Università degli Studi di Padova

Dipartimento di Fisica
e Astronomia
Galileo Galilei

XSOR
Capital

GROUP 13

MEAN REVERSION METRIC ON FINANCIAL DATASET

XSOR Capital

Laboratory of Computational physics

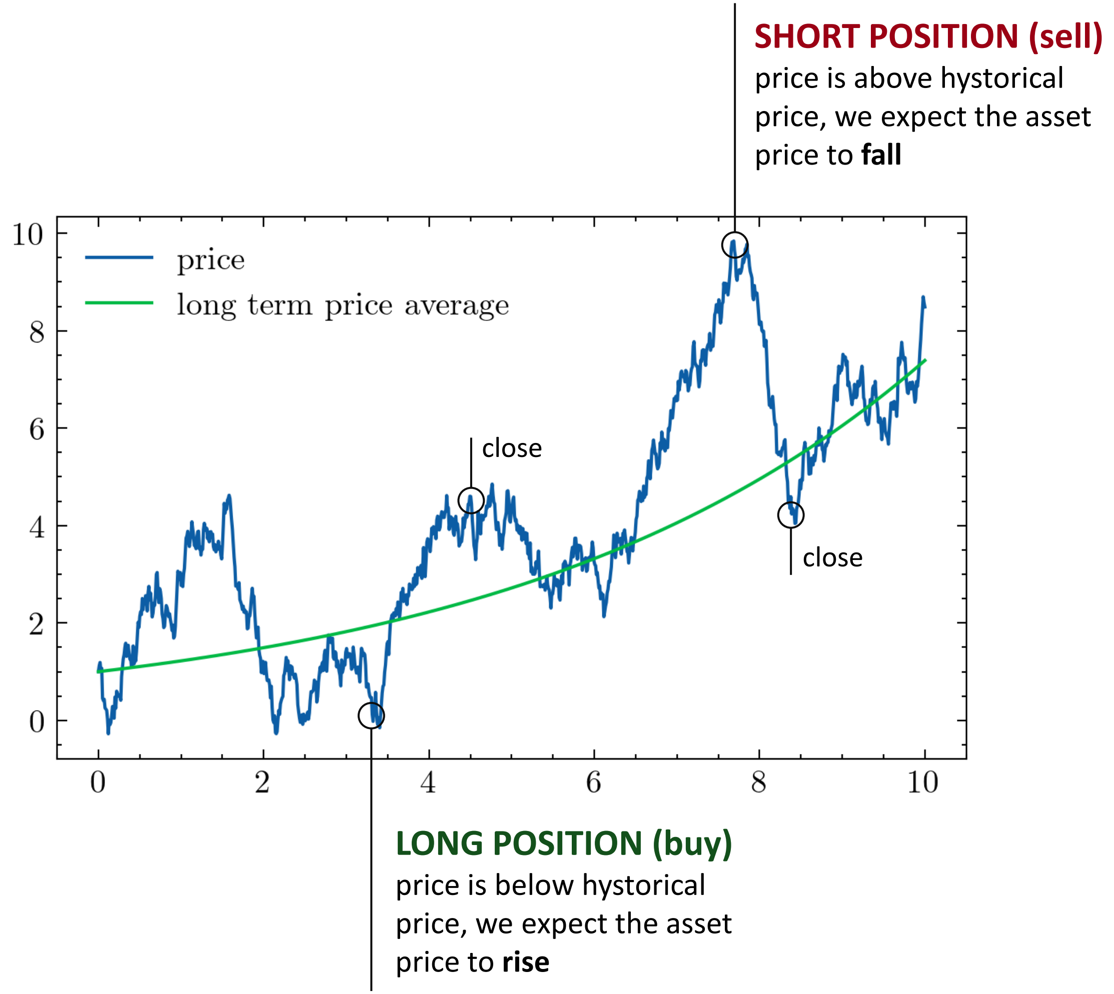
Prof. Marco Zanetti

Riccardo Corte
Alessandro Miotto
Lorenzo Rizzi

Introduction

Mean reversion is the theory that assets prices eventually return to their long-term average

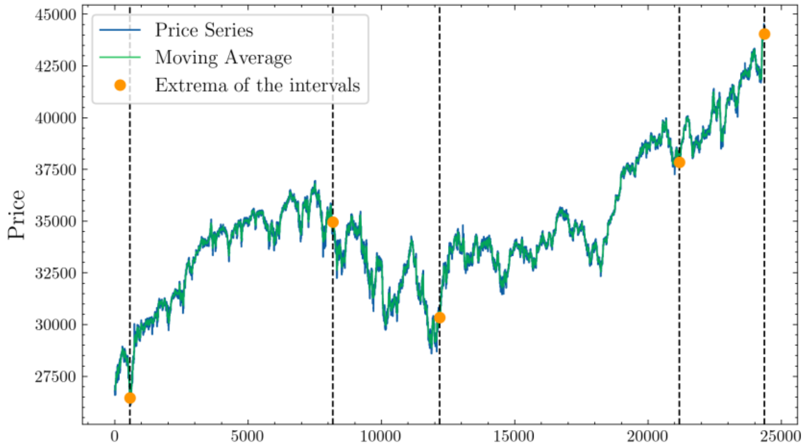
This project aims to develop a metric to quantify mean reversion in financial data and apply it to a trading strategy



Data collection and preprocessing

Data handling is primarily done using `pandas.DataFrame`, **data acquisition** rely on Yahoo! Finance's API aviable from `yfinance` python library. Also **data loading** from `.csv` is implemented

Trend points identification from the relative extrema of the price series using `scipy.signal.argrelextrema`




```python
import yfinance as yf

def download_asset(ticker_symbol : str, start : str, end : str) -> pd.DataFrame:
    # download the data from Yahoo Finance
    download_asset = yf.download(ticker_symbol, start=start, end=end)
    len_dataset = len(download_asset)
    deltaT = (download_asset.index[1] - download_asset.index[0]).total_seconds() / 3600
    data = pd.DataFrame({
        "Open": download_asset["Open"].values.reshape(len_dataset),
        "Close": download_asset["Close"].values.reshape(len_dataset),
        "AbsTime" : download_asset.index
    })
    data.index = map(int,np.arange(0, deltaT*len(data), deltaT))
    return data

def load_asset(name: str) -> pd.DataFrame:
    data = pd.read_csv(name, sep = "  ")
    data.index = np.arange(0, 0.25*len(data), 0.25)
    data["AbsTime"] = pd.to_datetime(data["<DATE>"] + " " + data["<TIME>"])
    data = data.rename(columns = {"<OPEN>": "Open", "<CLOSE>": "Close"})
    data["Return"] = (data["Close"]-data["Open"])/data["Open"] * 100
    return data
```

Dow Jones Index (2020-09-24 to 2024-11-13)

Stochastic processes approach

The price exhibits stronger mean reversion if its oscillations amplitude A are more pronounced than the price volatility σ

$$\eta \propto \frac{\sigma}{A}$$

η is smaller if amplitude A is greater than price fluctuations σ

Strategy

- **Smoothing and normalizing data using Savitzky–Golay filter** `scipy.signal.savgol_filter`
- **Detecting zeros (price cross its long-term price) and finding mean reversion period T**

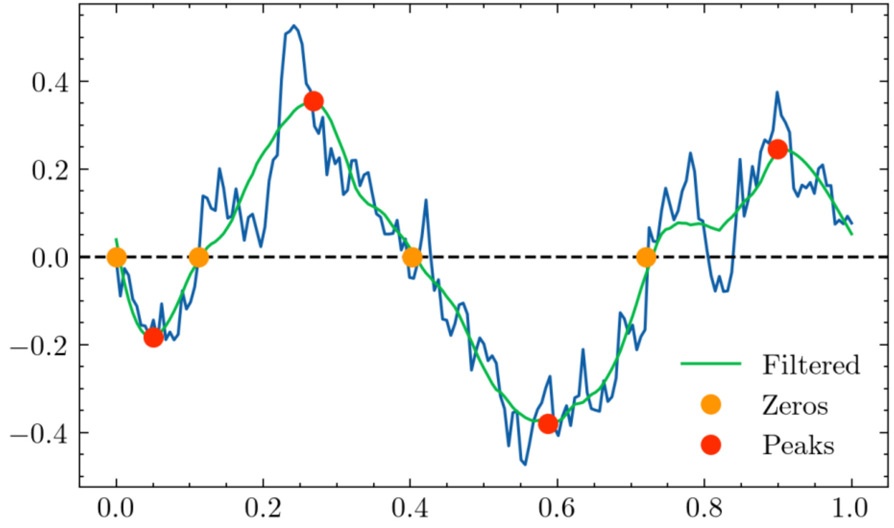
  To find period T we also used fourier transform `scipy.fftpack`
- **Using zeros as separation hint, we detect the signal peaks to find the amplitude A**
  `scipy.signal.find_peaks`
- **Maximum likelihood estimation to fit the stochastic process: we get the volatility σ**
  `scipy.optimize.minimize`

Stochastic processes approach: amplitude A

```python
def find_zeros(x : np.array) → np.array:
    zeros_index = []
    for i in range(len(x)-1):
        if x[i]*x[i+1] < 0:
            zeros.append(i)
    return np.array(zeros, dtype=int)
```

```python
from scipy.signal import savgol_filter

def find_period(x : np.array) → float:
    zeros = find_zeros(x)
    periods = []
    for i in range(len(zeros)-1):
        periods.append((zeros[i+1] - zeros[i])*2)
    period = np.mean(periods)
    return period
```

```python
from scipy.signal import find_peaks

def find_amplitudes(x : np.array, factor : float) → float:
    dist = factor * find_period(x)
    max_peaks, _ = find_peaks(x, distance=dist, height=0.0)
    min_peaks, _ = find_peaks(-x, distance=dist, height=0.0)
    return np.mean(np.concatenate((x[max_peaks], -x[min_peaks])))
```



After smoothing and standardizing the data, the average period T and amplitude A are computed

Similar results are obtained
using fourier transform

```
Period: 0.626 (112 days)
Period using Fourier: 0.500 (90 days)
Amplitude: 0.250
```

Real-world data EUR/USD exchange
Daily price from August 2023 to January 2024

Stochastic processes approach: volatility σ

Stochastic Differential Equation

(Ornstein–Uhlenbeck process)

$$dx = \overbrace{\theta(\mu(t) - x)}^{f(x,t,\boldsymbol{\theta})}dt + \overbrace{\sigma}^{g(\boldsymbol{\theta})}dB$$

Brownian motion

$\mathbb{E}[dB] = 0,\ Var[x] = 1$

Drift term, choosen such that the expected trajectory is a sinusoidal function

$$\mathbb{E}[x(t)] = A\, sin(\omega t + \varphi)$$

Maximum Likelihood Estimation

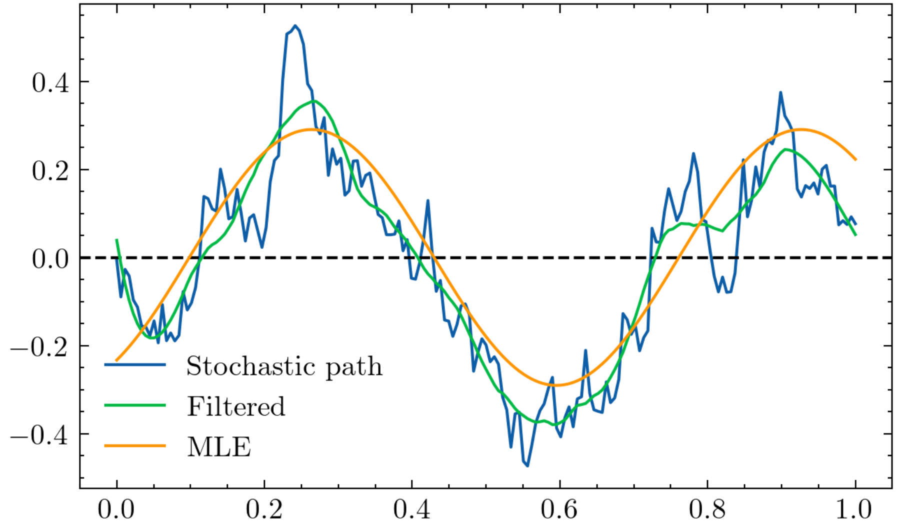
as PDF, we approximate the propagator using a normal distribution with appropriate variance

$$\mathcal{L}(x, t, \boldsymbol{\theta}) = \prod p_\theta(x, t|\boldsymbol{\theta})$$

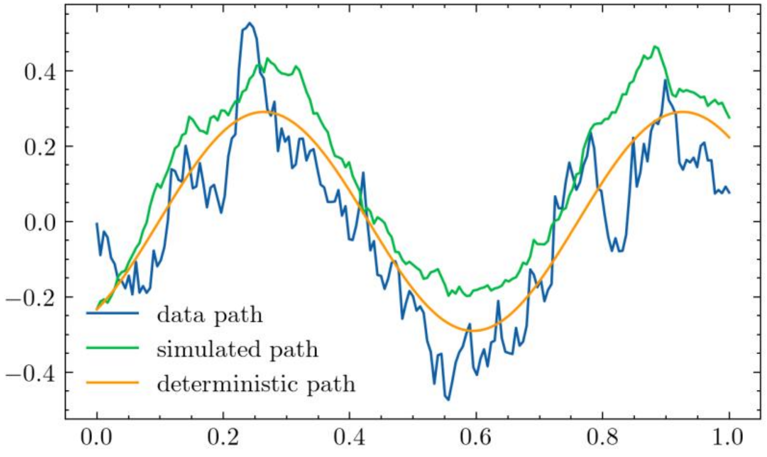
$$\boldsymbol{\theta} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(x, t, \boldsymbol{\theta})$$

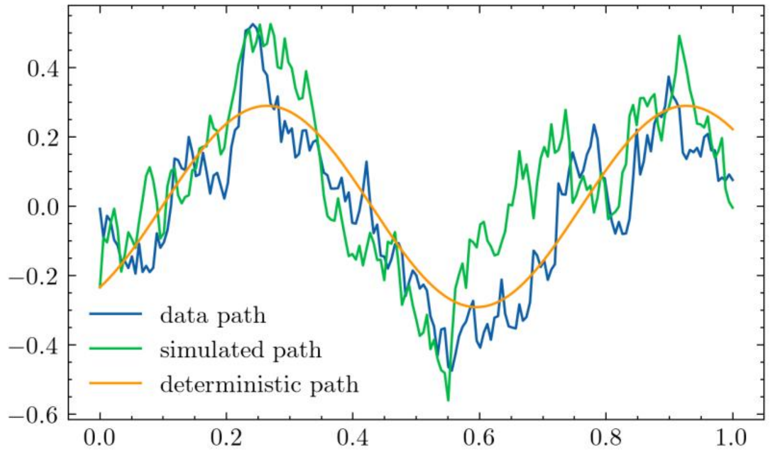
We use the previou result for the period as an hint for the minimization process

```
from scipy.optimize import minimize
import meanreversion as mr

def f(t, x, params):
    omega, phi, theta, _ = params
    return theta*(A*np.sin(omega*t+phi)-x)+A*omega*np.cos(omega*t+phi)

def g(x, params):
    _, _, _, sigma = params
    return sigma

guess = [frequency, 0.0, 1.0, 1.0]
res = minimize(mr.log_likelihood, guess,args=(x,t,f,g), method='L-BFGS-B')
```



Stochastic processes approach: volatility σ






```
Volatility using MLE: 0.1606
Volatility using AR(1): 0.8035
```

MLE to fit the stochastic differential equation has some issue:

- **computational costly**
- **underestimation of the volatility σ**

Time Series methods

(autoregressive model $AR(1)$)

$$X_{t+1} = c + \varphi X_t + \varepsilon_t$$

The parallel between continuous and discrete stochastic processes allows us to leverage established packages like `statsmodels` for efficient volatility estimation

```python
from statsmodels.tsa.arima.model import ARIMA

def volatility(x : pd.Series) → float:
    AR_model = ARIMA(x, order=(1,0,0), trend='n')
    res = AR_model.fit(method='burg')
    AR_phi = res.arparams[0]
    AR_sigma = res.params[-1]

    dt = 1
    theta_AR = -np.log(AR_phi)/dt
    sigma_AR = np.sqrt(AR_sigma * (2 * theta_AR / (1 - AR_phi**2)))

    return sigma_AR
```

By integrating the stochastic differential equation and discretizing time, we recognize the AR(1) terms

$$\sigma = \varepsilon_t \sqrt{\frac{2 \log \varphi}{\Delta t(\varphi^2 - 1)}}$$

Normality test approach

We check for mean reversion by testing if randomly sampled points are normally distributed: weak mean reversion suggests random noise drives the oscillations, which should be normal in an Ornstein-Uhlenbeck process

$$\boldsymbol{\eta} \propto \textit{Anderson-Darling statistics}$$

Anderson-Darling test tests the null hyphothesis that a sample is drawn from a specific distribution (in our case a Gaussian)

Strategy

- **Randomly samples points with average spacing $\boldsymbol{\lambda}$:** $S^{(i)} = \left\{x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)}\right\}$
  Poissonian sampling from `np.random.poisson`
- **Construct an hystogram with sample $\boldsymbol{S^{(i)}}$**
  Repeat for $\boldsymbol{i = 1, \dots, N}$
- **Aggregate all hystogram from all the samples $\boldsymbol{S^{(1)}, \dots, S^{(N)}}$ and test for normality** `scipy.stats.anderson`
- **Use the Anderson test statistics to obtain a normalized mean reversion metric**

Normality test approach

```python
from scipy.stats import anderson

def assess_normality(dataset : pd.Series, lambda_ : int) → float:
    x = np.array([0])
    n = lambda_

    for i in range(n):
        # we will sample the dataset choosing a value
        # every lambda points on average (poisson distribution)
        choice = [np.random.randint(lambda_)]
        for i in range(1, int(len(dataset)/lambda_) ):
            l = np.random.poisson(lambda_)
            if (int(choice[-1] + l) > len(dataset) - 1):
                break
            choice.append(int(choice[-1] + l))
        chosen_data = (dataset.values)[choice]
        x = np.concatenate((x,chosen_data), axis = 0)

    dev_from_normality = anderson(x).statistic
    normalized_norm = erf(dev_from_normality / (0.4 * lambda_))
    return normalized_norm
```

Strategy

- **Randomly samples points with average spacing $\boldsymbol{\lambda}$:** $S^{(i)} = \left\{x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)}\right\}$

  Poissonian sampling from `np.random.poisson`

- **Construct an hystogram with sample $\boldsymbol{S^{(i)}}$**

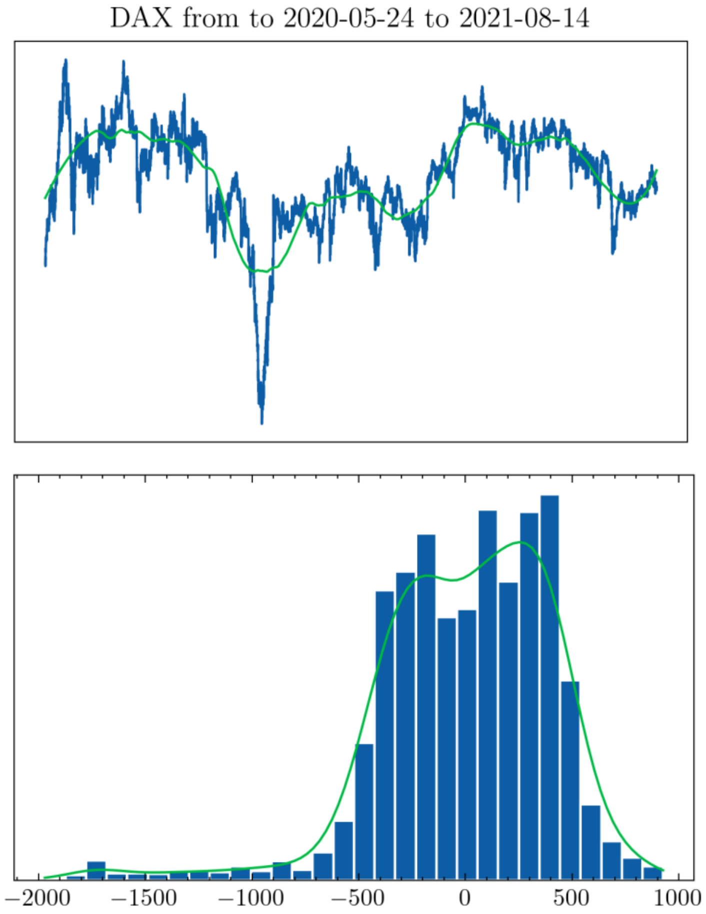
  Repeat for $\boldsymbol{i = 1, \dots, N}$

- **Aggregate all hystogram from all the samples $\boldsymbol{S^{(1)}, \dots, S^{(N)}}$ and test for normality** `scipy.stats.anderson`

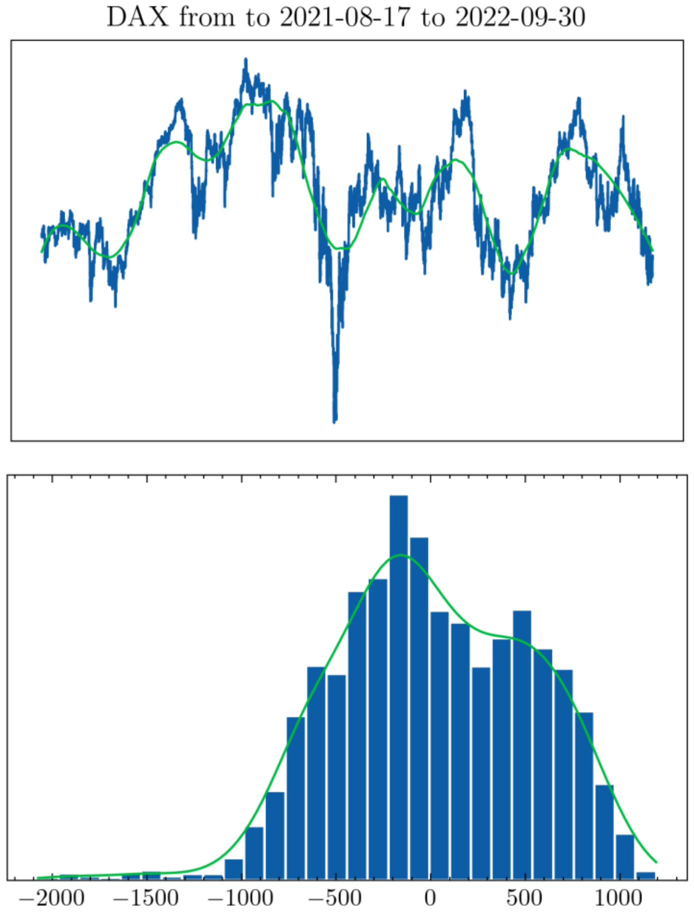
- **Use the Anderson test statistics to obtain a normalized mean reversion metric**

Normality test approach

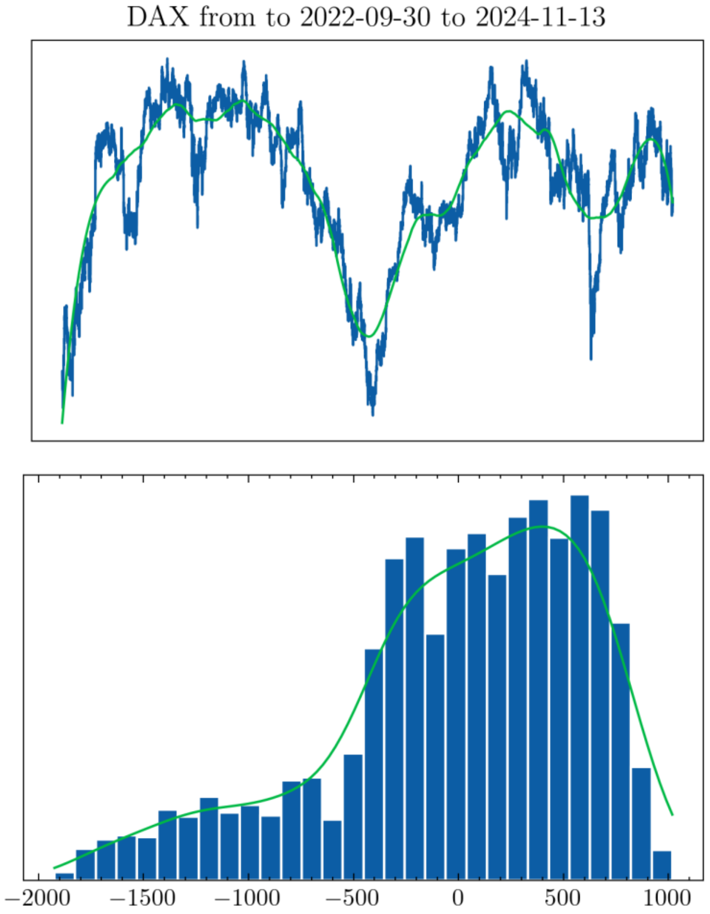
We analyze DAX Index data from May 24, 2020, to November 13, 2024, divided into three chunks. The first two chunks contain approximately 24,000 data points each, while the third contains around 49,000. Within each chunk, we sample every 1000th point ($\lambda = 1000$) and repeat this sampling process 1000 times ($m = 1000$).



MRI: 0.568



MRI: 0.212



MRI: 0.996

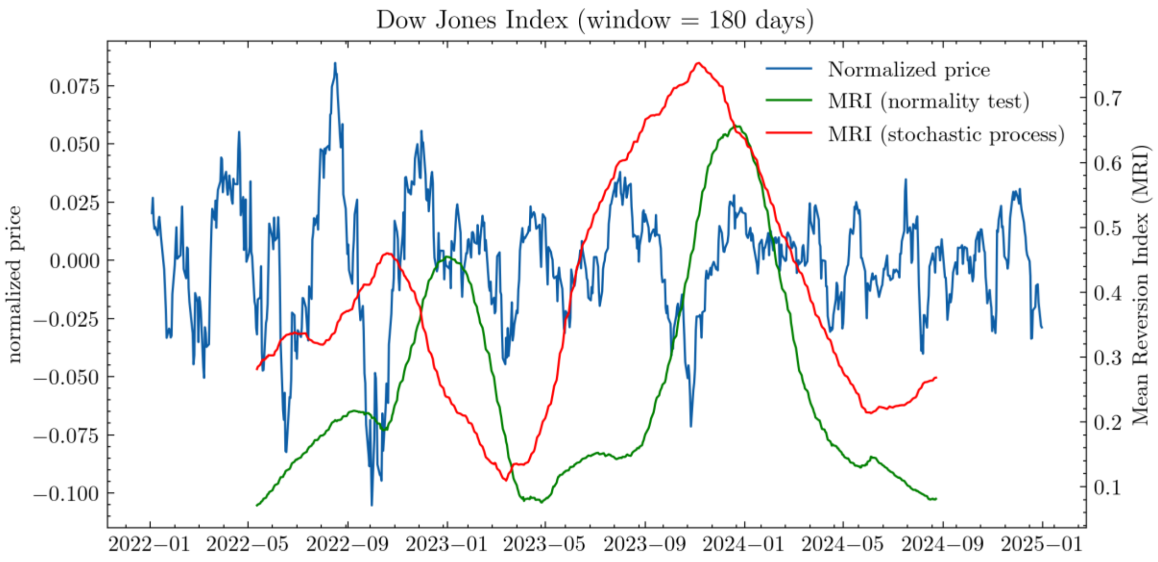
Methods comparison

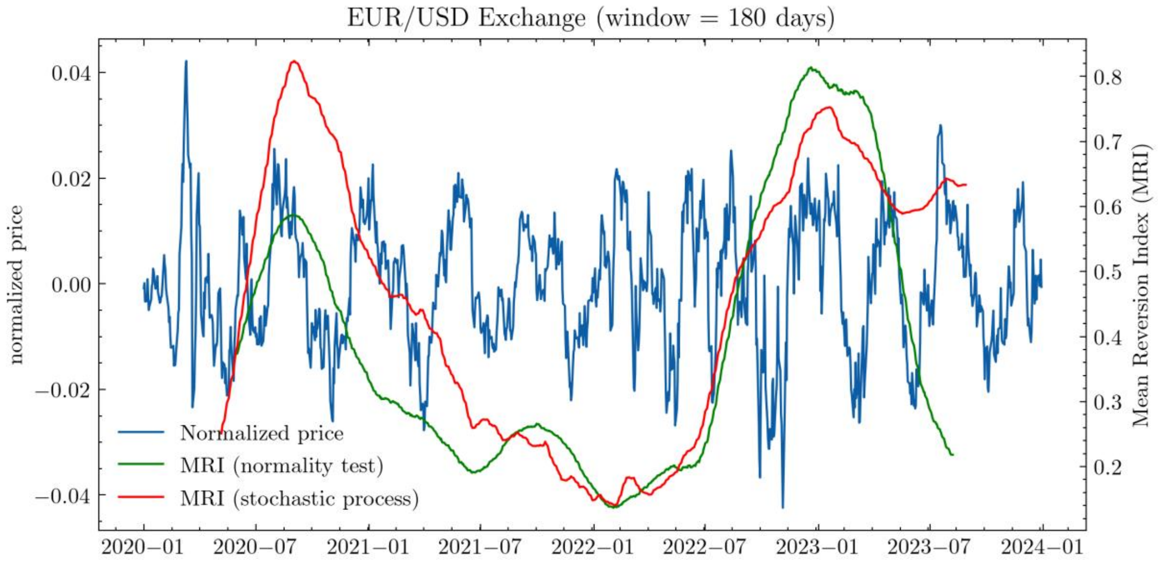
Both our stochastic process and normality test functions are designed for efficient integration with `pandas.DataFrame` rolling calculations, enabling real-time analysis

For comparison, the mean reversion index is normalized between zero and one

$$\eta = \begin{cases} 0 & \text{fluctuations are merely random noise} \\ 1 & \text{fluctuations are deterministic oscillations} \end{cases}$$

The two methods yield comparable results. Within the time window, the mean reversion index η approaches one during clear price oscillations and tends towards zero in periods dominated by noise





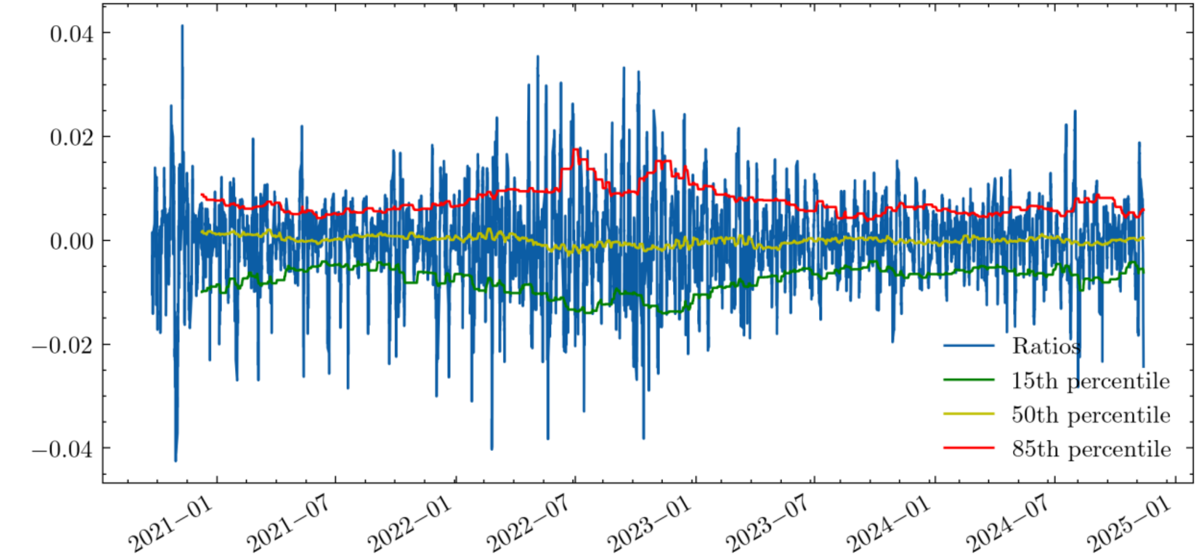
Trading strategy

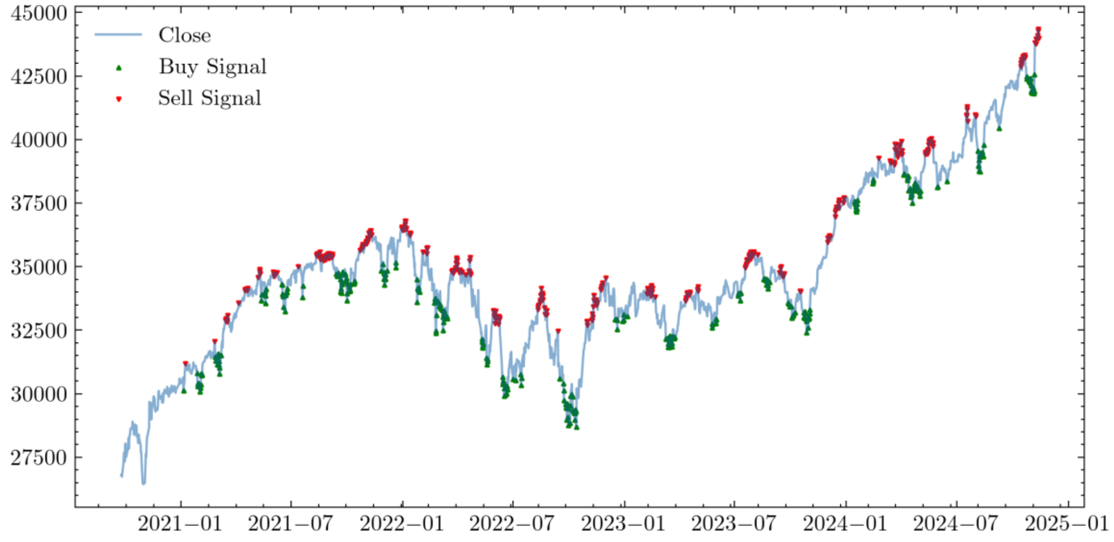
We buy low, sell high, and trade more when mean reversion is strong

↓

We use rolling percentiles to define Bollinger Bands, triggering buy orders below the 15th percentile and sell orders above the 85th

```
# Buy and sell signals:
#  - buy when the ratio is below the 15th percentile,
#  - sell when it is above the 85th percentile
df['Positions'] = np.where(df.Ratios > df['perc_85'], -1, 0)
df['Positions'] = np.where(df.Ratios < df['perc_15'], 1, df['Positions'])
df['Positions'] = df['Positions'].ffill()

# Compute current and previous buy and sell prices
df['Buy'] = np.where(df['Positions'] == 1 , df['Close'], np.nan)
df['Sell'] = np.where(df['Positions'] == -1 , df['Close'], np.nan)
df['Previous_Buy'] = df['Buy'].shift(1)
df['Previous_Sell'] = df['Sell'].shift(1)
```





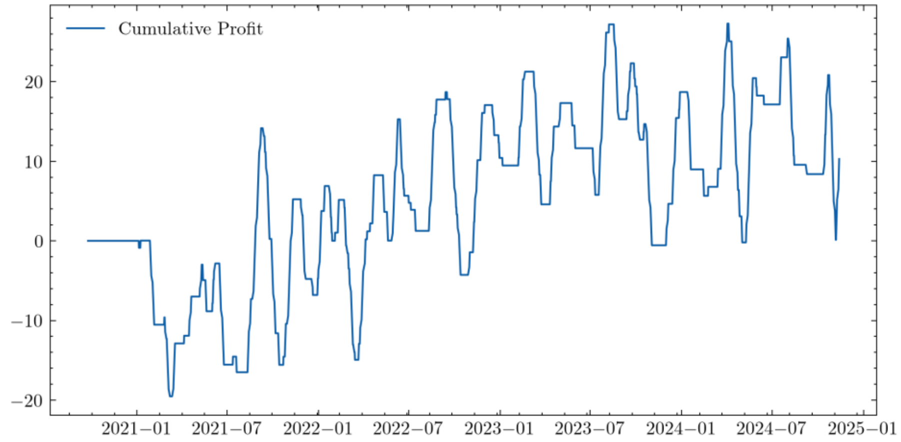
Dow Jones Index (data from 2021 to 2025)

Trading strategy

```python
def apply_trading_strategy(df):
    mv = df['Close'].mean()
    #Compute weighted strategy
    df['Investment'] = np.where(df['Buy'].notna(), df['Buy'] / mv * df['MRI'], 0)
    df['Revenue'] = np.where(df['Sell'].notna(), df['Sell'] / mv * df['MRI'], 0)
    df['Trade_Profit'] = df['Revenue'] - df['Investment']
    df['Cum_Profit'] = df['Trade_Profit'].cumsum()

    #Count buy/sell signals
    num_buys = df['Buy'].count()
    num_sells = df['Sell'].count()
    print(f"Total Buy Signals: {num_buys}")
    print(f"Total Sell Signals: {num_sells}")

    #Calculate total investment
    df['Investment_Per_Buy'] = df['Buy'] / mv
    total_investment = df['Investment_Per_Buy'].sum()
    print(f"Total Euro Invested: {total_investment}")

    #Compute and print total percentual profit
    total_profit_percent = df['Cum_Profit'].iloc[-1] / total_investment * 100
    print(f"Total percentual profit: {total_profit_percent:.2f}%")

    return None
```

Our trading function executes buy/sell orders based on our signals. Each trade invests a fixed, small amount (usually a fraction) of stock

Profitability is strongly correlated with price behavior: trending markets lead to losses, while sinusoidal patterns generate substantial gains



Without mean reversion index we always buy a fixed fraction of stock at each transaction

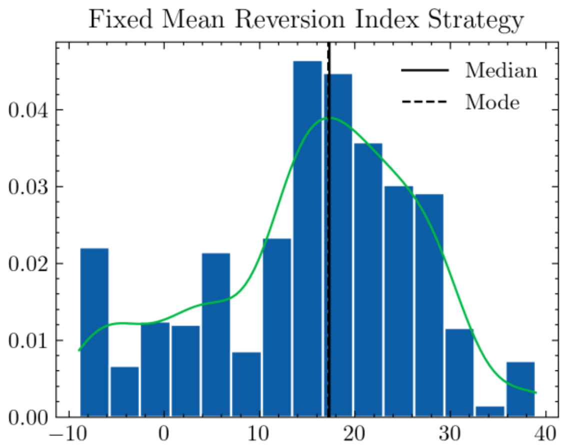
```
Buy transactions:  315
Sell transactions: 308
Investment: 309.8 €
Profit: 3.32%
```

Dow Jones Index (data from 2021 to 2025)

Trading strategy

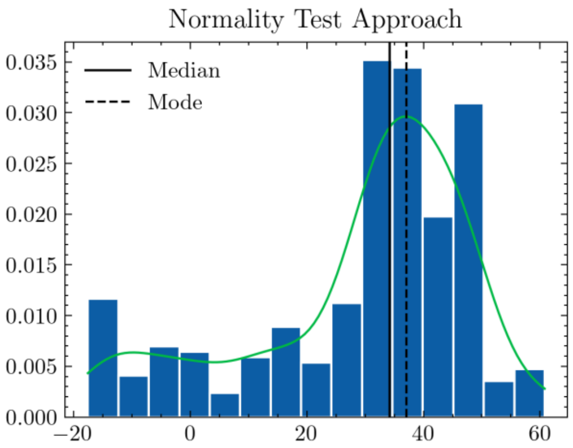
To assess the performance of our mean reversion metrics compared to a fixed-buy strategy, we analyze the distribution of cumulative profits

We expect the mean reversion strategies to shift the distribution towards positive values, indicating profitability.
We analyze the skewness to assess this shift



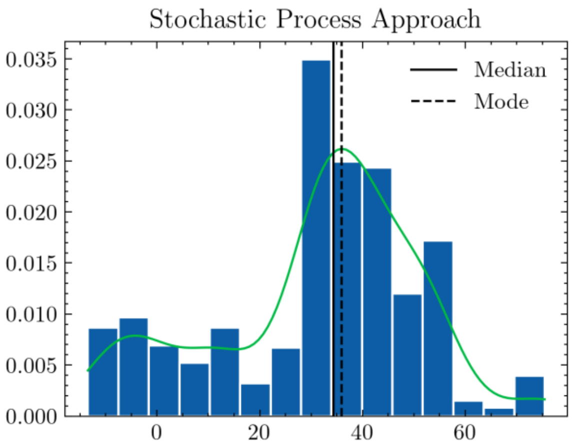
Mean reversion fixed: we always buy the same quantity of stocks at each transaction

```
Total profit: +5.81%
Skewness: -0.44
```



We buy more stocks with stronger mean reversion (computed with **normality test** approach)

```
Total profit: +14.4%
Skewness: -0.98
```



We buy more stocks with stronger mean reversion (computed with **stochastic processes** approach)

```
Total profit : +14.0%
Skewness: -0.52
```

Dow Jones Index (data from 2021 to 2025)