CERTIFICATE IN QUANTITATIVE FINANCE

FINAL PROJECT

MULTIVARIATE COINTEGRATION IN STATISTICAL ARBITRAGE

JOÃO RAMOS JUNGBLUT

2024

Contents

1 INTRODUCTION

Arbitrage refers to the act of taking advantage of discrepancies in asset prices across different scenarios to gain risk-free profit. The goal is to buy an asset at a lower price in one market and sell it at a higher price in another, capitalizing on this difference.

The underlying assumption behind such a concept is that financial markets are efficient, as per Fama [1970]. The hypothesis suggests that markets reflect all available information, making profit opportunities short-lived because participants tend to act quickly to exploit these differences, restoring price equilibrium. Therefore, it is easier to find arbitrage by trading multiple assets, precisely because it is more challenging to model hidden patterns in multiple variables than in just one.

Pairs Trading is a statistical approach developed to identify pairs of assets with price co-movement, aiming to anticipate convergence to equilibrium in situations of overvaluation or undervaluation. This strategy was developed in the 1980s, according to Cavalcante et al. [2016], and gained prominence in academia after the publication of the distance method by Gatev et al. [2006], which proposes selecting pairs by minimizing a distance criterion (Euclidean). Perlin [2009] assesses the effectiveness of this approach in the Brazilian market, which I will explore in this work.

However, Taylor [2011] argues that the distribution of stocks does not follow a normal distribution. This means that short-term linear relationships, characterized by metrics such as correlation, are inconsistent and unreliable. Therefore, it is more appropriate to use methods that capture non-linear dependence. For example, the use of copulas for constructing a mispricing index, as explored by Xie et al. [2016], and more recently, the mixture of copulas proposed by da Silva et al. [2023].

Cointegration, despite being a linear statistical method, is more appealing to work with because it captures long-term equilibrium relationships. This enables the identification of price divergences that align with a theory consistently. Some works provide practical insights, such as, for example, Chan [2021] and Diamond [2014].

In this approach, we are essentially interested in modeling two assets to produce a stationary time series called spread. Thus, if the two assets have a equilibrium relationship, this series tends to revert to the mean in the long term, allowing for long and short operations. An empirical analysis of the Brazilian market involving cointegration has already been conducted by Caldeira and Moura [2013]. They found high profitability in applying cointegration, but did not impose any restrictions on the pair selection universe.

Seeking to enhance the method, Ramos-Requena et al. [2017] introduced the use of the Hurst exponent to assess the quality of random walk diffusion, observing whether the spread is truly mean-reverting. This could be done through the variance test proposed by Lo and MacKinlay [1988], aiming to identify the hypothesis that financial asset prices follow or not a random walk. Alternatively, the evaluation could be performed through the half-life, calculated from the Ornstein-Uhlenbeck process, as employed by Teixeira [2014]. The latter used it as a criterion to determine the exit time of operations to reduce losses and increase profitability.

Finally, Sarmento and Horta [2021] combines the techniques described above with the use of unsupervised machine learning models to improve efficiency in identifying pairs of ETFs in intraday data. The author employs the OPTICS and DBSCAN algorithms to narrow the search for cointegrated pairs within specific clusters. Subsequently, filters are developed using the Hurst exponent and half-life for the selection of the most promising pairs.

That being said, some considerations must be made. The first of these concerns the sensitivity of these models to specific market conditions. The performance of pair-based strategies may be affected by extraordinary events or periods of extreme volatility, which can compromise the robustness and reliability of the results. Regarding this, Palomar [2020] demonstrates how to use the Kalman filter to make corrections to the spread when structural breaks occur, as well as how to weigh each asset in the operations.

Another significant shortcoming of the pair selection technique is the fact that it does not provide ways to construct a well-optimized portfolio. Simply identifying cointegrated pairs does not take into account effective diversification and resource allocation. In this regard, what I will do in this project is build a cointegrated portfolio using Johansen [1988] test which allows for exploring cointegration in a multivariate case.

I will use stock price data comprising the Bovespa index from 1996 to 2023, collected from Economática. The database will be consistently divided into one year in-sample periods and six month out-of-sample periods. I will seek cointegrated eigenvectors using the

Johansen test and construct stationary residuals with the aim of profiting from statistical arbitrage operations.

The structure of this work is organized as follows: in Chapter 2, I detail in Section 2.1 the specifications of VAR, Cointegration, VECM, Johansen Approach, and Maximum Likelihood Estimation. In Section 2.2, the strategy is presented, including the definition of entry and exit points, as well as the calculation of the percentage of each asset in the portfolio. Section 2.3 discusses the measures used to evaluate the portfolio's performance. In the following chapter, 3, I present the data used in 3.1 more clearly and the results in Section 3.2, along with the robustness test in 3.3. Finally, in the conclusion, Chapter 4, I revisit the results and fundamental concepts of the model to discuss improvements and future work. The codes are available in Appendix.

2 METHODOLOGY

2.1 Model Specification

The models for implementing the trading strategy will be presented here. First, I will explain the Vector Autoregressive (VAR), which serves as the basis for a deeper understanding of the others. Following that, I will address the concept of cointegration, and subsequently formulate the Vector Error Correction Model (VECM). The Johansen test will be used to identify cointegration vectors, and parameter estimation will then be performed through Maximum Likelihood. Finally, I will highlight how the strategy is executed through the spread, and the risk-return measures used to evaluate portfolio performance. The notations used are based on the works of Bueno [2018].

2.1.1 Vector Autoregressive (VAR)

The VAR expresses entire economic models by providing constraints and equations, allowing parameters to be estimated. It is significant because it examines the trajectory of endogenous variables in the presence of a structural shock. Following this approach, it is possible to use the information presented to separate long-term patterns from short-term ones, thereby identifying mispriced assets. These variations are studied using residuals caused by noise in the price series of financial assets, and modeling them leads to the VECM.

In a general sense, we can express a model of order p, with n endogenous variables, interconnected by a matrix A:

$$AX_t = B_0 + \sum_{i=1}^{p} B_i X_{t-i} + B\varepsilon_t, \quad \varepsilon_t \sim i.i.d.$$

A is a $n \times n$ matrix that defines the simultaneous constraints among the variables forming the $n \times 1$ endogenous variables at time t, X_t. The constant vector B_0 is $n \times 1$, B_i is a $n \times n$ matrix, B is a diagonal matrix of standard deviations, and ε_t is a $n \times 1$ vector of uncorrelated random disturbances.

Examining the explanation for a bivariate situation helps to better understand the model's endogeneity. Consider these equations:

$$x_{1,t} = b_{10} \boxed{-a_{12}x_{2,t}} + b_{11}x_{1,t-1} + b_{12}x_{2,t-1} + \sigma_{x_1}\varepsilon_{x_1,t}$$
$$x_{2,t} = b_{20} \boxed{-a_{21}x_{1,t}} + b_{21}x_{1,t-1} + b_{22}x_{2,t-1} + \sigma_{x_2}\varepsilon_{x_2,t}.$$

In this context, *(i)* $x_{1,t}$ and $x_{2,t}$ are stationary, *(ii)* $\varepsilon_{x_1,t} \sim RB(0,1)$ and $\varepsilon_{x_2,t} \sim RB(0,1)$, and *(iii)* $\varepsilon_{x_1,t} \perp \varepsilon_{x_2,t} \Rightarrow Cov(\varepsilon_{x_1,t}, \varepsilon_{x_2,t}) = 0$.

These conditions are required for the model's validity and the accurate interpretation of the results. By ensuring the stationarity of variables and the independence of error terms, we may proceed with explaining the reduced form of the simple model, which becomes:

$$\begin{aligned} X_t &= A^{-1}B_0 + A^{-1}\sum_{i=1}^{p} B_i X_{t-i} + A^{-1}B\varepsilon_t \\ &= \Phi_0 + \sum_{i=1}^{p} \Phi_i X_{t-i} + e_t. \end{aligned}$$

The simple model's reduced form is

$$X_t = \Phi_0 + \Phi_1 X_{t-1} + e_t,$$

where $\Phi_0 = A^{-1}B_0$, $\Phi_1 = A^{-1}B_1$, and $Ae_t = B\varepsilon_t$. The stability condition is ensured when the eigenvalues are positioned outside the unit circle $(I - \Phi_1 L)$, which ensures convergence over time. Moving on, the question arises about the presence of trends in the variables, i.e., if one may predict the other under what circumstances. This question introduces the notion of cointegration.

2.1.2 Cointegration

According to Engle and Granger [1987], the elements of the vector X_t, $n \times 1$, are said to be cointegrated of order (d,b), denoted by $X_t \sim CI(d,b)$, if (i) all elements of X_t are integrated of order d, $I(d)$, and (ii) there exists a non-zero vector β such that

$$u_t = X_t'\beta \sim I(d-b), \quad b > 0.$$

Therefore, when $X_t'\beta = 0$, β is the cointegration vector that defines a linear combination among the elements of X_t.

Consider $\beta = [\widetilde{\beta}_1, \widetilde{\beta}_2]$, which defines the long-term equilibrium between the variables $I(1)$, $x_{1,t}$, and $x_{2,t}$. Then,

$$\begin{bmatrix} x_{1,t} & x_{2,t} \end{bmatrix} \begin{bmatrix} \widetilde{\beta}_1 \\ \widetilde{\beta}_2 \end{bmatrix} = \widetilde{\beta}_1 x_{1,t} + \widetilde{\beta}_2 x_{2,t} = 0$$

$$\begin{bmatrix} x_{1,t} & x_{2,t} \end{bmatrix} \begin{bmatrix} 1 \\ \beta_2 \end{bmatrix} = x_{1,t} + \beta_2 x_{2,t} = 0.$$

With the normalization of $\beta_2 = \frac{\widetilde{\beta}_2}{\widetilde{\beta}_1}$, u_t can be considered the residual of one coordinate of X_t against the other variables. Thus, the variables are cointegrated, and β generates the residual whose order of integration is lower than the original variables. The same holds for cases with more than two variables.

The VAR is significant because it attempts to predict the trajectory of endogenous variables in the face of a structural shock. To trade, we must first assess the shock and determine entry and exit points. Given the condition for a stationary disturbance, the logical next step is to test the residuals, fit the best VAR model, and send the information to the VECM.

An important observation is that the deterministic trend dominates first-order non-stationarity, which, in turn, dominates stationarity in the sense of asymptotic convergence.

2.1.3 Vector Error Correction Model (VECM)

The VECM is nothing more than a conventional VAR, but it includes the error correction term. To visualize this, we need to consider a cointegration relationship given by:

$$x_{1,t} = \mu + \beta x_{2,t} + u_t.$$

So, there are ways to manipulate the VAR such that, if cointegration exists, the original model can be rewritten for the residuals to enter explicitly:

$$\Delta x_{1,t} = \alpha_1 \hat{u}_{t-1} + \sum_{j=1}^{p-1} \lambda_{11,j+1} \Delta x_{1,t-j} + \sum_{j=1}^{p-1} \lambda_{12,j+1} \Delta x_{2,t-j} + e_{x_1,t}$$
$$\Delta x_{2,t} = \alpha_2 \hat{u}_{t-1} + \sum_{j=1}^{p-1} \lambda_{21,j+1} \Delta x_{1,t-j} + \sum_{j=1}^{p-1} \lambda_{22,j+1} \Delta x_{2,t-j} + e_{x_2,t}.$$

This is the multivariate model, where each X_t is a $n \times 1$ vector of endogenous variables. Consider the VAR at the level, ignoring the presence of constants, to better comprehend the VECM.

$$X_t = \Phi_1 X_{t-1} + \Phi_2 X_{t-2} + \ldots + \Phi_p X_{t-p} + e_t$$
$$[I - (\Phi_1 L + \Phi_2 L^2 + \ldots + \Phi_L^p)] X_t = e_t$$
$$\Phi(L) X_t = e_t.$$

Note, when $L = 1$,

$$\Phi(1) = [I - (\Phi_1 + \Phi_2 + \ldots + \Phi_p)] = -\Phi.$$

The characteristic polynomial is given by:

$$\Phi(Z) = I - \sum_{i=1}^{p} \Phi_i Z^i,$$

where Z is a diagonal matrix of n elements. Linear algebra tells us that if the matrix's determinant is zero, its rank is not full. That is, $[\Phi(I)] = 0 \Longleftrightarrow \text{rank}(\Phi) < n$. As a result, the process has a unit root, and Z can be factored as follows:

$$\Phi(Z) = (I - Z)(I - \lambda_1 Z)(I - \lambda_2 Z) \ldots (I - \lambda_p Z).$$

A matrix's rank is the number of independent rows or columns. The number of columns and rows will always be fewer than or equal to the rank. This allows us to state Granger's theorem.

Theorem 1 (Granger's Theorem) *If* $|\Phi(Z)| = 0$ *implies* $Z > I$ *and* $0 < rank(\Phi) = r < n$*, then there exist matrices* α *and* β *with dimensions* $n \times r$ *such that:* $\Phi = \alpha\beta$*.*

- β is the matrix of **cointegration**.
- α is the matrix of **adjustment**.

The theorem expresses the idea that Φ can be decomposed into two multiplicative matrices. From this, we derive the VECM, recursively adding and subtracting past terms to generalize the equation. Thus, let

$$\begin{aligned}
X_t &= \Phi_1 X_{t-1} + \Phi_2 X_{t-2} + \phi_3 X_{t-3} + e_t \\
&= \Phi_1 X_{t-1} + \Phi_2 X_{t-2} \boxed{+\Phi_3 X_{t-2} - \Phi_3 X_{t-2}} + \Phi_3 X_{t-3} + e_t \\
&= \Phi_1 X_{t-1} + (\Phi_2 + \Phi_3) X_{t-2} - \Phi_3 \Delta X_{t-2} + e_t \\
&= \Phi_1 X_{t-1} \boxed{+(\Phi_2 + \Phi_3) X_{t-1} - (\Phi_2 + \Phi_3) X_{t-1}} + (\Phi_2 + \Phi_3) X_{t-2} - \Phi_3 \Delta X_{t-2} + e_t \\
&= (\Phi_1 + \Phi_2 + \Phi_3) X_{t-1} + (\Phi_2 + \Phi_3) \Delta X_{t-1} - \Phi_3 \Delta X_{t-2} + e_t \\
X_t \boxed{-X_{t-1}} &= \boxed{-X_{t-1}} + (\Phi_1 + \Phi_2 + \Phi_3) X_{t-1} + (\Phi_2 + \Phi_3) \Delta X_{t-1} - \Phi_3 \Delta X_{t-2} + e_t \\
\Delta X_t &= -[I - (\Phi_1 + \Phi_2 + \Phi_3)] X_{t-1} + (\Phi_2 + \Phi_3) \Delta X_{t-1} - \Phi_3 \Delta X_{t-2} + e_t \\
&= \Phi X_{t-1} + \sum_{i=1}^{2} \Lambda_i \Delta X_{t-i} + e_t,
\end{aligned}$$

with $\Lambda_i = \sum_{j=1+i}^{3} \Phi_j$, $i = 1, 2$.

The model is called *error correction* because it explains ΔX_t by two components: the short-term factors and the long-term relationship given by the coordinates of the vector of endogenous variables. It becomes evident, therefore, that in the presence of cointegration, it is always possible to associate the VAR with error correction, and that is precisely what the representation theorem deals with.

Theorem 2 (Granger Representation Theorem) *If* $X_t \sim CI(1,1)$*,* X_t *has a representation in the form of a VECM.*

2.1.4 Johansen's approach

Johansen presents a test to identify the rank of the matrix Φ and, consequently, estimate the cointegration vectors included in the matrix β. His methodology is intriguing because it is performed concurrently with the estimation of the cointegration model.

Consider the following non-stationary equation system in cointegration:

$$X_t = \beta_0 + \beta_1 X_{t-1} + u_t,$$

in vector form,

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix} = \begin{bmatrix} \beta_{10} \\ \beta_{20} \end{bmatrix} + \begin{bmatrix} \beta_{11} & \alpha_{11} \\ \alpha_{21} & \beta_{21} \end{bmatrix} \begin{bmatrix} x_{1,t-1} \\ X_{2,t-1} \end{bmatrix} + \begin{bmatrix} u_{1,t} \\ u_{2,t} \end{bmatrix}.$$

In this system, x_{1t} and x_{2t} are non-stationary $I(1)$ processes. A linear combination exists that is $I(0)$ when they are cointegrated. We can express the VAR model in terms of the $I(0)$ variables only.

$$\begin{bmatrix} x_{1,t} - x_{1,t-1} \\ x_{2,t} - x_{2,t-1} \end{bmatrix} = \begin{bmatrix} \beta_{10} \\ \beta_{20} \end{bmatrix} + \begin{bmatrix} \beta_{11} - 1 & \alpha_{11} \\ \alpha_{21} & \beta_{21} - 1 \end{bmatrix} \begin{bmatrix} x_{1,t-1} \\ X_{2,t-1} \end{bmatrix} + \begin{bmatrix} u_{1,t} \\ u_{2,t} \end{bmatrix}.$$

$$\begin{bmatrix} \Delta x_{1,t} \\ \Delta x_{2,t} \end{bmatrix} = \begin{bmatrix} \beta_{10} \\ \beta_{20} \end{bmatrix} + \begin{bmatrix} \Phi_{11} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix} \begin{bmatrix} x_{1,t-1} \\ X_{2,t-1} \end{bmatrix} + \begin{bmatrix} u_{1,t} \\ u_{2,t}. \end{bmatrix}.$$

Now, the model is represented in the VECM form, as explained by the Granger Representation Theorem.

$$\Delta X_t = \beta_0 + \Phi X_{t-1} + u_t$$

The matrix's rank indicates the number of linearly independent rows and columns. If one of the rows cannot be stated as a multiple of the other, they are considered independent. As a result, the existence of a $I(0)$ linear combination for x_1 and x_2 depends on the rank. Johansen [1991] proposed two tests to evaluate the matrix's rank: the trace test and the maximum eigenvalue test. We are solely interested in the maximum eigenvalue test for this work, where the hypothesis is formulated as follows, given r is the maximum number of cointegrated eigenvectors.

$$H_0 : rank(\Phi) < r \quad or \quad \Phi = \alpha\beta'$$

Even after defining the rank, the matrices α and β are not identifiable as they form an overparameterization of the model. However, we can delimit the cointegration space to $span(\beta)$.

2.1.5 Maximum Likelihood Estimation of Φ

The estimation of Φ can be done by maximum likelihood. Drawing from the discussions in Maddala and Kim [1998] and its references, consider the complete VAR:

$$X_t = \Phi_1 X_{t-1} + \Phi_2 X_{t-2} + \ldots + \Phi_p X_{t-p} + \delta d_t + e_t.$$

This model can be represented in error correction form, and the objective is to estimate using Maximum Likelihood subject to the constraint of incomplete rank: $\Phi = \alpha\beta'$.

$$\Delta X_t = \alpha\beta' X_{t-1} + \sum_{i=1}^{p} \Lambda_i \Delta X_{t-i} + \delta d_t + e_t$$

Consider vectorizing the above model, where $\Upsilon_{0,t} = \Delta X_t$, $\Upsilon_{1,t} = X_{t-1}$, $\Upsilon_{2,t} = [\Delta X'_{t-1} \Delta X'_{t-2} \ldots, d'_t]$, and $\Lambda = [\Lambda_1 \Lambda_2 \ldots \delta]$. The VECM is simplified to:

$$\Upsilon_{0,t} = \alpha\beta' \Upsilon_{1,t} + \Lambda \Upsilon_{2,t} + e_t.$$

The problem is to maximize the likelihood function subject to nonlinear constraints on the parameters given by:

$$ln L(\alpha, \beta, \Lambda, \Sigma) = -\frac{T}{2} ln|\Sigma| - \frac{1}{2} \sum_{t=1}^{T} e'_t \Sigma^{-1} e_t,$$

with the first-order conditions being:

$$0 = \sum_{t=1}^{T} (\Upsilon_{0,t} - \alpha\beta'\Upsilon_{1,t} + \hat{\Lambda}\Upsilon_{2,t})\Upsilon'_{2,t}$$
$$\Rightarrow \sum_{t=1}^{T} \Upsilon_{0,t}\Upsilon'_{2,t} = \alpha\beta' \sum_{t=1}^{T} \Upsilon_{1,t}\Upsilon'_{2,t} + \hat{\Lambda} \sum_{t=1}^{T} \Upsilon_{2,t}\Upsilon'_{2,t}.$$

Renaming the variables as $\Pi_{ij} = \frac{\sum^T t = 1 \Upsilon ij \Upsilon'_{ij}}{T}$, we can rewrite the previous equation as:

$$\Pi_{02} = \alpha\beta'\Pi_{12} + \hat{\Lambda}\Pi_{22}.$$

I will show that to obtain e_t, we do not need $\hat{\Lambda}$.

First, let's regress $\Upsilon_{0,t}$ on $\Upsilon_{2,t}$, that is,

$$\Upsilon_{0,t} = B\Upsilon_{1,t} + r_{0,t}$$

so that $\hat{B} = (\Upsilon_{0,t}\Upsilon'_{2,t})(\Upsilon_{2,t}\Upsilon'_{2,t})^{-1} = \Pi_{02}\Pi_{22}^{-1}$, and obtain the residuals

$$r\hat{_{0,t}} = \Upsilon_{0,t} - \Pi_{02}\Pi_{22}^{-1}\Upsilon_{2,t}.$$

Next, I will regress $\Upsilon_{1,t}$ on $\upsilon_{2,t}$ and obtain the residuals $r\hat{_{1,t}} = \Upsilon_{1,t} - \Pi_{12}\Pi^{-1}22\Upsilon 2,t$. Note that:

$$\begin{aligned} \hat{e_t} &= \Upsilon_{0,t} - \alpha\beta' r_{1,t} - (\Pi_{02}\Pi_{22}^{-1} - \alpha\beta'\Pi_{12}\Pi_{22}^{-1})\Upsilon_{2,t} \\ &= \Upsilon_{0,t} - \Pi_{02}\Pi_{22}^{-1}\Upsilon_{2,t} - \alpha\beta'(\Upsilon_{1,t} - \Pi_{12}\Pi_{22}^{-1}\Upsilon_{2,t}) \\ &= r\hat{_{0,t}} - \alpha\beta r\hat{_{1,t}}. \end{aligned}$$

Thus, the quantities $r\hat{_{0,t}}$ and $r\hat{_{1,t}}$ can be obtained from the auxiliary regressions, and we can estimate α and β. The new function to maximize becomes:

$$lnL(\alpha,\beta,\Sigma) = -\frac{T}{2}ln|\Sigma| - \frac{1}{2}\sum_{t=1}^{T}(r\hat{_{0,t}} - \alpha\beta r\hat{_{1,t}})'\Sigma^{-1}(r\hat{_{0,t}} - \alpha\beta r\hat{_{1,t}}).$$

Equivalent to maximizing this function, one can perform OLS regression between $r\hat{_{0,t}}$ and

$\hat{r_{1,t}}$. One way is to estimate α and Σ for a given β,

$$\hat{\alpha}(\beta) = S_{01}\beta(\beta' S_{11}\beta)$$

$$\hat{\Sigma}(\beta) = S_{00} - S_{01}\beta(\beta' S_{11}\beta)^{-1}\beta' S_{10}$$

with $S = T^{-1}\sum_{t=1}^{T}\hat{r_{i,t}}\hat{r_{j,t}},\ j = 0,1.$

The matrix β has not been estimated yet; nonetheless, starting from the last derivation, β can be found by maximizing the likelihood.

$$L(\beta)^{-\frac{2}{t}} = |\hat{\Sigma}(\beta)|,$$

where, according to Johansen [1991], we can obtain it through the maximum eigenvalue test.

$$L_{max}^{-\frac{2}{t}} = |S_{00}|\prod_{i=1}^{n}(1-\hat{\lambda}_i).$$

Thus, the cointegrated eigenvectors will define the matrix

$$\hat{\beta} = \left[\hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_r\right]$$

which is an $n \times r$ matrix.

In this context, let's clarify some details further.

$$\Pi_{02} = \alpha\beta'\Pi_{12} + \hat{\Lambda}\Pi_{22}$$

contains the coefficients associated with cointegration, with α and β being the coefficients we want to estimate. $r_{0,t}$ and $r_{1,t}$ are related to the variables deviating from their cointegration relationship (observed divergences), where $r_{0,t}$ is the residual of X_t (level) and $r_{1,t}$ is the residual of ΔX_t (trend). Furthermore, as we are only looking at the cointegrated space, the total residual presented will be

$$\hat{e}_t = \hat{\beta}' X_t.$$

2.2 Trading Strategy

The chosen strategy is similar to pairs trading using the cointegration method. Our approach, on the other hand, distinguishes itself by attempting to build a cointegrated portfolio. The central proposal is to identify periods when the movement of a financial asset deviates from the others in the short-term, anticipating a long-term convergence to an equilibrium.

Detecting these price divergences requires observing the residuals of the linear relationships. Although the strategy is almost identical to pairs trading, in the multivariate model, we have n financial assets. As explained in Section 2.1, the residuals are given by:

$$\hat{e}_t = \hat{\beta}' X_t.$$

To clarify the strategy's implementation, we shall use a separate nomenclature here. During the out-of-sample phase, the computed residuals will be referred to as spread and will represent the vector of price divergences to be exploited. The Z-score which is effectively the standardized spread, is calculated from these residuals and is represented as $Z_{\text{score}} = \frac{\hat{e}_t - \mu_e}{\sigma_e}$.

Meanwhile, the normalized cointegrated vector $\hat{\beta} = [1, \frac{\hat{\beta}_2}{\hat{\beta}_1}, \frac{\hat{\beta}_3}{\hat{\beta}_1}, \ldots, \frac{\hat{\beta}_n}{\hat{\beta}_1}]$, obtained by Maximum Likelihood Estimation (MLE), will constitute the hedge ratios. To determine the portfolio weights W, we divide the hedge ratios by their norm.

$$W = \frac{\hat{\beta}}{||\hat{\beta}||}$$

This way, we secure two critical points: first, that only available capital is used; second, that overly significant percentages are not concentrated in a single asset. The portfolio return is calculated as:

$$R_{port} = W R_t.$$

The strategy is built on the spread principle. Entry points are defined by the spread's distance from the mean, whereas exit points are determined by the spread's proximity to the mean. The Z-score is important in this context because it allows for the visualization of mean-reverting movements and determines entry and exit locations based on standard deviations.

I have used the following parameters:

- **Entry Point:** 1 standard deviation;
- **Exit Point:** 0 (mean);
- **Stop Loss:** 3 standard deviations.

When the Z-score reaches -1, we open a long position in the portfolio, closing it with a positive return at a Z-score of 0 or with a loss at -3. Conversely, as the Z-score achieve 1, we open a short position. The figure 1 depicts these locations, with the dashed green lines representing entrance points, the dashed black lines representing exit points, the dashed red line representing the Stop Loss, and the blue line representing the spread.

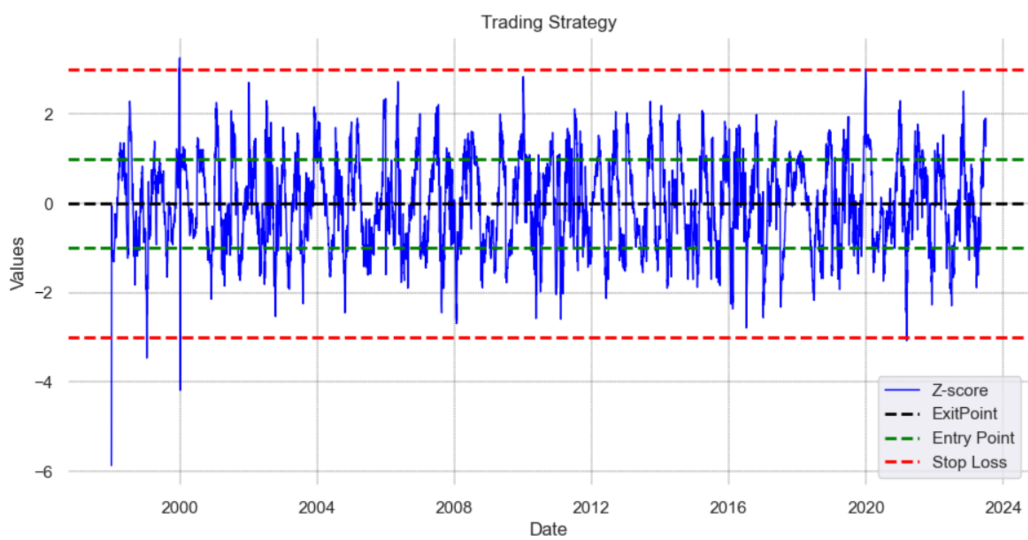

Figure 1: Trading Strategy

2.3 Performance Measures

The evaluation of the investment portfolio's performance involves the computation of diverse measures. These metrics offer a thorough evaluation of the portfolio's risk and return attributes, facilitating a more nuanced understanding of its risk and return characteristics across different market conditions. The following key measures are calculated: annualized return, annualized standard-deviation, annualized Sharpe ratio, Sortino ratio, $VaR_{0.95}$, $CVaR_{0.95}$ and Worst-Drawdown.

Annualized Return

The annualized return serves as a metric for gauging the average rate of return per year during a designated investment period. Typically computed using the formula:

$$\text{An. Ret.} = \left(\prod_{t=1}^{252} 1 + R_t\right)^{\frac{1}{252}} - 1$$

where R_t denotes the daily return, this measure encapsulates the compounded effect of daily returns over the entire investment horizon. For logarithmic returns r_t, the formula takes a slightly different form:

$$\text{An. Ret.} = \left(\frac{1}{252}\sum_{t=1}^{252} r_t\right)$$

This representation for logarithmic returns provides an alternative perspective on the annualized return, particularly when dealing with continuously compounded returns.

Annualized Standard Deviation

This metric quantifies the volatility or risk of the investment by measuring the dispersion of returns around the mean over a one-year period. The annualized Standard Deviation is the rescaled daily Standard Deviation and can be calculated as

$$\text{An. Std.-Dev.} = \sigma \times \sqrt{252}.$$

Sharpe Ratio

The Sharpe ratio evaluates the risk-adjusted return by comparing the portfolio's excess return over the risk-free rate to its standard deviation. It is a widely used metric in finance to assess the risk-adjusted performance of an investment or portfolio. It quantifies the excess return generated per unit of risk taken. The formula for the Sharpe ratio is given by:

$$\text{SR} = \frac{E[R] - R_f}{\sigma}.$$

Sortino Ratio

The Sortino ratio is a modified version of the Sharpe ratio that considers only downside risk, which is calculated using the semi-deviation. It tries to provide a more relevant measure of risk-adjusted performance, especially in instances when investors are concerned about downside volatility.

$$\text{Sortino} = \frac{E[R] - R_f}{\sigma^-}.$$

Value at Risk (VaR)

Value at Risk, denoted as $VaR_\beta(X)$, is a critical risk management metric that assesses the maximum expected loss with a $1-\beta$ confidence level over a defined time horizon. It serves as a powerful tool for quantifying the potential downside risk inherent in an investment portfolio.

The calculation for VaR_β involves determining the loss level at which there is a probability of exceeding it during the specified time frame. Mathematically, it can be expressed as follows:

$$\begin{aligned} Pr(x \leq VaR(X)) = 1 - \beta \\ VaR_\beta(X) &= \inf\{x \,|\, Pr(X > x) \leq 1 - \beta\} \\ &= \inf\{x \,|\, F_X(x) \geq \beta\} \end{aligned}$$

Here, I will be using VaR at 5% level.

Conditional Value at Risk (CVaR)

Conditional Value at Risk, denoted as $CVaR_\beta(X)$, represents a critical risk measure used to gauge the expected loss under extreme scenarios. It is calculated as the expected value of the random variable X conditional on X being less than or equal to its Value at Risk (VaR) at the same confidence level. This can be expressed mathematically as follows:

$$\begin{aligned} CVaR_\beta(X) &= E[X|X \leq VaR_\beta(X)] \\ &= \frac{1}{1-\beta}\int_0^{1-\beta} VaR_\alpha(X)\, d\alpha \end{aligned} \tag{1}$$

In essence, $CVaR_\beta(X)$ provides valuable insights into the potential loss magnitude beyond the $VaR_\beta(X)$ threshold, taking into account extreme scenarios with a confidence level of $1-\beta$. The level of CVaR used is 5%.

Worst-Drawdown

The worst-drawdown is a measure that quantifies the maximum percentage decline in a portfolio's value from a previous peak to the lowest subsequent point. It helps investors understand the largest loss they might have experienced during a specific investment period. The worst-drawdown can be calculated using the following formula:

$$\text{Worst-Drawdown} = \text{Maximum Drawdown} = \max_{i,j}\left(\frac{V_i - V_j}{V_i}\right) \times 100\%,$$

where V_i is the portfolio's value at time i, V_j is the lowest subsequent value after the peak at time j, and $\max_{i,j}$ represents the maximum value over all peak-to-trough periods.

3 EMPIRICAL ANALYSIS

To assess the effectiveness of the proposed strategy, I conducted a rigorous backtest, taking into account statistical biases. Initially, the data was collected and appropriately handled. To avoid model overfitting, cointegration analysis was only performed in the in-sample period, while trading operations were only undertaken in the out-of-sample period.

The assets used were randomly selected from the composition of the Bovespa index, mitigating potential issues related to data-snooping bias and survivorship bias. The generated signals were verified, and entries were made on the following day to avoid look-ahead bias, ensuring that the portfolio's return was calculated without incorporating future information.

Subsequently, risk and return measures were examined, and the portfolio's performance was compared to a benchmark, the Bovespa index, demonstrating the strategy's success. Transaction costs of 0.03% based on Frazzini et al. [2018] have been included. These steps were taken to ensure a thorough examination, considering numerous circumstances that could impact the backtest results.

3.1 Data

The historical data of adjusted close prices for stocks and the composition of the Bovespa index were collected from Economática database, covering the period from January 2, 1997, to December 28, 2023. A total of 203 assets were used during this period, including the tickers listed in Table 1.

The assets were separated between in-sample periods of one year and out-of-sample intervals of six months. The dataset is updated every six months. The assets in the in-sample dataset are always chosen because they were part of the Bovespa index composition at the time. The same assets are kept during the out-of-sample period, whether or not they were delisted. Figure 2 illustrates this division over time, with the in-sample period in blue and the out-of-sample period in green.

During the backtest execution, the priority was given to ensuring that the portfolio contained a total of 10 assets to avoid excessive diversification. The assets were selected

ABEV3	ACES4	AEDU3	ALLL11	ALLL3	ALPA4	AMBV4	AMER3	ARCE3
ARCE4	ARCZ6	ASAI3	ATMP3	AZUL4	B3SA3	BBAS3	BBAS4	BBDC3
BBDC4	BBSE3	BEEF3	BESP4	BHIA3	BIDI11	BISA3	BMTO4	BNCA3
BPAC11	BPAN4	BRAP4	BRDT4	BRFS3	BRKM5	BRML3	BRPR3	BRTP3
BRTP4	CASH3	CCRO3	CESP5	CESP6	CEVA4	CGAS5	CIEL3	CLSC4
CMET4	CMIG3	CMIG4	CMIN3	COGN3	CPFE3	CPLE6	CPSL3	CRFB3
CRTP5	CRUZ3	CSAN3	CSNA3	CSTB4	CTAX4	CTIP3	CVCB3	CYRE3
DASA3	DURA4	DXCO3	EBEN4	EBTP3	EBTP4	ECOR3	EGIE3	ELET3
ELET6	ELPL4	EMAE4	EMBR3	EMBR4	ENBR3	ENEV3	ENGI11	EPTE4
EQTL3	ERIC4	EVEN3	EZTC3	FIBR3	FLRY3	GEPA4	GETI4	GFSA3
GGBR4	GNDI3	GOAU4	GOLL4	HAPV3	HGTX3	HYPE3	IGTA3	IGTI11
INEP4	IRBR3	ITSA4	ITUB4	JBSS3	JHSF3	KLBN11	KLBN4	LAME4
LAND3	LCAM3	LIGT3	LIPR3	LOGG3	LREN3	LWSA3	MGLU3	MMXM3
MRFG3	MRVE3	MULT3	NETC4	NTCO3	OGXP3	OIBR3	OIBR4	PALF3
PCAR3	PCAR4	PDGR3	PETR3	PETR4	PETZ3	PMAM4	POMO4	POSI3
PRIO3	PRML3	PRTX3	PTIP4	QUAL3	RADL3	RAIL3	RCTB31	RCTB41
RDCD3	RDOR3	RENT3	REPA4	RLOG3	RRRP3	RSID3	RUMO3	SANB11
SAPR11	SBSP3	SDIA4	SHAP4	SLCE3	SMLS3	SOMA3	SUBA3	SULA11
SUZB3	SUZB5	SYNE3	TAEE11	TAMM4	TBLE6	TCOC4	TCSL4	TELB3
TELB4	TIMS3	TLCP4	TMAR5	TMAR6	TMCP4	TNEP4	TNLP3	TNLP4
TOTS3	TPRC6	TRJC6	TRPL4	TSPC3	TSPC6	UBBR11	UGPA3	UGPA4
UNIP6	USIM3	USIM5	VALE3	VALE5	VBBR3	VCPA4	VIVO4	VIVT3
VIVT4	VVAR11	WEGE3	WHMT3	YDUQ3				

Table 1: List of Assets

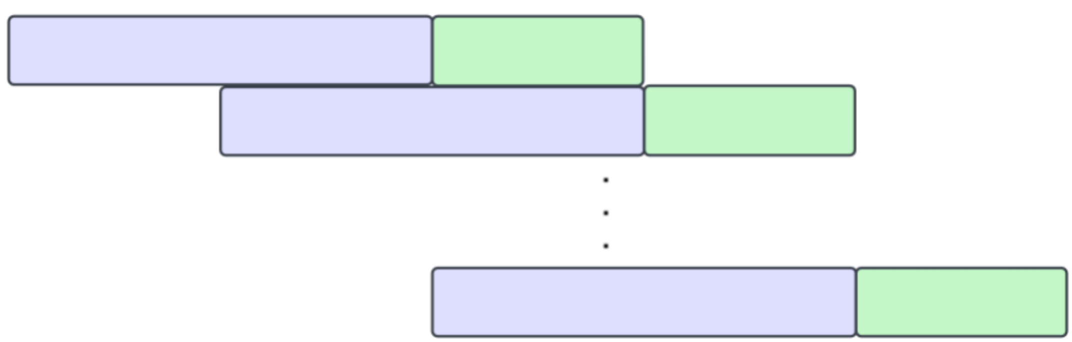

Figure 2: Period Splitting

randomly. Adhering to the concept that our portfolio is cointegrated, stocks were chosen, and cointegration was tested. If no cointegration relationship was found, an additional 10 assets were randomly selected until the hypothesis was satisfied. If, during the period, the hypothesis was rejected for all possible combinations, the portfolio's return for that period would be equal to zero, as no operations would be conducted.

3.2 Results

The empirical analysis compared the results generated by the cointegrated portfolio against a benchmark, the Bovespa index. It was found that the statistical arbitrage strategy achieved superior performance during the analyzed period. Table 2 presents the results for risk and return measures.

	Portfolio	**Benchmark**
Annualized Returns	23.1200	10.0900
Annualized Standard Deviation	21.0200	30.4700
Sharpe Ratio	1.1000	0.3300
Sortino Ratio	1.2500	0.4500
Annualized VaR 95%	-0.3100	-0.4600
Annualized CVaR 95%	-0.4900	-0.6900
Worst Drawdown	-0.3000	-0.8000

Table 2: Performance Metrics

According to the reported results, the portfolio's annualized return is much higher than the benchmark. One likely explanation for this exceptional performance is the strategy's capacity to create profits regardless of the economic condition, which leads to increased stability over time. This tendency is readily visible in figure 3, where the portfolio's development (blue line) contrasts with the benchmark's extensive period of lateralization (green line). The red and orange lines represent the portfolio's and benchmark's drawdowns, respectively.

Furthermore, the cointegrated portfolio exhibits lower volatility, with downturns being found to be less severe than the benchmark. This discovery implies that the portfolio has

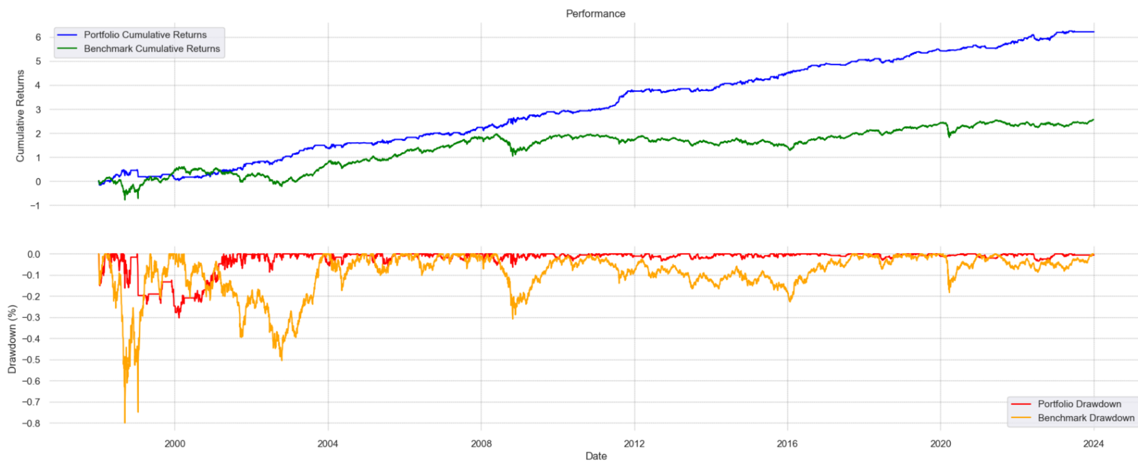

Figure 3: Performance

a faster recovery potential, which is critical for preserving consistency in results. These behavioral patterns demonstrate the strategy's robustness and efficiency.

The Sharpe and Sortino ratios contribute to this analysis since they reflect the link between expected return and risk, emphasizing the cointegrated portfolio's superiority. What is noteworthy here is not only the fact that the portfolio outperformed both indices, but also the observation that the Sortino ratio outperforms the Sharpe ratio. As a result, we underline that volatility during downturns, a negative factor, is lower than volatility overall, thus showing risk reduction.

Moreover, while having yearly VaR and CVaR values lower than the benchmark, our cointegrated portfolio has never experienced decreases at these levels. On the contrary, the Bovespa index fell by more than the value of the CVaR, indicating that the portfolio's performance is unaffected by market changes as a whole.

3.3 Robustness Test

A critical feature of a backtest is evaluating the outcomes acquired from factors, particularly when working with a statistical arbitrage approach. In this environment, portfolio returns must be independent of the causes underlying market behavior.

To execute a robustness test, portfolio returns must be regressed against these factors.

In this regard, I will use the Fama-French five-factor model, as indicated by Fama and French [2015]. The dependent variables in this are: RMRF (market return relative to the risk-free rate), SMB (small minus big), HML (high minus low), RMW (robust minus weak), and CMA (conservative minus aggressive).

The coefficients associated with each factor are estimated using the Ordinary Least Squares (OLS) method. The regression equation is expressed as follows:

$$R_{port} = 0.1573 \times RMRF - 0.0519 \times SMB + 0.0400 \times HML - 0.0491 \times RMW + 0.0175 \times CMA.$$

Table 3 presents the results of the relevant tests conducted in our analysis.

Test	**Test Statistic**	**P-Value**	**Conclusion**
R-squared	0.051	-	Low variability explanation
Jarque-Bera	10823.817	0.00	Non-normality of residuals
Omnibus	951.836	0.00	Low significance between parameters
Durbin-Watson	1.979	-	Possible autocorrelation in residuals

Table 3: Robustness Test Results

The analysis yields a R^2 near to zero, indicating that the factors evaluated cannot adequately explain the portfolio's performance. This low explained variability shows that the portfolio approach is genuinely market neutral. In the meantime, the Omnibus test shows that the explained variance is less than the unexplained variance, which indicates that the model parameters have minimal relevance, validating the notion that the included factors have little influence on the portfolio's performance. Finally, the Jarque-Bera test validates the residuals' non-normality. This lack of normality underscores the presence of patterns or behaviors not captured by the factors, emphasizing additional complexity in the portfolio returns.

4 CONCLUSION

The decision to use the statistical arbitrage technique to generate a cointegrated portfolio, rather than just pairs, was based on the Johansen's theoretical framework. This methodology enabled the execution of cointegration tests on several assets at the same time, supporting portfolio design based on the logic of trading through the spread, which represents price divergences from an equilibrium point.

The backtest findings show that arbitrage has the potential to earn rewards while posing no substantial risks. During the investigated time, the strategy produced great returns, outperforming the benchmark, while having lower drawdowns. The robustness test validated the independence of returns from market fluctuations, which is an expected characteristic of an arbitrage strategy.

A few words of caution are also in order. To begin, transaction costs were included in the backtest in addition to the steps taken to avoid statistical biases. Given the portfolio's short positions, these expenses can occasionally result in considerable losses, potentially negating the acquired return. My analysis revealed that even in this case, the cointegrated portfolio would outperform the market.

The pre-selection of assets is another unexplored element that could improve the results. Approaches such as Sarmento and Horta [2021], which uses unsupervised machine learning algorithms to discover stocks in clusters, are intriguing because they restrict the search, making it easier to locate assets with more common behaviors.

It is vital to note that structural breaks in the out-of-sample period may have an impact on the spread. As a result, improved spread modeling and forecasting can provide a clearer description of entry and exit points, eliminating arbitrary techniques. Meucci [2009] indicates that an Ornstein-Uhlenbeck process properly represents residuals, allowing for a more precise calculation of its mean and variance to produce a Z-score that better reflects price divergences. Other works, such as that of using the Markov Switch Regime Model to trade in different regimes, or by employing neural networks to predict when the spread returns to the mean, can be considered to improve the strategy.

All these suggestions can be incorporated into the same framework, providing a more sophisticated and concrete approach. Lastly, analyzing specific sub-periods could add more reliability to the presented results. Given the extensive database, it would be interesting to assess performance during significant events like the dot-com bubble, the 2008 financial crisis, and the COVID-19 pandemic.

REFERENCES

References

Rodrigo de Losso da Silveira Bueno. Econometria de séries temporais. 2018.

João Frois Caldeira and Gulherme Valle Moura. Seleção de uma carteira de pares de ações usando cointegração: Uma estratégia de arbitragem estatística. *Brazilian Review of Finance*, 11(1):49–80, 2013.

Rodolfo C Cavalcante, Rodrigo C Brasileiro, Victor LF Souza, Jarley P Nobrega, and Adriano LI Oliveira. Computational intelligence and financial markets: A survey and future directions. *Expert Systems with Applications*, 55:194–211, 2016.

Ernest P Chan. *Quantitative trading: how to build your own algorithmic trading business*. John Wiley & Sons, 2021.

Fernando AB Sabino da Silva, Flavio A Ziegelmann, and João F Caldeira. A pairs trading strategy based on mixed copulas. *The Quarterly Review of Economics and Finance*, 87: 16–34, 2023.

Richard V Diamond. Learning and trusting cointegration in statistical arbitrage. *Available at SSRN 2220092*, 2014.

Economática. EconomÁtica: Banco de dados. `http://www.economatica.com.br/index`$_f la.htm$.

Robert F Engle and Clive WJ Granger. Co-integration and error correction: representation, estimation, and testing. *Econometrica: journal of the Econometric Society*, pages 251–276, 1987.

Eugene F Fama. Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2):383–417, 1970.

Eugene F Fama and Kenneth R French. A five-factor asset pricing model. *Journal of financial economics*, 116(1):1–22, 2015.

Andrea Frazzini, Ronen Israel, and Tobias J Moskowitz. Trading costs. *Available at SSRN 3229719*, 2018.

Evan Gatev, William N Goetzmann, and K Geert Rouwenhorst. Pairs trading: Performance of a relative-value arbitrage rule. *The Review of Financial Studies*, 19(3):797–827, 2006.

Søren Johansen. Statistical analysis of cointegration vectors. *Journal of economic dynamics and control*, 12(2-3):231–254, 1988.

Søren Johansen. Estimation and hypothesis testing of cointegration vectors in gaussian vector autoregressive models. *Econometrica: journal of the Econometric Society*, pages 1551–1580, 1991.

Andrew W Lo and A Craig MacKinlay. Stock market prices do not follow random walks: Evidence from a simple specification test. *The review of financial studies*, 1(1):41–66, 1988.

Gangadharrao S Maddala and In-Moo Kim. Unit roots, cointegration, and structural change. 1998.

Attilio Meucci. Review of statistical arbitrage, cointegration, and multivariate ornstein-uhlenbeck. 2009.

Daniel P Palomar. Pair's trading. *Portfolio Optimization with R*, 2020.

Marcelo Scherer Perlin. Evaluation of pairs-trading strategy at the brazilian financial market. *Journal of Derivatives & Hedge Funds*, 15:122–136, 2009.

José Pedro Ramos-Requena, JE Trinidad-Segovia, and MA Sánchez-Granero. Introducing hurst exponent in pair trading. *Physica A: statistical mechanics and its applications*, 488:39–45, 2017.

Simão Moraes Sarmento and Nuno Horta. *A Machine Learning based Pairs Trading Investment Strategy*. Springer, 2021.

Stephen J Taylor. *Asset price dynamics, volatility, and prediction*. Princeton university press, 2011.

Ariel Amadeu Edwards Teixeira. *Pair trading in Bovespa with a quantitative approach: cointegration, Ornstein-Uhlenbeck equation and Kelly criterion.* PhD thesis, 2014.

Wenjun Xie, Zhao Zhi Toh, and Yuan Wu. Copula-based pairs trading in asia-pacific markets. , 24(4):1–17, 2016.

APPENDIX - PYTHON CODE

Import Packages

```
# Data Manipulation
import pandas as pd
import numpy as np
import random

# set display options
pd.options.display.float_format = "{:,.4f}".format
pd.set_option('display.max_columns', 100)

# Visualizaiton
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
sns.set(rc={'figure.figsize': (20, 8)})

# Cointegration test
from statsmodels.tsa.vector_ar.vecm import coint_johansen

# Robust analysis
import statsmodels.api as sm

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Data Manipulation

```python
# Load stock price data
AdjClose = pd.read_excel('StockPrice.xlsx', skiprows=[1], index_col=0, parse_dates=True)
AdjClose.index.name = 'date'
AdjClose.head()
```

```python
# Load Ibovespa market composition
MarketComposition = pd.read_excel("MarketComposition.xlsx")
MarketComposition.head()
```

```python
# Splitting in-sample and out-of-sample data
df = AdjClose.copy()

inSample = {}
outofSample = {}

update_date = pd.Timestamp('1998-01-01')
count = 1
while update_date <= pd.Timestamp('2024-01-01'):
    # In-sample is determined by market compostion date
    inSample_mask = (df.index > (update_date - pd.DateOffset(months=12))) & (df.index <= (update_date))
    inSample[count] = df.loc[inSample_mask]

    # Out-of-sample
    outofSample_mask = (df.index > (update_date)) & (df.index <= (update_date + pd.DateOffset(months=6)))
    outofSample[count] = df.loc[outofSample_mask]

```

```python
    # Update rule
    count += 1
    update_date += pd.DateOffset(months=6)

# In-sample tail
print("In-sample tail:\n", inSample[1].iloc[:, :5].tail())

# Out-of-sample head
print("Out-of-sample head:\n", outofSample[1].iloc[:, :5].head())
```

```python
# Keep only stocks from the Bovespa index
for index, date in enumerate(MarketComposition['Date'].unique().tolist()):
    # Select symbols from market compostion
    symbols = MarketComposition.loc[MarketComposition['Date'] == date, 'Symbols'].tolist()
    # Filter stocks and clean data
    inSample[index+1] = inSample[index+1][symbols].ffill()
    inSample[index+1] = inSample[index+1].dropna(axis=1, how='any')
    outofSample[index+1] = outofSample[index+1][inSample[index+1].columns].fillna(method='pad') # handle delisted stocks

# In-sample tail
print("In-sample tail:\n", inSample[1].iloc[:, :5].tail())

# Out-of-sample head
print("Out-of-sample head:\n", outofSample[1].iloc[:, :5].head())
```

Johansen Approach

```python
class Johansen:
    """
    This class performs the Johansen cointegration test and provides
    ↪ related calculations.

    Attributes:
    data : array-like
        Time series data for the cointegration test.
    det_order : int, optional
        Order of deterministic terms in the cointegration test.
    k_ar_diff : int, optional
        Number of lags to include in the cointegration test.
    ct : JohansenTestResult, optional
        Result of the cointegration test.
    """

    def __init__(self, data, det_order=0, k_ar_diff=1):
        """
        Initializes the Johansen class with the provided parameters.

        Parameters:
        data : array-like
            Time series data for the cointegration test.
        det_order : int, optional
            Order of deterministic terms in the cointegration test.
        k_ar_diff : int, optional
            Number of lags to include in the cointegration test.
        """
        self.data = data
```

```
        self.det_order = det_order
        self.k_ar_diff = k_ar_diff
        self.ct = coint_johansen(self.data,
         ↪  det_order=self.det_order, k_ar_diff=self.k_ar_diff)

    def n_coint_vectors(self):
        """
        Calculates the number of cointegrated vectors based on the
         ↪  cointegration test result.

        Returns:
        n_vectors : int
            Number of cointegrated vectors.
        """
        return len(np.where(self.ct.lr2 > self.ct.cvm[:, 1])[0])

    def beta(self):
        """
        Calculates the cointegrated eigenvectors (beta) based on the
         ↪  cointegration test result.

        Returns:
        beta : array-like
            Cointegrated eigenvectors.
        """
        return self.ct.evec[:, np.where(self.ct.lr2 > self.ct.cvm[:,
         ↪  1])[0]]

    def alpha(self):
        """
```

```python
        Calculates the estimated alpha matrix based on the cointegration test result.

        Returns:
        alpha : array-like
            Estimated alpha matrix.
        """
        r0t = np.transpose(self.ct.r0t)
        r1t = np.transpose(self.ct.rkt)

        S01 = np.dot(r0t, r1t.T) / r0t.shape[1]
        S11 = np.dot(r1t, r1t.T) / r0t.shape[1]

        beta = self.beta()
        return S01 @ beta @ np.linalg.inv(beta.T @ S11 @ beta)

    def Phi(self):
        """
        Calculates the Phi matrix based on the cointegration test result.

        Returns:
        Phi : array-like
            Phi matrix.
        """
        return np.dot(self.alpha(), self.beta().T)

    def residuals(self):
        """
        Calculates the residuals based on the cointegration test result.
```

```

        Returns:
        residuals : array-like
            Residuals.
        """
        return np.dot(self.Phi(), self.data.T)
```

Backtest Class

```
class Backtest:
    def __init__(self, inSample, outofSample, entry_point = 2,
    ↪ exit_point=0, stop_loss=None,det_order=0, k_ar_diff=1):
        """
        Initializes the Backtest class.

        Parameters:
        data : DataFrame
            DataFrame containing the financial data.
        det_order : int, optional
            Order of deterministic components in the cointegration
             ↪ test (default is 0).
        k_ar_diff : int, optional
            Number of lags to difference the data in the
             ↪ cointegration test (default is 1).
        """
        self.inSample = inSample
        self.outofSample = outofSample
        self.entry_point = entry_point
        self.exit_point = exit_point
        self.stop_loss = stop_loss
```

```python
        self.det_order = det_order
        self.k_ar_diff = k_ar_diff
        self.johansen = Johansen(inSample, det_order, k_ar_diff)
        self._hedge_ratio = self.hedge_ratio()

    def hedge_ratio(self):
        """
        Calculates the hedge ratio (Phi) based on the Johansen cointegration analysis.

        Returns:
        hedge_ratio : array-like
            Hedge Ratios.
        """
        HR = self.johansen.beta().T

        return HR / HR[0]

    def weight(self):
        """
        Calculates the weights for each asset in the portfolio.

        Returns:
        weights : array-like
            Portfolio weights.
        """

        return np.round(self._hedge_ratio / np.sum(self._hedge_ratio), 2)[0]

    def spread(self):
```

```python
        """
        Calculates the spread based on the hedge ratio and the financial data.

        Returns:
        spread : array-like
            Spread.
        """
        return np.nan_to_num(np.dot(self._hedge_ratio , self.outofSample.T))[0]

    def z_score(self):
        """
        Calculates the z-score of the spread.

        Returns:
        z_score : DataFrame
            Z-scores.
        """
        mean = np.mean(self.spread())
        std = np.std(self.spread())

        return (self.spread() - mean) / std

    def signal(self):
        """
        Generates trading signals based on the calculated z-score.

        Returns:
        signals : DataFrame
```

```
            DataFrame containing the generated trading signals (1: long, -1: short, 0: no signal).
        """
        position = 0
        signals = []
        z_score = self.z_score()

        # Loop through the data to generate signals
        for i in range(0, len(z_score)):
            t = z_score[i]

            # Generate signals based on z-score values
            # Entry points
            if t <= -self.entry_point and position == 0:
                position = 1  # open position
                signals.append(position)
            elif t >= self.entry_point and position == 0:
                position = -1  # open position
                signals.append(position)
            # Exit points
            elif t >= -self.exit_point and (position == 1 or position == 2):
                position = 0  # close position
                signals.append(position)
            elif t <= self.exit_point and (position == -1 or position == -2):
                position = 0  # close position
                signals.append(position)
            # Stop loss
            elif self.stop_loss is not None:
                if t <= -self.stop_loss and position == 1:
```

```python
                    position = 2  # close position
                    signals.append(position)
                elif t >= self.stop_loss and position == -1:
                    position = -2  # close position
                    signals.append(position)
                # Hold signal
                else:
                    signals.append(position)
            # Hold signal
            else:
                signals.append(position)

        return pd.DataFrame({"Signal": signals}, index=self.outofSample.index).shift(1)

    def port_rtn(self):
        """
        Calculates the daily returns of the portfolio based on the generated signals.

        Returns:
        port_returns : DataFrame
            DataFrame containing the calculated daily portfolio returns.
        """
        stock_rtn = np.log(self.outofSample).diff().fillna(0)
        port_rtn = []
        signal = self.signal()

        # Loop through the data to calculate portfolio returns
        for i in range(0, len(self.outofSample)):
```

```
            if signal.iloc[i, 0] == 1:
                rtn = np.dot(self.weight(), stock_rtn.iloc[i]) - 0.0003 # minus transaction costs
            elif signal.iloc[i, 0] == -1:
                rtn = - np.dot(self.weight(), stock_rtn.iloc[i]) - 0.0003 # minus transaction costs
            else:
                rtn = 0

            port_rtn.append(rtn)

        return pd.DataFrame({"Returns": port_rtn}, index=self.outofSample.index)
```

Performance Measures

```
class Performance:
    def __init__(self, portfolio, benchmark=None):
        """
        Initializes the Performance class with portfolio returns and an optional benchmark.

        Parameters:
        portfolio : pandas.Series
            Time series of portfolio returns.
        benchmark : pandas.Series, optional
            Time series of benchmark returns (default is None).
        """
        self.portfolio = portfolio
        self.benchmark = benchmark
```

```python

    def annual_rtn(self):
        """
        Calculates the annualized returns of the portfolio and
         ↪  benchmark.

        Returns:
        tuple
            Tuple containing the annualized portfolio returns and
             ↪  benchmark returns.
        """
        _portfolio = round(100 * (np.mean(self.portfolio) * 252), 2)
        _benchmark = None
        if self.benchmark is not None:
            _benchmark = round(100 * (np.mean(self.benchmark) *
             ↪  252), 2)
        return _portfolio, _benchmark

    def annual_std(self):
        """
        Calculates the annualized standard deviation of the
         ↪  portfolio and benchmark.

        Returns:
        tuple
            Tuple containing the annualized portfolio standard
             ↪  deviation and benchmark standard deviation.
        """
        _portfolio = round(100 * (np.std(self.portfolio) *
         ↪  np.sqrt(252)), 2)[0]
        _benchmark = None
```

```python
        if self.benchmark is not None:
            _benchmark = round(100 * (np.std(self.benchmark) * np.sqrt(252)), 2)[0]
        return _portfolio, _benchmark

    def sharpe_ratio(self):
        """
        Calculates the Sharpe ratio of the portfolio and benchmark.

        Returns:
        tuple
            Tuple containing the Sharpe ratio of the portfolio and benchmark.
        """
        _portfolio = round(self.annual_rtn()[0] / self.annual_std()[0], 2)
        _benchmark = None
        if self.benchmark is not None:
            _benchmark = round(self.annual_rtn()[1] / self.annual_std()[1], 2)
        return _portfolio, _benchmark

    def sortino_ratio(self):
        """
        Calculates the Sortino ratio of the portfolio and benchmark.

        Returns:
        tuple
            Tuple containing the Sortino ratio of the portfolio and benchmark.
        """
```

```python
        _portfolio_downside = round(100 * (np.std(self.portfolio[self.portfolio < 0]) * np.sqrt(252)), 2)
        _portfolio = round(self.annual_rtn()[0] / _portfolio_downside, 2)[0]
        _benchmark = None
        if self.benchmark is not None:
            _benchmark_downside = round(100 * (np.std(self.benchmark[self.benchmark < 0]) * np.sqrt(252)), 2)
            _benchmark = round(self.annual_rtn()[1] / _benchmark_downside, 2)[0]
        return _portfolio, _benchmark

    def annual_VaR(self):
        """
        Calculates the annualized Value at Risk (VaR) of the portfolio and benchmark.

        Returns:
        tuple
            Tuple containing the annualized portfolio VaR and benchmark VaR.
        """
        _portfolio = round(self.portfolio.quantile(0.05) * np.sqrt(252), 2)[0]
        _benchmark = None
        if self.benchmark is not None:
            _benchmark = round(self.benchmark.quantile(0.05) * np.sqrt(252), 2)[0]
        return _portfolio, _benchmark
```

```python

    def annual_CVaR(self):
        """
        Calculates the annualized Conditional Value at Risk (CVaR) of the portfolio and benchmark.

        Returns:
        tuple
            Tuple containing the annualized portfolio CVaR and benchmark CVaR.
        """
        _portfolio = round(self.portfolio[self.portfolio < self.portfolio.quantile(0.05)].mean() * np.sqrt(252), 2)[0]
        _benchmark = None
        if self.benchmark is not None:
            _benchmark =  round(self.benchmark[self.benchmark < self.benchmark.quantile(0.05)].mean() * np.sqrt(252), 2)[0]
        return _portfolio, _benchmark

    def worst_drawdown(self):
        """
        Calculates the worst drawdown of the portfolio and benchmark.

        Returns:
        tuple
            Tuple containing the worst drawdown of the portfolio and benchmark.
        """
```

```
        _portfolio_cum_rtn = self.portfolio.cumsum()
        _portfolio_max_rtn = _portfolio_cum_rtn.cummax()
        _portfolio_drawdown = (_portfolio_cum_rtn - _portfolio_max_rtn) / (1 + _portfolio_max_rtn)
        _portfolio_worst_drawdown = round(_portfolio_drawdown.min(), 2)[0]

        _benchmark_worst_drawdown = None
        if self.benchmark is not None:
            _benchmark_cum_rtn = self.benchmark.cumsum()
            _benchmark_max_rtn = _benchmark_cum_rtn.cummax()
            _benchmark_drawdown = (_benchmark_cum_rtn - _benchmark_max_rtn) / (1 + _benchmark_max_rtn)
            _benchmark_worst_drawdown = round(_benchmark_drawdown.min(), 2)[0]

        return _portfolio_worst_drawdown, _benchmark_worst_drawdown

    def plot_performance(self):
        """
        Plots the cumulative returns and drawdown of the portfolio and benchmark.
        """
        _portfolio_cum_rtn = self.portfolio.cumsum()
        _portfolio_max_rtn = _portfolio_cum_rtn.cummax()
        _portfolio_drawdown = (_portfolio_cum_rtn - _portfolio_max_rtn) / (1 + _portfolio_max_rtn)

        fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

        # Plot cumulative returns for portfolio
```

```python
        ax1.plot(_portfolio_cum_rtn.index, _portfolio_cum_rtn, label='Portfolio Cumulative Returns', color='blue')

        # Plot cumulative returns for benchmark if available
        if self.benchmark is not None:
            _benchmark_cum_rtn = self.benchmark.cumsum()
            ax1.plot(_benchmark_cum_rtn.index, _benchmark_cum_rtn, label='Benchmark Cumulative Returns', color='green')

        ax1.set_ylabel('Cumulative Returns')
        ax1.legend()

        # Plot drawdown for portfolio
        ax2.plot(_portfolio_drawdown.index, _portfolio_drawdown.values, color='red', label='Portfolio Drawdown')

        # Plot drawdown for benchmark if available
        if self.benchmark is not None:
            _benchmark_max_rtn = _benchmark_cum_rtn.cummax()
            _benchmark_drawdown = (_benchmark_cum_rtn - _benchmark_max_rtn) / (1 + _benchmark_max_rtn)
            ax2.plot(_benchmark_drawdown.index, _benchmark_drawdown.values, color='orange', label='Benchmark Drawdown')

        ax2.set_ylabel('Drawdown (%)')
        ax2.legend()

        # Set x-axis label only on the bottom subplot
        ax2.set_xlabel('Date')
```

```

        # Customize background
        fig.set_facecolor('white')
        ax1.set_facecolor('white')
        ax2.set_facecolor('white')
        ax1.grid(color='black', linestyle='--', linewidth=0.2)
        ax2.grid(color='black', linestyle='--', linewidth=0.2)

        plt.show()


    def summary_table(self):
        """
        Generates a summary table with various performance metrics of the portfolio and benchmark.

        Returns:
        pandas.DataFrame
            DataFrame containing the summary metrics.
        """
        metrics = pd.DataFrame({
            'portfolio': {'Annualized Returns': self.annual_rtn()[0],
                          'Annualized Standard Deviation': self.annual_std()[0],
                          'Sharpe Ratio': self.sharpe_ratio()[0],
                          'Sortino Ratio': self.sortino_ratio()[0],
                          'Annualized VaR 95%': self.annual_VaR()[0],
                          'Annualized CVaR 95%': self.annual_CVaR()[0],
```

```
                                'Worst Drawdown': self.worst_drawdown()[0]},
                'benchmark': {'Annualized Returns': self.annual_rtn()[1],
                                'Annualized Standard Deviation': self.annual_std()[1],
                                'Sharpe Ratio': self.sharpe_ratio()[1],
                                'Sortino Ratio': self.sortino_ratio()[1],
                                'Annualized VaR 95%': self.annual_VaR()[1],
                                'Annualized CVaR 95%': self.annual_CVaR()[1],
                                'Worst Drawdown': self.worst_drawdown()[1]}
                })

        return metrics
```

Trading Strategy Execution

```
# Define an empty dictionary to store the results
strategy = {}
portfolio = {}

for i in range(1, len(inSample)):
    # Set seed
    random.seed(2024)

    cointegrated = False

```

```python
    while not cointegrated:
        # Select stocks randomly
        selected_columns = random.sample(list(inSample[i].columns), 10)
        _inSample = inSample[i][selected_columns]
        _outofSample = outofSample[i][selected_columns]

        # Test for cointegration
        coint = Johansen(_inSample).n_coint_vectors()

        if coint >= 1:
            # Run backtest
            backtest = Backtest(inSample=_inSample, outofSample=_outofSample, entry_point=1, stop_loss=3)
            strategy[i] = pd.DataFrame({"Z-score": backtest.z_score()}, index=_outofSample.index)
            portfolio[i] = backtest.port_rtn()
            cointegrated = True
        else:
            # Set backtest to 0 or handle as needed
            portfolio[i] = pd.DataFrame([0] * len(_outofSample), index=_outofSample.index)

# concatenate all DataFrames into a single DataFrame
strategy = pd.concat(strategy.values(), axis=0)
portfolio = pd.concat(portfolio.values(), axis=0)

# Plot z-score and signals
# Plotting
fig, ax = plt.subplots(figsize=(10, 5))
```

```
ax.plot(strategy.index, strategy.iloc[:,0], label='Z-score', color='blue', linewidth=1)
ax.axhline(y=0, color='black', linestyle='--', linewidth=2, label='ExitPoint')
ax.axhline(y=-1, color='green', linestyle='--', linewidth=2, label='Entry Point')
ax.axhline(y=1, color='green', linestyle='--',linewidth=2)
ax.axhline(y=-3, color='red', linestyle='--', linewidth=2, label='Stop Loss')
ax.axhline(y=3, color='red', linestyle='--', linewidth=2)
ax.set_ylabel('Values')
ax.set_xlabel('Date')
ax.set_facecolor('white')
ax.grid(color='black', linestyle='--', linewidth=0.3)
ax.set_title("Trading Strategy")
ax.legend()
plt.show()
```

```
# Load Ibovespa index
ibov = pd.read_excel('Ibovespa.xlsx', skiprows=1, index_col=0, parse_dates=True).dropna()
ibov.index.name = 'date'

# Set as benchmark return
benchmark = pd.DataFrame({'Benchmark': np.log(ibov['IBOV']).diff().dropna()})
benchmark = benchmark.loc[portfolio.index[0]:portfolio.index[-1]]
benchmark.head()
```

```
performance = Performance(portfolio=portfolio, benchmark=benchmark)
performance.summary_table()
```

```
performance.plot_performance()
```

```
# Load Fama-French factors
ff_factors = pd.read_csv('BR-Fama-French-Factors/FF_Factors.csv',
    index_col=0, parse_dates=True)
ff_factors = ff_factors.loc[portfolio.index[0]:portfolio.index[-1],
    ['RMRF', 'SMB', 'HML', 'RMW', 'CMA']]
ff_factors.head()
```

```
# Merge portfolio and ff_factors
df = pd.merge(portfolio, ff_factors, left_index=True,
    right_index=True)
df.tail()
```

```
# Perform regression
X = df.iloc[:,1:]
y = df.iloc[:,0]
model = sm.OLS(y, X).fit()

# Display the results
model.summary()
```