Getting started with quantstrat

Guy Yollin

University of Washington

Joshua Ulrich

DV Trading

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
  - Software Engineering
    - r-programming.org
    - Insightful Corporation
    - Electro Scientific Industries, Vision Products Division
  - Hedge Fund
    - Rotella Capital Management
    - J.E. Moody, LLC
  - Academic
    - University of Washington
    - Oregon Graduate Institute
- Education
  - Oregon Graduate Institute, Computational Finance
  - Drexel University, Electrical Engineering
- Contact Info
  - `gyollin@uw.edu`
  - `http://www.linkedin.com/in/guyyollin`

Joshua Ulrich

- Professional Experience
  - Programming
    - fossfinance.com
    - R packages: TTR, xts, quantmod, blotter, quantstrat, pack, LSPM
  - Finance
    - DV Trading
    - Enterprise Bank & Trust
    - Wells Fargo Home Mortgage
- Education
  - University of Missouri–St. Louis, Economics
- Contact Info
  - `josh.m.ulrich@gmail.com`
  - `http://about.me/joshuaulrich`

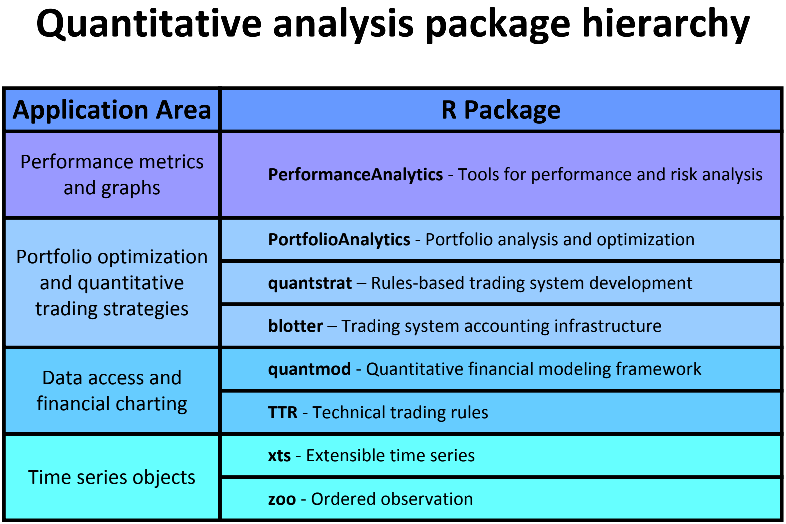
Outline

1. Introduction
2. Basic quantstrat strategy example
3. Position sizing
   - Position limits
   - User-supplied order sizing function
4. Stop orders
5. Parameter optimization

Outline

1. Introduction
2. Basic quantstrat strategy example
3. Position sizing
4. Stop orders
5. Parameter optimization

Packages for quantitative finance in R



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

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

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

Selected technical analysis indicators in TTR

Function	Description	Function	Description
stoch	stochastic oscillator	ADX	Directional Movement Index
aroon	Aroon indicator	ATR	Average True Range
BBands	Bollinger bands	CCI	Commodity Channel Index
chaikinAD	Chaikin Acc/Dist	chaikinVolatility	Chaikin Volatility
ROC	rate of change	momentum	momentum indicator
CLV	Close Location Value	CMF	Chaikin Money Flow
CMO	Chande Momentum Oscillator	SMA	simple moving average
EMA	exponential moving average	DEMA	double exp mov avg
VWMA	volume weighted MA	VWAP	volume weighed avg price
DonchianChannel	Donchian Channel	DPO	Detrended Price Oscillator
EMV	Ease of Movement Value	volatility	volatility estimators
MACD	MA converge/diverge	MFI	Money Flow Index
RSI	Relative Strength Index	SAR	Parabolic Stop-and-Reverse
TDI	Trend Detection Index	TRIX	Triple Smoothed Exponential Osc
VHF	Vertical Horizontal Filter	williamsAD	Williams Acc/Dist
WPR	William's % R	ZigZag	Zig Zag trend line

see Technical Analysis from A to Z by Steven Achelis

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`

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")
```

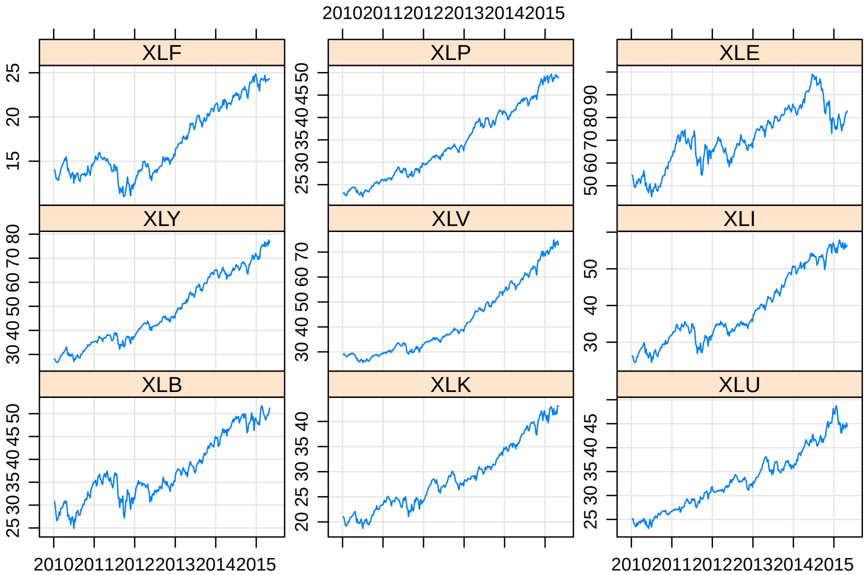
```
  getSymbols(symbols, from=startDate, to=endDate, index.class="POSIXct")
  for(symbol in symbols) {
      x<-get(symbol)
      x<-adjustOHLC(x,symbol.name=symbol)
      x<-to.weekly(x,indexAt='lastof',drop.time=TRUE)
      indexFormat(x)<-'%Y-%m-%d'
      colnames(x)<-gsub("x",symbol,colnames(x))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

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

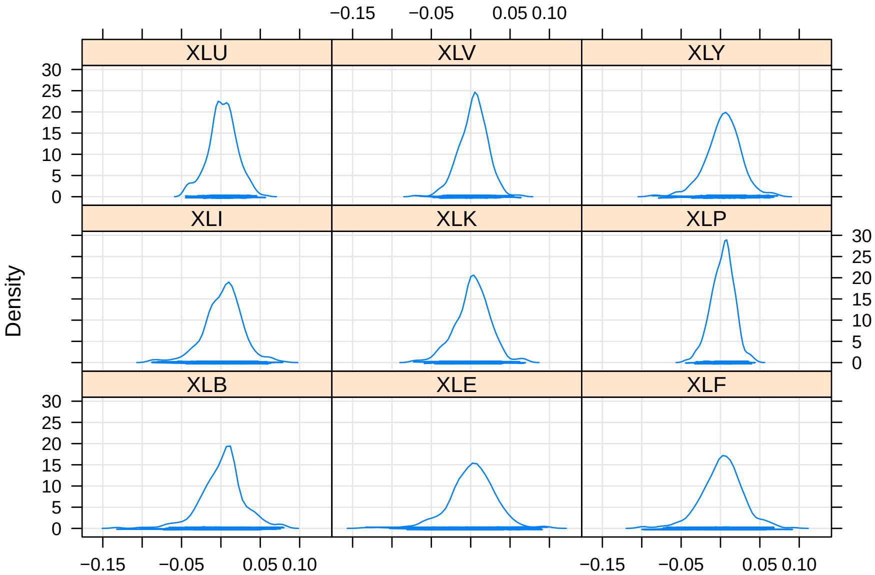
Compute returns

```r
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

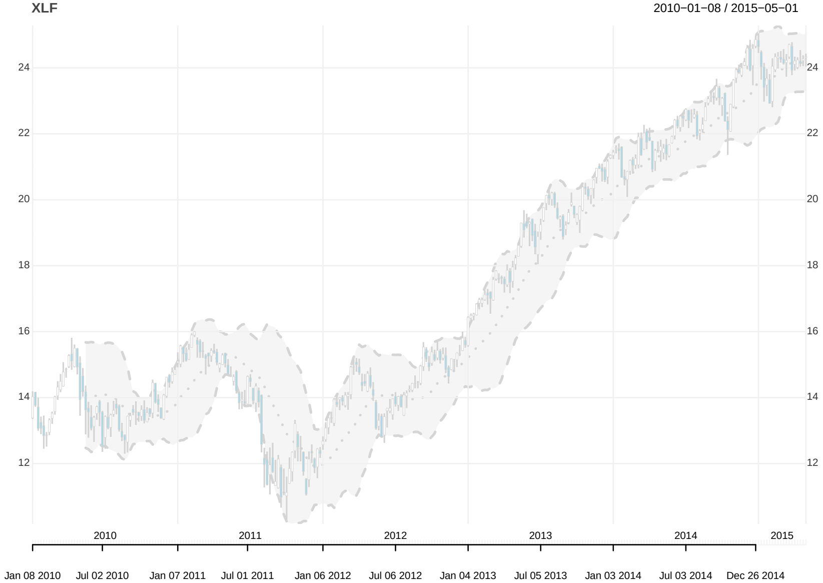
Pyramiding:

- 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 18.990975 19.855600 20.720225 1.04839408
## 2013-11-29 18.936071 19.924146 20.912221 1.04058184
## 2013-12-06 18.901882 19.966600 21.031319 0.90694374
## 2013-12-13 18.885953 19.998509 21.111064 0.81960137
## 2013-12-20 18.853322 20.041854 21.230386 0.88532983
## 2013-12-27 18.799933 20.110740 21.421546 0.96115346

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")
```

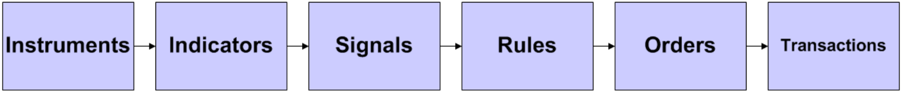
Bollinger bands



Outline

1. Introduction
2. Basic quantstrat strategy example
3. Position sizing
4. Stop orders
5. Parameter optimization

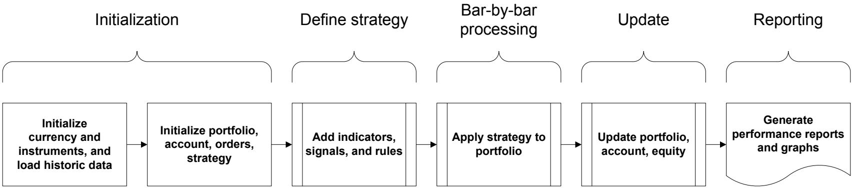
Quantstrat object model



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


Initialization
Define strategy
Bar-by-bar processing
Update
Reporting
Initialize currency and instruments, and load historic data
Initialize portfolio, account, orders, strategy
Add indicators, signals, and rules
Apply strategy to portfolio
Update portfolio, account, equity
Generate performance reports and graphs

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

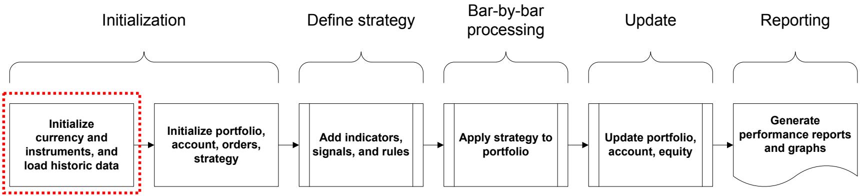
add.indicator	add an indicator to a strategy
add.signal	add a signal to a strategy
add.rule	add a rule to a strategy
add.distribution	add a distribution to a paramset in a strategy
add.constraint	add a constraint on 2 distributions within a paramset

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`

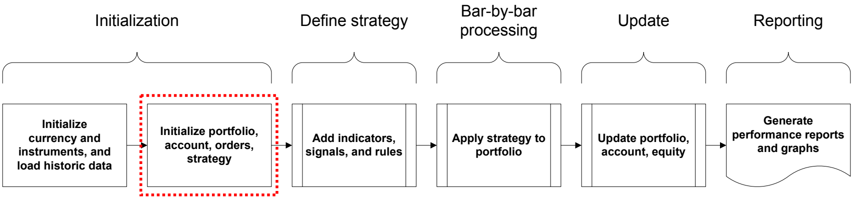
Initialize instruments


Initialization
Define strategy
Bar-by-bar processing
Update
Reporting
Initialize currency and instruments, and load historic data
Initialize portfolio, account, orders, strategy
Add indicators, signals, and rules
Apply strategy to portfolio
Update portfolio, account, equity
Generate performance reports and graphs

```r
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


Initialization
Define strategy
Bar-by-bar processing
Update
Reporting
Initialize currency and instruments, and load historic data
Initialize portfolio, account, orders, strategy
Add indicators, signals, and rules
Apply strategy to portfolio
Update portfolio, account, equity
Generate performance reports and graphs

```r
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)
```

```r
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

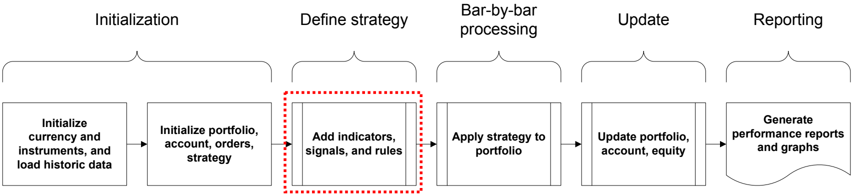
- Indicators are typically standard technical or statistical analysis outputs, such as moving averages, bands, or pricing models
- Indicators are applied before signals and rules, and the output of indicators may be used as inputs to construct signals or fire rules

```
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



Initialization
Define strategy
Bar-by-bar processing
Update
Reporting
Initialize currency and instruments, and load historic data
Initialize portfolio, account, orders, strategy
Add indicators, signals, and rules
Apply strategy to portfolio
Update portfolio, account, equity
Generate performance reports and graphs

```r
args(BBands)

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

```r
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:

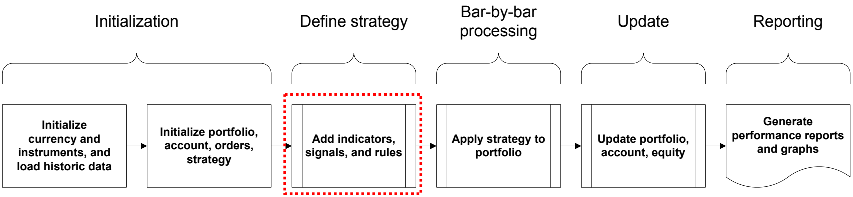
sigCrossover	crossover signal ("gt", "lt", "eq", "gte", "lte")
sigComparison	comparison signal ("gt", "lt", "eq", "gte", "lte")
sigThreshold	threshold signal ("gt", "lt", "eq", "gte", "lte")
sigPeak	peak/valley signals ("peak", "bottom")
sigFormula	signal calculated from a formula

```
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


Initialization
Define strategy
Bar-by-bar processing
Update
Reporting
Initialize currency and instruments, and load historic data
Initialize portfolio, account, orders, strategy
Add indicators, signals, and rules
Apply strategy to portfolio
Update portfolio, account, equity
Generate performance reports and graphs

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

```r
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

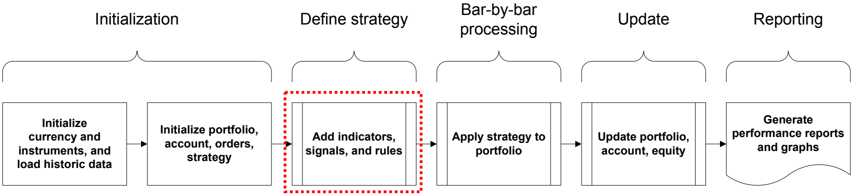
ruleSignal is the default rule to generate a trade order on a signal

```
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


Initialization
Define strategy
Bar-by-bar processing
Update
Reporting
Initialize currency and instruments, and load historic data
Initialize portfolio, account, orders, strategy
Add indicators, signals, and rules
Apply strategy to portfolio
Update portfolio, account, equity
Generate performance reports and graphs

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

```r
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
```

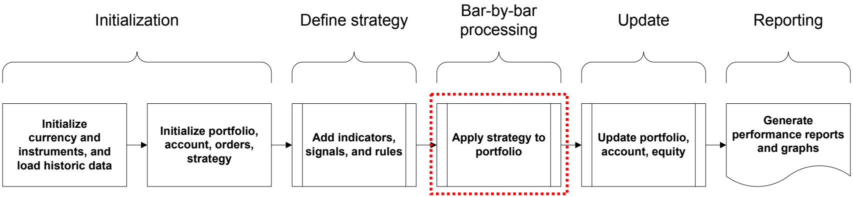
Main arguments:

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


Initialization
Define strategy
Bar-by-bar processing
Update
Reporting
Initialize currency and instruments, and load historic data
Initialize portfolio, account, orders, strategy
Add indicators, signals, and rules
Apply strategy to portfolio
Update portfolio, account, equity
Generate performance reports and graphs

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

```r
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             0.00000
## 2010-09-24     100 21.398265        0   2139.8265    21.398265             0.00000
## 2011-02-11     100 24.986291        0   2498.6291    24.986291             0.00000
## 2011-06-17    -200 22.878591        0  -4575.7182    22.878591           -62.73734
## 2012-02-03     100 26.151044        0   2615.1044    26.151044             0.00000
## 2012-03-23     100 28.383419        0   2838.3419    28.383419             0.00000
## 2012-08-24     100 29.044692        0   2904.4692    29.044692             0.00000
## 2012-09-14     100 30.004298        0   3000.4298    30.004298             0.00000
## 2012-11-23    -400 27.523743        0 -11009.4973    27.523743          -348.84793
## 2013-03-15     100 29.095923        0   2909.5923    29.095923             0.00000
## 2013-04-19     100 28.276998        0   2827.6998    28.276998             0.00000
## 2013-05-10     100 30.435106        0   3043.5106    30.435106             0.00000
## 2013-09-27     100 31.369301        0   3136.9301    31.369301             0.00000
## 2013-10-25     100 32.595273        0   3259.5273    32.595273             0.00000
## 2014-01-03     100 34.437750        0   3443.7750    34.437750             0.00000
## 2014-03-14     100 34.838643        0   3483.8643    34.838643             0.00000
## 2014-03-28     100 35.420604        0   3542.0604    35.420604             0.00000
## 2014-05-23     100 36.667740        0   3666.7740    36.667740             0.00000
## 2014-10-24    -900 38.887717        0 -34998.9453    38.887717          5685.21144
## 2014-11-14     100 41.225934        0   4122.5934    41.225934             0.00000
## 2015-03-13     100 41.320873        0   4132.0873    41.320873             0.00000
## 2015-05-01     100 43.100000        0   4310.0000    43.100000             0.00000
```

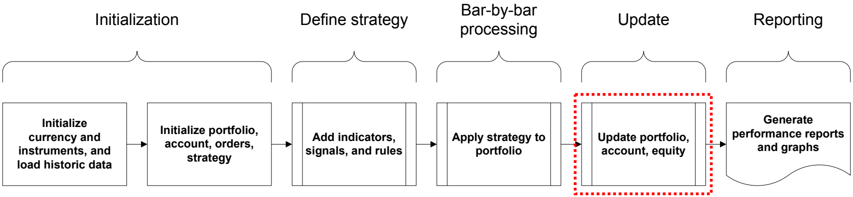
The mktdata object

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   41902800      71.3902  63.5691    68.0868  72.6045   0.896858             NA             NA
## 2015-01-09  71.1211  71.5796 68.8587   70.6527   35187500      70.6527  63.6107    68.2271  72.8436   0.731405             NA             NA
## 2015-01-16  70.7922  71.4500 68.2707   69.4368   41348200      69.4368  63.6509    68.3115  72.9720   0.651023             NA             NA
## 2015-01-23  69.5165  71.0414 68.4202   70.6926   43463400      70.6926  63.6756    68.3976  73.1195   0.675121             NA             NA
## 2015-01-30  70.6627  71.1012 69.4069   69.7557   31099800      69.7557  63.7435    68.5124  73.2812   0.665193             NA             NA
## 2015-02-06  69.9351  73.0546 68.8288   72.6958   45344600      72.6958  63.7672    68.6986  73.6301   0.786713             NA             NA
## 2015-02-13  72.2872  74.6393 72.2175   74.6393   24706000      74.6393  63.6991    69.0409  74.3827   0.948459              1             NA
## 2015-02-20  74.5297  75.2074 74.1609   75.1875   15468600      75.1875  63.7775    69.4860  75.1944   0.970005             NA             NA
## 2015-02-27  75.1376  76.2639 74.8885   75.7157   20855600      75.7157  64.1102    70.0185  75.9268   0.974265             NA             NA
## 2015-03-06  75.7655  76.6526 75.0081   75.1576   50333400      75.1576  65.2124    70.6510  76.0895   0.955552             NA             NA
## 2015-03-13  75.2174  75.6758 73.9815   74.9084   28468000      74.9084  66.0189    71.1351  76.2512   0.863574             NA             NA
## 2015-03-20  75.1875  76.9300 74.6692   76.7700   28895300      76.7700  66.4826    71.6078  76.7330   0.940500              1             NA
## 2015-03-27  76.7200  77.1300 74.2300   74.9100   23348000      74.9100  67.0506    72.0176  76.9845   0.842842             NA             NA
## 2015-04-02  75.1600  76.1500 74.5300   75.6900   24538300      75.6900  67.5221    72.3802  77.2383   0.816631             NA             NA
## 2015-04-10  75.2500  76.7500 75.1600   76.6700   21608400      76.6700  67.7936    72.7200  77.6463   0.852528             NA             NA
## 2015-04-17  76.6000  76.8300 74.9900   75.2300   27113800      75.2300  67.9554    72.9627  77.9701   0.771659             NA             NA
## 2015-04-24  75.5500  77.6900 75.5500   77.6300   20372300      77.6300  68.0744    73.2689  78.4634   0.854971             NA             NA
## 2015-05-01  77.8200  77.8900 74.9700   76.3500   36376200      76.3500  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


Initialization
Define strategy
Bar-by-bar processing
Update
Reporting
Initialize currency and instruments, and load historic data
Initialize portfolio, account, orders, strategy
Add indicators, signals, and rules
Apply strategy to portfolio
Update portfolio, account, equity
Generate performance reports and graphs

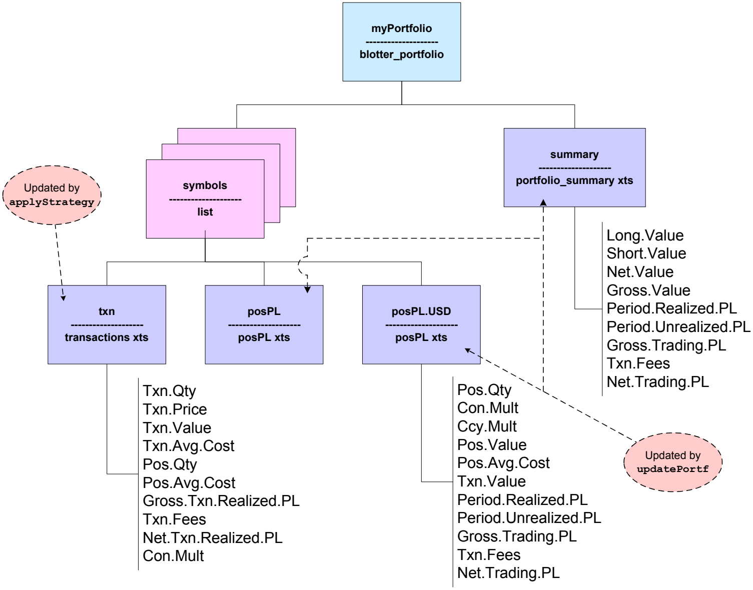
```r
updatePortf("multiAsset.bb1")
updateAcct("multiAsset.bb1")
updateEndEq("multiAsset.bb1")
```

Data integrity check

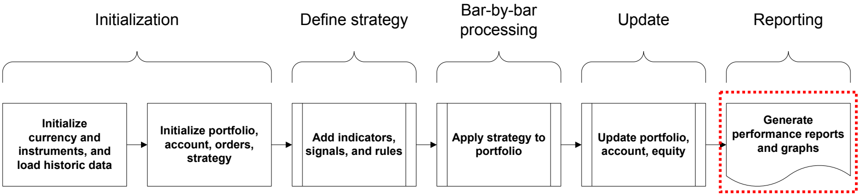
```r
checkBlotterUpdate <- function(port.st,account.st,verbose=TRUE)
{
  ok <- TRUE
  p <- getPortfolio(port.st)
  a <- getAccount(account.st)
  syms <- names(p$symbols)
  port.tot <- sum(sapply(syms,FUN = function(x) eval(parse(
    text=paste("sum(p$symbols",x,"posPL.USD$Net.Trading.PL)",sep="$")))))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



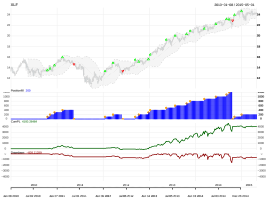
Generate position plots

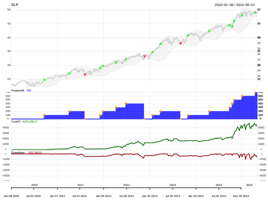

Initialization
Define strategy
Bar-by-bar processing
Update
Reporting
Initialize currency and instruments, and load historic data
Initialize portfolio, account, orders, strategy
Add indicators, signals, and rules
Apply strategy to portfolio
Update portfolio, account, equity
Generate performance reports and graphs

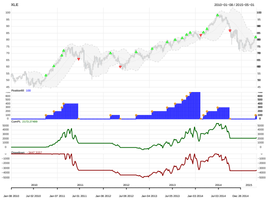
```r
par(mfrow=c(3,3))
for(symbol in symbols)
{
chart.Posn(Portfolio="multiAsset.bb1",Symbol=symbol,theme=myTheme,
        TA="add_BBands(n=20,sd=2)")
}
par(mfrow=c(1,1))
```

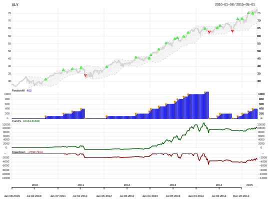
```r
chart.Posn("multiAsset.bb1","XLU",TA="add_BBands(n=20,sd=2)",theme=myTheme)
```

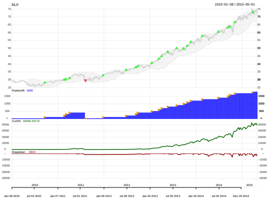
Position plots

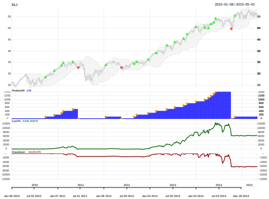


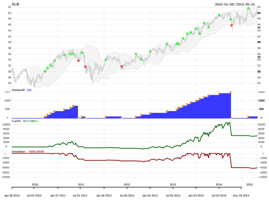


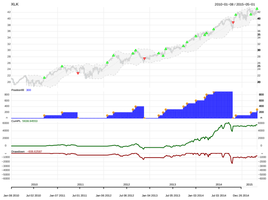


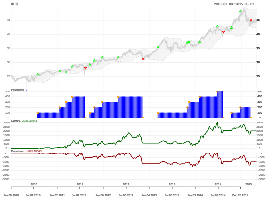




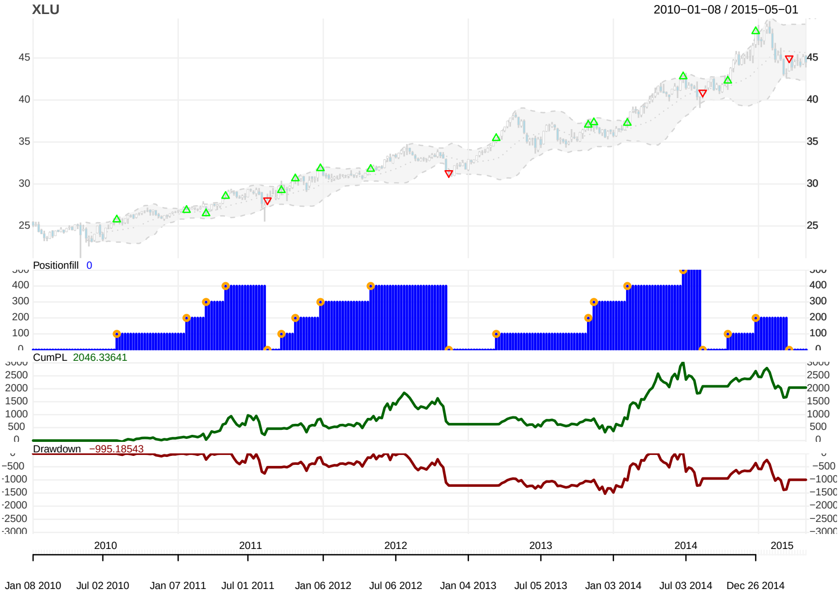








BBands strategy for XLU



Trade stats by instrument

```r
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      5072.0891      2173.2750      4100.2848      6435.6588      5638.9455      4270.2912      2046.3364     39409.3318     10184.8164
      Avg.Trade.PL     1280.13070      521.06875     1363.29270     2118.93489     1757.87539      538.86311      511.58410     −113.49591     2973.66745
      Med.Trade.PL     −60.808920     −34.599061      83.914102      43.796561     −62.737340     352.243350     315.593575    −113.495906    −510.664287
    Largest.Winner      5714.8456      3241.2785      4152.6642      6413.8443      5685.2114      1120.6512      1464.9364         0.0000     10401.7823
     Largest.Loser    −472.704914   −1087.805404    −146.700173    −100.836219    −348.847930       0.000000     −49.787119    −113.495906    −970.115644
     Gross.Profits      5822.7501      4161.5775      4236.5783      6457.6409      5685.2114      1616.5893      2096.1235         0.0000     10401.7823
      Gross.Losses    −702.227295   −2077.302487    −146.700173    −100.836219    −411.585270       0.000000     −49.787119    −113.495906   −1480.779931
  Std.Dev.Trade.PL     2966.04648     2035.45336     2418.41697     3720.20362     3404.17995      514.52035      668.85305                    6437.03669
  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      8.2918311      2.0033565     28.8791635     64.0408870     13.8129614                    42.1017241      0.0000000      7.0245295
     Avg.Win.Trade     2911.37505     2080.78874     2118.28913     3228.82044     5685.21144      538.86311      698.70784                   10401.78228
     Med.Win.Trade     2911.37505     2080.78874     2118.28913     3228.82044     5685.21144      352.24335      459.20840                   10401.78228
  Avg.Losing.Trade    −351.113648   −1038.651244    −146.700173    −100.836219    −205.792635                    −49.787119    −113.495906    −740.389966
  Med.Losing.Trade    −351.113648   −1038.651244    −146.700173    −100.836219    −205.792635                    −49.787119    −113.495906    −740.389966
      Avg.Daily.PL     1280.13070      521.06875     1363.29270     2118.93489     1757.87539      538.86311      511.58410     −113.49591     2973.66745
      Med.Daily.PL     −60.808920     −34.599061      83.914102      43.796561     −62.737340     352.243350     315.593575    −113.495906    −510.664287
  Std.Dev.Daily.PL     2966.04648     2035.45336     2418.41697     3720.20362     3404.17995      514.52035      668.85305                    6437.03669
        Ann.Sharpe      6.8513575      4.0638170      8.9486639      9.0417224      8.1973947     16.6255554     12.1418985                     7.3334221
      Max.Drawdown     −6262.4067     −4979.4763     −1546.2991     −6336.5250     −2398.6538     −1031.6819     −1525.9096     −5849.9224     −5205.4592
Profit.To.Max.Draw      0.8099265      0.4364465      2.6516764      1.0156448      2.3508793      4.1391548      1.3410600      6.7367273      1.9565644
 Avg.WinLoss.Ratio      8.2918311      2.0033565     14.4395817     32.0204435     27.6259228                    14.0339080                    14.0490590
 Med.WinLoss.Ratio      8.2918311      2.0033565     14.4395817     32.0204435     27.6259228                     9.2234380                    14.0490590
        Max.Equity     11070.4958      5620.5907      4708.3977     12565.6368      6245.5714      4832.6513      3041.5218     43219.3318     12977.5578
        Min.Equity  −1023.9870911   −616.4244781   −202.9134174   −252.5894328   −865.7031832     −9.7114636    −43.2138894   −113.4959059   −970.1156441
        End.Equity      5072.0891      2173.2750      4100.2848      6435.6588      5638.9455      4270.2912      2046.3364     39409.3318     10184.8164
```

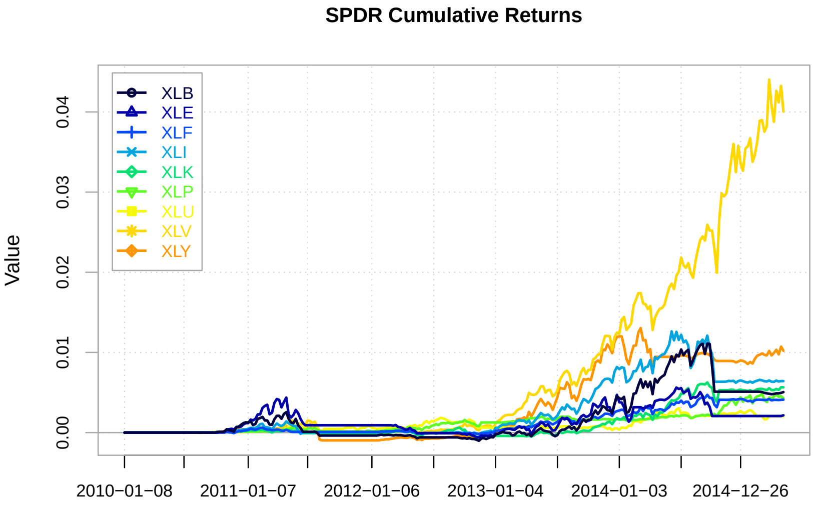
Individual asset returns

```
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. Introduction
2. Basic quantstrat strategy example
3. Position sizing
    - Position limits
    - User-supplied order sizing function
4. Stop orders
5. 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 Introduction

2 Basic quantstrat strategy example

3 Position sizing
- Position limits
- User-supplied order sizing function

4 Stop orders

5 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

```r
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)
```

```r
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

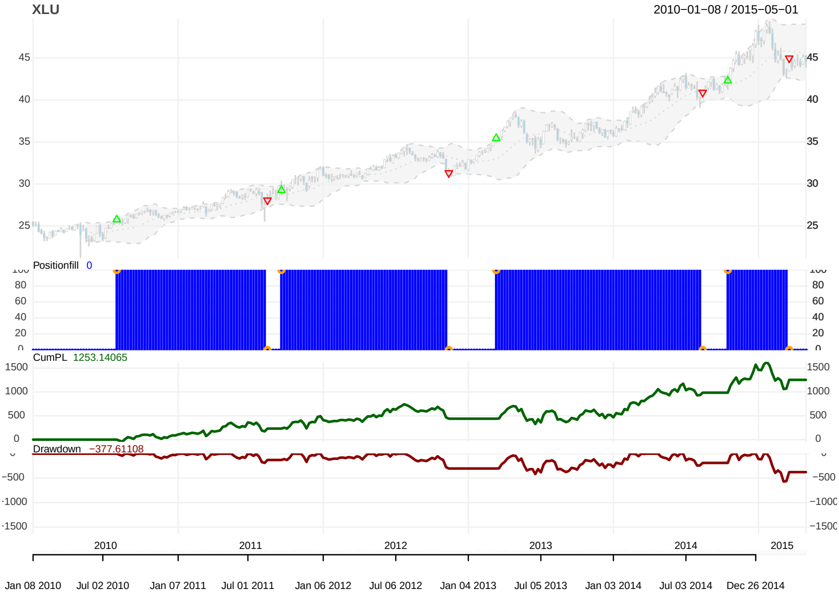
```
out <- applyStrategy("bbands",
  portfolios="multi.bb.limit",parameters=list(sd=2,n=20))
```

```
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. Introduction

2. Basic quantstrat strategy example

3. Position sizing
   - Position limits
   - User-supplied order sizing function

4. Stop orders

5. 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

```r
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:

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

Add rules with an order sizing function specified

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

```r
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

```r
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)
```

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

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

```r
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     49343.1251 -5986.5052 52995.7611
## 4 2014-11-07 2015-05-01     4200    4200        1         100539.18      1688.8224 -4256.5737  3861.1252
##   Pct.Net.Trading.PL      Pct.MAE     Pct.MFE tick.Net.Trading.PL    tick.MAE   tick.MFE
## 1        0.083444625 -0.012328767 0.180979092         113.0183631  -16.698224 245.120169
## 2        0.003192848 -0.042922806 0.120965134           4.2731316  -57.445515 161.893059
## 3        0.474144992 -0.057525166 0.509243682         736.4645544  -89.350823 790.981510
## 4        0.016797655 -0.042337463 0.038404185          40.2100570 -101.346992  91.931553
```

- Each order is approximately $100,000 in value

Outline

1. Introduction
2. Basic quantstrat strategy example
3. Position sizing
4. Stop orders
5. 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

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

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

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

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

Add rules

```r
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')
```

```r
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

```r
stopLossPercent <- 0.03
```

```r
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

```r
trailingStopPercent <- 0.07
```

```r
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

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

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

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

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

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

## [1] TRUE
```

Outline

1. Introduction
2. Basic quantstrat strategy example
3. Position sizing
4. Stop orders
5. 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

```r
args(add.distribution)
```

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

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

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

Optimization range for stop loss

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

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

Define parameter constraint

```r
args(add.distribution.constraint)

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

```r
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

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

```r
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

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

```r
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

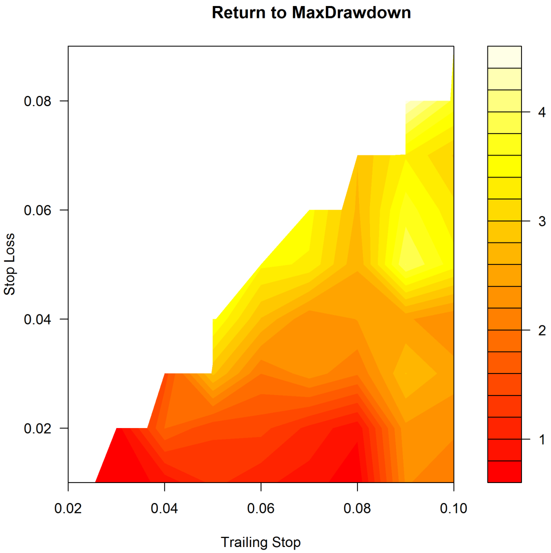
```
names(results)
```

```
##  [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

```r
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