Targeting Risk; Volatility and Leverage Management

HangukQuant[1, 2*]

August 30, 2022

[1] [2]

Interim Report

Abstract

A trader's primary job is to identify edge and then execute them. In this paper we take a preliminary look at risk management and implement two methods - volatility targeting and leverage targeting. We make interesting observations and provide prelude to future discussions in portfolio management.

[1]***1**: hangukquant@gmail.com, hangukquant.substack.com

[2]***2**: DISCLAIMER: the contents of this work are not intended as investment, legal, tax or any other advice, and is for informational purposes only. It is illegal to make unauthorized copies, forward to an unauthorized user or to post this article electronically without express written consent by HangukQuant.

Contents

1 Introduction

Portfolio management is a multi-variate goal. Some traders' goal might be regulated by exogenous pressures such as investor preference and fund mandate. Some traders' goal is to achieve the highest possible risk-adjusted returns. Some traders only considers the terminal wealth; others are path dependent. Wealth drawdown and duration of under-performance is a concern for many. Portfolio management can be applied at the portfolio level, strategy level and asset level. Different (& often similar) techniques may be applied at each level of management.

We reduce the dimensionality of our problem and only consider the risk-management at the strategy level. We ignore their diversification effects on the overall portfolio and study a univariate strategy portfolio consisting of many assets. We intentionally keep our study simple as a prelude to further discussions and explore other issues in the future. Our implementation compares two variants of volatility targeting and one of leverage targeting. Python code is demonstrated.

2 Facts & Empirical Facts

2.1 Moment Stickiness

The different return moments have different stickiness. Some cluster and others do not. Stickiness of moments determine whether market observables are useful in predicting future randomness. If information today carries no value in predicting variables for tomorrow, then it is likely that including estimations of said variable in an alpha/risk model would lead to noisy results. We take a look. We consider 2 strategies, asset universe from *GSPC* index constituents with data from Jan 1 2010 to Jan 1 2020. Let's first treat the strategies as black boxes, and we use the strategy daily returns as our experimental time series. We discuss the play-strategies later in the paper.

In order to see if there is autocorrelation (information) in today's variables about tomorrow, we can create autocorrelation plots using (partial) ACF functions. We test up to 3Δ lag of return moments up to the 4^{th}. In addition, we can get their (un)likelihoods with p-values obtained from Ljung-Box, with indication that time-series is autocorrelated if the p-value is lower than some α significance.

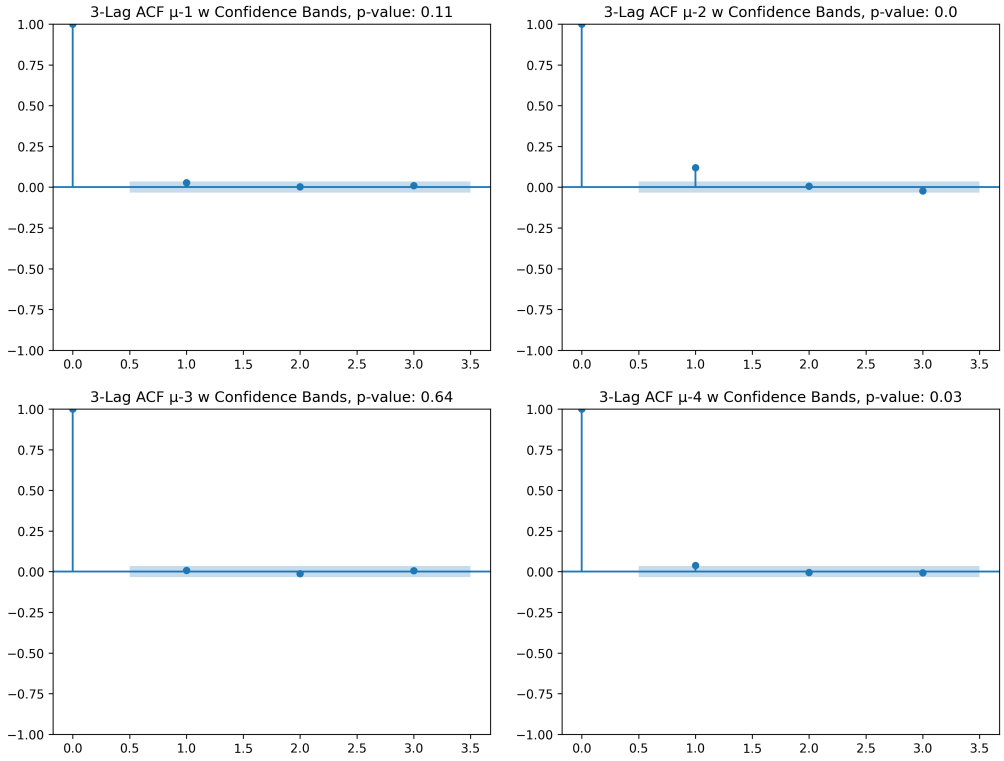

Figure 1: A Single Trial of Strategy Run: ACF Plots (Confidence Bands) - Return Moment $\mu_{1..4}$

We consider 2 strategies. For each strategy, we implement 3 risk control variants and run 13 trials by randomly sampling 100 tickers from the *GSPC* dataset. The details are not important here, but rather we want to see the difference in moment distributions. For each of the 39 runs, we run ACF and Ljung-Box and test what percentage of the time-series return moments exhibit memory.

2.1.1 Strategy 1

$p \backslash \mu_k$	$k=1$	$k=2$	$k=3$	$k=4$
$p \leq 0.05$	3	39	5	7
$p \geq 0.05$	36	0	34	32
$\bar{p}$	0.42	0.00	0.38	0.40

Table 1: Strategy 1, Summary of Auto-correlation Tests (Ljung-Box)

2.1.2 Strategy 2

$p \backslash \mu_k$	$k=1$	$k=2$	$k=3$	$k=4$
$p \leq 0.05$	9	39	19	22
$p \geq 0.05$	30	0	20	17
$\bar{p}$	0.35	0.00	0.22	0.18

Table 2: Strategy 2, Summary of Auto-correlation Tests (Ljung-Box)

We see clearly that the second moment of return series μ_2 is auto-correlated. All 39 trials (combinations of assets and risk management variants) on both strategies mathematically verfies that volatility 'clusters', not only on asset returns but also on portfolio returns. This is fairly intutive, a portfolio/strategy trades assets, and if underlying asset has volatility clusters then so would the portfolio. This is followed by the fourth moment, the third moment and the first moment. In increasing fashion, auto-correlative exhibits in order of μ_2, μ_4, μ_3, μ_1.

A more important inference might be that when we are doing risk-modelling, our risk model should take into consideration the more 'informational' moments as inputs. Including moments which carry less information about the future introduces noise to our risk management since the future will have no memory about the related variables.

2.2 Geometric Wealth Maximization

Many would be tempted to stop reading the paper upon reading this line - we assume normality of returns. We are not particularly interested in satisfying market assumptions yet. Our focus is on comparison of the different approaches (which may later be refined to relax these assumptions for practical usage) and how they measure up. Then the assumption states that the wealth distribution is log-normal, with geometric mean $\left(\mu - \frac{\sigma^2}{2}\right)$ (see *Hull*). The variance drag is the σ-penalty term. Under leverage factor L, this term evaluates to $\left(L\mu - \frac{(L\sigma)^2}{2}\right)$, and the first order derivative $w.r.t\ L$ yields $\left(\mu - L\sigma^2\right)$ and our optimal leverage equals $\frac{\mu}{\sigma^2}$. We may relax the normality assumptions and perform similar analysis to obtain leverage under considerations of non-trivial skew and kurtosis.

3 Risk Management

Risk management roughly applies to (i) 'how much of our capital should we allocate to a strategy' and (ii) 'how much of our capital should we allocate to an asset'?. There is extensive literature on these topics. We explore 2 main options for (i) and consider 1 for (ii).

Answering (ii): Suppose we are trading an effect that is harvested by trading a basket of assets. Each asset is labelled an alpha score representing their return-expectation relative to other assets in the basket as computed by the strategy. This can be generalized to different strengths of alpha, but assume unit alpha-measure such that their return expectancy is labelled ± 1, 1 representing we want to long the asset and -1 representing short. A reasonable approach would be to take position sizing inverse to their dollar volatilities, such that the each forecast estimate affects the portfolio's future capital by roughly the same amount. If an asset A is half as volatile as asset B and their return expectation is the same, we want to bet twice in A as we did in B, such that if we were right in one bet and wrong in the other, we would end up roughly the same. This is reasonable assuming we have no reason to be more confident in one position than the other. This is the approach we would adopt.

Answering (i): How much capital we want to allocate to a strategy depends on the trader's goal; whether it is to maximise terminal wealth, be aggressive or target a certain degree of variability in their capital per unit time. We discuss some options.

3.1 Volatility Targeting

This is the method whereby the trader targets a certain level of volatility (variability) in his capital. It is unlikely he can predict returns with high accuracy (or he would have been rich) but he can control the degree to which his capital varies to a reasonable degree (by betting big or small). We discussed this before, but an important rule of thumb is that our wealth (modelled as a Brownian motion) gains quadratic variation per unit time. Equivalently volatility scales as a square root of time. Assuming 253 trading days roughly per year, volatility computed at $y\%$ with daily returns has annualized volatility estimate of $\sqrt{253} * y\%$. For example, we may want to say 'let volatility (variability) of our portfolio returns to be about 15% per year'. We can then compute the desired vol at the daily level by taking $15\%/\sqrt{253}$ and hope to scale our positions such that the standard deviation of our daily returns match this value. We can do this on two separate levels to illustrate where our risk is coming from. First, we can target this volatility level by dividing the capital we have by the cardinality of our traded universe and bet in accordance to how much risk we want to allocate to the position. However, by taking these decisions individually for each position without taking into consideration other traded assets in the strategy, we would not be able to achieve our target-ed level of volatility. We would systematically obtain lower volatility levels on the overall portfolio returns as we did not take into consideration diversification effects and (possibly) directionally opposite trades that neutralise market beta. Our two variants on volatility targeting would implement a strategy-level scalar that compensates this volatility loss from diversification effects. Both variants are actually quite similar but have implicit differences when implemented.

3.1.1 Variant 1

The first variant would be to take the empiricist approach. We do not have a model to compute the *strat-scalar*. Instead, we first assign an arbitrary scalar, and start trading. We can then take the market observables, which is our realised daily returns from the traded strategy and then accordingly increase or decrease our scalar up to try to 'hit' our target. The steps are in high-level (i) how much did our portfolio vary in the last x days, (ii) how much were we scaling when we achieved those variability in the last x days, and (iii) so do we need to increase or decrease our scalar from our baseline in (ii) to hit our target?

3.1.2 Variant 2

The second variant would be to compute forward-1 volatility with a volatility model such as GARCH, EWMA *etc* and then deciding our scalar as our volatility target divided by our estimate. We would implement the EWMA model with $\lambda = 0.94$ as is commonly cited in practice (by RiskMetrics). For those who do not have experience with volatility estimations, Google *time-series volatility prediction*. We would not spare time covering such a common topic. We also ignore the subtle differences between the time-series models, they are all somewhat similar and it is likely that the most stable out-of-sample is an ensemble. Our choice of EWMA was for speed considerations, which (refer literature) is computed $\sigma_t^2 = 0.06u_t^2 + 0.94\sigma_{t-1}^2$ and can be computed with an on-line update as opposed to rolling computations. The space and time requirement is also constant, while other methods require the entire array of values for the look-back period/scale linearly with lookback-window length.

3.2 Leverage Targeting

3.2.1 Variant 3

Recall in Section 2.2 that in the continuous return space, there is volatility drag and high levels of variance (which scales with leverage) can affect performance. We then found the optimal leverage is $\frac{\mu}{\sigma^2}$ - but also noted in the earlier discussion that μ, the first return moment has little memory, and therefore might introduce noise to our risk model. We take a look at its implementation and briefly see how it performs.

3.2.1.1 A Side Note: when implementing this in practise, the return factor is too noisy such that the leverage would have significant ranges. We cap the leverage from both sides, so that we would not end up with unreasonable leverage values.

3.3 Differences in Volatility vs Leverage Targeting

Note that other than the obvious differences between the two risk management approaches, one is ex-ante and the other is ex-post. With vol-targeting, we are issuing a strategy scalar that impacts

leverage, while in leverage targeting we are first computing our relative allocations and fixing the leverage for wealth maximisation.

4 Code Implementation

First, we begin by importing some custom libraries discussed before on *HangukQuant*. However, these libraries have nothing to do with the implementation itself and can be replaced with your own code. They have just been included such that our regular readers can reference them if needed. We attempt to get a *dict* of OHLCV dataframes for a list of tickers from our database.

```
import asyncio
import random
import datetime
import numpy as np
import pandas as pd
import multiprocessing as mp
import matplotlib.pyplot as plt

#custom libraries
import general_utils as gu #simple I/O and pickling
import data_service.data_master as DM #from our data service paper

trade_start = datetime.datetime(2010, 1, 1)
trade_end = datetime.datetime(2020, 1, 1)

#get a dictionary of ohlcvs for instruments
async def batch_insert_ohlcv(loop, instruments):
    LIMIT = 250
    ohlcvs = {}
    for i in range(0, len(instruments), LIMIT):
        temp_insts = instruments[i : i + LIMIT]
        temp_ohlcvs = await data_master
            .get_equity_service()
            .asyn_batch_get_ohlcv(
                tickers=temp_insts,
                exchanges=list(["US" for _ in range(len(temp_insts))]),
```

```
                period_start=trade_start,
                period_end=trade_end,
                read_db=True,
                insert_db=True
            )
        for i in range(len(temp_insts)):
            ohlcvs[temp_insts[i]] = temp_ohlcvs[i]
    return ohlcvs
```

We do some set up to generate random 100-len asset universe for 3 studies each on 13 trials for our testing purposes. We use the *Alpha* object, which is the core of our backtesting engine.

```
if __name__ == "__main__":
    load_disk = True
    data_master = DM.DataMaster()
    res = data_master.get_basket_service().get_index_components("GSPC")
    instruments = list(res.Code)
    if not load_disk:
        loop = asyncio.get_event_loop()
        ohlcvs = loop.run_until_complete(batch_insert_ohlcv(loop, instruments))
        gu.save_file("ohlcvs.pickle", ohlcvs)
    else:
        ohlcvs = gu.load_file("ohlcvs.pickle")

    for inst in instruments:
        if ohlcvs[inst] is None or len(ohlcvs[inst]) == 0:
            print(inst)
            del ohlcvs[inst]
        else:
            ohlcvs[inst].set_index("datetime", inplace=True)

    random.seed(1)
    trial = 0

    while trial < 13:
        print("trial :", trial)
        instruments = random.sample(list(ohlcvs.keys()), 100)
        print("performing study with size-{} universe".format(len(instruments)))
```

```
        alphas = [Alpha(
            instruments=instruments,
            dfs=ohlcvs,
            configs={
                "start" : trade_start,
                "end": trade_end,
                "longsize": 33,
                "shortsize": 33,
                "study": i
            }) for i in range(1, 4)]

        def run_strat(alpha):
            portfolio_df = alpha.run_simulation()
            return portfolio_df

        with mp.Manager() as manager:
            with mp.Pool(mp.cpu_count()) as pool:
                records = pool.map(run_strat, alphas)

        gu.save_file("{}_.pickle".format(trial), (instruments, records))
        trial += 1
```

Before we introduce the core of our backtesting engine, we discuss some testing quirks and introduce the 2 play-strategies.

4.1 Dynamic Asset Universe

Firstly, we don't know that our dataset consists of OHLCVs that span the entire period. Some assets might only have been listed halfway though the testing period, while others might have halted trading in the middle. We also don't know that their traded days are uniform (for instance FX trading holidays are different NYSE holidays). We fix this with a simple trick: we know that the calendar dates are invariant across assets traded - we create a date range from the start date to end date and 'slot' our OHLCV measurements into the corresponding dates. Note that this would create empty rows for which we do not have measurments, such as weekends or when the asset ceased to exist/trade for that period. We then use a simple heuristic to filter out the actively

traded stocks: by labelling a measurement as a valid 'sample' if the close of that day is different from the prior sample. If there are continuous stretches of 5 or more dates where our samples are not varying, then we flag them as inactive and exclude them from our traded universe. Excluding inactive stocks from our asset universe is important - since their randomness degenerates and the volatility of their returns go to zero; taking positions inversely proportional to their volatility sends positions to *infinity*.

This quick and dirty trick however is not cost-less, we need to ensure that it does not affect the implementation of our strategy and risk management. By doing a naive $\sigma_t^2 = 0.06u_t^2 + 0.94\sigma_{t-1}^2$ the u_t^2 would evaluate to 0 when t is a corresponding weekend or market holiday. This would cause our σ_t^2 to decrease systematically over trading breaks (and hence systematically increase position sizing on Mondays/after holidays) and cause a 'Sizing Weekend Effect'. In order to prevent this quirk, we need to forward fill our volatility estimates when we know that our portfolio has not been traded on that date. The same applies to our volatility calculation for individual asset returns that are used in our risk control - we need to forward fill volatility computations before forward fill-ing the closing prices (for instance, *line 15* in the code below comes before *line 21*). Other computations such as computing performance statistics need a similar adjustment.

4.2 2-Play Strategies

The first is a $\Delta 1$-scaled mean reversion, encoded

'div(minus(open , close) , minus(high , low))'

and the next is a generic cross-sectional 1-Y momentum strategy.

```
"""
Strategy 1
"""
self.dfs[inst]["alpha"] =
    (self.dfs[inst]["open"] - self.dfs[inst]["close"])
    / (self.dfs[inst]["high"] - self.dfs[inst]["low"])

"""
Strategy 2
"""
```

```
self.dfs[inst]["alpha"] =
    -1 +
    self.dfs[inst]["adj_close"]
    / self.dfs[inst]["adj_close"].shift(365)
```

We are not really concerned about their attractiveness - obviously there are trivial assumptions that do not follow practical trading considerations. For instance, assuming fill on close is not feasible, but for our demonstrations these would be assumed away.

4.3 Alpha Testing

And most importantly:

```
class Alpha():

    def __init__(self, instruments, dfs, configs):
        self.instruments = instruments
        self.dfs = dfs
        self.configs = configs

    def get_trade_datetime_range(self):
        return (self.configs["start"], self.configs["end"])

    def compute_metas(self, index):
        for inst in self.instruments:
            print(inst)
            df = pd.DataFrame(index=index)
            self.dfs[inst]["vol"] =
            (
                -1 + self.dfs[inst]["adj_close"]
                / self.dfs[inst].shift(1)["adj_close"]
            ).rolling(30).std()
            self.dfs[inst] = df.join(self.dfs[inst])
            self.dfs[inst] = self.dfs[inst]
                .fillna(method="ffill")
                .fillna(method="bfill")
            self.dfs[inst]["ret"] = -1 +
                self.dfs[inst]["adj_close"] / self.dfs[inst].shift(1)["adj_close"]
```

```
            self.dfs[inst]["sampled"] = self.dfs[inst]["adj_close"] !=
                self.dfs[inst].shift(1)["adj_close"]
            self.dfs[inst]["active"] = self.dfs[inst]["sampled"]
                .rolling(5)
                .apply(lambda x: int(np.any(x))).fillna(0)

            """
            Strategy 1
            """
            self.dfs[inst]["alpha"] =
                (self.dfs[inst]["open"] - self.dfs[inst]["close"])
                / (self.dfs[inst]["high"] - self.dfs[inst]["low"])

            """
            Strategy 2
            """
            self.dfs[inst]["alpha"] =
                -1 +
                self.dfs[inst]["adj_close"]
                / self.dfs[inst]["adj_close"].shift(365)

            self.dfs[inst]["eligible"] =
                (
                ~np.isnan(self.dfs[inst]["alpha"]) &
                self.dfs[inst]["active"] &
                self.dfs[inst]["vol"] > 0
                )

    def init_portfolio_settings(self, trade_range):
        portfolio_df = pd.DataFrame(index=trade_range)
            .reset_index()
            .rename(columns={"index" : "datetime"})
        portfolio_df.loc[0, "capital"] = 10000
        portfolio_df.loc[0, "ewma"] = 0.001
        return portfolio_df

    def compute_eligibles_and_comp(self, date):
```

```
        eligibles = [inst for inst in self.instruments
            if self.dfs[inst].loc[date, "eligible"]
        ]
        non_eligibles = [inst for inst in self.instruments
            if not self.dfs[inst].loc[date, "eligible"]
        ]
        return eligibles, non_eligibles

    def get_lever(self, portfolio_df, instruments, date, idx, nominal_tot,
    leverage, lookback):

        nominal_ret_hist = portfolio_df[:idx]["nominal ret"]
            .replace(0, np.nan)
            .dropna()
            .tail(lookback)

        if len(nominal_ret_hist) == lookback:
            sigma = nominal_ret_hist.std()
            lever = max(0.5, nominal_ret_hist.mean() / (sigma**2))
            lever = min(10, lever)
            leverage_scalar = lever / leverage
            for inst in instruments:
                newpos = portfolio_df.loc[idx, "{} units".format(inst)]
                    * leverage_scalar
                portfolio_df.loc[idx, "{} units".format(inst)] = newpos
            return nominal_tot * leverage_scalar
        else:
            return nominal_tot

    def get_strat_scaler(self, portfolio_df, lookback, portfolio_vol, idx, default
    , study=1):

        if study == 1:
            capital_ret_hist = portfolio_df[:idx]["capital ret"]
                .replace(0, np.nan)
                .dropna()
                .tail(lookback)
```

```
            strat_scalar_hist = portfolio_df["strat scalar"]
                .loc[capital_ret_hist.index]
            if len(capital_ret_hist) == lookback:
                ann_realized_vol = capital_ret_hist.std() * np.sqrt(253)
                scalar_hist = np.mean(strat_scalar_hist)
                strat_scalar = scalar_hist * portfolio_vol / ann_realized_vol
                return strat_scalar
            else:
                return default
        elif study == 2:
            ann_realized_vol = np.sqrt(portfolio_df.loc[idx - 1, "ewma"] * 252)
            return  portfolio_vol / ann_realized_vol
        else:
            pass

    def get_pnl_stats(
        self,
        date,
        prev,
        portfolio_df,
        instruments,
        idx,
        historicals,
        close_col="adj_close"
    ):
        day_pnl = 0
        nominal_ret = 0
        for inst in instruments:
            units_held = portfolio_df.loc[idx - 1, inst + " units".format(inst)]
            if units_held != 0:
                delta_price = historicals[inst].loc[date, close_col] -
                    historicals[inst].loc[prev, close_col]
                inst_pnl = delta_price * units_held
                day_pnl += inst_pnl
                nominal_ret += portfolio_df.loc[idx - 1, inst + " w"]
                    * historicals[inst].loc[date, "ret"]

```

```
        capital_ret = nominal_ret * portfolio_df.loc[idx - 1, "leverage"]
        portfolio_df.loc[idx, "capital"] =
            portfolio_df.loc[idx - 1, "capital"] + day_pnl
        portfolio_df.loc[idx, "daily pnl"] = day_pnl
        portfolio_df.loc[idx, "nominal ret"] = nominal_ret
        portfolio_df.loc[idx, "capital ret"] = capital_ret
        portfolio_df.loc[idx, "ewma"] = 0.06 * (nominal_ret**2) +
            0.94 * portfolio_df.loc[idx - 1, "ewma"] \
            if nominal_ret != 0 else portfolio_df.loc[idx - 1, "ewma"]
        return day_pnl, nominal_ret

    def run_simulation(self):
        """
        Settings
        """
        portfolio_vol = 0.20
        trade_datetime_range = pd.date_range(
            start=self.get_trade_datetime_range()[0],
            end=self.get_trade_datetime_range()[1],
            freq="D"
        )

        """
        Compute Metas
        """
        self.compute_metas(index=trade_datetime_range)

        """
        Initialisations
        """
        portfolio_df = self.init_portfolio_settings(
            trade_range=trade_datetime_range)

        for i in portfolio_df.index:
            date = portfolio_df.loc[i, "datetime"]
            strat_scalar = 2

```

```
            eligibles, non_eligibles = self.compute_eligibles_and_comp(date=date)

            if i != 0:
                date_prev = portfolio_df.loc[i - 1 , "datetime"]
                day_pnl, nominal_ret = self.get_pnl_stats(
                    date=date,
                    prev=date_prev,
                    portfolio_df=portfolio_df,
                    instruments=self.instruments,
                    idx=i,
                    historicals=self.dfs,
                    close_col="adj_close"
                )
                if self.configs["study"] in [1, 2]:
                    strat_scalar = self.get_strat_scaler(
                        portfolio_df=portfolio_df,
                        lookback=30,
                        portfolio_vol=portfolio_vol,
                        idx=i,
                        default=strat_scalar,
                        study=self.configs["study"]
                    )
            portfolio_df.loc[i, "strat scalar"] = strat_scalar

            alpha_scores = {}
            for inst in eligibles:
                alpha_scores[inst] = self.dfs[inst].loc[date, "alpha"]

            alpha_scores = {k: v for k,v in
                sorted(alpha_scores.items(), key=lambda pair: pair[1])}
            alphalong = list(alpha_scores.keys())[-self.configs["longsize"]:]
            alphashort = list(alpha_scores.keys())[:self.configs["shortsize"]]

            for inst in non_eligibles:
                portfolio_df.loc[i , "{} w".format(inst)] = 0
                portfolio_df.loc[i , "{} units".format(inst)] = 0

```

```
            nominal_tot = 0
            for inst in eligibles:
                forecast = 1 if inst in alphalong else 0
                forecast = -1 if inst in alphashort else forecast
                vol_target=1/(self.configs["longsize"]+self.configs["shortsize"])
                    * portfolio_df.loc[i, "capital"]
                    * portfolio_vol / np.sqrt(253)
                dollar_vol =  self.dfs[inst].loc[date, "vol"]
                    * self.dfs[inst].loc[date, "adj_close"]
                position = strat_scalar * forecast * vol_target / dollar_vol
                portfolio_df.loc[i, inst + " units"] = position
                nominal_tot += abs(
                    position * self.dfs[inst].loc[date, "adj_close"]
                )

            for inst in eligibles:
                units = portfolio_df.loc[i, "{} units".format(inst)]
                nominal_inst = units * self.dfs[inst].loc[date, "adj_close"]
                inst_w = nominal_inst / nominal_tot
                portfolio_df.loc[i, "{} w".format(inst)] = inst_w

            if nominal_tot > 0 and self.configs["study"] == 3:
                nominal_tot = self.get_lever(
                    portfolio_df=portfolio_df,
                    instruments=self.instruments,
                    date=date,
                    idx=i,
                    nominal_tot=nominal_tot,
                    leverage=nominal_tot / portfolio_df.loc[i, "capital"],
                    lookback=30
                )

            portfolio_df.loc[i, "nominal"] = nominal_tot
            portfolio_df.loc[i, "leverage"] = portfolio_df.loc[i, "nominal"]
                / portfolio_df.loc[i, "capital"]
            print(portfolio_df.loc[i])

```

```
246         return portfolio_df
```

5 Performance

First and foremost, how well did our risk management strategies track target volatility? We compare variant 1 and variant 2 of volatility targeting measure on both strategies.

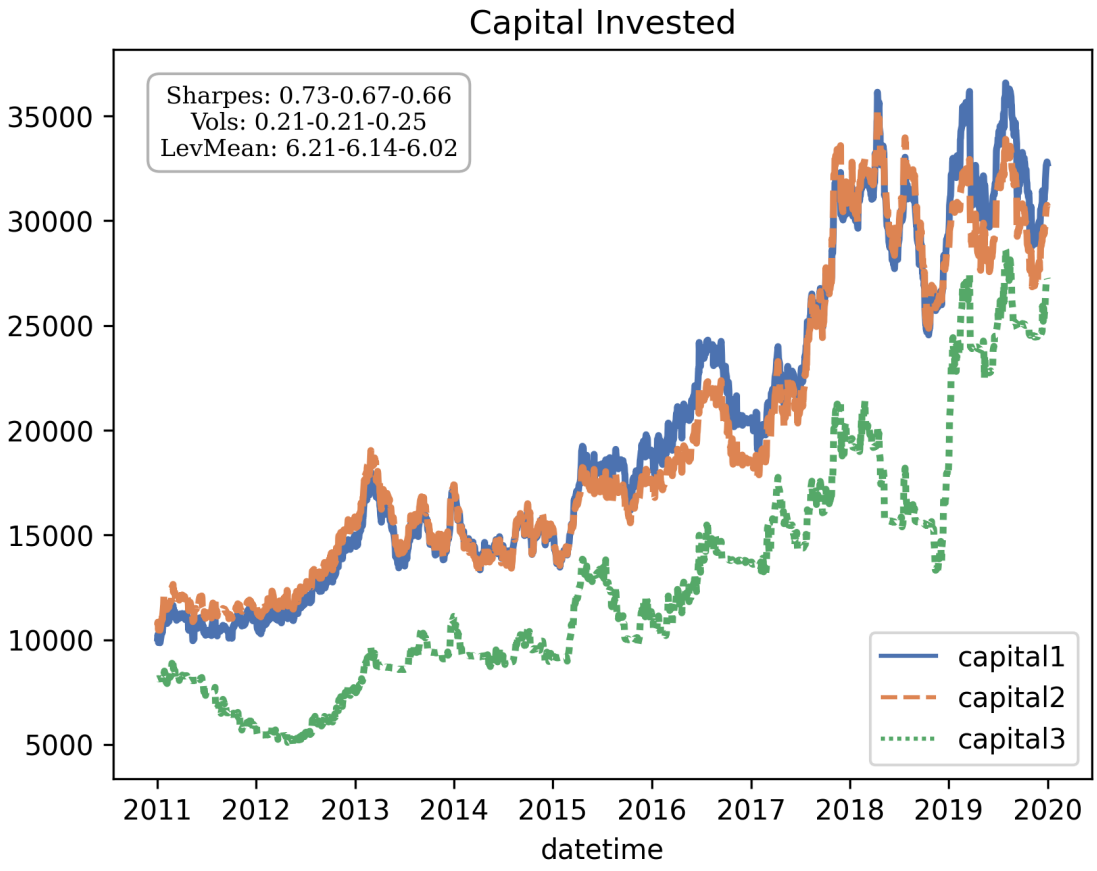

Figure 2: Example Run of Our 3 risk-variants, we see that the volatility targeting variants track each other fairly closely.

5.0.1 Volatility Tracking

Variant.Strategy	1.1	1.2	2.1	2.2
$\bar{\sigma}$	0.21	0.20	0.21	0.19
$\sigma(\sigma)$	0.001	0.001	0.003	0.004

Table 3: Summary of Realised vs Target Volatility (Trials: 13 * 4)

Both variants appear to track our volatility target closely, doing a good job of tracking our target risk. Let's consider its effect on portfolio returns.

5.0.2 Portfolio Effects

Variant.Strategy	1.1	2.1	3.1	1.2	2.2	3.2
$\bar{Sharpe}$	0.59	0.47	0.65	0.16	0.24	−0.10
$\bar{CAGR}$	0.11	0.08	0.14	0.01	0.03	−0.01
$\bar{Rank}$	1.92	2.84	1.23	1.92	1.08	3.0

Table 4: Summary of Portfolio Returns with Risk-Variants

We show mean Sharpe over the trials, CAGR of the strategy returns under different risk variants and terminal wealth ranks ($\sim$ CAGR). Here, we see some interesting results. In the first strategy (intraday momentum reversion), the leverage targeting approach produced the best CAGR and Sharpe, while EWMA vol-targeting performed comparatively poorly. In the second strategy (generic 12-M momentum), the EWMA variant performs best, with our leverage targeting variant performing the worst.

6 Some Musings

The result was interesting for a few reasons - a priori I had the opinion that leverage targeting would behave the worst out of the 3 in all cases, since it is a noisy factor. This was the case for the second strategy but not in our first experiment. Secondly, the auto-correlative exhibits were stronger for the first-moment in strategy 2, so in any case I expected that if leverage targeting was beneficial, it would be more beneficial for the second strategy than the first strategy. I was wrong on this. This needs a more extensive review - admittedly we took the ex post portfolio returns after the risk variant was already applied; it would be a cleaner study to first study raw, unleveraged portfolio return's statistical properties, followed by studying how the risk variants transform the distributions.

If I had to guess, in the first strategy we were trading an 'effect' that is short term mean-reversion. This effect may be market regimental. In the second strategy, we were trading the momentum 'effect' but the underlying alpha is less time-varying, such that we are actually trading 'assets'. Applying a fast varying risk technique to a slow moving alpha might have introduced significant noise relative to the underlying signal, harming the performance. Either way, it is difficult to inference any reasonable conclusions from such a small sample - but this is worth a further look in a more extensive, detailed paper studying thousands of alpha models. ***Perhaps we should not dismiss even the fast decaying first moment when risk-modelling for portfolio allocations.***

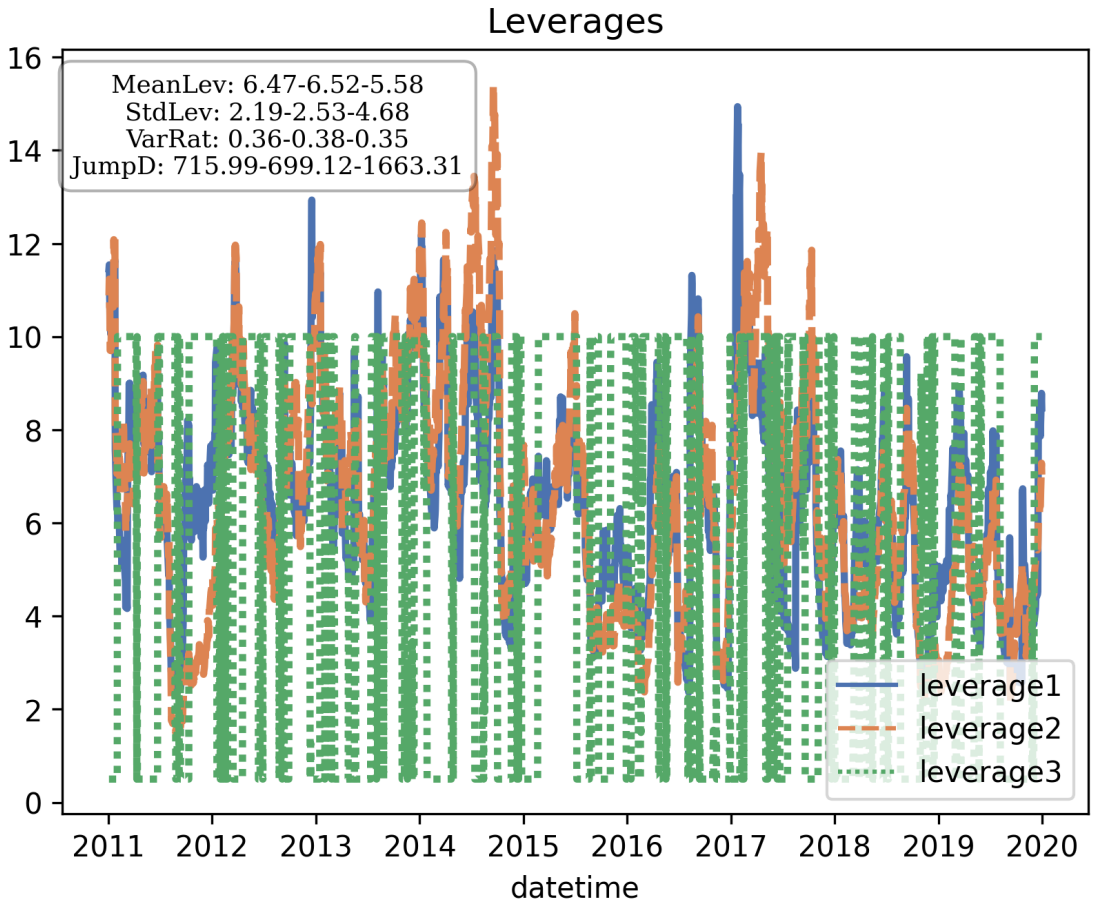

Figure 3: Example Portfolio Leverage Time Series.

7 Important Notes

7.1 Notes on Leverage

Variant.Strategy	1.1	2.1	3.1	1.2	2.2	3.2
$\bar{L}$	6.24	6.26	5.73	3.62	3.28	5.17
$\sigma(\bar{L})$	1.97	2.33	4.65	1.27	1.50	4.60
$\bar{\Sigma}\|L_t - L_{t-1}\|$	678	665	1566	336	292	1827

Table 5: Summary of Portfolio Leverage with Risk-Variants

The performance comparisons assume that the trades are friction-less. But trading costs are real. We can proxy the relative trading costs by seeing the absolute changes in leverage over time. In particular, if we are constantly levering up and down our whole market exposure, we are eating market costs. We see that our EWMA variant suffers the least cost penalty, with the leverage targeting suffering up to 6 times the approximate trading volume in nominal terms - this is expected due to the return factor itself being jumpy. This is a significant penalty to the leverage targeting method and favors our EWMA variant.

7.2 Notes on Volatility Decomposition

We can verify that the strategy scalar is the compensating scalar due to loss of risk/volatility from diversification (imperfect correlation) and neutralising market beta. We already did the hard work before. Running an alpha with *long_size* 1 and *short_size* 0 should allows us to run a trivial 'buy and hold' strategy with the risk-management of our choice.

```
alpha = Alpha(
    instruments=instruments[:1],
    dfs=ohlcvs,
    configs={
        "start" : trade_start,
        "end": trade_end,
        "longsize": 1,
        "shortsize": 0,
        "study": 1
    }
)
```

Running the simulation as above should give a strategy scalar around 1 since there are no diversification effects or cancelling of beta when a single asset is traded. The resulting *strat scalar* for single asset hovers around 1 for the entire trading period, with a mean of 0.93.

We then see the strat scalar when multiple assets of directionally neutral trades are taken. We see a consistently higher value, with mean of 3.55.

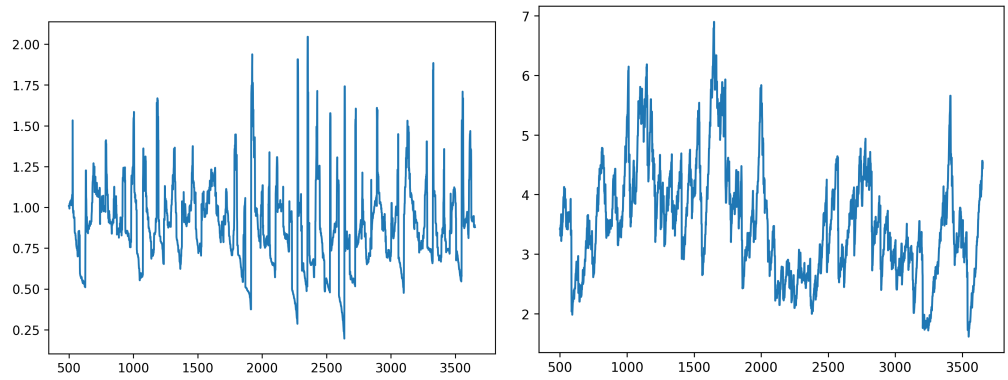

Figure 4: Strat Scalar with Single Asset (Left) and Multiple Assets (Right)

7.3 Notes on Time Taken

```
alpha = Alpha(
    instruments=instruments[:1],
    dfs=ohlcvs,
    configs={
        "start" : trade_start,
        "end": trade_end,
        "longsize": 1,
        "shortsize": 0,
        "study": 1 #vs 2
    }
)
```

We noted that the variant 2 (EWMA) is cheaper computationally than the variant 1. We run the configuration as above on both variants and compare time for execution. Variant 1 took **26.188** seconds to complete, while Variant 2 took **20.337** seconds to complete. While this may not seem significant yet, running this on only 6000 strategies would cost 100 minutes of difference in time execution!

7.4 Notes on Leveraged Wealth

We can also verify the math on our leverage, that $\frac{\mu}{\sigma^2}$ approximately maximises our geometric wealth (we only approximately display normality). With the same configurations on the single asset trial,

we compute

```
df = df.loc[df["capital ret"] != 0]
lev = df["capital ret"].mean() / df["capital ret"].std() **2
print(lev)
```

giving optimal leverage **3.74**. We apply different constant-leverage targeting and observe the difference in terminal wealth.

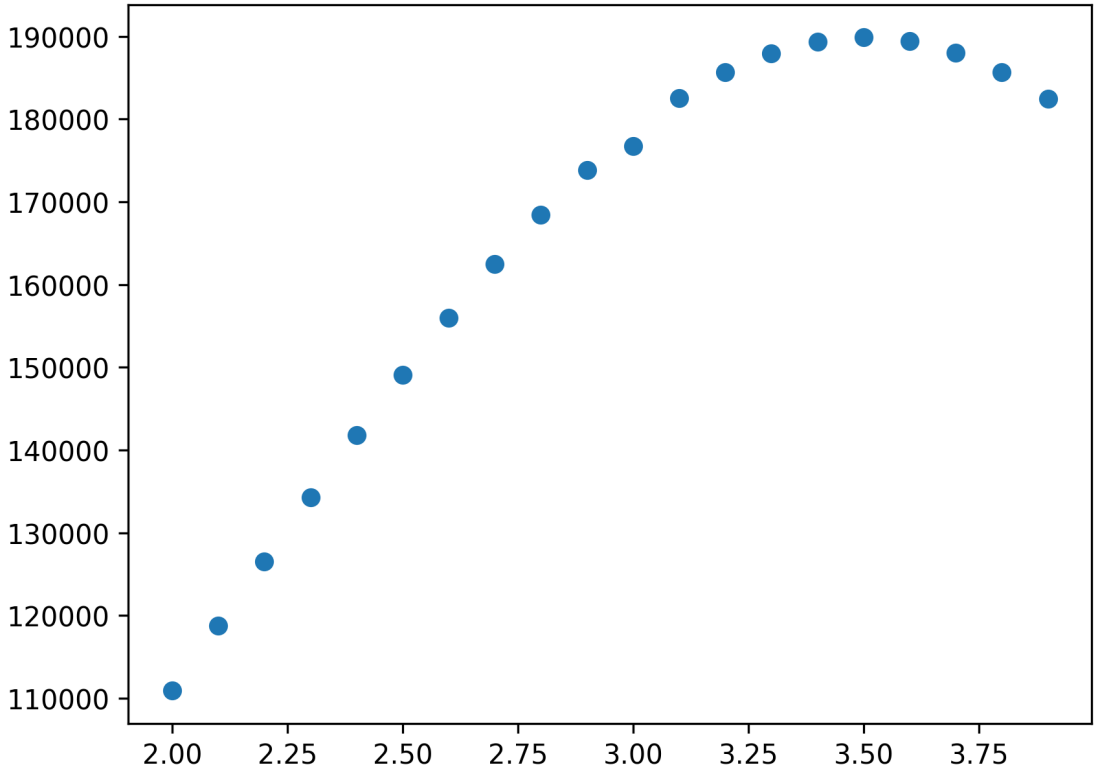

Figure 5: Terminal Wealth against Leverage.

The relationships between terminal wealth and leverage shows that our math and code implementation behaves as expected. Obviously, we have the benefit of hindsight to verify that this is the optimal leverage, but we would not be able to achieve this in practise (as we demonstrated) - *cues GIF of Captain Hindsight.

8 Note to Readers

We originally had another discussion in mind when writing the pieces of code demonstrated here - we intended to discuss the implementation of a robust, efficient testing engine. However, as we typed up the code, we realised there were some interesting points of discussion, and after making some interesting observations created a (interim) report.

While we did not explicitly discuss the *Alpha* class in detail, we intend to dive deeper into this testing engine in the weeks/months to come. In particular, we believe that this is the minimum requirement any systematic approach needs to address - the computation of edge, the relative allocations of edge and the overall allocations of edge. However, we can do much better. We want to decrease the latency between the birth of an hypothesis and evaluation of its merits by increasing the robustness of our quant tools.

We intend to discuss in great detail (for paid readers) how to make this fundamentally 'solid' vanilla implementation into a high performance quant tool that we can use to test any trading ideas within minutes and go from programmer to quant. We include discussions in code profiling, under the hood optimizations, data buses, Cython-ic and JIT compilers to achieve C-like perfomances in our Python application. We also want to include things like FX converters to allow us to trade multi-FX products. As we mentioned, this year, another of HangukQuant's objective is to go beyond alpha discussions to talk about creating your own quantitative systems and tools to take you to the next level in your quantitative journey. Look forward to seeing you there.