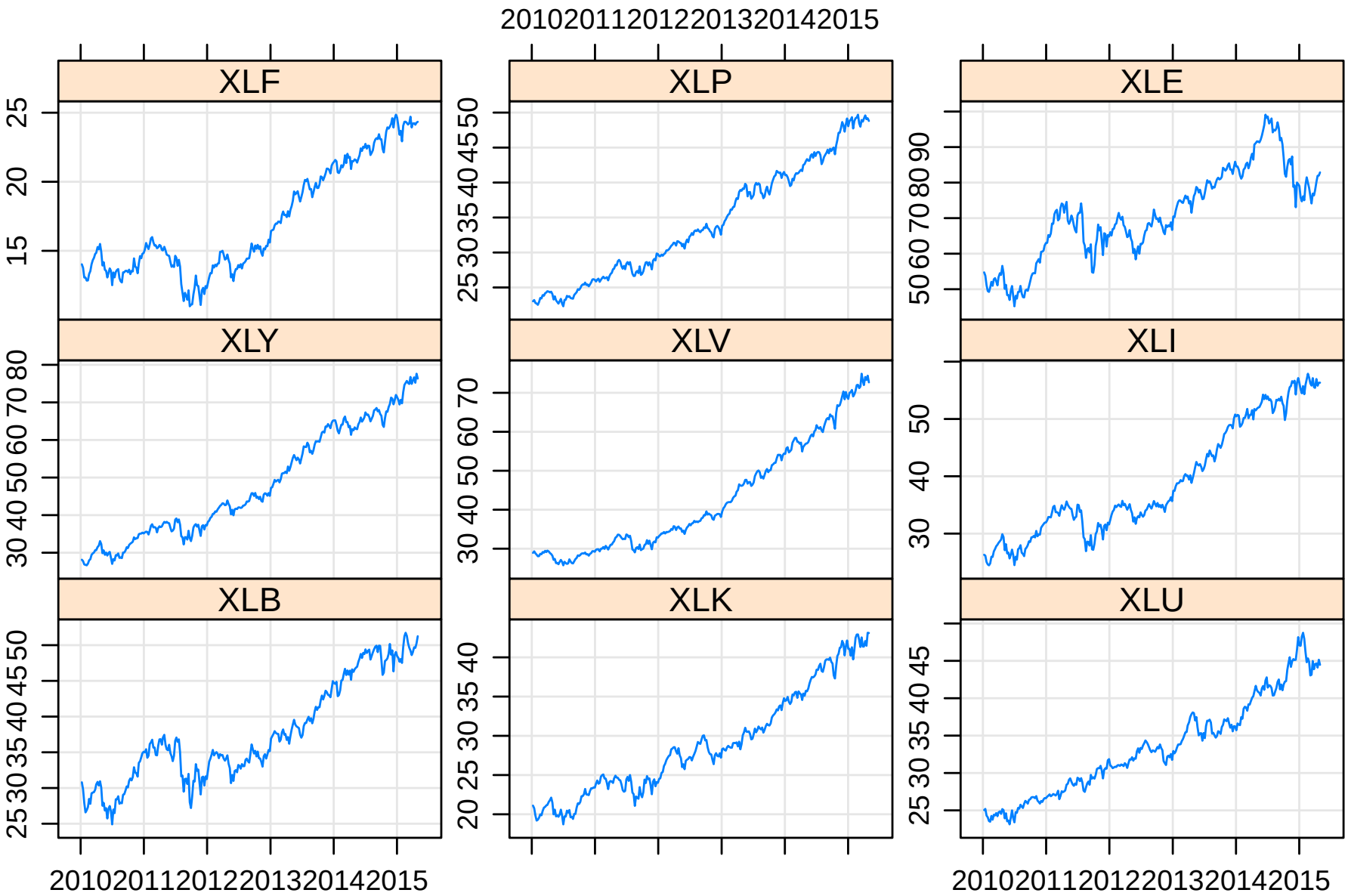
```
getSymbols(symbols, from=startDate, to=endDate, index.class="POSIXct")
for(symbol in symbols) {
    x<-get(symbol)
    x<-adjustOHLC(x,symbol.name=symbol)
    x<-to.weekly(x,indexAt='lastof',drop.time=TRUE)
    indexFormat(x)<-'%Y-%m-%d'
    colnames(x)<-gsub("x",symbol,colnames(x))
    assign(symbol,x)
}
```

- Set timezone
- Use `POSIXct` as index class for historic quotes

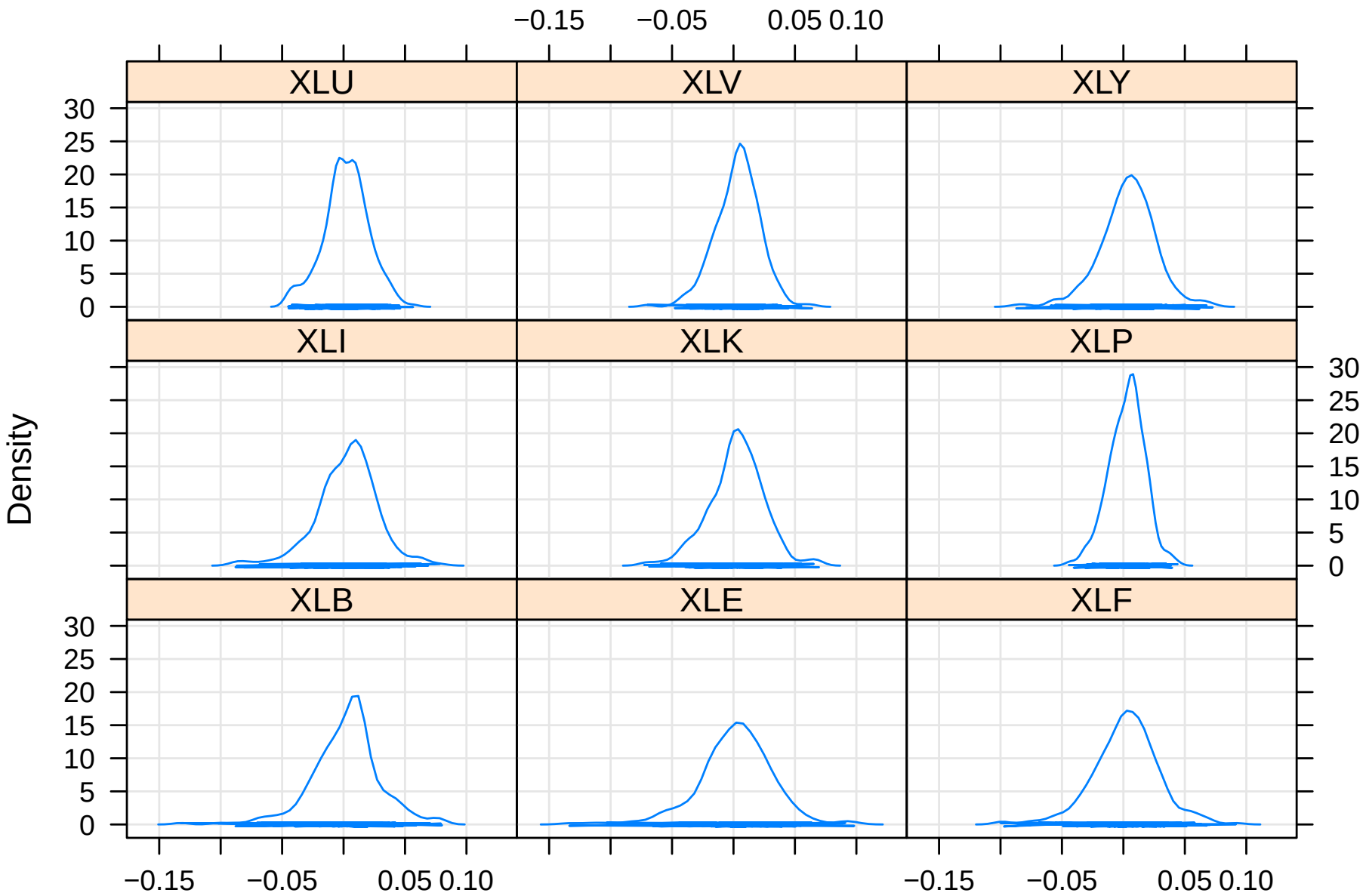
Compute returns

```
prices <- NULL
for(i in 1:length(symbols))
  prices <- cbind(prices,Cl(get(symbols[i])))
colnames(prices) <- symbols
returns <- diff(log(prices))[-1, ]
num.ass <- ncol(returns)

xyplot(prices, xlab = "", layout = c(3, 3),type=c("l","g"))
stacked.df <- stack(as.data.frame(returns))
colnames(stacked.df) <- c("returns", "symbol")

densityplot(~returns | symbol, stacked.df, cex = 0.25, xlab="",type=c("l","g"))
```

Sector Select SPDRs



Sector Select SPDRs



Bollinger bands

- Bollinger bands are a volatility-sensitive price channel
- Published by John Bollinger in the early 1980s
- RSI Calculation
  - MA(nMA) = simple moving average of the weighted-close
  - Upper Band = MA(nMA) + nSD $\times$ StdDev(C)
  - Lower Band = MA(nMA) - nSD $\times$ StdDev(C)
  - nMA typically 20
  - nSD typically in the range of 2 to 3
- Interpretation
  - Trade channel reversals between the upper and lower bands
  - Trade channel break-outs above/below the bands

Long-only Bollinger Band breakout strategy

Buy rule:

- Buy long when the High crosses above the upper band

Exit rule:

- Exit when the Low crosses below the lower band

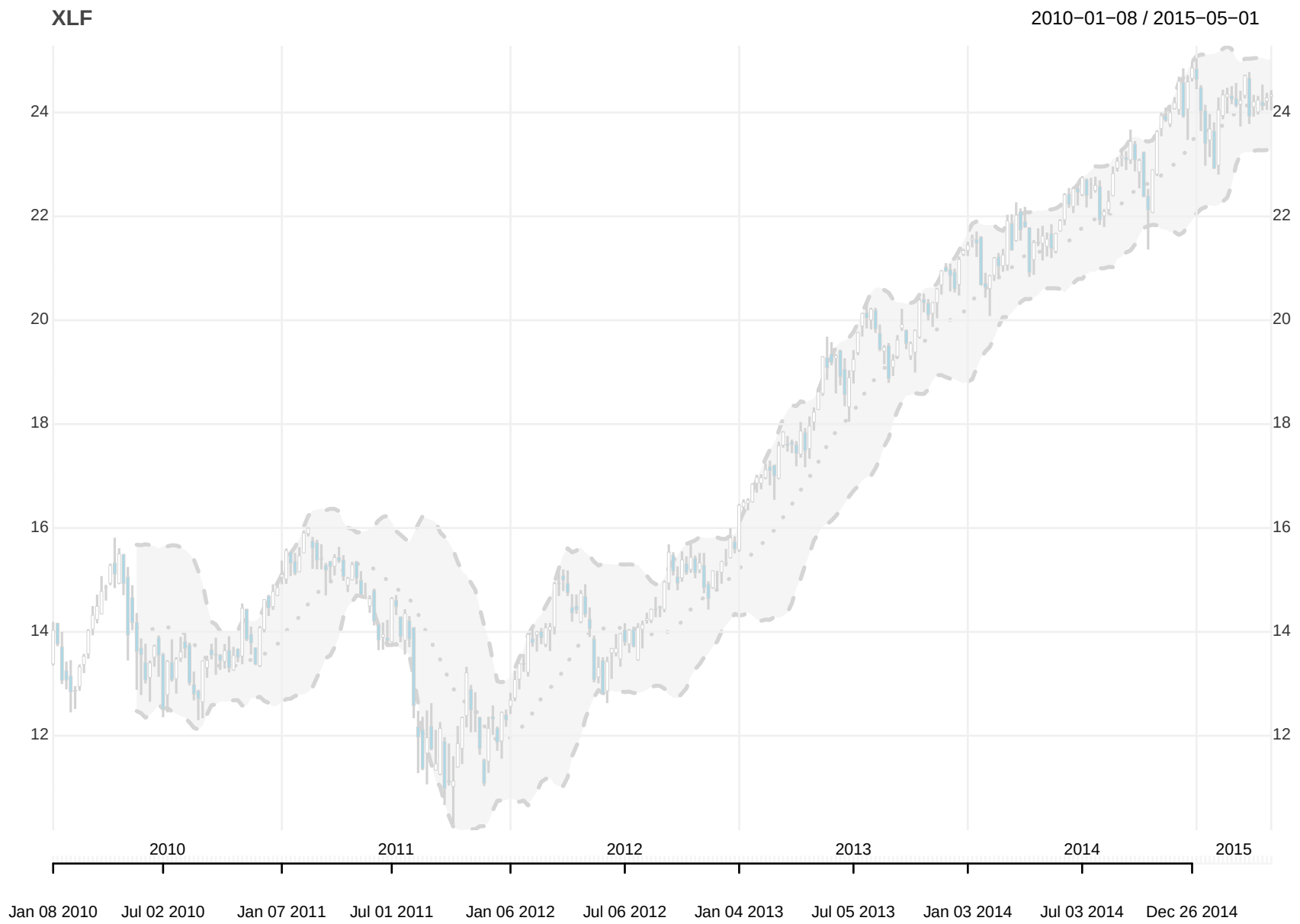
Pyramiding:

- Multiple orders in the same direction

Calculate and plot Bollinger bands

```
args(BBands)

## function (HLC, n = 20, maType, sd = 2, ...)
## NULL

b <- BBands(HLC=HLC(XLF["2013"]), n=20, sd=2)
tail(b)

##                    dn      mavg        up       pctB
## 2013-11-22 18.990974 19.855600 20.720225 1.04839406
## 2013-11-29 18.936071 19.924146 20.912221 1.04058157
## 2013-12-06 18.901882 19.966600 21.031319 0.90694354
## 2013-12-13 18.885953 19.998509 21.111064 0.81960132
## 2013-12-20 18.853322 20.041854 21.230386 0.88533002
## 2013-12-27 18.799933 20.110740 21.421546 0.96115360

myTheme<-chart_theme()
myTheme$col$dn.col<-'lightblue'
myTheme$col$dn.border <- 'lightgray'
myTheme$col$up.border <- 'lightgray'
chart_Series(XLF,TA='add_BBands(lwd=2)',theme=myTheme,name="XLF")
```

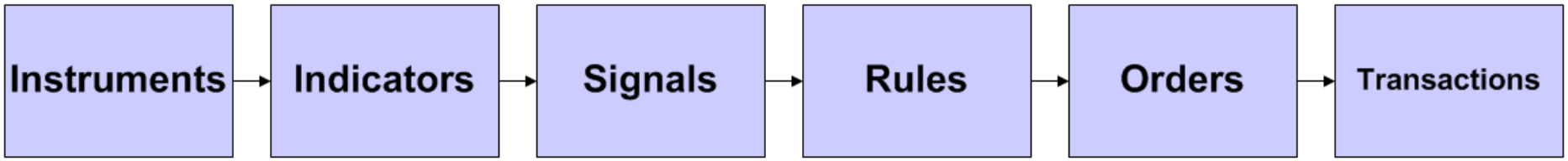
Bollinger bands



Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
6. Stop orders
7. Parameter optimization

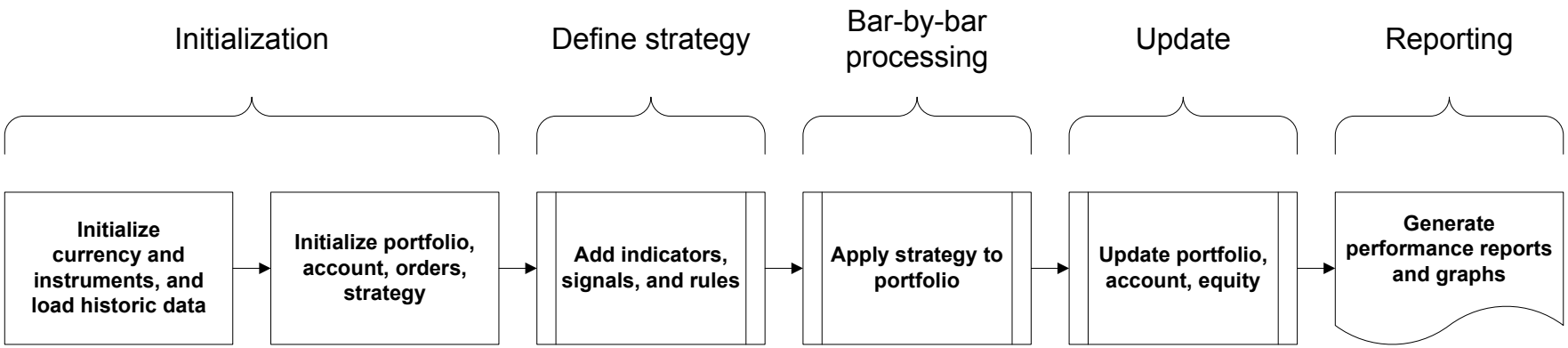
Quantstrat object model



Generic Signal-Based Strategy Modeling:

- Instruments contain market data
- Indicators are quantitative values derived from market data
- Interaction between indicators and market data are used to generate signals (e.g. crossovers, thresholds)
- Rules use market data, indicators, signals, and current account/portfolio characteristics to generate orders
- Interaction between orders and market data generates transactions

Basic strategy backtesting workflow for quantstrat



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
Analysis	
chart.Posn	chart market data, position size, and cumulative P&L
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

Key quantstrat functions

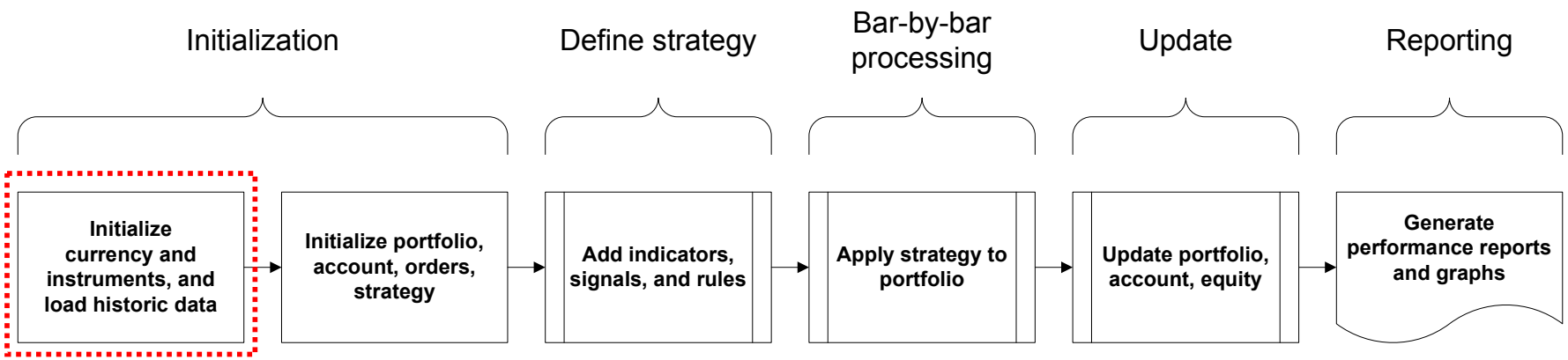
Initialization	
initOrders	initialize order container
strategy	constructor for strategy object

Strategy definition	
add.indicator	add an indicator to a strategy
add.signal	add a signal to a strategy
add.rule	add a rule to a strategy
add.distribution	add a distribution to a paramset in a strategy
add.constraint	add a constraint on 2 distributions within a paramset

Processing	
applyStrategy	apply the strategy to arbitrary market data
addPosLimit	add position and level limits at timestamp
apply.paramset	apply a paramset to the strategy
applyStrategy.rebalancing	apply the strategy to data with periodic rebalancing

The functions in quantstrat are used in conjunction with the functions in `blotter`

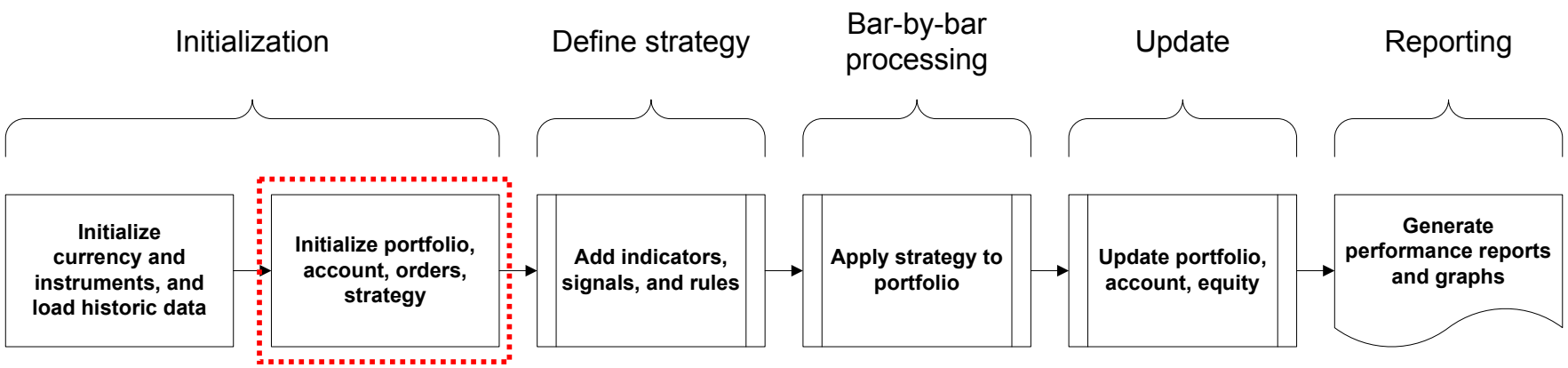
Initialize instruments




```
library(quantstrat)
initDate <- '2009-12-31'
initEq <- 1e6
currency("USD")
stock(symbols, currency="USD", multiplier=1)
```

- Initialize currency instrument first and then stock instrument
- Important that portfolio, account, and orderbook initialization date be before start of data

Initialize portfolio, account, and orders object




```
rm.strat("multiAsset.bb1") # remove portfolio, account, orderbook if re-run
initPortf(name="multiAsset.bb1", symbols, initDate=initDate)
initAcct(name="multiAsset.bb1", portfolios="multiAsset.bb1",
  initDate=initDate, initEq=initEq)
initOrders(portfolio="multiAsset.bb1", initDate=initDate)
```

```
strategy("bbands", store=TRUE)
```

- The function `rm.strat` removes any existing portfolio, account, or orderbook objects which facilitates re-running the code
- The function `strategy` initializes and new strategy object

The add.indicator function

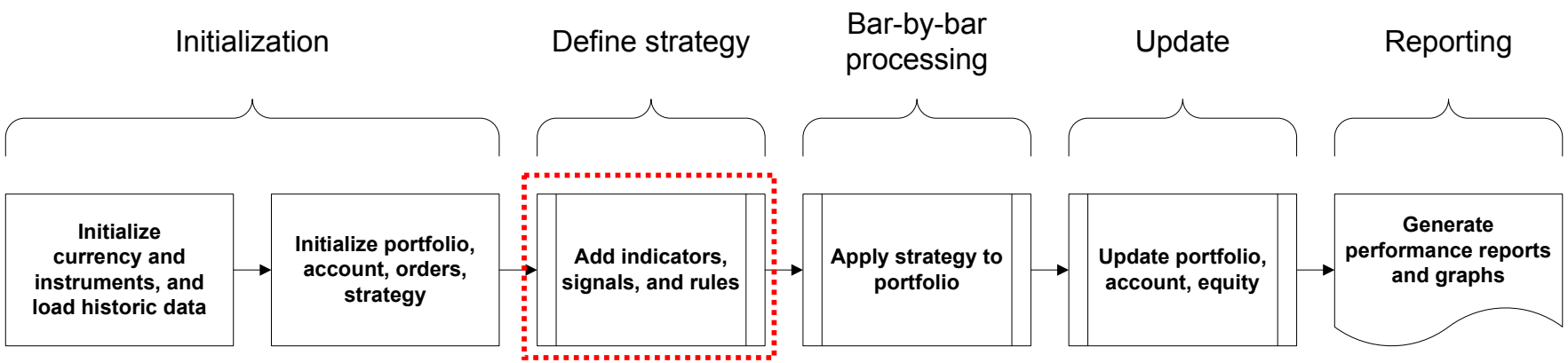
- Indicators are typically standard technical or statistical analysis outputs, such as moving averages, bands, or pricing models
- Indicators are applied before signals and rules, and the output of indicators may be used as inputs to construct signals or fire rules

```
args(add.indicator)

## function (strategy, name, arguments, parameters = NULL, label = NULL,
##     ..., enabled = TRUE, indexnum = NULL, store = FALSE)
## NULL
```

Main arguments:

strategy	strategy object
name	name of the indicator (must be an R function)
arguments	arguments to be passed to the indicator function
label	name to reference the indicator

Define indicators




```
args(BBands)

## function (HLC, n = 20, maType, sd = 2, ...)
## NULL
```

```
add.indicator("bbands", name = "BBands",
  arguments = list(HLC = quote(HLC(mktdata)), maType='SMA'), label='bbInd')
```

- `quote()` returns it's argument without evaluating
- `mktdata` is the time series object that holds the current symbols data during evaluation

The add.signals function

quantstrat supports the following signal types:

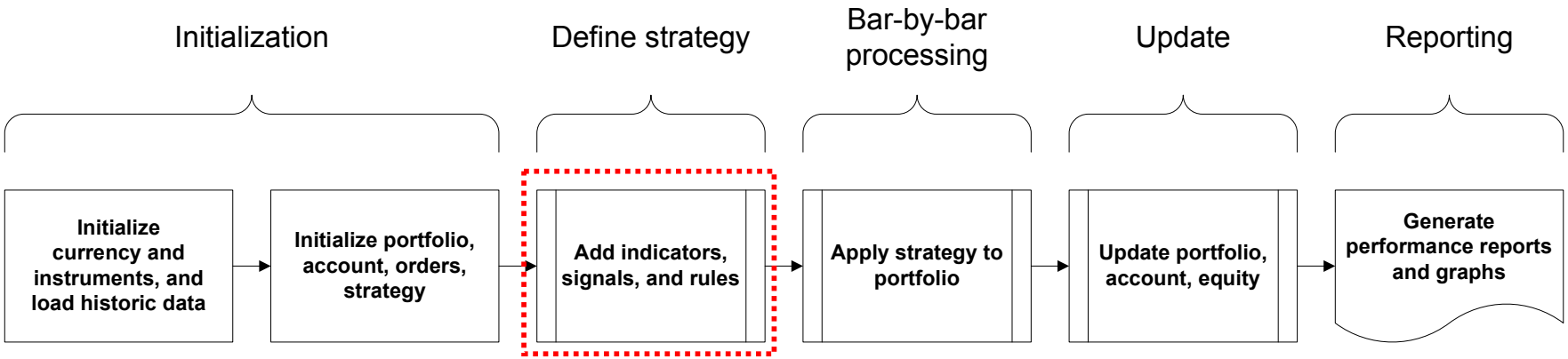
sigCrossover	crossover signal ("gt", "lt", "eq", "gte", "lte")
sigComparison	comparison signal ("gt", "lt", "eq", "gte", "lte")
sigThreshold	threshold signal ("gt", "lt", "eq", "gte", "lte")
sigPeak	peak/valley signals ("peak", "bottom")
sigFormula	signal calculated from a formula

```
args(add.signal)

## function (strategy, name, arguments, parameters = NULL, label = NULL,
##     ..., enabled = TRUE, indexnum = NULL, store = FALSE)
## NULL
```

Main arguments:

strategy	strategy object
name	name of the signal, must correspond to an R function
arguments	arguments to be passed to the signal function

Define signals




```
add.signal("bbands", name="sigCrossover",
  arguments=list(columns=c("High","up"),relationship="gt"),
  label="H.gt.UpperBand")
```

```
add.signal("bbands", name="sigCrossover",
  arguments=list(columns=c("Low","dn"),relationship="lt"),
  label="L.lt.LowerBand")
```

The add.rules function

The function `add.rule` adds a rule to a strategy

```
args(add.rule)

## function (strategy, name, arguments, parameters = NULL, label = NULL,
##     type = c(NULL, "risk", "order", "rebalance", "exit", "enter",
##         "chain"), parent = NULL, ..., enabled = TRUE, indexnum = NULL,
##     path.dep = TRUE, timespan = NULL, store = FALSE, storefun = TRUE)
## NULL
```

Main arguments:

strategy	strategy object
name	name of the rule (typically `ruleSignal`)
arguments	arguments to be passed to the rule function
type	type of rule ("risk","order","rebalance","exit","enter")
label	user supplied text label for rule

The ruleSignal function

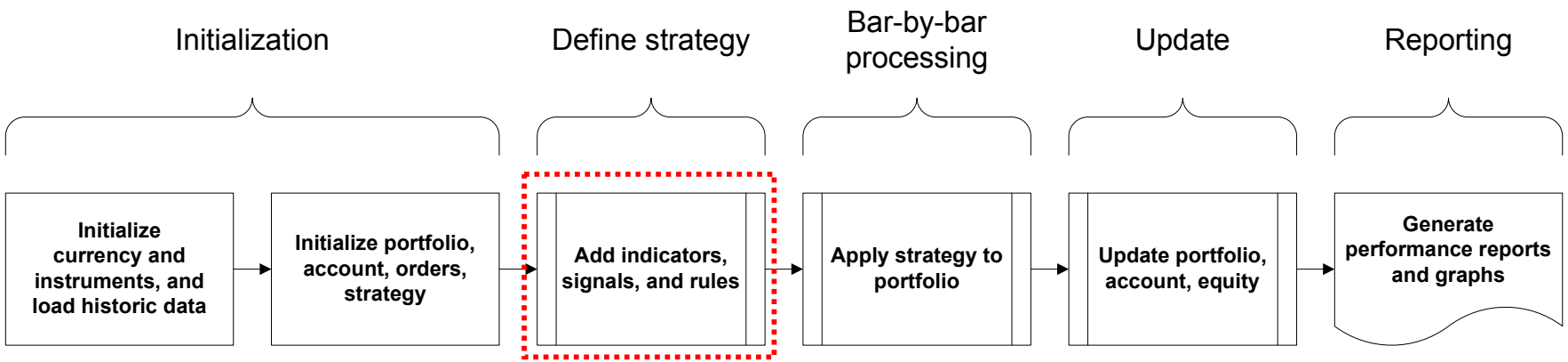
ruleSignal is the default rule to generate a trade order on a signal

```
args(ruleSignal)

## function (mktdata = mktdata, timestamp, sigcol, sigval, orderqty = 0,
##     ordertype, orderside = NULL, orderset = NULL, threshold = NULL,
##     tmult = FALSE, replace = TRUE, delay = 1e-04, osFUN = "osNoOp",
##     pricemethod = c("market", "opside", "active"), portfolio,
##     symbol, ..., ruletype, TxnFees = 0, prefer = NULL, sethold = FALSE,
##     label = "", order.price = NULL, chain.price = NULL, time.in.force = "")
## NULL
```

Main arguments:

sigcol	column name to check for signal
sigval	signal value to match
orderqty	quantity for order or 'all', modified by osFUN
ordertype	"market","limit","stoplimit","stoptrailing","iceberg"
orderside	"long", "short", or NULL
osFUN	function or name of order sizing function (default is osNoOp)

Add rules




```
add.rule("bbands", name='ruleSignal',
  arguments=list(sigcol="H.gt.UpperBand",sigval=TRUE,
    orderqty=+100, ordertype='market', orderside='long'),
  type='enter',
  label='LongEntry')
```

```
add.rule("bbands", name='ruleSignal',
  arguments=list(sigcol="L.lt.LowerBand",sigval=TRUE,
    orderqty= 'all', ordertype='market', orderside='long'),
  type='exit',
  label='LongExit')
```

- Long-only channel breakout system with pyramiding

The applyStrategy function

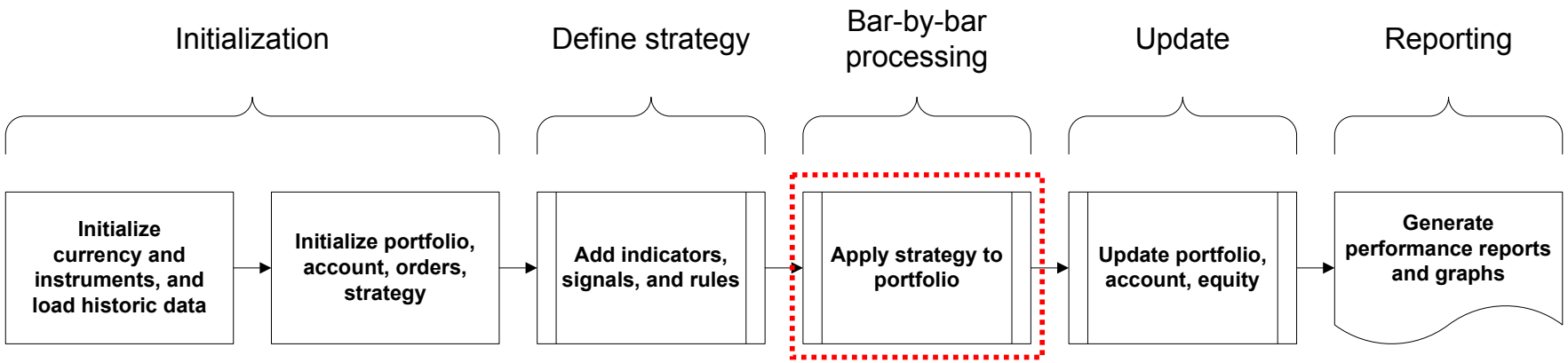
The applyStrategy function applies the strategy to a portfolio and generates transactions according to the strategy rules and the market data

```
args(applyStrategy)

## function (strategy, portfolios, mktdata = NULL, parameters = NULL,
##     ..., debug = FALSE, symbols = NULL, initStrat = FALSE, updateStrat = FALSE,
##     initBySymbol = FALSE, gc = FALSE, delorders = FALSE)
## NULL
```

Main arguments:

strategy	an object of type 'strategy'
portfolios	a list of portfolios to apply the strategy to
parameters	named list of parameters to be applied during evaluation of the strategy

Applying strategy to a multi-asset portfolio




```
nSD = 2
nMA = 20
```

```
out <- applyStrategy("bbands",
  portfolios="multiAsset.bb1",parameters=list(sd=nSD,n=nMA))
```

- Indicator parameters can be passed when applying the strategy; for this run the length of the moving average is 20 and the standard deviation multiplier is 2

Apply the strategy

Calling `applyStrategy` generates transactions in the specified portfolio.

```
getTxns(Portfolio="multiAsset.bb1", Symbol="XLK")
```

```
##            Txn.Qty Txn.Price Txn.Fees   Txn.Value Txn.Avg.Cost Net.Txn.Realized.PL
## 2009-12-31       0  0.000000        0      0.0000     0.000000            0.000000
## 2010-09-24     100 21.398266        0   2139.8266    21.398266            0.000000
## 2011-02-11     100 24.986291        0   2498.6291    24.986291            0.000000
## 2011-06-17    -200 22.878592        0  -4575.7184    22.878592          -62.737246
## 2012-02-03     100 26.151043        0   2615.1043    26.151043            0.000000
## 2012-03-23     100 28.383419        0   2838.3419    28.383419            0.000000
## 2012-08-24     100 29.044692        0   2904.4692    29.044692            0.000000
## 2012-09-14     100 30.004298        0   3000.4298    30.004298            0.000000
## 2012-11-23    -400 27.523743        0 -11009.4973    27.523743         -348.847836
## 2013-03-15     100 29.095924        0   2909.5924    29.095924            0.000000
## 2013-04-19     100 28.276998        0   2827.6998    28.276998            0.000000
## 2013-05-10     100 30.435106        0   3043.5106    30.435106            0.000000
## 2013-09-27     100 31.369303        0   3136.9303    31.369303            0.000000
## 2013-10-25     100 32.595273        0   3259.5273    32.595273            0.000000
## 2014-01-03     100 34.437751        0   3443.7751    34.437751            0.000000
## 2014-03-14     100 34.838644        0   3483.8644    34.838644            0.000000
## 2014-03-28     100 35.420604        0   3542.0604    35.420604            0.000000
## 2014-05-23     100 36.667740        0   3666.7740    36.667740            0.000000
## 2014-10-24    -900 38.887717        0 -34998.9453    38.887717         5685.210959
## 2014-11-14     100 41.225935        0   4122.5935    41.225935            0.000000
## 2015-03-13     100 41.320875        0   4132.0875    41.320875            0.000000
## 2015-05-01     100 43.099998        0   4309.9998    43.099998            0.000000
```

The mktdata object

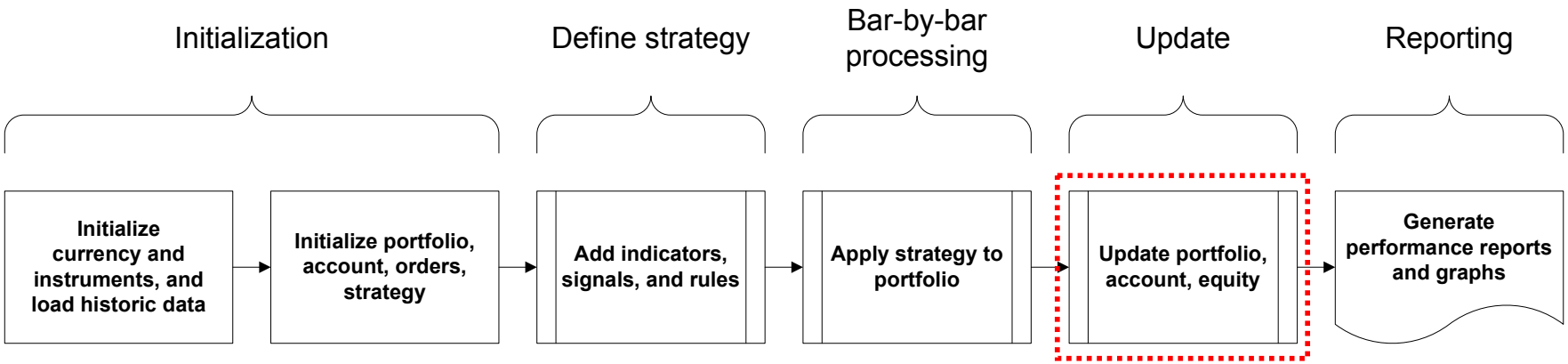
mktdata is a special variable constructed during the execution of applyStrategy. It is a time series object which contains the historic price data for the current symbol being evaluated as well as the calculated indicators and signals:

```
mktdata["2015"]
```

```
##            XLY.Open XLY.High XLY.Low XLY.Close XLY.Volume XLY.Adjusted dn.bbInd mavg.bbInd up.bbInd pctB.bbInd H.gt.UpperBand L.lt.LowerBand
## 2015-01-02  71.9484  72.7257 70.9019   71.3902   41914300      70.3154  63.5691    68.0868  72.6045   0.896858              NA             NA
## 2015-01-09  71.1211  71.5796 68.8587   70.6527   35195200      69.5890  63.6107    68.2271  72.8436   0.731405              NA             NA
## 2015-01-16  70.7922  71.4500 68.2707   69.4368   41348200      68.3914  63.6509    68.3115  72.9720   0.651023              NA             NA
## 2015-01-23  69.5165  71.0414 68.4202   70.6926   43466100      69.6283  63.6756    68.3976  73.1195   0.675122              NA             NA
## 2015-01-30  70.6627  71.1012 69.4069   69.7557   31099800      68.7055  63.7435    68.5124  73.2812   0.665193              NA             NA
## 2015-02-06  69.9351  73.0546 68.8288   72.6958   45344600      71.6014  63.7672    68.6986  73.6301   0.786714              NA             NA
## 2015-02-13  72.2872  74.6393 72.2175   74.6393   24706000      73.5156  63.6991    69.0409  74.3827   0.948459               1             NA
## 2015-02-20  74.5297  75.2074 74.1609   75.1875   15469200      74.0555  63.7775    69.4860  75.1944   0.970005              NA             NA
## 2015-02-27  75.1376  76.2639 74.8885   75.7157   20855600      74.5758  64.1102    70.0185  75.9268   0.974265              NA             NA
## 2015-03-06  75.7655  76.6526 75.0081   75.1576   50333400      74.0261  65.2124    70.6510  76.0895   0.955552              NA             NA
## 2015-03-13  75.2174  75.6758 73.9815   74.9084   28468000      73.7806  66.0189    71.1351  76.2512   0.863574              NA             NA
## 2015-03-20  75.1875  76.9300 74.6692   76.7700   28895300      75.6142  66.4826    71.6078  76.7330   0.940500               1             NA
## 2015-03-27  76.7200  77.1300 74.2300   74.9100   23348000      73.7822  67.0506    72.0176  76.9845   0.842842              NA             NA
## 2015-04-02  75.1600  76.1500 74.5300   75.6900   24538300      74.5505  67.5221    72.3802  77.2383   0.816631              NA             NA
## 2015-04-10  75.2500  76.7500 75.1600   76.6700   21610300      75.5157  67.7936    72.7200  77.6463   0.852528              NA             NA
## 2015-04-17  76.6000  76.8300 74.9900   75.2300   27277800      74.0974  67.9554    72.9627  77.9701   0.771659              NA             NA
## 2015-04-24  75.5500  77.6900 75.5500   77.6300   20444300      76.4612  68.0744    73.2689  78.4634   0.854971              NA             NA
## 2015-05-01  77.8200  77.8900 74.9700   76.3500   36547900      75.2005  68.4660    73.5930  78.7200   0.774074              NA             NA
```

- Inspecting mktdata can be very helpful in understanding strategy processing and debugging

Update portfolio and account



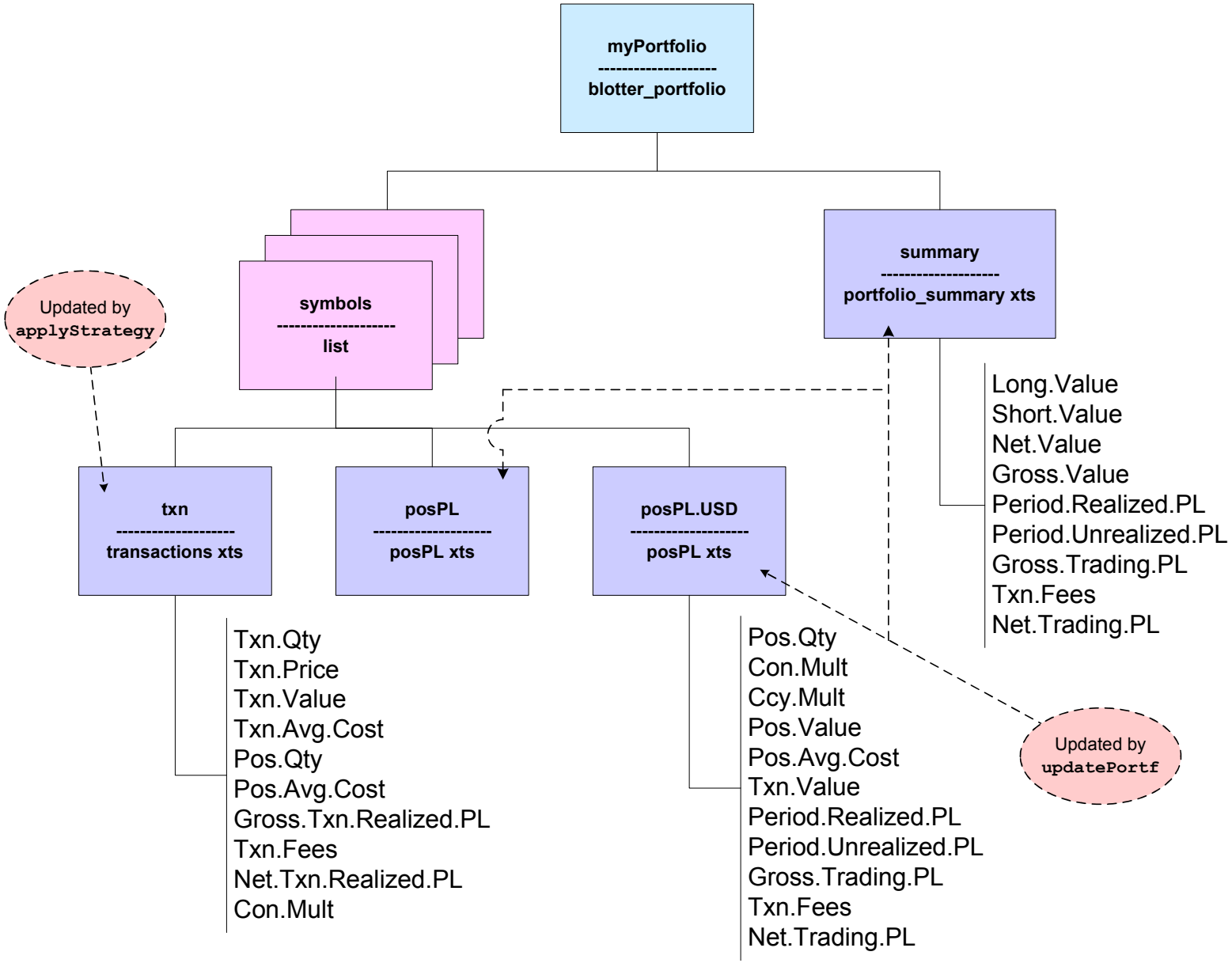

```
updatePortf("multiAsset.bb1")
updateAcct("multiAsset.bb1")
updateEndEq("multiAsset.bb1")
```

Data integrity check

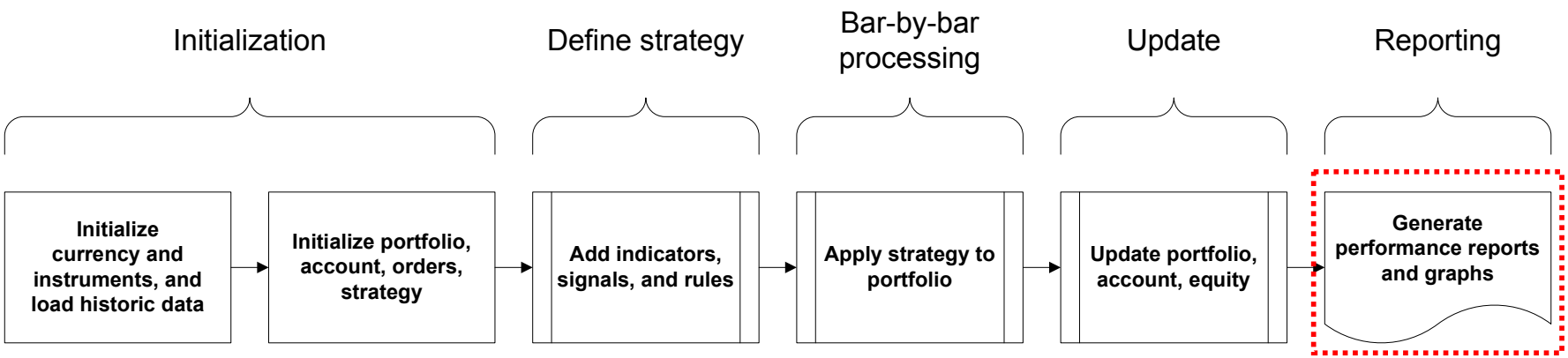
```
checkBlotterUpdate <- function(port.st,account.st,verbose=TRUE)
{
  ok <- TRUE
  p <- getPortfolio(port.st)
  a <- getAccount(account.st)
  syms <- names(p$symbols)
  port.tot <- sum(sapply(syms,FUN = function(x) eval(parse(
    text=paste("sum(p$symbols",x,"posPL.USD$Net.Trading.PL)",sep="$")))))
  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("multiAsset.bb1","multiAsset.bb1")
```

```
## [1] TRUE
```

How the blotter_portfolio object gets updated



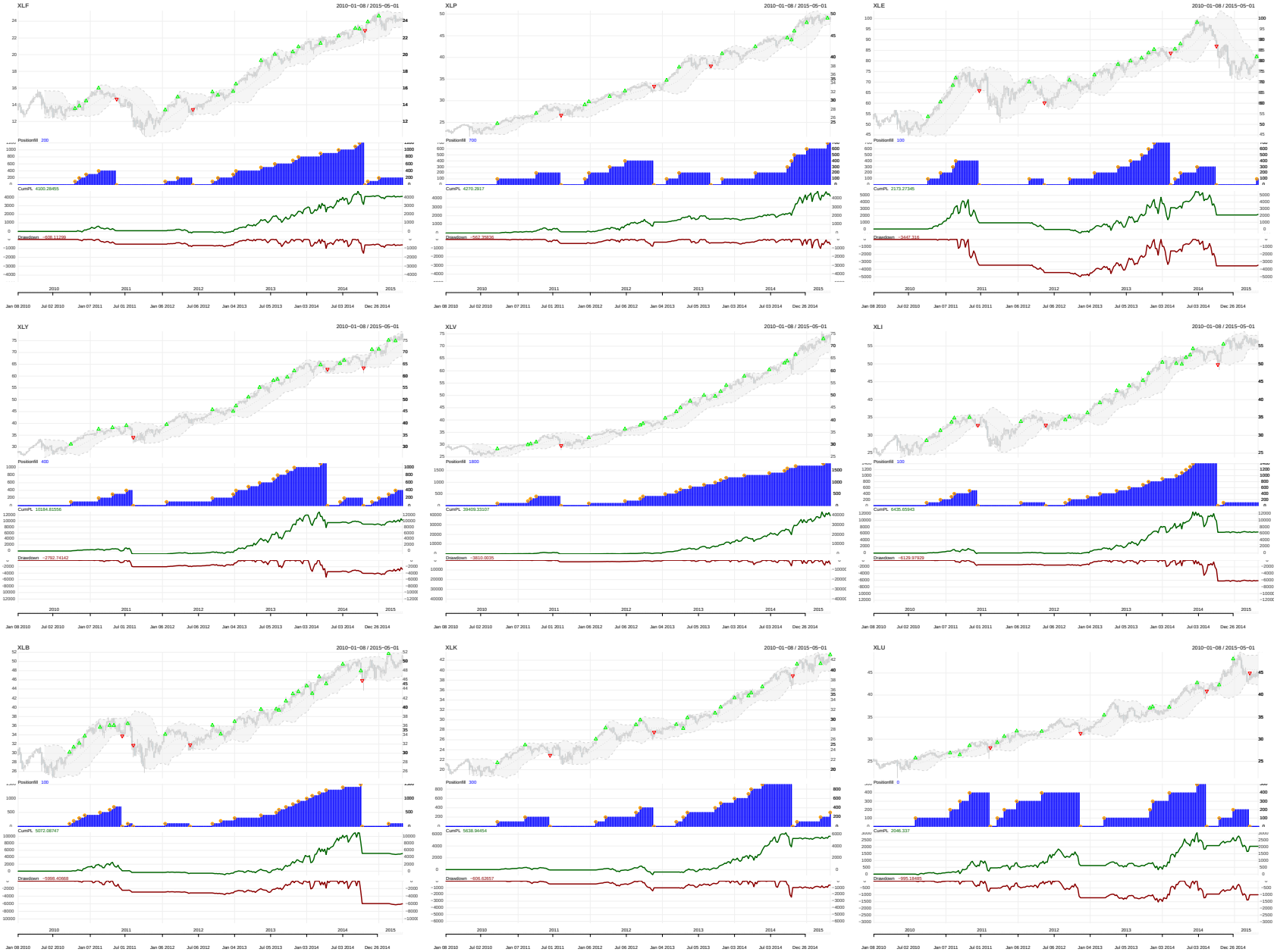
Generate position plots



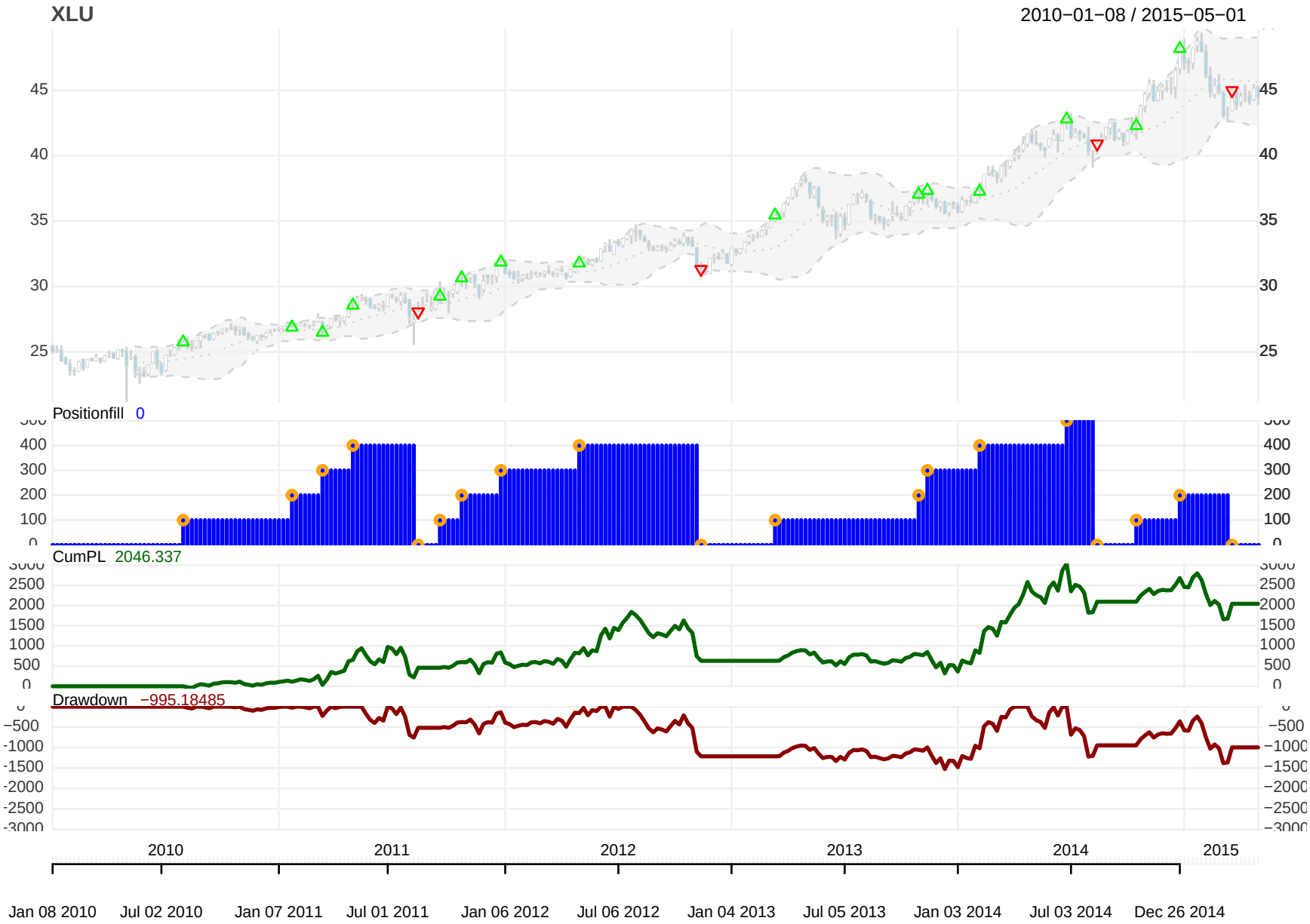

```
par(mfrow=c(3,3))
for(symbol in symbols)
{
chart.Posn(Portfolio="multiAsset.bb1",Symbol=symbol,theme=myTheme,
        TA="add_BBands(n=20,sd=2)")
}
par(mfrow=c(1,1))
```

```
chart.Posn("multiAsset.bb1","XLU",TA="add_BBands(n=20,sd=2)",theme=myTheme)
```

Position plots



BBands strategy for XLU



Trade stats by instrument

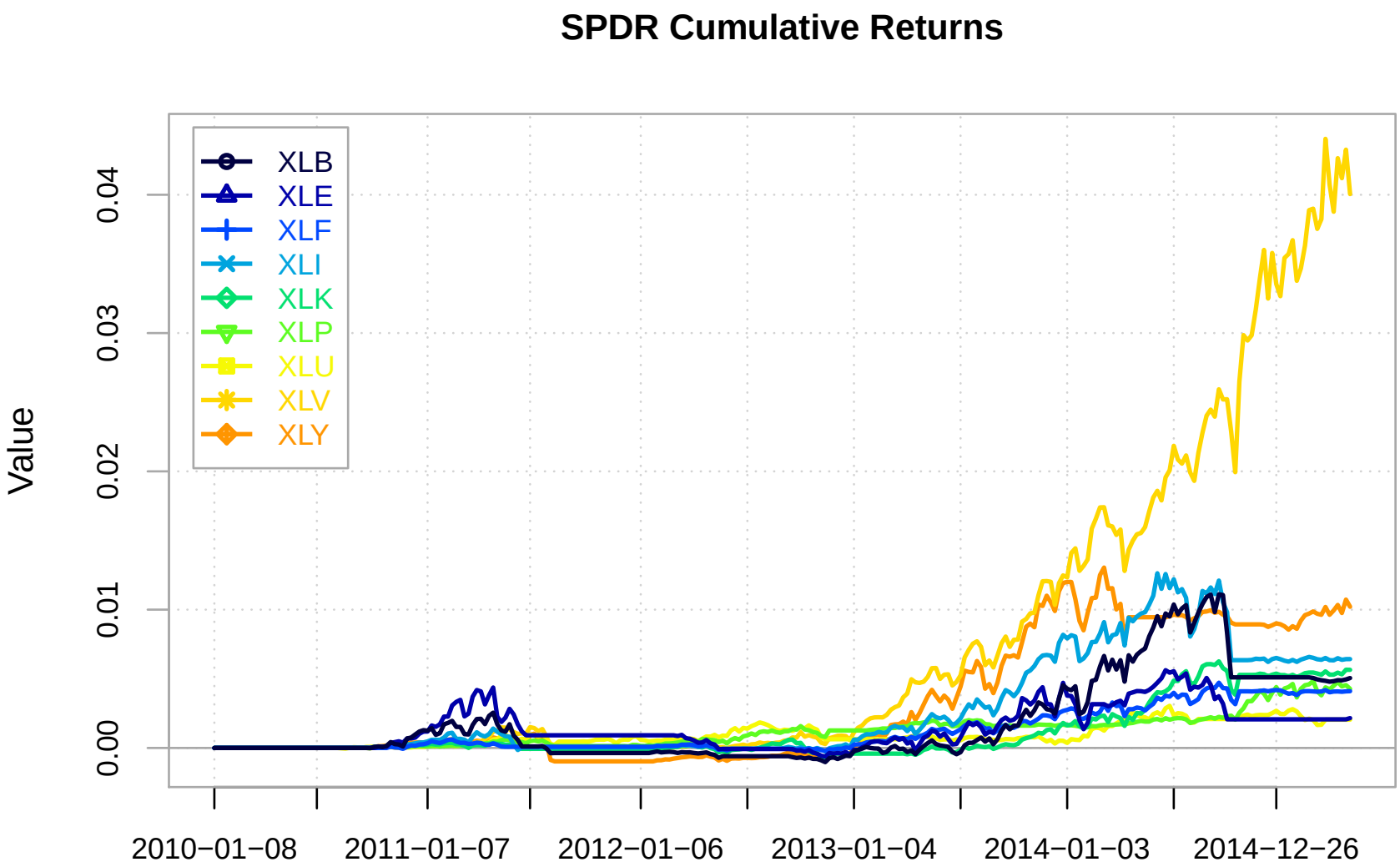
```
textplot(t(tradeStats("multiAsset.bb1")))
```

	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
Portfolio	multiAsset.bb1	multiAsset.bb1	multiAsset.bb1	multiAsset.bb1	multiAsset.bb1	multiAsset.bb1	multiAsset.bb1	multiAsset.bb1	multiAsset.bb1
Symbol	XLB	XLE	XLF	XLI	XLK	XLP	XLU	XLV	XLY
Num.Txns	29	20	23	24	21	18	19	23	24
Num.Trades	4	4	3	3	3	3	4	1	3
Net.Trading.PL	5072.0875	2173.2734	4100.2846	6435.6594	5638.9445	4270.2917	2046.3370	39409.3311	10184.8156
Avg.Trade.PL	1280.13029	521.06836	1363.29260	2118.93507	1757.87529	538.86292	511.58425	−113.49647	2973.66748
Med.Trade.PL	−60.809191	−34.599290	83.914102	43.796927	−62.737246	352.243542	315.593676	−113.496467	−510.664586
Largest.Winner	5714.8446	3241.2778	4152.6639	6413.8447	5685.2110	1120.6504	1464.9367	0.0000	10401.7824
Largest.Loser	−472.70501	−1087.80580	−146.70017	−100.83641	−348.84784	0.00000	−49.78702	−113.49647	−970.11536
Gross.Profits	5822.7486	4161.5762	4236.5780	6457.6416	5685.2110	1616.5888	2096.1240	0.0000	10401.7824
Gross.Losses	−702.22739	−2077.30279	−146.70017	−100.83641	−411.58508	0.00000	−49.78702	−113.49647	−1480.77995
Std.Dev.Trade.PL	2966.04607	2035.45310	2418.41680	3720.20379	3404.17962	514.51981	668.85308		6437.03674
Percent.Positive	50.000000	50.000000	66.666667	66.666667	33.333333	100.000000	75.000000	0.000000	33.333333
Percent.Negative	50.000000	50.000000	33.333333	33.333333	66.666667	0.000000	25.000000	100.000000	66.666667
Profit.Factor	8.2918278	2.0033556	28.8791615	64.0407751	13.8129665		42.1018172	0.0000000	7.0245295
Avg.Win.Trade	2911.37428	2080.78812	2118.28899	3228.82082	5685.21096	538.86292	698.70801		10401.78238
Med.Win.Trade	2911.37428	2080.78812	2118.28899	3228.82082	5685.21096	352.24354	459.20806		10401.78238
Avg.Losing.Trade	−351.11369	−1038.65139	−146.70017	−100.83641	−205.79254		−49.78702	−113.49647	−740.38997
Med.Losing.Trade	−351.11369	−1038.65139	−146.70017	−100.83641	−205.79254		−49.78702	−113.49647	−740.38997
Avg.Daily.PL	1280.13029	521.06836	1363.29260	2118.93507	1757.87529	538.86292	511.58425	−113.49647	2973.66748
Med.Daily.PL	−60.809191	−34.599290	83.914102	43.796927	−62.737246	352.243542	315.593676	−113.496467	−510.664586
Std.Dev.Daily.PL	2966.04607	2035.45310	2418.41680	3720.20379	3404.17962	514.51981	668.85308		6437.03674
Ann.Sharpe	6.8513563	4.0638145	8.9486639	9.0417228	8.1973950	16.6255673	12.1419015		7.3334221
Max.Drawdown	−6262.4066	−4979.4775	−1546.2992	−6336.5265	−2398.6520	−1031.6813	−1525.9099	−5849.9213	−5205.4559
Profit.To.Max.Draw	0.80992625	0.43644609	2.65167606	1.01564468	2.35088059	4.13915769	1.34106015	6.73672840	1.95656553
Avg.WinLoss.Ratio	8.2918278	2.0033556	14.4395807	32.0203876	27.6259330		14.0339391		14.0490590
Med.WinLoss.Ratio	8.2918278	2.0033556	14.4395807	32.0203876	27.6259330		9.2234493		14.0490590
Max.Equity	11070.4942	5620.5894	4708.3975	12565.6387	6245.5711	4832.6501	3041.5218	43219.3346	12977.5570
Min.Equity	−1023.9874431	−616.4251266	−202.9135129	−252.5894445	−865.7029954	−9.7115518	−43.2139725	−113.4964667	−970.1153599
End.Equity	5072.0875	2173.2734	4100.2846	6435.6594	5638.9445	4270.2917	2046.3370	39409.3311	10184.8156

Individual asset returns

```
rets.multi <- PortfReturns("multiAsset.bb1")
colnames(rets.multi) <- sort(symbols)
round(tail(rets.multi,5),6)

##                  XLB     XLE      XLF       XLI       XLK       XLP XLU
## 2015-04-02  3.6e-05 0.0e+00  5.6e-05 -0.000012  0.000000  0.000252   0
## 2015-04-10  7.3e-05 0.0e+00  6.0e-06  0.000154  0.000142  0.000282   0
## 2015-04-17 -8.0e-06 0.0e+00 -2.2e-05 -0.000117 -0.000122 -0.000324   0
## 2015-04-24  6.4e-05 0.0e+00  3.0e-05  0.000052  0.000332  0.000091   0
## 2015-05-01  9.9e-05 8.9e-05  1.2e-05  0.000001 -0.000006 -0.000266   0
##                  XLV       XLY
## 2015-04-02 -0.001890  0.000312
## 2015-04-10  0.003726  0.000392
## 2015-04-17 -0.001386 -0.000576
## 2015-04-24  0.001980  0.000960
## 2015-05-01 -0.003078 -0.000512

chart.CumReturns(rets.multi, colorset= rich10equal, legend.loc = "topleft",
  main="SPDR Cumulative Returns",minor.ticks=FALSE)
```

Cumulative returns by asset



Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
   - Position limits
   - User-supplied order sizing function
6. Stop orders
7. Parameter optimization

Position Sizing Methods

There are 5 primary position sizing scenarios:

- Fixed order size with rules that prohibit pyramiding
- Fixed order size with rules that allow pyramiding (no fixed position size)
- Order size and position limit controlled via `addPosLimit`
- Order size controlled via user-supplied order sizing function
  - `osFUN` argument of `ruleSignal`
- Order/position size determined as a percent of account equity
  - `applyStrategy.rebalancing`

Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
   - Position limits
   - User-supplied order sizing function
6. Stop orders
7. Parameter optimization

Position limits and levels

- Position limits are set for the portfolio as a run-time parameter
- The function `osMaxPos` implements simple levels[†] based maximum positions
- The position sizing function `osMaxPos` must be passed via the `osFUN` argument of `ruleSignal`
- The maximum position and levels are accessed via the functions `addPosLimit` and `getPosLimit`

[†]The level is the number of pyramiding orders needed to reach the position limit

The ruleSignal function

ruleSignal is the default rule to generate a trade order on a signal

```
args(ruleSignal)

## function (mktdata = mktdata, timestamp, sigcol, sigval, orderqty = 0,
##     ordertype, orderside = NULL, orderset = NULL, threshold = NULL,
##     tmult = FALSE, replace = TRUE, delay = 1e-04, osFUN = "osNoOp",
##     pricemethod = c("market", "opside", "active"), portfolio,
##     symbol, ..., ruletype, TxnFees = 0, prefer = NULL, sethold = FALSE,
##     label = "", order.price = NULL, chain.price = NULL, time.in.force = "")
## NULL
```

Main arguments:

sigcol	column name to check for signal
sigval	signal value to match
orderqty	quantity for order or 'all', modified by osFUN
ordertype	"market","limit","stoplimit","stoptrailing","iceberg"
orderside	"long", "short", or NULL
osFUN	function or name of order sizing function (default is osNoOp)

Add rules with an order sizing function specified

```
enable.rule("bbands",type="enter",label="LongEntry",enabled=FALSE)
```

```
add.rule("bbands", name='ruleSignal',
  arguments=list(sigcol="H.gt.UpperBand",sigval=TRUE,
    orderqty=+100, ordertype='market', orderside='long',
    osFUN='osMaxPos'),
  type='enter',
  label='LimitedLongEntry')
```

- Use function `enable.rule` to enable and disable strategy rules
- The `ruleSignal` argument `osFUN` is set to `osMaxPos`

The addPosLimit function

The function addPosLimit adds position and level limits to a strategy

```
args(addPosLimit)

## function (portfolio, symbol, timestamp, maxpos, longlevels = 1,
##     minpos = -maxpos, shortlevels = longlevels)
## NULL
```

Main arguments:

portfolio	text name of the portfolio
symbol	instrument identifier
maxpos	maximum long position size
longlevels	number of levels

- Setting levels to 1 results in an order size of the maximum size

Initialize portfolio and add position limits

Position limits apply to individual assets in the portfolio

```
rm.strat("multi.bb.limit") # remove portfolio, account, orderbook if re-run
initPortf(name="multi.bb.limit", symbols, initDate=initDate)
initAcct(name="multi.bb.limit", portfolios="multi.bb.limit",
  initDate=initDate, initEq=initEq)
initOrders(portfolio="multi.bb.limit", initDate=initDate)
```

```
for(symbol in symbols)
{
  addPosLimit("multi.bb.limit", symbol, initDate, 100, 1 )
}
```

- Position limits are separated from the strategy and are a run-time constraint to the portfolio

Applying, update, and plot

```
out <- applyStrategy("bbands",
  portfolios="multi.bb.limit",parameters=list(sd=2,n=20))
```

```
updatePortf("multi.bb.limit")
updateAcct("multi.bb.limit")
updateEndEq("multi.bb.limit")
```

```
checkBlotterUpdate("multi.bb.limit","multi.bb.limit")

## [1] TRUE
```

```
chart.Posn("multi.bb.limit","XLU",TA="add_BBands(n=20,sd=2)",theme=myTheme)
```

BBands strategy for XLU with position limit

Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
   - Position limits
   - User-supplied order sizing function
6. Stop orders
7. Parameter optimization

The osNoOp function

The function osNoOp is the default order sizing function

```
args(osNoOp)

## function (timestamp, orderqty, portfolio, symbol, ruletype, ...)
## NULL
```

Main arguments:

timestamp	timestamp (coercible into a POSIXct object) that will mark the time of order insertion
orderqty	the order quantity; modified by osFUN
portfolio	name of the portfolio for the order
symbol	symbol of instrument
ruletype	one of "risk", "order", "rebalance", "enter", "exit"

Define order sizing function

```
osFixedDollar <- function(timestamp, orderqty, portfolio, symbol, ruletype, ...)
{
  pos <- getPosQty(portfolio, symbol, timestamp)
  if( isTRUE(all.equal(pos,0)) )
  {
    ClosePrice <- as.numeric(Cl(mktdata[timestamp,]))
    orderqty <- sign(orderqty)*round(tradeSize/ClosePrice,-2)
  } else {
    orderqty <- 0
  }
  return(orderqty)
}
```

- Fixed dollar order size:

$$\text{orderqty} = \frac{\text{tradeSize}}{\mathit{ClosePrice}}$$

Add rules with an order sizing function specified

```
enable.rule("bbands",type="enter",label="LimitedLongEntry",enabled=FALSE)
```

```
add.rule("bbands", name='ruleSignal',
  arguments=list(sigcol="H.gt.UpperBand",sigval=TRUE,
    orderqty=+100, ordertype='market', orderside='long',
    osFUN='osFixedDollar'),
  type='enter',
  label='FixedLongEntry')
```

- Use function `enable.rule` to enable and disable strategy rules
- The `ruleSignal` argument `osFUN` is set to `osFixedDollar`

Initialize, applying, and update

```
rm.strat("fixed.dollar") # remove portfolio, account, orderbook if re-run
initPortf(name="fixed.dollar", symbols, initDate=initDate)
initAcct(name="fixed.dollar", portfolios="fixed.dollar",
  initDate=initDate, initEq=initEq)
initOrders(portfolio="fixed.dollar", initDate=initDate)
```

```
tradeSize <- 100000
out <- applyStrategy("bbands",
  portfolios="fixed.dollar",parameters=list(sd=2,n=20))
```

```
updatePortf("fixed.dollar")
updateAcct("fixed.dollar")
updateEndEq("fixed.dollar")
```

```
checkBlotterUpdate("fixed.dollar","fixed.dollar")

## [1] TRUE
```

Per-trade statistics

```
perTradeStats("fixed.dollar","XLF")
```

```
##        Start        End Init.Pos Max.Pos Num.Txns Max.Notional.Cost Net.Trading.PL        MAE        MFE
## 1 2010-10-22 2011-05-20     7500    7500        2         101580.87      8476.3772 -1252.3668 18384.0127
## 2 2012-01-20 2012-06-08     7600    7600        2         101714.21       324.7580 -4365.8591 12303.8725
## 3 2012-09-14 2014-10-24     6700    6700        2         104067.59     49343.1187 -5986.5116 52995.7547
## 4 2014-11-07 2015-05-01     4200    4200        1         100539.18      1688.8224 -4256.5737  3861.1294
##   Pct.Net.Trading.PL      Pct.MAE     Pct.MFE tick.Net.Trading.PL    tick.MAE   tick.MFE
## 1        0.083444625 -0.012328767 0.180979092         113.0183632  -16.698224 245.120170
## 2        0.003192848 -0.042922806 0.120965134           4.2731316  -57.445515 161.893059
## 3        0.474144901 -0.057525223 0.509243589         736.4644582  -89.350919 790.981413
## 4        0.016797655 -0.042337463 0.038404227          40.2100569 -101.346993  91.931652
```

- Each order is approximately $100,000 in value

Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
6. Stop orders
7. Parameter optimization

Ordersets and order chains

To implement stop-loss or trailing-stop orders, quantstrat utilizes the concept of ordersets and order chains:

orderset	An orderset is a collection of OCO orders
OCO order	One-Cancels-Other (OCO) orders are grouped orders such that when one is filled, all others in the orderset are cancelled
order chain	An order chain defines an order (child) which will be created when another order (parent) is filled

The ruleSignal function

Stoplimit-related arguments:

orderset	A tag identifying the orderset; if one order of the set is filled, all others are canceled
threshold	A numeric or name of indicator column in mktdata
tmult	If TRUE, threshold is a percent multiplier for price, not a scalar
replace	If an orderset is specified and replace=TRUE, all open orders for the orderset will be replaced
prefer	The preferred order price

Define indicators and signals

```
strategy("bbands", store=TRUE)
```

```
add.indicator("bbands", name = "BBands",
  arguments = list(HLC = quote(HLC(mktdata)), maType='SMA'), label='bbInd')
```

```
add.signal("bbands", name="sigCrossover",
  arguments=list(columns=c("High","up"),relationship="gt"),
  label="H.gt.UpperBand")
```

```
add.signal("bbands", name="sigCrossover",
  arguments=list(columns=c("Low","dn"),relationship="lt"),
  label="L.lt.LowerBand")
```

Add rules

```
add.rule("bbands", name='ruleSignal',
  arguments=list(sigcol="H.gt.UpperBand",sigval=TRUE,
    orderqty=+100,
    ordertype='market',
    orderside='long',
    osFUN='osFixedDollar',
    orderset='ocolong'),
  type='enter',
  label='LongEntry')
```

```
add.rule("bbands", name='ruleSignal',
  arguments=list(sigcol="L.lt.LowerBand",sigval=TRUE,
    orderqty= 'all',
    ordertype='market',
    orderside='long',
    orderset='ocolong'),
  type='exit',
  label='LongExit')
```

Long stop loss

```
stopLossPercent <- 0.03
```

```
add.rule("bbands",name='ruleSignal',
  arguments = list(sigcol="H.gt.UpperBand", sigval=TRUE,
    replace=FALSE,
    orderside='long',
    ordertype='stoplimit',
    tmult=TRUE,
    threshold=quote( stopLossPercent ),
    orderqty='all',
    orderset='ocolong'
  ),
  type='chain', parent="LongEntry",
  label='StopLossLong'
)
```

- Belongs to orderset `ocolong`
- Rule type is `'chain'` and parent is 'LongEntry'

Trailing stop loss

```
trailingStopPercent <- 0.07
```

```
add.rule("bbands", name = 'ruleSignal',
  arguments=list(sigcol="H.gt.UpperBand" , sigval=TRUE,
    replace=FALSE,
    orderside='long',
    ordertype='stoptrailing',
    tmult=TRUE,
    threshold=quote(trailingStopPercent),
    orderqty='all',
    orderset='ocolong'
  ),
  type='chain', parent="LongEntry",
  label='StopLossTrailing'
)
```

- Belongs to orderset `ocolong`
- Rule type is `'chain'` and parent is 'LongEntry'

Apply stoplosses

```
rm.strat("bb.stop") # remove portfolio, account, orderbook if re-run
```

```
initPortf(name="bb.stop", symbols, initDate=initDate)
initAcct(name="bb.stop", portfolios="bb.stop",
  initDate=initDate, initEq=initEq)
initOrders(portfolio="bb.stop", initDate=initDate)
```

```
tradeSize <- 100000
out<-applyStrategy("bbands" , portfolios="bb.stop",
  parameters=list(sd=2,n=20))
```

```
updatePortf("bb.stop")
updateAcct("bb.stop")
updateEndEq("bb.stop")
```

```
checkBlotterUpdate("bb.stop","bb.stop")

## [1] TRUE
```

Outline

1. Preliminaries: quantitative analysis packages
2. Preliminaries: time-series objects
3. Introduction to blotter and quantstrat
4. Basic quantstrat strategy example
5. Position sizing
6. Stop orders
7. Parameter optimization

Parallel computing with foreach

- The foreach package facilitates easily-accessible parallel processing in R
- The foreach function is a for-like looping construct where each iteration of the for loop can be run in parallel if a multicore processor (now very common) is available
- Each loop iteration returns a result and these results can be combined in a variety of ways depending on their data type
- foreach requires that you register a *parallel backend*
  - On Windows platforms, doParallel is the recommend parallel backend
  - On Linux/Mac platforms, doMC is the recommend parallel backend
  - doSNOW is a parallel backend that can run on both Windows and Linux

Setup parallel backend and test foreach

```
library(parallel)
detectCores()

## [1] 8
```

```
if( Sys.info()['sysname'] == "Windows" )
{
  library(doParallel)
  registerDoParallel(cores=detectCores())
} else {
  library(doMC)
  registerDoMC(cores=detectCores())
}
```

```
foreach(i=1:8, .combine=c) %dopar% sqrt(i)

## [1] 1.0000000 1.4142136 1.7320508 2.0000000 2.2360680 2.4494897 2.6457513
## [8] 2.8284271
```

- All sqrt operations are run in parallel via separate processes on a multi-core processor

Optimization in quantstrat

Optimization in quantstrat is implemented using a concept call a paramset; along with paramsets, there are distributions and constraints.

paramset A paramset is a collection of variables that will be optimized subject to their range of allowed values (distribution) and any constraints between them

distribution A distribution in a paramset is simply the range of values that a variable is allowed to take (e.g. fastMA $=$ 1:20)

constraint A constraint is a relationship that must be true between two distributions in a paramset (e.g. fastMA $<$ slowMA)

Optimization functions in quantstrat

The following functions implement parameter optimization in quantstrat:

add.distribution	Creates a distribution in paramset, where a distribution consists of the name of a variable in a strategy component plus a range of values for this variable.
add.distribution.constraint	Creates a constraint on 2 distributions in a paramset, i.e. a restriction limiting the allowed combinations from the ranges for distribution 1 and distribution 2.
apply.paramset	Runs `applyStrategy` once for each parameter combination as specified by the parameter distributions and constraints in the paramset. `apply.paramset` will do parallel processing on multiple cores if available.

Optimization range for stop loss

```
args(add.distribution)

## function (strategy, paramset.label, component.type, component.label,
##     variable, weight = NULL, label, store = TRUE)
## NULL
```

```
stopLossPercentRange <- seq(0.01,0.10,by=0.01)
```

```
add.distribution("bbands",
  paramset.label = "STOPOPT",
  component.type = "chain",
  component.label = "StopLossLong",
  variable = list( threshold = stopLossPercentRange ),
  label = "StopLossLongDist"
)
```

Optimization range for stop loss

```
trailingPercentRange <- seq(0.01,0.10,by=0.01)
```

```
add.distribution("bbands",
  paramset.label = "STOPOPT",
  component.type = "chain",
  component.label = "StopLossTrailing",
  variable = list( threshold = trailingPercentRange ),
  label = "StopLossTrailingDist"
)
```

Define parameter constraint

```
args(add.distribution.constraint)

## function (strategy, paramset.label, distribution.label.1, distribution.label.2,
##     operator, label, store = TRUE)
## NULL
```

```
add.distribution.constraint("bbands",
        paramset.label = 'STOPOPT',
        distribution.label.1 = 'StopLossLongDist',
        distribution.label.2 = 'StopLossTrailingDist',
        operator = '<',
        label = 'StopCon'
)
```

- StopLossLong must be less than StopLossTrailing

Initialize portfolio, account, and orders

```
rm.strat("bb.opt") # remove portfolio, account, orderbook if re-run
```

```
initPortf(name="bb.opt", symbols, initDate=initDate)
initAcct(name="bb.opt", portfolios="bb.opt",
  initDate=initDate, initEq=initEq)
initOrders(portfolio="bb.opt", initDate=initDate)
```

The apply.paramset function

The function apply.paramset function will run applyStrategy() on portfolio.st, once for each parameter combination as specified by the parameter distributions and constraints in the paramset

```
args(apply.paramset)

## function (strategy.st, paramset.label, portfolio.st, account.st,
##     mktdata = NULL, nsamples = 0, user.func = NULL, user.args = NULL,
##     calc = "slave", audit = NULL, packages = NULL, verbose = FALSE,
##     paramsets, ...)
## NULL
```

Main arguments:

strategy.st	text name of the strategy
paramset.label	text name of the paramset
portfolio.st	text name of the portfolio
nsamples	if nsamples $>$ 0 then take a sample of size nsamples from the paramset

Apply strategy and verify

```
if( Sys.info()['sysname'] == "Windows" )
{
  library(doParallel)
#  registerDoParallel(cores=detectCores())
  registerDoSEQ()
} else {
  library(doMC)
  registerDoMC(cores=detectCores())
}
```

```
results <- apply.paramset("bbands", paramset.label = "STOPOPT",
  portfolio="bb.opt", account="bb.opt", nsamples=0)
```

As of 2015-05-26, apply.paramset does not appear to run properly in parallel on Windows. To run on a Windows platform, load the doParallel package but do not call the registerDoParallel function; apply.paramset will then be able to run in sequential rather than parallel mode.

Results returns from apply.paramset

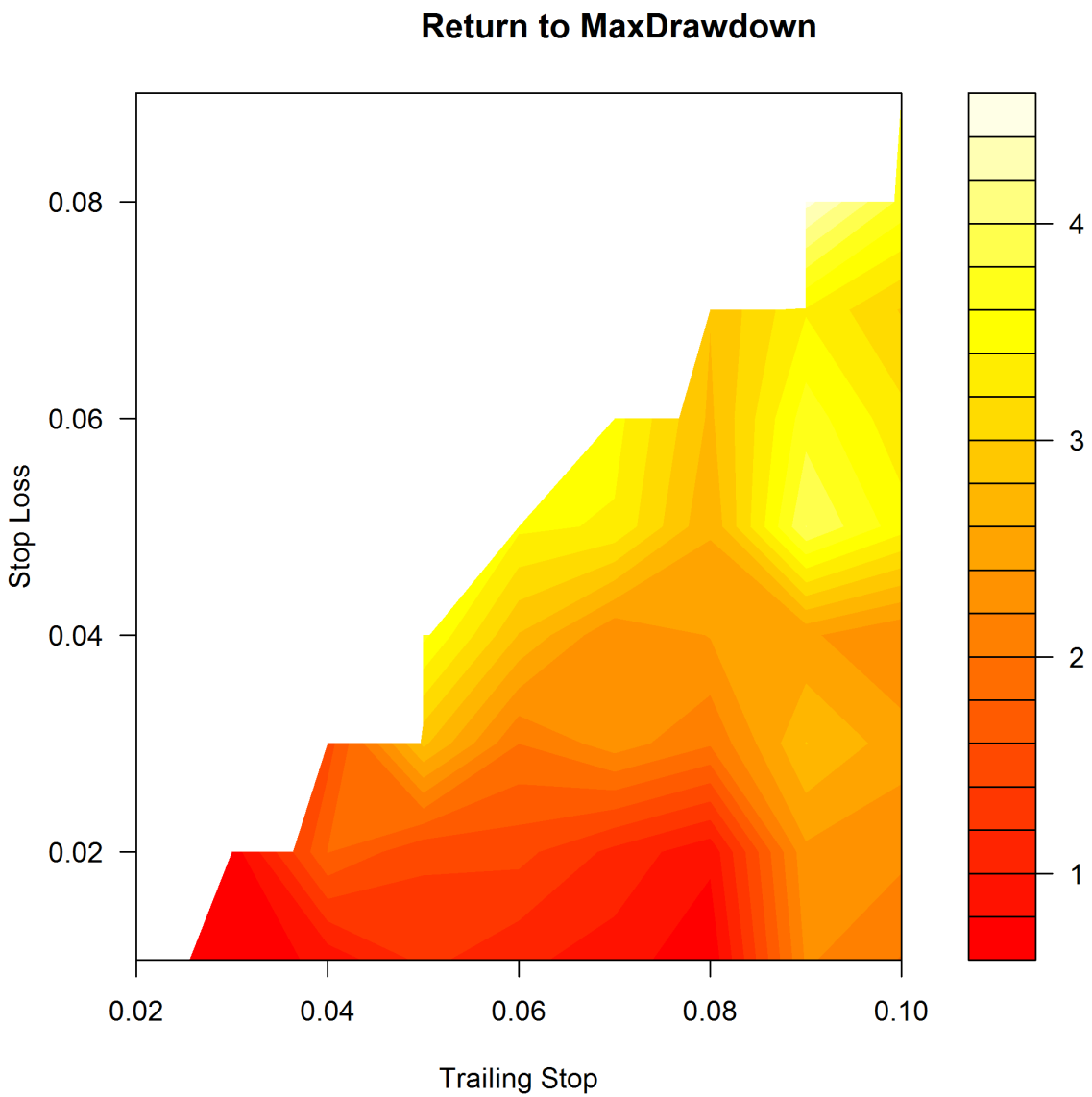
```
names(results)
```

```
##  [1] "bb.opt.1"   "tradeStats" "bb.opt.2"   "bb.opt.3"   "bb.opt.4"   "bb.opt.5"
##  [7] "bb.opt.6"   "bb.opt.7"   "bb.opt.8"   "bb.opt.9"   "bb.opt.10"  "bb.opt.11
## [13] "bb.opt.12"  "bb.opt.13"  "bb.opt.14"  "bb.opt.15"  "bb.opt.16"  "bb.opt.17
## [19] "bb.opt.18"  "bb.opt.19"  "bb.opt.20"  "bb.opt.21"  "bb.opt.22"  "bb.opt.23
## [25] "bb.opt.24"  "bb.opt.25"  "bb.opt.26"  "bb.opt.27"  "bb.opt.28"  "bb.opt.29
## [31] "bb.opt.30"  "bb.opt.31"  "bb.opt.32"  "bb.opt.33"  "bb.opt.34"  "bb.opt.35
## [37] "bb.opt.36"  "bb.opt.37"  "bb.opt.38"  "bb.opt.39"  "bb.opt.40"  "bb.opt.41
## [43] "bb.opt.42"  "bb.opt.43"  "bb.opt.44"  "bb.opt.45"
```

Heatmaps of strategy performance

```
z <- tapply(X=results$tradeStats$Profit.To.Max.Draw,
  INDEX=list(results$tradeStats$StopLossTrailingDist,results$tradeStats$StopLossLong
  FUN=median)
x <- as.numeric(rownames(z))
y <- as.numeric(colnames(z))

filled.contour(x=x,y=y,z=z,color = heat.colors,
  xlab="Trailing Stop",ylab="Stop Loss")
title("Return to MaxDrawdown")
```

Return to maximum drawdown



Lecture references

- TradeAnalytics project page on R-forge:
  `http://r-forge.r-project.org/projects/blotter/`
  - documents and demos for:
    - blotter package
    - quantstrat package

- Using quantstrat by Jan Humme & Brian Peterson
  `http://www.rinfinance.com/agenda/2013/workshop/Humme+Peterson.pdf`

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

[†] demos are located in the directory: `.../R-3.x.x/library/quantstrat/demo`

Conclusion

- Questions
- Download presentation and code:
  `https://github.com/gyollin/quantstrat-tutorial.git`
- Thank you for attending