In that algorithm, mistakenly (or for the purpose of simplicity), I used calendar days as the time to expiry, when it should have been business days, which also accounts for weekends, and holidays, which are an irritating artifact to keep track of.

So here’s the salient change, in the loop that calculates times to expiry:

source("tradingHolidays.R")

masterlist <- list()

timesToExpiry <- list()

for(i in 1:length(contracts)) {

# obtain data

contract <- contracts[i]

dataFile <- paste0(stem, contract, "\_VX.csv")

expiryYear <- paste0("20",substr(contract, 2, 3))

expiryMonth <- monthMaps$monthNum[monthMaps$futureStem == substr(contract,1,1)]

expiryDate <- dates$dates[dates$dateMon == paste(expiryYear, expiryMonth, sep="-")]

data <- tryCatch(

{

suppressWarnings(fread(dataFile))

}, error = function(e){return(NULL)}

)

if(!is.null(data)) {

# create dates

dataDates <- as.Date(data$`Trade Date`, format = '%m/%d/%Y')

# create time to expiration xts

toExpiry <- xts(bizdays(dataDates, expiryDate), [order.by](http://order.by)=dataDates)

colnames(toExpiry) <- contract

timesToExpiry[[i]] <- toExpiry

# get settlements

settlement <- xts(data$Settle, [order.by](http://order.by)=dataDates)

colnames(settlement) <- contract

masterlist[[i]] <- settlement

}

}

The one salient line in particular, is this:

toExpiry <- xts(bizdays(dataDates, expiryDate), [order.by](http://order.by)=dataDates)

What is this bizdays function? It comes from the bizdays package in R.

There’s also the tradingHolidays.R script, which makes further use of the bizdays package. Here’s what goes on under the hood in tradingHolidays.R, for those that wish to replicate the code:

easters <- read.csv("easters.csv", header = FALSE)

easterDates <- as.Date(paste0(substr(easters$V2, 1, 6), easters$V3), format = '%m/%d/%Y')-2

nonEasters <- read.csv("nonEasterHolidays.csv", header = FALSE)

nonEasterDates <- as.Date(paste0(substr(nonEasters$V2, 1, 6), nonEasters$V3), format = '%m/%d/%Y')

weekdayNonEasters <- nonEasterDates[which(!weekdays(nonEasterDates) %in% c("Saturday", "Sunday"))]

hurricaneSandy <- as.Date(c("2012-10-29", "2012-10-30"))

holidays <- sort(c(easterDates, weekdayNonEasters, hurricaneSandy))

holidays <- holidays[holidays > as.Date("2003-12-31") & holidays < as.Date("2019-01-01")]

require(bizdays)

create.calendar("HolidaysUS", holidays, weekdays = c("saturday", "sunday"))

bizdays.options$set(default.calendar = "HolidaysUS")

There are two CSVs that I manually compiled, but will share screenshots of–they are the easter holidays (because they have to be adjusted for turning Sunday to Friday because of Easter Fridays), and the rest of the national holidays.

Here is what the easters csv looks like:



And the nonEasterHolidays, which contains New Year’s Day, MLK Jr. Day, President’s Day, Memorial Day, Independence Day, Labor Day, Thanksgiving Day, and Christmas Day (along with their observed dates) CSV:

Furthermore, we need to adjust for the two days that equities were not trading due to Hurricane Sandy.

So then, the list of holidays looks like this:

> holidays

[1] "2004-01-01" "2004-01-19" "2004-02-16" "2004-04-09" "2004-05-31" "2004-07-05" "2004-09-06" "2004-11-25"

[9] "2004-12-24" "2004-12-31" "2005-01-17" "2005-02-21" "2005-03-25" "2005-05-30" "2005-07-04" "2005-09-05"

[17] "2005-11-24" "2005-12-26" "2006-01-02" "2006-01-16" "2006-02-20" "2006-04-14" "2006-05-29" "2006-07-04"

[25] "2006-09-04" "2006-11-23" "2006-12-25" "2007-01-01" "2007-01-02" "2007-01-15" "2007-02-19" "2007-04-06"

[33] "2007-05-28" "2007-07-04" "2007-09-03" "2007-11-22" "2007-12-25" "2008-01-01" "2008-01-21" "2008-02-18"

[41] "2008-03-21" "2008-05-26" "2008-07-04" "2008-09-01" "2008-11-27" "2008-12-25" "2009-01-01" "2009-01-19"

[49] "2009-02-16" "2009-04-10" "2009-05-25" "2009-07-03" "2009-09-07" "2009-11-26" "2009-12-25" "2010-01-01"

[57] "2010-01-18" "2010-02-15" "2010-04-02" "2010-05-31" "2010-07-05" "2010-09-06" "2010-11-25" "2010-12-24"

[65] "2011-01-17" "2011-02-21" "2011-04-22" "2011-05-30" "2011-07-04" "2011-09-05" "2011-11-24" "2011-12-26"

[73] "2012-01-02" "2012-01-16" "2012-02-20" "2012-04-06" "2012-05-28" "2012-07-04" "2012-09-03" "2012-10-29"

[81] "2012-10-30" "2012-11-22" "2012-12-25" "2013-01-01" "2013-01-21" "2013-02-18" "2013-03-29" "2013-05-27"

[89] "2013-07-04" "2013-09-02" "2013-11-28" "2013-12-25" "2014-01-01" "2014-01-20" "2014-02-17" "2014-04-18"

[97] "2014-05-26" "2014-07-04" "2014-09-01" "2014-11-27" "2014-12-25" "2015-01-01" "2015-01-19" "2015-02-16"

[105] "2015-04-03" "2015-05-25" "2015-07-03" "2015-09-07" "2015-11-26" "2015-12-25" "2016-01-01" "2016-01-18"

[113] "2016-02-15" "2016-03-25" "2016-05-30" "2016-07-04" "2016-09-05" "2016-11-24" "2016-12-26" "2017-01-02"

[121] "2017-01-16" "2017-02-20" "2017-04-14" "2017-05-29" "2017-07-04" "2017-09-04" "2017-11-23" "2017-12-25"

[129] "2018-01-01" "2018-01-15" "2018-02-19" "2018-03-30" "2018-05-28" "2018-07-04" "2018-09-03" "2018-11-22"

[137] "2018-12-25"

So once we have a list of holidays, we use the bizdays package to set the holidays and weekends (Saturday and Sunday) as our non-business days, and use that function to calculate the correct times to expiry.

So, now that we have the updated expiry structure, we can write a function that will correctly replicate the four main volatility ETNs–XIV, VXX, ZIV, and VXZ.

Here’s the English explanation:

VXX is made up of two contracts–the front month, and the back month, and has a certain number of trading days (AKA business days) that it trades until expiry, say, 17. During that timeframe, the front month (let’s call it M1) goes from being the entire allocation of funds, to being none of the allocation of funds, as the front month contract approaches expiry. That is, as a contract approaches expiry, the second contract gradually receives more and more weight, until, at expiry of the front month contract, the second month contract contains all of the funds–just as it \*becomes\* the front month contract. So, say you have 17 days to expiry on the front month. At the expiry of the previous contract, the second month will have a weight of 17/17–100%, as it becomes the front month. Then, the next day, that contract, now the front month, will have a weight of 16/17 at settle, then 15/17, and so on. That numerator is called dr, and the denominator is called dt.

However, beyond this, there’s a second mechanism that’s responsible for the VXX looking like it does as compared to a basic futures contract (that is, the decay responsible for short volatility’s profits), and that is the “instantaneous” rebalancing. That is, the returns for a given day are today’s settles multiplied by yesterday’s weights, over yesterday’s settles multiplied by yesterday’s weights, minus one. That is, (S\_1\_t \* dr/dt\_t-1 + S\_2\_t \* 1-dr/dt\_t-1) / (S\_1\_t-1 \* dr/dt\_t-1 + S\_2\_t-1 \* 1-dr/dt\_t-1) – 1 (I could use a tutorial on LaTeX). So, when you move forward a day, well, tomorrow, today’s weights become t-1. Yet, when were the assets able to be rebalanced? Well, in the ETNs such as VXX and VXZ, the “hand-waving” is that it happens instantaneously. That is, the weight for the front month was 93%, the return was realized at settlement (that is, from settle to settle), and immediately after that return was realized, the front month’s weight shifts from 93%, to, say, 88%. So, say Credit Suisse ([that issues these ETNs](http://app.velocitysharesetns.com/files/prospectus/PRICING_SUPPLEMENT_No__VLS_ETN-1_A31_long_form_2.PDF) ), has $10,000 (just to keep the arithmetic and number of zeroes tolerable, obviously there are a lot more in reality) worth of XIV outstanding after immediately realizing returns, it will sell $500 of its $9300 in the front month, and immediately move them to the second month, so it will immediately go from $9300 in M1 and $700 in M2 to $8800 in M1 and $1200 in M2. When did those $500 move? Immediately, instantaneously, and if you like, you can apply Clarke’s Third Law and call it “magically”.

The only exception is the day after roll day, in which the second month simply becomes the front month as the previous front month expires, so what was a 100% weight on the second month will now be a 100% weight on the front month, so there’s some extra code that needs to be written to make that distinction.

That’s the way it works for VXX and XIV. What’s the difference for VXZ and ZIV? It’s really simple–instead of M1 and M2, VXZ uses the exact same weightings (that is, the time remaining on front month vs. how many days exist for that contract to be the front month), uses M4, M5, M6, and M7, with M4 taking dr/dt, M5 and M6 always being 1, and M7 being 1-dr/dt.

In any case, here’s the code.

syntheticXIV <- function(termStructure, expiryStructure) {

# find expiry days

zeroDays <- which(expiryStructure$C1 == 0)

# dt = days in contract period, set after expiry day of previous contract

dt <- zeroDays + 1

dtXts <- expiryStructure$C1[dt,]

# create dr (days remaining) and dt structure

drDt <- cbind(expiryStructure[,1], dtXts)

colnames(drDt) <- c("dr", "dt")

drDt$dt <- na.locf(drDt$dt)

# add one more to dt to account for zero day

drDt$dt <- drDt$dt + 1

drDt <- na.omit(drDt)

# assign weights for front month and back month based on dr and dt

wtC1 <- drDt$dr/drDt$dt

wtC2 <- 1-wtC1

# realize returns with old weights, "instantaneously" shift to new weights after realizing returns at settle

# assumptions are a bit optimistic, I think

valToday <- termStructure[,1] \* lag(wtC1) + termStructure[,2] \* lag(wtC2)

valYesterday <- lag(termStructure[,1]) \* lag(wtC1) + lag(termStructure[,2]) \* lag(wtC2)

syntheticRets <- (valToday/valYesterday) - 1

# on the day after roll, C2 becomes C1, so reflect that in returns

zeroes <- which(drDt$dr == 0) + 1

zeroRets <- termStructure[,1]/lag(termStructure[,2]) - 1

# override usual returns with returns that reflect back month becoming front month after roll day

syntheticRets[index(syntheticRets)[zeroes]] <- zeroRets[index(syntheticRets)[zeroes]]

syntheticRets <- na.omit(syntheticRets)

# vxxRets are syntheticRets

vxxRets <- syntheticRets

# repeat same process for vxz -- except it's dr/dt \* 4th contract + 5th + 6th + 1-dr/dt \* 7th contract

vxzToday <- termStructure[,4] \* lag(wtC1) + termStructure[,5] + termStructure[,6] + termStructure[,7] \* lag(wtC2)

vxzYesterday <- lag(termStructure[,4]) \* lag(wtC1) + lag(termStructure[, 5]) + lag(termStructure[,6]) + lag(termStructure[,7]) \* lag(wtC2)

syntheticVxz <- (vxzToday/vxzYesterday) - 1

# on zero expiries, next day will be equal (4+5+6)/lag(5+6+7) - 1

zeroVxz <- (termStructure[,4] + termStructure[,5] + termStructure[,6])/

lag(termStructure[,5] + termStructure[,6] + termStructure[,7]) - 1

syntheticVxz[index(syntheticVxz)[zeroes]] <- zeroVxz[index(syntheticVxz)[zeroes]]

syntheticVxz <- na.omit(syntheticVxz)

vxzRets <- syntheticVxz

# write out weights for actual execution

if(last(drDt$dr!=0)) {

print(paste("Previous front-month weight was", round(last(drDt$dr)/last(drDt$dt), 5)))

print(paste("Front-month weight at settle today will be", round((last(drDt$dr)-1)/last(drDt$dt), 5)))

if((last(drDt$dr)-1)/last(drDt$dt)==0){

print("Front month will be zero at end of day. Second month becomes front month.")

}

} else {

print("Previous front-month weight was zero. Second month became front month.")

print(paste("New front month weights at settle will be", round(last(expiryStructure[,2]-1)/last(expiryStructure[,2]), 5)))

}

return(list(vxxRets, vxzRets))

}

So, a big thank you goes out to [Michael Kapler](https://www.linkedin.com/in/michael-kapler-mmf-cfa-92a1a02/) of [Systematic Investor Toolbox](http://systematicinvestor.github.io/) for originally doing the replication and providing his code. My code essentially does the same thing, in, hopefully a more commented way.

So, ultimately, does it work? Well, using my updated term structure code, I can test that.

While I’m not going to paste my entire term structure code (again, [available here](https://quantstrattrader.wordpress.com/2017/04/27/creating-a-vix-futures-term-structure-in-r-from-official-cboe-settlement-data/), just update the script with my updates from this post), here’s how you’d run the new function:

> out <- syntheticXIV(termStructure, expiryStructure)

[1] "Previous front-month weight was 0.17647"

[1] "Front-month weight at settle today will be 0.11765"

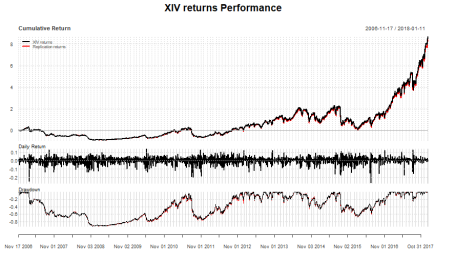
And since it returns both the vxx returns and the vxz returns, we can compare them both.

compareXIV <- na.omit(cbind(xivRets, out[[1]] \* -1))

colnames(compareXIV) <- c("XIV returns", "Replication returns")

charts.PerformanceSummary(compareXIV)

With the result:



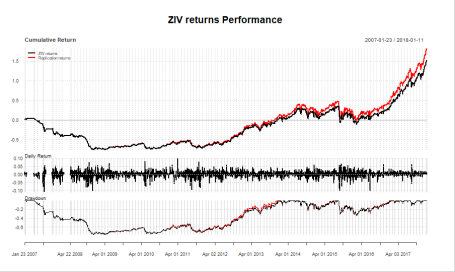
Basically, a perfect match.

Let’s do the same thing, with ZIV.

compareZIV <- na.omit(cbind(ZIVrets, out[[2]]\*-1))

colnames(compareZIV) <- c("ZIV returns", "Replication returns")

charts.PerformanceSummary(compareZIV)



So, rebuilding from the futures does a tiny bit better than the ETN. But the trajectory is largely identical.

That concludes this post. I hope it has shed some light on how these volatility ETNs work, and how to obtain them directly from the futures data published by the CBOE, which are the inputs to my term structure algorithm.

This also means that for institutions interested in trading my strategy, that they can obtain leverage to trade the futures-composite replicated variants of these ETNs, at greater volume.

Thanks for reading.