Interactive Brokers

FRE6871 & FRE7241, Fall 2018

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

September 28, 2018

Package *IBrokers* for Using Interactive Brokers

Interactive Brokers (IB) is a brokerage company which provides an API for computer-driven trading, and it also provides extensive documentation:

Interactive Brokers Main Page
Interactive Brokers Documentation
Interactive Brokers Course

Disclaimer: I do have a personal account with Interactive Brokers.
But I do not have any other relationship with Interactive Brokers, and I do not endorse or recommend them.
Warning: Active trading is extremely risky, and most people lose money.
I advise not to trade with your own capital!

The package *IBrokers* contains R functions for executing IB commands using the Interactive Brokers API.

The package *IBrokers* has extensive documentation.

```
> # install package IBrokers
> install.packages("IBrokers")
> # load package IBrokers
> library(IBrokers)
> # get documentation for package IBrokers
> # get short description
> packageDescription("IBrokers")
> # load help page
> help(package="IBrokers")
> # list all datasets in "IBrokers"
> data(package="IBrokers")
> # list all objects in "IBrokers"
> ls("package:IBrokers")
> # remove IBrokers from search path
> detach("package:IBrokers")
```

Connecting to Interactive Brokers via the API

Connecting to Interactive Brokers via the API requires being logged into the IB Trader Workstation (*TWS*) or the IB Gateway (*IBG*).

You can Download the TWS and install it.

The *TWS* settings must be configured to enable the API: *File* → *Global Configuration* → *API*

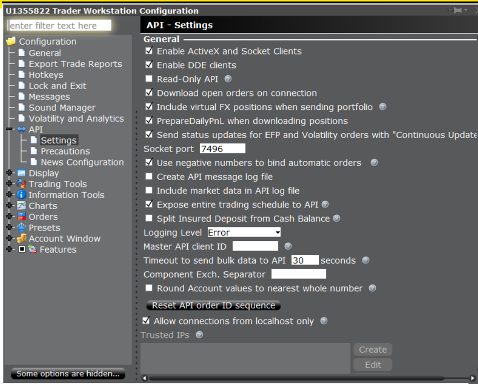
Interactive Brokers offers extensive API Documentation

The functions `twsConnect()` and `ibgConnect()` open a connection to the Interactive Brokers API, via either the *TWS* or the *IBG*.

The parameter `port` should be assigned to the value of the *socket port* displayed in *TWS* or *IBG*.

The function `twsDisconnect()` closes the Interactive Brokers API connection.

The function `reqAccountUpdates()` returns a list with Interactive Brokers account information.



API - Settings

General

- ☑ Enable ActiveX and Socket Clients
- ☑ Enable DDE clients
- □ Read-Only API
- ☑ Download open orders on connection
- ☑ Include virtual FX positions when sending portfolio
- ☑ PrepareDailyPnL when downloading positions
- ☑ Send status updates for EFP and Volatility orders with "Continuous Update

Socket port 7496

- ☑ Use negative numbers to bind automatic orders
- □ Create API message log file
- □ Include market data in API log file
- ☑ Expose entire trading schedule to API
- □ Split Insured Deposit from Cash Balance

Logging Level Error

Master API client ID

Timeout to send bulk data to API 30 seconds

Component Exch. Separator

- □ Round Account values to nearest whole number
- ☑ Allow connections from localhost only

Trusted IPs

```
> # connect to Interactive Brokers TWS
> ib_connect <- twsConnect(port=7497)
> # or connect to IB Gateway
> # ib_connect <- ibgConnect(port=4002)
> # check connection
> isConnected(ib_connect)
> # download Interactive Brokers account informa
> ib_account <- reqAccountUpdates(ib_connect)
> twsPortfolioValue(ib_account)
> # close the Interactive Brokers API connection
> twsDisconnect(ib_connect)
```

Defining Contracts Using Package *IBrokers*

The function twsEquity() defines a stock contract (IB contract object).

The functions twsFuture() and twsCurrency() define futures and currency contracts.

The function reqContractDetails() returns a list with information on the IB instrument.

The package *twsInstrument* contains utility functions for enhancing the package *IBrokers*.

To define an IB contract, first look it up by keyword in the online IB Contract and Symbol Database, and find the *Conid* for that instrument.

Enter the *Conid* into the function twsInstrument::getContract(), which will return the IB contract object for that instrument.

Interactive Brokers offers more information about financial contracts here: IB Traded Products

```
> # define AAPL stock contract (object)
> con_tract <- twsEquity("AAPL")
> # define CHF currency contract
> con_tract <- twsCurrency("CHF", currency="USD"
> # define S&P Emini future December 2018 contra
> con_tract <- twsFuture(symbol="ES",
+   exch="GLOBEX", expiry="201812")
> # define 10yr Treasury future December 2018 co
> con_tract <- twsFuture(symbol="ZN",
+   exch="ECBOT", expiry="201812")
> # define euro currency future October 2018 con
> con_tract <- twsFuture(symbol="EUR",
+   exch="GLOBEX", expiry="201810")
> # define Gold future July 2018 contract
> con_tract <- twsFuture(symbol="GC",
+   exch="NYMEX", expiry="201807")
> # test if contract object is correct
> is.twsContract(con_tract)
> # download list with instrument information
> reqContractDetails(conn=ib_connect, Contract=c
> # install the package twsInstrument
> install.packages("twsInstrument", repos="http:
> # define euro future using getContract() and C
> con_tract <- twsInstrument::getContract("31763
> # download list with instrument information
> reqContractDetails(conn=ib_connect, Contract=c
```

Defining *VIX* Futures Contracts

VIX futures have both monthly and weekly contracts, with the monthly contracts being the most liquid.

In order to uniquely specify a *VIX* futures contract, the parameter `"local"` should be passed into the function `twsFuture()`, with the local security name.

For example, *VXV8* is the local security name (symbol) for the monthly *VIX* futures contract expiring on October 17th, 2018.

VX40V8 is the local security name (symbol) for the weekly *VIX* futures contract expiring on October 3rd, 2018.

The function `reqContractDetails()` returns a list with information on the IB instrument.

VIX futures are traded on the CFE (CBOE Futures Exchange): http://cfe.cboe.com/

```
> # define VIX monthly and weekly futures Octobe
> sym_bol <- "VIX"
> con_tract <- twsFuture(symbol=sym_bol,
+   exch="CFE", expiry="201810")
> # define VIX monthly futures October 2018 cont
> con_tract <- twsFuture(symbol=sym_bol,
+   local="VXV8", exch="CFE", expiry="201810")
> # define VIX weekly futures October 3rd 2018 c
> con_tract <- twsFuture(symbol=sym_bol,
+   local="VX40V8", exch="CFE", expiry="201810")
> # download list with instrument information
> reqContractDetails(conn=ib_connect,
+   Contract=con_tract)
```

Downloading Historical Data from Interactive Brokers

The function `reqHistoricalData()` downloads historical data to a *csv* file.

The historical bar data fields are `"Open"`, `"High"`, `"Low"`, `"Close"`, `"Volume"`, `"WAP"`, `"Count"`. (`"WAP"` is the weighted average price.)

Interactive Brokers offers more information about historical market data:
  IB Historical Market Data
  IB Historical Bar Data Fields

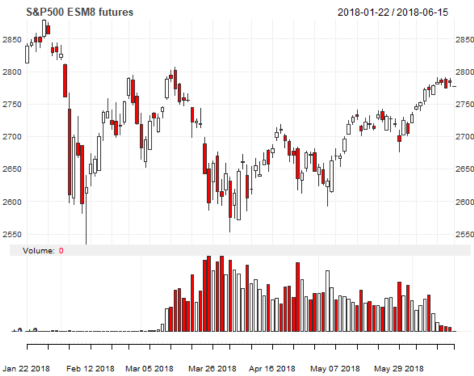
```
> # define S&P Emini futures December 2018 contr
> sym_bol <- "ES"
> con_tract <- twsFuture(symbol=sym_bol,
+   exch="GLOBEX", expiry="201812")
> # open file for data download
> data_dir <- "C:/Develop/data/ib_data"
> dir.create(data_dir)
> file_name <- file.path(data_dir, paste0(sym_bo
> file_connect <- file(file_name, open="w")
> # connect to Interactive Brokers TWS
> ib_connect <- twsConnect(port=7497)
> # download historical data to file
> reqHistoricalData(conn=ib_connect,
+   Contract=con_tract,
+   # whatToShow="MIDPOINT",
+   # endDateTime=ib_time,
+   barSize="1 day", duration="6 M",
+   file=file_connect)
> # close data file
> close(file_connect)
> # close the Interactive Brokers API connection
> twsDisconnect(ib_connect)
```

Downloading Historical Data for Expired Futures Contracts

Historical data for expired futures contracts can be downloaded using `reqHistoricalData()` with the appropriate expiry date, and the parameter `include_expired="1"`.

For example, *ESM8* is the symbol for the *S&P500* emini futures expiring in June 2018.

```
> # define S&P Emini futures June 2018 contract
> sym_bol <- "ES"
> con_tract <- twsFuture(symbol=sym_bol,
+   include_expired="1",
+   exch="GLOBEX", expiry="201806")
> # open file for ESM8 data download
> file_name <- file.path(data_dir, paste0(sym_bo
> file_connect <- file(file_name, open="w")
> # connect to Interactive Brokers TWS
> ib_connect <- twsConnect(port=7497)
> # download historical data to file
> reqHistoricalData(conn=ib_connect,
+   Contract=con_tract,
+   barSize="1 day", duration="2 Y",
+   file=file_connect)
> # close data file
> close(file_connect)
> # close the Interactive Brokers API connectio
> twsDisconnect(ib_connect)
```



```
> # load OHLC data and coerce it into xts series
> price_s <- data.table::fread(file_name)
> data.table::setDF(price_s)
> price_s <- xts::xts(price_s[, 2:6],
+   order.by=as.Date(as.POSIXct.numeric(price_s[
+     tz="America/New_York", origin="1970-01-01"
> colnames(price_s) <- c("Open", "High", "Low",
> # plot OHLC data in x11 window
> chart_Series(x=price_s, TA="add_Vo()",
+   name="S&P500 ESM8 futures")
> # plot dygraph
```

Downloading Live *TAQ* Data from Interactive Brokers

The function reqMktData() downloads live (real-time) trades and quotes (*TAQ*) data from Interactive Brokers.

The function eWrapper() formats the real-time market events (trades and quotes), so they can be displayed or saved to a file.

The method eWrapper.MktData.CSV() formats the real-time *TAQ* data so it can be saved to a *csv* file.

The real-time *TAQ* data fields are *BidSize, BidPrice, AskPrice, AskSize, Last, LastSize, Volume*.

BidPrice is the quoted bid price, *AskPrice* is the quoted offer price, and *Last* is the most recent traded price.

The *TAQ* data is spaced irregularly in time, with data recorded each time a new trade or quote arrives.

```
> # define S&P Emini futures December 2018 contr
> sym_bol <- "ES"
> con_tract <- twsFuture(symbol=sym_bol,
+   exch="GLOBEX", expiry="201812")
> # open file for data download
> data_dir <- "C:/Develop/data/ib_data"
> # dir.create(data_dir)
> file_name <- file.path(data_dir, paste0(sym_bo
> file_connect <- file(file_name, open="w")
> # connect to Interactive Brokers TWS
> ib_connect <- twsConnect(port=7497)
> # download live data to file
> reqMktData(conn=ib_connect,
+      Contract=con_tract,
+      eventWrapper=eWrapper.MktData.CSV(1),
+      file=file_connect)
> # close data file
> close(file_connect)
> # close the Interactive Brokers API connection
> twsDisconnect(ib_connect)
```

Downloading Live *OHLC* Data from Interactive Brokers

The function `reqRealTimeBars()` downloads live (real-time) *OHLC* market data from Interactive Brokers.

The method `eWrapper.RealTimeBars.CSV()` formats the real-time *OHLC* data so it can be saved to a *csv* file.

```
> # define S&P Emini futures December 2018 contr
> sym_bol <- "ES"
> con_tract <- twsFuture(symbol=sym_bol,
+   exch="GLOBEX", expiry="201812")
> # open file for data download
> data_dir <- "C:/Develop/data/ib_data"
> # dir.create(data_dir)
> file_name <- file.path(data_dir, paste0(sym_bo
> file_connect <- file(file_name, open="w")
> # connect to Interactive Brokers TWS
> ib_connect <- twsConnect(port=7497)
> # download live data to file
> reqRealTimeBars(conn=ib_connect,
+      Contract=con_tract, barSize="1",
+      eventWrapper=eWrapper.RealTimeBars.CSV(1)
+      file=file_connect)
> # close data file
> close(file_connect)
> # close the Interactive Brokers API connection
> twsDisconnect(ib_connect)
> # Load OHLC data and coerce it into xts series
> price_s <- data.table::fread(file_name)
> data.table::setDF(price_s)
> price_s <- xts::xts(price_s[, 2:6],
+   order.by=as.POSIXct.numeric(price_s[, 1],
+     tz="America/New_York", origin="1970-01-01"
> colnames(price_s) <- c("Open", "High", "Low",
> # Plot OHLC data in x11 window
```

Event Processing Using eWrapper() and twsCALLBACK()

Functions which process real-time market events (like `reqMktData()` and `reqRealTimeBars()`) rely on the functions `eWrapper()` and `twsCALLBACK`.

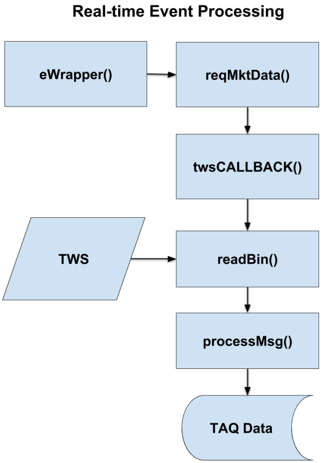
The function `eWrapper()` creates an *eWrapper* object, consisting of a data environment and functions for adding new data to it.

The function `reqMktData()` accepts an *eWrapper* object and passes it into `twsCALLBACK()`.

The function `twsCALLBACK()` processes market events by calling functions `readBin()` and `processMsg()` in a `while()` loop.

The *TWS* broadcasts real-time market events in the form of `character` strings, which are captured by `readBin()`.

The `character` strings are then parsed by `processMsg()`, and copied to the data environment of the *eWrapper* object.



Financial and Commodity Futures Contracts

The underlying assets delivered in *commodity futures* contracts are commodities, such as grains (corn, wheat), or raw materials and metals (oil, aluminum).

The underlying assets delivered in *financial futures* contracts are financial assets, such as stocks, bonds, and currencies.

Many futures contracts use cash settlement instead of physical delivery of the asset.

Futures contracts on different underlying assets can have quarterly, monthly, or even weekly expiration dates.

The front month futures contract is the contract with the closest expiration date to the current date.

Symbols of futures contracts are obtained by combining the contract code with the month code and the year.

For example, *ESZ8* is the symbol for the *S&P500* index *E-mini* futures expiring in December 2018.

Futures contract	Code
S&P500 index	ES
10yr Treasury	ZN
VIX index	VX
Gold	GC
Oil	CL
Euro FX	EC
Swiss franc	SF
Japanese Yen	JY

Month	Code
January	F
February	G
March	H
April	J
May	K
June	M
July	N
August	Q
September	U
October	V
November	X
December	Z

Interactive Brokers offers more information about futures contracts:

IB Contract and Symbol Database

IB Traded Products

List of Popular Futures Contracts.

E-mini Futures Contracts

E-mini futures are contracts with smaller notionals and tick values, which are more suitable for retail investors.

For example, the *E-mini* *QM* oil futures notional is 500 barrels, while the standard *CL* oil futures notional is 1,000 barrels.

The tick value is the change in the dollar value of the contract due to a one tick change in the underlying price.

For example, the tick value of the *S&P500* *E-mini* future is $12.50, and one tick is 0.25.

So if the *S&P500* index changes by one tick (0.25), then the contract value changes by $12.50.

In other words, if the *S&P500* index changes by one unit, then the *E-mini* contract value changes by $50, while the standard contract value changes by $250.

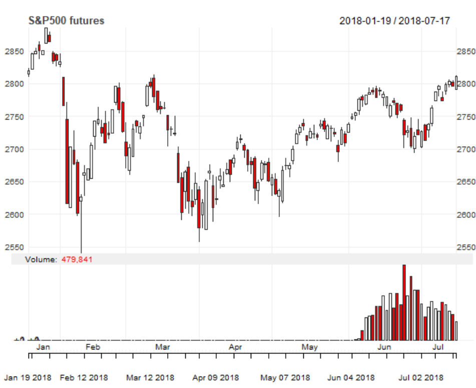
Futures contract	Standard	E-mini
S&P500 index	SP	ES
10yr Treasury	ZN	ZN
VIX index	VX	delisted
Gold	GC	YG
Oil	CL	QM
Euro FX	EC	E7
Swiss franc	SF	MSF
Japanese Yen	JY	J7

Plotting *S&P500* Futures Data

The function `data.table::fread()` reads *csv* files over five times faster than function `read.csv()`!

The function `as.POSIXct.numeric()` coerces a numeric value representing the *moment of time* into a `POSIXct` *date-time*, equal to the *clock time* in the local *time zone*.

```
> # load data for S&P Emini futures December 201
> sym_bol <- "ES"
> data_dir <- "C:/Develop/data/ib_data"
> file_name <- file.path(data_dir, paste0(sym_bo
> # read a data table from CSV file
> price_s <- data.table::fread(file_name)
> # coerce price_s into data frame
> data.table::setDF(price_s)
> # or
> # price_s <- data.table::as.data.frame.data.t
> #   data.table::fread(file_name))
> # first column of price_s is a numeric date-ti
> tail(price_s)
> # coerce price_s into xts series
> price_s <- xts::xts(price_s[, 2:6],
+   order.by=as.Date(as.POSIXct.numeric(price_s
+     tz="America/New_York",
+     origin="1970-01-01")))
> colnames(price_s) <- c("Open", "High", "Low", "Close", "Volume")
> tail(price_s)
```



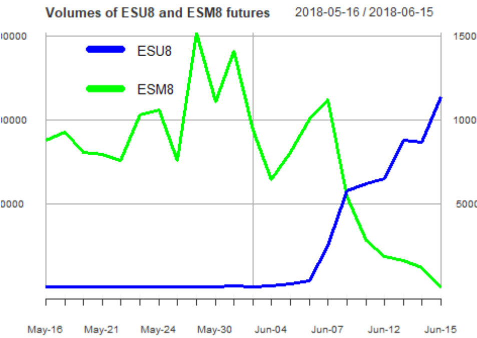
```
> # plot OHLC data in x11 window
> x11(width=5, height=4)  # open x11 for plottin
> par(mar=c(5, 5, 2, 1), oma=c(0, 0, 0, 0))
> chart_Series(x=price_s, TA="add_Vo()",
+   name="S&P500 futures")
> # plot dygraph
> dygraphs::dygraph(price_s[, 1:4], main="OHLC p
+   dyCandlestick()
```

Consecutive Contract Futures Volumes

The trading volumes of a futures contract drop significantly shortly before its expiration, and the successive contract volumes increase.

The contract with the highest trading volume is usually considered the most liquid contract.

```
> # load ESU8 data
> data_dir <- "C:/Develop/data/ib_data"
> file_name <- file.path(data_dir, "ESU8.csv")
> ES_U8 <- data.table::fread(file_name)
> data.table::setDF(ES_U8)
> ES_U8 <- xts::xts(ES_U8[, 2:6],
+   order.by=as.Date(as.POSIXct.numeric(ES_U8[,
+     tz="America/New_York", origin="1970-01-01"
> colnames(ES_U8) <- c("Open", "High", "Low", "C
> # load ESM8 data
> file_name <- file.path(data_dir, "ESM8.csv")
> ES_M8 <- data.table::fread(file_name)
> data.table::setDF(ES_M8)
> ES_M8 <- xts::xts(ES_M8[, 2:6],
+   order.by=as.Date(as.POSIXct.numeric(ES_M8[,
+     tz="America/New_York", origin="1970-01-01
> colnames(ES_M8) <- c("Open", "High", "Low", "(
```



```
> # plot last month of ESU8 and ESM8 volume data
> en_d <- end(ES_M8)
> star_t <- (en_d - 30*24*60^2)
> vol_ume <- cbind(Vo(ES_U8),
+   Vo(ES_M8))[paste0(star_t, "/", en_d)]
> colnames(vol_ume) <- c("ESU8", "ESM8")
> col_ors <- c("blue", "green")
> plot(vol_ume, col=col_ors, lwd=3, major.ticks=
+      format.labels="%b-%d", observation.based=
+      main="Volumes of ESU8 and ESM8 futures")
> legend("topleft", legend=colnames(vol_ume), co
+   title=NULL, bty="n", lty=1, lwd=6, inset=0.1
```

Chaining Together Futures Prices

Chaining futures means splicing together prices from several consecutive futures contracts.

A continuous futures contract is a time series of prices obtained by chaining together prices from consecutive futures contracts.

The price of the continuous contract is equal to the most liquid contract times a scaling factor.

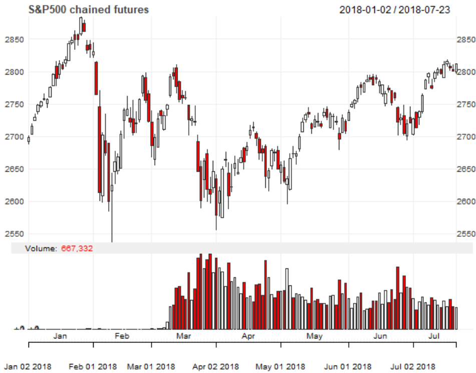
When the next contract becomes more liquid, then the continuous contract price is rolled over to that contract.

Futures contracts with different maturities (expiration dates) trade at different prices because of the futures curve, which causes price jumps between consecutive futures contracts.

The old contract price is multiplied by a scaling factor after that contract is rolled, to remove price jumps.

So the continuous contract prices are not equal to the past futures prices.

Interactive Brokers offers information about Continuous Contract Futures market data:



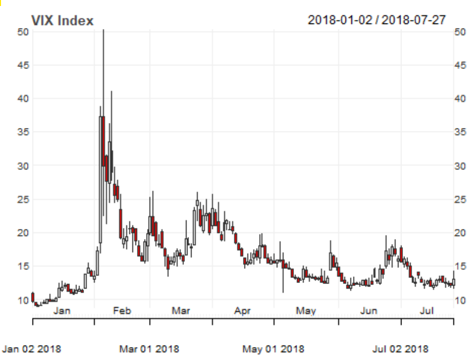
```
> # find date when ESU8 volume exceeds ESM8
> exceed_s <- (vol_ume[, "ESU8"] > vol_ume[, "ES
> in_deks <- min(which(exceed_s))
> # in_deks <- match(TRUE, exceed_s)
> # scale the ES_M8 prices
> in_deks <- index(exceed_s[in_deks])
> fac_tor <- as.numeric(Cl(ES_U8[in_deks])/Cl(ES
> ES_M8[, 1:4] <- fac_tor*ES_M8[, 1:4]
> # calculate continuous contract prices
> chain_ed <- rbind(ES_M8[index(ES_M8) < in_deks
+                   ES_U8[index(ES_U8) >= in_deks])
```

VIX Volatility Index

The *VIX* Volatility Index is an estimate of expected stock market volatility, calculated from the implied volatilities of options on the *S&P500* Index (SPX).

The *VIX* index is not a directly tradable asset, but it can be traded using *VIX* futures.

The CBOE provides daily historical data for the *VIX* index.



```
> # download VIX index data from CBOE
> vix_index <- data.table::fread("http://www.cbo
> class(vix_index)
> dim(vix_index)
> tail(vix_index)
> sapply(vix_index, class)
> vix_index <- xts(vix_index[, -1],
+   order.by=as.Date(vix_index$Date, format="%m/%d/%Y"))
> colnames(vix_index) <- c("Open", "High", "Low", "Close")
> # Save the VIX data to binary file
> load(file="C:/Develop/data/ib_data/vix_cboe.RData")
> ls(vix_env)
> vix_env$vix_index <- vix_index
> ls(vix_env)
> save(vix_env, file="C:/Develop/data/ib_data/vix_cboe.RData")
> # plot OHLC data in x11 window
> chart_Series(x=vix_index["2018"], name="VIX Index")
```

VIX Futures Contracts

VIX futures are cash-settled futures contracts on the *VIX* Index.

The most liquid *VIX* futures are with monthly expiration dates (CBOE Expiration Calendar), but weekly *VIX* futures are also traded.

These are the VIX Futures Monthly Expiration Dates from 2004 to 2019.

VIX futures are traded on the CFE (CBOE Futures Exchange):

http://cfe.cboe.com/
http://www.cboe.com/vix

VIX Contract Specifications:

VIX Contract Specifications
VIX Expiration Calendar

```
> # Read CBOE monthly futures expiration dates
> date_s <- read.csv(
+   file="C:/Develop/R/lecture_slides/data/futur
+   stringsAsFactors=FALSE)
> date_s <- as.Date(date_s[, 1])
> year_s <- format(date_s, format="%Y")
> year_s <- substring(year_s, 4)
> # monthly futures contract codes
> month_codes <- c("F", "G", "H", "J", "K", "M",
> sym_bols <- paste0("VX", month_codes, year_s)
> date_s <- as.data.frame(date_s)
> colnames(date_s) <- "monthly_expiration_dates"
> rownames(date_s) <- sym_bols
> # write dates to CSV file, with row names
> write.csv(date_s, row.names=TRUE,
+   file="C:/Develop/R/lecture_slides/data/futur
> # read back CBOE futures expiration dates
> date_s <- read.csv(file="C:/Develop/R/lecture_
+   stringsAsFactors=FALSE, row.names=1)
> date_s[, 1] <- as.Date(date_s[, 1])
```

VIX Futures Curve

Futures contracts with different expiration dates trade at different prices, known as the *futures curve* (or *term structure*).

The *VIX* futures curve is similar to the interest rate *yield curve*, which displays yields at different bond maturities.

The *VIX* futures curve is not the same as the *VIX* index term structure.

More information about the *VIX* Index and the *VIX* futures curve:

VIX Futures
VIX Futures Data
VIX Futures Curve
VIX Index Term Structure

```
> # Load VIX futures data from binary file
> load(file="C:/Develop/data/ib_data/vix_cboe.RD
> # Get all VIX futures for 2018 except January
> sym_bols <- ls(vix_env)
> sym_bols <- sym_bols[grep("*8", sym_bols)]
> sym_bols <- sym_bols[-1]
> # Specify dates for curves
> low_vol <- as.Date("2018-01-11")
> hi_vol <- as.Date("2018-02-05")
> # Extract all VIX futures prices on the dates
> curve_s <- lapply(sym_bols, function(sym_bol)
+   x_ts <- get(x=sym_bol, envir=vix_env)
+   Cl(x_ts[c(low_vol, hi_vol)])
+ })  # end lapply
> curve_s <- rutils::do_call(cbind, curve_s)
> colnames(curve_s) <- sym_bols
> curve_s <- t(coredata(curve_s))
> colnames(curve_s) <- c("Contango 01/11/2018",
+                  "Backwardation 02/05/2018")
```

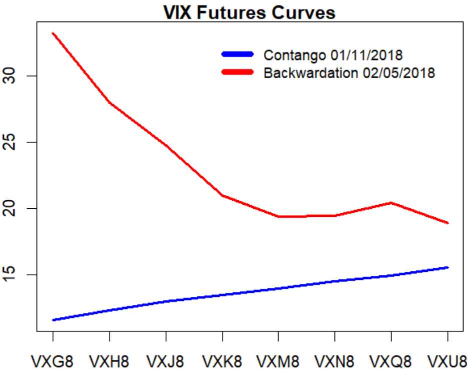
Contango and *Backwardation* of *VIX* Futures Curve

When prices are *low* then the futures curve is usually *upward sloping*, known as *contango*.

Futures prices are in *contango* most of the time.

When prices are *high* then the curve is usually *downward sloping*, known as *backwardation*.

```
> x11()
> par(mar=c(3, 2, 1, 1), oma=c(0, 0, 0, 0))
> plot(curve_s[, 1], type="l", lty=1, col="blue"
+      xaxt="n", xlab="", ylab="", ylim=range(cu
+      main="VIX Futures Curves")
> axis(1, at=(1:NROW(curve_s)), labels=rownames(
> lines(curve_s[, 2], lty=1, lwd=3, col="red")
> legend(x="topright", legend=colnames(curve_s),
+  inset=0.05, cex=0.9, bty="n",
+  col=c("blue", "red"), lwd=6, lty=1)
```



Futures Prices at Constant Maturity

A constant maturity futures price is the price of a hypothetical futures contract with an expiration date at a fixed number of days in the future.

Futures prices at a constant maturity can be calculated by interpolating the prices of contracts with neighboring expiration dates.

```
> # Read CBOE futures expiration dates
> date_s <- read.csv(file="C:/Develop/R/lecture_
+   stringsAsFactors=FALSE, row.names=1)
> sym_bols <- rownames(date_s)
> expiration_dates <- as.Date(date_s[, 1])
> to_day <- as.Date("2018-05-07")
> maturi_ty <- to_day + 30
> # Find neighboring futures contracts
> in_deks <- match(TRUE, expiration_dates > matu
> # in_deks <- min(which(expiration_dates > to_d
> expiration_dates[in_deks-1]
> expiration_dates[in_deks]
> front_symbol <- sym_bols[in_deks-1]
> back_symbol <- sym_bols[in_deks]
> front_date <- expiration_dates[in_deks-1]
> back_date <- expiration_dates[in_deks]
> # Load VIX futures data from binary file
> load(file="C:/Develop/data/vix_data/vix_cboe.R
> front_price <- get(x=front_symbol, envir=vix_e
> # front_price <- vix_env$front_symbol
> front_price <- as.numeric(Cl(front_price[to_da
> back_price <- get(x=back_symbol, envir=vix_env
> back_price <- as.numeric(Cl(back_price[to_day]
> # Calculate the constant maturity 30-day futur
> fra_c <- as.numeric(maturi_ty - front_date) /
+   as.numeric(back_date - front_date)
> pric_e <- (fra_c*back_price +
+   (1-fra_c)*front_price)
```

VIX Futures Investing

The volatility index moves in the opposite direction to the underlying asset price.

An increase in the *VIX* index coincides with a drop in stock prices, and vice versa.

Taking a *long* position in *VIX* futures is similar to a *short* position in stocks, and vice versa.

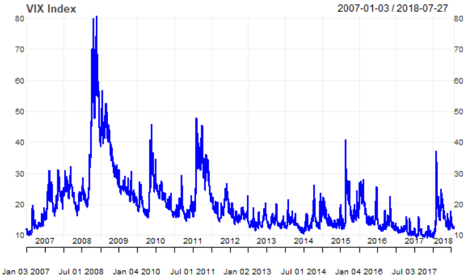
There are several exchange-traded funds (*ETFs*) and exchange traded notes (*ETNs*) which are linked to *VIX* futures.

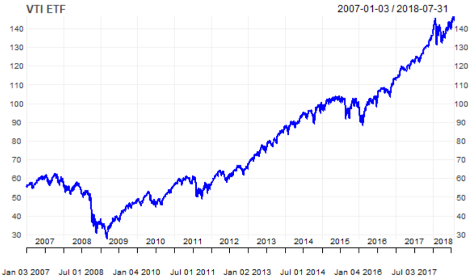
VXX is an *ETN* providing the total return of a *long VIX* futures contract.

SVXY is an *ETF* providing the total return of a *short VIX* futures contract.

Standard and Poor's explains the calculation of the Total Return on VIX Futures Indices

```
> # plot VIX and SVXY data in x11 window
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- "blue"
> chart_Series(x=Cl(vix_env$vix_index["2007/"]),
+         theme=plot_theme, name="VIX Index")
> chart_Series(x=Cl(rutils::etf_env$VTI["2007/"]
+         theme=plot_theme, name="VTI ETF")
```





VIX Crash on February 5th 2018

The *SVXY* and *XIV* ETFs rallied strongly after the financial crisis of 2008, so they became very popular with individual investors, and became very "crowded trades".

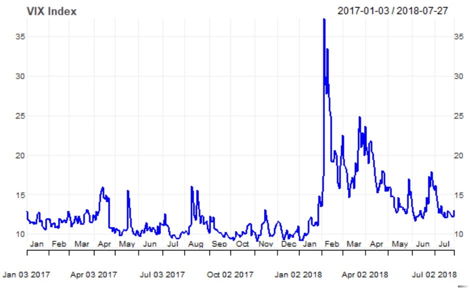
The *SVXY* and *XIV* ETFs had $3.6 billion of assets at the beginning of 2018.

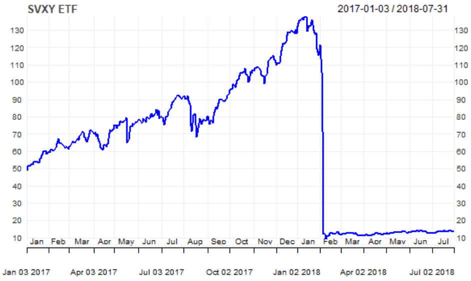
On February 5th 2018 the U.S. stock markets experienced a mini-crash, which was exacerbated by *VIX* futures short sellers.

As a result, the *XIV* ETF hit its termination event and its value dropped to zero:

Volatility Caused Stock Market Crash
XIV ETF Termination Event

```
> chart_Series(x=Cl(vix_env$vix_index["2017/2018
+         theme=plot_theme, name="VIX Index")
> chart_Series(x=Cl(rutils::etf_env$SVXY["2017/2
+         theme=plot_theme, name="SVXY ETF")
```


VIX Index
2017-01-03 / 2018-07-27
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr May Jun Jul
Jan 03 2017 Apr 03 2017 Jul 03 2017 Oct 02 2017 Jan 02 2018 Apr 02 2018 Jul 02 2018


SVXY ETF
2017-01-03 / 2018-07-31
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr May Jun Jul
Jan 03 2017 Apr 03 2017 Jul 03 2017 Oct 02 2017 Jan 02 2018 Apr 02 2018 Jul 02 2018