This post will detail a rather important finding I found while implementing a generalized framework for momentum asset allocation backtests. Namely, that when computing momentum (and other financial measures for use in asset allocation, such as volatility and correlations), measuring formal months, from start to end, has a large effect on strategy performance.

So, first off, I am in the job market, and am actively looking for a full-time role (preferably in New York City, or remotely), or a long-term contract. Here is my [resume](http://cutt.ly/FezrgO4), and here is my [LinkedIn profile](https://www.linkedin.com/in/ilyakipnis/). Furthermore, I’ve been iterating on my volatility strategy, and given that I’ve seen other services with large drawdowns, or less favorable risk/reward profiles charge $50/month, I think following my trades can be a reasonable portfolio diversification tool. [Read about it and subscribe here.](https://www.patreon.com/quantstrattrader) I believe that my body of work on this blog speaks to the viability of employing me, though I am also learning Python to try and port over my R skills over there, as everyone seems to want Python, and R much less so, hence the difficulty transferring between opportunities.

Anyhow, one thing I am working on is a generalized framework for tactical asset allocation (TAA) backtests. Namely, those that take the form of “sort universe by momentum, apply diversification weighting scheme”–namely, the kinds of strategies that the folks over at AllocateSmartly deal in. I am also working on this framework and am happy to announce that as of the time of this writing, **I will happily work with individuals that want more customized TAA backtests**, as the AllocateSmartly FAQs state that AllocateSmartly themselves do not do custom backtests. The framework I am currently in the process of implementing is designed to do just that. However, after going through some painstaking efforts to compare apples to apples, I came across a very important artifact. Namely, that there is a fairly large gulf in performance between measuring months from their formal endpoints, as opposed to simply approximating months with 21-day chunks (E.G. 21 days for 1 month, 63 for 3, and so on).

Here’s the code I’ve been developing recently–the long story short, is that the default options essentially default to [Adaptive Asset Allocation](https://allocatesmartly.com/adam-butler-gestaltu-adaptive-asset-allocation/), but depending on the parameters one inputs, it’s possible to get to something as simple as dual momentum (3 assets, invest in top 1), or as [complex as KDA](https://quantstrattrader.wordpress.com/2019/02/27/kda-robustness-results/), with options to fine-tune it even further, such as to account for the luck-based timing that [Corey Hoffstein at Newfound Research loves to write about](https://blog.thinknewfound.com/2019/07/timing-luck-and-systematic-value/) (speaking of whom, he and the awesome folks at ReSolve Asset Management have launched a new ETF called ROMO–Robust Momentum–I recently bought a bunch in my IRA because a buy-it-and-forget-it TAA ETF is pretty fantastic as far as buy-and-hold investments go). Again, I set a bunch of defaults in the parameters so that most of them can be ignored.

require(PerformanceAnalytics)

require(quantmod)

require(tseries)

stratStats <- function(rets) {

stats <- rbind(table.AnnualizedReturns(rets), maxDrawdown(rets))

stats[5,] <- stats[1,]/stats[4,]

stats[6,] <- stats[1,]/UlcerIndex(rets)

rownames(stats)[4] <- "Worst Drawdown"

rownames(stats)[5] <- "Calmar Ratio"

rownames(stats)[6] <- "Ulcer Performance Index"

return(stats)

}

getYahooReturns <- function(symbols, return\_column = "Ad") {

returns <- list()

for(symbol in symbols) {

getSymbols(symbol, from = '1990-01-01', adjustOHLC = TRUE)

if(return\_column == "Ad") {

return <- Return.calculate(Ad(get(symbol)))

colnames(return) <- gsub("\\.Adjusted", "", colnames(return))

} else {

return <- Return.calculate(Op(get(symbol)))

colnames(return) <- gsub("\\.Open", "", colnames(return))

}

returns[[symbol]] <- return

}

returns <- na.omit(do.call(cbind, returns))

return(returns)

}

symbols <- c("SPY", "VGK", "EWJ", "EEM", "VNQ", "RWX", "IEF", "TLT", "DBC", "GLD")

returns <- getYahooReturns(symbols)

canary <- getYahooReturns(c("VWO", "BND"))

# offsets endpoints by a certain amount of days (I.E. 1-21)

dailyOffset <- function(ep, offset = 0) {

ep <- ep + offset

ep[ep < 1] <- 1

ep[ep > nrow(returns)] <- nrow(returns)

ep <- unique(ep)

epDiff <- diff(ep)

if(last(epDiff)==1) {

# if the last period only has one observation, remove it

ep <- ep[-length(ep)]

}

return(ep)

}

# computes total weighted momentum and penalizes new assets (if desired)

compute\_total\_momentum <- function(yearly\_subset,

momentum\_lookbacks, momentum\_weights,

old\_weights, new\_asset\_mom\_penalty) {

empty\_vec <- data.frame(t(rep(0, ncol(yearly\_subset))))

colnames(empty\_vec) <- colnames(yearly\_subset)

total\_momentum <- empty\_vec

for(j in 1:length(momentum\_lookbacks)) {

momentum\_subset <- tail(yearly\_subset, momentum\_lookbacks[j])

total\_momentum <- total\_momentum + Return.cumulative(momentum\_subset) \*

momentum\_weights[j]

}

# if asset returns are negative, penalize by \*increasing\* negative momentum

# this algorithm assumes we go long only

total\_momentum[old\_weights == 0] <- total\_momentum[old\_weights==0] \*

(1-new\_asset\_mom\_penalty \* sign(total\_momentum[old\_weights==0]))

return(total\_momentum)

}

# compute weighted correlation matrix

compute\_total\_correlation <- function(data, cor\_lookbacks, cor\_weights) {

# compute total correlation matrix

total\_cor <- matrix(nrow=ncol(data), ncol=ncol(data), 0)

rownames(total\_cor) <- colnames(total\_cor) <- colnames(data)

for(j in 1:length(cor\_lookbacks)) {

total\_cor = total\_cor + cor(tail(data, cor\_lookbacks[j])) \* cor\_weights[j]

}

return(total\_cor)

}

# computes total weighted volatility

compute\_total\_volatility <- function(data, vol\_lookbacks, vol\_weights) {

empty\_vec <- data.frame(t(rep(0, ncol(data))))

colnames(empty\_vec) <- colnames(data)

# normalize weights if not already normalized

if(sum(vol\_weights) != 1) {

vol\_weights <- vol\_weights/sum(vol\_weights)

}

# compute total volrelation matrix

total\_vol <- empty\_vec

for(j in 1:length(vol\_lookbacks)) {

total\_vol = total\_vol + StdDev.annualized(tail(data, vol\_lookbacks[j])) \* vol\_weights[j]

}

return(total\_vol)

}

check\_valid\_parameters() {

if(length(mom\_weights) != length(mom\_lookbacks)) {

stop("Momentum weight length must be equal to momentum lookback length.") }

if(length(cor\_weights) != length(cor\_lookbacks)) {

stop("Correlation weight length must be equal to correlation lookback length.")

}

if(length(vol\_weights) != length(vol\_lookbacks)) {

stop("Volatility weight length must be equal to volatility lookback length.")

}

}

# computes weights as a function proportional to the inverse of total variance

invVar <- function(returns, lookbacks, lookback\_weights) {

var <- compute\_total\_volatility(returns, lookbacks, lookback\_weights)^2

invVar <- 1/var

return(invVar/sum(invVar))

}

# computes weights as a function proportional to the inverse of total volatility

invVol <- function(returns, lookbacks, lookback\_weights) {

vol <- compute\_total\_volatility(returns, lookbacks, lookback\_weights)

invVol <- 1/vol

return(invVol/sum(invVol))

}

# computes equal weight portfolio

ew <- function(returns) {

return(StdDev(returns)/(StdDev(returns)\*ncol(returns)))

}

# computes minimum

minVol <- function(returns, cor\_lookbacks, cor\_weights, vol\_lookbacks, vol\_weights) {

vols <- compute\_total\_volatility(returns, vol\_lookbacks, vol\_weights)

cors <- compute\_total\_correlation(returns, cor\_lookbacks, cor\_weights)

covs <- t(vols) %\*% as.numeric(vols) \* cors

min\_vol\_rets <- t(matrix(rep(1, ncol(covs))))

min\_vol\_wt <- portfolio.optim(x=min\_vol\_rets, covmat = covs)$pw

names(min\_vol\_wt) <- rownames(covs)

return(min\_vol\_wt)

}

asset\_allocator <- function(returns,

canary\_returns = NULL, # canary assets for KDA algorithm and similar

mom\_threshold = 0, # threshold momentum must exceed

mom\_lookbacks = 126, # momentum lookbacks for custom weights (EG 1-3-6-12)

# weights on various momentum lookbacks (EG 12/19, 4/19, 2/19, 1/19)

mom\_weights = rep(1/length(mom\_lookbacks),

length(mom\_lookbacks)),

# repeat for correlation weights

cor\_lookbacks = mom\_lookbacks, # correlation lookback

cor\_weights = rep(1/length(mom\_lookbacks),

length(mom\_lookbacks)),

vol\_lookbacks = 20, # volatility lookback

vol\_weights = rep(1/length(vol\_lookbacks),

length(vol\_lookbacks)),

# number of assets to hold (if all above threshold)

top\_n = floor(ncol(returns)/2),

# diversification weight scheme (ew, invVol, invVar, minVol, etc.)

weight\_scheme = "minVol",

# how often holdings rebalance

rebalance\_on = "months",

# how many days to offset rebalance period from end of month/quarter/year

offset = 0,

# penalize new asset mom to reduce turnover

new\_asset\_mom\_penalty = 0,

# run Return.Portfolio, or just return weights?

# for use in robust momentum type portfolios

compute\_portfolio\_returns = TRUE,

verbose = FALSE,

# crash protection asset

crash\_asset = NULL,

...

) {

# normalize weights

mom\_weights <- mom\_weights/sum(mom\_weights)

cor\_weights <- cor\_weights/sum(cor\_weights)

vol\_weights <- vol\_weights/sum(vol\_weights)

# if we have canary returns (I.E. KDA strat), align both time periods

if(!is.null(canary\_returns)) {

smush <- na.omit(cbind(returns, canary\_returns))

returns <- smush[,1:ncol(returns)]

canary\_returns <- smush[,-c(1:ncol(returns))]

empty\_canary\_vec <- data.frame(t(rep(0, ncol(canary\_returns))))

colnames(empty\_canary\_vec) <- colnames(canary\_returns)

}

# get endpoints and offset them

ep <- endpoints(returns, on = rebalance\_on)

ep <- dailyOffset(ep, offset = offset)

# initialize vector holding zeroes for assets

empty\_vec <- data.frame(t(rep(0, ncol(returns))))

colnames(empty\_vec) <- colnames(returns)

weights <- empty\_vec

# initialize list to hold all our weights

all\_weights <- list()

# get number of periods per year

switch(rebalance\_on,

"months" = { yearly\_periods = 12},

"quarters" = { yearly\_periods = 4},

"years" = { yearly\_periods = 1})

for(i in 1:(length(ep) - yearly\_periods)) {

# remember old weights for the purposes of penalizing momentum of new assets

old\_weights <- weights

# subset one year of returns, leave off first day

return\_subset <- returns[c((ep[i]+1):ep[(i+yearly\_periods)]),]

# compute total weighted momentum, penalize potential new assets if desired

momentums <- compute\_total\_momentum(return\_subset,

momentum\_lookbacks = mom\_lookbacks,

momentum\_weights = mom\_weights,

old\_weights = old\_weights,

new\_asset\_mom\_penalty = new\_asset\_mom\_penalty)

# rank negative momentum so that best asset is ranked 1 and so on

momentum\_ranks <- rank(-momentums)

selected\_assets <- momentum\_ranks <= top\_n & momentums > mom\_threshold

selected\_subset <- return\_subset[, selected\_assets]

# case of 0 valid assets

if(sum(selected\_assets)==0) {

weights <- empty\_vec

} else if (sum(selected\_assets)==1) {

# case of only 1 valid asset -- invest everything into it

weights <- empty\_vec + selected\_assets

} else {

# apply a user-selected weighting algorithm

# modify this portion to select more weighting schemes

if (weight\_scheme == "ew") {

weights <- ew(selected\_subset)

} else if (weight\_scheme == "invVol") {

weights <- invVol(selected\_subset, vol\_lookbacks, vol\_weights)

} else if (weight\_scheme == "invVar"){

weights <- invVar(selected\_subset, vol\_lookbacks, vol\_weights)

} else if (weight\_scheme == "minVol") {

weights <- minVol(selected\_subset, cor\_lookbacks, cor\_weights,

vol\_lookbacks, vol\_weights)

}

}

# include all assets

wt\_names <- names(weights)

if(is.null(wt\_names)){wt\_names <- colnames(weights)}

zero\_weights <- empty\_vec

zero\_weights[wt\_names] <- weights

weights <- zero\_weights

weights <- xts(weights, [order.by](http://order.by)=last(index(return\_subset)))

# if there's a canary universe, modify weights by fraction with positive momentum

# if there's a safety asset, allocate the crash protection modifier to it.

if(!is.null(canary\_returns)) {

canary\_subset <- canary\_returns[c(ep[i]:ep[(i+yearly\_periods)]),]

canary\_subset <- canary\_subset[-1,]

canary\_mom <- compute\_total\_momentum(canary\_subset,

mom\_lookbacks, mom\_weights,

empty\_canary\_vec, 0)

canary\_mod <- mean(canary\_mom > 0)

weights <- weights \* canary\_mod

if(!is.null(crash\_asset)) {

if(momentums[crash\_asset] > mom\_threshold) {

weights[,crash\_asset] <- weights[,crash\_asset] + (1-canary\_mod)

}

}

}

all\_weights[[i]] <- weights

}

# combine weights

all\_weights <- do.call(rbind, all\_weights)

if(compute\_portfolio\_returns) {

strategy\_returns <- Return.portfolio(R = returns, weights = all\_weights, verbose = verbose)

return(list(all\_weights, strategy\_returns))

}

return(all\_weights)

}

#out <- asset\_allocator(returns, offset = 0)

kda <- asset\_allocator(returns = returns, canary\_returns = canary,

mom\_lookbacks = c(21, 63, 126, 252),

mom\_weights = c(12, 4, 2, 1),

cor\_lookbacks = c(21, 63, 126, 252),

cor\_weights = c(12, 4, 2, 1), vol\_lookbacks = 21,

weight\_scheme = "minVol",

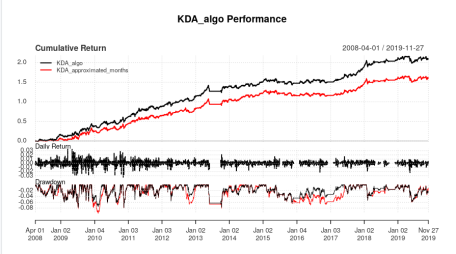
crash\_asset = "IEF")

The one thing that I’d like to focus on, however, are the lookback parameters. Essentially, assuming daily data, they’re set using a \*daily lookback\*, as that’s what AllocateSmartly did [when testing my own KDA Asset Allocation algorithm.](https://allocatesmartly.com/ilya-kipnis-defensive-adaptive-asset-allocation/) Namely, the salient line is this:

“For all assets across all three universes, at the close on the last trading day of the month, calculate a “momentum score” as follows:(12 \* (p0 / p21 – 1)) + (4 \* (p0 / p63 – 1)) + (2 \* (p0 / p126 – 1)) + (p0 / p252 – 1)Where p0 = the asset’s price at today’s close, p1 = the asset’s price at the close of the previous trading day and so on. 21, 63, 126 and 252 days correspond to 1, 3, 6 and 12 months.”

So, to make sure I had apples to apples when trying to generalize KDA asset allocation, I compared the output of my new algorithm, asset\_allocator (or should I call it allocate\_smartly ?=] ) to my formal KDA asset allocation algorithm.

Here are the results:



KDA\_algo KDA\_approximated\_months

Annualized Return 0.10190000 0.08640000

Annualized Std Dev 0.09030000 0.09040000

Annualized Sharpe (Rf=0%) 1.12790000 0.95520000

Worst Drawdown 0.07920336 0.09774612

Calmar Ratio 1.28656163 0.88392257

Ulcer Performance Index 3.78648873 2.62691398

Essentially, the long and short of it is that I modified my original KDA algorithm until I got identical output to my asset\_allocator algorithm, then went back to the original KDA algorithm. The salient difference is this part:

# computes total weighted momentum and penalizes new assets (if desired)

compute\_total\_momentum <- function(yearly\_subset,

momentum\_lookbacks, momentum\_weights,

old\_weights, new\_asset\_mom\_penalty) {

empty\_vec <- data.frame(t(rep(0, ncol(yearly\_subset))))

colnames(empty\_vec) <- colnames(yearly\_subset)

total\_momentum <- empty\_vec

for(j in 1:length(momentum\_lookbacks)) {

momentum\_subset <- tail(yearly\_subset, momentum\_lookbacks[j])

total\_momentum <- total\_momentum + Return.cumulative(momentum\_subset) \*

momentum\_weights[j]

}

# if asset returns are negative, penalize by \*increasing\* negative momentum

# this algorithm assumes we go long only

total\_momentum[old\_weights == 0] <- total\_momentum[old\_weights==0] \*

(1-new\_asset\_mom\_penalty \* sign(total\_momentum[old\_weights==0]))

return(total\_momentum)

}

Namely, the part that further subsets the yearly subset by the lookback period, in terms of days, rather than monthly endpoints. Essentially, the difference in the exact measurement of momentum–that is, the measurement that explicitly selects \*which\* instruments the algorithm will allocate to in a particular period, unsurprisingly, has a large impact on the performance of the algorithm.

And lest anyone think that this phenomena no longer applies, here’s a yearly performance comparison.

KDA\_algo KDA\_approximated\_months

2008-12-31 0.1578348930 0.062776766

2009-12-31 0.1816957178 0.166017499

2010-12-31 0.1779839604 0.160781537

2011-12-30 0.1722014474 0.149143148

2012-12-31 0.1303019332 0.103579674

2013-12-31 0.1269207487 0.134197066

2014-12-31 0.0402888320 0.071784979

2015-12-31 -0.0119459453 -0.028090873

2016-12-30 0.0125302658 0.002996917

2017-12-29 0.1507895287 0.133514924

2018-12-31 0.0747520266 0.062544709

2019-11-27 0.0002062636 0.008798310

Of note: the variant that formally measures momentum from monthly endpoints consistently outperforms the one using synthetic monthly measurements.

So, that will do it for this post. I hope to have a more thorough walk-through of the asset\_allocator function in the very near future before moving onto Python-related matters (hopefully), but I thought that this artifact, and just how much it affects outcomes, was too important not to share.

An iteration of the algorithm capable of measuring momentum with proper monthly endpoints should be available in the near future.

Thanks for reading.