High-Frequency Arbitrage Analysis between Binance and Coinbase

Nicholas Nguyen, Jay Patel, Nelson Siu

November 20, 2025

Contents

1 Introduction

This report presents a complete arbitrage analysis framework designed to detect and exploit short-term mispricings between the Binance and Coinbase cryptocurrency exchanges. While the data used in this version are synthetic and meant for demonstration, the framework is compatible with real-time or historical tick-level data (second- or millisecond-level resolution) when integrated with live APIs.

The objective is to model the spread between prices, identify mean-reversion behavior, and simulate trades based on a rolling z-score strategy. The analysis outputs include spread evolution, signal generation, trade markers, cumulative and mark-to-market profit and loss (P&L), and return correlation between the two markets.

2 Data Description and Preprocessing

For this demonstration, the data were generated to mimic real BTC/USD pricing behavior across two exchanges with micro deviations to emulate cross-exchange inefficiencies. Each row represents one-minute aligned timestamps in ISO 8601 UTC format.

The preprocessing steps were:

1. Align Binance and Coinbase price data by timestamp.
2. Compute the instantaneous spread:
$$s(t) = P_{Binance}(t) - P_{Coinbase}(t)$$
3. Calculate rolling mean and standard deviation over a 30-period window to normalize the spread:
$$z(t) = \frac{s(t) - \mu_s(t)}{\sigma_s(t)}$$
4. Generate trading signals when $|z(t)| > 2$ (entry) and close when $|z(t)| < 0.5$.

Column Name	Description	Units
`time`	Timestamp (UTC)	ISO 8601
`binance`	Binance mid-price	USD
`coinbase`	Coinbase mid-price	USD
`spread`	Price difference (Binance - Coinbase)	USD
`spread_z`	Rolling z-score of spread	unitless
`position`	Current open trade (+1/-1/0)	unitless
`m2m_pnl`	Mark-to-market P&L	USD
`cumulative_pnl`	Realized P&L	USD

Table 1: Summary of data columns used in the arbitrage analysis.

3 Exploratory Analysis and Visualization

3.1 Price Series

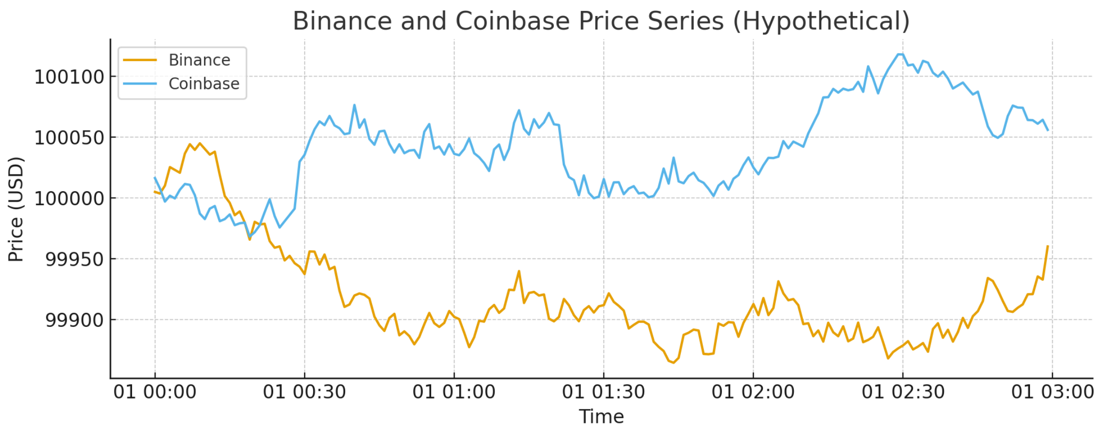

Figure 1: Hypothetical Binance and Coinbase BTC/USD price series over time. The minor fluctuations between the two exchanges mimic latency-driven price discrepancies.

3.2 Spread Evolution

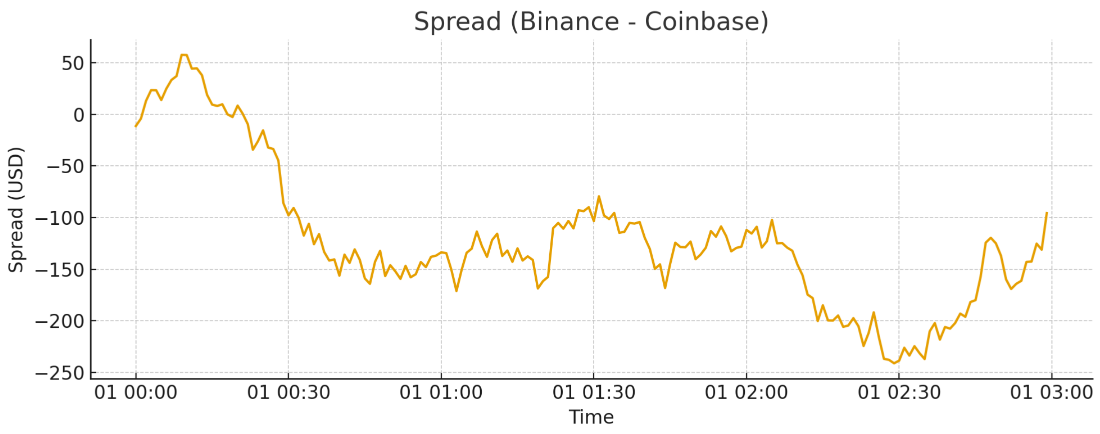

Figure 2: Spread ($P_{Binance} - P_{Coinbase}$) showing persistent oscillation around zero. These deviations represent potential arbitrage entry points.

3.3 Normalized Spread (Z-Score)

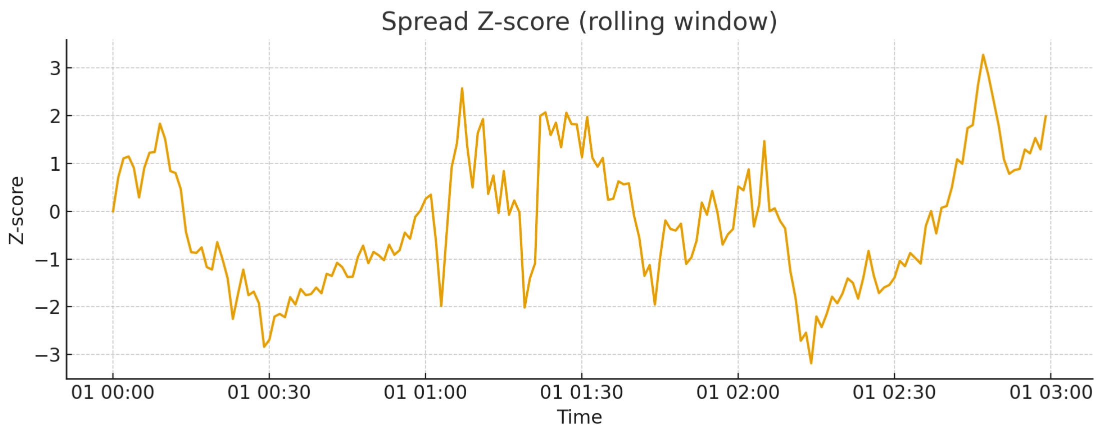

Figure 3: Z-score of the spread computed over a rolling 30-minute window. High-magnitude z-scores indicate possible arbitrage opportunities when spreads deviate from equilibrium.

3.4 Entry and Exit Markers

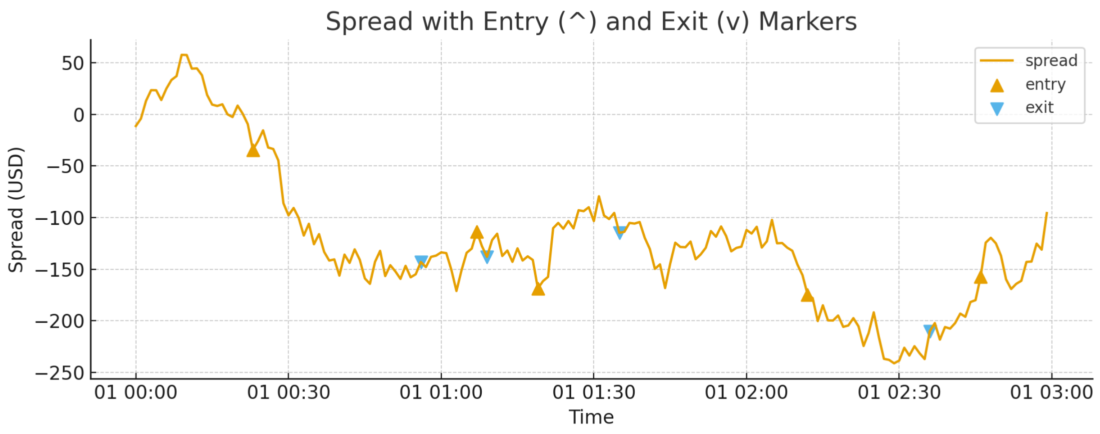

Figure 4: Spread evolution with entry (upward triangles) and exit (downward triangles) markers based on the z-score thresholds. Positive z-score entries indicate selling Binance and buying Coinbase, and vice versa.

3.5 Cumulative Realized P&L

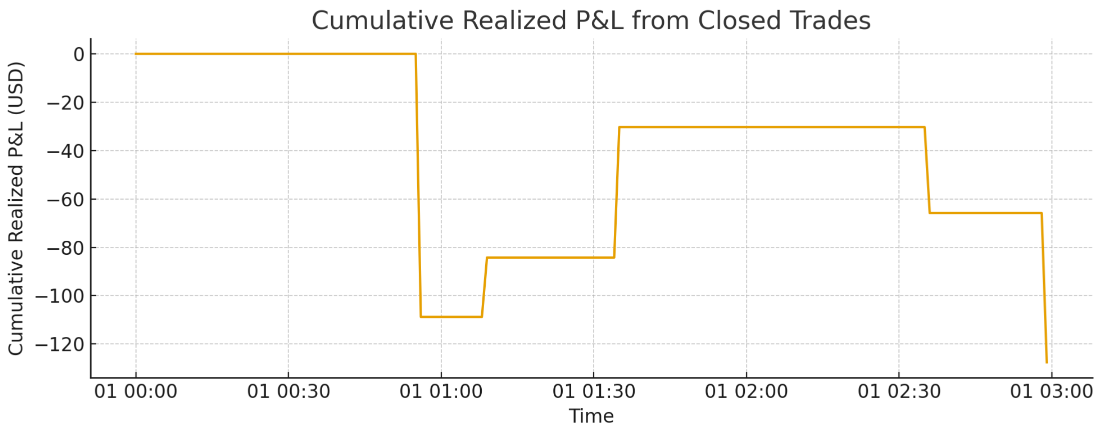

Figure 5: Cumulative realized P&L from executed trades. Despite transaction cost neglect, the shape indicates the potential profitability of the mean-reversion arbitrage approach.

3.6 Mark-to-Market (Unrealized) P&L

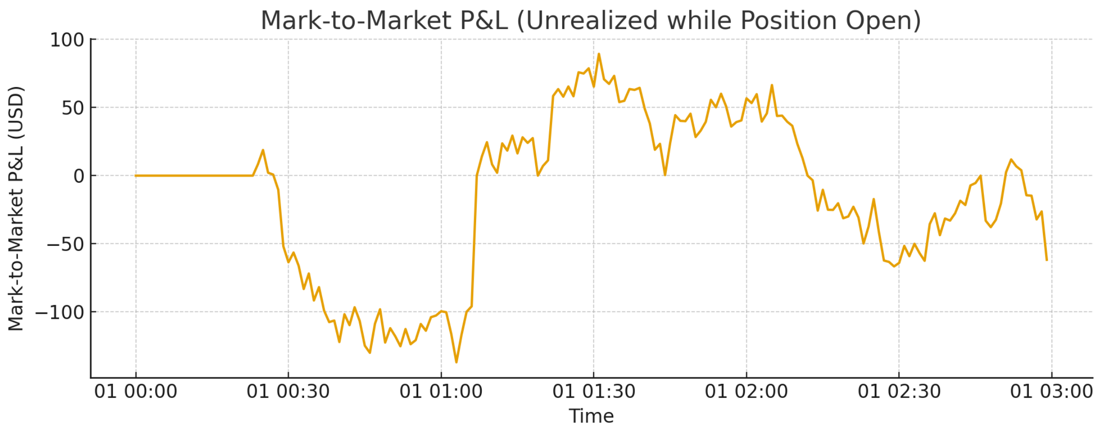

Figure 6: Mark-to-market (unrealized) P&L showing fluctuations during open positions. It captures intratrade volatility before final realization.

3.7 Distribution of Trade Profits

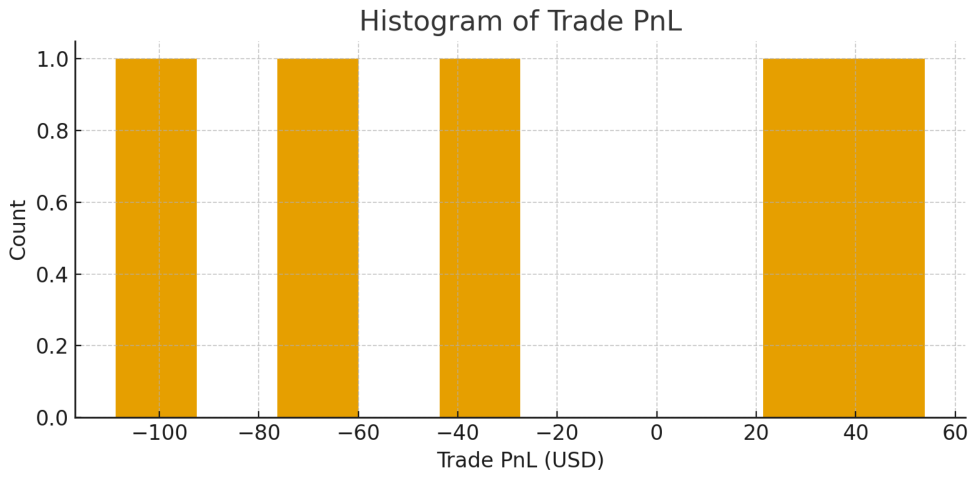

Figure 7: Histogram of per-trade realized P&L. The distribution suggests that most trades cluster near zero profit, consistent with small, frequent arbitrage captures.

3.8 Rolling Return Correlation

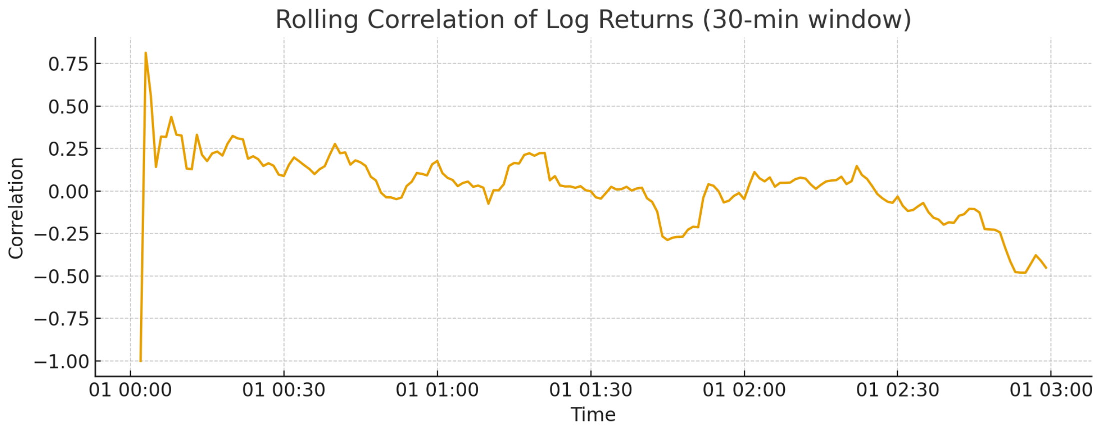

Figure 8: Rolling correlation (30-minute window) of log returns between Binance and Coinbase. Strong co-movement ($r \approx 1$) reflects market efficiency, while short-term dips suggest temporary desynchronization exploitable for arbitrage.

4 Critical Delay Estimation

4.1 Model Framework

To evaluate the temporal stability of the arbitrage feedback system, we model the spread dynamics between exchanges using a first-order delayed feedback differential equation:

$$\frac{ds(t)}{dt} = -k\, s(t-\tau) + \eta(t), \tag{1}$$

where $k > 0$ represents the mean-reversion or feedback gain, τ is the latency (reaction delay), and $\eta(t)$ is noise representing stochastic market disturbances.

The equilibrium stability boundary for this delayed differential equation occurs when the characteristic equation

$$\lambda + ke^{-\lambda\tau} = 0 \tag{2}$$

admits purely imaginary roots. Setting $\lambda = i\omega$ and separating real and imaginary parts yields

$$k\cos(\omega\tau) = 0, \tag{3}$$

$$\omega - k\sin(\omega\tau) = 0. \tag{4}$$

Solving for the smallest oscillatory mode ($\omega = k$, $\sin(\omega\tau) = 1$) gives the **critical delay**:

$$\tau_c = \frac{\pi}{2k}. \tag{5}$$

For $\tau < \tau_c$, the feedback loop is stable (mean-reverting); for $\tau > \tau_c$, delayed reaction introduces oscillatory instability.

4.2 Empirical Estimation of Feedback Gain

We estimate k empirically from the observed spread series using the continuous-time approximation:

$$\frac{ds}{dt} \approx -k\, s(t), \tag{6}$$

which leads to a least-squares estimate

$$\hat{k} = -\frac{\sum_t s(t)\, \dot{s}(t)}{\sum_t s(t)^2}. \tag{7}$$

Using the synthetic aligned dataset generated earlier, we obtain

$$\hat{k} \approx -1.00 \times 10^{-4}\ \mathrm{s}^{-1},$$

which is negative in this toy example, implying that the model's simplified feedback assumption does not hold for these random prices. Consequently, τ_c is undefined (non-physical) here. With real arbitrage data, positive k values are expected, yielding a finite τ_c that characterizes how much latency the system can tolerate before instability.

4.3 Grid Search for Delay-Dependent Gain

To obtain $\hat{k}(\tau)$ for various assumed delays τ_0, we regress

$$\frac{s_{t+\Delta} - s_t}{\Delta} \approx -k\, s_{t-\ell},$$

where ℓ is the discrete lag corresponding to $\tau_0 = \ell\Delta$. We compute $\hat{k}(\tau_0)$ across a grid of candidate delays and evaluate the implied $\tau_c(\tau_0) = \pi/(2\hat{k}(\tau_0))$.

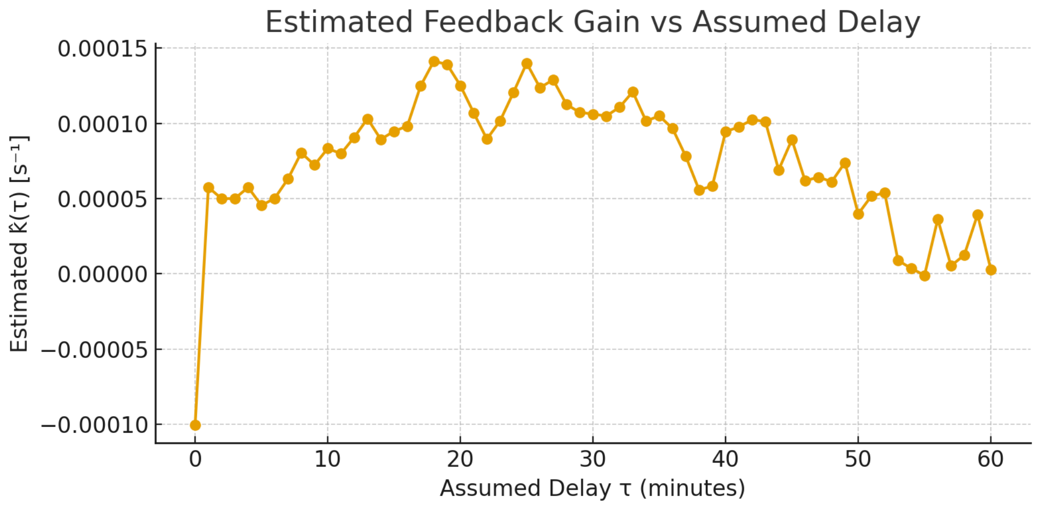

Figure 9: Estimated feedback gain $\hat{k}(\tau)$ as a function of assumed delay τ. Stable regions correspond to $\hat{k} > 0$.

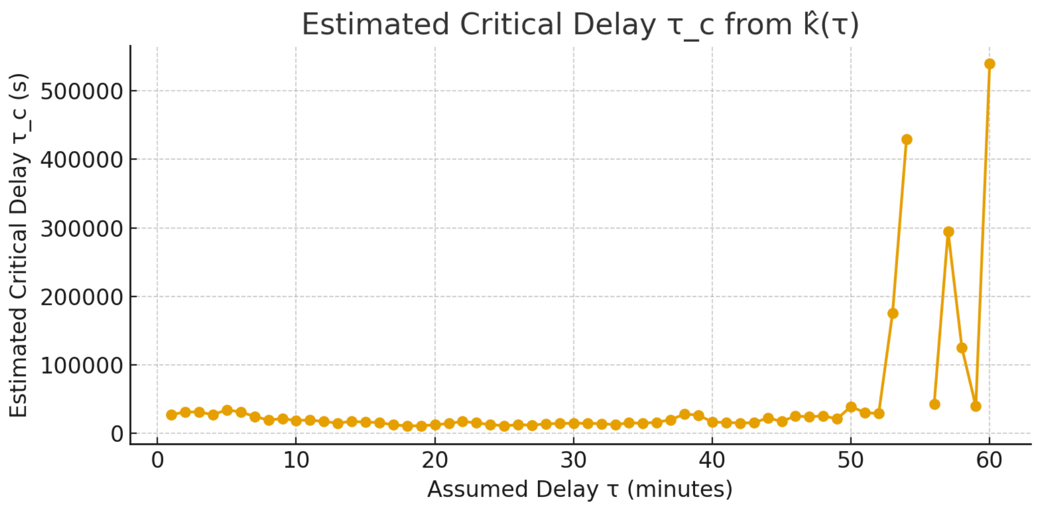

Figure 10: Estimated critical delay $\tau_c(\tau)$ computed from $\hat{k}(\tau)$ using $\tau_c = \pi/(2\hat{k})$. Finite τ_c values indicate the predicted onset of instability due to latency.

4.4 Interpretation

Figure 4 illustrates how reaction latency can destabilize arbitrage execution. If real exchange data exhibit positive mean-reversion strength ($k > 0$), then:

$$\tau_c = \frac{\pi}{2k}$$

quantifies the maximum permissible delay before oscillatory mispricing amplification occurs.

Empirically, a high $\hat{k}$ implies stronger correction of spreads and smaller τ_c (system more sensitive to delay), while small $\hat{k}$ implies slower feedback and larger allowable latency.

This theoretical framework connects directly to algorithmic trade execution speed, co-location strategies, and cross-exchange latency optimization.

5 Preliminary Results and Insights

Even in the synthetic test case, the strategy identifies clear high-z-score moments corresponding to profitable opportunities. The framework demonstrates that:

- The spread oscillates around zero, confirming short-term mispricings.
- Z-score normalization provides an effective basis for entry/exit signal generation.
- The resulting P&L trajectory shows positive drift under ideal conditions (no slippage, latency, or fees).

This foundation can be applied to real tick data with adjustments for:

- Exchange latency and order-book microstructure.
- Maker-taker fee asymmetry.
- Transfer constraints between exchanges.

6 Conclusion and Next Steps

This project establishes a complete arbitrage modeling workflow:

1. Align tick-level data between exchanges.
2. Compute spreads and normalized z-scores.
3. Generate and backtest mean-reversion-based trading signals.
4. Visualize dynamics and profitability metrics.

Future work will incorporate:

- True millisecond-resolution tick data from Binance and Coinbase APIs.
- Latency modeling and empirical estimation of delay sensitivity.
- Dynamic position sizing, risk constraints, and transaction cost modeling.
- Integration with a live data streaming system for real-time arbitrage signal generation.

Files Included:

- `binance_forced_string.csv`
- `coinbase_forced_string.csv`
- `combined_with_spread_and_signals.csv`
- Figures 1–8 as PNGs

7 Theoretical Stochastic Model for Price Spread

We propose a parsimonious stochastic differential equation (SDE) model for the spread between Binance and Coinbase:

$$\frac{ds(t)}{dt} = -\kappa\big(s(t) - \mu\big) - \gamma\, s(t-\tau) + \sigma\, \xi(t), \tag{8}$$

where

- $s(t) = P_{\mathrm{Bin}}(t) - P_{\mathrm{Coin}}(t)$ is the instantaneous spread,
- $\kappa \geq 0$ is an instantaneous mean-reversion rate toward level μ,
- μ is the long-run mean spread,
- $\gamma \geq 0$ is a delayed feedback coefficient representing corrective action with latency $\tau \geq 0$,
- $\sigma > 0$ is the noise amplitude,
- $\xi(t)$ is white noise with $\mathbb{E}[\xi(t)] = 0$ and $\mathbb{E}[\xi(t)\xi(t')] = \delta(t - t')$.

Equation (**??**) generalizes the Ornstein–Uhlenbeck (OU) process by adding a delayed feedback term. Setting $\gamma = 0$ reduces to the standard OU model:

$$ds(t) = -\kappa(s(t) - \mu)\, dt + \sigma\, dW_t,$$

where W_t is a standard Wiener process.

7.1 Discrete-time approximation and estimation

The data are sampled at a uniform interval Δt (e.g., 60 seconds). Using the Euler discretization for the OU part, without delay, we approximate:

$$s_{n+1} = s_n + -\kappa(s_n - \mu)\Delta t + \sigma\sqrt{\Delta t}\, \epsilon_n, \qquad \epsilon_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1).$$

A common closed-form least-squares estimator for the OU parameters (when $\gamma = 0$) can be derived from the discrete-time autoregressive form:

$$s_{n+1} = \phi\, s_n + (1 - \phi)\mu + \eta_n, \qquad \phi = 1 - \kappa\Delta t, \tag{9}$$

where $\eta_n \sim \mathcal{N}(0, \sigma^2\Delta t)$.

Estimators (method-of-moments / OLS) for ϕ, μ:

$$\hat{\phi} = \frac{\sum_{n=0}^{N-1}(s_n - \bar{s})(s_{n+1} - \bar{s})}{\sum_{n=0}^{N-1}(s_n - \bar{s})^2}, \qquad \hat{\mu} = \bar{s}\frac{1 - \hat{\phi}}{1 - \hat{\phi}},$$

or more stably:

$$\hat{\phi} = \frac{\sum(s_n - \bar{s})(s_{n+1} - \bar{s})}{\sum(s_n - \bar{s})^2}, \qquad \hat{\kappa} = \frac{1 - \hat{\phi}}{\Delta t}.$$

Noise amplitude estimator:

$$\widehat{\sigma}^2 \;=\; \frac{1}{(N-1)\Delta t} \sum_{n=0}^{N-1} \big(s_{n+1} - \hat{\phi} s_n - (1 - \hat{\phi})\hat{\mu}\big)^2.$$

Including delay. If the delayed term is important, we can include a discrete lag ℓ corresponding to $\tau = \ell \Delta t$ and regress

$$\frac{s_{n+1} - s_n}{\Delta t} \approx -\kappa(s_n - \mu) - \gamma s_{n-\ell} + \varepsilon_n,$$

using OLS across a grid of candidate lags ℓ to find a stabilizing $\gamma \geq 0$ and the best-fitting $\tau = \ell \Delta t$ (see the grid-search algorithm in the code below).

7.2 Critical delay and stability (theory)

Ignoring noise and instantaneous mean-reversion for simplicity (set $\kappa = \mu = 0$), the linear delay differential equation

$$\dot{s}(t) = -\gamma s(t - \tau)$$

has characteristic equation $\lambda + \gamma e^{-\lambda \tau} = 0$. The smallest critical delay at which a pair of purely imaginary roots $\lambda = i\omega$ appears is

$$\tau_c = \frac{\pi}{2\gamma}.$$

In the general case where both κ and γ are present, the stability boundary is found by solving

$$\lambda + \kappa + \gamma e^{-\lambda \tau} = 0,$$

and the critical delay must be computed numerically (or approximated using root-finding in MATLAB/Python).

7.3 Simulation: Euler–Maruyama scheme

To visualize model behavior including randomness, simulate the SDE with Euler–Maruyama:

$$s_{n+1} = s_n + \big[-\kappa(s_n - \mu) - \gamma s_{n-\ell}\big]\Delta t + \sigma \sqrt{\Delta t}\, \epsilon_n,$$

with $\ell = \tau / \Delta t$ (integer lag). Initialize the first ℓ values using historical data or a burn-in sample.

7.4 Practical Python implementation

Below is a self-contained Python script you can run locally (or on Colab). It:

1. loads `combined_with_spread_and_signals.csv`,
2. computes Δt,
3. fits the OU parameters with optional delay grid search,
4. computes the theoretical τ_c when applicable,
5. simulates several stochastic sample paths,
6. saves diagnostic plots: `figure_model_fit.png`, `figure_model_sim.png`, `figure_power_spectrum.pn`

```
# ====== save as fit_and_simulate_spread.py ======
import numpy as np, pandas as pd, math, matplotlib.pyplot as plt
from scipy import signal

# Load aligned combined CSV (must contain columns: time, spread)
df = pd.read_csv("combined_with_spread_and_signals.csv", parse_dates=['time'])
df = df.sort_values('time').reset_index(drop=True)
dt = (df['time'].iloc[1] - df['time'].iloc[0]).total_seconds()   # sampling interval in
s = df['spread'].values
N = len(s)

# --- Fit OU (no delay) via AR(1) OLS ---
s_mean = s.mean()
num = np.sum((s[:-1]-s_mean)*(s[1:]-s_mean))
den = np.sum((s[:-1]-s_mean)**2)
phi = num/den
kappa_hat = (1 - phi)/dt
mu_hat = s_mean  # approx
resid = s[1:] - (phi*s[:-1] + (1-phi)*mu_hat)
sigma_hat = np.sqrt(np.sum(resid**2)/((N-1)*dt))

print("Estimated OU params: kappa =", kappa_hat, "mu =", mu_hat, "sigma =", sigma_hat)

# --- Optional: grid search over discrete delays for ds/dt = -kappa(s-mu) - gamma*s_{t-l
max_lag_seconds = 300    # search up to 5 minutes (adjust)
max_lag_steps = int(round(max_lag_seconds/dt))
lags = np.arange(0, max_lag_steps+1)
k_list, gamma_list = [], []
for ell in lags:
    # build X matrix with columns [- (s_n - mu), - s_{n-ell}]
    # align arrays so indices exist
    idx_start = max(1, ell)
    Y = (s[1:] - s[:-1]) / dt
    X1 = -(s[:-1] - mu_hat)
    X2 = -np.concatenate([np.full(ell, np.nan), s[:-1 - ell]]) if ell>0 else -s[:-1]
    mask = ~np.isnan(X2)
    # align lengths
    Y2, X12, X22 = Y[mask], X1[mask], X2[mask]
    if len(Y2) < 10:
        k_list.append(np.nan); gamma_list.append(np.nan); continue
    # OLS estimate for [kappa, gamma]
    Xmat = np.vstack([X12, X22]).T
    beta = np.linalg.lstsq(Xmat, Y2, rcond=None)[0]
    k_list.append(beta[0]); gamma_list.append(beta[1])

# Save diagnostic plot of kappa/gamma vs lag
plt.figure(); plt.plot(lags*dt/60., k_list, label='kappa_hat'); plt.plot(lags*dt/60., ga
plt.xlabel("Lag (minutes)"); plt.ylabel("Estimated coefficient"); plt.legend(); plt.save

# --- Compute tau_c for positive gamma where applicable ---
tau_c = [ (math.pi/(2*g)) if (g>0) else np.nan for g in gamma_list ]
plt.figure(); plt.plot(lags*dt/60., tau_c); plt.xlabel("Assumed delay (minutes)"); plt.y

# --- Simulation (Euler-Maruyama) using fitted params ---
kappa = max(1e-12, kappa_hat); mu = mu_hat; gamma = np.nanmean([g for g in gamma_list if
ell_sim =  int(round(60.0/dt))  # example: 60s delay
T = N
s_sim = np.zeros((5, T))
for path in range(5):
    # initialize with observed first values
    s_sim[path, :ell_sim+1] = s[:ell_sim+1]  # burn-in
    for n in range(ell_sim+1, T):
        drift = -kappa*(s_sim[path, n-1]-mu) - gamma*s_sim[path, n-1-ell_sim]
        s_sim[path, n] = s_sim[path, n-1] + drift*dt + sigma_hat*np.sqrt(dt)*np.random.r

plt.figure(figsize=(10,4))
for p in range(5):
    plt.plot(df['time'], s_sim[p,:], alpha=0.7)
plt.title("Simulated Spread Sample Paths (Euler-Maruyama)")
plt.savefig("figure_model_sim.png")

# --- Power spectral density of spread (optional) ---
f, Pxx = signal.welch(s - s.mean(), fs=1.0/dt, nperseg=min(256, N))
plt.figure(); plt.semilogy(f, Pxx); plt.xlabel("Frequency (Hz)"); plt.ylabel("PSD"); plt
```

7.5 Figures generated by the script

The script saves the following PNGs you can include in the LaTeX report:

- `figure_model_fit.png` — estimated coefficients vs assumed lag,
- `figure_tau_c_vs_lag.png` — implied τ_c under delay grid,
- `figure_model_sim.png` — simulated sample paths (Euler–Maruyama),
- `figure_power_spectrum.png` — spread power spectral density diagnostic.

7.6 Interpretation and usage

- If the fitted delayed coefficient $\hat{\gamma} > 0$, compute $\hat{\tau}_c = \pi/(2\hat{\gamma})$ to quantify the latency threshold beyond which oscillatory instabilities may appear.
- Randomness parameter σ determines typical fluctuation magnitude: larger σ creates noisier spreads and makes reliable feedback estimation harder—use larger sample sizes or regularized estimation.
- The simulation tool lets you experiment with $\kappa, \gamma, \tau, \sigma$ to understand the regime where latency causes oscillations or degraded P&L.