Final Project

Pairs Trading Strategy Design & Backtest

(TS)

Heidi Pang

heidi.pang92@gmail.com - : heidi123123

Project lead by Dr. Richard Diamond

CQF Institute

August 22, 2024

Contents

1 Introduction

This work concludes my certification with the CQF Institute. The Final Project topic *Pairs Trading Strategy Design & Backtest* discusses methods to identify cointegration between assets and how to exploit this relationship in a trading strategy. The strategy is based on the assumption that the spread between cointegrated assets is a stationary process following a mean-reversion process, stochastically modeled by an Ornstein-Uhlenbeck (OU) process.
Before diving into the trading discussion, Chapter 2 introduces the mathematical foundations relevant to cointegration analysis, including the Engle-Granger cointegration analysis procedure. All concepts will also be illustrated numerically using Python and applied to a real example where I analyze the cointegration between the US shares of PepsiCo (ticker PEP) and The Coca-Cola Company (ticker KO) in Chapter 3.
Subsequently, we will explore an interesting trading opportunity that presents itself in the context of cointegration. The presented strategy bets on a mean-reverting residual spread and takes offsetting positions in the respective asset pairs under a certain hedge ratio, therefore essentially trading the spread with long and short positions. The strategy will be tested using a training/testing data set approach inspired by Machine Learning methods. Here, only part of the data (training data) is used to determine the trading strategy parameters, while performance testing is conducted on the test data set. The data split follows a chronological order, ensuring that the training data *precedes* the test data, simulating a real-world scenario where an investor only has access to past data when making trading decisions.

1.1 Historical Background

Stochastic time series analysis has long been of interest in various fields of research. However, significant research into cointegration only began in the 1980s, with key contributions from Granger in 1981 [8], Engle & Granger in 1987 [7], and Johansen in 1988 [10]. In 1990, McDermott [12] was one of the first to describe how the concept could be applied economically in a pairs trading strategy. Although individual stocks often exhibit random walk behavior that is challenging to forecast, pairs trading on cointegrated assets leverages the assumption that certain stock pairs exhibit co-movement, i.e. loosely speaking cointegrated pairs "move together" to a certain extent.
More recent works by researchers and practitioners including Diamond in 2013 [3], Huck & Afawubo in 2015 [9], Caldeira & Moura in 2013 [2] confirm that this approach remains highly relevant and profitable, even in distressed or high-volatility markets, such as cryptocurrencies (e.g., Nair, 2021 [13]).

1.2 Tabular Overview of Applied Methods

Mathematical Concept	Description
Ordinary least squares (OLS) regression	First step for the Engle-Granger cointegration test, where OLS regression is used to estimate the long-term relationship between two time series. The residuals from this regression are used in the following steps.
Augmented Dickey-Fuller (ADF) test	Statistical test used to determine if a time series is stationary by testing for the presence of a unit root. Implemented using the `adfuller` function from the `statsmodels` library.
Equilibrium correction model (ECM)	Model used in Engle-Granger step 2 to capture short-term deviations from the long-term equilibrium μ_e.
Cointegration analysis (Engle-Granger 2-step method)	Uses the above methods to statistically assess the long-term relationship between two time series (`ticker1` and `ticker2`).
Ornstein-Uhlenbeck (OU) process	A stochastic process frequently used to model mean-reverting behavior. OU parameters are estimated via Maximum Likelihood Estimation (MLE) using the `minimize` function from `scipy.optimize`. Numerical optimization can converge to local minima and is dependent on initial parameters.
Pairs trading strategy with PnL calculation	Position management class with calculation of realized and unrealized Profit and Loss (PnL) based on position sizes and price movements. The PnL is calculated for each day and accumulated over time.
Comparison with return of benchmark index	The performance of the cointegration pairs trading portfolio is measured against a passive investment strategy where a benchmark index is held over the same time period. For comparison purposes, the initial index investment is scaled to match the initial portfolio investment value.

Table 1.1: Overview of mathematical concepts and their implementations

2 Applied Methodology

Cointegration is a long-term statistical property that applies to a collection of time-dependent random variables $(X_1, ..., X_k)$. For this work, we will focus on pairwise cointegrated time series X_t and Y_t. To establish a foundation for the further study of mathematical concepts, we will specify some key definitions and notations in the following. Alongside the mathematical framework, I will also present potential numerical implementations for each concept discussed starting in Section 2.2. The code snippets shown will be applied to a real-world example using time series data for the prices of The Coca-Cola Company (KO) and PepsiCo (PEP) in Chapter 3. Some of the more routine implementation details are omitted for brevity. For the full implementation, please refer to the Jupyter notebook or Appendix A.

2.1 Definitions

Let X_t, Y_t be two time series.

Definition 2.1.1 X_t is called a *stationary* stochastic process if the joint probability distribution does not change when shifted in time, i.e.

$$P\left(X_1 = x_1, ..., X_{t_n} = x_{t_n}\right) = P\left(X_{t_1+\tau} = x_{t_1}, ..., X_{t_n+\tau} = x_{t_n}\right), \forall \text{ times } t_i, \tau.$$

One could also say that the joint cumulative distribution function of X_t is independent of time.

It immediately follows that X_t must define a *mean-reverting* process, i.e., deviations from the long-term equilibrium mean will tend to revert back to it over time.

Definition 2.1.2 X_t and Y_t are called *cointegrated of order* $d = 1$, if we can write

$$Y_t = \beta \cdot X_t + e_t$$

for a stationary process e_t.

Order of integration $I(d)$ is a concept in statistics, where d denotes the number of times a time series needs to be differenced in order for it to become stationary. If not otherwise stated, all cointegrations considered in this work will be of order 1.
However, for the context of financial time series and the numerical methods implemented in the following, it will be more practical to stick to the following (equivalent) definition of cointegration:

Definition 2.1.3 X_t and Y_t are called *cointegrated of order* $d = 1$, if we can write $Y_t = \beta_0 + \beta_1 \times X_t + e_t$ for a stationary process e_t.

- β_0 as the intercept describes the part of the time series of Y_t that is independent of X_t.
- β_1 models the linear dependency between X_t and Y_t in the long-term equilibrium. In practice, this can be used as a *hedge ratio*, because for every unit price change in X_t, Y_t prices will increase β_1.
- e_t are the residuals obtained through an ordinary least squares regression from Y_t onto X_t. In the context of cointegration, they must be stationary.

2.2 Ordinary Least Squares

In the context of linear regression, one seeks the coefficients $\vec{\beta} \in \mathbb{R}^p$ to satisfy

$$\vec{Y} = \mathbf{X}\vec{\beta} + \vec{e}, \tag{2.1}$$

where $\vec{Y} \in \mathbb{R}^n$ denotes the dependent variables (observed), $\vec{e} \in \mathbb{R}^n$ denotes the vector of residuals and the linear equation matrix

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} = \begin{bmatrix} 1 & x_{12} & \cdots & x_{1p} \\ 1 & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n2} & \cdots & x_{np} \end{bmatrix} \in \mathbb{R}^{n \times p},$$

where the first column with 1s represents the intercept term. So only the other $p - 1$ columns represent actual regressors.
In an over-determined system with more equations than regressors, i.e. $n > p$, there usually exists no exact solution, so Ordinary Least Squares (OLS) attempts to find the best possible solution. "Best" is defined as minimal with respect to sum of squares of the residuals $\vec{e}$, i.e. the OLS approach solves a minimization problem to solve for

$$\hat{\beta} = \arg\min_{\beta} \left|\left|\vec{Y} - \mathbf{X}\vec{\beta}\right|\right| = \arg\min_{\beta} \sum_{t=1}^{n} \left(Y_t - \sum_{k=1}^{n} x_{tk}\beta_k \right)^2. \tag{2.2}$$

This minimization problem can be rewritten into

$$\begin{aligned} \mathbf{X}^T\mathbf{X}\hat{\beta} &= \mathbf{X}^T\vec{Y} \\ \Leftrightarrow \hat{\beta} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\vec{Y} \in \mathbb{R}^p. \end{aligned}$$

Now this has become a $p \times p$-dimensional square system of equations with a unique solution $\hat{\beta}$. The residuals are given as $\vec{e} = \vec{Y} - \mathbf{X}\vec{\beta}$.
Remark 2.2.1 In the following, I will drop the arrows for the vectors. Matrix algebra still applies in a multi-dimensional setting.

2.2.1 Implementation

```python
import numpy as np
import pandas as pd


def least_squares_regression(y, X):
    """Perform least squares regression to obtain beta coefficients and residuals."""
    X = np.hstack([np.ones((X.shape[0], 1)), X])  # add y-intercept to X
    beta = np.linalg.inv(X.T @ X) @ (X.T @ y)  # least squares regression for beta
    residuals = y - X @ beta
    return beta, residuals


# Perform ordinary least square (OLS) regression
y = train_data[ticker1].values
X = train_data[ticker2].values.reshape(-1, 1)
beta, residuals = least_squares_regression(y, X)
```

Note that some prior data handling and downloading needs to be executed in the above code snippet before the provided implementation can run. The full code can be found in Appendix A.

2.3 ADF Test

The Augmented Dickey-Fuller (ADF) Test is a statistical test used to assess the stationarity of a time series e_t. There are different variations of ADF tests (with or without trend, with or without constant). For brevity, we will only explain the implemented ADF test with constant but without trend.
The null hypothesis H_0 is that the tested time series e_t has a unit root, i.e. it is non-stationary. The underlying model follows a differenced auto-regressive model with p lags $AR(p)$

$$\Delta e_t = \alpha + \gamma e_{t-1} + \delta_1 \Delta e_{t-1} + \cdots + \delta_p \Delta e_{t-p} + u_t, \tag{2.3}$$

where $\Delta e_t = e_t - e_{t-1}$ denotes the first differences of the time series considered, $\alpha, \gamma, \delta_i \in \mathbb{R}^2$ are model coefficients and u_t denotes the error term.
The null hypothesis of testing for a unit root then becomes testing for $\gamma = 0$ which would result in a random walk which is of course non-stationary, as discussed in [4].

Similar to other statistical tests, the ADF test produces a test statistic that is compared to a predefined critical value to decide whether to reject or fail to reject H_0. Equivalently, H_0 can also be rejected upon examining the p-value: If the p-value is less than a predefined significance level (e.g. 5%), H_0 should be rejected and the tested time series is stationary. The following table summarizes some frequently used critical values for the ADF test statistic:

Significance Level	**1%**	**5%**	**10%**
ADF Test Statistic	-3.434	-2.863	-2.568

Table 2.1: Critical values for the ADF test at 1%, 5%, and 10% significance levels.

The optimal lag p can be determined by comparing the Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) for a different number of lags. The lowest value of these statistics gives the optimal number of lags.

2.3.1 Implementation

I chose to use the `statsmodels` [1] implementation of the ADF test. In my opinion, it is more important to be able to understand and interpret the returned results correctly, rather than calculating the actual test statistics and p-values myself. The `adfuller` function available in the `statsmodels` library uses the AIC to find the best number of lags (if not otherwise specified with an explicit `maxlag` input argument). I chose to also include `maxlag` in my function as an optional argument.

```python
from statsmodels.tsa.stattools import adfuller


def perform_adf_test(residuals, significance_level, maxlag=None):
    """Perform the Augmented Dickey-Fuller (ADF) test to check for the presence of a unit root in a time series.
    H0: time series has a unit root (i.e. non-stationary)"""
    adf_test = adfuller(residuals, maxlag=maxlag, autolag=None)
    # autolag=None will set lag nbr to maxlag (no lag optimization) if maxlag is not None
    # if maxlag and autolag are both None, then lag nbr will be optimized using AIC
    adf_statistic, p_value, lags = adf_test[0], adf_test[1], adf_test[2]

    print(f"ADF Statistic: {adf_statistic:.4f}")
    print(f"p-value: {p_value:.4f}")
    print(f"Number of lags: {lags}")

    if p_value < significance_level:
        print(f"The residuals are stationary (reject null hypothesis) "
              f"at the {significance_level * 100}% significance level.")
    else:
        print(f"The residuals are not stationary (fail to reject null hypothesis) "
              f"at the {significance_level * 100}% significance level.")
    return adf_test


adf_test_result = perform_adf_test(train_data['residuals'], significance_level=0.05, maxlag=1)
```

Output:

```
ADF Statistic: -4.1409
p-value: 0.0008
Number of lags: 1
The residuals are stationary (reject null hypothesis) at the 5.0% significance
level.
```

2.4 Error Correction Model

The Error Correction Model (ECM), also known as the Equilibrium Correction Model, relates the short-term dynamics of a time series to its long-term equilibrium. If two time

series X_t and Y_t are sufficiently cointegrated, their residual process $e_t = Y_t - \beta_0 - \beta_1 X_t$ will have a meaningful long-term equilibrium and will mean-revert around that equilibrium mean μ_e.
Let $\Delta X_t = X_t - X_{t-1}$ and $\Delta Y_t = Y_t - Y_{t-1}$ denote first differences or absolute returns of the respective time series. Considering such differenced processes removes long-term continuous effects and is more suitable to analyze short-term dynamics. The ECM can then be expressed as

$$\Delta Y_t = \alpha_0 + \alpha_1 \cdot \Delta X_t + \alpha_2 \cdot e_{t-1} + \varepsilon_t. \tag{2.4}$$

The coefficient vector $\alpha = (\alpha_0, \alpha_1, \alpha_2)$ is once again estimated by OLS in the ECM regression. ε_t denotes the residuals of this second OLS.
From (2.4), we see that ECM captures how the dependent variable Y_t responds to short-term changes in the regressor variable X_t. In particular, ECMs allow for the estimation of the speed of error correction from the long-term equilibrium μ_e by examining the coefficient α_2 of the lagged residuals e_t. Note that the lagged residuals considered should be shifted with the lag size used in the ADF test of Engle-Granger step 1. This coefficient is expected to be negative but close to zero. The negative sign shows that deviations from μ_e will be "pulled back" to mean-revert over time. The magnitude of α_2 indicate the actual speed of mean-reversion.

2.4.1 Implementation

```python
import pandas as pd


def get_differences(data, columns):
    """Calculate the returns (differences) Delta y_t = y_t-y_{t-1} for
       ↪ the specified columns in the dataframe."""
    return data[columns].diff().dropna()


def fit_ecm(data, target_column, independent_column, lag_size=1):
    """Step2 of the Engle-Granger procedure: fit the Equilibrium
       ↪ Correction Model (ECM)."""
    data_delta = get_differences(data, [target_column,
       ↪ independent_column])
    data_delta['lagged_residuals'] = data['residuals'].shift(lag_size)
       ↪ # lag the residuals
    data_delta = data_delta.dropna()

    # OLS to obtain ECM coefficients & residuals
    y = data_delta[target_column].values
    X = data_delta[[independent_column, "lagged_residuals"]].values
    ecm_coefficients, ecm_residuals = least_squares_regression(y, X)

    ecm_residuals = pd.DataFrame(ecm_residuals, index=data_delta.index,
       ↪ columns=["ECM_residuals"])  # convert to pd.df
    return {'coefficients': ecm_coefficients, 'residuals': ecm_residuals
       ↪ }


# Example call: fit the ECM for KO and PEP
ecm_results = fit_ecm(train_data, "KO", "PEP")
```

2.5 Engle-Granger Procedure

We have now discussed all the building blocks required for the Engle-Granger procedure, a traditional two-step approach for analyzing cointegration between two assets.

1. **Residual analysis between price time series X_t, Y_t:**

   [i] Perform a linear regression of Y_t onto X_t to model their long-term relationship using OLS. The regression equation is: $Y_t = \beta_0 + \beta_1 X_t + e_t$, where e_t is the residual.

   [ii] If X_t and Y_t are cointegrated, then the residuals e_t must be stationary. Hence, we conduct the ADF test to check for stationarity of $e_t = Y_t - \beta_0 - \beta_1 X_t$. If the p-value is below a predefined significance level, e.g. 5%, we reject the null hypothesis H_0 of a unit root and conclude that the residuals are a stationary process. Consequently, X_t and Y_t are cointegrated at this significance level, e.g. 5%.

2. **Error Correction Model (ECM)**: The ECM follows (2.4), where e_{t-1} denotes the lagged residuals from the previous regression, ε_t denotes the "new" residual terms, inferred from regressing the ECM. The coefficient vector $\alpha = (\alpha_0, \alpha_1, \alpha_2)$ is estimated by OLS in the ECM regression. Of particular interest is α_2, the error correction term coefficient of the residual term e_{t-1}, which represents the speed of adjustment towards the long-term equilibrium.

Remark 2.5.1 There exist also other methods to test for cointegration relationships, notably the Johansen test [11]. One advantage of the Johansen test is its ability to identify cointegrated relationships among more than two assets simultaneously using underlying $VAR(p)$ models and vector error correction models (VECM).

2.5.1 Implementation

Step 2 of Engle-Granger procedure has been implemented in Section 2.4.1. All the methods applied in step 1 are also prepared. We only need to combine all the steps accurately now:

```python
def perform_engle_granger_step1(ticker1, ticker2, index_ticker, train_data, test_data,
                                plotting, significance_level, maxlag):
    """Step1 of the Engle-Granger procedure."""

    # OLS regression to obtain regression coefficients beta & residuals
    y = train_data[ticker1].values
    X = train_data[ticker2].values.reshape(-1, 1)
    beta, residuals = least_squares_regression(y, X)
    train_data['residuals'] = residuals
    test_data['residuals'] = calculate_test_residuals(test_data, beta, ticker1, ticker2)

    if plotting:  # plot normalized asset prices and residuals - code omitted here
        plot_assets_and_residuals(train_data, test_data, ticker1, ticker2, index_ticker)

    # perform ADF test
    adf_test_result = perform_adf_test(train_data['residuals'], significance_level, maxlag)
    return train_data, test_data, beta, adf_test_result
```

```


# train 70% & test 30% split to backtest our estimated parameters in
    ↪ trading strategy later
# function omitted here - full code in Jupyter notebook
train_data, test_data = split_data(data, split_ratio=0.7)

# Engle-Granger procedure - Step 1
train_data, test_data, beta, adf_test_result =
    ↪ perform_engle_granger_step1(
                        ticker1, ticker2, index_ticker,
                        train_data, test_data, plotting,
                        significance_level, maxlag=None)
# Engle-Granger procedure - Step 2: ECM
lag_size = adf_test_result[2]
ecm_results = fit_ecm(train_data, ticker1, ticker2, lag_size)
```

2.6 Ornstein-Uhlenbeck Process

In financial markets, the residuals e_t between two cointegrated assets tend to revert to a long-term equilibrium μ_e, rather than drifting apart indefinitely. This mean-reverting behavior can be modeled through an Ornstein-Uhlenbeck (OU) process defined by

$$de_t = -\theta \left(e_t - \mu_e\right) dt + \sigma_{OU} dW_t, \tag{2.5}$$

where $\theta > 0$ is the speed of mean-reversion to the long-term equilibrium mean μ_e, $\sigma_{OU} > 0$ is the standard deviation of the process, and W_t denotes a standard Brownian motion process. In discrete time, the process can be written as

$$e_t = e_{t-1} - \theta \left(e_{t-1} - \mu_e\right) \cdot \delta t + \sigma_{OU} \sqrt{\delta t} \cdot Z_t, \tag{2.6}$$

where $Z_t \sim \mathcal{N}(0, 1)$ and δt denotes the discrete time step.
While μ_e, σ_{OU} could be estimated from the residuals time series e_t, estimating the parameter θ requires some more work. The implemented option estimates all 3 Ornstein-Uhlenbeck parameters $(\theta, \mu_e, \sigma_{OU})$ using maximum likelihood estimation, described in the following.

2.6.1 Maximum Likelihood Estimation for an OU Process

Maximum Likelihood Estimation (MLE) is a popular statistical method to estimate the parameters of an assumed probability model by maximizing the likelihood function. In other words, MLE seeks to find the set of parameters $\hat{\phi} = (\hat{\phi}_1, \ldots, \hat{\phi}_n)$ that make the observed data most probable under the assumed model. Note that maximizing likelihood is equivalent to minimizing the negative log-likelihood, since logarithms are monotonically increasing functions.
In the context of the OU process, MLE is applied to estimate the parameters $\hat{\theta}$, $\hat{\mu}_e$, $\hat{\sigma}_{OU}$ that best describe the mean-reverting behavior of the residuals e_t between two cointegrated assets. Let $\mathcal{L}(\theta, \mu_e, \sigma_{OU} | e_0, ..., e_N)$ denote the likelihood function for the OU parameters $\theta, \mu_e, \sigma_{OU}$ given residuals $e_0, ..., e_N$. Then, we are interested to find

$$\begin{aligned}
(\hat{\theta}, \hat{\mu}_e, \hat{\sigma}_{OU}) &= \arg\max_{\theta, \mu_e, \sigma_{OU}} \mathcal{L}(\theta, \mu_e, \sigma_{OU} | e_0, ..., e_N) \\
&= -\arg\min_{\theta, \mu_e, \sigma_{OU}} \ln\left(\mathcal{L}(\theta, \mu_e, \sigma_{OU} | e_0, ..., e_N)\right)
\end{aligned}$$

Theorem 2.6.1 The likelihood function is defined as:

$$\mathcal{L}(\theta, \mu_e, \sigma_{OU} | e_0, ..., e_N) = \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi\sigma_{OU}^2 \delta t}} \exp\left(-\frac{(e_t - (e_{t-1} + \theta(\mu_e - e_{t-1})\delta t))^2}{2\sigma_{OU}^2 \delta t}\right),$$

where N is the total number of observations.

Notation: In the following, e_t denotes the time series of residuals between the cointegrated asset pair X_t, Y_t. $\exp(t)$ denotes the exponential function, i.e., the single letter e shall *not* denote the mathematical constant $e \approx 2.71828$.

Proof. We begin by deriving the mean and variance of the residuals in the OU process from the defining SDE (2.5).

- $\mathbb{E}[e_t|e_0]$: Since the stochastic component of the SDE (2.5) has mean 0, we can focus on the simplified SDE including only the deterministic part of (2.5):

$$de_t = -\theta \left(e_t - \mu_e\right) dt. \tag{2.7}$$

By separation of variables and plugging in initial values, we obtain the solution

$$e_t = \mu_e + \exp(-\theta t) \left(e_0 - \mu_e\right). \tag{2.8}$$

It follows that $\mathbb{E}[e_t|e_0] = \mu_e + \exp(-\theta t)\left(e_0 - \mu_e\right)$.

- **Var**$[e_t|e_0]$: Next, we consider the full solution to the SDE (2.5) derived for the multi-dimensional case in [3]:

$$e_t = \mu_e + \exp(-\theta t) \left(e_0 - \mu_e\right) + \sigma_{OU} \int_0^t e^{-\theta(t-s)} dW_s.$$

The variance of e_t given e_0 is then derived from the stochastic integral:

$$\mathrm{Var}[e_t|e_0] = \mathrm{Var}\left[\sigma_{OU} \int_0^t e^{-\theta(t-s)} dW_s\right] = \sigma_{OU}^2 \mathrm{Var}\left[\int_0^t e^{-\theta(t-s)} dW_s\right]$$

By Itô isometry, this variance can be written as

$$\mathrm{Var}[e_t|e_0] = \sigma_{OU}^2 \int_0^t e^{-2\theta(t-s)} ds = \frac{\sigma_{OU}^2}{2\theta}\left(1 - e^{-2\theta t}\right),$$

where the last equality is obtained from standard calculus.

We now transition from the continuous-time process to the discrete-time process with small time steps δt. The distribution for $(e_t|e_{t-1}) = (e_t|e_{t-\delta t})$ can be approximated as Gaussian with mean and variance derived above. Recalling that for small δt: $\exp(-\theta \delta t) \approx 1 - \theta \delta t$, it follows that

$$e_t | e_{t-1} \sim \mathcal{N}\left(e_{t-1} + \theta(\mu_e - e_{t-1})\delta t, \sigma_{OU}^2 \delta t\right).$$

Finally, the likelihood function $\mathcal{L}(\theta, \mu_e, \sigma_{OU}|e_0, \ldots, e_N)$ is the joint probability of observing the exact sequence $e_0, \ldots, e_N$ of residuals. Since the OU process is Markovian, each e_t

depends only on e_{t-1}. The likelihood can therefore be expressed as the product of the conditional probabilities:

$$\begin{aligned}\mathcal{L}(\theta, \mu_e, \sigma_{OU} | e_0, \ldots, e_N) &= \prod_{t=1}^{N} P(e_t | e_{t-1}) \\ &= \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi\sigma_{OU}^2 \delta t}} \exp\left(-\frac{(e_t - (e_{t-1} + \theta(\mu_e - e_{t-1})\delta t))^2}{2\sigma_{OU}^2 \delta t} \right).\end{aligned}$$

□

2.6.2 Half-Life

In the context of mean-reverting processes like the Ornstein-Uhlenbeck process, the concept of half-life provides a useful measure of how quickly the process reverts to its mean. Although the exact time for full mean-reversion can be difficult to estimate due to the continuous nature of the process and the dependence on the magnitude of deviation, the half-life offers a more perceptible metric.

Definition 2.6.2 The *half-life* h of an OU process is the average time it takes for the process to revert half-way back to the mean after a deviation.

Lemma 2.6.3 The half-life h of an OU process (2.5) is given as

$$h = \frac{\ln(2)}{\theta}.$$

Proof. Since the diffusion part of the SDE (2.5) does not affect the half-life of an OU process, we focus on the deterministic part (2.7), re-utilizing the solution (2.8). Half-life h is defined as the average time it takes for the deviation from e_0 to revert $\frac{1}{2}$ back to the mean μ_e. Formally,

$$\begin{aligned} & e_h - \mu_e = \frac{1}{2}(e_0 - \mu_e) \\ \Rightarrow \qquad & e_h = \mu_e + \frac{1}{2}(e_0 - \mu_e) \overset{(2.8)}{=} \mu_e + \exp(-\theta h)\,(e_0 - \mu_e)\,. \end{aligned}$$

Canceling μ_e and $(e_0 - \mu_e)$ gives

$$\frac{1}{2} = \exp(-\theta h) \quad \Leftrightarrow \quad h = \frac{\ln(2)}{\theta}.$$

□

Hence, the half-life metric h provides insight into the speed of mean reversion: the higher θ, the lower half-life h and the quicker the process will return to equilibrium.

2.6.3 Implementation

We use a discrete version of the OU process with time steps δt as described in (2.6). The OU parameters $\theta, \mu_e, \sigma_{OU}$ are estimated by minimizing the negative log-likelihood function derived in Theorem 2.6.1. The implementation utilizes a specific function `norm.logpdf` to calculate the log-probability density for the normal distribution. Using such standard package implementations make the code more efficient, robust and shorter.

```python
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from scipy.stats import norm


def ou_likelihood(params, residuals, dt):
    """Calculates the negative log-likelihood of an Ornstein-Uhlenbeck process"""
    theta, mu_e, sigma_ou = params
    likelihood = 0
    for t in range(1, len(residuals)):
        mean = residuals[t-1] + theta * (mu_e - residuals[t-1]) * dt
        variance = sigma_ou**2 * dt
        # increment the log-likelihood by normal log-pdf of the next residual using mean and variance
        likelihood += norm.logpdf(residuals[t], loc=mean, scale=np.sqrt(variance))
    return -likelihood


def estimate_ou_params(residuals, dt=1):  # dt = 1: daily prices, so usually time increment dt = 1
    """Estimate Ornstein-Uhlenbeck process parameters using maximum likelihood estimation.
    The OU process is given as: d(residuals)_t = -theta (residuals_t - mu_e) dt + sigma_ou dW_t"""
    residuals = np.array(residuals)
    initial_params = [0.1, np.mean(residuals), np.std(residuals)]  # [theta0, mu_ou0, sigma_ou0]
    # we minimize negative log-likelihood, which is equivalent to using maximum likelihood estimator (MLE)
    result = minimize(ou_likelihood, initial_params, args=(residuals, dt), method="L-BFGS-B")
    theta, mu_e, sigma_ou = result.x
    return theta, mu_e, sigma_ou


def get_half_life(theta, dt=1):
    """
    Calculate the half-life of an Ornstein-Uhlenbeck process.
    """
    half_life = np.log(2) / (theta * dt)
    return half_life


# example usage
theta, mu_e, sigma_ou = estimate_ou_params(train_data['residuals'])
ou_params = {'theta': theta, 'mu_e': mu_e, 'sigma_ou': sigma_ou}
half_life = get_half_life(theta)
```

3 Pairs Trading Strategy Design

In the following section, we will explore a trading strategy involving a pair of historically cointegrated shares. We will apply mathematical concepts from Chapter 2, such as the Engle-Granger procedure and Ornstein-Uhlenbeck (OU) process modeling for the mean-reverting residual process. These concepts will be used to build our pairs trading strategy step by step." For the original Python source code to generate plots and perform the analytical functions, refer to Appendix A or the Juypyter notebook.
This so-called pairs trading is a market-neutral strategy, meaning that it is designed to target returns independent of overall market direction by offsetting long and short positions in cointegrated share pairs to hedge against larger market fluctuations. Historically, such strategies have performed profitably even in times of financial crises. The idea is based on the assumption that the prices of cointegrated shares move somewhat parallel over time and any significant divergence tends to revert over time. However, should the spread between the asset pair start trending instead of correcting itself to the historic long-term equilibrium, cointegrated pair trading can also bear significant risks.
Our case study in this chapter will mainly focus on the assets of The Coca-Cola Company (KO) and PepsiCo (PEP), although many other examples of such pairs or even series exist in the market. We will analyze some other potentially cointegrated asset pairs and present results of corresponding strategies in Section 3.2.
Useful references to develop the trading strategy include the works of Diamond, e.g. [3], [6], [5].

3.1 Case Study on KO & PEP

3.1.1 Data Preparation & Splitting

We start by downloading the daily closing prices for both tickers, along with the S&P 500 index data via the SPY ETF for performance benchmarking purposes. Since cointegration describes long-term relationships between assets, a look-back window of at least 2-3 years is recommended. Our example will use a history dating back to 2020-01-01. As of August 2024, the downloaded data includes nearly 1200 end-of-day prices.
We proceed by dividing the data into a training set (70%) and test set (30%) according to Fig. 3.1. This method, inspired from Machine Learning techniques, mirrors real-world trading conditions where a strategy is developed and optimized using historical data (the training set) and then validated on more recent, unseen data (the test set). By using a chronological split, the data division simulates a trader defining their strategy based on past market behavior before applying it to future market conditions.
The training and test data are `pandas.DataFrames`. Their structure is shown in Table 3.1.

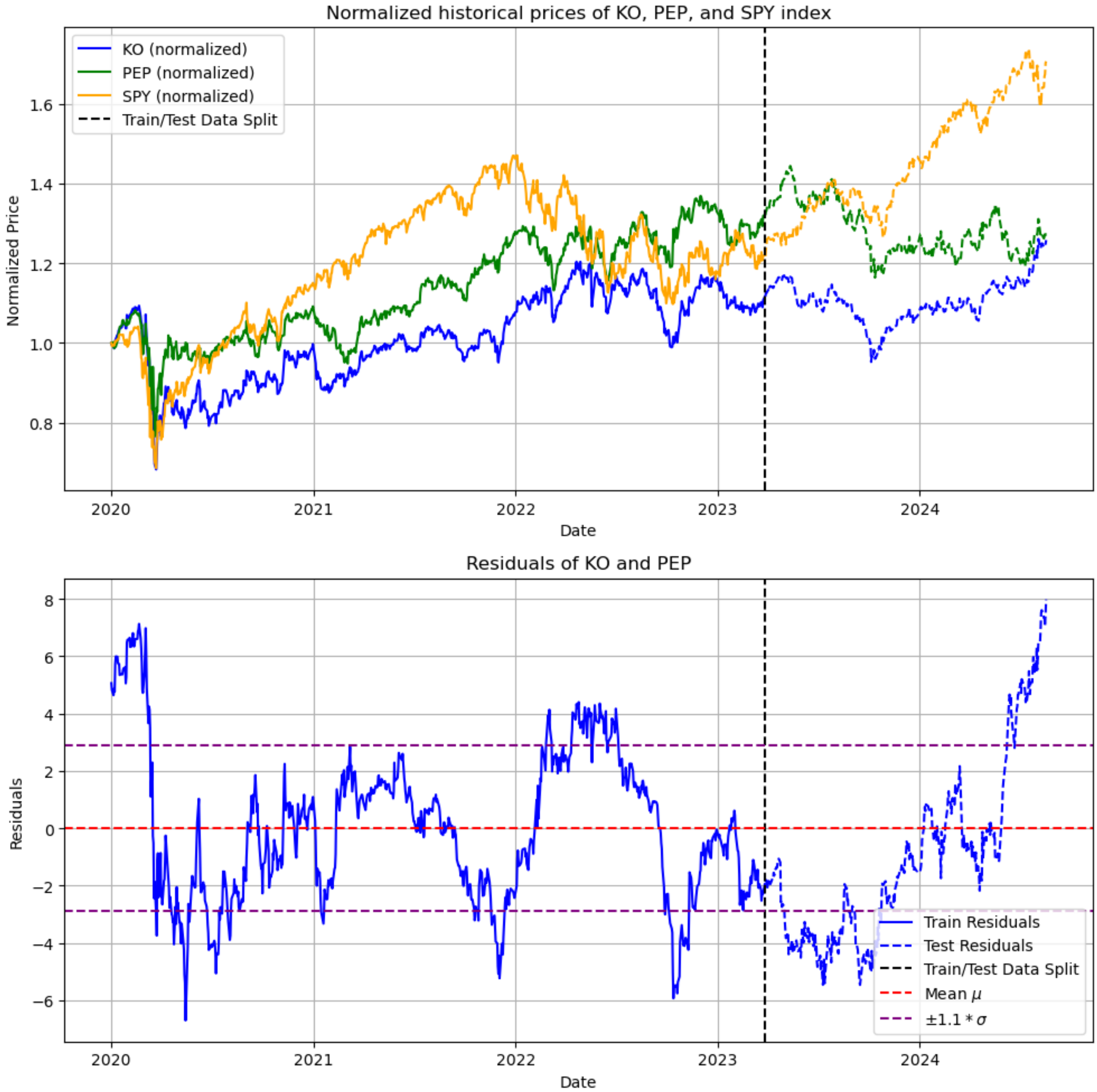


Figure 3.1: Normalized price and residual time series since 2020 for KO, PEP and SPY

3.1.2 Engle-Granger Analysis

Step 1: OLS & ADF Test

Following the separation of prices into training and test set, a two-step Engle-Granger analysis according to Section 2.5 can be conducted. We regress the price time series of KO onto the prices of PEP using Ordinary Least Squares (OLS) regression, i.e., we seek to find $\beta = (\beta_0, \beta_1)$, s.t.

$$\begin{aligned} Y_t &= \beta_0 + \beta_1 X_t + e_t \\ \Rightarrow \quad \mathrm{KO}_t &= \beta_0 + \beta_1 \mathrm{PEP}_t + e_t. \end{aligned}$$

The residual time series e_t is added as a column to both the training and test DataFrames. For example, `train_data.head()` becomes:

Moreover, the regression output estimates `beta` at `array([8.33829699, 0.30621774])`. This means that the prices of Coca Cola (KO) and Pepsi (PEP) can be expressed as

$$\mathrm{KO}_t = 8.33829699 + 0.30621774 \cdot \mathrm{PEP}_t + e_t$$

Date	KO	PEP	SPY
2020-01-02	54.990002	135.820007	324.869995
2020-01-03	54.689999	135.630005	322.410004
2020-01-06	54.669998	136.149994	323.640015
2020-01-07	54.250000	134.009995	322.730011
2020-01-08	54.349998	134.699997	324.450012

Date	KO	PEP	SPY
2023-03-28	61.419998	179.429993	395.600006
2023-03-29	61.860001	180.669998	401.350006
2023-03-30	61.849998	180.830002	403.700012
2023-03-31	62.029999	182.300003	409.390015
2023-04-03	62.400002	182.500000	410.950012

Table 3.1: Structure of `train_data` and `test_data`

and the *hedge ratio* between KO and PEP equals $\beta_1 = 0.30621774$.
To statistically assess the stationarity of the residuals e_t, we perform an ADF test for e_t. With a lag size set to 1 without lag optimization, we find that an ADF statistic is -2.9216. According to Table 2.1, this means that the null hypothesis is rejected at 5% significance level and we conclude that KO and PEP are sufficiently cointegrated.
Note that a lag optimization using AIC suggested an optimal lag of 21. With this higher-lag model, the ADF statistic decreased further to -3.4231, reinforcing the stationarity of the residuals. Hence, the conclusion about cointegration remains consistent regardless of the lag choice.

Step 2: Error Correction Model

The ECM regression for this example can be formulated as

$$\Delta\mathrm{KO}_t = \alpha_0 + \alpha_1 \Delta\mathrm{PEP}_t + \alpha_2 e_{t-1} + \varepsilon_t.$$

Running the numerical ECM implementation yields

$$\alpha = (\alpha_0, \alpha_1, \alpha_2) \approx (-0.007, 0.285, -0.020)\,.$$

- The intercept term α_0 captures any constant offset in the relationship between KO and PEP prices.
- α_1 describes the linear relationship between KO and PEP for shorter-term periods. It signifies that a unit change in $\Delta X_t = \Delta\mathrm{PEP}_t$ results in a $\alpha_1 \approx 0.285$ change in $\Delta Y_t = \Delta\mathrm{KO}_t$.

Date	KO	PEP	SPY	residuals
2020-01-02	54.990002	135.820007	324.869995	5.061209
2020-01-03	54.689999	135.630005	322.410004	4.819388
2020-01-06	54.669998	136.149994	323.640015	4.640158
2020-01-07	54.250000	134.009995	322.730011	4.875466
2020-01-08	54.349998	134.699997	324.450012	4.764173

- In financial time series analysis, a negative α_2 is essential for the concept of cointegration, because it confirms that the two stocks tend to adjust themselves towards equilibrium after any short-term divergence. Our example of $\alpha_2 \approx -0.020$ indeed indicates a mean-reversion towards the long-term mean over time.

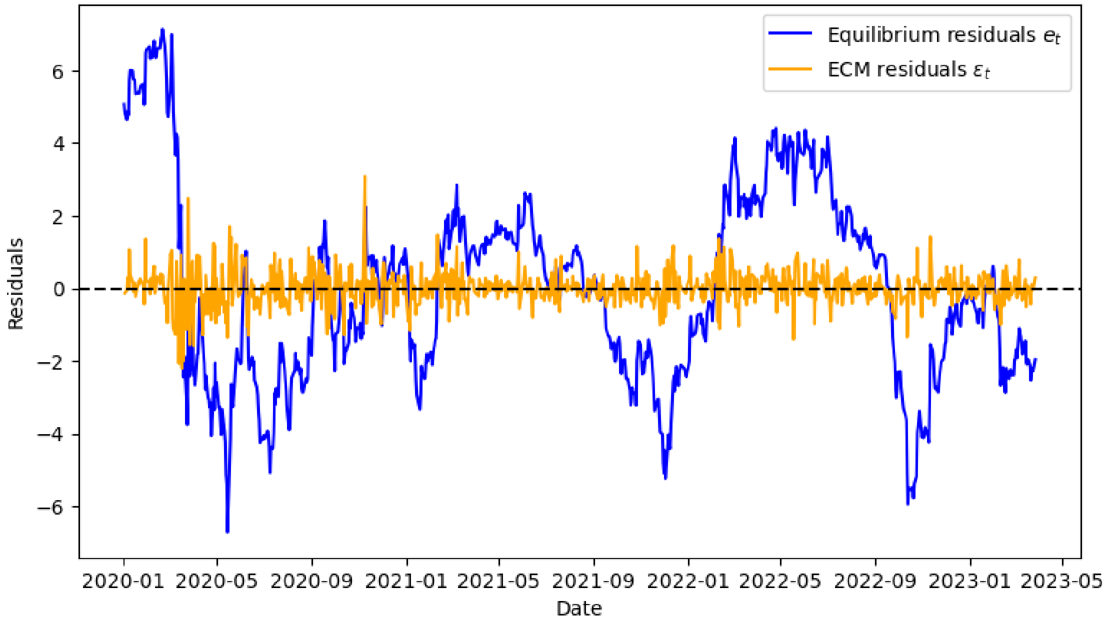


Figure 3.2: Comparison of residuals from Engle-Granger Method

We also observe that while the previously calculated residuals e_t represent deviations from the long-term equilibrium between KO and PEP, the ECM residual time series ε_t represent deviations in the changes in KO prices after accounting for changes in PEP prices, with short-term dynamics and mean-reverting behavior removed from the model. Therefore, ε_t should expectedly be lower in magnitude than the long-term equilibrium residuals e_t and hover closely around 0 with no obvious patterns - similar to white noise residuals, reflecting that most of the short-term dynamics and mean-reverting corrections to the long-term mean have already been accounted for. Indeed Fig. 3.2 depicts the described behavior.

3.1.3 Mean-Reverting Process Modeling via OU Process

A natural choice to model a mean-reverting process such as the stationary residuals e_t is given by the Ornstein-Uhlenbeck process

$$de_t = -\theta(e_t - \mu_e)dt + \sigma_{OU} dW_t,$$

where θ is the speed of mean-reversion to long-term equilibrium mean μ_e and σ_{OU} is the standard deviation (volatility) of the process. As presented in Section 2.6, we will estimate the OU parameters $\theta, \mu_e, \sigma_{OU}$ given residuals $e_0, ..., e_N$ between KO and PEP using a MLE approach. We find that

$$(\theta, \mu_e, \sigma_{ou}) \approx (0.0197, -0.4343, 0.4925)\,.$$

From $\theta \approx 0.4925$, we obtain a half-life of 35.10 trading days according to Lemma 2.6.3.

Fig. 3.3 indeed visualizes that a simulated OU process displays similar dynamics as the actual residuals time series e_t. Some deviations from the original residual time series e_t are expected, since the OU process is a continuous-time stochastic process simulated from an SDE including Brownian increments (randomness).

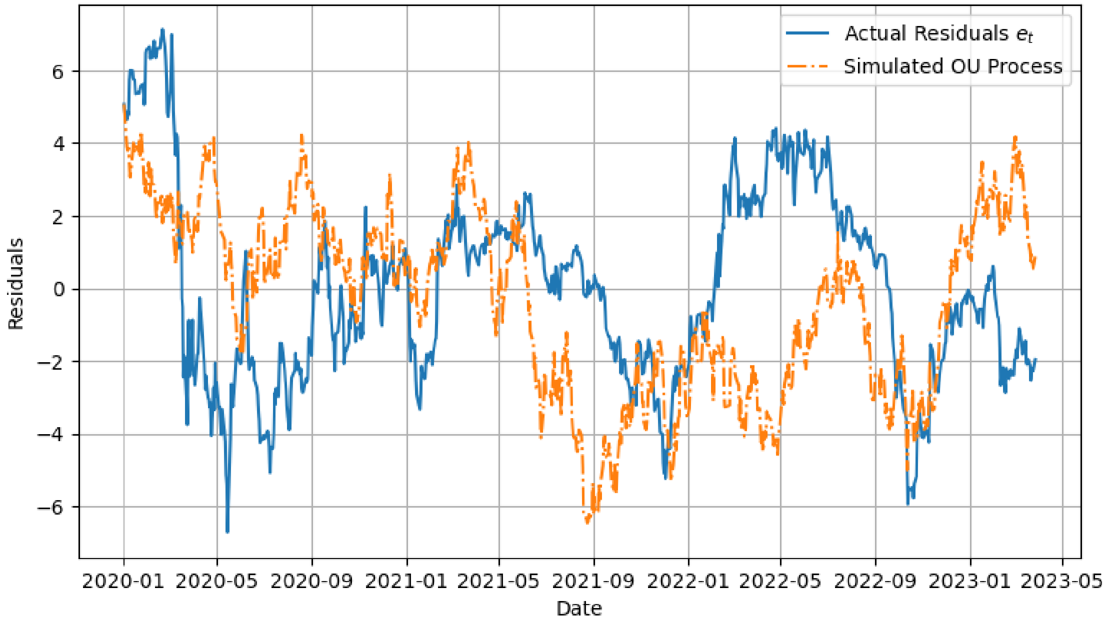


Figure 3.3: Actual residuals vs. simulated Ornstein-Uhlenbeck process

3.1.4 Trading Strategy Description

Our trading strategy constructs a simple portfolio with two opposing positions in the cointegrated asset pair KO and PEP. As described earlier, the strategy aims to profit from temporary deviations in the price relationship, assuming that their spread divergence will eventually revert to the long-term mean, $\mu_e \approx -0.4343$. The portfolio class implementation can be found in Appendix A.2. The code replicates the decision-making process of a real trader, who - after thorough cointegration analysis - bets on the spread between KO and PEP to correct over time.
The entry and exit signals to open or close positions are defined by a symmetric band around the equilibrium mean of $\mu_e \pm Z \cdot \sigma_{eq}$, where Z is a factor to be determined and $\sigma_{eq} = \frac{\sigma_{OU}}{\sqrt{2\theta}}$ [6]. When the spread

$$e_t = \mathrm{KO}_t - \beta_0 - \beta_1 \cdot \mathrm{PEP}_t$$

between KO and PEP prices deviates significantly from μ_e, moving beyond the upper or lower bounds of the band, the strategy signals an opportunity to enter a position, anticipating a reversion to the mean. Positions are subsequently closed again once the spread reverts to the mean. More specifically:
Entry signals: If the portfolio is flat, the positions are entered as follows:

- If the spread crosses the upper bound $\mu_e + Z \cdot \sigma_{eq}$, a short KO/long PEP position is triggered. This occurs when KO is overpriced relative to PEP.
- If the spread crosses the lower bound $\mu_e - Z \cdot \sigma_{eq}$, a long KO/short PEP position is triggered. This occurs when KO is underpriced relative to PEP.

The long/short ratio is equal to the hedge ratio $\beta_1 = 0.30621774$, obtained previously through the initial OLS regression.
Exit signals: If the portfolio currently has long/short positions, they are closed as follows:

- If the spread reverts to just below the mean μ_e after having entered positions when it crossed above $\mu_e + Z \cdot \sigma_{eq}$, the short KO/long PEP position is closed.
- If the spread reverts to just above the mean μ_e after having entered positions when it crossed below $\mu_e - Z \cdot \sigma_{eq}$, the long KO/short PEP position is closed.

Upon closing the positions, the realized PnL from the trades is recorded.
Additionally to the realized PnL, while positions are still open, the code continuously calculates the unrealized PnL based on market movements, providing a comprehensive view of the strategy's performance in real time.

3.1.5 Strategy Demo for $Z = 1$

To provide a clearer understanding of the strategy's design, we demonstrate it for a fixed value of $Z = 1$, i.e., the entry signal bounds around the long-term mean will be defined as

$$\mu_e \pm Z \cdot \sigma_{eq} = -0.4343 \pm 1 \cdot 2.4781 = (-2.9124, 2.0438)\,.$$

We evaluate the strategy on the *test* data set, using strategy parameters derived from the training data. These parameters include the hedge ratio $\beta_1 = 0.3062$, equilibrium mean $\mu_e = -0.4343$, and equilibrium standard deviation $\sigma_{eq} = 2.4781$.
The execution of the resulting strategy is illustrated in Fig. 3.4, where entry and exit points are marked with red dots. The positions taken are shown in Fig. 3.5 and the resulting PnL in Fig. 3.6.

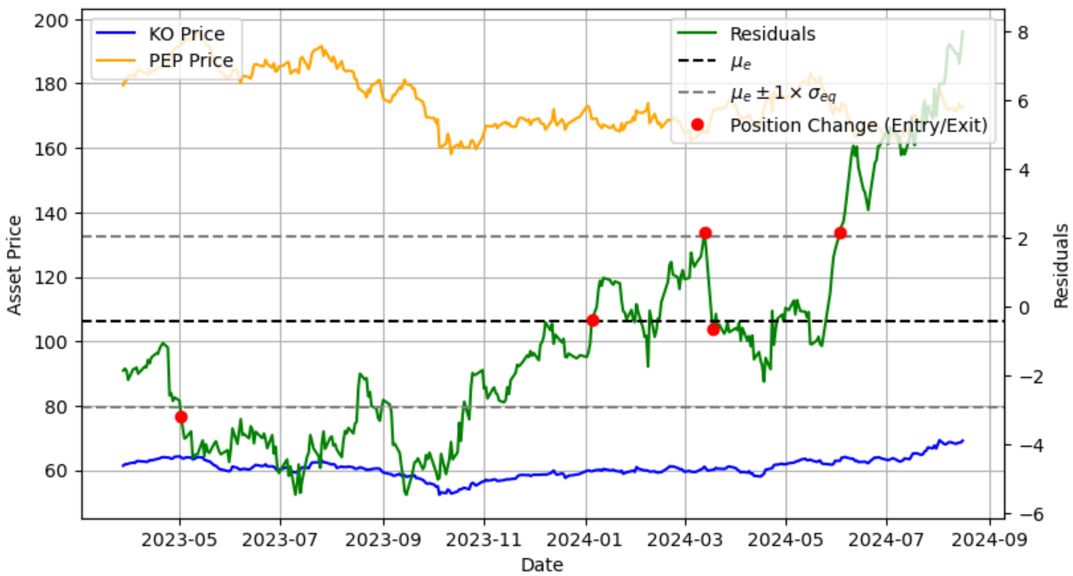


Figure 3.4: Asset prices and residuals with equilibrium band for $Z = 1$ for KO and PEP. Red dots mark the entry and exit points of the trading strategy.

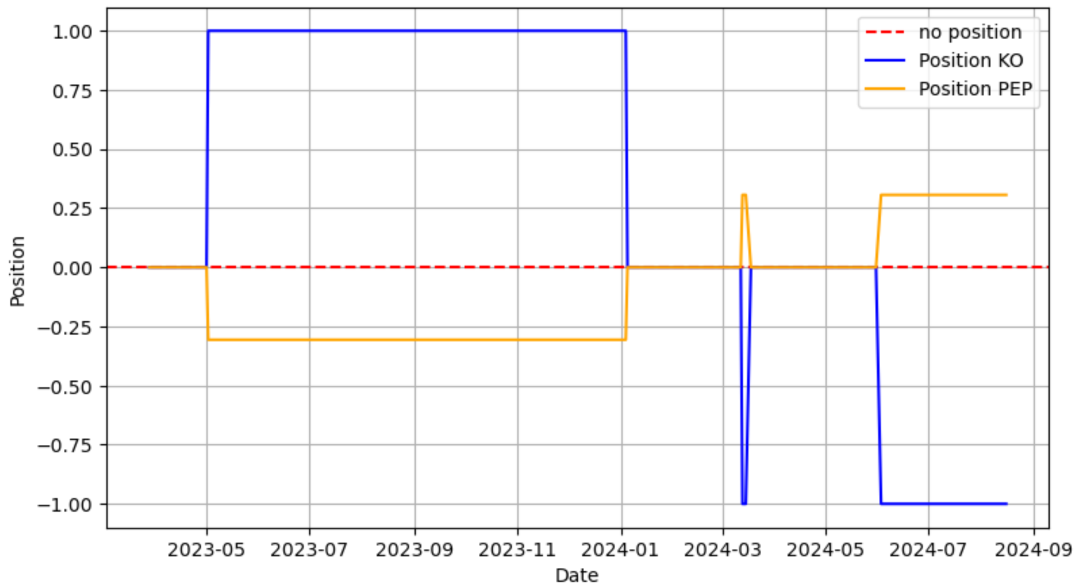


Figure 3.5: Positions for KO and PEP taken over the test data period.

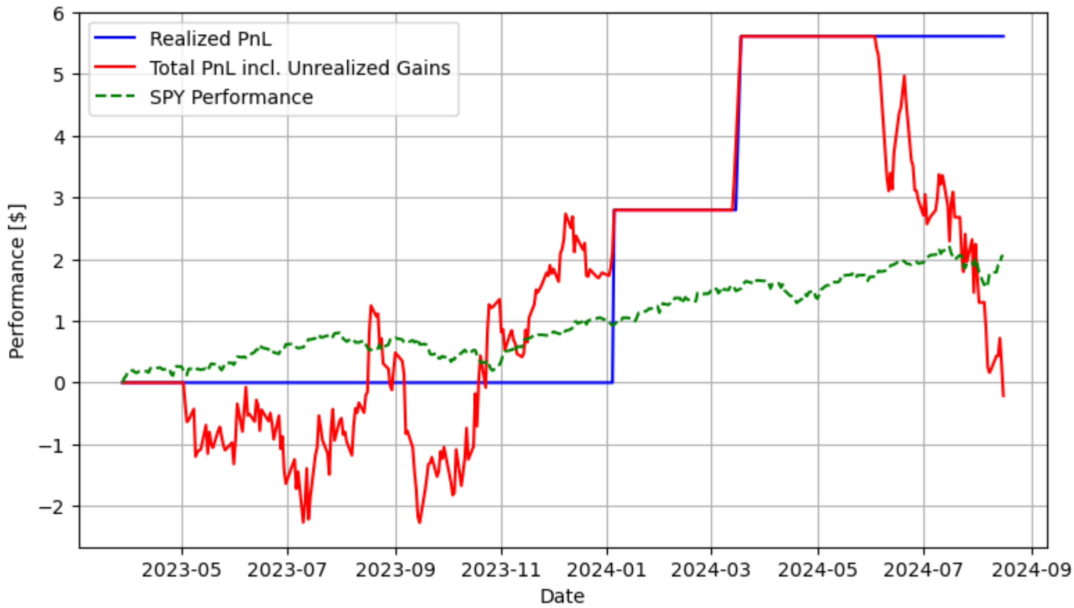


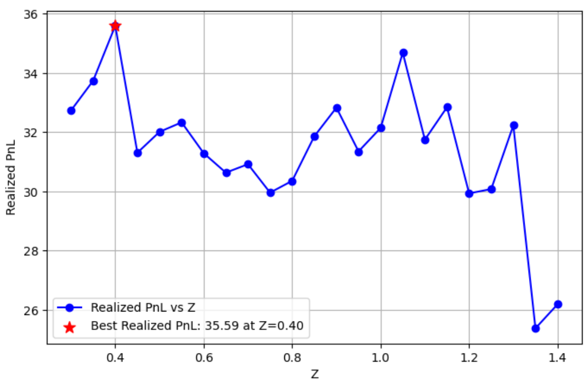
Figure 3.6: PnL evolution over the test data period, compared against the S&P 500 index. The unrealized PnL has been declining after the last trade, whose position is still open, was executed, since the residuals continue to deviate further away from μ_e. Since no stop-loss mechanism was implemented in the strategy, these losses can potentially increase immeasurably.

Comparison Against Passive Index Investment Strategy

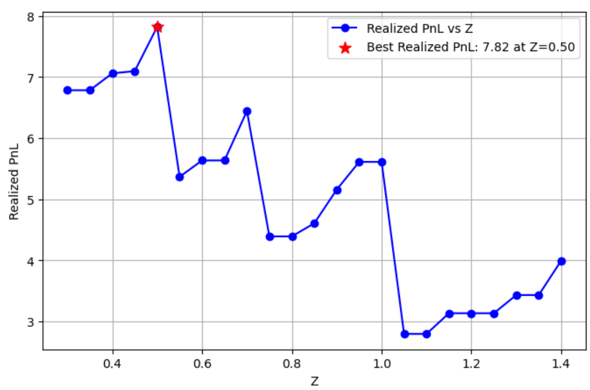
Fig. 3.6 compares the returns of the pairs trading strategy against the performance of the S&P 500 index. More specifically, we compare the pairs strategy against a passive investment strategy where the trader simply buys into the equity index (e.g. through a replicating ETF). To ensure the strategies are comparable, we scale the index investment to match the initial amount used to execute the first trade for the pairs trading strategy, looking at the actual market prices of the shares when the positions were opened, to determine our portfolio nominal.

3.1.6 The Impact of Z - An Optimization Study

The parameter Z plays a crucial role in our trading strategy, as it determines the width of the band around the equilibrium mean μ_e. Consequently, Z directly influences the entry signals and overall performance of the strategy. Following the recommendation of Diamond [6], we systematically varied Z across an equidistant range of $[0.3, 1.4]$ to identify the optimal value Z^* that yields the highest PnL.
This optimization was performed on the training data set to again simulate a real-world trading scenario, where historical data is used to determine strategy parameters before applying them in practice. As shown in Fig. 3.7a, $Z^* = 0.4$ produced the highest PnL on the training data. Therefore, a trader would be inclined to select $Z^* = 0.4$ for further trading purpose.
However, when applying this value to the test data, Fig. 3.7b reveals that $Z^* = 0.4$ did not yield the best results. Instead, the test data indicated that $Z = 0.5$ would have lead to higher PnL. This discrepancy illustrates the challenge of overfitting that can also occur in Machine Learning, where a model optimized on training data may not generalize perfectly to unseen test data.



(a) A study varying Z on the *training data* shows that the highest realized PnL is achieved with $Z^* = 0.4$.



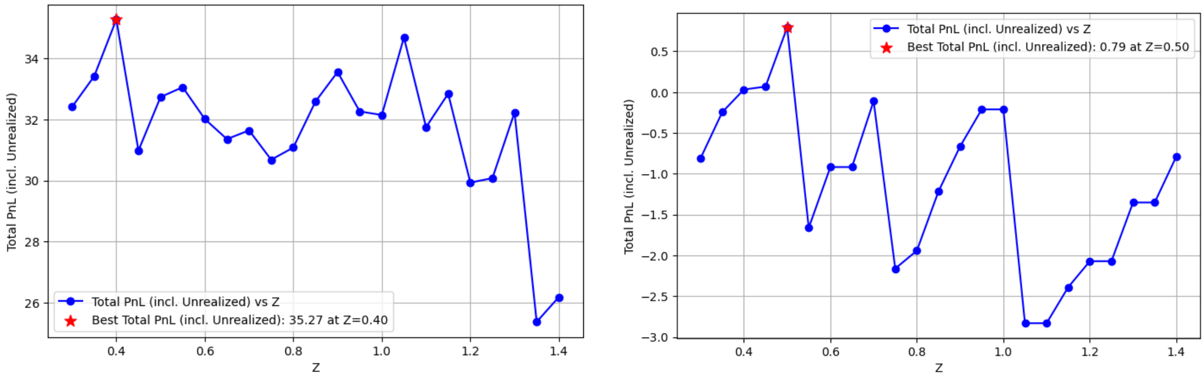
(b) $Z^* = 0.4$ does not prevail as optimal under the *test data set*; the highest realized PnL is achieved with $Z = 0.5$.

Figure 3.7: Comparison of **realized** PnL across varying Z values during training and test phases. The overall lower PnL values in the test data are not necessarily alarming. Not only was the training data used to fine-tune parameters, but the training data set is more than twice as large, providing more than double the trading opportunities.

Fig. 3.7 presents only realized PnL, i.e., gains that have already been secured after positions were closed. For a comprehensive evaluation, a trader would likely consider unrealized PnL as well to make decisions for the future. Especially for the test data set reflecting on more recent data points, unrealized PnL could differ significantly from realized PnL, if the most recent market movements on positions that are currently still open have deviated far from the equilibrium mean μ_e. The overview of total PnL = Realized PnL + Unrealized Gains is shown in Fig. 3.8.

3.1.7 Optimal Strategy with $Z^* = 0.4$

Despite the test data performing more profitably at $Z = 0.5$, we base our optimization of the Z value on the training data and identify the optimal Z^* as 0.4. This resulting strategy would therefore take the following parameters:



(a) Varying Z for the *training data* results in similar findings for realized and unrealized PnL.

(b) For the *test data*, unrealized losses severely impact the strategy performance for all values of Z.

Figure 3.8: Comparison of **total** PnL across varying Z values during training and test phases. Unlike Fig. 3.7, this graph additionally includes unrealized gains, providing a more comprehensive view of strategy performance.

- hedge ratio $\beta_1 = 0.3062$, obtained from OLS regression for KO and PEP prices,
- equilibrium mean $\mu_e = -0.4343$ and standard deviation $\sigma_{eq} = 2.4781$, derived from the MLE for the OU parameters,
- $Z = Z^* = 0.4$, obtained from testing various values for Z on the training data.

The entry signal bands are defined as

$$\mu_e \pm Z \cdot \sigma_{eq} = -0.4343 \pm 0.4 \cdot 2.4781 = (-1.4255, 0.5570)\,.$$

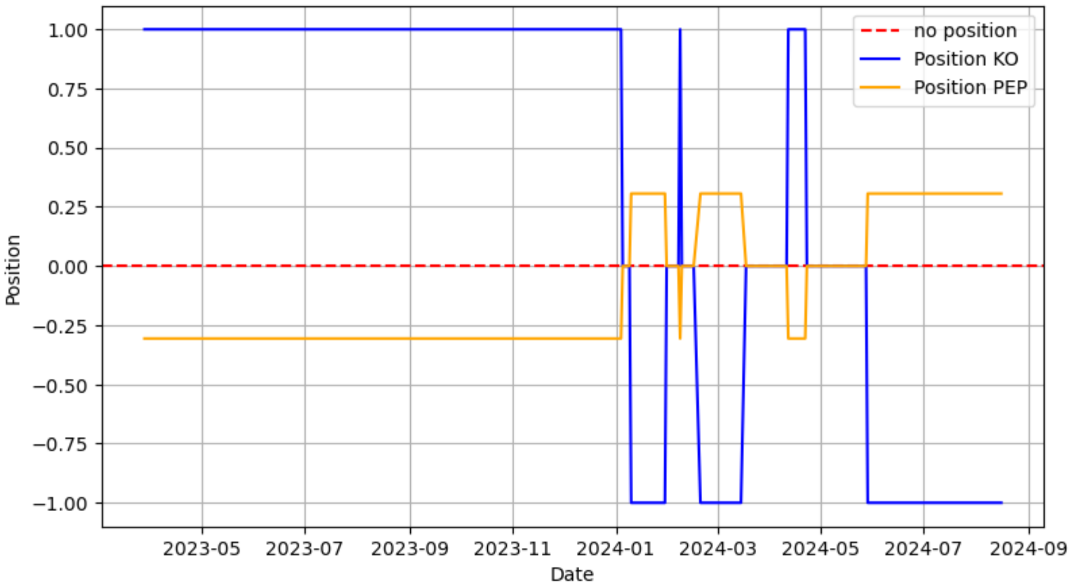


Figure 3.9: Positions for KO and PEP taken over the test data period.

The trading strategy immediately opens long/short positions since the residual process begins beyond the entry signal, if these positions were not already open in the training period (Fig. 3.9). These positions remained open for almost a year as the spread took an unusually long time to mean-revert during 2023 (Fig. 3.10). This could be attributed to

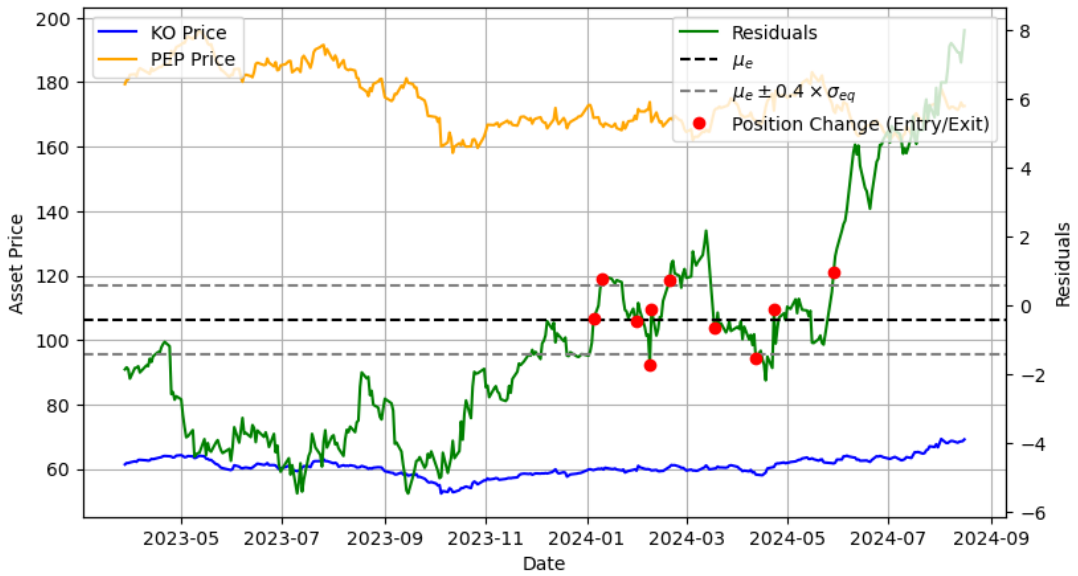


Figure 3.10: Asset prices and residuals with equilibrium band for $Z^* = 0.4$ for KO and PEP.

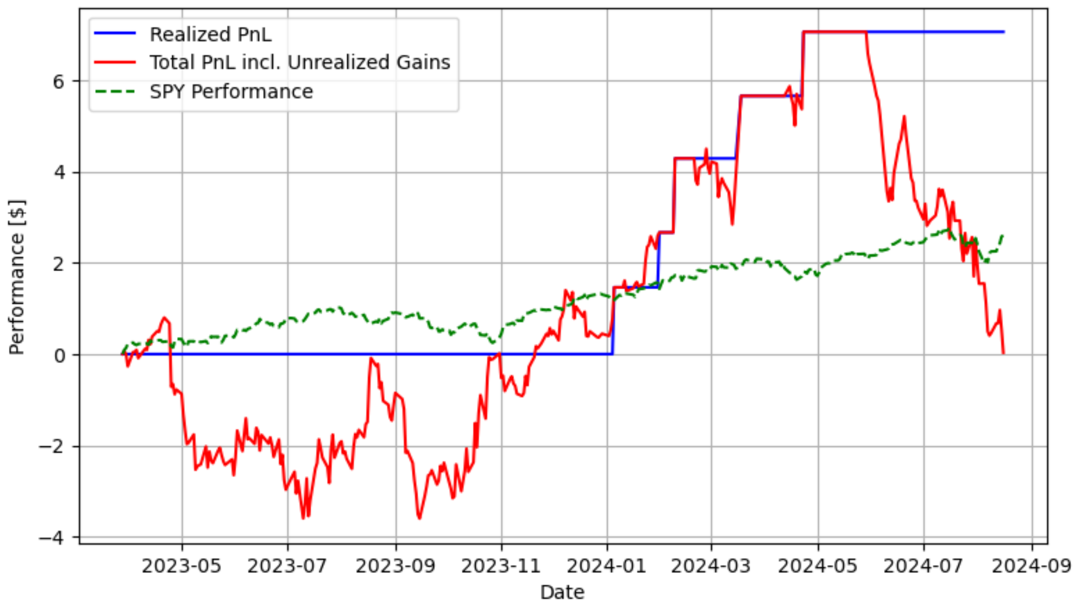


Figure 3.11: PnL evolution over the test data period, compared against the S&P 500 index. The unrealized PnL has been declining since the last trade (still open) was executed, as the residuals continue to deviate further away from μ_e.
Performance: *Realized PnL* = 7.061, *Total PnL* (incl. unrealized): 0.0316.

the Federal Reserve's interest rate policies, disruptions in the US financial markets in early 2023, a continued shift in consumer preferences towards healthier alternatives versus soda products, ongoing geopolitical issues in Russia or Israel, or other market-driven factors. Following this period, the spread fluctuated around the equilibrium mean μ_e, leading the strategy to open and close positions more frequently and profitably until June 2024. Subsequently, with KO share prices rising and PEP prices declining, the spread has been widening, deviating further from the equilibrium, resulting in accumulating unrealized losses. Despite this, the realized PnL of the strategy still outperforms the S&P 500 benchmark when scaled to the same nominal value, as illustrated in Fig. 3.11.

As Fig. 3.12 illustrates, the performance during the training period significantly outperformed the benchmark, and trading activity was higher. In an overall sideways-trending market (indicated by the relatively flat performance of the S&P 500 index during this period), our strategy successfully capitalized on short-term price fluctuations between KO and PEP, leading to substantial realized and unrealized profits. The increased trading frequency during this period suggests that the strategy was well-tuned to arbitrage on inefficiencies and temporary deviations in the price relationship between KO and PEP.

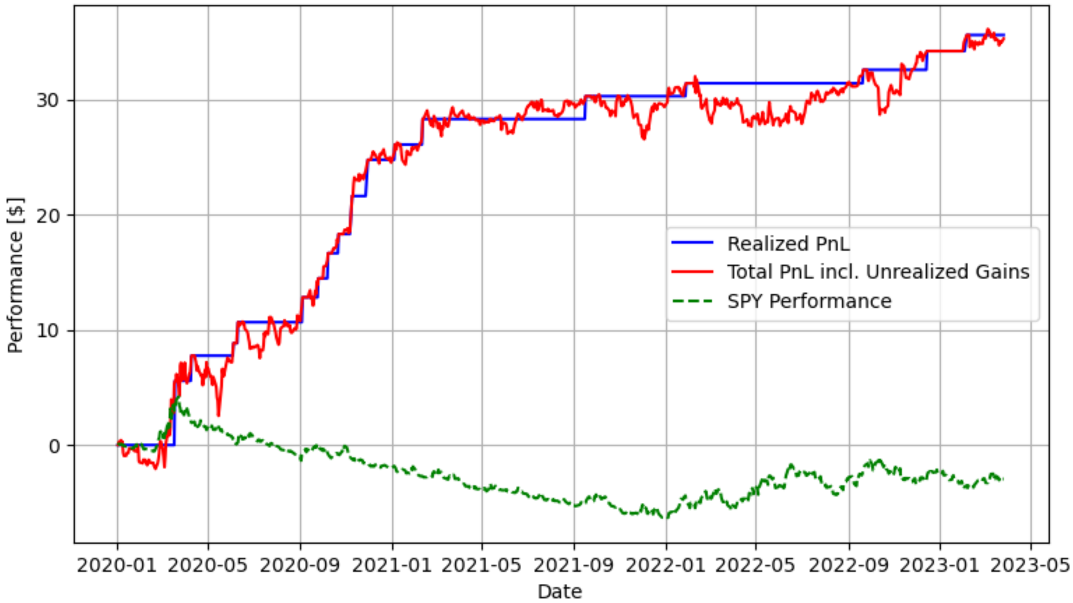


Figure 3.12: PnL evolution over the training data period, compared against the S&P 500 index. The comparison assumes that the initial investment in the index and (KO/PEP) strategy was equal, with the index held passively over the entire time period.
Performance: *Realized PnL* = 35.5934, *Total PnL* (incl. unrealized): 35.2687.

However, the transition from the training period to the test period seemingly underwent significant changes in market dynamics resulting in less predictable behavior, likely influenced by various macroeconomic factors. This not only reduced the frequency of profitable trades but also increased the risk of prolonged deviations from the equilibrium in the test period. This highlights the importance of adapting the strategy to evolving market conditions and considering additional risk management measures, such as implementing a stop-loss mechanism, to mitigate potential losses, further discussed in Section 3.1.9.

3.1.8 Reversing the Roles of KO and PEP

To provide a more thorough analysis of the long-term relationship of KO and PEP, one could also try to reverse the roles of KO and PEP in the pairs trading design and start by regressing the prices of Y=PEP onto the prices of X=KO. This reversal is worth exploring for several reasons:

- **OLS regression is not symmetrical in X_t and Y_t**: Reversing KO and PEP in the regression could lead to different hedge ratios and residuals, potentially resulting in distinct trading signals and performance outcomes.
- **Impact on cointegration and error-correction models**: The cointegration relationship might change when reversing the roles, affecting the equilibrium relationship and the speed of mean-reversion.

- **Different optimal parameters could impact strategy performance**: The realized and unrealized PnL may differ based on which stock is treated as the dependent variable, since new insights into the KO-PEP asset pair might be provided.
- **New perspective on the market revealing asymmetries**: Reversing the roles of the asset pair may reveal asymmetries in market behavior of stock 1 relative to stock 2.
- **Robustness check**: Conducting a reverse analysis can also act as a robustness and reliability check for the existing strategy in Section 3.1.7.

We obtain a new regression equation of

$$\begin{aligned} & Y_t = \beta_0 + \beta_1 X_t + e_t \\ \Rightarrow \quad & \mathrm{PEP}_t = \beta_0 + \beta_1 \mathrm{KO}_t + e_t \\ \Rightarrow \quad & \mathrm{PEP}_t = 9.15949691 + 2.61111547 \cdot \mathrm{KO}_t + e_t, \end{aligned}$$

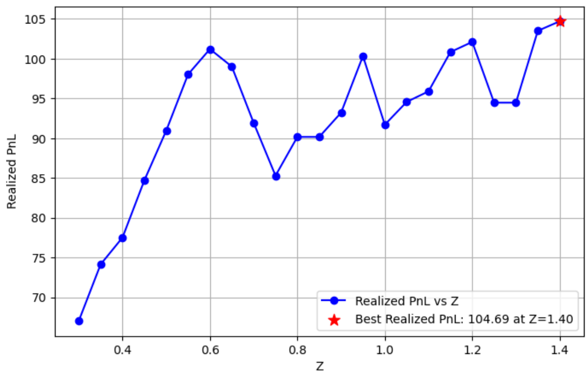
with a hedge ratio of $\beta_1 = 2.61111547$. Using a fixed lag size of 1, the ADF test shows an ADF statistic value of -2.6576 at p-value 0.0817 for the reversed asset pair. According to Table 2.1, we fail to reject to null hypothesis at a significance level of 5% and conclude that the asset pair is not cointegrated. But our entire strategy including the ECM and OU process are based on the assumption that the residuals between the pair would mean-revert! Can it even be applied now?
Let us try to optimize the lag size to obtain a more favorable result: The implementation in Section 2.4.1 indeed allows lag optimization using AIC. We now obtain an ADF statistic of -3.0287, confirming the stationarity of the residuals at significance level of 5%. The optimal lag size with respect to AIC is then given as 21 which indicates that the relationship between PEP and KO might be more complex than initially modeled.
The following parameters, optimized on the training set with reversed asset roles for PEP and KO, were obtained (cf. Section 3.1.7 or the original asset pair order of KO and PEP):

- hedge ratio $\beta_1 = 2.6111$, obtained from OLS regression for PEP and KO prices,
- optimal lag size with respect to AIC: 21
- equilibrium mean $\mu_e = 1.7735$ and standard deviation $\sigma_{eq} = -7.2135$, derived from the MLE for the OU parameters,
- $Z = Z^* = 1.4$, obtained from testing various values for Z on the training data, illustrated in Fig. 3.13. ($Z^* = 1.35$ would also be a valid choice.)

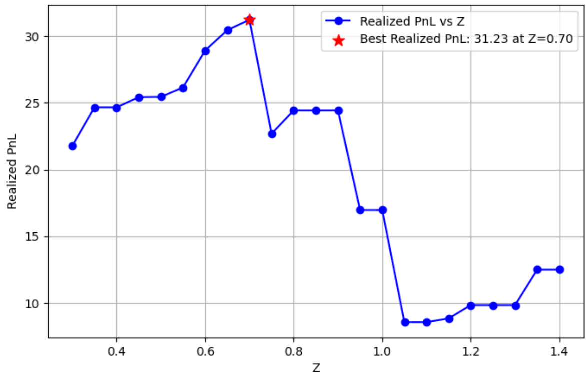
The entry signal bands are defined as

$$\mu_e \pm Z \cdot \sigma_{eq} = 1.7735 \pm 1.4 \cdot 7.2135 = (-8.3254, 11.8725)\,.$$

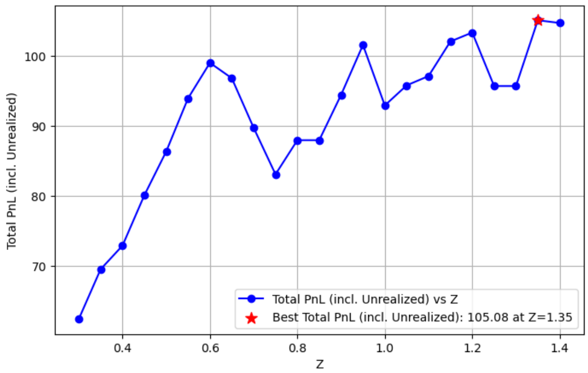
Given that $Z^* = 1.4$ is now relatively large, the trading signals are more spread out, resulting in less trading activity, as seen in Fig. 3.14. However, the strategy seems to promise more stable returns in Fig. 3.15, since the PnL evolution indicates that the unrealized losses are less significant compared to when the roles of PEP and KO were not reversed. In particular, the composed strategy with reversed asset roles now outperforms the S&P 500 index in both realized and total PnL terms.



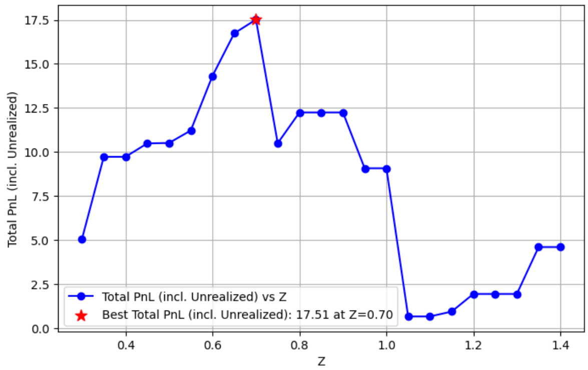
(a) Realized PnL on *training data*



(b) Realized PnL on *test data*



(c) Total PnL, incl. unrealized PnL on *train data*



(d) Total PnL, incl. unrealized PnL on *test data*

Figure 3.13: Comparison of **realized** (Fig. 3.13a, 3.13b) and **total** PnL, incl. unrealized PnL (Fig. 3.13c, 3.13d) across varying Z values. The optimal Z values of 1.35 and 1.40 with comparable impact on the PnL (Fig. 3.13a, 3.13c) unfortunately do not translate effectively for the test data sets (Fig. 3.13b, 3.13d).

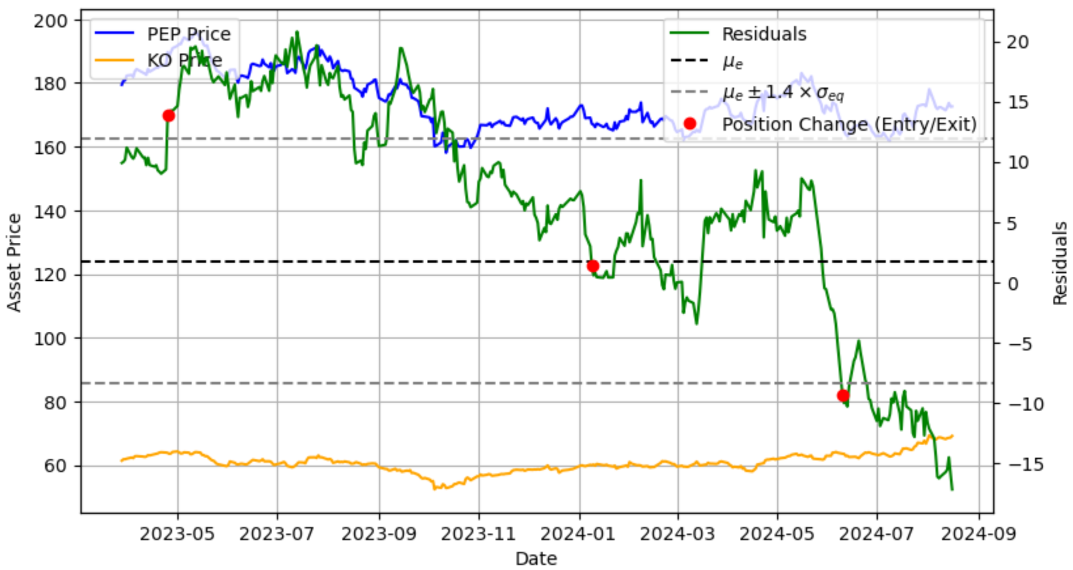


Figure 3.14: Asset prices and residuals with equilibrium band for $Z = 1.4$ for PEP and KO. Red dots mark the entry and exit points of the trading strategy.

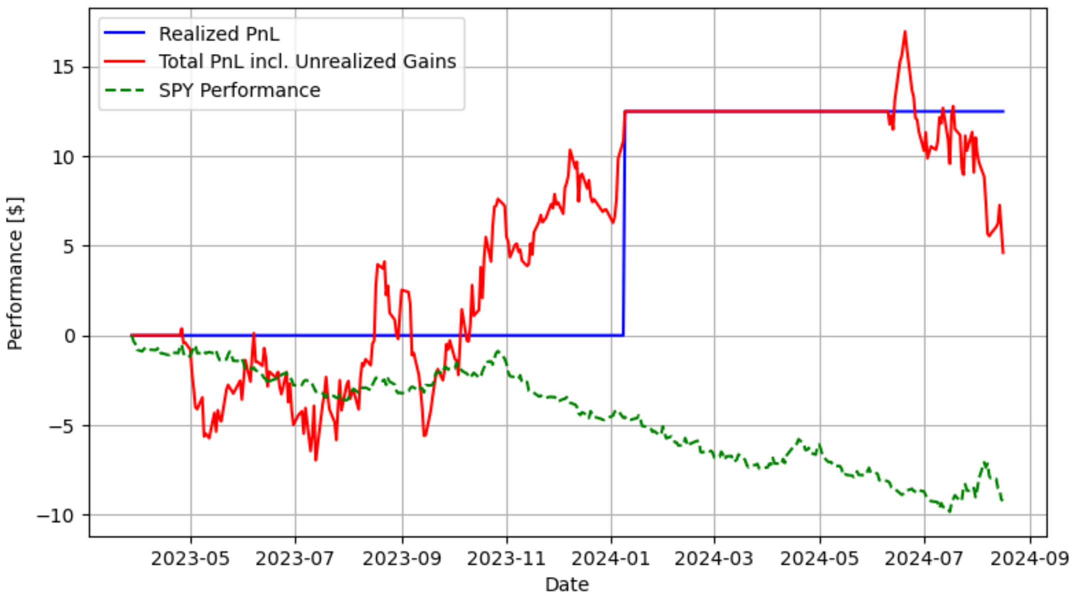


Figure 3.15: PnL evolution over the test data period, compared against the S& P500 index. The unrealized Pnl declined after the last trade was executed, but the loss is less significant compared to the unrealized PnL in the strategy where the asset roles of PEP and KO were reversed in 3.15. Other than the optimal strategy before, this composition outperforms the index. Note that the negative trend in the index is not a bug, but rather reflects the scaling of the index investment to match the initial pairs trading investment, which started with a net short position.

3.1.9 Conclusion & Improvement Ideas

The trading strategy based on the optimized parameters over the training period $Z^* = 0.4$ would have demonstrated exceptional performance during that period (Fig. 3.12), significantly outperforming the S&P 500 benchmark. However, its actual performance was finally measured on the test data, where it faced difficulties due to changing market dynamics.

Suggested Improvements

To enhance the profitability, adaptability and risk profile of the strategy, several following improvement ideas arise:

- **More frequent parameter adjustments**: Regular updates to parameters such as the hedge ratio β_1, or Z could improve the responsiveness of the strategy to market changes and trends.
- **Stop-loss implementation**: The introduction of a stop-loss mechanism would limit potential losses from prolonged deviations from the mean. Such losses could increase indefinitely without stop-loss functionality. With a stop-loss logic, the risk of significant drawdowns during periods of market disruption or unpredicted trends in asset relationships - as observed towards the end of the test period in 3.10 - could be minimized.
- **Inclusion of more than two cointegrated assets**: Expanding the portfolio to include more assets enhances diversification and likely results in more stable returns, enhancing the strategy's robustness and risk profile.

- **Continuous risk analysis**: Implementation & monitoring of additional risk measures such as VaR, Expected Shortfall, annualized volatilities, rolling beta, and the Sharpe ratio would provide a more comprehensive view of the strategy's risk profile. A meticulous risk analysis can identify potential vulnerabilities and improve overall returns. Real-world financial institutions have a plethora of risk metrics that are monitored daily, including backtesting procedures and limit frameworks.
- **Machine learning integration**: Incorporating machine learning techniques to predict regime changes or optimize parameters could further enhance performance.

By implementation of above improvements, the trading strategy can be better equipped to navigate ever-changing markets, minimize risks, and capitalize on profit opportunities more effectively. Continuous fine-tuning and adaptation are essential to maintain the competitiveness and long-term profit in the market.
But even in their current form, both pair trading strategies - whether for (KO/PEP) (Section 3.1.7) or (PEP/KO) (Section 3.1.8) - were able to efficiently exploit certain market situations. The reversed strategy presented in Section 3.1.8 was able to model the mean-reversed residual process more accurately, judging from its outperformance of the S&P 500 index during the same time period, despite not even selecting the optimal Z parameter for the test data set, as was shown in Fig. 3.13b & 3.13d.

3.2 Other Candidate Share Pairs: Cointegrated? Suitable for Pairs Trading?

In this section, we expand our analysis to briefly explore other share pairs that could potentially be suitable for a cointegrated pairs trading strategy. The methodology applied is identical to the one explained in-depth for the previous (KO/PEP) example with the following steps:

1. On the training data set: Engle-Granger step 1: OLS

$$Y_t = \beta_0 + \beta_1 X_t + e_t$$

   followed by ADF cointegration test
2. On the training data set: If cointegrated in 1: Engle-Granger step 2: ECM
3. On the training data set: Ornstein-Uhlenbeck mean-reversion process for the residuals with MLE for the process parameters
4. On the training data set: Iteratively test values for threshold Z to identify the optimal value to signal position entry
5. On the **test** data set: Evaluate trading strategy.

Notation: (A/B) should denote the asset pair (Y_t/X_t). Example: In (KO/PEP), KO shall denote the dependent variable with regressor X_t =PEP.
Relevant code snippets are provided in Appendix A.4, additional plots are also shown in the accompanying Jupyter notebook.

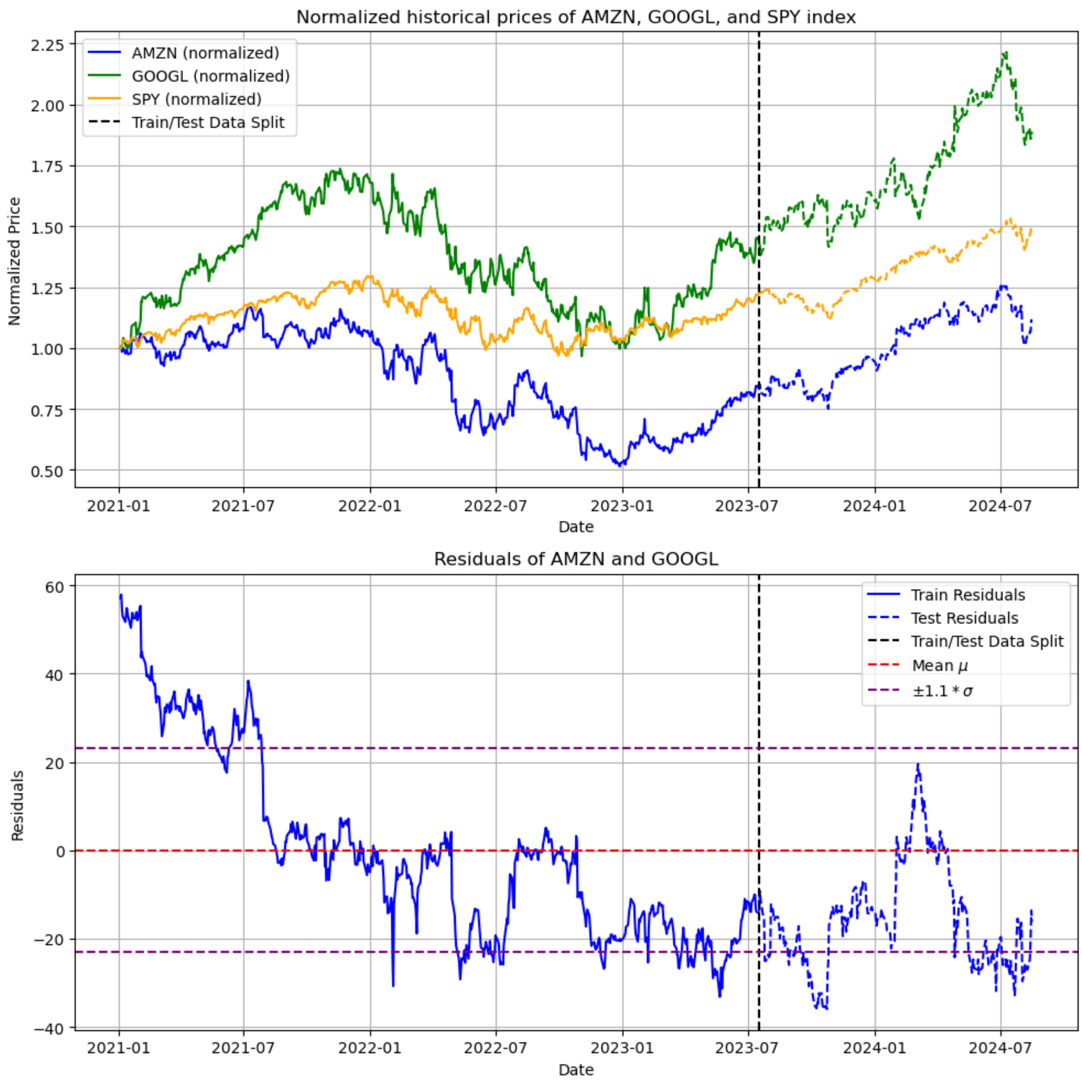
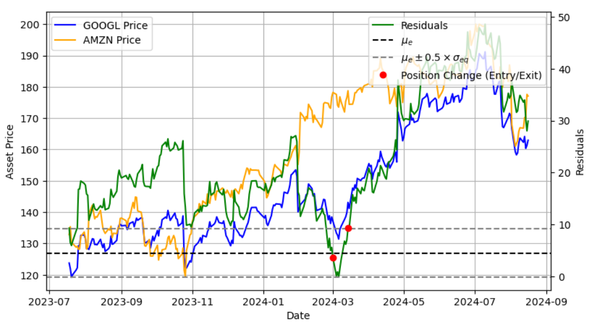


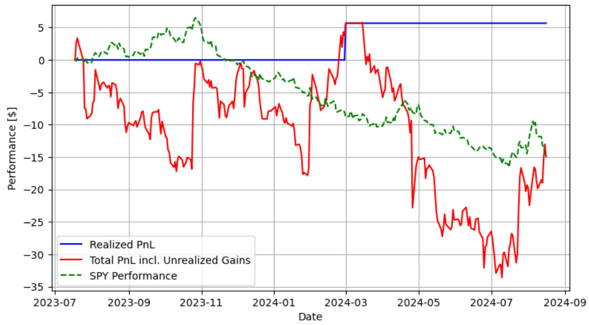
Figure 3.16: Residuals and asset prices plotted over training and test period. Some co-movement is already visible in the plot, but we will still statistically confirm the visual intuition.

3.2.1 Google and Amazon

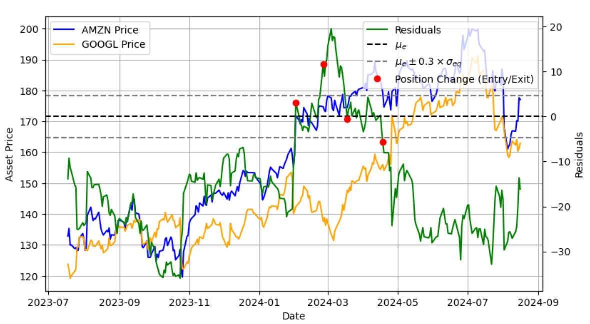
Given their market dominance, analyzing two of the Big Tech companies such as Alphabet (GOOGL, formerly traded and often still referred to as the Google stock) and Amazon (AMZN) is a natural choice. As a start date, we pick 2021-01-01.
Both pair orders (Y_t =GOOGL/X_t =AMZN) and (Y_t =AMZN/X_t =GOOGL) are cointegrated at 5% significance level (ADF statistic of -3.1243 and -2.8773 respectively) in the training data, indicating a stable long-term relationship between the two stock prices. The long-term mean for (Y_t =GOOGL/X_t =AMZN) is found at $\mu_e = 4.5005$ with a half-life of 40.5 trading days. Fig. 3.16 shows the share price movements and the residuals calculated from the linear combination of the two. While the residuals suggest that the spread is on its way to revert to its long-term equilibrium again, there has been an evident deviation from μ_e during the test period. This deviation suggests a potential change in the relationship occurred in the test period.
The results of the pairs trading strategy, optimized based on the training data, are illustrated in Fig. 3.17. In both pair orders, the residual process spends a significant amount of time completely outside of the signal bands during the test period. Indeed, both bands



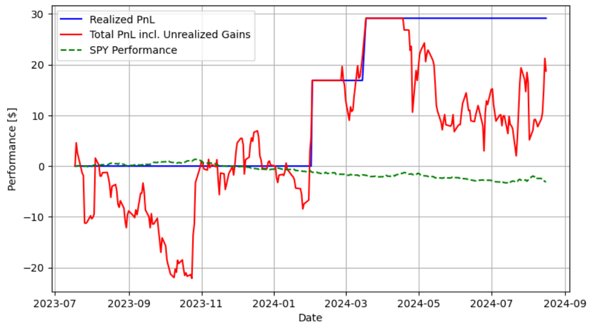
(a) Prices & residuals for Y_t=GOOGL, X_t=MSFT



(b) PnL evolution for Y_t=GOOGL, X_t=AMZN



(c) Prices & residual for Y_t=AMZN, X_t=GOOGL



(d) PnL evolution for Y_t=AMZN, X_t=GOOGL

Figure 3.17: Pairs trading strategy for (AAPL/MSFT) and (MSFT/AAPL).

were set relatively narrow from ($Z = 0.5$ and $Z = 0.3$) - most likely because the residuals fluctuated relatively close to the equilibrium mean during the training period, generating many profitable signals with lower Z values. The long stretches of time spent outside of the bands indicates that both strategies were not optimally designed to handle the regime change displayed in the test period, since only few entry signals were detected. Moreover, the performance of the strategy is significantly impacted by the order of the pairs, with the PnL for (Y_t=AMZN, X_t=GOOGL) differing by an order of magnitude from 10^1 vs.10^0.

3.2.2 Apple and Microsoft

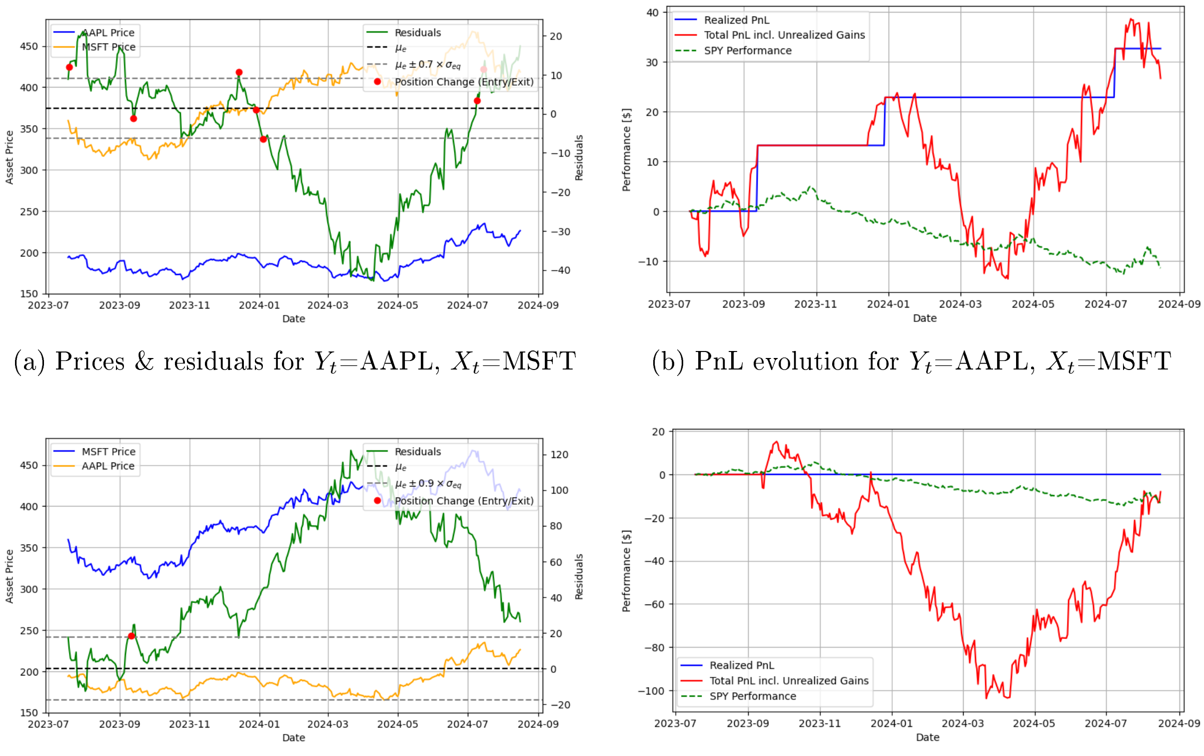
Another look at a Tech Titan couple: We analyze Apple (AAPL) and Microsoft (MSFT) shares with a start date of 2021-01-01. Surprisingly, they are not cointegrated since the ADF test implementation from Section 2.3.1 yields:

```
ADF Statistic: -2.2307
p-value: 0.1953
Number of lags: 22
The residuals are not stationary (fail to reject null hypothesis) at the 5.0%
significance level.
```

Reversing the pair order from (Y_t =MSFT/X_t =AAPL) does not help either:

```
ADF Statistic: -1.9822
p-value: 0.2944
Number of lags: 20
```

```
The residuals are not stationary (fail to reject null hypothesis) at the 5.0%
significance level.
```
These results suggest that MSFT and AAPL do not exhibit the cointegration necessary for the described pairs trading strategy. However, if we were to proceed with such a strategy despite the lack of cointegration, Fig. 3.18 illustrates that the result would not be optimal.



(a) Prices & residuals for Y_t=AAPL, X_t=MSFT

(b) PnL evolution for Y_t=AAPL, X_t=MSFT

(c) Prices & residuals for Y_t=MSFT, X_t=AAPL

(d) PnL evolution for Y_t=MSFT, X_t=AAPL

Figure 3.18: Pairs trading strategy for (AAPL/MSFT) and (MSFT/AAPL). For this example, the order of the shares significantly impacts the strategy performance. While the former pair would have been quite profitable, the latter produced only losses. Besides, from a cointegration perspective, these asset pairs would not be suitable. The heavy decline on the unrealized PnL in Fig. 3.18d comes from the position opened in September 2023 (marked with red dot in 3.18c). Would this be a real-world investment, the investor would still be waiting for the spread to revert to the long-term equilibrium, while only sitting on unrealized losses so far ...

Even though MSFT and AAPL are highly correlated, the lack of cointegration suggests that they do not maintain a stable long-term equilibrium relationship. Hence, they may not be suitable for pairs trading, where mean-reversion of the residuals is the driver for profitability.

3.2.3 Nestlé and Roche

Two of the most dominant Swiss stocks are Nestlé (NESN.SW) and the pharmaceutical company Roche (ROG.SW). To mitigate the impact of recent market regime changes which had affected previous analyses on pairs like (KO/PEP), (GOOGL/AMZN), and (AAPL/MSFT), a longer history of prices was used, starting from 2019-01-01 onwards. Earlier data was not available from the data provider `yfinance`. However, as Fig. 3.19 shows, avoiding a regime change completely was not entirely successful.

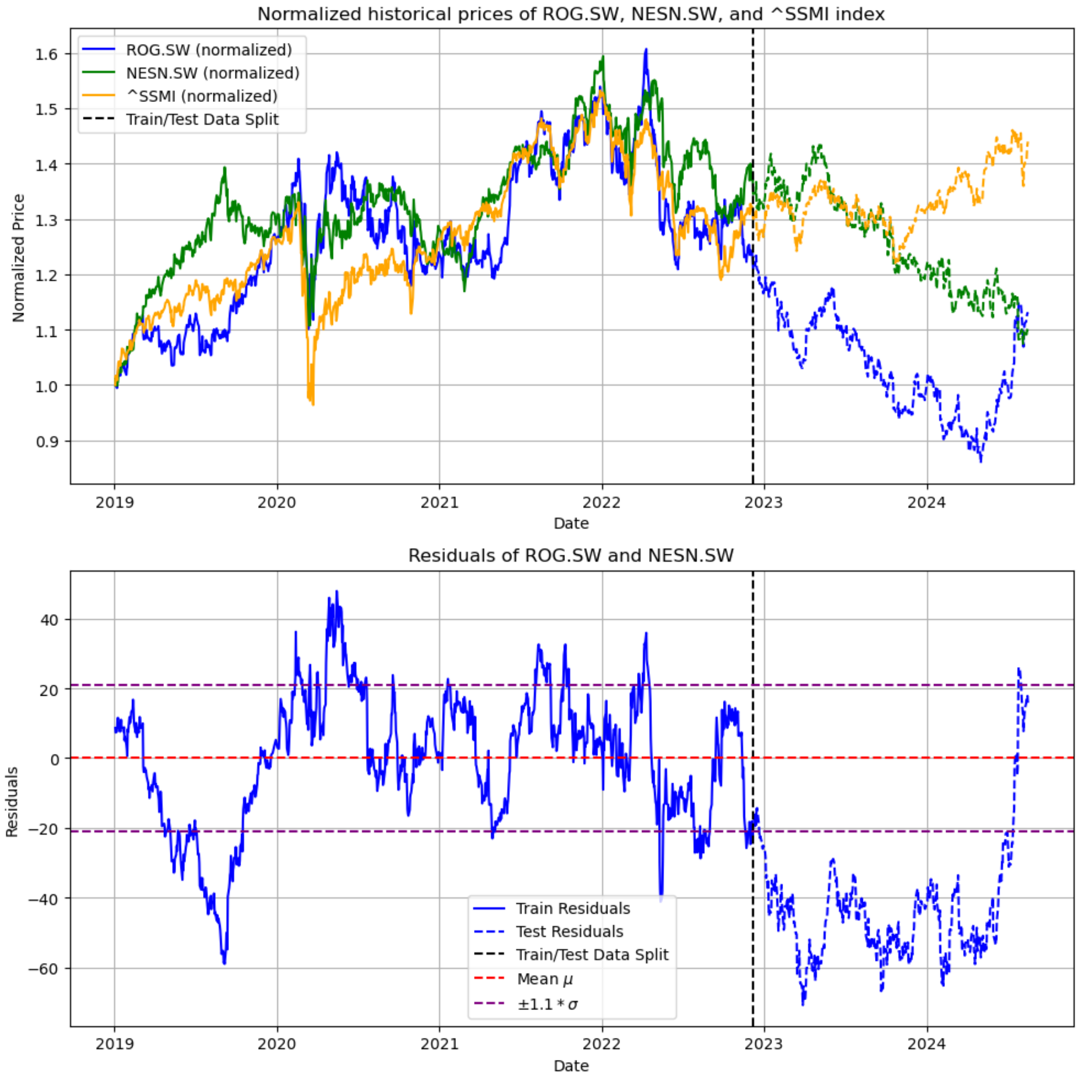
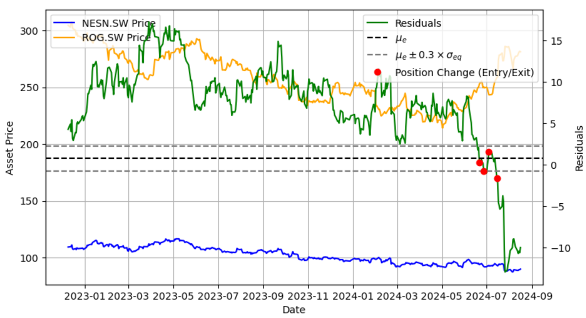


Figure 3.19: Residuals and asset prices plotted over training and test period. Unfortunately, the test residuals have a visibly lower mean than the training residuals had displayed.

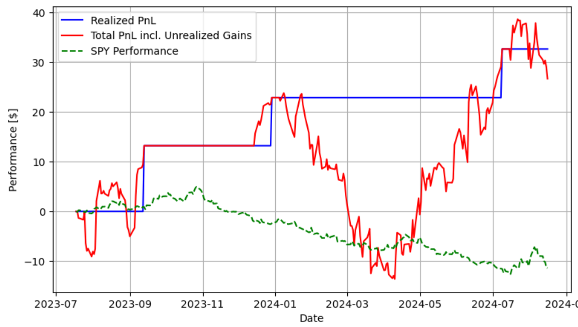
The ADF test confirms their cointegration at 5% significance with p-values 0.0193 for (NESN.SW/ROG.SW) and 0.0439 for (ROG.SW/NESN.SW). The long-term mean for (Y_t =NESN.SW/X_t =ROG.SW) is found at $\mu_e = 0.7425$ with a half-life of 31.31 trading days. The reversed pair has a long-term mean at $\mu_e = -0.0006$ with a slightly longer half-life of 34.43 days.

While the pairs trading strategies constructed for both pairs manage to outperform the benchmark index during a relatively sideways-trending period of the Swiss Market Index (Fig. 3.20b, 3.20d), it does take a considerable amount of time for the residuals to correct their deviations back to the long-term equilibrium, leading to accumulation of significant unrealized losses along the way. Although those losses do not materialize over the observed time window since the position is able to close profitably, they could potentially trigger early exits for more risk-averse traders using a manual stop-loss.

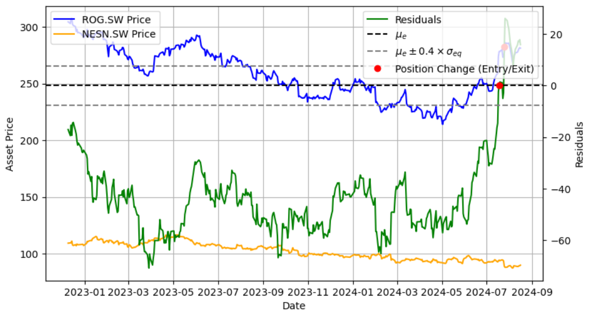
We conclude from Fig. 3.20 that this pair shows potential for outperforming the benchmark index, but this example also highlights the risks associated with ever-changing market dynamics and the impact of prolonged reversion periods. This emphasizes the need for careful monitoring and stop-loss possibilities.



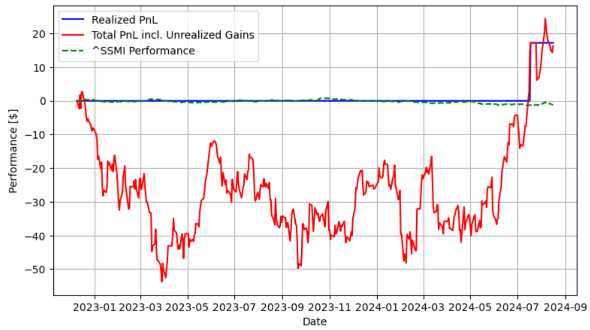
(a) Prices & residuals for Y_t=NESN, X_t=ROG



(b) PnL evolution for Y_t=NESN, X_t=ROG



(c) Prices & residuals for Y_t=ROG, X_t=NESN



(d) PnL evolution for Y_t=ROG, X_t=NESN

Figure 3.20: Pairs trading strategy for (NESN.SW/ROG.SW) and (ROG.SW/NESN.SW).

3.2.4 Gold Price and Gold Futures

In this section, we extend our cointegration study to another asset class: precious metals and commodities. Commodity prices are often highly correlated and are frequently considered to be cointegrated with their respective futures prices. We analyze the gold share price (GLD), a physically backed gold ETF, against gold futures (GC=F), using data extending back to 2019-01-01. Fig. 3.21 illustrates how closely the gold share and future prices are linked over time. As a traditional safe-haven asset, gold typically experiences increased demand during periods of market volatility, which becomes evident when comparing these assets to the S&P 500 index. Unlike gold, the index shows significant price divergence, e.g., during the 2020 market crash and subsequent recovery, highlighting gold's role as a stabilizing force during economic turbulence, even when the broader US equity market, represented by S&P 500, is in turmoil.

The gold/gold future residuals display very frequent, short-lived deviations from equilibrium, illustrated by the high number of oscillations in Fig. 3.21. These oscillations present potential opportunities for pairs trading, as the residuals tend to mean-revert quickly in this example. However, this pair is not sufficiently cointegrated according to Table 2.1, with an ADF statistic of -1.7108 for (Y_t=GLD, X_t=GC=F) and -1.6770 for the reversed pair. Although Fig. 3.21 clearly shows mean-reverting behavior in the residual paths, they do not follow a stationary process with significant but fleeting deviations from the mean. Despite this, one could still exploit these fast-paced deviations, visualized as narrow spikes in the residuals, by following a pairs trading strategy as described subsequently.

We employ the same methods used in a traditional cointegration pairs trading strategy, modeling the mean-reverting non-stationary spread using an Ornstein-Uhlenbeck process with estimated parameters $\theta, \mu_e, \sigma_{OU}$. From the high $\theta = 0.4386$, we deduce a very short half-life of 1.58 trading days for the pair (GLD/GC=F). Figs. 3.22a & 3.22c show that

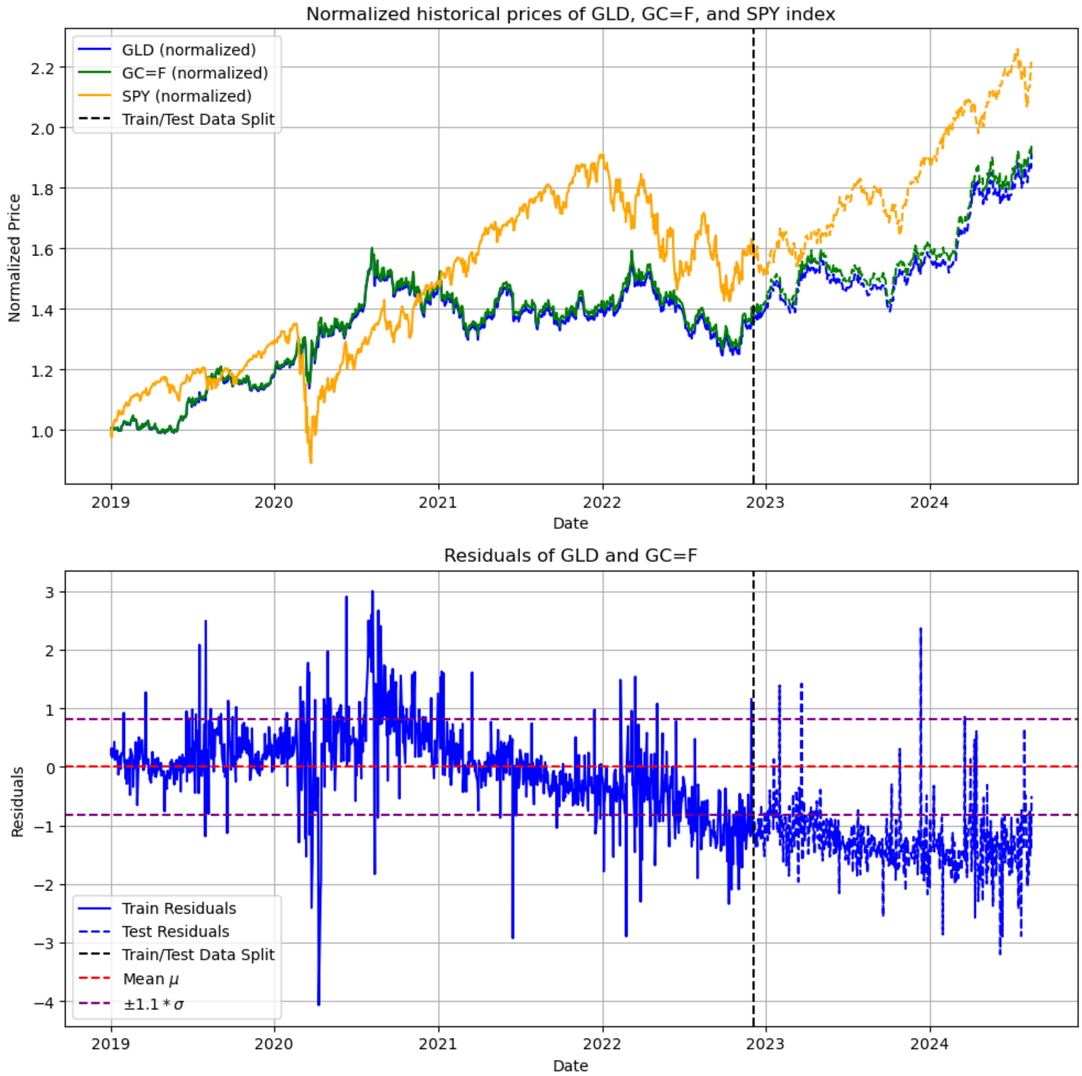
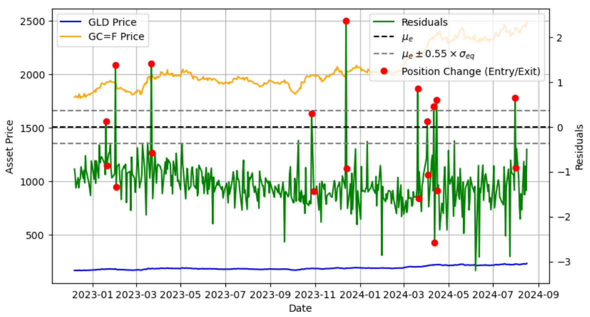
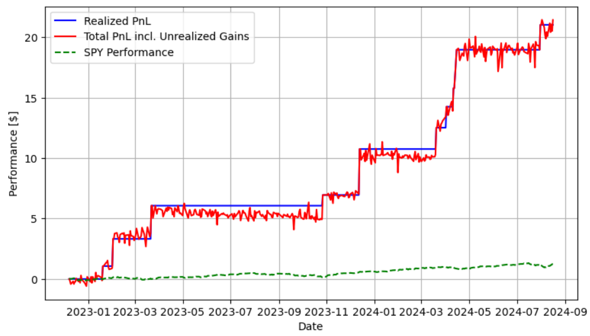


Figure 3.21: Residuals and gold share and gold future prices plotted over training and test period. The S&P 500 index shows some dissimilar price dynamics compared to the gold assets which often attract investors in times of financial turbulence.

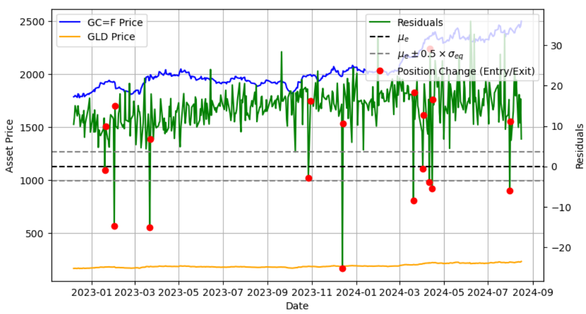
the residual process is almost always below or almost always above the equilibrium band $\mu_e \pm Z\sigma_{eq}$. The economic interpretation of the residuals consistently remaining below or above the signal band is identical in both cases: it indicates that in both scenarios, the gold shares GLD are undervalued relative to the future prices. Consequently, in both strategy configurations, GLD is always the long position, while GC=F is shortened at different hedge ratios. This is shown in Fig. 3.23. When X_t =GC=F is the independent variable, the resulting hedge ratio equals $\beta_{Y_t=GLD} = 0.0914$. Conversely, when GLD is the independent variable, the hedge ratio obtained is $\beta_{X_t=GLD} = 10.9156$.
Performance-wise, the observed strategy dynamics in Fig. 3.22 reveal very similar trends in PnL evolution for both pair orders, yet the absolute returns differ significantly by a factor of 10, depending on the role of the dependent and independent variables in the OLS regression, as depicted in Figs. 3.22b & 3.22d. This asymmetry arises because the OLS regression used to obtain the residuals and hedge ratios is not fully symmetric. For asset pairs with significant price disparities, such as gold futures (GC=F) trading around levels of \$2000 $\pm$ \$500 and GLD trading in a range of \$130 to \$230 during the observed periods, this asymmetry can become more pronounced. In this case, the different absolute



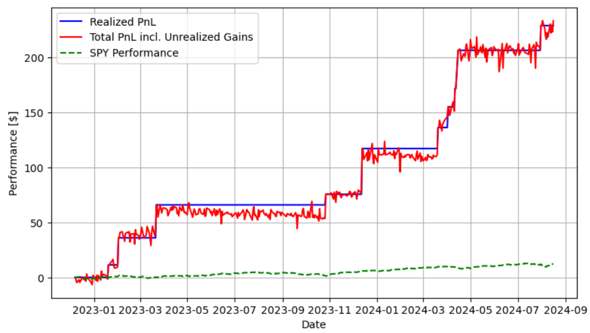
(a) Prices & residuals for Y_t=GLD, X_t=GC=F



(b) PnL evolution for Y_t=GLD, X_t=GC=F

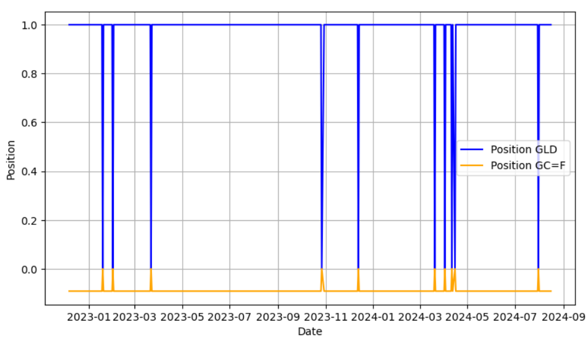


(c) Prices & residuals for Y_t=GC=F, X_t=GLD

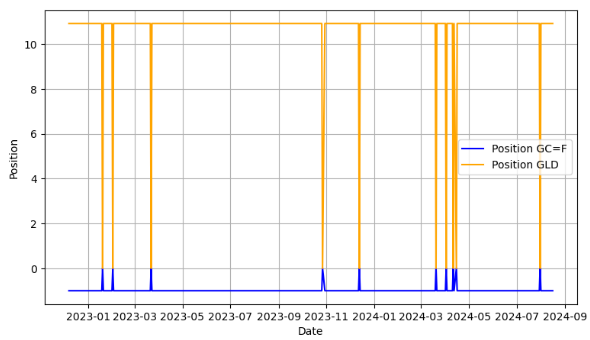


(d) PnL evolution for Y_t=GC=F, X_t=GLD

Figure 3.22: Pairs trading strategy for (GLD/GC=F) and (GC=F/GLD).



(a) Positions taken in Y_t=GLD, X_t=GC=F



(b) Positions taken in Y_t=GC=F, X_t=GLD

Figure 3.23: The positions the pairs trading strategy for (GLD/GC=F) and (GC=F/GLD) would effectively be opening during test phase, based on the parameters and hedge ratios estimated from the training phase.

PnL returns were observed because the initial investment in the portfolio was different - at around \$3.0246 for Y_t=GLD, X_t=GC=F and at around \$30.09889 for the role-reversed pair.
Overall, both strategies perform favorably over the test period. The difference in absolute PnL returns stems from the different initial investment values. Both asset constellations are suitable for pairs trading, although their residual process is not stationary according to the ADF test. Despite the lack of strong cointegration, the presence of frequent mean-reversion allowed pairs trading to effectively capitalize on market inefficiencies, providing a profitable trading strategy nonetheless.

A Python Code

The example code snippets most suitable for execution of real-world analysis are found in Appendix A.4.
A Jupyter notebook demonstrating various cases with corresponding visualizations is also available.

A.1 Auxiliary Code for Plots

```
import matplotlib.pyplot as plt
import numpy as np


def plot_assets_and_residuals(train_data, test_data, ticker1, ticker2,
    ↪ index_ticker="SPY", split_ratio=0.7):
    plt.figure(figsize=(10, 10))

    # normalize historical prices for train data
    normalized_train_ticker1 = train_data[ticker1] / train_data[ticker1
        ↪ ].iloc[0]
    normalized_train_ticker2 = train_data[ticker2] / train_data[ticker2
        ↪ ].iloc[0]
    normalized_train_index = train_data[index_ticker] / train_data[
        ↪ index_ticker].iloc[0]

    # normalize historical prices for test data
    normalized_test_ticker1 = test_data[ticker1] / train_data[ticker1].
        ↪ iloc[0]
    normalized_test_ticker2 = test_data[ticker2] / train_data[ticker2].
        ↪ iloc[0]
    normalized_test_index = test_data[index_ticker] / train_data[
        ↪ index_ticker].iloc[0]

    # determine the split date
    split_date = train_data.index[-1]

    # plot normalized prices
    plt.subplot(2, 1, 1)
    plt.plot(train_data.index, normalized_train_ticker1, label=f"{
        ↪ ticker1} (normalized)", color="blue")
    plt.plot(train_data.index, normalized_train_ticker2, label=f"{
        ↪ ticker2} (normalized)", color="green")
    plt.plot(train_data.index, normalized_train_index, label=f"{
        ↪ index_ticker} (normalized)", color="orange")

    plt.plot(test_data.index, normalized_test_ticker1, color="blue",
        ↪ linestyle="--")
```

```python
    plt.plot(test_data.index, normalized_test_ticker2, color="green", linestyle="--")
    plt.plot(test_data.index, normalized_test_index, color="orange", linestyle="--")

    plt.axvline(split_date, color="black", linestyle="--", label="Train/Test Data Split")
    plt.title(f"Normalized historical prices of {ticker1}, {ticker2}, and {index_ticker} index")
    plt.xlabel("Date")
    plt.ylabel("Normalized Price")
    plt.legend()
    plt.grid(True)

    # plot residuals for train and test data
    plt.subplot(2, 1, 2)
    plt.plot(train_data.index, train_data['residuals'], label="Train Residuals", color="blue")
    plt.plot(test_data.index, test_data['residuals'], label="Test Residuals", color="blue", linestyle="--")

    plt.axvline(split_date, color="black", linestyle="--", label="Train/Test Data Split")
    mean = train_data['residuals'].mean()
    stdev = train_data['residuals'].std()
    plt.axhline(mean, color="r", linestyle='--', label=f"Mean $\mu$")
    plt.axhline(mean + 1.1 * stdev, color="purple", linestyle="--", label="$\pm1.1*\sigma$")
    plt.axhline(mean - 1.1 * stdev, color="purple", linestyle="--")

    plt.title(f"Residuals of {ticker1} and {ticker2}")
    plt.xlabel("Date")
    plt.ylabel("Residuals")
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()


def compare_ecm_residuals(data, ecm_results):
    plt.figure(figsize=(15, 8))
    # original residuals
    plt.plot(data.index, data['residuals'], label="Equilibrium residuals $u_t$", color="blue")
    # align indices for lagged ECM residuals
    plt.plot(data.index[1:], ecm_results['residuals'], label="ECM residuals $\epsilon_t$", color="orange")
    plt.axhline(0, color="black", linestyle="--")
    plt.title("Comparison of Residuals from Engle-Granger Method")
    plt.xlabel("Date")
    plt.ylabel("Residuals")
    plt.legend()
    plt.show()


def simulate_ou_process(theta, mu_e, sigma_ou, initial_value, num_steps, dt=1):
    """Simulate an Ornstein-Uhlenbeck process."""
```

```python
    ou_process = np.zeros(num_steps)
    ou_process[0] = initial_value
    for t in range(1, num_steps):
    ou_process[t] = ou_process[t-1] + theta * (mu_e - ou_process[t-1]) *
        ↪ dt + sigma_ou * np.sqrt(dt) * np.random.normal()
    return ou_process


def plot_ou_process_and_residuals(data, theta, mu_e, sigma_ou):
    """Plot simulated OU process against actual residuals."""
    # Simulate an OU process with the estimated parameters
    num_steps = len(data['residuals'])
    initial_value = data['residuals'].iloc[0]
    simulated_residuals = simulate_ou_process(theta, mu_e, sigma_ou,
        ↪ initial_value, num_steps)

    # Plot the actual residuals and the simulated OU process
    plt.figure(figsize=(14, 8))
    plt.plot(data.index, data['residuals'], label="Actual Residuals
        ↪ $e_t$")
    plt.plot(data.index, simulated_residuals, label="Simulated OU
        ↪ Process", linestyle="-.")
    plt.xlabel("Date")
    plt.ylabel("Residuals")
    plt.title("Actual Residuals vs. Simulated Ornstein-Uhlenbeck Process
        ↪ ")
    plt.legend()
    plt.grid(True)
    plt.show()


def plot_pnl_table(pnl_table, best_z, best_pnl, pnl_key):
    """Plot the PnL values achieved for different Z-values. Mark the
        ↪ best Z-value with a star."""
    plt.figure(figsize=(10, 6))
    plt.plot(pnl_table['Z'], pnl_table[pnl_key], marker="o", linestyle="
        ↪ -", color="b", label="PnL vs Z")

    # Highlight the best PnL point with a star
    plt.scatter(best_z, best_pnl, color='r', marker="*", s=100, zorder
        ↪ =5,
    label=f"Best PnL: {best_pnl:.2f} at Z={best_z:.2f}")

    # Add text annotation to best PnL
    plt.text(best_z, best_pnl, f"  Z={best_z:.2f}\n  PnL={best_pnl:.2f}"
        ↪ , color="r", bbox=dict(facecolor="white"))

    plt.title(f"{pnl_key} vs Z")
    plt.xlabel("Z")
    plt.ylabel(f"{pnl_key}")
    plt.grid(True)
    plt.show()


def plot_positions(portfolio):
    """Plot the positions of ticker1 and ticker2 over time."""
    dates = portfolio.data.index

    plt.figure(figsize=(14, 6))
```

```
    plt.axhline(0, color="r", linestyle="--", label="no position")
    plt.plot(dates, portfolio.positions[portfolio.ticker1],
             label=f"Position {portfolio.ticker1}", color="blue")
    plt.plot(dates, portfolio.positions[portfolio.ticker2],
             label=f"Position {portfolio.ticker2}", color="orange")
    plt.xlabel("Date")
    plt.ylabel("Position")
    plt.title(f"Positions for {portfolio.ticker1} and {portfolio.ticker2} Over Time")
    plt.legend()
    plt.grid(True)
    plt.show()


def plot_asset_prices_and_residuals(portfolio):
    """Plot the asset prices and residuals with sigma_eq-bands using subplots."""
    dates = portfolio.data.index

    fig, ax1 = plt.subplots(figsize=(14, 6))

    # primary y-axis: asset prices
    ax1.plot(dates, portfolio.data[portfolio.ticker1], label=f"{portfolio.ticker1} Price", color="blue")
    ax1.plot(dates, portfolio.data[portfolio.ticker2], label=f"{portfolio.ticker2} Price", color="orange")
    ax1.set_xlabel("Date")
    ax1.set_ylabel("Asset Price")
    ax1.legend(loc="upper left")
    ax1.grid(True)

    # secondary y-axis: residuals
    residuals = pd.Series(portfolio.data['residuals'], index=dates)
    ax2 = ax1.twinx()
    ax2.plot(dates, residuals, label="Residuals", color="green")
    ax2.axhline(portfolio.mu_e, color="black", linestyle="--", label=r"$\mu_e$")
    upper_bound, lower_bound = portfolio.calculate_optimal_bounds()
    sigma_band_label = r"$\mu_e \pm " + str(round(portfolio.z, 2)) + r" \times \sigma_{eq}$"
    ax2.axhline(upper_bound, color="grey", linestyle="--", label=sigma_band_label)
    ax2.axhline(lower_bound, color="grey", linestyle="--")
    ax2.set_ylabel("Residuals")

    # mark entry and exit signals based on changes in positions
    positions_ticker1 = portfolio.positions[portfolio.ticker1]
    position_changes = positions_ticker1.diff().fillna(0)
    signals = position_changes != 0  # track where a position is changed
    ax2.plot(dates[signals], residuals[signals], "ro", label="Position Change (Entry/Exit)")
    ax2.legend(loc="upper right")

    plt.title(f"Asset Prices and Residuals with Thresholds for {portfolio.ticker1} and {portfolio.ticker2}")
    plt.show()


def get_portfolio_investment(portfolio):
```

```python
    """Calculate the initial dollar investment in the portfolio."""
    ticker1 = portfolio.ticker1
    ticker2 = portfolio.ticker2

    # boolean series for non-zero positions in ticker1 and ticker2
    condition_ticker1 = portfolio.positions[ticker1] != 0
    condition_ticker2 = portfolio.positions[ticker2] != 0
    combined_condition = np.logical_or(condition_ticker1, condition_ticker2)

    # find index of the first non-zero position
    first_trade_date_idx = portfolio.positions[combined_condition].index[0]

    # entry prices at first trade date
    entry_price_ticker1 = portfolio.data.loc[first_trade_date_idx, ticker1]
    entry_price_ticker2 = portfolio.data.loc[first_trade_date_idx, ticker2]

    # positions of first trade date
    position_ticker1 = portfolio.positions.loc[first_trade_date_idx, ticker1]
    position_ticker2 = portfolio.positions.loc[first_trade_date_idx, ticker2]
    return entry_price_ticker1 * position_ticker1 + entry_price_ticker2 * position_ticker2


def plot_pnl_against_index(portfolio, index_ticker):
    """Plot the portfolio PnL compared to index performance."""
    realized_pnl = portfolio.realized_daily_pnl
    total_pnl = portfolio.unrealized_daily_pnl + portfolio.realized_daily_pnl
    dates = portfolio.data.index

    # calculate cumulative index return and scale to portfolio investment
    index_prices = portfolio.data[index_ticker]
    initial_index_price = index_prices.iloc[0]
    normalized_index_return = (index_prices - initial_index_price) / initial_index_price
    index_return_scaled = normalized_index_return * get_portfolio_investment(portfolio)

    plt.figure(figsize=(14, 6))
    plt.plot(dates, realized_pnl, label="Realized PnL", color="blue")
    plt.plot(dates, total_pnl, label="Total PnL incl. Unrealized Gains", color="red")
    plt.plot(dates, index_return_scaled, label=f"{index_ticker} Performance", color="green", linestyle="--")
    plt.title(f"PnL Evolution over Test Data Period - compared against {index_ticker} index")
    plt.xlabel("Date")
    plt.ylabel("Performance [$]")
    plt.legend()
    plt.grid(True)
    plt.show()
```

A.2 Auxiliary Code for Pairs Trading Strategy Design and Portfolio Risk Measures

```python
import numpy as np
import pandas as pd


class Portfolio:
    def __init__(self, data, ticker1, ticker2, ou_params, hedge_ratio, z):
        self.data = data
        self.ticker1 = ticker1
        self.ticker2 = ticker2
        self.theta = ou_params['theta']
        self.sigma_ou = ou_params['sigma_ou']
        self.mu_e = ou_params['mu_e']
        self.hedge_ratio = hedge_ratio
        self.z = z
        self.realized_daily_pnl = pd.Series(index=data.index, dtype=float).fillna(0)
        self.unrealized_daily_pnl = pd.Series(index=data.index, dtype=float).fillna(0)
        self.positions = pd.DataFrame(index=data.index, columns=[ticker1, ticker2]).fillna(0)
        self.returns = []
        self.manage_positions()

    def calculate_optimal_bounds(self):
        """Calculate the upper and lower bounds for trading, CQF FP Workshop 2, sl. 15."""
        sigma_eq = self.sigma_ou / np.sqrt(2 * self.theta)
        bound1 = self.mu_e + self.z * sigma_eq
        bound2 = self.mu_e - self.z * sigma_eq
        return tuple(sorted([bound1, bound2], reverse=True))  # return upper bound first, then lower bound


    def enter_position(self, row, position_ticker1, position_ticker2):
        """Enter a position based on the current market conditions."""
        entry_price_ticker1 = row[self.ticker1]
        entry_price_ticker2 = row[self.ticker2]
        return position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2

    def calculate_trade_pnl(self, row, position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2):
        """Calculate the PnL of the trade."""
        if position_ticker1 == 1 and position_ticker2 == -self.hedge_ratio:
            trade_pnl = (row[self.ticker1] - entry_price_ticker1) + (entry_price_ticker2 - row[self.ticker2]) * self.hedge_ratio
        elif position_ticker1 == -1 and position_ticker2 == self.hedge_ratio:
            trade_pnl = (entry_price_ticker1 - row[self.ticker1]) + (row[self.ticker2] - entry_price_ticker2) * self.hedge_ratio
        else:
```

```
            trade_pnl = 0
        return trade_pnl

    def calculate_unrealized_pnl(self, row, position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2):
        """Calculate the unrealized PnL for the day based on the movement of ticker1 and ticker2."""
        if position_ticker1 != 0 and position_ticker2 != 0:  # If a position is open
            unrealized_pnl = self.calculate_trade_pnl(row, position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2)
        else:
            unrealized_pnl = 0  # No unrealized PnL if no position is open
        return unrealized_pnl

    def append_return(self, trade_pnl, entry_price_ticker1, entry_price_ticker2):
        """Append the return (either simple or log) to the self.returns list."""
        simple_return = trade_pnl / (entry_price_ticker1 + self.hedge_ratio * entry_price_ticker2)
        self.returns.append(simple_return)

    def close_position(self, row, position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2):
        """Close the position and calculate realized PnL."""
        trade_pnl = self.calculate_trade_pnl(row, position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2)
        if trade_pnl != 0:
            self.append_return(trade_pnl, entry_price_ticker1, entry_price_ticker2)
            self.realized_daily_pnl.at[row.name] += trade_pnl  # add realized PnL
        return 0, 0, 0, 0  # reset positions and entry prices

    def manage_positions(self):
        """Manage positions for the trading strategy exploiting mean-reversion of 2 cointegrated assets using
        hedge ratio (beta1) previously obtained in Engle-Granger step 1 and track daily PnL."""
        position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2 = 0, 0, 0, 0
        upper_bound, lower_bound = self.calculate_optimal_bounds()

        # initialize the first value of realized PnL to 0 - will be needed later in the loop to be carried forward
        previous_realized_pnl = 0

        for index, row in self.data.iterrows():
            residual = row['residuals']

            # entry conditions
            if position_ticker1 == 0 and position_ticker2 == 0:
                if residual > upper_bound:  # very positive spread
                    # short ticker1 (over-valued from equilibrium), long ticker2
```

```
                    position_ticker1, position_ticker2 = -1, self.hedge_ratio
                elif residual < lower_bound:  # very negative spread
                    # long ticker1 (under-valued from equilibrium), short ticker2
                    position_ticker1, position_ticker2 = 1, -self.hedge_ratio
                position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2 = \
                    self.enter_position(row, position_ticker1, position_ticker2)

            # exit conditions -> close positions
            elif (position_ticker1 == 1 and position_ticker2 == -self.hedge_ratio and residual >= self.mu_e) or \
                    (position_ticker1 == -1 and position_ticker2 == self.hedge_ratio and residual <= self.mu_e):
                position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2 = \
                    self.close_position(row, position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2)

            # calculate unrealized PnL for the day
            unrealized_pnl = self.calculate_unrealized_pnl(row, position_ticker1, position_ticker2, entry_price_ticker1, entry_price_ticker2)
            self.unrealized_daily_pnl.at[index] = unrealized_pnl

            # carry forward realized PnL
            self.realized_daily_pnl.at[index] += previous_realized_pnl
            previous_realized_pnl = self.realized_daily_pnl.at[index]

            # store the positions for this date
            self.positions.at[index, self.ticker1] = position_ticker1
            self.positions.at[index, self.ticker2] = position_ticker2

    def get_cumulative_pnl(self):
        """Return cumulative realized PnL."""
        return self.realized_daily_pnl.iloc[-1]

    def get_total_pnl(self):
        """Return total PnL, combining realized and unrealized PnL."""
        return self.realized_daily_pnl.add(self.unrealized_daily_pnl, fill_value=0).iloc[-1]


class RiskMetrics:
    def __init__(self, returns):
        self.returns = returns

    def calculate_var(self, confidence_level=0.95):
        """Calculate Value at Risk (VaR) at the given confidence level."""
        if len(self.returns) > 0:
            var = np.percentile(self.returns, (1 - confidence_level) * 100)
        else:
            var = 0
```

```
        return var

    def calculate_expected_shortfall(self, confidence_level=0.95):
        """Calculate Expected Shortfall (ES) at the given confidence level."""
        var = self.calculate_var(confidence_level)
        if len(self.returns) > 0:
            expected_shortfall = np.mean([r for r in self.returns if r < var])
        else:
            expected_shortfall = 0
        return expected_shortfall

    def run_full_analysis(self):
        return {'VaR': self.calculate_var(),
                'ES': self.calculate_expected_shortfall()}


def find_best_pnl(pnl_table, pnl_key):
    """In a pnl_table dataframe, find the highest PnL value and its corresponding Z"""
    best_row = pnl_table.loc[pnl_table[pnl_key].idxmax()]
    return best_row['Z'], best_row[pnl_key]


def backtest_strategy_for_z_values(data, ticker1, ticker2, ou_params, hedge_ratio, z_values,
                                   plotting=False, maximize_realized=True):
    """Test Z values in range of z_values (iterable) and calculate Pnl for every Z value."""
    results = []
    for z in z_values:
        portfolio = Portfolio(data, ticker1, ticker2, ou_params, hedge_ratio, z)
        realized_pnl = portfolio.get_cumulative_pnl()
        total_pnl = portfolio.get_total_pnl()
        risk_metrics = RiskMetrics(portfolio.returns)
        metrics = risk_metrics.run_full_analysis()
        results.append({'Z': z, 'Realized PnL': realized_pnl, 'Total PnL': total_pnl, **metrics})
    results_df = pd.DataFrame(results)
    pnl_key = "Realized PnL" if maximize_realized else "Total PnL"
    best_z, best_pnl = find_best_pnl(results_df, pnl_key)
    if plotting:
        plot_pnl_table(results_df, best_z, best_pnl, pnl_key)
    return results_df, best_z
```

A.3 Auxiliary Code for Cointegration Analysis

```
# Ignore warnings
import warnings

warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
```

```python
import yfinance as yf
from scipy.optimize import minimize
from scipy.stats import norm
from statsmodels.tsa.stattools import adfuller


def download_data(ticker, start_date="2019-01-01"):
    """Download ticker data from yfinance library."""
    df = yf.download(ticker, start=start_date)
    df.dropna(inplace=True)
    return df


def prepare_time_series(data1, data2, ticker1, ticker2, index_ticker):
    """Prepare the time series data of historical prices to use later for cointegration analysis."""
    index_data = download_data(index_ticker)  # download index data

    # among the 3 dataframes: determine latest start date and trim all dataframes to start there
    latest_start_date = max(data1.index.min(), data2.index.min(), index_data.index.min())
    data1 = data1[data1.index >= latest_start_date]
    data2 = data2[data2.index >= latest_start_date]
    index_data = index_data[index_data.index >= latest_start_date]

    # combine the 3 trimmed dataframes into 1
    data = pd.concat([data1['Close'], data2['Close'], index_data['Close']], axis=1).dropna()
    data.columns = [ticker1, ticker2, index_ticker]
    return data


def split_data(data, split_ratio=0.8):
    """Splits the input data into training and testing subsets based on the provided split ratio."""
    split_index = int(len(data) * split_ratio)
    train_data = data.iloc[:split_index]
    test_data = data.iloc[split_index:]
    return train_data, test_data


def calculate_test_residuals(data, beta, ticker1, ticker2):
    """Calculate the residuals for a given dataset using the provided beta vector."""
    y = data[ticker1].values
    X = data[ticker2].values
    X_with_intercept = np.hstack([np.ones((X.shape[0], 1)), X.reshape(-1, 1)])
    residuals = y - X_with_intercept @ beta
    return pd.Series(residuals, index=data.index)


def least_squares_regression(y, X):
    """Perform least squares regression to obtain beta coefficients and residuals."""
    X = np.hstack([np.ones((X.shape[0], 1)), X])  # add y-intercept to X
    beta = np.linalg.inv(X.T @ X) @ (X.T @ y)  # least squares regression for beta
```

```python
    residuals = y - X @ beta
    return beta, residuals


def perform_adf_test(residuals, significance_level, maxlag=None):
    """Perform the Augmented Dickey-Fuller (ADF) test to check for the presence of a unit root in a time series.
    H0: time series has a unit root (i.e. non-stationary)"""
    adf_test = adfuller(residuals, maxlag=maxlag, autolag=None)
    # autolag=None will set lag nbr to maxlag (no lag optimization) if maxlag is not None
    # if maxlag and autolag are both None, then lag nbr will be optimized using AIC
    adf_statistic, p_value, lags = adf_test[0], adf_test[1], adf_test[2]

    print(f"ADF Statistic: {adf_statistic:.4f}")
    print(f"p-value: {p_value:.4f}")
    print(f"Number of lags: {lags}")

    if p_value < significance_level:
        print(f"The residuals are stationary (reject null hypothesis) "
              f"at the {significance_level * 100}% significance level.")
    else:
        print(f"The residuals are not stationary (fail to reject null hypothesis) "
              f"at the {significance_level * 100}% significance level.")
    return adf_test


def perform_engle_granger_step1(ticker1, ticker2, index_ticker, train_data, test_data,
                                plotting, significance_level, maxlag):
    """Step1 of the Engle-Granger procedure."""

    # OLS regression to obtain regression coefficients beta & residuals
    y = train_data[ticker1].values
    X = train_data[ticker2].values.reshape(-1, 1)
    beta, residuals = least_squares_regression(y, X)
    train_data['residuals'] = residuals
    test_data['residuals'] = calculate_test_residuals(test_data, beta, ticker1, ticker2)

    if plotting:  # plot normalized asset prices and residuals
        plot_assets_and_residuals(train_data, test_data, ticker1, ticker2, index_ticker)

    # perform ADF test
    adf_test_result = perform_adf_test(train_data['residuals'], significance_level, maxlag)
    return train_data, test_data, beta, adf_test_result


def get_differences(data, columns):
    """Calculate the returns (differences) Delta y_t = y_t-y_{t-1} for the specified columns in the dataframe."""
    return data[columns].diff().dropna()


def fit_ecm(data, target_column, independent_column, lag_size=1):
```

```python
    """Step2 of the Engle-Granger procedure: fit the Equilibrium Correction Model (ECM)."""
    data_delta = get_differences(data, [target_column, independent_column])
    data_delta['lagged_residuals'] = data['residuals'].shift(lag_size) # lag the residuals
    data_delta = data_delta.dropna()

    # OLS to obtain ECM coefficients & residuals
    y = data_delta[target_column].values
    X = data_delta[[independent_column, "lagged_residuals"]].values
    ecm_coefficients, ecm_residuals = least_squares_regression(y, X)

    ecm_residuals = pd.DataFrame(ecm_residuals, index=data_delta.index, columns=["ECM_residuals"])  # convert to pd.df
    return {'coefficients': ecm_coefficients, 'residuals': ecm_residuals}


def ou_likelihood(params, residuals, dt):
    """Calculates the negative log-likelihood of an Ornstein-Uhlenbeck process"""
    theta, mu_e, sigma_ou = params
    likelihood = 0
    for t in range(1, len(residuals)):
        mean = residuals[t-1] + theta * (mu_e - residuals[t-1]) * dt
        variance = sigma_ou**2 * dt
        # increment the log-likelihood by normal log-pdf of the next residual using mean and variance
        likelihood += norm.logpdf(residuals[t], loc=mean, scale=np.sqrt(variance))
    return -likelihood


def estimate_ou_params(residuals, dt=1):  # dt = 1: daily prices, so usually time increment dt = 1
    """Estimate Ornstein-Uhlenbeck process parameters using maximum likelihood estimation.
    The OU process is given as: d(residuals)_t = -theta (residuals_t - mu_e) dt + sigma_ou dW_t"""
    residuals = np.array(residuals)
    initial_params = [0.1, np.mean(residuals), np.std(residuals)]  # [theta0, mu_ou0, sigma_ou0]
    # we minimize negative log-likelihood, which is equivalent to using maximum likelihood estimator (MLE)
    result = minimize(ou_likelihood, initial_params, args=(residuals, dt), method="L-BFGS-B")
    theta, mu_e, sigma_ou = result.x
    return theta, mu_e, sigma_ou


def get_half_life(theta, dt=1):
    """Calculate the half-life of an Ornstein-Uhlenbeck process."""
    half_life = np.log(2) / (theta * dt)
    return half_life
```

A.4 Example Case Studies

The following code contains the main implementation for the cointegration analysis (based on code from Appendix A.3) and designs the pairs trading strategy parameters (based on code from Appendix A.2). Plots are generated using the functions defined in Appendix A.1.

```
def analyze_cointegration(ticker1, ticker2, index_ticker="SPY",
                          plotting=False, start_date="2019-01-01",
                          significance_level=0.05, maxlag=None):
    """Analyze cointegration between two assets ticker1 & ticker2 after start_date <YYYY-MM-DD>."""
    print(f"-" * 100)
    print(f"Analyzing cointegration between {ticker1} and {ticker2}...")

    df1 = download_data(ticker1, start_date)
    df2 = download_data(ticker2, start_date)
    data = prepare_time_series(df1, df2, ticker1, ticker2, index_ticker)

    # test / train split:
    train_data, test_data = split_data(data, split_ratio=0.7)

    # Engle-Granger procedure - Step 1
    train_data, test_data, beta, adf_test_result = perform_engle_granger_step1(
        ticker1, ticker2, index_ticker,
        train_data, test_data, plotting,
        significance_level, maxlag)
    # Engle-Granger procedure - Step 2: ECM
    lag_size = adf_test_result[2]
    ecm_results = fit_ecm(train_data, ticker1, ticker2, lag_size)
    print(f"Equilibrium mean-reversion coefficient: {ecm_results['coefficients'][-1]:2f}")

    # Engle-Granger procedure - Step 3 (inofficial): fit OU process to mean-reverting residuals
    theta, mu_e, sigma_ou = estimate_ou_params(train_data['residuals'])
    print(f"Estimated OU parameters: theta={theta:.4f}, mu_e={mu_e:.4f}, sigma_ou={sigma_ou:.4f}")
    print(f"Half-life of OU process: {get_half_life(theta):.2f} days")
    ou_params = {'theta': theta, 'mu_e': mu_e, 'sigma_ou': sigma_ou}
    return train_data, test_data, beta, adf_test_result, ecm_results, ou_params


def analyze_trading_strategy(train_data, test_data, ticker1, ticker2,
                             ou_params, hedge_ratio, index_ticker="SPY",
                             test_z_values=np.arange(0.3, 1.5, 0.1)):
    # Backtesting: in-sample performance evaluation on train_data to find best z-value
    train_results, z_best = backtest_strategy_for_z_values(
        train_data, ticker1, ticker2, ou_params, hedge_ratio, test_z_values)

    # Implement + analyze strategy with optimized z=z_best on test_data
    test_portfolio = Portfolio(test_data, ticker1, ticker2, ou_params, hedge_ratio, z=z_best)
    plot_positions(test_portfolio)
    plot_asset_prices_and_residuals(test_portfolio)
```

```
    plot_pnl_against_index(test_portfolio, index_ticker)


############# Example cases #############


# Coca-Cola and Pepsi
ticker1 = "KO"
ticker2 = "PEP"
start_date = "2020-01-01"
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker1, ticker2, start_date=start_date, maxlag=1)
analyze_trading_strategy(train_data, test_data, ticker1, ticker2, ou_params, beta[1])
# Role reversal
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker2, ticker1, start_date=start_date, maxlag=1)
analyze_trading_strategy(train_data, test_data, ticker2, ticker1, ou_params, beta[1])

# Apple and Microsoft --> not cointegrated!
ticker1 = "AAPL"
ticker2 = "MSFT"
start_date = "2021-01-01"
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker1, ticker2, start_date=start_date)

# Google and Amazon
ticker1 = "GOOGL"
ticker2 = "AMZN"
start_date = "2021-01-01"
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker1, ticker2, start_date=start_date)
analyze_trading_strategy(train_data, test_data, ticker1, ticker2, ou_params, beta[1])
# Role reversal
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker2, ticker1, start_date=start_date)
analyze_trading_strategy(train_data, test_data, ticker2, ticker1, ou_params, beta[1])

# Nestle and Roche
ticker1 = "NESN.SW"
ticker2 = "ROG.SW"
start_date = "2019-01-01"
index_ticker = "^SSMI"
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker1, ticker2, index_ticker=index_ticker, start_date=start_date)
analyze_trading_strategy(train_data, test_data, ticker1, ticker2, ou_params, beta[1],
                         index_ticker=index_ticker)
# Role reversal
```

```python
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker2, ticker1, index_ticker=index_ticker, start_date=start_date)
analyze_trading_strategy(train_data, test_data, ticker2, ticker1, ou_params, beta[1],
                         index_ticker=index_ticker)

# Exxon Mobil and Chevron --> not cointegrated!
ticker1 = "XOM"
ticker2 = "CVX"
start_date = "2019-01-01"
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker1, ticker2, start_date=start_date)

# Gold commodity and Gold futures --> not cointegrated, but also suitable for pairs trading!
ticker1 = "GLD"
ticker2 = "GC=F"
start_date = "2019-01-01"
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker1, ticker2)
analyze_trading_strategy(train_data, test_data, ticker1, ticker2, ou_params, beta[1])
# Role reversal
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker2, ticker1, index_ticker=index_ticker, start_date=start_date)
analyze_trading_strategy(train_data, test_data, ticker2, ticker1, ou_params, beta[1])

# Counter-example --> not cointegrated, not suitable for pairs trading
ticker1 = "AMZN"
ticker2 = "BYND"
start_date = "2019-01-01"
train_data, test_data, beta, adf_test_result, ecm_results, ou_params = analyze_cointegration(
        ticker1, ticker2, start_date=start_date)
analyze_trading_strategy(train_data, test_data, ticker1, ticker2, ou_params, beta[1])
```

Bibliography

[1] Statsmodels Documentation on statsmodels.tsa.stattools.adfuller. `https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html`, 2024. [Online; accessed 2024-08-15].

[2] João Caldeira and Guilherme V Moura. Selection of a Portfolio of Pairs based on Cointegration: A Statistical Arbitrage Strategy. *Available at SSRN 2196391*, 2013.

[3] Richard Diamond. The Secrets of Learning (and Trusting) Cointegration in Spread Trading. *SSRN Electronic Journal*, 2013.

[4] Richard Diamond. JA246.8 Cointegration for Trading: Modelling Long Run Relationships in Time Series, July 2024.

[5] Richard Diamond. JA24T8 Final Project Tutorial II, July 2024.

[6] Richard Diamond. JA24W2 Final Project Workshop II: CQF Final Projects, Workshop Two - Cohort Jan 2024, July 2024.

[7] Robert F. Engle and C. W. J. Granger. Co-Integration and Error Correction: Representation, Estimation, and Testing. *Econometrica*, 55(2):251–276, 1987.

[8] C.W.J. Granger. Some Properties of Time Series Data and Their Use in Econometric Model Specification. *Journal of Econometrics*, 16(1):121–130, 1981.

[9] Nicolas Huck and Komivi Afawubo. Pairs Trading and Selection Methods: Is Cointegration Superior? *Applied Economics*, 47(6):599–613, 2015.

[10] Søren Johansen. Statistical Analysis of Cointegration Vectors. *Journal of Economic Dynamics and Control*, 12(2-3):231–254, 1988.

[11] Søren Johansen. Estimation and Hypothesis Testing of Cointegration Vectors in Gaussian Vector Autoregressive Models. *Econometrica: Journal of the Econometric Society*, pages 1551–1580, 1991.

[12] C. John McDermott. Cointegration: Origins and Significance for Economists. *New Zealand Economic Papers*, 24(1):1–23, 1990.

[13] Saji Thazhungal Govindan Nair. Pairs Trading in Cryptocurrency Market: A Long-Short Story. *Investment Management and Financial Innovations*, 18(3):127–141, 2021.