WiDS Kalman Filtered Trend Trader:Week 2

Aarav Malde and Krishang Krishna

Contents

1 Introduction

The Kalman Filter is a recursive algorithm used for estimating the state of a dynamic system from noisy observations. It has wide applications in signal processing, control systems, and finance. In the context of pairs trading, the Kalman Filter is used to model and estimate the dynamic relationship between two financial assets, providing a robust framework for identifying trading signals.

1.1 Hypothesis Testing

Hypothesis testing is a fundamental statistical tool used to evaluate assumptions (or hypotheses) about a dataset. It involves the following key elements:

- **Null Hypothesis (H_0):** The null hypothesis represents the default assumption or claim.
- **Alternative Hypothesis (H_a):** The alternative hypothesis is the opposite of the null hypothesis, representing a different claim.
- **Test Statistic:** A test statistic is a value computed from the sample data that is compared to a critical value or threshold to determine whether to reject H_0. Examples include z-scores, t-statistics, or the test statistic from the Augmented Dickey-Fuller (ADF) test(no need to know what these do, they are just tests to find out which of the HYPOTHESIS hold true)..
- **Significance Level (α):** The p-value quantifies the evidence against the null hypothesis. A smaller p-value indicates stronger evidence to reject H_0. For instance, if the p-value is less than α, we reject H_0.
- **P-value:** The p-value quantifies the evidence against the null hypothesis. A smaller p-value indicates stronger evidence to reject H_0. For instance, if the p-value is less than α, we reject H_0..
- **Decision:** Based on the comparison of the test statistic with the critical value or the p-value with α, we either reject or fail to reject H_0.

1.2 Stationarity and Non-Stationarity

Stationarity is a commonly untested assumption in time series analysis. We generally assume that a dataset is stationary when the parameters of its data-generating process do not change over time. To illustrate, consider two time series: A and B.

- **Series A:** This is a stationary time series where parameters like the mean and standard deviation remain constant over time. It reflects a stable system whose statistical properties do not depend on the time at which they are measured.

- **Series B:** This is a non-stationary time series where the parameters vary with time. For instance, the mean might increase steadily as time progresses, indicating a trend or evolving dynamics in the system.

Mathematical Context

The concept of stationarity can be understood through the probability density function (PDF) of a Gaussian distribution:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

Here:

- μ is the mean of the distrubution
- σ is the standard deviation
- σ^2 is the variance, which is the square of the standard deviation

The empirical rule states that approximately 66% of the data in a Gaussian distribution lies within $\mu \pm \sigma$. This means that random samples are more likely to cluster around the mean.

Illustration of Stationary vs. Non-Stationary Time Series

To illustrate the difference between stationary and non-stationary time series:

- For **Stationary Series** A, the mean (μ) and variance (σ^2) remain constant throughout the time period. This stability ensures that statistical methods designed for stationary data yield reliable results.
- For **Non-Stationary Series** B, the mean evolves over time, such as increasing linearly. This dependency on time violates the assumption of stationarity and often requires techniques like differencing or detrending for proper analysis.

Visualization

- The stationary series appears to fluctuate consistently around a fixed mean.
- The non-stationary series exhibits a clear trend, where the mean shifts as time progresses.

Most statistical methods, including stationarity tests and Kalman filtering, assume stationarity in the underlying data.

2 Augmented Dickey-Fuller Test (ADF)

The Augmented Dickey-Fuller (ADF) test is a statistical tool used to determine if a time series is stationary. This section explains the mathematical foundations and notations used in the ADF test.

Unit Root and Non-Stationarity

Definition of a Unit Root:
A unit root in a time series occurs when the coefficient α in the following equation equals 1:

$$Y_t = \alpha Y_{t-1} + X_t + \epsilon_t \tag{2}$$

where:

- Y_t: Value of the time series at time t.
- Y_{t-1}: Lagged value of the time series (from the previous time step).
- X_t: Exogenous variable (an explanatory variable or deterministic component like a trend or constant).
- ϵ_t: White noise error term.

If $\alpha = 1$, the time series exhibits a random walk:

$$Y_t = Y_{t-1} + X_t + \epsilon_t \tag{3}$$

This implies that shocks to the system (ϵ_t) have a persistent effect, and the series does not revert to a mean value, making it non-stationary. Conversely, when $|\alpha| < 1$, the series is stationary, as it tends to revert to a mean and has constant variance over time.
Differencing and Stationarity
The number of unit roots in a time series indicates the number of differencing operations needed to make it stationary. For example, if a series has one unit root ($\alpha = 1$), taking the first difference:

$$\Delta Y_t = Y_t - Y_{t-1} \tag{4}$$

can remove the non-stationarity, resulting in a stationary series.

The Hypothesis Test

The ADF test is framed as a hypothesis test:

$$H_0 : \phi = 1 \Rightarrow y_t \sim I(1)$$

$$H_1 : |\phi| < 1 \Rightarrow y_t \sim I(0)$$

- Under H_0, the time series has a unit root and is non-stationary.
- Under H_1, the time series is stationary.

And the Test statistic is computed as:

$$t_{\phi=1} = \frac{\hat{\phi} - 1}{SE(\hat{\phi})} \tag{5}$$

Where:

- $\hat{\phi}$: The estimated value of the autoregressive coefficient.
- $SE(\hat{\phi})$: The standard error of $\hat{\phi}$, representing the uncertainty in the estimate.

Distribution of the Test Statistic

Under Stationarity (H_1) If the series is stationary, the test statistic $t_{\phi=1}$ follows a normal distribution:

$$t_{\phi=1} \overset{A}{\sim} \mathcal{N}(0, 1)$$

Under Non-Stationarity (H_0) If the series is non-stationary, the distribution of the test statistic diverges, and the time series retains a unit root.

Practical Interpretation

The ADF test determines whether a time series is stationary:

- A p-value below a chosen significance level (e.g., 0.05) implies rejection of the null hypothesis, concluding that the series is stationary.
- A p-value above the significance level indicates failure to reject the null hypothesis, suggesting that the series is non-stationary.
  If the math seems too complicated, its not an issue you can skip this for now. The more important point to understand is how we can use this test and the statistic obtained to test for the stationarity of a process.

3 Trading Signals

When conducting any type of trading strategy, it's essential to clearly define the points at which trades will be executed. That is, what is the best indicator to determine when to buy or sell a particular stock? This section will explore how to define trading signals using a ratio time series and how these signals guide buying and selling decisions in a trading strategy.

Setup Rules

We are going to use the ratio time series created between two stocks to determine whether to buy or sell at any given moment in time. The ratio represents the relative movement between the two stocks, and the signal to trade is based on changes in this ratio. To predict the direction of the ratio movement, we first create a prediction variable Y. The prediction rule is defined as follows:

$$Y_t = \text{sign}(\text{Ratio}_{t+1} \text{ - } \text{Ratio}_t)$$

Where:

1. Y_t is the trading signal at time t.
2. Ratio_t represents the ratio between the two stocks at time t
3. sign(x) is the sign function, which returns:
   - 1 for positive changes in the ratio (indicating a "buy" signal).
   - -1 for negative changes in the ratio (indicating a "sell" signal).

The ratio moving upwards signals a "buy", and the ratio moving downwards signals a "sell". The strength of this approach is that we do not need to predict the absolute future price of the stocks but merely whether the ratio will move up or down. (If this confuses you know, read further ahead and it will make more sense then).

3.1 Feature Engineering

To improve the model's ability to predict the direction of the ratio's movement, we need to identify key features that help determine the ratio's future direction. Given that the ratios tend to revert to their mean over time, moving averages and statistical measures related to the mean are likely to be useful features for the model.
Here are the features we will use:

1. 60-day Moving Average of the Ratio: This helps capture long-term trends.
2. 5-day Moving Average of the Ratio: This helps capture short-term fluctuations.
3. 60-day Standard Deviation: This measures the volatility of the ratio over the long term.
4. Z-score: The Z-score is a normalized value that indicates how many standard deviations the current ratio is from its mean.

The following Python code demonstrates how to calculate these features:

```
ratios_mavg5 = train.rolling(window=5, center=False).mean()
ratios_mavg60 = train.rolling(window=60, center=False).mean()
std_60 = train.rolling(window=60, center=False).std()

zscore_60_5 = (ratios_mavg5 - ratios_mavg60) / std_60
```

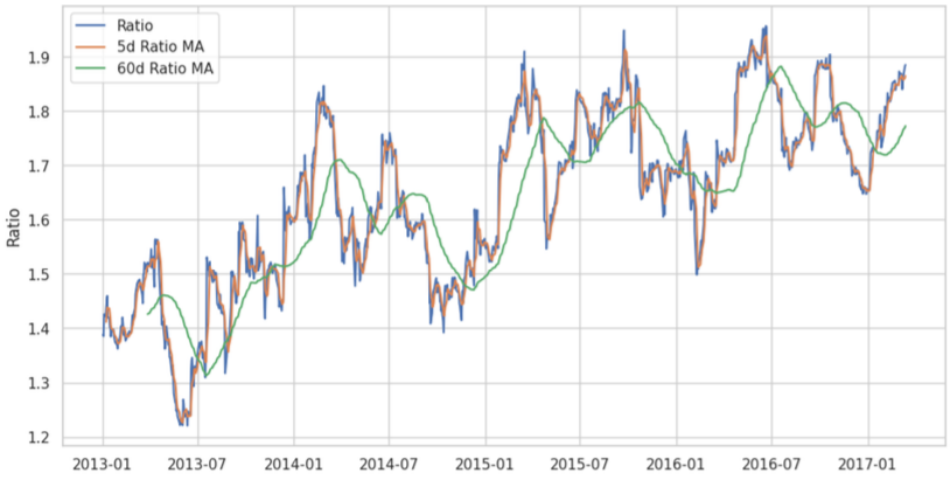

Figure 1: The plot shows the ratio alongside its 5-day and 60-day moving average

In the plot, the blue line represents the ratio, the orange line represents the 5-day moving average, and the green line represents the 60-day moving average. By examining these moving averages, we can get a sense of whether the ratio is trending upward or downward over different time horizons.

3.2 Creating a Model

The key idea behind mean-reversion trading is that asset prices tend to fluctuate around a long-term average. To model this behavior, we use the Z-score, which measures how far the current stock price deviates from its recent mean in terms of standard deviations. A Z-score greater than 1 or less than -1 indicates that the price is significantly away from its mean and is likely to revert, thereby providing a trading signal.

The trading signals are defined as follows:

1. Buy Signal (1): If the Z-score is below -1, the stock is considered undervalued relative to its recent average, and a buy signal is generated.

2. Sell Signal (-1): If the Z-score is above 1, the stock is considered overvalued relative to its recent average, and a sell signal is generated.

These Z-score thresholds are used to determine when to enter or exit trades. In this strategy, the Z-score is computed using the difference between a short-term (5-day)

moving average and a long-term (60-day) moving average of the stock price, normalized by the rolling standard deviation over the 60-day window.

The following code generates the buy and sell signals based on the Z-score:

```
buy[zscore_60_5 > -1] = 0
sell[zscore_60_5 < 1] = 0
```

3.3 Implementing the Trading Strategy

Once the buy and sell signals are defined, we implement the trading strategy by simulating trades using Z-score–based thresholds. In this strategy, the goal is to exploit mean-reverting behavior in the price of a single asset by comparing the current price to its recent historical average.

To implement this strategy, we simulate the following actions:

1. When the Z-score is less than -1, it triggers a buy signal, indicating that the stock price is significantly below its recent mean and is expected to revert upward.

2. When the Z-score is greater than 1, it triggers a sell signal, indicating that the stock price is significantly above its recent mean and is expected to revert downward.

3. When the Z-score lies between -0.75 and 0.75, any open positions are exited, effectively clearing the portfolio.

The following Python function simulates the above-mentioned trading strategy:

```
def trade(S, window1, window2):
# If window length is 0, algorithm does not make sense
if (window1 == 0) or (window2 == 0):
    return 0

# Compute rolling mean and rolling standard deviation
ma1 = S.rolling(window=window1, center=False).mean()
ma2 = S.rolling(window=window2, center=False).mean()
std = S.rolling(window=window2, center=False).std()

zscore = (ma1 - ma2) / std

# Simulate trading
money = 0
position = 0  # +1 for long, -1 for short, 0 for no position

for i in range(len(S)):
    # Buy signal
    if zscore[i] < -1 and position == 0:
```

```
            money -= S[i]
            position = 1

        # Sell signal
        elif zscore[i] > 1 and position == 0:
            money += S[i]
            position = -1

        # Exit position
        elif abs(zscore[i]) < 0.75 and position != 0:
            money += position * S[i]
            position = 0

    # Close any remaining position at the end
    money += position * S.iloc[-1]

    return money
```

This function simulates trading activity, where S represents the stock's price series and window1 and window2 define the lengths of the short-term and long-term moving average windows. Trading decisions are made based on the standardized deviation of the price from its recent mean, and the function returns the total profit or loss generated by the strategy.

3.4 Where Kalman Filter Comes In

While iterating over various window lengths can help identify the optimal trading window, we must also be cautious of the risk of overfitting. Overfitting occurs when a model performs exceptionally well on historical data but fails to generalize to new, unseen data. This issue arises because the model has simply memorized the historical data rather than uncovering genuine patterns or trends that would hold in future market conditions. Thus, finding a robust and reliable window length is essential to avoid the overfitting pitfall. In our model, we used rolling parameter estimates to generate signals for trading, but we may want to go further and optimize the window length. To achieve this, we can iterate over all possible, reasonable window lengths and select the one that maximizes our profit and loss (PnL) on the training data. Here's a simple approach to find the optimal window length:

```
# Find the window length 0-254
# that gives the highest returns using this strategy
length_scores = [trade(df['ADBE'].iloc[:1057],
                 df['MSFT'].iloc[:1057], l, 5)
                 for l in range(255)]
```

```
best_length = np.argmax(length_scores)
print ('Best window length:', best_length)
length_scores2 = [trade(df['ADBE'].iloc[1057:],
                  df['MSFT'].iloc[1057:],1,5)
                  for l in range(255)]
print (best_length, 'day window:', length_scores2[best_length])
# Find the best window length based on this dataset,
# and the returns using this window length
best_length2 = np.argmax(length_scores2)
print (best_length2, 'day window:', length_scores2[best_length2])
```

In the above code, we test all possible window lengths from 0 to 254 and compute the PnL for both training and testing data. The best window length corresponds to the one that maximizes the PnL. From the results, it is evident that using a window length above around 50 days offers a reasonable choice for our trading strategy. However, we must avoid picking a length that is too short and susceptible to noise, which would overfit our model.

To further reduce the risk of overfitting, we can use economic reasoning to choose the window length. Trading decisions driven purely by data can be misleading if not backed by a solid economic rationale. Therefore, combining data-driven insights with fundamental knowledge of market cycles, company-specific events, and economic factors can help refine our window selection process.

Moreover, an even more sophisticated approach would be to replace static window lengths with adaptive techniques. One such technique is the **Kalman filter**, which is a recursive method for estimating the state of a system from noisy measurements. The Kalman filter doesn't require us to specify a fixed window length, and it can dynamically adjust its estimates based on incoming data.

4 The Kalman Filter

4.1 Mathematical Formulation

The Kalman Filter operates on a linear dynamic system described by the following equations:

$$x_k = Ax_{k-1} + w_{k-1}, \quad w_{k-1} \sim \mathcal{N}(0, Q) \tag{6}$$

$$z_k = Hx_k + v_k, \quad v_k \sim \mathcal{N}(0, R) \tag{7}$$

where:

- x_k is the state vector at time k,
- z_k is the observation vector at time k,

- A is the state transition matrix,
- H is the observation matrix,
- Q is the process noise covariance,
- R is the measurement noise covariance.

These equations originate from modeling the evolution of a system (state transition equation) and the relationship between the system's state and observed measurements (observation equation). Process noise (w_{k-1}) represents uncertainty in system dynamics, while measurement noise (v_k) accounts for observational inaccuracies.

4.1.1 Prediction Step

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} \tag{8}$$

$$P_{k|k-1} = AP_{k-1|k-1}A^T + Q \tag{9}$$

where: $\hat{x}_{k|k-1}$ is the predicted state, and $P_{k|k-1}$ is the predicted covariance. These equations propagate the state forward in time and account for uncertainties introduced by process noise.

4.1.2 Update Step

The state and covariance are updated using the observation z_k:

$$K_k = P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1} \tag{10}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - H\hat{x}_{k|k-1}) \tag{11}$$

$$P_{k|k} = (I - K_kH)P_{k|k-1} \tag{12}$$

where K_k is the Kalman Gain. The Kalman Gain determines the weighting between the prediction and the new observation, ensuring that more accurate measurements contribute more to the state estimate.

4.2 Derivation and Intuition

The Kalman Filter equations can be derived from Bayesian principles. The goal is to estimate the posterior probability distribution of the state x_k given the observations up to time k, $z_{1:k}$. The prediction step estimates $p(x_k|z_{1:k-1})$, while the update step computes $p(x_k|z_{1:k})$ using Bayes' theorem. Intuitively:

1. The prediction step propagates the state forward, assuming the system dynamics described by A.

2. The update step corrects this prediction using the new observation, balancing the uncertainties in the prediction ($P_{k|k-1}$) and measurement noise (R).

5 Trading Using Kalman Filter

Systematic trading involves extracting actionable signals from noisy financial time series in order to make informed trading decisions. Market prices are assumed to be noisy observations of an underlying latent process such as fair value, trend, or regime. The Kalman Filter provides a principled framework to estimate these latent states in real time and design trading strategies around them.

5.1 Hurst Test

The Hurst exponent H is used to characterize the behavior of a financial time series and to determine whether a mean-reversion or trend-following trading strategy is appropriate.

The Hurst exponent is computed using:

$$\tau(l) = \sqrt{\text{Var}(X_{t+l} - X_t)} \tag{13}$$

$$H = \frac{\log(\tau(l))}{\log(l)} \tag{14}$$

Interpretation:

- $H < 0.5$: Mean-reverting behavior (suitable for contrarian strategies)
- $H = 0.5$: Random walk (limited predictability)
- $H > 0.5$: Persistent or trending behavior (suitable for trend-following strategies)

In trading applications, the Hurst exponent can be applied to:

- Asset prices or returns
- Residuals from a trend or fair-value model
- Filtered price deviations obtained from a Kalman Filter

This allows traders to select strategies aligned with the underlying market dynamics.

5.2 Half-Life of Mean Reversion

For mean-reverting trading strategies, the half-life of mean reversion quantifies the expected time required for a price deviation to revert halfway back to its equilibrium level. This provides guidance on trade holding periods and strategy horizons.

The half-life is computed as:

$$\text{Half-life} = \frac{-\ln(2)}{\hat{\phi}} \tag{15}$$

where $\hat{\phi}$ is obtained from the regression:

$$\Delta x_t = \alpha + \phi x_{t-1} + \epsilon_t \tag{16}$$

A shorter half-life indicates faster mean reversion and favors short-term trading, while a longer half-life suggests slower reversion and longer holding periods.

5.3 Kalman Filter State-Space Model for Trading

In a trading context, the Kalman Filter is used to estimate latent market states such as fair value or trend from observed prices.

A common model assumes that the observed price y_t is a noisy observation of an unobserved state μ_t:

$$y_t = \mu_t + \epsilon_t \tag{17}$$

The latent state evolves over time according to:

$$\mu_t = \mu_{t-1} + w_t \tag{18}$$

where ϵ_t and w_t represent observation noise and process noise, respectively.

5.4 Kalman Filter Regression for Trading Signals

The Kalman Filter can also be used to estimate time-varying relationships in trading models such as trends, factor loadings, or market betas:

$$y_t = \alpha_t + \beta_t z_t + \epsilon_t \tag{19}$$

where z_t may represent time, a market factor, or another explanatory variable.

The state vector is defined as:

$$\theta_t = \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix} \tag{20}$$

Prediction step:

$$\hat{\theta}_{t|t-1} = A\hat{\theta}_{t-1|t-1} \tag{21}$$

Update step:

$$\hat{\theta}_{t|t} = \hat{\theta}_{t|t-1} + K_t(y_t - H_t\hat{\theta}_{t|t-1}) \tag{22}$$

Observation matrix:

$$H_t = [1 \quad z_t] \tag{23}$$

5.5 Trading Signal Generation

Trading signals are generated based on deviations between the observed price and the estimated latent state. A typical mean-reversion signal is defined as:

$$\text{Deviation}_t = y_t - \hat{\mu}_t \tag{24}$$

Positions are initiated when the deviation exceeds a predefined threshold and exited when the deviation reverts toward zero. The Kalman Filter covariance provides an estimate of uncertainty, which can be used for dynamic position sizing and risk management.

6 Code Implementation and Explanation

6.1 KalmanFilterAverage

```
def KalmanFilterAverage(x):
    # Construct a Kalman filter
    from pykalman import KalmanFilter
    kf = KalmanFilter(transition_matrices = [1],
                      observation_matrices = [1],
                      initial_state_mean = 0,
                      initial_state_covariance = 1,
                      observation_covariance=1,
                      transition_covariance=.01)

    # Use the observed values of the price to get a rolling mean
    state_means, _ = kf.filter(x.values)
    state_means = pd.Series(state_means.flatten(), index=x.index)
    return state_means
```

This function computes a smoothed rolling mean using a Kalman Filter assuming a constant state with noise.

6.2 KalmanFilterTrend

```
    n = len(price)
    t = np.arange(n)

    # Observation matrix: [1, t]
    obs_mat = np.vstack([np.ones(n), t]).T[:, np.newaxis]

    delta = 1e-4
    trans_cov = delta / (1 - delta) * np.eye(2)

    kf = KalmanFilter(
        n_dim_obs=1,
        n_dim_state=2,
        initial_state_mean=[price.iloc[0], 0],
        initial_state_covariance=np.eye(2),
        transition_matrices=np.eye(2),
        observation_matrices=obs_mat,
        observation_covariance=1,
        transition_covariance=trans_cov
    )
```

```
20
21      state_means, state_covs = kf.filter(price.values)
22      alpha = state_means[:, 0]   # Fair value
23      beta = state_means[:, 1]    # Trend slope
24
25      return alpha, beta
```

The `KalmanFilterTrend` model applies a Kalman Filter to estimate the time-varying trend of a single asset. The observed price is modeled as a noisy measurement of an underlying latent process:

$$P_t = \alpha_t + \beta_t t + \epsilon_t, \tag{25}$$

where α_t represents the dynamic fair value of the asset and β_t captures the instantaneous trend. Both parameters are allowed to evolve over time according to a state-space formulation, enabling the model to adapt to changing market conditions.

6.3 Backtest Function

```
1  def backtest(price):
2      """
3      Mean-reversion trading using Kalman Filter fair value
           estimation
4      """
5      df = pd.DataFrame({'price': price})
6
7      # Estimate fair value and trend
8      fair_value, trend = KalmanFilterTrend(price)
9      df['fair_value'] = fair_value
10
11     # Compute deviation
12     df['deviation'] = df['price'] - df['fair_value']
13
14     # Z-score of deviation
15     lookback = 60
16     mean_dev = df['deviation'].rolling(lookback).mean()
17     std_dev = df['deviation'].rolling(lookback).std()
18     df['zscore'] = (df['deviation'] - mean_dev) / std_dev
19
20     # Trading rules
21     entry_z = 2
22     exit_z = 0
23
24     df['position'] = 0
```

```
    df.loc[df['zscore'] < -entry_z, 'position'] = 1
    df.loc[df['zscore'] >  entry_z, 'position'] = -1
    df.loc[df['zscore'].abs() < exit_z, 'position'] = 0

    df['position'] = df['position'].replace(to_replace=0, method=
        'ffill')
    df['position'].iloc[0] = 0

    # Returns
    df['returns'] = df['price'].pct_change()
    df['strategy_returns'] = df['position'].shift(1) * df['
        returns']

    df['cum_returns'] = (1 + df['strategy_returns']).cumprod()

    # Sharpe ratio
    if df['strategy_returns'].std() != 0:
        sharpe = (df['strategy_returns'].mean() /
                  df['strategy_returns'].std()) * sqrt(252)
    else:
        sharpe = 0.0

    return df, sharpe
```

The `backtest` function evaluates a mean-reversion trading strategy based on deviations of the observed price from the Kalman-filtered fair value. Trading signals are generated using standardized deviations, with long positions initiated when the price is significantly below the estimated fair value and short positions when it is significantly above. Strategy performance is assessed using cumulative returns and the annualized Sharpe ratio.

Z-score:

$$z_t = \frac{\text{Deviation}_t - \mu_t}{\sigma_t} \tag{26}$$

Sharpe ratio: Calculates cumulative portfolio returns and the Sharpe ratio for performance evaluation

$$\text{Sharpe} = \frac{\mu_{returns}}{\sigma_{returns}}\sqrt{252} \tag{27}$$

where 252 is the number of trading days in a year.

The Sharpe Ratio is a widely used metric in finance to measure the performance of an investment or trading strategy relative to its risk. It is defined as:

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p} \tag{28}$$

where:
• R_p: The mean return of the portfolio or strategy.
• R_f : The risk-free rate of return, typically the return of government bonds.
• σ_p: The standard deviation of the portfolio's returns, representing the risk or volatility.
Interpretation
• A higher Sharpe Ratio indicates better risk-adjusted returns, meaning the strategy generates higher excess returns per unit of risk.
• A negative Sharpe Ratio implies that the strategy underperforms the risk-free rate.
Annualized Sharpe Ratio
In the backtest function, the Sharpe Ratio is annualized using:

$$\text{Annualized Sharpe Ratio} = \frac{\mu_{returns}}{\sigma_{returns}}\sqrt{T} \tag{29}$$

where:
• $\mu_{returns}$: The mean daily returns of the strategy.
• $\sigma_{returns}$: The standard deviation of the daily returns.
• T : The number of trading days in a year (typically T = 252).
Limitations
While the Sharpe Ratio is a useful metric, it has certain limitations:
• It assumes returns are normally distributed, which may not always be the case.
• It does not account for the impact of extreme events or non-linear risks.
Despite these limitations, the Sharpe Ratio remains a standard tool for evaluating risk-adjusted performance in finance.

6.4 Backtesting: In-Sample and Out-of-Sample Testing

Backtesting is a critical process in financial modeling and algorithmic trading used to evaluate the performance of a strategy using historical data. It allows practitioners to understand how a model or strategy would have performed in the past, providing insights into its potential future performance. However, it is essential to follow robust practices, such as splitting data into in-sample and out-of-sample sets, to avoid overfitting and ensure realistic results.

In-Sample Testing In-sample testing involves training and calibrating the model using a subset of historical data. This process helps the model learn patterns, optimize parameters, and fit the data effectively.

- **Purpose**: To develop and fine-tune the strategy.
- **Caution**:Overfitting may occur if the model performs exceptionally well on this data but fails to generalize to unseen data.

Example: A portfolio optimization algorithm is trained on stock prices from 2010 to 2018 to determine optimal weights for assets.

Out-of-Sample Testing Out-of-sample testing evaluates the model's performance on unseen data that was not used during training. This step is crucial for testing the model's robustness and generalizability.

- Purpose: To validate the strategy's effectiveness on new data and avoid overfitting.
- Caution: A significant drop in performance from in-sample to out-of-sample indicates overfitting or poor model generalization.

Example: After training on data from 2010 to 2018, the same portfolio optimization algorithm is tested on stock prices from 2019 to 2022.

6.5 Drawdown in Backtesting

In the context of backtesting a pairs trading strategy, drawdown is defined as the largest observed loss from a peak to a trough in the value of a portfolio or trading strategy. Mathematically, the drawdown at time t is given by:

$$\text{Drawdown}(t) = \frac{V_{\text{peak}} - V_t}{V_{\text{peak}}} \tag{30}$$

Where:

- V_{peak} is the maximum value of the portfolio up to time t,
- V_t is the value of the portfolio at time t.

The maximum drawdown (MDD) is the largest value of drawdown observed during the backtesting period:

$$\text{MDD} = \max_t \text{Drawdown}(t) \tag{31}$$

Drawdown is arguably more important than the Sharpe ratio, because:

1. Mean-reversion strategies can stay wrong for long periods. Prices can keep moving away from the estimated fair value, causing extended losses.
2. Kalman filters adapt, but not instantly. During regime shifts, the estimated fair value may lag, increasing drawdown.

6.6 Conclusion

The Kalman Filter is a powerful tool for real-time estimation in trading. It provides robust parameter estimates, enabling traders to identify and act on profitable opportunities. By integrating statistical tests like the Hurst exponent and ADF test, as well as computing the half-life of mean reversion, the Kalman Filter-based approach ensures a disciplined and systematic trading strategy.