Pairs trading as univariate statistical arbitrage

Universidad Nacional de Educación a Distancia

Raúl Pareja Martín
Trabajo de Fin de Máster

rpareja32@alumno.uned.es

September 24, 2023

Abstract

We present a comparision of 3 different modelling approaches to univariate statistical arbitrage. Making use of the pairs selection framework of Sarmento & Horta (2020) we show how to improve the returns of a trading strategy based on those pairs, for a period of time from year 2000 to 2022 and divided by semesters. The first method considered is a non-parametric threshold rule over the cointegrated residual spread (-0.81%). We next move to use the parameters of a fitted Orstein-Uhlenbeck process (-0.1%) and finish with an ARMA(1,1) (0.81%).

INDEX

1 Introduction

Pairs trading is a strategy consisted in choosing a combination of securities that showed some type of predictable relationship, and trading them long-short when this relationship temporally breaks, expecting it to recover.

Traditional approaches for pairs selection used distance (Gatev et al. (2006)) and correlation (Chen et al. (2012)), but these proved to be suboptimal (Krauss, 2015). Distance metrics for pairs selection do not account enough for the variance of the series, as a stable distance of 0 will show up high in our ranked pairs but clearly it will not be tradable. And a perfect correlation can give spurious results, as two perfect correlated stocks but one with returns of double magnitude than the other (maybe due to different degrees of financial leverage) will present no mean reversion whatsoever. Those approaches have then been replaced with a variety of methods for pairs (or portfolio in the multivariate case) selection, signal extraction and weight allocation during the last 15 years. Before presenting our approach, we will define what statistical arbitrage is, and distinct between 3 broad categories of it.
Marco Avellaneda defined statistical arbitrage with 3 pillars: "(i) trading signals are systematic, or rules-based, as opposed to driven by fundamentals, (ii) the trading book is market-neutral, in the sense that it has zero beta with the market, and (iii) the mechanism for generating excess returns is statistical." Therefore, for a pairs trading strategy to be considered statistical arbitrage it must define systematic trading rules based on a statistical valuation method, while maintaining a market-hedged allocation. We can divide statistical arbitrage in:

1. Univariate: using the classic approach of pairs trading where we trade a security vs. another to construct a portfolio of pairs.

2. Quasi-multivariate: trading a portfolio of securities vs. a particular one. This less crowded approach enhances efficiency in transaction costs by netting positions in the shared security across the portfolio.

3. Multivariate: build a sufficiently sparse (to constraint transaction costs) and volatile (to have enough trading opportunities) mean-reverting portfolio of assets. The optimization routines and machine learning methods developed in this context are usually easy to transform from a mean-reversion objective to momentum.

Quasi-multivariate and fully multivariate approaches have empirical evidence of higher returns. Further, in the last years it is not clear if univariate statistical arbitrage is still profitable in developed markets[1]. Do & Faff (2010) conclude that the Gatev et al. (2006) distance method coupled with long-short thresholds to the cointegrated residual spread experienced a decay in monthly returns from 1% to 0.6%, between both publication dates. More recently, Fil (2020) reports just 0.11% of monthly returns from 1990. He also tests the cointegration rule for pairs selection giving 0.18%. Our results should be even lower than the latter as our dataset starts from year 2000.

1.1 Data description

We use a 6-month formation (or in-sample) period, in which we select the pairs that we think will perform better in the also 6-month trading (out-of-sample) period. We stick with 6-month trading periods, but it could be 3-month, or any other combination[2]. The back-tests then proceeds in a walk-forward manner, i.e. forming pairs in semester i and trading them in i+1, forming new pairs in i+1 and trading them in i+2, and so on.

Our dataset comprises 25 industry sectors ETFs and 3880 stock tickers with at least 100 million dollars of market capitalization[3] from the United States between january 2000 and july 2023 (47 semesters). These are active companies at the date of writing this work[4], and therefore the number of stocks in the first years is much smaller. For example, in the first semester we have 1421 securities, in the 10th semester 1674, 2012 securities for the 20th, 2435 for the 30th and 3062 in the 40th.

We download them from `https://stockanalysis.com/stocks/`, and get open prices from yahoo finance. We choose open prices as they are tradable prices, to the contrary of closing prices. We do not model slippage costs to therefore execute orders at the exact open price, furthermore, our dataset only shows the midprice. As just the spread can go past 200 basis points for the most illiquid securities, we should estimate high

[1]There is evidence on the current profitability of pairs trading in emerging markets. In Balladares et al. (2021) are reported consistent returns of over 15% for some emerging markets as Israel, Greece or Brazil.

[2]We have also tested 3-month and 1-year formation periods along with 3-month and 1-year trading periods respectively, but got worse results, notably worse in the latter case.

[3]In Jeppesen (2020) it is found that pairs trading is net long illiquid stocks and in Hossein et al. (2016) is reported that the liquidity factor is inversely correlated to each pair's return. In those papers is concluded that returns are related to the liquidity provision from trading the pairs in times of stress.

[4]Thus we have survivorship bias, however, as our trading horizon is just 6 months and we do not base our trading strategy in any fundamental value metric we can obviate this fact.

transaction costs. These costs are usually set around 30-50 basis points, so we will report returns for 50 bp and 100 bp, the latter being a better estimation.

2 Pairs selection

We select pairs following the proposed framework in Sarmento & Horta (2020)[5], which is based in the clustering algorithm OPTICS (Ordering Points to Identify the Clustering Stucture) to reduce the potential pairs and a series of statistical tests to find the tradable ones.

2.1 OPTICS

The OPTICS algorithm works finding clusters of similar securities within the Principal Component Analysis (PCA) space. Therefore, we start by using PCA to get a set of k orthogonal factors to explain each return times series. PCA is also an unsupervised algorithm, that provides a low-rank matrix approximation over standardized data via eigenvalue decomposition or singular value decomposition (SVD). We show the latter as it is the one we use with the sklearn package. Our implementation follow these steps:

1. Standardize data, using the well-known formula:

$$z_i = \frac{x_i - \mu}{\sigma} \tag{1}$$

where x_i are returns, μ the mean return and σ the volatility of the series.

2. Compute covariance matrix (we can also use the correlation):

$$\Sigma = \frac{1}{n-1}\sum_{i=1}^{n}(z_i - \bar{z})(z_i - \bar{z})^T \tag{2}$$

where n is the number of samples, z_i the standardized returns, $\bar{z}$ the mean (0) and Σ the resulting covariance matrix.

3. SVD decomposition of the covariance matrix:

$$\Sigma = USV^T \tag{3}$$

where U is the left singular vectors matrix, S is the diagonal matrix of singular values s_i and V columns are called right singular vectors.

4. The k right singular vectors denote our k factors, or principal components, that can be used to project our data:

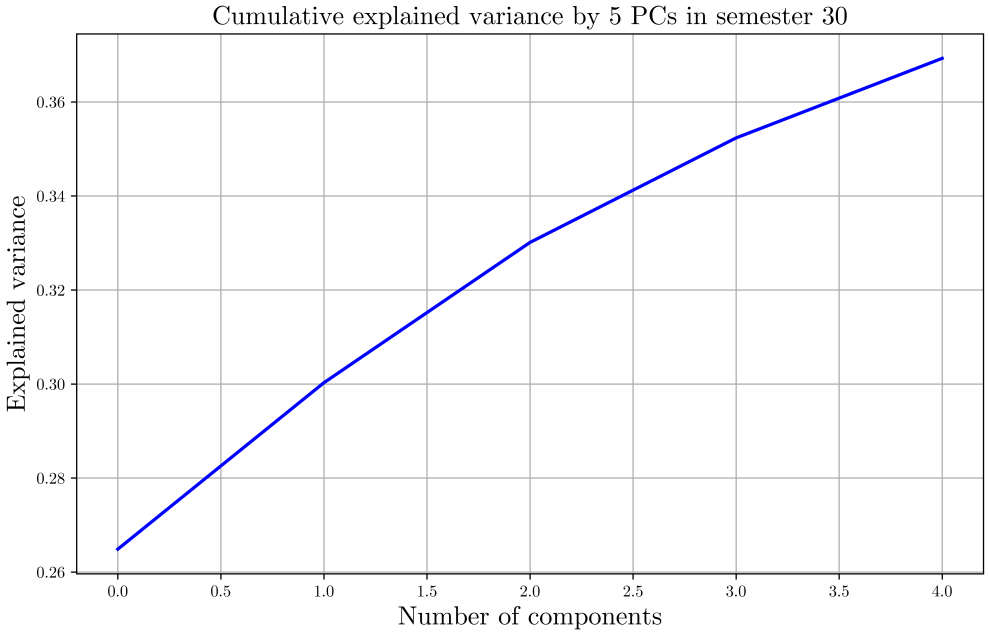
$$X_{\text{projected}} = X \cdot V_k \tag{4}$$

where $X_{\text{projected}}$ represents how much the principal component k contributes to each data point. In our case the first principal component is known as the market component, because it has a positive sign in each element of the first singular vector.

We choose $k = 5$ factors to capture the most significant sources of variation in the data[6], which explain 39% of the total variance in our dataset at semester 30. This quantity varies with the economic cycle, as during crises the needed number of factors to explain this same percentage will notably increase (Avellaneda & Lee (2009)). After that, OPTICS looks for similarity within the PCA factors that explain the returns.

[5]Implementing this framework is straightforward as we can adapt the implementation from the original paper `https://github.com/simaomsarmento/PairsTrading`, or the more accessible version from `https://github.com/streater512/MLforFinance`. The latter just needed debugging work, as some pairs triggered errors in different parts of the code. The 3905 stock tickers are counted after this data cleaning.

[6]The framework is robust to the number of chosen factors, moving k to 7 and 10 barely changes the resulting tradable pairs. This is because PCA is used just to signal potential pairs by OPTICS, the subsequent statistical tests are the real burden.



OPTICS is closely related to DBSCAN (Density-Based Clustering of Applications with Noise), this other algorithm requires 2 parameters: ϵ and $MinPts$, which correspond to the density of the cluster and the minimum of points a cluster must contain. Let us define some common concepts to both algorithms:

$\epsilon - neighborhood$. The set of data points within a distance ϵ from point q.

$$N(q, \epsilon) = \{p \in D \mid distance(p, q) \leq \epsilon\} \tag{5}$$

Here, $N(q, \epsilon)$ is the $\epsilon - neighborhood$ of point q, D is the dataset, $distance(p, q)$ is the distance metric used to measure the distance between points p and q and ϵ is the maximum distance threshold (or density).

$Core - point$. A point q is considered a $core - point$ if there are at least $MinPts$ data points (including itself) within its $\epsilon - neighborhood$, i.e.:

$$|N(p, \epsilon)| \geq MinPts \tag{6}$$

where $N(p, \epsilon)$ is the $\epsilon - neighborhood$ of point p and $MinPts$ is the minimum number of points required to form a dense region.

DBSCAN starts by looking at the core points in the $\epsilon - neighborhood$ of every point and identifying the core points with more than $MinPts$ neighbours, then finds the density matching points and assign each non-core point to a nearby cluster if they are $\epsilon - neighborhood$, otherwise discards it as noise. The problem with this algorithm is that it clusters the data explicitly. This makes the results very sensitive to the election of the parameters, as it will found clusters of just a particular density ϵ, while neglecting other potential clusters of different densities.

On the other hand, OPTICS extracts the clustering structure of the dataset by previously identifying density-connected points. To do this, it makes use of the $core - distance$ and $reachability - distance$, defined as:

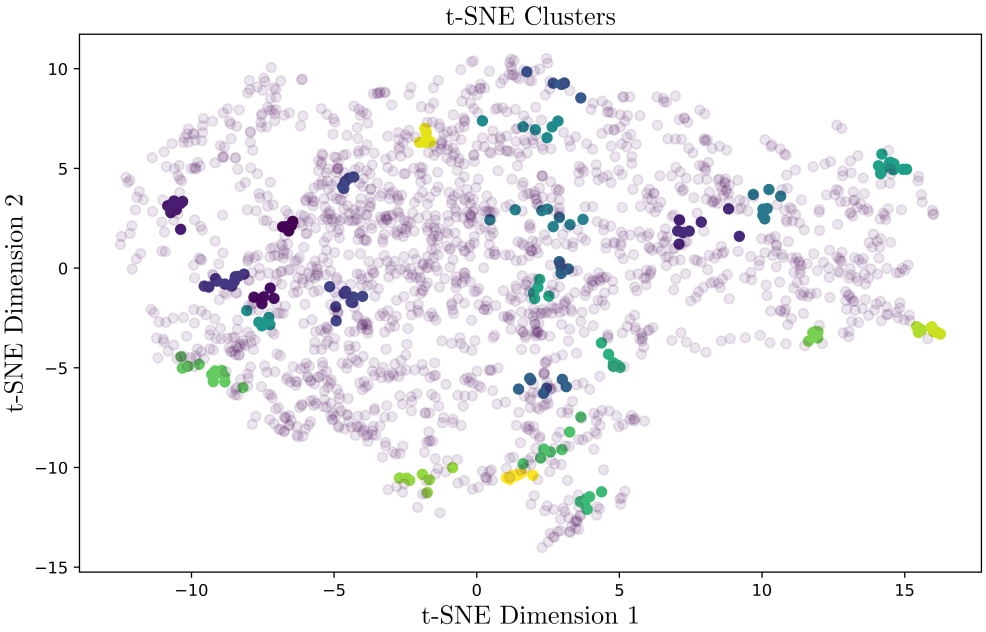
$$core - distance_{\epsilon, MinPts}(p) = \begin{cases} MinPts - distance(p), & \text{if } p \text{ is a } core - point \\ \text{undefined}, & \text{otherwise} \end{cases} \tag{7}$$

where $MinPts - distance(p)$ is the distance from a point p to its $MinPts$ neighbor, and therefore, the $core - distance$ is the smallest distance between a core point and a point in its $\epsilon - neighborhood$.

$$reachability - distance_{\epsilon,MinPts}(p,q) = \begin{cases} \max(core - distance(q), distance(p,q)), & \text{if } p \text{ is a } core - point \\ \text{undefined}, & \text{otherwise} \end{cases} \tag{8}$$

the $reachability - distance(p,q)$ can be interpreted such that p is distance-reachable from q. By computing the $core - distance$ of each point and forming a queue of ordered points based on the $reachability - distance$, it is able to create a reachability plot that allows OPTICS to group points close to each other.

Using t-SNE we can get a representation of the OPTICS clusters in 3 dimensions. These clusters usually approximate industry sectors, but with better trading results as the original paper points out.



The result is that the potential pairs are the sum of all the possible combinations between the members of each cluster. Consider semester 30 as an example, before appling OPTICS with 2435 securities we had $C(n,k) = \frac{2435!}{2!\cdot(2435-2)!} = 2963395$ potential pairs. OPTICS reduces this to 1942 potential pairs, a number that is viable in computation time for the statistical tests.

2.2 Statistical filtering

The following 4 consecutive tests will filter the potential pairs into our finally tradable pairs. Only the pairs that overcome a test will move onto the next.

1. Engle-Granger 2-step test p-value < 0.05. In the first step it finds the possibly cointegrated residual:

$$Y_{0,t} = \alpha + \beta \cdot Y_{1,t} + X_t \tag{9}$$

where $Y_{0,t}$ and $Y_{1,t}$ are the time series of each security at time t, spanning 6 months. α, β and ϵ are the intercept, slope and residual, respectively. And in the second step it performs an Augmented-Dickey-Fuller test for stationarity to the residuals, where the null hypothesis is that there is an unit root present i.e. the residuals are not a stationary process. Rejecting this null with at least a 95% percent accuracy leads us to assume that the spread series are cointegrated, therefore we can explain the series linearly with more accuracy.

$$\Delta X_t = \alpha + \rho \cdot X_{t-1} + \sum_{i=1}^{p-1} \phi_i \cdot \Delta X_{t-i} + \epsilon_t \tag{10}$$

Where ΔX_t represents the regressed differenced residuals, ρ is the coefficient of the lagged residual X_{t-1}, ϕ_i are coefficients for lagged differences that we use in the test statistic and p is the lag order. We get two p-values for each pair from using a different security as the independent variable, we save the lowest and correct the ordering of the pair accordingly.

2. Hurst exponent < 0.5. The Hurst exponent is a measure of the long-term memory of the series. Hurst of less than 0.5 shows anti-persistent behavior, which indicates negative autocorrelation and mean-reversion. The contrary is true for a Hurst > 0.5.

$$\mathbb{E}\left[\frac{R_n}{S_n}\right] = Cn^H, \quad n \to \infty \tag{11}$$

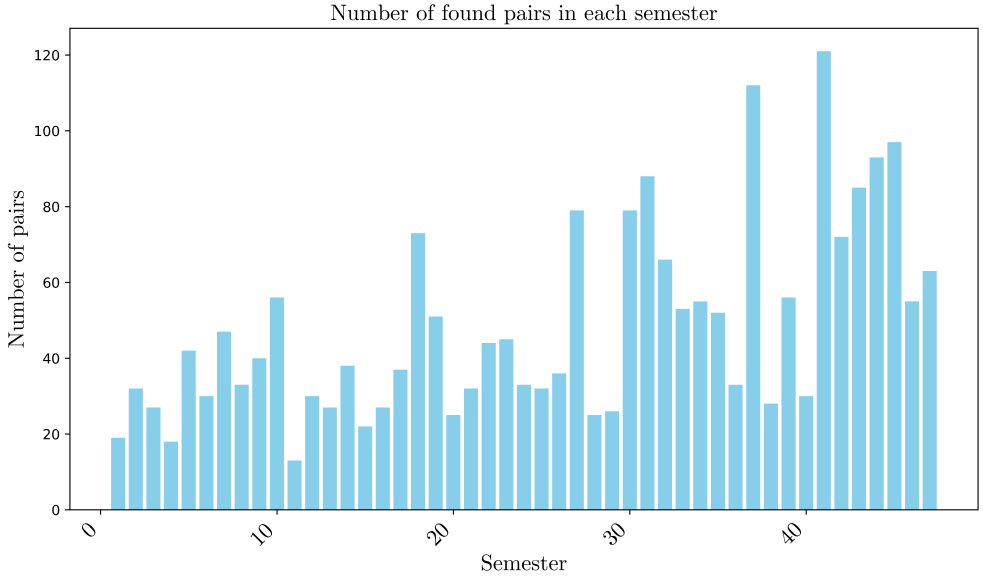
Where R_n is the range of the first n cumulative deviations from the mean, R_n is the series of the first n standard deviations, n is the number of observations and C is a constant.

3. $1 <$ Half-life < 42. The half-life refers to the time it takes for a series to revert halfway back to its mean, we set the maximum threshold value to 1/5 the number of trading days in our formation/trading period and the minimum to a day. The results are usually between 2 and 5 days.

$$\text{Half-Life} = -\frac{\ln(2)}{\hat{\beta}_1} \tag{12}$$

Where $\hat{\beta}_1$ is the slope coefficient of the lagged value of the series in a regression against the first differences of it.

4. Number of mean-crossings > 12. The series crosses its mean value at least 12 times. This test try to ensure that we have enough trading opportunities. This value usually moves from 30 to 60.
The number of resulting found pairs from this process can be seen in the image below. We will simply apply equal weighting to the portfolio as we cannot superficially infer any predictability from any of the statistics. Risk-contribution or other types of in-sample portfolio optimization are possible but we do not implement them here.



3 Trading strategy

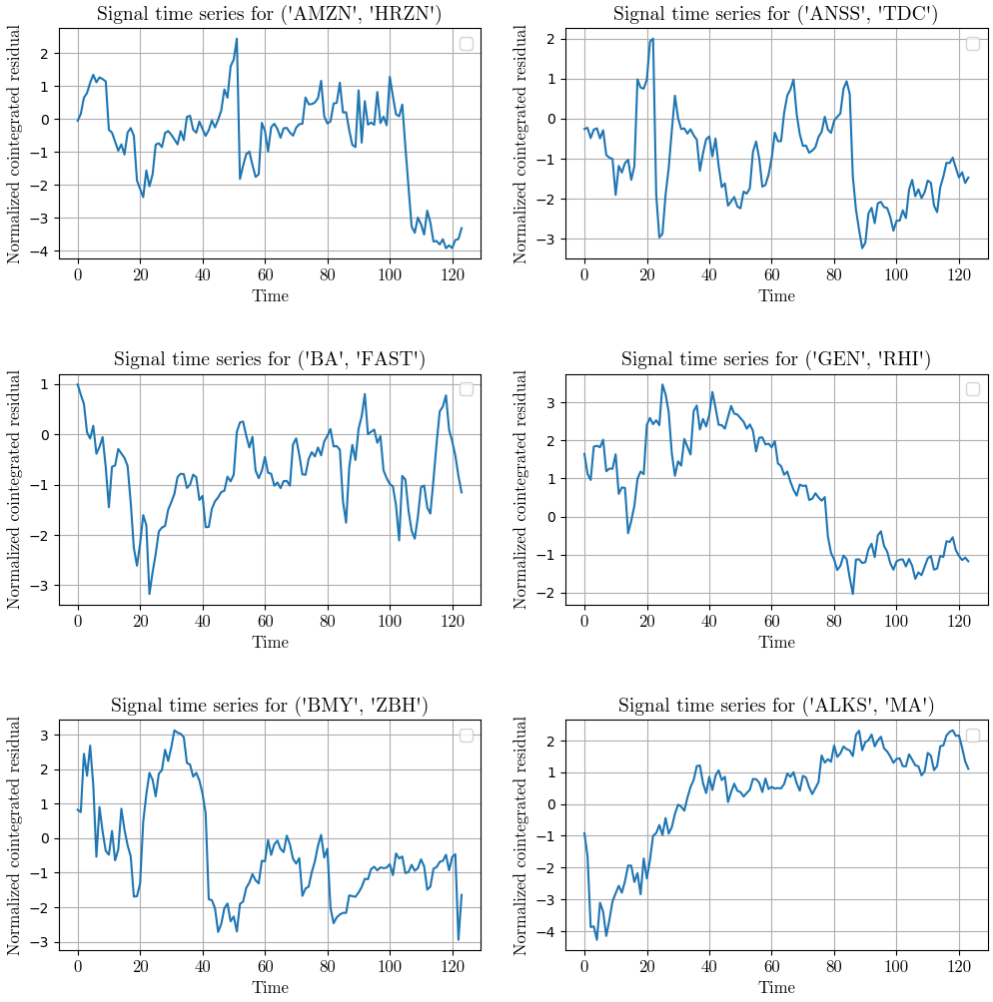
Now we are going to choose how to trade the pairs. For this we will construct a model that will signal us when has the spread surpassed a significant threshold and we should enter a trade. We will test different model setups.

3.1 Standardization of the cointegrated residuals

Our naive approach is defined by signals from the standardization of the cointegrated residual spread X_t in (9) (we will from now refer to it as just 'residual'). This residual comes from a regression of two cointegrated stock price series, leading us to assume the residual as an ideosyncratic, noise component that mean-reverts. At each day of our trading period we will look back the number of trading days in a semester[7], estimate a linear regression model (following the imposed order of variables by the Engle-Granger test's p-value) and save the intercept and slope coefficients. We next use these values to construct the residual series:

$$X_t = Y_{0,t} - (\alpha_t + \beta_t \cdot Y^T_{1,t}), \qquad \text{for each } t \text{ in } (0, T) \tag{13}$$

where t spans one semester and $(0, T)$ the current trading period. Finally we standardize the last (most recent) residual value according to (1). We signal a -1 (short spread) when the model feeds us this value over 2 standard deviations further from the mean and a +1 (long) when it shows the value under -2. We close positions when the residual crosses 0. In the following plots we can see the evolution of this signal for our first 6 pairs in the semester 30, which corresponds to the first half of 2014. It looks good enough to trade it.



We can see a caveat of our back-test here, we close every position on the last day of the trading period although we can be sure that it will revert in the future[8]. This can be easily solved in practice by not closing

[7] In each semester we have 123-126 data points. We also test look-back periods of 1/3 and 2/3 of this number with slightly worse results, it may be beneficial to even increase this estimation period.

[8] I have not found a pair that did not revert to mean after some time.

the position, but implementing this will mess the uniformed periodization of returns. Another detail is that we have repeated securities in different pairs for some semesters, which can have a positive and negative impact in the returns: positive from being able to net positions in that security as in a quasi-multivariate context, but negative in case that the positions accumulate in one leg of the trade with the consequent effect on market impact.

In the next step we have to differentiate between a beta-neutral or dollar-neutral approach to position sizing for market-neutrality. For dollar neutrality we simply open short and long positions of equal dollar amounts, and therefore we transform this signal into a position equal to it for the security used as the independent variable in the linear regression, and a position equal to its negative value for the dependent variable. This way the strategy is almost (due to transactions costs) self-financed, which makes it returns fundamentally different as the capital needed to run the strategy will be much less than the assumed 1 monetary unit. To adapt the beta-weighting to our supposed 1 unit capital we do:

For day in **range**(0, trading days):
  If absolute value(beta) <= 1:
    position_X[i] = signal_X[i] * beta
    position_Y[i] = signal_Y[i]
  else:
    position_X[i] = signal_X[i]
    position_Y[i] = signal_Y[i] / beta

This way we are effectively cutting the allocation to one leg of our trade expecting a more uncorrelated return with respect to the market. However, this is not clear Two Sigma, year. We simulate a beta rebalancing each 5 trading days. And with the dollar-neutral we do not rebalance, simply accumulating the returns after opening the long-short in equal amounts.

The pair strategy log returns are next computed at each leg of the trade as the log return at day i by the position at day i-1, minus the first difference of the position at day i multiplied by the transaction costs[9]. Then we take the cumulative sum, transform to raw returns and add each leg's returns. This is a simple vectorized back-testing algorithm, not a proper event-driven one. We show it in APPENDIX A1. The rest of the code is uploaded at `https://github.com/rauup0/Univariate-Statistical-Arbitrage`.

We compute volatility from daily returns following:

$$\text{Portfolio volatility} = \sqrt{w^T \cdot \text{cov}(R) \cdot w} \tag{14}$$

where w is a column vector of portfolio weights and R a matrix containing the daily returns of each pair. We annualize this number (without accounting for the autocorrelation of the series[10]) using $\sqrt{252}$. Annualizing the 6-month return with compounding and taking in account the 3-month t-bill annualized rate we can then compute the sharpe ratios.

Performance Metric	**0 bp ts. costs**	**50 bp ts. costs**	**100 bp ts. costs**
Mean 6-month return (B-N)	3.19%	-0.32%	-3.74%
Daily return volatility (B-N)	0.38%	0.38%	0.38%
Annualized excess sharpe ratio (B-N)	0.72	-0.63	-1.88
Mean 6-month return (D-N)	6.5%	2.31%	-1.75%
Daily return volatility (D-N)	0.43%	0.43%	0.44%
Annualized excess sharpe ratio (D-N)	1.6	0.17	-1.14

Table 1: Performance metrics for beta-neutral (B-N) and dollar-neutral (D-N) allocation

Above we can see the performance of dollar and beta-neutral strategies with 0, 50 and 100 bp of commisions each. Neither of the more realistic strategies with 100 bp of commisions per trade have a positive mean 6-month return. In the literature is common to see an optimization step to find the most appropriate

[9]This methodology follows `https://www.oreilly.com/library/view/python-for-algorithmic/9781492053347/ch04.html`.

[10]In Sarmento & Horta (2020) this correction to the sharpe ratio is <0.01 for every model considered. There are cases when this correction is much more useful than in our statistical arbitrage setup.

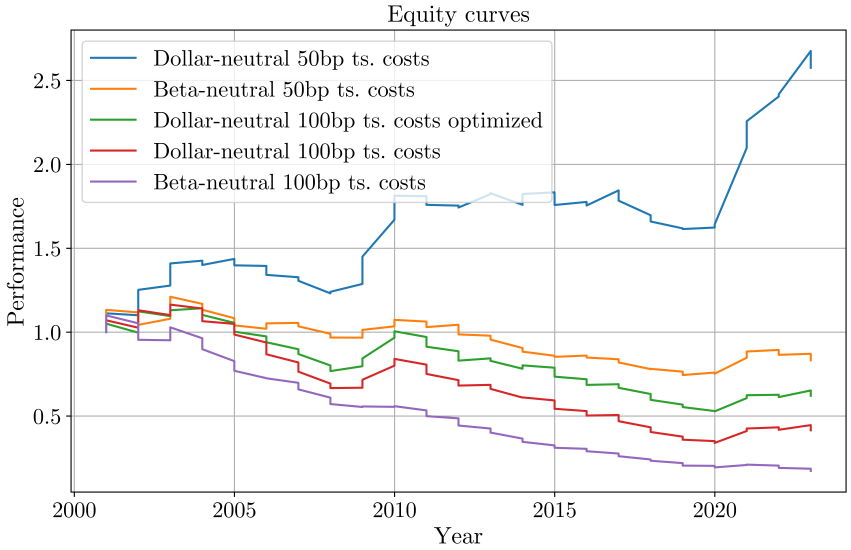
thresholds, therefore, we perform an in-sample grid search over +/-[1.7, 1.9, 2.1, 2.3, 2.5] thresholds to open positions, and +/-[0.4, 0.2, 0] to close them. We perform it to the dollar-neutral strategy with 100 bp of transaction costs. The +/-[2.5] and 0 thresholds produce the higher in-sample returns, we show the (out-of-sample) results below along the equity curves.

Performance Metric	**0 bp ts. costs**	**50 bp ts. costs**	**100 bp ts. costs**
Mean 6-month return (B-N)	2.13%	-0.29%	-2.69%
Daily return volatility (B-N)	0.33%	0.33%	0.33%
Annualized excess sharpe ratio (B-N)	0.49	-0.6	-1.88
Mean 6-month return (D-N)	4.65%	1.84%	-0.91%
Daily return volatility (D-N)	0.37%	0.38%	0.38%
Annualized excess sharpe ratio (D-N)	1.2	0.08	-0.95

Table 2: Performance metrics for B-N and D-N allocation with optimized thresholds for D-N 100 bp

The optimization reduces the loss for the target strategy setup from -1.75% to -0.91%, a +84 bp benefit that is not enough to make a positive return. Furthermore, the optimization lowers the return for the 50 bp of transaction costs variant in almost -0.5%, which means that the estimation of the transaction costs is very important if we want to optimize thresholds. However, it seems that the beta allocation is more robust to this change, the 50 bp ts. costs setup stays roughly the same (-0.03%) and the 100 bp increases in +1.05%.

Focusing on the differences between beta and dollar-neutralized allocation, the former looks to have a smoothing effect over the time series. But the volatility drop cannot compensate the drop in returns, leading to worse sharpe ratios.



3.2 Modeling the residual as an Ornstein-Uhlenbeck process

In the previous section we created a mean-reverting spread by regressing one stock over another and saving the residuals. We were applying a relative-value trading rule to the residual after factoring out the variance explained by the paired stock. The signal came from taking the mean value and variance of the residual series within the estimation window and standardizing the last residual spread value. Now instead, we will model the cumulative residual itself and use the mean and variance given by the coefficients of the new model to standardize the spread value at each time step and construct our signal.
Let us denote $X_t, ..., X_T$ the residual spread process in (13), and Y_0, Y_1 the stock returns for the independent

and dependent variables, respectively. Note that in the previous section we used the same notation but referred to prices instead of returns [11]. We can express (13) in continuous differential form as:

$$\frac{dY_{0,t}}{Y_{0,t}} = \alpha dt + \beta \frac{dY_{1,t}}{Y_{1,t}} + dX_t \tag{15}$$

taking the transformation $X_t = \sum_{t=1}^{T} X_t$ and assuming gaussian noise (so we cannot capture the fat-tail behavior of financial times series) we can write the cumulative residual dX_t as an Ornstein-Uhlenbeck process:

$$dX_t = \kappa(\mu - X_t)dt + \sigma dW_t \tag{16}$$

where κ is known as the speed of mean-reversion of the series, μ is the long-term mean value and σ its volatility. X_t is the observed process at time t and dW_t a Wiener process In discrete form this model is described by an AR(1) with gaussian noise:

$$X_t = a + bX_{t-1} + \epsilon_t \tag{17}$$

We fit an AR(1) at each day of our trading period, i.e. regressing the lagged cumulative residual value on the current one, using the same estimation window as before —6 months—. We next save the intercept and slope coefficients and compute:

$$\kappa = -\ln(b) \cdot \text{estimation window length} \tag{18}$$

$$\mu = \frac{a}{1-b} \tag{19}$$

$$\sigma_{X_t} = \sqrt{\frac{\sigma_\epsilon^2}{1-b^2}} \tag{20}$$

where σ_ϵ^2 is the variance of the gaussian noise ϵ. We next plug these values into the formula for the s-score:

$$\text{s score} = \frac{-\mu}{\sigma_{X_t}} - \frac{a}{\kappa \cdot \sigma_{X_t}} \tag{21}$$

where $\frac{a}{\kappa \cdot \sigma_{X_t}}$ is a correction term for the effect of the intercept in the model, that imposes a slope in our thresholds around the s-scores. These thresholds are now normalized and we trade them long-short at -2/+2. The back-test code is displayed in APPENDIX A2. This is the approach showed in Avellaneda & Lee (2009) but they did it in a quasi-multivariate setup using PCA as a factor model. They selected an ETF and a set of tradable stocks against it, they performed PCA to the return space and, by projecting this set of factors, they isolated a residual process from the shared factors within the group. With this signaled tradable residual they applied the modelling above, and archived a 0.9 sharpe ratio from 2003 to 2007 (and 1.1 during 1998-2003). The strategy is still not able to generate a positive mean return, but there is an improvement with respect to the previous trading strategy: +1.1% for the 50 bp and +3.47% for the 100 bp of transaction costs in the case of the B-N allocation, -1.04% for the 50 bp and +1.65% for the 100 bp of transaction costs in the case of D-N. The improvement is not robust with respect to every strategy assumption, but in general is better.

Next we perform an in-sample optimization over +/-[1.8, 2, 2.2, 2.4, 2.6] thresholds to open positions and +/-[0.1, 0] to close them. This yields -1.8 as the optimal threshold to open a long spread position and 2.2 to open the short spread position, along with the 0 threshold to close them. Again, we only perform this grid search for the dollar-neutral allocation with 100 bp of transaction costs.

We get the first positive mean returns for the strategies that assume 100 bp of transaction costs, increasing in 0.76% and 0.77% for beta- and dollar-neutral versions. The strategies with 50 bp of transaction costs

[11] In the previous section we were indifferent (we tried both returns and prices and get the same results) to this election, but now stick to returns to follow Avellaneda & Lee (2010).

Performance Metric	50 bp ts. costs	100 bp ts. costs
Mean 6-month return (B-N)	0.79%	-0.27%
Daily return volatility (B-N)	0.2%	0.2%
Annualized excess sharpe ratio (B-N)	-0.19	-1.02
Mean 6-month return (D-N)	1.27%	-0.1%
Daily return volatility (D-N)	0.23%	0.23%
Annualized excess sharpe ratio (D-N)	-0.08	-0.96

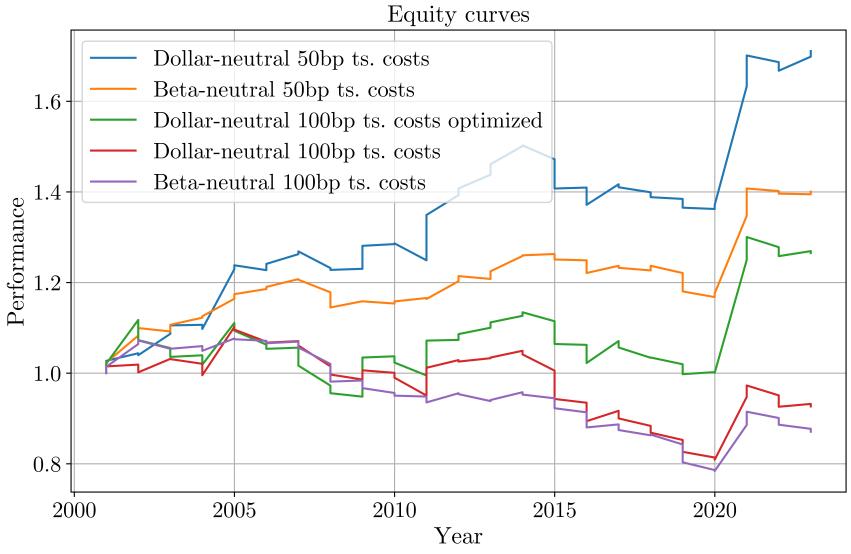
Table 3: Performance metrics for beta-neutral (B-N) and dollar-neutral (D-N) allocation

Performance Metric	50 bp ts. costs	100 bp ts. costs
Mean 6-month return (B-N)	1.76%	0.49%
Daily return volatility (B-N)	0.22%	0.23%
Annualized excess sharpe ratio (B-N)	0.17	-0.68
Mean 6-month return (D-N)	2.06%	0.63%
Daily return volatility (D-N)	0.24%	0.25%
Annualized excess sharpe ratio (D-N)	0.29	-0.59

Table 4: Performance metrics for B-N and D-N allocations using optimized thresholds for D-N 100 bp

improve 0.97 and 0.79 even more than the targeted commisions level in the optimization. This is probably due to lack of efficacy of the not optimized thresholds, the 50 bp strategies could almost surely be further improved more if they were targeted in the optimization.

In the equity curves below we can see a decent behavior by the 50 bp strategies, archiving near-profitability out of crisis times. The optimized 100 bp D-N strategy is only profitable due to the quick 2020 coronavirus crash.



3.3 Fitting an ARMA(p, q)

Now instead of fitting an AR(1) to the cumulative residual spread and compute the Ornstein-Uhlenbeck parameters to get our trading signals, we will fit an ARMA(p, q) and generate a signal when the predicted cumulative residual reaches a certain threshold. We will do this by, at the first day of the trading period, look back a semester and compute every possible ARMA(p, q) combination for p and q < 9, i.e.:

$$X_t = \mu + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \ldots + \theta_q \varepsilon_{t-q} + \varepsilon_t \tag{22}$$

where X_t is the cumulative residual series, μ is the intercept, that corresponds to the mean of a stationary process. $\phi_1, \phi_2, \ldots, \phi_p$ are the autoregressive (AR) coefficients for p < 9, and $\theta_1, \theta_2, \ldots, \theta_q$ are the moving average (MA) coefficients for q < 9. We next extract the Akaike Information Criterion, defined by:

$$AIC = 2k - 2\ln(L) \tag{23}$$

where k is the number of estimated parameters and L the maximum likelihood of the model. And next select the (p, q) order that minimizes this criterion, sticking to it for the rest of the trading period. This is computationally expensive, specially given our big dataset [12]. We signal a +1 (long spread) when the predicted cumulative residual crosses under the 10th percentile of the lookback window, and a -1 (short spread) when it passes the 90th percentile. This procedure gives a -0.07% for the D-N approach with 100 bp of transaction costs, which is practically equal to the non-optimized result in the previous section. Additional in-sample optimization work for the percentile thresholds should be considered.

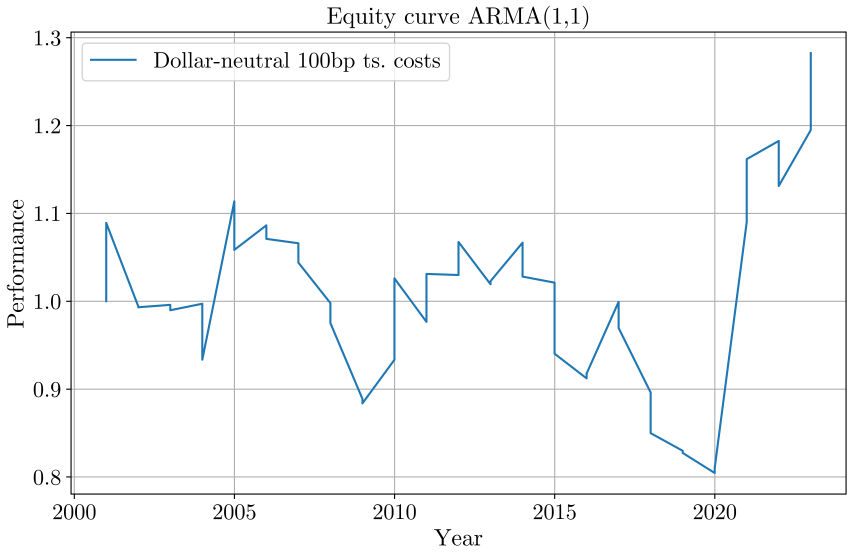
Simply fitting an ARMA(1,1) to the same 100 bp D-N assumption we get even better results, in fact the best result up to this point.

Performance Metric	**100 bp ts. costs**
Mean 6-month return (D-N)	0.81%
Daily return volatility (D-N)	0.48%
Annualized excess sharpe ratio (D-N)	-0.25

Table 5: Performance metrics for D-N allocation and 100 bp of ts. costs

This is an improvement of +0.18% over the optimized Ornstein-Uhlenbeck, and without doing any optimization. This shows that the AIC is probably not a good tool to select the best model in our context.

Paying attention to the returns of each individual pair, it seem to show that the losses are the same that in previous approaches, maybe even bigger, but the winners look to be much more exploited. The equity curve does not look attractive, all the gains are from the coronavirus crash.



[12] We had to cut the length of the lasts 10 semesters to 30 pairs to get the results in time. The execution of the code until that point lasted more than a day, and took another 10 hours to finish. Furthermore, some ARMA orders triggered CovergenceWarnings that additionally enlarged the execution time. We simply removed those model orders as candidates for trading, and if the error triggered in the trading period (something that happened a couple of times) we repeated the previous signaled residual.

4 From univariate to multivariate statistical arbitrage

We showed our trading models only applied to the univariate case. As previously said, this approach is reported by the literature to underperform quasi- and multivariate approaches (Krauss, 2015). To adapt the pairs selection framework to a quasi-multivariate context we should find a group of candidate pairs for one particular security, which in a equities context can be an ETF or index as in the mentioned PCA approach of Avellaneda & Lee (2009), but also another individual stock. For example, In Chen et al. (2012) the return of a portfolio of securities is regressed against a particular one, archiving returns of 36% per year in contrast to the 9% of the univariate strategy. This shows that by using a portfolio of securities, we can extract a mean-reverting residual signal more purely i.e. we disentangle the ideosyncratic returns better than by using just one stock vs. another.

For a multivariate mean-reverting portfolio, Galenko et al. (2006) use the Vector Error Correction Model (an extension of more popular Vector Autoregressive model to account for long-term relationships), to compute a cointegration vector between a portfolio of assets. This strategy yields up to 70% annualized returns without accounting for transaction costs. But the more influential work seems to be Cuturi & d'Aspremont (2015), they present semi-definite relaxations that allow us to optimize via grid search three statistics:

1. Predictability. Measures how close to noise the series is. A time series is less predictable as it moves more closely to a purely random, white noise.

2. Portmanteau criterion. This statistic is 0 for a white noise process. It is a measure of serial autocorrelation.

3. Crossing statistic. As our mean-crossing statistic, measures how many times the mean value level is crossed.

A variety of papers implement this framework. Imai & Nakagama (2020), for example, report returns around 8-12% in equities. Zhao & Palomar (2018) show the method applied in a univariate format outperforms the cointegrated residual approach we used in 3.1., but they do not display a mean value return, just a chart of the accumulated profit and loss.

5 Conclusions

We have considered the pairs selection framework of Sarmento & Horta (2020), along with 3 different setups for signal extraction: non-parametric rules over the cointegrated residual, modeling the residual as an Ornstein-Uhlenbeck process and as an ARMA(p, q).

Let us compare our results with the literature, Fil (2020) showed a 0.18% of monthly returns, that compounded to 6 months gives 0.9% (using 3.1). This value is given with assumed (approximately) 45 bp of transaction costs in a dollar-neutral setup. Comparing with 3.1. at 50 bp and D-N allocation as well, our result is 2.31%, (+1.41%). This can be attributed to different pairs selection procedure: it his case is a simple cointegration method, but we use the more refined approach signaled above. Rad et al. (2016) showed an annual 4.37% return, and annualizing our 6-month 2.31% we get 4.64%, a very similar value.

The Ornstein-Uhlenbeck method in 3.2 from Avellaneda & Lee is not directly comparable at all, as the latter is in a quasi-multivariate context that differs substantially from ours. This process is used in many papers, but with very different assumptions and implementation (such as differents trading rules, and optimal threshold search). Our results are good enough to conclude this approach as better than the previous.

Finally, the ARMA(p, q) model did not give a great result and could not improve the more simple ARMA(1, 1), which gave us a 0.81% 6-month return that falls short compared to the 5.57% annual return in Sarmento & Horta (2020). But there are differences between both approaches. Their formation period is of 3-years (which makes the model much better trained), they select the most appropriate order using a validation period, and the signals are defined in a slightly different way. We refer to their paper for more details.

We have also seen the different sensibilities of dollar-neutral and beta-neutral allocations, being the latter more robust to threshold changes, but the former more profitable. Something reasonable as it uses all available capital, in constrast to the beta-neutral position sizing. The benefit of de-correlating the returns of our strategy with respect to the market do show, as they are already very uncorrelated with the market.

We conclude that the basic univariate statistical arbitrage strategy is not robust to every transaction cost assumption in the US equity markets. But further improvements in relation to the pairs selection and modelling approach used clearly pay off.

6 APPENDIX A1

This is the naive cointegration setup, we loop over this code for all semesters in our dataset.

```
def calc_zscore(spread):
    zscore = (spread - np.mean(spread))/np.std(spread)
    return zscore
def calc_norm_spread(pair, lookback_dict, i):
    security_0 = pair[0]
    security_1 = pair[1]

    end_index = len(open_bysemester['Semester_' + str(e)])  +i
    # to change estimation window
    start_index = i # + int(len(open_bysemester['Semester_' + str(e)]) * 2/3)
    window_data0 = lookback_dict[start_index:end_index][security_0]
    window_data1 = lookback_dict[start_index:end_index][security_1]

    x = sm.add_constant(np.asarray(window_data0))
    y = np.asarray(window_data1)

    model = sm.OLS(y, x)
    result = model.fit()

    alpha, beta = result.params[0], result.params[1]

    spread_res = y - (alpha + beta * x.T[1]) # construct residual series

    norm_spread = calc_zscore(spread_res)

    return pd.Series(norm_spread), beta

def pnl(risk_free, prices_dict, optimal_pairs, lookback_dict, closelong, closeshort,
sellth, buyth, fee):
    last_cum_returns = []
    pair_returns = []
    pair_daily_returns = []
    """
prices_dict contains open0 and open1, lookback_dict is the same as prices_dict but also
includes the previous semester. Risk_free is the daily 3-month t-bill rate
    """
    for pair, df in zip(optimal_pairs, prices_dict.values()):
        signals0 = np.zeros(len(df))
        pos1 = np.zeros(len(df))
        pos0 = np.zeros(len(df))
        days_since_trade = 0
        beta = 0

        for i in range(0, len(df)):
            if i < 1:
                prev_signal = 0
            else:
                prev_signal = signals0[i - 1]

            norm_spread, curr_beta = calc_norm_spread(pair, lookback_dict, i)
            curr_spread = norm_spread.iloc[-1]  # Extract the last value of the series

            if days_since_trade % 6 == 0:
                # Update beta every 5 days
                beta = curr_beta
```

```
        if (curr_spread >= sellth and prev_signal == 0) or
        (curr_spread <= buyth and prev_signal == 0):
            # Enter trade at current_beta
            signals0[i] = -1 if curr_spread >= sellth else 1
            days_since_trade = 1  # Reset the days since trade
            beta = curr_beta  # Set beta to current beta

        elif closelong < curr_spread and prev_signal == 1:
            signals0[i] = 0

        elif closeshort > curr_spread and prev_signal == -1:
            signals0[i] = 0

        else:
            signals0[i] = prev_signal

        if days_since_trade > 0:
            days_since_trade += 1

        # remove # to beta-neutral

        if abs(beta) <= 1:
            pos0[i] = -signals0[i] # * beta
            pos1[i] = signals0[i]
        else:
            pos0[i] = -signals0[i]
            pos1[i] = signals0[i] #/ beta

    df['pos0'] = pos0
    df['pos1'] = pos1

    pos0[-1] = 0
    pos1[-1] = 0

    df['pos0_diff'] = df['pos0'].diff().abs()
    df['pos1_diff'] = df['pos1'].diff().abs() # units buyed or selled for ts. costs

    df['logret_stock0'] = np.log(df['open0'] / df['open0'].shift(1))
    df['logret_stock1'] = np.log(df['open1'] / df['open1'].shift(1))

    df['logret_leg0'] = df['logret_stock0'] * df['pos0'].shift(1) -
    (df['pos0_diff'] * fee)
    df['logret_leg1'] = df['logret_stock1'] * df['pos1'].shift(1) -
    (df['pos1_diff'] * fee)

    df['cumulative_return0'] = df['logret_leg0'].dropna().cumsum().apply(np.exp)
    df['cumulative_return1'] = df['logret_leg1'].dropna().cumsum().apply(np.exp)

    last_cumulative_return0 = df['cumulative_return0'].iloc[-1] -1
    last_cumulative_return1 = df['cumulative_return1'].iloc[-1] -1

    total_return = (last_cumulative_return0 + last_cumulative_return1) *100 # return
    pair_returns.append((pair,total_return)) # create a list with each pair's return

    pair_ret = (np.exp(df['logret_leg0']) +  # compute daily pair returns for sharpe
                np.exp(df['logret_leg1']))
    pair_daily_returns.append(pair_ret)
```

```
    Final_returns = pd.DataFrame(pair_returns, columns=['Pair', 'Last_Cumpair_Return'])
    freturn =  Final_returns['Last_Cumpair_Return'].mean()

    #  print(Final_returns) to see the return corresponding to each pair

    pair_daily_returnsf = pd.concat(pair_daily_returns, axis=1)

    weights = np.array([1 / len(optimal_pairs)] * len(optimal_pairs))
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(pair_daily_returnsf.cov(), weights)))

    rfrate = risk_free.mean()

  #  portfolio_excess_return = freturn*2 - rfrate # annualize without coumpounding
    annualized_portfolio_excess_return = (((1+freturn/100) ** 2) - 1)*100 - rfrate
    sharpe_ratio = annualized_portfolio_excess_return / (portfolio_volatility*np.sqrt(252)*100)

    return freturn, portfolio_volatility, sharpe_ratio
```

7 APPENDIX A2

To adapt the previous code for Ornstein-Uhlenbeck we just need to substitute calc_norm_spread() for calc_ou_par().

```
def calc_ou_par(pair, result_dict, i):
    security_0 = pair[0]
    security_1 = pair[1]

    end_index = len(open_bysemester['Semester_' + str(e)])  +i
    start_index = i #+ int(len(open_bysemester['Semester_' + str(e)]) * 2/3)

    #now we use returns
    window_data0 = result_dict[start_index:end_index][security_0].pct_change().dropna()
    window_data1 = result_dict[start_index:end_index][security_1].pct_change().dropna()


    x = sm.add_constant(np.asarray(window_data0))
    y = np.asarray(window_data1)

    # Get parameters and calculate residual
    model = sm.OLS(y, x)
    result = model.fit()

    alpha, beta = result.params[0], result.params[1]

    res = y - (alpha + beta * x.T[1])

    cumres = res.cumsum()
    cumres_series = pd.Series(cumres)

    # model the cumulative residual with an AR(1)
    x_cumres = sm.add_constant(np.asarray(cumres_series.shift(1).dropna()))
    y_cumres = cumres_series[1:].reset_index(drop=True)
    model_cumres = sm.OLS(y_cumres, x_cumres)
    result_cumres = model_cumres.fit()
    a, b = result_cumres.params[0], result_cumres.params[1]

    aux_residuals = result_cumres.resid
    var_auxres = np.var(aux_residuals)
```

```
    # Calculate kappa and m
    kappa = -math.log(b) * len(open_bysemester['Semester_' + str(e)])
    m = a / (1 - b)

    sigma = math.sqrt(var_auxres * 2 * kappa / (1 - b**2))
    sigma_eq = math.sqrt(var_auxres / (1 - b**2))
    s = -m / sigma_eq
    s_mod = s - a/(kappa*sigma_eq)

    return s_mod, beta
```

8 APPENDIX A3

Now we substitute calc_our_par() for calc_model_par().

```
# Initialize a dictionary to store the selected ARMA order for each pair.
arma_orders = {}

def fit_arma(pair, p, q, cumres):
    try:
        model_cumres = smt.ARIMA(cumres, order=(p, 0, q),
        enforce_stationarity=False, enforce_invertibility=False)
        result_cumres = model_cumres.fit()
        aic = result_cumres.aic
        return aic, pair, p, q
    except ConvergenceWarning:
        pass
    except LinAlgError:
        print(f"LinAlgError occurred for ARMA order ({p}, 0, {q}) at i=0. Continuing with the next orde
        return float('inf'), pair, p, q

def calc_model_par(pair, result_dict, i, semester, high_quantile=0.9,
low_quantile=0.1, previous_s_mod=None):

    security_0 = pair[0]
    security_1 = pair[1]

    end_index = len(open_bysemester['Semester_' + str(e)]) + i
    start_index = i

    window_data0 = result_dict[start_index:end_index][security_0].pct_change().dropna()
    window_data1 = result_dict[start_index:end_index][security_1].pct_change().dropna()

    x = sm.add_constant(np.asarray(window_data0))
    y = np.asarray(window_data1)

    # Get parameters and calculate spread
    model = sm.OLS(y, x)
    result = model.fit()

    alpha, beta = result.params[0], result.params[1]

    res = y - (alpha + beta * x.T[1])
    cumres = res.cumsum()
    scaler = StandardScaler()
    cumres = scaler.fit_transform(cumres.reshape(x.shape[0], 1))
    cumres = pd.Series(data=cumres.flatten())
```

```
    last = cumres.iloc[-1]
    positive_changes = cumres[cumres > 0]
    negative_changes = cumres[cumres < 0]
    long_threshold = positive_changes.quantile(q=high_quantile, interpolation='linear')
    short_threshold = negative_changes.quantile(q=low_quantile, interpolation='linear')

    # Check if an ARMA order is already selected for this pair
    if (pair, semester) in arma_orders:
        order = arma_orders[(pair, semester)]
    else:
        # Iterate through AR and MA orders
        best_order = None
        best_aic = float('inf')

        # Parallelize the calculation of AIC for different ARMA orders
        aic_results = Parallel(n_jobs=-1)(
            delayed(fit_arma)(pair, p, q, cumres)
            for p in range(1, 9)  # AR order (1-8)
            for q in range(1, 9)  # MA order (1-8)
        )

        # Find the best order (minimum AIC) among all combinations
        for aic, pair, p, q in aic_results:
            if aic < best_aic:
                best_aic = aic
                best_order = (p, 0, q)

        # Store the selected ARMA order for this pair
        arma_orders[(pair, semester)] = best_order
        print(best_order, pair)
        order = best_order

    # Fit the ARMA model with the selected order
    try:
        model_cumres = smt.ARIMA(cumres, order=order,
        enforce_stationarity=False, enforce_invertibility=False)
        result_cumres = model_cumres.fit()
        aux_residuals = result_cumres.resid
        m = result_cumres.forecast(steps=1)
    except (ConvergenceWarning, LinAlgError):
        # Handle the exception when optimization fails to converge
        print(f"ARIMA optimization did not converge for pair {pair}. Using the previous s_mod value.")
        if previous_m is not None:
            m, beta, long_threshold,
            short_threshold = previous_m, beta, long_threshold, short_threshold
        else:
            m, beta = 0, beta
    else:
        previous_m = m.copy()

    return m.values, beta, long_threshold,
short_threshold, previous_s_mod
```

9 Bibliography

Rad, H., Low, R. K. Y., & Faff, R. (2016). The profitability of pairs trading strategies: distance, cointegration and copula methods. Quantitative Finance, 16(10), 1541-1558.

Do, B., & Faff, R. (2010). Does simple pairs trading still work?. Financial Analysts Journal, 66(4), 83-95.

Jeppesen, S. F., Matoso, L., & Richter, M. (2020). Liquidity Risk Premium in Pairs Trading.

Fil, M. (2020). Gold Standard Pairs Trading Rules: Are They Valid?. arXiv preprint arXiv:2010.01157.

Balladares, K., Ramos-Requena, J. P., Trinidad-Segovia, J. E., & Sánchez-Granero, M. A. (2021). Statistical arbitrage in emerging markets: a global test of efficiency. Mathematics, 9(2), 179.

Avellaneda, M., & Lee, J. H. (2010). Statistical arbitrage in the US equities market. Quantitative Finance, 10(7), 761-782.

Sarmento, S. M., & Horta, N. (2020). Enhancing a pairs trading strategy with the application of machine learning. Expert Systems with Applications, 158, 113490.

Elliott, R. J., Van Der Hoek, J., & Malcolm, W. P. (2005). Pairs trading. Quantitative Finance, 5(3), 271-276.

Chen, H., Chen, S., Chen, Z., & Li, F. (2012). Empirical investigation of an equity pairs trading strategy. Management Science, 65(1), 370-389.

Imai, T., & Nakagawa, K. (2020). Statistical arbitrage strategy in multi-asset market using time series analysis. Journal of Mathematical Finance, 10(2), 334-344.

Galenko, A., Popova, E., & Popova, I. (2012). Trading in the presence of cointegration. The Journal of Alternative Investments, 15(1), 85-97.

Zhao, Z., & Palomar, D. P. (2018). Mean-reverting portfolio with budget constraint. IEEE Transactions on Signal Processing, 66(9), 2342-2357.