In [ ]:
```
pip install cma
```

```
Collecting cma
  Downloading cma-4.2.0-py3-none-any.whl.metadata (7.7 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from cma) (2.0.2)
Downloading cma-4.2.0-py3-none-any.whl (288 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 288.2/288.2 kB 5.0 MB/s eta 0:00:00
Installing collected packages: cma
Successfully installed cma-4.2.0
```

In [ ]:
```
```

In [ ]:
```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("BTC_5min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

df_train['log_mid'] = np.log(df_train['midpoint'])
df_train['returns'] = df_train['log_mid'].diff().fillna(0)
df_cv['log_mid'] = np.log(df_cv['midpoint'])
df_cv['returns'] = df_cv['log_mid'].diff().fillna(0)
df_test['log_mid'] = np.log(df_test['midpoint'])
df_test['returns'] = df_test['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
```

```
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0, 0, 0, 0.005, 0.005], sigma0=0.2, options={'seed'
    return res[0][:3], res[0][3:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(0, 200), (200, 400), (400, 600), (600, 800), (800, 1000
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
```

```
        if end > len(df_train):
            continue
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], windo
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        raise ValueError("No positions generated.")

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
```

```
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

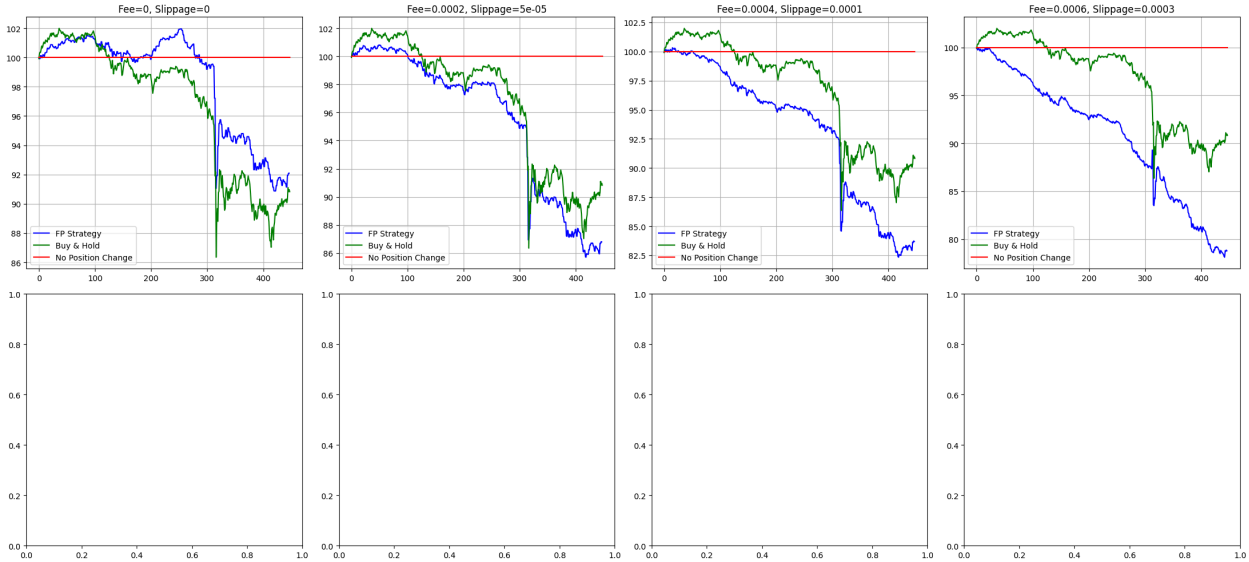
```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:06
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.205561248416842e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.1
    2     16 -9.998120739002303e-03 1.1e+00 1.56e-01  1e-01  2e-01 0:00.1
    3     24 -9.998120739002303e-03 1.3e+00 1.50e-01  1e-01  1e-01 0:00.1
    4     32 -9.998120739002303e-03 1.3e+00 1.41e-01  1e-01  1e-01 0:00.2
termination on tolflatfitness=1 (Tue Jul 22 12:55:06 2025)
final/bestever f-value = -9.998121e-03 -1.205561e-02 after 33/1 evaluations
incumbent solution: [ 0.31590449, -0.04909103, 0.04118488, -0.04839484, 0.00552
435]
std deviation: [0.14594795, 0.12267427, 0.12276597, 0.13515398, 0.12102708]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:06
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.291500981654004e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.291500981654004e-02 1.1e+00 1.64e-01  2e-01  2e-01 0:00.0
    3     24 -1.291500981654004e-02 1.3e+00 1.59e-01  1e-01  2e-01 0:00.1
    5     40 -1.291500981654004e-02 1.7e+00 1.62e-01  1e-01  2e-01 0:00.1
termination on tolfun=1e-11 (Tue Jul 22 12:55:06 2025)
final/bestever f-value = -1.291501e-02 -1.291501e-02 after 41/1 evaluations
incumbent solution: [ 0.40011878, -0.06385712, 0.35694482, -0.01333117, -0.1498
5136]
std deviation: [0.15644004, 0.15241561, 0.15801123, 0.14376748, 0.14718481]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:06
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.155850104232492e-02 1.0e+00 1.77e-01  2e-01  2e-01 0:00.0
    2     16 -1.155850104232492e-02 1.2e+00 1.96e-01  2e-01  2e-01 0:00.1
    3     24 -1.208476650053392e-02 1.5e+00 2.12e-01  2e-01  2e-01 0:00.1
   47    376 -2.382440409724573e-02 5.6e+00 2.16e-02  7e-03  1e-02 0:01.0
termination on tolflatfitness=1 (Tue Jul 22 12:55:07 2025)
final/bestever f-value = -2.382440e-02 -2.578270e-02 after 377/160 evaluations
incumbent solution: [ 0.6873574, -0.360027, -0.16736631, 0.29529307, -0.980733,
]
std deviation: [0.01155544, 0.00689992, 0.01420874, 0.01275234, 0.0105386, ]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:07
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.984921417001885e-02 1.0e+00 1.86e-01  2e-01  2e-01 0:00.0
    2     16 -4.171318765821752e-02 1.3e+00 2.28e-01  2e-01  2e-01 0:00.1
    3     24 -4.284401647458225e-02 1.4e+00 2.29e-01  2e-01  2e-01 0:00.1
   29    232 -4.193922019478080e-02 9.5e+00 1.29e-01  6e-02  2e-01 0:00.7
termination on tolflatfitness=1 (Tue Jul 22 12:55:08 2025)
final/bestever f-value = -4.193922e-02 -4.520543e-02 after 233/37 evaluations
incumbent solution: [ 0.3581037, 0.81814676, -0.03697863, 0.35502899, -1.013728
27]
std deviation: [0.17450989, 0.09583299, 0.07821421, 0.144468, 0.06009161]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:08
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.600736986859786e-02 1.0e+00 2.19e-01  2e-01  2e-01 0:00.0
    2     16 -1.277184547451071e-02 1.4e+00 2.12e-01  2e-01  2e-01 0:00.0
    3     24 -2.076867108979741e-02 1.3e+00 2.30e-01  2e-01  3e-01 0:00.0
```

```
   82    656 -4.521080898880570e-02 1.7e+01 9.72e-03  1e-03  6e-03 0:01.0
termination on tolflatfitness=1 (Tue Jul 22 12:55:09 2025)
final/bestever f-value = -4.521081e-02 -4.521081e-02 after 657/554 evaluations
incumbent solution: [ 0.06560135, -0.5050873, 0.55714296, -0.08359742, 0.578675
37]
std deviation: [0.0011962, 0.00233316, 0.00619348, 0.00269375, 0.00533027]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:10
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.130561248416842e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -9.748120739002303e-03 1.1e+00 1.94e-01  2e-01  2e-01 0:00.0
    3     24 -9.748120739002303e-03 1.4e+00 2.06e-01  2e-01  3e-01 0:00.0
    4     32 -9.748120739002303e-03 1.6e+00 2.23e-01  2e-01  3e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:10 2025)
final/bestever f-value = -9.748121e-03 -1.130561e-02 after 33/1 evaluations
incumbent solution: [ 0.38166952, 0.18211967, 0.23122277, -0.34284639, -0.26470
548]
std deviation: [0.27954971, 0.19515199, 0.21877682, 0.2214751, 0.19157162]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:10
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.266500981654003e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.266500981654003e-02 1.2e+00 1.73e-01  1e-01  2e-01 0:00.0
    3     24 -1.266500981654003e-02 1.3e+00 1.69e-01  1e-01  2e-01 0:00.0
termination on tolfun=1e-11 (Tue Jul 22 12:55:10 2025)
final/bestever f-value = -1.266501e-02 -1.266501e-02 after 25/1 evaluations
incumbent solution: [ 0.2655284, 0.12755054, -0.0098567, -0.18169926, -0.03262
7, ]
std deviation: [0.16786853, 0.13567847, 0.16333954, 0.17735817, 0.15726892]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:10
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.130850104232492e-02 1.0e+00 1.77e-01  2e-01  2e-01 0:00.0
    2     16 -1.130850104232492e-02 1.2e+00 1.96e-01  2e-01  2e-01 0:00.0
    3     24 -1.130850104232492e-02 1.5e+00 2.09e-01  2e-01  3e-01 0:00.0
    5     40 -1.130850104232492e-02 1.8e+00 2.71e-01  2e-01  3e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:10 2025)
final/bestever f-value = -1.130850e-02 -1.130850e-02 after 41/1 evaluations
incumbent solution: [ 0.71352422, -0.45180045, -0.00177631, -0.07845463, 0.1232
6766]
std deviation: [0.33509148, 0.26769813, 0.25033114, 0.25675303, 0.24335033]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:10
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.959921417001885e-02 1.0e+00 1.86e-01  2e-01  2e-01 0:00.0
    2     16 -4.096318765821752e-02 1.3e+00 2.28e-01  2e-01  2e-01 0:00.0
    3     24 -4.209401647458225e-02 1.4e+00 2.29e-01  2e-01  2e-01 0:00.0
   18    144 -4.096318765821752e-02 2.8e+00 1.10e-01  6e-02  1e-01 0:00.2
termination on tolflatfitness=1 (Tue Jul 22 12:55:10 2025)
final/bestever f-value = -4.096319e-02 -4.916028e-02 after 145/99 evaluations
incumbent solution: [ 0.70377758, 0.33657809, 0.2502649, 0.12661194, -0.6021867
4]
std deviation: [0.08967546, 0.08725592, 0.10266319, 0.08643766, 0.05956808]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:10
```

```
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.800736986859786e-02 1.0e+00 2.19e-01  2e-01  2e-01 0:00.0
    2     16 -1.079724949586594e-02 1.4e+00 1.87e-01  2e-01  2e-01 0:00.0
    3     24 -5.079247795850506e-03 1.4e+00 1.75e-01  2e-01  2e-01 0:00.0
   81    648 -5.896943956458636e-02 5.2e+01 1.11e-02  8e-04  2e-02 0:01.0
termination on tolflatfitness=1 (Tue Jul 22 12:55:11 2025)
final/bestever f-value = -5.896944e-02 -5.896944e-02 after 649/459 evaluations
incumbent solution: [-0.10534958, 0.06501391, 0.87219455, -0.98010885, -0.59600
393]
std deviation: [0.00178792, 0.00077632, 0.01407956, 0.01880077, 0.00080085]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:12
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.055561248416842e-02 1.0e+00 1.71e-01  2e-01  2e-01 0:00.0
    2     16 -9.498120739002303e-03 1.2e+00 1.71e-01  1e-01  2e-01 0:00.0
    3     24 -9.498120739002303e-03 1.3e+00 1.73e-01  2e-01  2e-01 0:00.0
   70    560 -3.945513118105799e-02 1.5e+01 2.16e-02  1e-02  2e-02 0:00.9
termination on tolflatfitness=1 (Tue Jul 22 12:55:13 2025)
final/bestever f-value = -3.945513e-02 -4.029294e-02 after 561/377 evaluations
incumbent solution: [ 0.24939841, -0.44297941, -0.47368071, -0.21551126, -0.327
43751]
std deviation: [0.00969337, 0.01101288, 0.02282395, 0.01127726, 0.00953502]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:13
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.241500981654003e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.241500981654003e-02 1.2e+00 1.73e-01  1e-01  2e-01 0:00.0
    3     24 -1.241500981654003e-02 1.3e+00 1.69e-01  1e-01  2e-01 0:00.0
termination on tolfun=1e-11 (Tue Jul 22 12:55:13 2025)
final/bestever f-value = -1.241501e-02 -1.241501e-02 after 25/1 evaluations
incumbent solution: [ 0.2655284, 0.12755054, -0.0098567, -0.18169926, -0.03262
7, ]
std deviation: [0.16786853, 0.13567847, 0.16333954, 0.17735817, 0.15726892]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:13
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.105850104232492e-02 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -1.105850104232492e-02 1.2e+00 1.80e-01  2e-01  2e-01 0:00.0
    3     24 -1.105850104232492e-02 1.4e+00 1.65e-01  1e-01  2e-01 0:00.0
    4     32 -1.105850104232492e-02 1.6e+00 1.62e-01  1e-01  2e-01 0:00.1
termination on tolfun=1e-11 (Tue Jul 22 12:55:13 2025)
final/bestever f-value = -1.105850e-02 -1.105850e-02 after 33/1 evaluations
incumbent solution: [ 0.28190102, -0.01916817, 0.33630567, 0.001244, -0.0272612
8]
std deviation: [0.16930979, 0.12653961, 0.16293266, 0.15105468, 0.14760307]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:13
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.934921417001885e-02 1.0e+00 1.86e-01  2e-01  2e-01 0:00.0
    2     16 -4.021318765821752e-02 1.3e+00 2.28e-01  2e-01  2e-01 0:00.0
    3     24 -4.134401647458225e-02 1.4e+00 2.29e-01  2e-01  2e-01 0:00.0
   29    232 -4.021318765821752e-02 8.5e+00 2.38e-01  9e-02  3e-01 0:00.4
termination on tolflatfitness=1 (Tue Jul 22 12:55:13 2025)
```

```
final/bestever f-value = -4.021319e-02 -4.840355e-02 after 233/31 evaluations
incumbent solution: [ 0.93570147, 0.86734003, -0.29676034, 0.80940813, -0.86890
442]
std deviation: [0.20081519, 0.16943903, 0.34895125, 0.22746912, 0.09026084]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:14
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.000736986859786e-02 1.0e+00 2.14e-01  2e-01  2e-01 0:00.0
    2     16 -7.708079615686994e-03 1.3e+00 1.89e-01  2e-01  2e-01 0:00.0
    3     24 -2.629423047120167e-02 1.4e+00 1.82e-01  2e-01  2e-01 0:00.0
   58    464 -5.290586632555988e-02 4.5e+01 2.20e-02  1e-03  2e-02 0:00.7
termination on tolflatfitness=1 (Tue Jul 22 12:55:14 2025)
final/bestever f-value = -5.290587e-02 -5.364453e-02 after 465/372 evaluations
incumbent solution: [-0.12361798, -0.03012147, 0.64897389, -1.03326038, -0.0455
3568]
std deviation: [0.0046016, 0.00131285, 0.01946853, 0.02224319, 0.01108534]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:15
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.355612484168422e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.0
    2     16 -9.098120739002303e-03 1.3e+00 1.68e-01  2e-01  2e-01 0:00.0
    3     24 -9.098120739002303e-03 1.5e+00 1.71e-01  2e-01  2e-01 0:00.0
    5     40 -9.098120739002303e-03 1.5e+00 2.05e-01  2e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:15 2025)
final/bestever f-value = -9.098121e-03 -9.355612e-03 after 41/1 evaluations
incumbent solution: [ 0.23929485, 0.42119757, -0.1225442, 0.20442031, 0.0093856
8]
std deviation: [0.18105081, 0.21763304, 0.19715551, 0.20336185, 0.18199153]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:15
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.201500981654004e-02 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     16 -1.478668317634206e-02 1.3e+00 1.52e-01  1e-01  2e-01 0:00.0
    3     24 -1.201500981654004e-02 1.3e+00 1.43e-01  1e-01  1e-01 0:00.0
    5     40 -1.201500981654004e-02 1.4e+00 1.43e-01  1e-01  1e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:15 2025)
final/bestever f-value = -1.201501e-02 -1.478668e-02 after 41/10 evaluations
incumbent solution: [ 0.16639756, 0.22433235, 0.1971701, -0.13693955, 0.0329301
7]
std deviation: [0.14351349, 0.12799919, 0.13543524, 0.13111079, 0.11781487]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:15
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.065850104232492e-02 1.0e+00 1.68e-01  2e-01  2e-01 0:00.0
    2     16 -1.065850104232492e-02 1.3e+00 1.64e-01  1e-01  2e-01 0:00.0
    3     24 -1.065850104232492e-02 1.4e+00 1.63e-01  1e-01  2e-01 0:00.0
termination on tolflatfitness=1 (Tue Jul 22 12:55:15 2025)
final/bestever f-value = -1.065850e-02 -1.065850e-02 after 25/1 evaluations
incumbent solution: [ 0.18139731, 0.0062772, 0.33740101, 0.12399251, -0.0387336
5]
std deviation: [0.14770581, 0.13934279, 0.17646013, 0.16417341, 0.14634815]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:15
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
```

```
    1      8 -3.894921417001886e-02 1.0e+00 1.86e-01  2e-01  2e-01 0:00.0
    2     16 -3.901318765821753e-02 1.3e+00 2.28e-01  2e-01  2e-01 0:00.0
    3     24 -4.014401647458225e-02 1.4e+00 2.29e-01  2e-01  2e-01 0:00.0
   60    480 -4.721027643853841e-02 1.7e+01 3.08e-02  4e-03  4e-02 0:00.8
termination on tolflatfitness=1 (Tue Jul 22 12:55:16 2025)
final/bestever f-value = -4.721028e-02 -5.204919e-02 after 481/344 evaluations
incumbent solution: [ 1.02419938, 0.05891981, -0.0216954, 0.03677866, -0.564986
9, ]
std deviation: [0.01833835, 0.00367617, 0.04135389, 0.02334756, 0.00545738]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:55:16
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.185342513960433e-03 1.0e+00 2.00e-01  2e-01  2e-01 0:00.0
    2     16 -1.316506160603871e-03 1.3e+00 1.74e-01  2e-01  2e-01 0:00.0
    3     24 -1.316506160603871e-03 1.4e+00 1.74e-01  2e-01  2e-01 0:00.0
    5     40 -1.316506160603871e-03 1.4e+00 1.49e-01  1e-01  2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:55:16 2025)
final/bestever f-value = -1.316506e-03 -2.185343e-03 after 41/2 evaluations
incumbent solution: [-0.28108575, 0.20001354, -0.09025238, -0.12212912, 0.16062
764]
std deviation: [0.1496401, 0.13567133, 0.11965983, 0.13042166, 0.15779182]
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000            92.09          -7.91           90.83
-9.17    100.0             0.0
0.0002   0.00005            86.79         -13.21           90.83
-9.17    100.0             0.0
0.0004   0.00010            83.69         -16.31           90.83
-9.17    100.0             0.0
0.0006   0.00030            78.79         -21.21           90.83
-9.17    100.0             0.0
```

In [ ]:
```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt
```

```
np.random.seed(42)
random_seed = 42

df = pd.read_csv("BTC_1min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

df_train['log_mid'] = np.log(df_train['midpoint'])
df_train['returns'] = df_train['log_mid'].diff().fillna(0)
df_cv['log_mid'] = np.log(df_cv['midpoint'])
df_cv['returns'] = df_cv['log_mid'].diff().fillna(0)
df_test['log_mid'] = np.log(df_test['midpoint'])
df_test['returns'] = df_test['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x
```

```
def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0, 0, 0, 0.005, 0.005], sigma0=0.2, options={'seed'
    return res[0][:3], res[0][3:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(0, 500), (500, 1000), (1000, 1500), (1500, 2000), (2000
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        if end > len(df_train):
            continue
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
```

```
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], window
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        raise ValueError("No positions generated.")

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
```

```
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

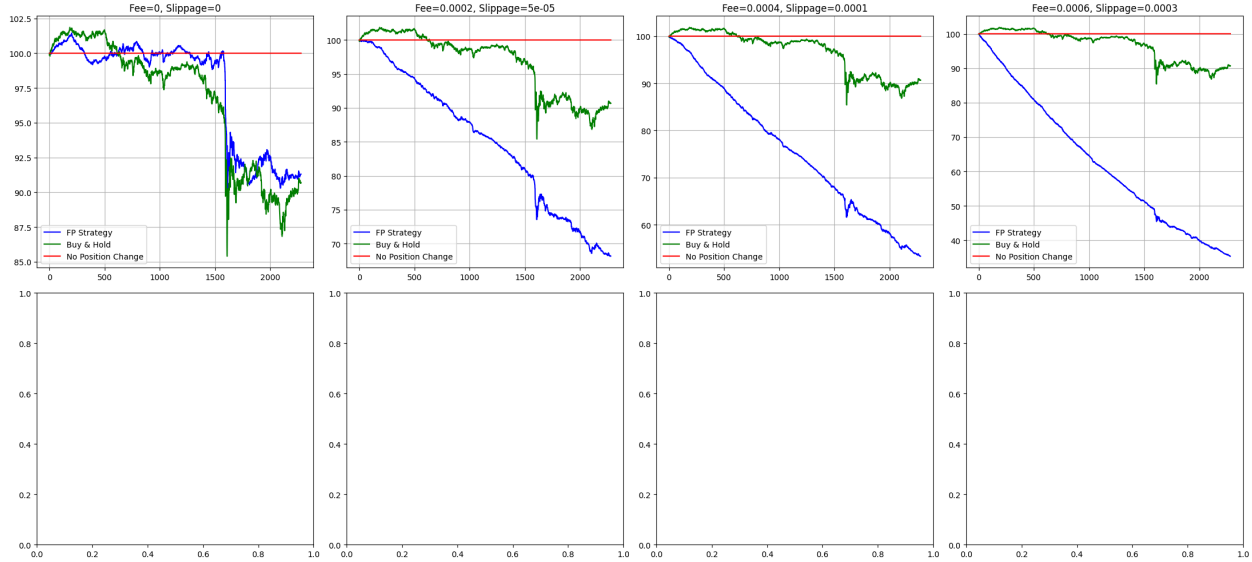
```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:24
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.441963413058090e-02 1.0e+00 2.01e-01  2e-01  2e-01 0:00.0
    2     16 -4.169058070135634e-02 1.4e+00 2.08e-01  2e-01  2e-01 0:00.1
    3     24 -3.684787103446929e-02 1.4e+00 2.07e-01  2e-01  2e-01 0:00.1
   75    600 -6.203562239241656e-02 5.5e+01 2.58e-01  9e-02  3e-01 0:02.0
termination on tolflatfitness=1 (Tue Jul 22 12:57:27 2025)
final/bestever f-value = -6.203562e-02 -6.423332e-02 after 601/164 evaluations
incumbent solution: [-4.20007249, 1.42307344, 0.96444844, 1.9999852, -3.605037
4, ]
std deviation: [0.29951799, 0.09296883, 0.12113019, 0.30080589, 0.13899404]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:27
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.545676823044296e-02 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -3.230812528117966e-02 1.1e+00 1.84e-01  2e-01  2e-01 0:00.1
    3     24 -3.288485130128826e-02 1.4e+00 1.80e-01  2e-01  2e-01 0:00.1
   81    648 -4.599338684974974e-02 1.7e+01 4.84e-03  5e-04  4e-03 0:02.2
termination on tolflatfitness=1 (Tue Jul 22 12:57:29 2025)
final/bestever f-value = -4.599339e-02 -4.689141e-02 after 649/83 evaluations
incumbent solution: [ 0.13633863, -0.3024853, 0.13560242, -0.85352317, -0.02103
352]
std deviation: [0.00050228, 0.00053364, 0.00100745, 0.00399062, 0.00193761]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:29
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.956362366069669e-03 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -7.956362366069669e-03 1.1e+00 1.91e-01  2e-01  2e-01 0:00.1
    3     24 -7.956362366069669e-03 1.4e+00 2.03e-01  2e-01  3e-01 0:00.1
termination on tolfun=1e-11 (Tue Jul 22 12:57:29 2025)
final/bestever f-value = -7.956362e-03 -7.956362e-03 after 25/1 evaluations
incumbent solution: [ 0.53668077, 0.04018252, -0.06557064, -0.00132741, 0.00838
982]
std deviation: [0.25032579, 0.18042018, 0.19751683, 0.191196, 0.17439106]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:29
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.116277670236371e-03 1.0e+00 1.81e-01  2e-01  2e-01 0:00.0
    2     16 -1.080744099690811e-02 1.3e+00 1.67e-01  2e-01  2e-01 0:00.1
    3     24 -8.116277670236371e-03 1.3e+00 1.56e-01  1e-01  2e-01 0:00.1
   31    248 -1.092560175495549e-02 5.5e+00 9.03e-02  4e-02  1e-01 0:01.5
termination on tolflatfitness=1 (Tue Jul 22 12:57:31 2025)
final/bestever f-value = -1.092560e-02 -1.177946e-02 after 249/167 evaluations
incumbent solution: [-0.00729908, 1.02639732, 0.58061605, -0.09969351, 0.701190
98]
std deviation: [0.03728471, 0.1038881, 0.09548655, 0.06355374, 0.05261837]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:31
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.727633945427527e-03 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     16 -9.727633945427527e-03 1.2e+00 1.84e-01  2e-01  2e-01 0:00.1
    3     24 -1.297538266637055e-02 1.3e+00 1.84e-01  2e-01  2e-01 0:00.1
   92    736 -3.906527979665952e-02 9.0e+00 3.06e-02  7e-03  2e-02 0:03.2
```

```
  100    800 -3.947599219343800e-02 8.0e+00 2.02e-02  4e-03  8e-03 0:03.4
  124    992 -3.996296762361951e-02 1.2e+01 6.79e-03  5e-04  3e-03 0:04.0
termination on tolflatfitness=1 (Tue Jul 22 12:57:35 2025)
final/bestever f-value = -3.996297e-02 -4.119132e-02 after 993/837 evaluations
incumbent solution: [ 0.3197678, -0.59490718, 0.80472471, -0.53620604, 1.729017
43]
std deviation: [0.00140106, 0.00049668, 0.00283898, 0.00165452, 0.0016572, ]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:37
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.395565427846193e-03 1.0e+00 2.11e-01  2e-01  2e-01 0:00.0
    2     16 -6.322634096482458e-03 1.3e+00 1.88e-01  2e-01  2e-01 0:00.1
    3     24 -6.031461741954912e-03 1.5e+00 1.68e-01  1e-01  2e-01 0:00.1
  100    800 -3.689278110072888e-02 3.6e+01 1.25e-02  6e-04  1e-02 0:02.8
  104    832 -3.689278110072888e-02 4.5e+01 1.07e-02  5e-04  1e-02 0:02.9
termination on tolflatfitness=1 (Tue Jul 22 12:57:40 2025)
final/bestever f-value = -3.689278e-02 -4.788259e-02 after 833/343 evaluations
incumbent solution: [ 1.33963485, 0.27948156, -0.27056806, -1.61193533, -1.0662
8795]
std deviation: [0.0129223, 0.00048322, 0.00340088, 0.01008565, 0.00159393]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:40
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.230422398393880e-02 1.0e+00 1.77e-01  2e-01  2e-01 0:00.0
    2     16 -9.196145941334194e-03 1.2e+00 1.97e-01  2e-01  2e-01 0:00.1
    3     24 -9.799989345629284e-03 1.5e+00 1.87e-01  2e-01  2e-01 0:00.1
   88    704 -3.667359608932563e-02 1.3e+01 4.31e-03  5e-04  2e-03 0:02.9
termination on tolflatfitness=1 (Tue Jul 22 12:57:43 2025)
final/bestever f-value = -3.667360e-02 -3.694553e-02 after 705/589 evaluations
incumbent solution: [ 0.14289178, -0.11102579, 0.43327161, -0.2820452, -0.10225
579]
std deviation: [0.00051729, 0.00066081, 0.00188688, 0.00108147, 0.00166271]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:43
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.706362366069669e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.0
    2     16 -1.712290418903353e-02 1.3e+00 1.92e-01  2e-01  2e-01 0:00.1
    3     24 -7.706362366069669e-03 1.6e+00 2.02e-01  2e-01  2e-01 0:00.1
   27    216 -1.712290418903353e-02 5.2e+00 7.23e-02  3e-02  7e-02 0:01.3
termination on tolflatfitness=1 (Tue Jul 22 12:57:45 2025)
final/bestever f-value = -1.712290e-02 -1.802493e-02 after 217/39 evaluations
incumbent solution: [ 0.43088572, 0.63694094, 0.26843666, 0.06709402, -0.572785
06]
std deviation: [0.03943244, 0.06453129, 0.06639951, 0.05978795, 0.03211024]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:45
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.691809770997565e-03 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -8.371878019403339e-03 1.3e+00 1.61e-01  2e-01  2e-01 0:00.1
    3     24 -7.691809770997565e-03 1.6e+00 1.61e-01  2e-01  2e-01 0:00.1
   13    104 -7.691809770997565e-03 2.6e+00 1.24e-01  7e-02  1e-01 0:00.6
termination on tolflatfitness=1 (Tue Jul 22 12:57:45 2025)
final/bestever f-value = -7.691810e-03 -1.000113e-02 after 105/73 evaluations
incumbent solution: [ 0.14186528, 0.02769144, -0.1346926, -0.14339952, -0.07232
```

652]
std deviation: [0.0666515, 0.11135855, 0.1269637, 0.12473736, 0.13246674]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:45
2025)
Iterat #Fevals   function value axis ratio sigma min&max std t[m:s]
    1   8 -9.477633945427527e-03 1.0e+00 1.71e-01 2e-01 2e-01 0:00.0
    2   16 -9.749937848507556e-03 1.3e+00 1.52e-01 1e-01 2e-01 0:00.1
    3   24 -9.477633945427527e-03 1.3e+00 1.43e-01 1e-01 1e-01 0:00.1
    4   32 -9.477633945427527e-03 1.5e+00 1.43e-01 1e-01 1e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:57:46 2025)
final/bestever f-value = -9.477634e-03 -9.749938e-03 after 33/12 evaluations
incumbent solution: [0.19106127, -0.08355099, -0.10074718, -0.0757329, 0.10204
028]
std deviation: [0.12547872, 0.13432473, 0.14567355, 0.13280575, 0.12329136]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:48
2025)
Iterat #Fevals   function value axis ratio sigma min&max std t[m:s]
    1   8 -3.895565427846193e-03 1.0e+00 2.00e-01 2e-01 2e-01 0:00.0
    2   16 -2.399134588533581e-03 1.4e+00 1.90e-01 2e-01 2e-01 0:00.1
    3   24 -4.137393292236835e-03 1.4e+00 1.84e-01 2e-01 2e-01 0:00.1
   75   600 -1.738441770963568e-02 2.6e+01 4.59e-03 8e-04 3e-03 0:02.0
termination on tolflatfitness=1 (Tue Jul 22 12:57:50 2025)
final/bestever f-value = -1.738442e-02 -1.738442e-02 after 601/450 evaluations
incumbent solution: [0.10667093, -0.23298351, -0.29407823, -0.29902053, 0.1093
1384]
std deviation: [0.0007596, 0.00139351, 0.00139619, 0.00316542, 0.00286711]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:50
2025)
Iterat #Fevals   function value axis ratio sigma min&max std t[m:s]
    1   8 -9.054223983938795e-03 1.0e+00 1.71e-01 2e-01 2e-01 0:00.0
    2   16 -8.946145941334194e-03 1.3e+00 1.69e-01 2e-01 2e-01 0:00.0
    3   24 -1.128348114578459e-02 1.4e+00 1.69e-01 1e-01 2e-01 0:00.1
    6   48 -8.946145941334194e-03 1.7e+00 1.63e-01 1e-01 2e-01 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 12:57:50 2025)
final/bestever f-value = -8.946146e-03 -1.128348e-02 after 49/17 evaluations
incumbent solution: [0.18613751, 0.26906981, -0.01050676, 0.17981718, -0.13540
598]
std deviation: [0.12683684, 0.15304764, 0.19212389, 0.15975073, 0.15153581]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:50
2025)
Iterat #Fevals   function value axis ratio sigma min&max std t[m:s]
    1   8 -7.456362366069669e-03 1.0e+00 1.71e-01 2e-01 2e-01 0:00.0
    2   16 -1.637290418903353e-02 1.3e+00 1.92e-01 2e-01 2e-01 0:00.1
    3   24 -7.456362366069669e-03 1.6e+00 2.02e-01 2e-01 2e-01 0:00.1
   28   224 -1.637290418903353e-02 6.2e+00 1.14e-01 5e-02 1e-01 0:00.8
termination on tolflatfitness=1 (Tue Jul 22 12:57:51 2025)
final/bestever f-value = -1.637290e-02 -1.637290e-02 after 225/15 evaluations
incumbent solution: [0.78984283, 0.61672514, 0.3367195, -0.05006749, -0.543754
01]
std deviation: [0.09013844, 0.1119309, 0.06925293, 0.14990792, 0.05036692]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:51
2025)
Iterat #Fevals   function value axis ratio sigma min&max std t[m:s]
    1   8 -7.441809770997565e-03 1.0e+00 1.74e-01 2e-01 2e-01 0:00.0

```
    2     16 -7.621878019403339e-03 1.3e+00 1.61e-01  2e-01  2e-01 0:00.1
    3     24 -7.441809770997565e-03 1.6e+00 1.61e-01  2e-01  2e-01 0:00.1
    7     56 -7.441809770997565e-03 2.0e+00 2.28e-01  2e-01  3e-01 0:00.2
termination on tolflatfitness=1 (Tue Jul 22 12:57:51 2025)
final/bestever f-value = -7.441810e-03 -7.621878e-03 after 57/15 evaluations
incumbent solution: [ 0.28980417, 0.30334007, -0.00799526, -0.84907397, 0.16957
228]
std deviation: [0.21887444, 0.19410581, 0.22074146, 0.3035431, 0.21018946]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:51
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.227633945427527e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.0
    2     16 -9.227633945427527e-03 1.3e+00 1.68e-01  2e-01  2e-01 0:00.1
    3     24 -9.227633945427527e-03 1.4e+00 1.71e-01  2e-01  2e-01 0:00.1
    5     40 -9.227633945427527e-03 1.8e+00 1.56e-01  1e-01  2e-01 0:00.1
termination on tolfun=1e-11 (Tue Jul 22 12:57:51 2025)
final/bestever f-value = -9.227634e-03 -9.227634e-03 after 41/1 evaluations
incumbent solution: [ 0.29341182, 0.01571146, 0.38014266, 0.19582524, -0.072107
86]
std deviation: [0.13384939, 0.14219705, 0.16403192, 0.15325955, 0.13533883]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:54
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -3.095565427846197e-03 1.0e+00 2.00e-01  2e-01  2e-01 0:00.0
    2     16 -6.969412066152144e-03 1.4e+00 1.82e-01  2e-01  2e-01 0:00.1
    3     24 -1.662673668685263e-05 1.5e+00 1.70e-01  1e-01  2e-01 0:00.1
   11     88 4.082416332135009e-04 2.9e+00 1.47e-01  1e-01  2e-01 0:00.3
termination on tolflatfitness=1 (Tue Jul 22 12:57:54 2025)
final/bestever f-value = 4.082416e-04 -6.969412e-03 after 89/13 evaluations
incumbent solution: [ 0.51647869, -0.05843548, -0.44217309, -0.45356807, 0.0116
2647]
std deviation: [0.16580037, 0.10472692, 0.17823626, 0.14592363, 0.10169303]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:54
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.546145941334194e-03 1.0e+00 1.79e-01  2e-01  2e-01 0:00.0
    2     16 -8.546145941334194e-03 1.3e+00 1.86e-01  2e-01  2e-01 0:00.0
    3     24 -8.546145941334194e-03 1.5e+00 1.96e-01  2e-01  2e-01 0:00.1
    8     64 -8.546145941334194e-03 1.9e+00 2.08e-01  2e-01  2e-01 0:00.2
termination on tolfun=1e-11 (Tue Jul 22 12:57:54 2025)
final/bestever f-value = -8.546146e-03 -8.546146e-03 after 65/5 evaluations
incumbent solution: [ 0.70664199, -0.15461356, 0.09594236, -0.24590495, 0.06440
784]
std deviation: [0.20599741, 0.17665923, 0.24492267, 0.18628693, 0.16750415]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:54
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.056362366069669e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.0
    2     16 -1.517290418903353e-02 1.3e+00 2.03e-01  2e-01  2e-01 0:00.1
    3     24 -1.517290418903353e-02 1.8e+00 2.16e-01  2e-01  2e-01 0:00.1
   26    208 -1.517290418903353e-02 1.0e+01 1.36e-01  6e-02  2e-01 0:00.7
termination on tolflatfitness=1 (Tue Jul 22 12:57:55 2025)
final/bestever f-value = -1.517290e-02 -1.517290e-02 after 209/15 evaluations
incumbent solution: [ 1.05188649, 0.77447768, -0.62333343, -0.18947736, -0.5960
```

```
6083]
std deviation: [0.10893681, 0.15274283, 0.15003477, 0.16027303, 0.05843498]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:55
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.041809770997566e-03 1.0e+00 1.74e-01  2e-01  2e-01 0:00.0
    2     16 -7.041809770997566e-03 1.3e+00 1.73e-01  2e-01  2e-01 0:00.1
    3     24 -7.041809770997566e-03 1.5e+00 1.68e-01  1e-01  2e-01 0:00.1
    5     40 -7.041809770997566e-03 1.6e+00 1.74e-01  1e-01  2e-01 0:00.1
termination on tolfun=1e-11 (Tue Jul 22 12:57:55 2025)
final/bestever f-value = -7.041810e-03 -7.041810e-03 after 41/1 evaluations
incumbent solution: [0.27704897, 0.09294729, 0.28373077, 0.17614967, 0.1913378
8]
std deviation: [0.17025798, 0.16355975, 0.16587028, 0.14384269, 0.17260769]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:57:55
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.827633945427528e-03 1.0e+00 1.71e-01  2e-01  2e-01 0:00.1
    2     16 -8.827633945427528e-03 1.3e+00 1.68e-01  2e-01  2e-01 0:00.1
    3     24 -8.827633945427528e-03 1.4e+00 1.71e-01  2e-01  2e-01 0:00.1
    5     40 -8.827633945427528e-03 1.7e+00 1.68e-01  1e-01  2e-01 0:00.2
termination on tolfun=1e-11 (Tue Jul 22 12:57:56 2025)
final/bestever f-value = -8.827634e-03 -8.827634e-03 after 41/1 evaluations
incumbent solution: [ 0.37340367, 0.14749911, 0.03200432, 0.11451962, -0.215587
07]
std deviation: [0.15924328, 0.14447809, 0.18384392, 0.15205373, 0.15166416]
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000            91.30          -8.70           90.67
-9.33    100.0             0.0
0.0002   0.00005            68.15         -31.85           90.67
-9.33    100.0             0.0
0.0004   0.00010            53.32         -46.68           90.67
-9.33    100.0             0.0
0.0006   0.00030            35.38         -64.62           90.67
-9.33    100.0             0.0
```

In [ ]:

```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("BTC_1sec.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_co
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

df_train['log_mid'] = np.log(df_train['midpoint'])
df_train['returns'] = df_train['log_mid'].diff().fillna(0)
df_cv['log_mid'] = np.log(df_cv['midpoint'])
df_cv['returns'] = df_cv['log_mid'].diff().fillna(0)
df_test['log_mid'] = np.log(df_test['midpoint'])
df_test['returns'] = df_test['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
```

```
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0, 0, 0, 0.005, 0.005], sigma0=0.2, options={'seed'
    return res[0][:3], res[0][3:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(0, 5000), (5000, 10000), (10000, 15000), (15000, 20000)
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        if end > len(df_train):
            continue
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []
```

```
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], windo
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:en
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        raise ValueError("No positions generated.")

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
```

```
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:30
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.526229089390085e-02 1.0e+00 1.75e-01  2e-01  2e-01 0:00.2
/tmp/ipython-input-4-1455862918.py:55: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-4-1455862918.py:55: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    2     16 -1.641531617207015e-02 1.2e+00 1.85e-01  2e-01  2e-01 0:00.4
    3     24 -1.824139523899682e-02 1.3e+00 1.96e-01  2e-01  2e-01 0:00.7
/tmp/ipython-input-4-1455862918.py:53: RuntimeWarning: overflow encountered in
scalar multiply
  mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
```

```
   16    128 -6.310840890263769e-02 3.0e+00 3.43e-01  2e-01  4e-01 0:03.7
   27    216 -7.523560598786716e-02 4.2e+00 1.49e-01  7e-02  2e-01 0:07.8
   49    392 -7.995890086868584e-02 8.9e+00 3.11e-02  9e-03  4e-02 0:12.9
   73    584 -8.118571258484053e-02 1.7e+01 6.57e-03  9e-04  7e-03 0:19.0
   92    736 -8.126575128741287e-02 1.1e+01 2.36e-03  2e-04  1e-03 0:24.0
termination on tolflatfitness=1 (Tue Jul 22 12:58:54 2025)
final/bestever f-value = -8.126575e-02 -8.126575e-02 after 737/628 evaluations
incumbent solution: [ 0.58209795, -1.66012098, 1.98540641, 0.10691739, 0.031202
31]
std deviation: [0.00057081, 0.00020772, 0.00148053, 0.00027078, 0.00038028]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:58:54
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.667937332978454e-02 1.0e+00 1.93e-01  2e-01  2e-01 0:00.2
    2     16 -3.127055706997339e-02 1.3e+00 2.33e-01  2e-01  2e-01 0:00.5
    3     24 -3.785275723090642e-02 1.7e+00 2.46e-01  2e-01  3e-01 0:00.7
   16    128 -5.759990115121028e-02 2.6e+00 2.25e-01  2e-01  2e-01 0:03.7
   31    248 -6.213116284022924e-02 5.1e+00 1.27e-01  6e-02  2e-01 0:07.8
   49    392 -6.321692924231925e-02 1.8e+01 6.49e-02  2e-02  1e-01 0:12.9
   76    608 -6.356150633701496e-02 3.7e+01 1.46e-02  3e-03  3e-02 0:19.0
  100    800 -6.358926401356157e-02 5.1e+01 5.61e-03  6e-04  1e-02 0:25.7
  103    824 -6.358926401356157e-02 5.6e+01 3.69e-03  4e-04  8e-03 0:26.4
termination on tolflatfitness=1 (Tue Jul 22 12:59:21 2025)
final/bestever f-value = -6.358926e-02 -6.360430e-02 after 825/667 evaluations
incumbent solution: [ 0.1923839, -1.41025431, 2.00238695, -0.00908117, 0.033663
36]
std deviation: [0.00060377, 0.00035744, 0.00759554, 0.00037509, 0.00059436]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:59:21
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.555387481442750e-02 1.0e+00 1.88e-01  2e-01  2e-01 0:00.2
    2     16 -4.781351261230959e-02 1.2e+00 2.44e-01  2e-01  3e-01 0:00.4
    3     24 -4.435216349091853e-02 1.4e+00 2.98e-01  3e-01  3e-01 0:00.6
    9     72 -5.273405777725237e-02 2.0e+00 3.45e-01  2e-01  4e-01 0:03.7
   23    184 -6.244732650030826e-02 3.8e+00 2.01e-01  1e-01  3e-01 0:08.1
   42    336 -6.270443969559025e-02 9.5e+00 6.67e-02  2e-02  9e-02 0:13.1
   69    552 -6.306750196968736e-02 2.5e+01 1.77e-02  4e-03  2e-02 0:19.1
   95    760 -6.313617624933698e-02 6.1e+01 2.99e-02  5e-03  5e-02 0:26.2
  100    800 -6.324332147680423e-02 6.7e+01 2.03e-02  3e-03  3e-02 0:27.3
  127   1016 -6.332946534643291e-02 6.0e+01 4.26e-03  3e-04  5e-03 0:34.0
termination on tolflatfitness=1 (Tue Jul 22 12:59:55 2025)
final/bestever f-value = -6.332947e-02 -6.332947e-02 after 1017/897 evaluations
incumbent solution: [ 0.2157967, -1.19115509, 2.48227998, -0.11236989, 0.467023
5, ]
std deviation: [0.0004817, 0.00041042, 0.00468099, 0.00054592, 0.00031433]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 12:59:55
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -2.002445396868602e-02 1.0e+00 2.03e-01  2e-01  2e-01 0:00.4
    2     16 -6.505408191543438e-03 1.4e+00 2.44e-01  2e-01  3e-01 0:00.8
    3     24 -3.137814090194446e-02 1.9e+00 2.61e-01  2e-01  3e-01 0:01.2
   15    120 -8.008032134900844e-02 2.6e+00 2.87e-01  2e-01  3e-01 0:04.2
   33    264 -8.739347572397627e-02 5.1e+00 1.41e-01  5e-02  1e-01 0:08.3
   53    424 -9.283088490379932e-02 6.9e+00 3.50e-02  7e-03  3e-02 0:13.5
```

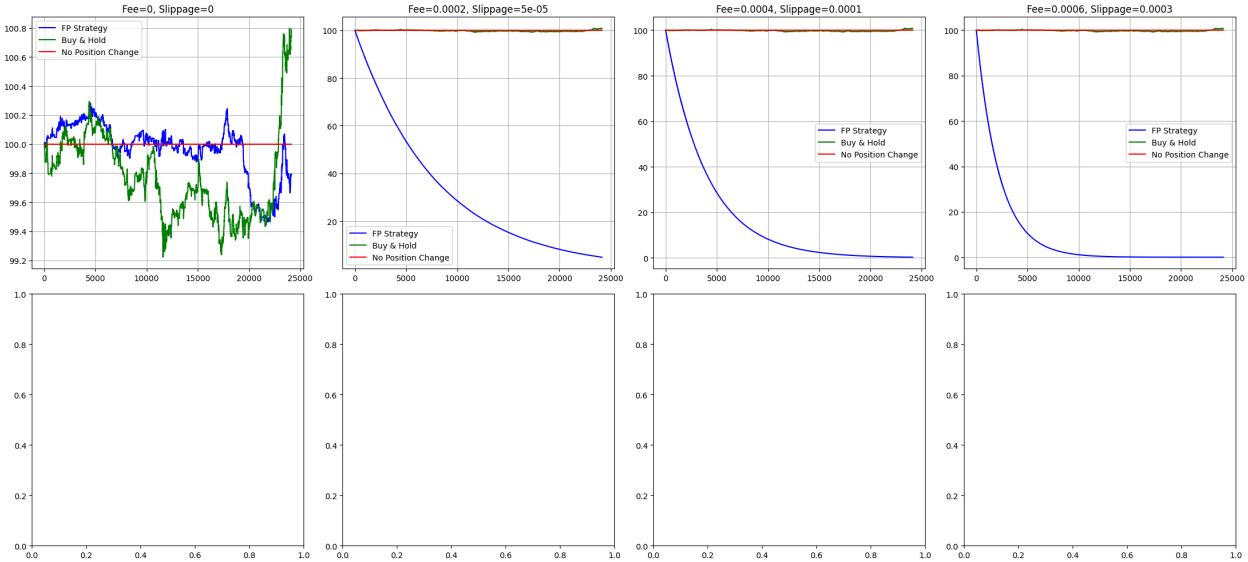
```
   77    616 -9.324635931084302e-02 8.6e+00 6.98e-03  9e-04  4e-03 0:19.7
  100    800 -9.326939962776670e-02 1.4e+01 2.80e-03  2e-04  2e-03 0:25.4
  102    816 -9.326939962776670e-02 1.5e+01 2.07e-03  2e-04  1e-03 0:26.2
termination on tolflatfitness=1 (Tue Jul 22 13:00:22 2025)
final/bestever f-value = -9.326940e-02 -9.326940e-02 after 817/587 evaluations
incumbent solution: [ 0.13182977, -1.42789695, 2.28557088, -0.41013601, 0.64751
858]
std deviation: [0.00016474, 0.00022972, 0.00114404, 0.0004396, 0.00036091]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:00:22
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.842230926198951e-02 1.0e+00 1.82e-01  2e-01  2e-01 0:00.4
    2     16 -1.559106961495793e-02 1.1e+00 1.70e-01  1e-01  2e-01 0:00.8
    3     24 -1.266975106671353e-02 1.3e+00 1.57e-01  1e-01  2e-01 0:01.2
   16    128 -5.108418222495992e-02 3.2e+00 2.30e-01  1e-01  3e-01 0:04.4
   33    264 -5.946717653598377e-02 4.2e+00 7.27e-02  2e-02  7e-02 0:08.4
   52    416 -6.077754714593198e-02 8.6e+00 2.29e-02  3e-03  2e-02 0:13.7
   74    592 -6.118601869454565e-02 3.2e+01 7.02e-03  1e-03  7e-03 0:19.9
  100    800 -6.121486722554792e-02 3.4e+01 3.18e-03  3e-04  3e-03 0:26.9
  122    976 -6.121486722554792e-02 3.4e+01 2.13e-03  8e-05  1e-03 0:32.5
termination on tolflatfitness=1 (Tue Jul 22 13:00:54 2025)
final/bestever f-value = -6.121487e-02 -6.121487e-02 after 977/638 evaluations
incumbent solution: [-0.1671144, -1.04510834, 1.30385492, -0.00703088, 0.021344
49]
std deviation: [1.85672109e-04, 7.61430553e-05, 1.39744776e-03, 1.86581301e-04,
4.59837626e-04]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:23
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -8.183271237626272e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.2
    2     16 -8.183271237626272e-03 1.4e+00 1.87e-01  2e-01  2e-01 0:00.4
    3     24 -8.183271237626272e-03 1.4e+00 1.78e-01  2e-01  2e-01 0:00.7
    8     64 -8.183271237626272e-03 2.8e+00 2.13e-01  2e-01  2e-01 0:01.8
termination on tolflatfitness=1 (Tue Jul 22 13:01:25 2025)
final/bestever f-value = -8.183271e-03 -8.183271e-03 after 65/5 evaluations
incumbent solution: [ 0.50892885, -0.44939857, 0.10006429, 0.08723223, -0.09569
162]
std deviation: [0.20490649, 0.21075391, 0.16613889, 0.24480949, 0.21630153]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:25
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.636951165988918e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.2
    2     16 -5.636951165988918e-03 1.4e+00 1.86e-01  2e-01  2e-01 0:00.4
    3     24 -5.636951165988918e-03 1.5e+00 1.72e-01  1e-01  2e-01 0:00.7
   10     80 -5.636951165988918e-03 2.9e+00 1.49e-01  9e-02  2e-01 0:02.6
termination on tolfunhist=1e-12 (Tue Jul 22 13:01:27 2025)
final/bestever f-value = -5.636951e-03 -5.636951e-03 after 81/5 evaluations
incumbent solution: [0.30383884, 0.05261629, 0.08982376, 0.24788079, 0.1522066
7]
std deviation: [0.14750058, 0.09306576, 0.10079402, 0.22730337, 0.10734575]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:28
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -6.655639376399847e-03 1.0e+00 2.12e-01  2e-01  2e-01 0:00.4
```

```
    2     16 -6.730945384041120e-03 1.5e+00 1.98e-01  2e-01  2e-01 0:00.8
    3     24 -6.905287472510707e-03 1.5e+00 1.90e-01  2e-01  2e-01 0:01.2
   15    120 -6.905287472510707e-03 1.9e+00 2.10e-01  1e-01  2e-01 0:04.5
   16    128 -6.905287472510707e-03 1.9e+00 2.20e-01  1e-01  3e-01 0:04.7
termination on tolflatfitness=1 (Tue Jul 22 13:01:32 2025)
final/bestever f-value = -6.905287e-03 -6.905287e-03 after 129/17 evaluations
incumbent solution: [-0.87224076, -0.75212339, -0.03288368, -0.15283645, -0.027
43716]
std deviation: [0.20146315, 0.25547575, 0.19950111, 0.15007175, 0.13437741]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:32
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.213083312735591e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.2
    2     16 -3.981821912843742e-03 1.4e+00 2.36e-01  2e-01  2e-01 0:00.4
    3     24 -4.003168866829554e-03 1.5e+00 2.69e-01  2e-01  3e-01 0:00.7
   17    136 -3.947242408872398e-03 2.7e+00 1.15e-01  6e-02  1e-01 0:03.9
   34    272 -4.300214718057915e-03 7.3e+00 3.49e-02  6e-03  3e-02 0:07.9
   51    408 -4.391744912503881e-03 2.2e+01 2.25e-02  1e-03  2e-02 0:13.0
   62    496 -4.391744912503881e-03 5.3e+01 1.06e-02  3e-04  1e-02 0:15.4
termination on tolflatfitness=1 (Tue Jul 22 13:01:48 2025)
final/bestever f-value = -4.391745e-03 -4.391745e-03 after 497/103 evaluations
incumbent solution: [ 0.09558938, 0.54783636, 0.04729134, -1.05521361, -0.06332
685]
std deviation: [0.00908881, 0.00027094, 0.00464428, 0.00954817, 0.00572071]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:01:48
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.115278122318961e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.2
    2     16 -1.129387992570707e-02 1.5e+00 1.88e-01  2e-01  2e-01 0:00.4
    3     24 -1.151662033651493e-02 1.4e+00 1.79e-01  2e-01  2e-01 0:00.7
   17    136 -1.300206428404141e-02 2.9e+00 7.08e-02  4e-02  7e-02 0:03.7
   31    248 -1.837488754944769e-02 4.9e+00 4.00e-02  1e-02  4e-02 0:08.4
   54    432 -1.939225060681707e-02 1.7e+01 1.36e-02  1e-03  2e-02 0:13.6
   79    632 -2.005474457347858e-02 5.1e+01 1.80e-02  5e-04  2e-02 0:19.8
  100    800 -1.939225060681707e-02 7.7e+01 8.22e-03  2e-04  1e-02 0:25.2
  102    816 -1.939225060681707e-02 9.4e+01 6.22e-03  1e-04  7e-03 0:25.7
termination on tolflatfitness=1 (Tue Jul 22 13:02:14 2025)
final/bestever f-value = -1.939225e-02 -2.026684e-02 after 817/342 evaluations
incumbent solution: [-0.59904152, 0.02750091, -0.03814549, -0.46483592, 0.35103
885]
std deviation: [0.00737424, 0.00013229, 0.00336361, 0.00372613, 0.00035589]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:41
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -7.933271237626272e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.2
    2     16 -7.933271237626272e-03 1.4e+00 1.87e-01  2e-01  2e-01 0:00.5
    3     24 -7.933271237626272e-03 1.4e+00 1.78e-01  2e-01  2e-01 0:00.7
    8     64 -7.933271237626272e-03 2.5e+00 1.57e-01  1e-01  2e-01 0:01.9
termination on tolfun=1e-11 (Tue Jul 22 13:02:43 2025)
final/bestever f-value = -7.933271e-03 -7.933271e-03 after 65/5 evaluations
incumbent solution: [ 0.41568251, -0.01606387, 0.02562045, -0.22570055, -0.0655
5129]
std deviation: [0.15845521, 0.12621165, 0.11023669, 0.18397556, 0.16015497]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:43
```

```
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.386951165988918e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.2
    2     16 -5.386951165988918e-03 1.4e+00 1.86e-01  2e-01  2e-01 0:00.4
    3     24 -5.386951165988918e-03 1.5e+00 1.72e-01  1e-01  2e-01 0:00.7
   10     80 -5.386951165988918e-03 2.6e+00 1.33e-01  8e-02  2e-01 0:02.3
termination on tolfunhist=1e-12 (Tue Jul 22 13:02:46 2025)
final/bestever f-value = -5.386951e-03 -5.386951e-03 after 81/5 evaluations
incumbent solution: [ 0.41084579, 0.08976846, 0.16224174, -0.10109284, 0.236994
07]
std deviation: [0.13566125, 0.07751884, 0.09652535, 0.17880693, 0.10557102]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:46
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -6.155639376399847e-03 1.0e+00 2.12e-01  2e-01  2e-01 0:00.3
    2     16 -6.230945384041121e-03 1.5e+00 1.98e-01  2e-01  2e-01 0:00.6
    3     24 -6.655287472510707e-03 1.5e+00 1.97e-01  2e-01  2e-01 0:01.0
    9     72 -6.655287472510707e-03 2.6e+00 1.93e-01  1e-01  3e-01 0:03.3
termination on tolflatfitness=1 (Tue Jul 22 13:02:49 2025)
final/bestever f-value = -6.655287e-03 -6.655287e-03 after 73/17 evaluations
incumbent solution: [-0.95322671, -0.2375552, -0.01950084, -0.22538552, 0.04064
482]
std deviation: [0.27346415, 0.15582777, 0.22695519, 0.18770286, 0.12083337]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:02:49
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -9.630833127355913e-04 1.0e+00 1.92e-01  2e-01  2e-01 0:00.3
    2     16 -3.481821912843742e-03 1.4e+00 2.14e-01  2e-01  2e-01 0:00.5
    3     24 -2.483475297240801e-03 1.7e+00 2.34e-01  2e-01  3e-01 0:00.7
   16    128 -4.681484056715647e-03 3.2e+00 9.93e-02  5e-02  1e-01 0:03.7
   34    272 -5.257529615121271e-03 1.1e+01 4.93e-02  1e-02  7e-02 0:07.9
   50    400 -5.163306330170926e-03 2.4e+01 5.22e-02  1e-02  8e-02 0:13.0
   75    600 -5.209978786579029e-03 3.7e+01 4.35e-02  5e-03  6e-02 0:19.0
   98    784 -5.343731741020875e-03 9.4e+01 1.30e-02  7e-04  1e-02 0:25.4
termination on tolflatfitness=1 (Tue Jul 22 13:03:15 2025)
final/bestever f-value = -5.343732e-03 -8.720793e-03 after 785/284 evaluations
incumbent solution: [-0.10126422, 0.21247312, 0.1975043, -0.66124898, -0.143606
19]
std deviation: [0.00268999, 0.00073239, 0.01373753, 0.00841329, 0.00727812]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:03:15
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -1.090278122318961e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.4
    2     16 -1.090278122318961e-02 1.5e+00 2.10e-01  2e-01  2e-01 0:00.7
    3     24 -1.090278122318961e-02 1.4e+00 1.98e-01  2e-01  2e-01 0:00.9
   10     80 -1.090278122318961e-02 1.7e+00 1.33e-01  1e-01  1e-01 0:02.4
termination on tolfunhist=1e-12 (Tue Jul 22 13:03:17 2025)
final/bestever f-value = -1.090278e-02 -1.090278e-02 after 81/8 evaluations
incumbent solution: [-0.23671151, 0.12706257, -0.35140306, -0.11938256, 0.09422
676]
std deviation: [0.12353808, 0.09598735, 0.11027349, 0.10867325, 0.12359789]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:03:45
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
```

```
    1      8 -7.533271237626272e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.2
    2     16 -7.533271237626272e-03 1.4e+00 1.87e-01  2e-01  2e-01 0:00.4
    3     24 -7.533271237626272e-03 1.4e+00 1.78e-01  2e-01  2e-01 0:00.7
    8     64 -7.533271237626272e-03 2.5e+00 1.57e-01  1e-01  2e-01 0:01.8
termination on tolfun=1e-11 (Tue Jul 22 13:03:47 2025)
final/bestever f-value = -7.533271e-03 -7.533271e-03 after 65/5 evaluations
incumbent solution: [ 0.41568251, -0.01606387, 0.02562045, -0.22570055, -0.0655
5129]
std deviation: [0.15845521, 0.12621165, 0.11023669, 0.18397556, 0.16015497]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:03:47
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -4.986951165988919e-03 1.0e+00 1.92e-01  2e-01  2e-01 0:00.2
    2     16 -4.986951165988919e-03 1.4e+00 1.86e-01  2e-01  2e-01 0:00.4
    3     24 -4.986951165988919e-03 1.5e+00 1.72e-01  1e-01  2e-01 0:00.7
   10     80 -4.986951165988919e-03 2.4e+00 1.05e-01  6e-02  1e-01 0:02.2
termination on tolfun=1e-11 (Tue Jul 22 13:03:49 2025)
termination on tolfunhist=1e-12 (Tue Jul 22 13:03:49 2025)
termination on tolflatfitness=1 (Tue Jul 22 13:03:49 2025)
final/bestever f-value = -4.986951e-03 -4.986951e-03 after 81/5 evaluations
incumbent solution: [ 0.11385735, 0.07430859, 0.13496384, -0.16922882, 0.035492
17]
std deviation: [0.10873923, 0.05950163, 0.07007542, 0.13183049, 0.07834297]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:03:49
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.355639376399848e-03 1.0e+00 2.12e-01  2e-01  2e-01 0:00.2
    2     16 -5.430945384041121e-03 1.5e+00 1.98e-01  2e-01  2e-01 0:00.4
    3     24 -6.255287472510708e-03 1.5e+00 1.97e-01  2e-01  2e-01 0:00.7
    9     72 -6.255287472510708e-03 2.6e+00 1.93e-01  1e-01  3e-01 0:02.0
termination on tolflatfitness=1 (Tue Jul 22 13:03:51 2025)
final/bestever f-value = -6.255287e-03 -6.255287e-03 after 73/17 evaluations
incumbent solution: [-0.95322671, -0.2375552, -0.01950084, -0.22538552, 0.04064
482]
std deviation: [0.27346415, 0.15582777, 0.22695519, 0.18770286, 0.12083337]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:03:52
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1      8 -5.630833127355914e-04 1.0e+00 1.89e-01  2e-01  2e-01 0:00.3
    2     16 -1.666168424937521e-03 1.3e+00 1.96e-01  2e-01  2e-01 0:00.7
    3     24 -2.576668282146667e-03 1.4e+00 1.93e-01  2e-01  2e-01 0:01.0
   13    104 -5.630833127355914e-04 2.6e+00 1.89e-01  1e-01  2e-01 0:04.2
   32    256 -4.018743370785029e-03 6.3e+00 7.33e-02  2e-02  9e-02 0:08.4
   54    432 -4.543731741020875e-03 3.1e+01 3.69e-02  2e-03  5e-02 0:13.5
   75    600 -4.543731741020875e-03 2.8e+02 2.91e-02  4e-04  6e-02 0:19.7
   83    664 -4.543731741020875e-03 4.4e+02 3.91e-02  5e-04  8e-02 0:21.6
termination on tolflatfitness=1 (Tue Jul 22 13:04:13 2025)
final/bestever f-value = -4.543732e-03 -4.543732e-03 after 665/119 evaluations
incumbent solution: [ 0.08499159, 0.20654124, -0.4596759, -0.15668804, -0.06596
824]
std deviation: [0.0627836, 0.00050847, 0.03390653, 0.07734724, 0.01744258]
(4_w,8)-aCMA-ES (mu_w=2.6,w_1=52%) in dimension 5 (seed=42, Tue Jul 22 13:04:13
2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
```

```
    1      8 -1.050278122318961e-02 1.0e+00 2.12e-01  2e-01  2e-01 0:00.2
    2     16 -1.050278122318961e-02 1.5e+00 2.10e-01  2e-01  2e-01 0:00.5
    3     24 -1.050278122318961e-02 1.4e+00 1.98e-01  2e-01  2e-01 0:00.7
    8     64 -1.050278122318961e-02 1.8e+00 1.27e-01  8e-02  1e-01 0:01.9
termination on tolfun=1e-11 (Tue Jul 22 13:04:15 2025)
final/bestever f-value = -1.050278e-02 -1.050278e-02 after 65/8 evaluations
incumbent solution: [-0.43578784, 0.05531627, -0.2244616, -0.11739515, 0.128639
35]
std deviation: [0.14934851, 0.08203384, 0.10552345, 0.11588823, 0.11007877]
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000            99.79          -0.21          100.79
0.79    100.0             0.0
0.0002   0.00005             4.91         -95.09          100.79
0.79    100.0             0.0
0.0004   0.00010             0.24         -99.76          100.79
0.79    100.0             0.0
0.0006   0.00030             0.00        -100.00          100.79
0.79    100.0             0.0
```

In [ ]:

```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt
from numba import njit
from sklearn.preprocessing import RobustScaler
from scipy.stats import norm
from statsmodels.tsa.statespace.tools import constrain_stationary_univariate

np.random.seed(42)
random_seed = 42

df = pd.read_csv("BTC_5min.csv", parse_dates=['system_time'], index_col='syste
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}'] * (1 + 0.
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}'] * (1 + 0.
```

```
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = np.log1p(df[bid_cols + ask_cols].sum(axis=1))
df['queue_slope'] = (df['bids_notional_0'] - df['bids_notional_5']) / (df['bic

scaler = RobustScaler()
features = ['obi', 'dobi', 'depth', 'queue_slope']
df[features] = scaler.fit_transform(df[features])

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

df_train['log_mid'] = np.log(df_train['midpoint'])
df_train['returns'] = df_train['log_mid'].diff().fillna(0)
df_cv['log_mid'] = np.log(df_cv['midpoint'])
df_cv['returns'] = df_cv['log_mid'].diff().fillna(0)
df_test['log_mid'] = np.log(df_test['midpoint'])
df_test['returns'] = df_test['log_mid'].diff().fillna(0)

@njit
def trading_strategy(signal, threshold, volatility):
    positions = np.zeros(len(signal))
    for i in range(1, len(signal)):
        z_score = signal[i] / (volatility[i] + 1e-8)
        if z_score > threshold:
            positions[i] = min(positions[i-1] + 0.1, 1)
        elif z_score < -threshold:
            positions[i] = max(positions[i-1] - 0.1, -1)
        else:
            positions[i] = positions[i-1] * 0.95
    # Manual diff with prepend=0
    trades = np.zeros(len(positions))
    trades[0] = positions[0]
    for i in range(1, len(positions)):
        trades[i] = positions[i] - positions[i-1]
    return positions, trades

@njit
def apply_trading_costs(positions, trades, returns, fee, slip, volatility):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip * volatility[1:len(positions)][trade_mask]
    net_pnl = raw_pnl - costs
    return net_pnl

@njit
```

```
def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
    a0, a1, a2, a3 = mu_params
    b0, b1, b2 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1] + a3 * np.tanh(x[t-1])
        sigma = np.exp(b0 + b1 * np.log1p(np.abs(x[t-1])) + b2)
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * np.random.randn()
    return x

def optimize_threshold(signal, returns, fee, slip, volatility):
    thresholds = np.geomspace(0.001, 0.1, 20)
    best_pnl = -np.inf
    best_thresh = 0.01
    for t in thresholds:
        pos, trades = trading_strategy(signal, t, volatility)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip, vola
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0

    def objective(params):
        mu_params = params[:4]
        sigma_params = params[4:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        pos, trades = trading_strategy(signal, 0.01, np.ones_like(signal)*vola
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip, np

    res = fmin(objective, [0, -0.5, 0.5, 0.1, -2, 0.1, 0.01], sigma0=0.2, opti
    return res[0][:4], res[0][4:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i*200, (i+1)*200) for i in range(5)]
    segment_models = []
    segment_thresholds = []

    for start, end in train_segments:
        if end > len(df_train):
```

```
            continue
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 5
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []

    for start in range(0, len(cv_returns) - window_size + 1, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], windo
            volatility = np.sqrt(np.mean(np.diff(signal)**2))
            pos, trades = trading_strategy(signal, segment_thresholds[i], np.c
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    test_volatility = []

    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        pos, trades = trading_strategy(signal, threshold, np.ones_like(signal)
        test_positions.append(pos)
        test_trades.append(trades)
        test_volatility.extend([volatility]*len(pos))

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_volatility = np.array(test_volatility[:len(fp_positions)])
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns), len(fp_volatility))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
```

```
    fp_volatility = fp_volatility[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    npc_returns = fp_positions * bh_returns - (fee + slip * fp_volatility) * (
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee:.4f}, Slippage={slip:.5f}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:33 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.865589331754940e-02 1.0e+00 2.28e-01  2e-01  3e-01 0:04.5
    2     40 -9.881547612215692e-03 1.5e+00 2.48e-01  2e-01  3e-01 0:04.5
    3     60 -1.537656772190879e-02 1.8e+00 2.84e-01  3e-01  3e-01 0:04.5
  100   2000 -1.306173577874299e-02 1.9e+01 3.76e-01  9e-02  2e-01 0:04.9
  200   4000 -1.345685209424963e-02 8.4e+01 4.99e-01  4e-02  2e-01 0:05.3
  300   6000 -1.347989566576635e-02 1.3e+02 1.89e-01  4e-03  3e-02 0:05.6
  335   6700 -1.347989566576635e-02 1.2e+02 6.41e-02  1e-03  7e-03 0:05.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:28:39 2025)
final/bestever f-value = -1.346710e-02 -1.865589e-02 after 6701/11 evaluations
incumbent solution: [ 1.77788044, -1.63571475, 2.44848813, 1.19564384, -2.36130
106, -0.3517948, -1.88996694]
std deviation: [0.00474003, 0.00099089, 0.00619454, 0.0020067, 0.00295947, 0.00
364893, 0.00687644]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:39 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.248324963887662e-02 1.0e+00 2.00e-01  2e-01  2e-01 0:00.0
    2     40 -1.735811777046355e-02 1.4e+00 1.99e-01  2e-01  2e-01 0:00.0
    3     60 -1.455819703546411e-02 1.4e+00 1.96e-01  2e-01  2e-01 0:00.0
  100   2000 -1.366419597382149e-02 6.8e+01 1.72e-01  2e-02  2e-01 0:00.6
  200   4000 -1.371345041680542e-02 4.3e+02 1.81e-01  1e-02  2e-01 0:01.4
  300   6000 -1.362939698688251e-02 7.4e+02 1.06e+00  2e-02  7e-01 0:02.1
  400   8000 -1.349963644669388e-02 7.4e+02 1.21e+00  8e-03  2e-01 0:02.8
  500  10000 -1.373296537956090e-02 1.7e+03 2.20e+00  1e-02  2e-01 0:03.4
  600  12000 -1.371345041680542e-02 6.7e+03 3.97e+00  1e-02  2e-01 0:03.9
  625  12500 -1.373502979669263e-02 5.4e+03 1.98e+00  6e-03  7e-02 0:04.0
termination on tolstagnation=192 (Tue Jul 22 17:28:45 2025)
final/bestever f-value = -1.341377e-02 -1.735812e-02 after 12501/34 evaluations
incumbent solution: [ 1.90788806, -1.01146948, 4.5637735, -0.23304482, -4.45649
003, 1.72830369, -0.71974638]
std deviation: [0.02598142, 0.00586637, 0.06136764, 0.01717869, 0.07327441, 0.0
35952, 0.03141468]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:45 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.326019106347805e-02 1.0e+00 1.83e-01  2e-01  2e-01 0:00.0
    2     40 -1.292091348177404e-02 1.3e+00 1.84e-01  2e-01  2e-01 0:00.0
    3     60 -1.346260273091175e-02 1.6e+00 1.53e-01  1e-01  2e-01 0:00.0
  100   2000 -1.927148089549917e-02 3.7e+01 4.13e-01  8e-02  6e-01 0:00.3
/tmp/ipython-input-2-2243444373.py:103: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
```

```
  200   4000 -2.341191048233586e-02 2.4e+02 3.76e-01  1e-02  6e-01 0:00.6
  300   6000 -2.063727277544291e-02 1.8e+03 9.90e-02  2e-03  1e-01 0:00.9
  400   8000 -1.912161541608984e-02 4.3e+03 1.10e-01  2e-03  9e-02 0:01.2
  500  10000 -1.974059151709575e-02 4.3e+04 4.18e-01  8e-03  4e-01 0:01.5
  560  11200 -2.015998435578581e-02 1.8e+05 3.37e-01  9e-03  5e-01 0:01.6
termination on tolstagnation=192 (Tue Jul 22 17:28:47 2025)
final/bestever f-value = -2.111056e-03 -3.218309e-02 after 11201/2393 evaluatio
ns
incumbent solution: [-0.02537601, -0.65905462, 0.92710548, 0.66472522, -0.91211
019, -2.38329023, 0.64913239]
std deviation: [0.00920291, 0.04820588, 0.0132609, 0.05700696, 0.2917268, 0.182
66776, 0.48063158]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:47 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -3.747889238525293e-02 1.0e+00 2.06e-01  2e-01  2e-01 0:00.0
    2     40 -3.779090058354680e-02 1.4e+00 2.26e-01  2e-01  2e-01 0:00.0
    3     60 -3.748192637246568e-02 1.6e+00 2.57e-01  2e-01  3e-01 0:00.0
   71   1420 -4.308580750673965e-02 4.8e+02 1.57e-01  4e-04  2e-01 0:00.2
termination on tolfunhist=1e-12 (Tue Jul 22 17:28:48 2025)
final/bestever f-value = -4.308581e-02 -4.871628e-02 after 1421/449 evaluations
incumbent solution: [ 0.48946418, 0.09244575, 0.371696, 0.03516327, -1.8133402
9, -0.05156629, 0.33413629]
std deviation: [0.12690799, 0.00044731, 0.1839425, 0.13587488, 0.11914386, 0.14
229918, 0.17486596]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:48 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -3.840349465540650e-02 1.0e+00 1.83e-01  2e-01  2e-01 0:00.0
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:1452:
RuntimeWarning: invalid value encountered in subtract
  a = op(a[slice1], a[slice2])
```

```
    2     40 -3.768212498877268e-02 1.3e+00 1.67e-01  1e-01  2e-01 0:00.0
    3     60 -2.822935816103023e-02 1.3e+00 1.60e-01  1e-01  2e-01 0:00.0
   91   1820 -4.139052382129690e-02 1.1e+03 1.50e-01  4e-03  3e-01 0:00.3
termination on tolfunhist=1e-12 (Tue Jul 22 17:28:48 2025)
final/bestever f-value = -4.139052e-02 -4.251968e-02 after 1821/138 evaluations
incumbent solution: [-0.80092877, -0.10572393, 5.12763849, -0.04080604, -0.2538
8465, -2.03484853, -2.40614098]
std deviation: [0.02506752, 0.00386756, 0.15965646, 0.0208356, 0.26915238, 0.14
894375, 0.1895651, ]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:48 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.024379769299004e-02 1.0e+00 2.37e-01  2e-01  3e-01 0:00.6
    2     40 -8.057105655962312e-03 1.4e+00 2.82e-01  2e-01  3e-01 0:00.6
    3     60 -8.117880697337751e-03 1.6e+00 3.15e-01  3e-01  4e-01 0:00.6
NOTE (module=cma, iteration=99):
condition in coordinate system exceeded 1.4e+08, rescaled to 1.0e+00,
condition changed from 1.6e+08 to 9.2e+01
  100   2000 -8.164315144759364e-03 9.6e+00 1.41e-01  4e-05  4e-01 0:00.8
NOTE (module=cma, iteration=192):
condition in coordinate system exceeded 1.3e+08, rescaled to 1.0e+00,
condition changed from 3.7e+08 to 2.4e+03
  200   4000 -8.164326301433369e-03 4.8e+01 4.46e-02  3e-09  4e-01 0:01.1
  215   4300 -8.164326301706262e-03 7.3e+01 3.25e-02  6e-10  5e-01 0:01.1
termination on tolfunhist=1e-12 (Tue Jul 22 17:28:49 2025)
final/bestever f-value = -8.164326e-03 -1.024380e-02 after 4301/11 evaluations
incumbent solution: [ 4.52907773e-09, -8.06881219e+00, 1.41916793e-10, 6.851988
38e+00, -4.91585602e+00, -7.16798810e+00, -1.82648858e+01]
std deviation: [2.38208979e-09, 2.43547205e-01, 5.89383151e-10, 2.11592576e-01,
4.42379457e-02, 1.93902724e-01, 5.26409152e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:49 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -8.355767207377140e-03 1.0e+00 2.39e-01  2e-01  3e-01 0:00.0
    2     40 -8.426996287837137e-03 1.5e+00 2.69e-01  2e-01  3e-01 0:00.0
    3     60 -8.434568733555737e-03 1.4e+00 2.78e-01  2e-01  3e-01 0:00.0
  100   2000 -8.524960113329805e-03 8.7e+03 8.80e-02  3e-05  3e-01 0:00.3
NOTE (module=cma, iteration=104):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 1.3e+02
  200   4000 -8.524971187865379e-03 1.1e+04 2.38e-02  3e-09  2e-01 0:00.6
NOTE (module=cma, iteration=205):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 2.4e+08 to 1.4e+03
  221   4420 -8.524971189384362e-03 4.7e+01 1.94e-02  4e-10  2e-01 0:00.6
termination on tolfunhist=1e-12 (Tue Jul 22 17:28:50 2025)
final/bestever f-value = -8.524971e-03 -8.524971e-03 after 4421/4418 evaluation
s
incumbent solution: [ 6.44479803e-09, -8.86718974e-01, 7.54344833e-11, -6.60370
176e-01, -1.99466477e+01, -3.95167439e+00, -2.04809809e+00]
std deviation: [3.37360579e-09, 1.58181321e-02, 4.26618084e-10, 1.50620741e-02,
1.91169404e-01, 8.94612655e-02, 5.25445976e-02]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:50 2025)
```

```
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -8.280304249726370e-03 1.0e+00 2.50e-01  2e-01  3e-01 0:00.0
    2     40 -8.333864274336652e-03 1.5e+00 2.75e-01  2e-01  3e-01 0:00.0
    3     60 -8.303603860518930e-03 1.5e+00 2.91e-01  2e-01  3e-01 0:00.0
  100   2000 -8.445754786770811e-03 1.5e+04 1.81e-01  5e-05  4e-01 0:00.3
NOTE (module=cma, iteration=102):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.4e+08 to 5.1e+02
  200   4000 -8.445774124060636e-03 6.8e+03 2.78e-01  3e-08  7e-01 0:00.6
NOTE (module=cma, iteration=209):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.2e+08 to 2.4e+02
  235   4700 -8.445774130225344e-03 1.5e+01 1.48e-01  1e-09  4e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:28:51 2025)
final/bestever f-value = -8.445774e-03 -8.445774e-03 after 4701/4693 evaluation
s
incumbent solution: [ 4.62936058e-09, -1.89896674e+01, -5.46086552e-11, 1.79180
299e+01, -8.28772973e+00, -3.67513374e+00, -1.36091660e+01]
std deviation: [5.78796775e-09, 2.70933416e-01, 1.08367464e-09, 2.51396548e-01,
1.22248938e-01, 3.50480205e-01, 3.39631184e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:51 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -3.483812166354031e-02 1.0e+00 2.33e-01  2e-01  3e-01 0:00.0
    2     40 -3.516830217480201e-02 1.5e+00 2.92e-01  3e-01  3e-01 0:00.0
    3     60 -3.521173336785900e-02 1.6e+00 3.56e-01  3e-01  4e-01 0:00.0
  100   2000 -3.528190885066668e-02 4.9e+03 8.62e-02  4e-05  2e-01 0:00.3
NOTE (module=cma, iteration=108):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.1e+08 to 7.1e+01
/tmp/ipython-input-2-2243444373.py:103: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
```

```
  200   4000 -3.701721554827758e-02 1.6e+02 1.59e-01  2e-06  3e-01 0:00.6
  300   6000 -3.708262187449086e-02 2.2e+03 4.25e-02  5e-08  4e-02 0:00.8
  400   8000 -3.701726748190457e-02 4.6e+03 2.21e-02  4e-09  1e-02 0:01.1
  500  10000 -3.701726741056538e-02 3.8e+04 2.20e-02  1e-09  1e-02 0:01.4
  600  12000 -3.698687400873157e-02 4.7e+04 1.23e-02  2e-10  3e-03 0:01.7
  615  12300 -3.698687402815318e-02 4.1e+04 8.44e-03  9e-11  2e-03 0:01.7
termination on tolstagnation=192 (Tue Jul 22 17:28:54 2025)
final/bestever f-value = -3.682826e-02 -3.728157e-02 after 12301/10373 evaluati
ons
incumbent solution: [ 5.95305919e-07, -2.11555316e+00, -3.86899515e-08, 1.34199
600e-01, -1.31975844e+01, 1.27627855e+00, -4.97115533e+00]
std deviation: [3.03204565e-09, 1.81295538e-04, 8.79861127e-11, 1.84298457e-04,
1.64390580e-03, 1.16583142e-03, 1.64813530e-03]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:54 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.153279676891384e-02 1.0e+00 1.95e-01  2e-01  2e-01 0:00.0
    2     40 -8.008389482627519e-03 1.3e+00 2.39e-01  2e-01  3e-01 0:00.0
    3     60 -2.724532560951949e-03 1.4e+00 2.59e-01  2e-01  3e-01 0:00.0
  100   2000 -2.459423446614810e-02 2.4e+02 1.13e-01  2e-02  2e-01 0:00.3
  200   4000 -2.458909550278620e-02 9.4e+02 2.29e-01  1e-02  2e-01 0:00.6
  300   6000 -2.464487627135241e-02 2.0e+03 2.91e-01  3e-03  1e-01 0:01.0
  400   8000 -2.465438570394406e-02 3.7e+03 2.86e-01  1e-03  4e-02 0:01.4
  500  10000 -2.469131814677678e-02 1.8e+04 7.24e-02  3e-04  8e-03 0:01.8
  600  12000 -2.473030992787972e-02 4.1e+04 8.46e-02  2e-04  5e-03 0:02.2
  700  14000 -2.469570729524282e-02 1.6e+05 9.00e-02  1e-04  4e-03 0:02.6
  800  16000 -2.469337904609135e-02 3.7e+05 5.47e-02  4e-05  2e-03 0:02.9
/usr/local/lib/python3.11/dist-packages/cma/utilities/utils.py:349: UserWarnin
g:
        geno-pheno transformation introduced based on the
        current covariance matrix with condition 1.0e+12 -> 1.0e+00,
        injected solutions become "invalid" in this iteration (time=Jul 22 17:2
8:58 2025 class=CMAEvolutionStrategy method=alleviate_conditioning iteration=84
5)
  warnings.warn(msg + ' (time={}'.format(time.asctime()[4:]) +
```

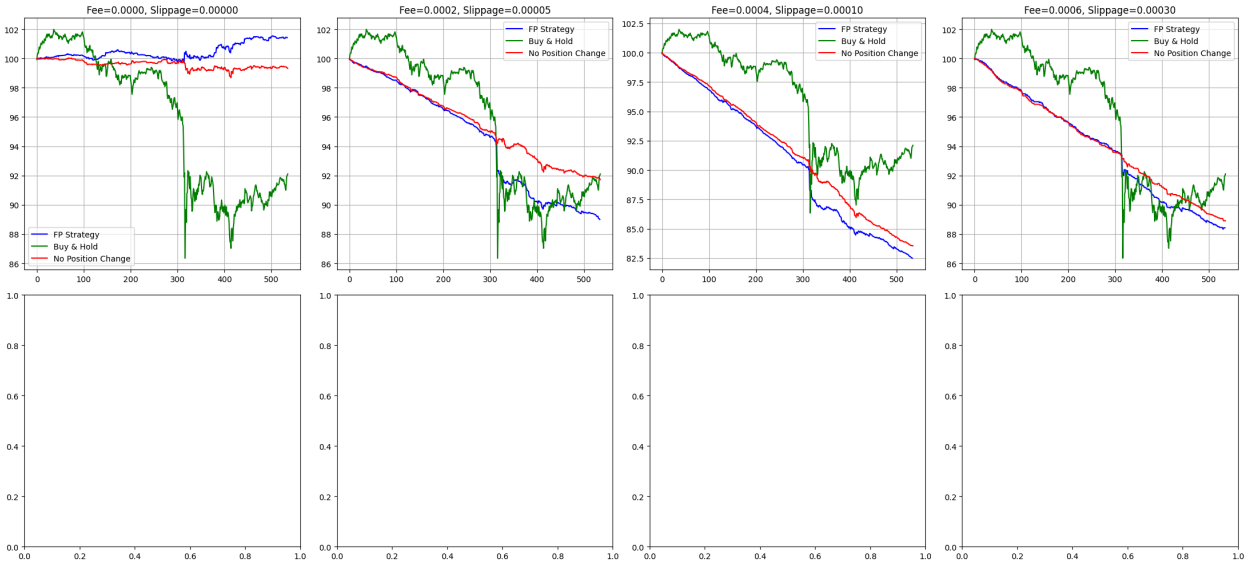
```
  900  18000 -2.469524659971337e-02 6.3e+00 5.16e-02  3e-02  6e-02 0:03.2
 1000  20000 -2.480792821856578e-02 3.1e+01 1.64e-01  3e-02  2e-01 0:03.5
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
 1040  20800 -2.469500447185911e-02 6.0e+01 1.42e-01  2e-02  2e-01 0:03.7
termination on tolstagnation=192 (Tue Jul 22 17:28:59 2025)
final/bestever f-value = -2.305821e-02 -3.436310e-02 after 20801/2069 evaluatio
ns
incumbent solution: [-0.03115427, -0.78712754, 0.1092173, 0.77442998, -4.147211
73, -2.41512886, -0.6447994, ]
std deviation: [0.01558955, 0.12795157, 0.05171175, 0.01613424, 0.09016415, 0.1
6696917, 0.08162757]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
8:59 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 1.506197321451245e-02 1.0e+00 2.30e-01  2e-01  2e-01 0:00.0
    2     40 1.449270817036060e-02 1.3e+00 2.54e-01  2e-01  3e-01 0:00.0
    3     60 -5.712375153276003e-03 1.4e+00 2.42e-01  2e-01  3e-01 0:00.0
  100   2000 -5.964302904157719e-03 5.0e+03 1.62e-01  6e-05  3e-01 0:00.3
NOTE (module=cma, iteration=113):
condition in coordinate system exceeded 1.4e+08, rescaled to 1.0e+00,
condition changed from 1.6e+08 to 3.9e+01
  200   4000 -5.964326297114181e-03 3.7e+03 1.19e-01  9e-09  3e-01 0:00.6
NOTE (module=cma, iteration=218):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 3.7e+01
  244   4880 -5.964326301754329e-03 1.5e+01 1.37e-01  5e-10  4e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:29:00 2025)
final/bestever f-value = -5.964326e-03 -5.964326e-03 after 4881/4817 evaluation
s
incumbent solution: [ 3.09433638e-09, -8.12749993e-01, 3.95560700e-11, -1.00718
078e-01, -9.09937035e+00, -4.21120297e+00, -1.55533431e+01]
std deviation: [2.82941982e-09, 1.35586395e-01, 4.91858648e-10, 2.22911169e-01,
2.71839752e-01, 1.52912199e-01, 4.44343346e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:00 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -5.996353779759270e-03 1.0e+00 2.45e-01  2e-01  3e-01 0:00.0
    2     40 -6.156772252196675e-03 1.5e+00 2.71e-01  2e-01  3e-01 0:00.0
    3     60 -6.094701067489123e-03 1.5e+00 2.87e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=94):
condition in coordinate system exceeded 1.4e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 4.1e+02
  100   2000 -6.324945458171412e-03 2.1e+01 2.94e-01  5e-05  1e+00 0:00.3
NOTE (module=cma, iteration=193):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 2.3e+08 to 9.0e+03
  200   4000 -6.324971187411557e-03 8.4e+01 6.24e-02  3e-09  6e-01 0:00.6
  227   4540 -6.324971189355053e-03 6.7e+01 4.54e-02  2e-10  4e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:29:01 2025)
final/bestever f-value = -6.324971e-03 -6.324971e-03 after 4541/4517 evaluation
s
incumbent solution: [ 2.20485674e-09, 7.33322293e+00, 3.90927627e-11, -9.144151
75e+00, -6.25875008e+00, 2.32994559e+01, -1.68377804e+01]
std deviation: [1.54343598e-09, 1.36524739e-01, 2.14377243e-10, 1.56971840e-01,
```

```
6.33919355e-02, 4.39299855e-01, 2.80386638e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:01 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -2.428849807562106e-03 1.0e+00 2.39e-01  2e-01  3e-01 0:00.0
    2     40 -6.010292133093996e-03 1.4e+00 2.63e-01  2e-01  3e-01 0:00.0
    3     60 -6.094030786003840e-03 1.4e+00 2.72e-01  2e-01  3e-01 0:00.0
  100   2000 -6.245756241842601e-03 9.4e+03 6.90e-02  2e-05  2e-01 0:00.3
NOTE (module=cma, iteration=102):
condition in coordinate system exceeded 1.3e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 5.5e+01
  200   4000 -6.245774123373815e-03 6.5e+03 5.16e-02  8e-09  3e-01 0:00.6
NOTE (module=cma, iteration=210):
condition in coordinate system exceeded 1.5e+08, rescaled to 1.0e+00,
condition changed from 2.1e+08 to 3.4e+02
  247   4940 -6.245774130390690e-03 7.4e+01 1.34e-01  5e-10  1e+00 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:29:02 2025)
final/bestever f-value = -6.245774e-03 -6.245774e-03 after 4941/4878 evaluation
s
incumbent solution: [ 2.31878326e-09, 3.10633890e+00, 6.69025502e-11, -4.084888
82e+00, -8.37743363e+00, -1.50212867e+01, -1.69700107e+01]
std deviation: [3.59401764e-09, 3.95330621e-01, 4.91730524e-10, 3.11681505e-01,
2.72716225e-01, 1.28043755e+00, 9.08947760e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:02 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -1.073912000151813e-02 1.0e+00 2.35e-01  2e-01  3e-01 0:00.0
    2     40 -3.188580141155666e-02 1.5e+00 2.38e-01  2e-01  3e-01 0:00.0
    3     60 -3.294380006801986e-02 1.5e+00 2.73e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=98):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.6e+08 to 8.8e+01
  100   2000 -3.308191348549014e-02 8.7e+00 8.68e-02  2e-05  2e-01 0:00.3
NOTE (module=cma, iteration=195):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 2.2e+02
  200   4000 -3.308192637069080e-02 1.3e+01 2.82e-02  2e-09  3e-01 0:00.6
  221   4420 -3.308192637221530e-02 1.9e+01 2.47e-02  4e-10  4e-01 0:00.6
termination on tolfun=1e-11 (Tue Jul 22 17:29:03 2025)
final/bestever f-value = -3.308193e-02 -3.319685e-02 after 4421/1327 evaluation
s
incumbent solution: [ 2.30049275e-09, -1.87021804e+00, -2.98640012e-11, 7.68852
398e-01, -7.35266447e+00, 6.35365714e+00, -1.57299499e+01]
std deviation: [2.15627504e-09, 4.18809678e-02, 3.63282626e-10, 5.02501321e-02,
4.52142765e-02, 9.36448942e-02, 4.03768528e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:03 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 6.531098390667942e-04 1.0e+00 2.27e-01  2e-01  2e-01 0:00.0
    2     40 -6.781124081943628e-03 1.4e+00 2.87e-01  3e-01  3e-01 0:00.0
    3     60 -4.611715409472156e-04 1.7e+00 3.47e-01  3e-01  4e-01 0:00.0
  100   2000 -6.038560340303426e-04 5.5e+03 1.14e-01  6e-05  3e-01 0:00.3
NOTE (module=cma, iteration=113):
condition in coordinate system exceeded 1.3e+08, rescaled to 1.0e+00,
```

```
condition changed from 2.1e+08 to 6.4e+01
  200   4000 -6.038889586625303e-04 4.0e+03 3.19e-02  8e-09  2e-01 0:00.6
NOTE (module=cma, iteration=211):
condition in coordinate system exceeded 1.3e+08, rescaled to 1.0e+00,
condition changed from 2.2e+08 to 5.7e+03
  238   4760 -6.038889664691780e-04 1.3e+02 2.27e-02  3e-10  5e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:29:04 2025)
final/bestever f-value = -6.038890e-04 -1.328810e-02 after 4761/677 evaluations
incumbent solution: [-1.24181275e-09, -2.88530714e+00, -1.03273310e-11, 1.56046
406e+00, -1.61275031e+01, -1.11966697e+01, -7.99035678e+00]
std deviation: [1.39385946e-09, 9.44109592e-02, 2.74961857e-10, 8.45497165e-02,
5.06637418e-01, 3.25476772e-01, 9.29686221e-02]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:04 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 1.780980119869510e-02 1.0e+00 2.29e-01  2e-01  2e-01 0:00.0
    2     40 -3.247343212775585e-03 1.4e+00 3.13e-01  3e-01  4e-01 0:00.0
    3     60 -3.354936543746356e-03 1.7e+00 3.53e-01  3e-01  4e-01 0:00.0
NOTE (module=cma, iteration=95):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 8.1e+02
  100   2000 -3.764271269575610e-03 2.7e+01 9.01e-02  2e-05  4e-01 0:00.3
NOTE (module=cma, iteration=197):
condition in coordinate system exceeded 1.3e+08, rescaled to 1.0e+00,
condition changed from 3.0e+08 to 4.1e+03
  200   4000 -3.764326288255369e-03 6.8e+01 3.14e-02  2e-09  3e-01 0:00.6
  237   4740 -3.764326301362078e-03 7.9e+01 7.17e-02  3e-10  8e-01 0:00.7
termination on tolfun=1e-11 (Tue Jul 22 17:29:05 2025)
final/bestever f-value = -3.764326e-03 -3.764326e-03 after 4741/4726 evaluation
s
incumbent solution: [ 1.72024251e-09, -8.81156574e+00, -1.87946771e-11, 7.29832
495e+00, -3.74792177e+00, -5.63539865e-01, -2.03023148e+01]
std deviation: [2.88025295e-09, 2.27577219e-01, 2.98369938e-10, 1.66079708e-01,
1.80340952e-01, 2.25241769e-01, 7.62290996e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:05 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -3.165342497479497e-03 1.0e+00 2.46e-01  2e-01  3e-01 0:00.0
    2     40 -3.458045385762018e-03 1.5e+00 2.82e-01  2e-01  3e-01 0:00.0
    3     60 -3.472659631220387e-03 1.4e+00 3.15e-01  3e-01  3e-01 0:00.0
  100   2000 -4.124907576174095e-03 1.2e+04 1.17e-01  3e-05  3e-01 0:00.3
NOTE (module=cma, iteration=100):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.5e+08 to 2.0e+01
NOTE (module=cma, iteration=192):
condition in coordinate system exceeded 1.3e+08, rescaled to 1.0e+00,
condition changed from 1.2e+08 to 3.6e+01
  200   4000 -4.124971183532770e-03 4.7e+00 5.06e-02  4e-09  3e-01 0:00.6
  241   4820 -4.124971189552524e-03 1.3e+02 3.41e-02  9e-11  3e-01 0:00.7
termination on tolfunhist=1e-12 (Tue Jul 22 17:29:06 2025)
final/bestever f-value = -4.124971e-03 -4.124971e-03 after 4821/4633 evaluation
s
incumbent solution: [ 3.73212003e-10, -4.31127160e-01, -8.05913744e-12, 2.49031
462e-01, -4.51720723e+00, -7.50166406e+00, -1.98250861e+01]
```

```
std deviation: [2.53341581e-10, 3.29227821e-02, 8.86209897e-11, 4.55893674e-02,
3.96375537e-02, 1.34545450e-01, 2.99688814e-01]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:06 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -2.053853569118362e-03 1.0e+00 2.36e-01  2e-01  3e-01 0:00.0
    2     40 -3.438943506064186e-03 1.5e+00 2.84e-01  2e-01  3e-01 0:00.0
    3     60 -3.420520839718777e-03 1.6e+00 2.86e-01  2e-01  3e-01 0:00.0
  100   2000 -4.045698779226953e-03 1.1e+04 2.52e-01  8e-05  8e-01 0:00.3
NOTE (module=cma, iteration=100):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.2e+08 to 2.0e+02
NOTE (module=cma, iteration=197):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 2.4e+08 to 7.0e+02
  200   4000 -4.045774127055141e-03 3.1e+01 2.36e-02  1e-09  2e-01 0:00.6
  227   4540 -4.045774130369110e-03 7.6e+01 3.05e-02  2e-10  4e-01 0:00.7
termination on tolfun=1e-11 (Tue Jul 22 17:29:07 2025)
final/bestever f-value = -4.045774e-03 -4.045774e-03 after 4541/4481 evaluation
s
incumbent solution: [ 6.31284347e-10, -8.43937861e-01, -2.64294421e-11, 3.33055
646e-02, -2.00253314e+01, 2.68843962e-01, -4.42461731e+00]
std deviation: [9.97629793e-10, 1.21843007e-01, 2.10428300e-10, 1.20981942e-01,
3.97747991e-01, 1.77228182e-01, 7.61064317e-02]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:07 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 -2.863633188517489e-02 1.0e+00 2.36e-01  2e-01  3e-01 0:00.0
    2     40 -2.996896082794025e-02 1.5e+00 2.74e-01  2e-01  3e-01 0:00.0
    3     60 -3.037952602687343e-02 1.7e+00 3.10e-01  3e-01  4e-01 0:00.0
NOTE (module=cma, iteration=93):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 6.5e+01
  100   2000 -3.088188732274574e-02 9.8e+00 1.64e-01  3e-05  8e-01 0:00.3
NOTE (module=cma, iteration=190):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.0e+08 to 9.3e+01
  200   4000 -3.088192637041853e-02 1.3e+01 2.63e-02  2e-09  3e-01 0:00.6
  208   4160 -3.088192637165000e-02 1.2e+01 2.35e-02  7e-10  3e-01 0:00.7
termination on tolfun=1e-11 (Tue Jul 22 17:29:07 2025)
final/bestever f-value = -3.088193e-02 -3.088193e-02 after 4161/4147 evaluation
s
incumbent solution: [ 1.48146811e-09, 3.22056771e-01, -1.37243017e-10, -5.56825
669e-01, -1.87296289e+01, 2.91800766e+00, -3.40828663e+00]
std deviation: [2.15478533e-09, 3.45487240e-02, 7.38747745e-10, 4.25480695e-02,
2.62959531e-01, 3.21849569e-02, 6.68011249e-02]
(10_w,20)-aCMA-ES (mu_w=5.9,w_1=27%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:08 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     20 4.395567408033896e-03 1.0e+00 2.25e-01  2e-01  2e-01 0:00.0
    2     40 2.229413264210941e-03 1.4e+00 2.68e-01  2e-01  3e-01 0:00.0
    3     60 -3.623207155538475e-03 1.9e+00 2.78e-01  2e-01  4e-01 0:00.0
NOTE (module=cma, iteration=97):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
```

```
condition changed from 1.0e+08 to 2.5e+02
  100   2000 1.596149013059719e-03 1.4e+01 1.65e-01  6e-05  6e-01 0:00.4
  200   4000 1.596111048027896e-03 1.2e+04 3.77e-02  8e-09  2e-01 0:00.8
NOTE (module=cma, iteration=203):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 2.5e+08 to 5.7e+03
  222   4440 1.596111034863455e-03 9.4e+01 3.44e-02  1e-09  3e-01 0:00.9
termination on tolfun=1e-11 (Tue Jul 22 17:29:09 2025)
final/bestever f-value = 1.596111e-03 -3.623207e-03 after 4441/43 evaluations
incumbent solution: [-4.18939709e-09, 1.19487655e+00, 2.13265950e-10, -1.585781
39e+00, -6.17627678e+00, -5.90585793e+00, -1.58218591e+01]
std deviation: [2.66818724e-09, 6.69427389e-02, 1.14182962e-09, 8.05302657e-02,
6.12127817e-02, 9.59493077e-02, 2.85527020e-01]
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000           101.42           1.42           92.11
-7.89    99.33           -0.67
0.0002   0.00005            89.00         -11.00           92.11
-7.89    91.73           -8.27
0.0004   0.00010            82.49         -17.51           92.11
-7.89    83.55          -16.45
0.0006   0.00030            88.42         -11.58           92.11
-7.89    88.91          -11.09
```

In [ ]:

```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt
from numba import njit
from sklearn.preprocessing import RobustScaler

np.random.seed(42)
random_seed = 42

# Load 1-minute data
df = pd.read_csv("BTC_1min.csv", parse_dates=['system_time'], index_col='syste
```

```python
# Feature engineering with noise reduction
for j in range(15):
    noise_factor = 0.05  # Reduced noise for 1min data
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}'] * (1 + nc
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}'] * (1 + nc

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]

# Enhanced OBI calculation with smoothing
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().rolling(5, min_periods=1).mean().fillna(0)  # 5-
df['depth'] = np.log1p(df[bid_cols + ask_cols].sum(axis=1))
df['queue_slope'] = (df['bids_notional_0'] - df['bids_notional_5']) / (df['bid

# Feature scaling
scaler = RobustScaler()
features = ['obi', 'dobi', 'depth', 'queue_slope']
df[features] = scaler.fit_transform(df[features])

# Adjusted time splits for 1min data (more recent test set)
train_end = int(len(df) * 0.5)  # 50% training
cv_end = int(len(df) * 0.75)    # 25% validation
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

# Returns calculation
for df_part in [df_train, df_cv, df_test]:
    df_part['log_mid'] = np.log(df_part['midpoint'])
    df_part['returns'] = df_part['log_mid'].diff().fillna(0)

# Trading strategy with position smoothing
@njit
def trading_strategy(signal, threshold, volatility):
    positions = np.zeros(len(signal))
    for i in range(1, len(signal)):
        z_score = signal[i] / (volatility[i] + 1e-8)
        if z_score > threshold:
            positions[i] = min(positions[i-1] + 0.05, 1)  # Slower position bu
        elif z_score < -threshold:
            positions[i] = max(positions[i-1] - 0.05, -1)
        else:
            positions[i] = positions[i-1] * 0.98  # Slower decay
    # Manual diff calculation
    trades = np.zeros(len(positions))
    trades[0] = positions[0]
    for i in range(1, len(positions)):
        trades[i] = positions[i] - positions[i-1]
    return positions, trades

@njit
```

```
def apply_trading_costs(positions, trades, returns, fee, slip, volatility):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip * volatility[1:len(positions)][trade_mask]
    net_pnl = raw_pnl - costs
    return net_pnl

@njit
def simulate_fp(mu_params, sigma_params, x0, obi, timesteps, dt):
    a0, a1, a2, a3 = mu_params
    b0, b1, b2 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1] + a3 * np.tanh(x[t-1]/3.0)  # Sm
        sigma = np.exp(b0 + b1 * np.log1p(np.abs(x[t-1])/10.0) + b2)  # More s
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * np.random.randn()
    return x

def optimize_threshold(signal, returns, fee, slip, volatility):
    thresholds = np.geomspace(0.0005, 0.05, 25)  # Wider range for 1min
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t, volatility)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip, vola
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0
    dt = 1.0

    def objective(params):
        mu_params = params[:4]
        sigma_params = params[4:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        pos, trades = trading_strategy(signal, 0.005, np.ones_like(signal)*vol
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip, np

    res = fmin(objective, [0, -0.3, 0.3, 0.05, -1.5, 0.05, 0.005],
               sigma0=0.15, options={'seed':random_seed, 'popsize':25, 'maxite
    return res[0][:4], res[0][4:]

# Adjusted fee structure for 1min trading
fees = [0, 0.0001, 0.0002, 0.0003]
slippages = [0, 0.00002, 0.00005, 0.0001]
```

```
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    # Adjusted training segments for 1min (smaller windows)
    train_segments = [(i*500, (i+1)*500) for i in range(6)]  # 500-min (8.3hr)
    segment_models = []
    segment_thresholds = []

    for start, end in train_segments:
        if end > len(df_train):
            continue
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end]['obi
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    # Adjusted window size for 1min (30-min windows)
    window_size = 30
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values
    selected_model_indices = []

    for start in range(0, len(cv_returns) - window_size + 1, window_size//2):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, cv_obi[start:end], windo
            volatility = np.sqrt(np.mean(np.diff(signal)**2))
            pos, trades = trading_strategy(signal, segment_thresholds[i], np.c
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []
    test_volatility = []

    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_obi[start:end], window_s
        volatility = np.sqrt(np.mean(np.diff(signal)**2))
```

```
        pos, trades = trading_strategy(signal, threshold, np.ones_like(signal)
        test_positions.append(pos)
        test_trades.append(trades)
        test_volatility.extend([volatility]*len(pos))

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_volatility = np.array(test_volatility[:len(fp_positions)])
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns), len(fp_volatility))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_volatility = fp_volatility[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    npc_returns = fp_positions * bh_returns - (fee + slip * fp_volatility) * (
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee:.4f}, Slippage={slip:.5f}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:11 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -6.218165185803152e-03 1.0e+00 1.57e-01  1e-01  2e-01 0:01.3
    2     50 -3.756141581336973e-03 1.3e+00 1.59e-01  1e-01  2e-01 0:01.3
    3     75 -1.510502296541780e-02 1.5e+00 1.61e-01  1e-01  2e-01 0:01.3
/tmp/ipython-input-3-1096312254.py:108: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:1452:
RuntimeWarning: invalid value encountered in subtract
  a = op(a[slice1], a[slice2])
```

```
  100   2500 -2.958879900155347e-02 5.2e+01 1.65e-01  2e-02  2e-01 0:01.7
  150   3750 -3.064222685493495e-02 1.1e+02 4.82e-02  3e-03  3e-02 0:02.0
termination on maxiter=150 (Tue Jul 22 17:29:14 2025)
final/bestever f-value = -2.234882e-02 -3.536534e-02 after 3751/186 evaluations
incumbent solution: [-0.20356515, -0.27016074, -2.91851938, 1.03296212, -2.6439
0289, 0.27879838, 1.75327956]
std deviation: [0.00272984, 0.0048344, 0.03026478, 0.01626592, 0.02263051, 0.02
948489, 0.02101151]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:14 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -1.651478708301495e-02 1.0e+00 1.61e-01  2e-01  2e-01 0:00.0
    2     50 -1.176861108684326e-02 1.4e+00 1.64e-01  1e-01  2e-01 0:00.0
    3     75 -1.225114237091240e-02 1.5e+00 1.60e-01  1e-01  2e-01 0:00.0
  100   2500 -1.318131665502729e-02 1.8e+01 1.70e-01  5e-02  1e-01 0:00.5
  150   3750 -1.284766160891513e-02 6.7e+01 1.93e-01  3e-02  2e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:15 2025)
final/bestever f-value = -1.174004e-02 -1.665057e-02 after 3751/2814 evaluation
s
incumbent solution: [ 0.23992425, -0.4039523, 0.48392292, -0.30424847, -1.50762
435, -0.04569542, 0.16095444]
std deviation: [0.02718795, 0.08101687, 0.05677429, 0.09026037, 0.12267573, 0.1
1059145, 0.16292049]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:15 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -8.852844332382849e-03 1.0e+00 1.70e-01  2e-01  2e-01 0:00.0
    2     50 -8.465886873131191e-03 1.5e+00 2.03e-01  2e-01  3e-01 0:00.0
    3     75 -9.398292930713481e-03 1.8e+00 2.21e-01  2e-01  3e-01 0:00.0
   35    875 -1.013893949988844e-02 5.6e+01 3.76e-02  1e-03  4e-02 0:00.2
termination on tolflatfitness=1 (Tue Jul 22 17:29:15 2025)
final/bestever f-value = -1.013894e-02 -1.013894e-02 after 876/180 evaluations
incumbent solution: [ 0.37514562, 0.21823093, 0.29541059, -0.0692322, -1.105234
95, 0.02117173, -0.00203955]
std deviation: [0.02578678, 0.00100976, 0.03277523, 0.03091648, 0.04323557, 0.0
198326, 0.04153768]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:15 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -6.037349136487879e-03 1.0e+00 1.60e-01  1e-01  2e-01 0:00.0
    2     50 -8.609059113819305e-03 1.4e+00 1.56e-01  1e-01  2e-01 0:00.0
    3     75 -6.556461744611059e-03 1.4e+00 1.50e-01  1e-01  2e-01 0:00.0
  100   2500 -6.088260489506682e-03 3.2e+01 7.21e-02  2e-02  7e-02 0:00.4
  150   3750 -6.060955604900186e-03 3.3e+01 4.91e-02  1e-02  4e-02 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:16 2025)
final/bestever f-value = -5.905757e-03 -8.609059e-03 after 3751/41 evaluations
incumbent solution: [ 0.42388367, -0.35063268, 0.72711485, -0.02797295, -1.8063
4309, -0.25892072, -0.2320492, ]
std deviation: [0.01044476, 0.01585845, 0.01084566, 0.04099918, 0.03261578, 0.0
2324314, 0.02616796]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:16 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -1.118955390579609e-02 1.0e+00 1.47e-01  1e-01  2e-01 0:00.0
```

```
    2     50 -1.397788406038184e-02 1.3e+00 1.40e-01  1e-01  1e-01 0:00.0
    3     75 -1.616713857344926e-02 1.4e+00 1.32e-01  1e-01  1e-01 0:00.0
  100   2500 -1.209882185229709e-02 1.3e+02 2.56e-01  2e-02  3e-01 0:00.5
  113   2825 -1.209882185229709e-02 1.9e+02 1.13e-01  8e-03  8e-02 0:00.5
termination on tolfunhist=1e-12 (Tue Jul 22 17:29:16 2025)
final/bestever f-value = -1.209882e-02 -1.616714e-02 after 2826/63 evaluations
incumbent solution: [ 1.21076659, -0.05100807, 2.43096005, -1.60932393, -3.1215
2306, -0.17546728, -1.46962156]
std deviation: [0.04077859, 0.00843253, 0.07911299, 0.01898675, 0.06410836, 0.0
4715728, 0.08447037]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:16 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -1.158321717780932e-02 1.0e+00 1.70e-01  2e-01  2e-01 0:00.0
    2     50 -1.151022176585119e-02 1.5e+00 1.91e-01  2e-01  2e-01 0:00.0
    3     75 -1.160988674716368e-02 1.6e+00 1.98e-01  2e-01  2e-01 0:00.0
  100   2500 -2.033885046348534e-02 1.7e+02 1.05e-01  1e-02  1e-01 0:00.5
  150   3750 -2.042321788267430e-02 5.7e+02 3.82e-02  2e-03  4e-02 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:17 2025)
final/bestever f-value = -1.949474e-02 -2.051832e-02 after 3751/1288 evaluation
s
incumbent solution: [ 0.11857013, -0.23668812, 0.97151074, 1.2901327, -3.045343
23, 0.00324726, 0.27611065]
std deviation: [0.00197617, 0.00415961, 0.01035653, 0.02177164, 0.03452867, 0.0
2402654, 0.04367541]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:17 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 2.642897232208833e-03 1.0e+00 1.66e-01  2e-01  2e-01 0:00.6
    2     50 4.124977335946099e-04 1.4e+00 2.09e-01  2e-01  3e-01 0:00.6
    3     75 5.982309120425126e-04 1.5e+00 2.75e-01  2e-01  3e-01 0:00.6
NOTE (module=cma, iteration=97):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 2.7e+01
  100   2500 5.384649431401257e-04 5.1e+00 2.49e-01  2e-05  3e-01 0:01.0
  150   3750 5.384611636921489e-04 4.2e+02 2.03e-01  1e-07  4e-01 0:01.2
termination on maxiter=150 (Tue Jul 22 17:29:19 2025)
final/bestever f-value = 5.384612e-04 4.124977e-04 after 3751/45 evaluations
incumbent solution: [-7.42590717e-07, -3.96735875e-01, -1.75834281e-08, -7.7802
8599e-01, -7.54656367e+00, -7.15372314e-01, -1.03916143e+01]
std deviation: [8.99183142e-07, 1.41772000e-01, 1.27982257e-07, 3.61979721e-01,
2.08198166e-01, 2.16997011e-01, 4.42951811e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:19 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -2.968729237370760e-03 1.0e+00 1.72e-01  1e-01  2e-01 0:00.0
    2     50 -6.370048159692578e-03 1.6e+00 1.70e-01  1e-01  2e-01 0:00.0
    3     75 -7.501495839263902e-03 1.6e+00 1.91e-01  2e-01  2e-01 0:00.0
  100   2500 -8.935758510279151e-03 1.7e+02 1.22e-01  6e-03  2e-01 0:00.5
  150   3750 -8.777949013127085e-03 4.6e+02 4.78e-02  2e-03  6e-02 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:20 2025)
final/bestever f-value = -8.197569e-03 -9.075632e-03 after 3751/1558 evaluation
s
incumbent solution: [ 0.06671154, -0.22627532, 0.20019959, 0.97247514, -3.95034
```

```
563, -0.22954197, -0.89156392]
std deviation: [0.00152807, 0.00716821, 0.00511126, 0.03215137, 0.05705992, 0.0
1932305, 0.02690674]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:20 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -4.148373938259885e-03 1.0e+00 1.68e-01  1e-01  2e-01 0:00.0
    2     50 -4.172498671346354e-03 1.5e+00 1.97e-01  2e-01  2e-01 0:00.0
    3     75 -4.184920198331490e-03 1.7e+00 2.33e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=89):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.2e+08 to 2.8e+01
  100   2500 -4.261064092041729e-03 7.2e+00 1.38e-01  1e-05  4e-01 0:00.6
  150   3750 -4.261064757642724e-03 5.6e+02 6.64e-02  6e-08  2e-01 0:00.9
termination on maxiter=150 (Tue Jul 22 17:29:21 2025)
final/bestever f-value = -4.261065e-03 -4.261065e-03 after 3751/3689 evaluation
s
incumbent solution: [ 3.69383558e-07, -1.24248989e+00, -4.17175342e-09, 2.77059
336e+00, -1.15686023e+01, 2.71173367e+00, -5.88376216e+00]
std deviation: [2.73937437e-07, 4.82017561e-02, 5.53818632e-08, 8.47942938e-02,
1.55039297e-01, 1.07927575e-01, 1.67452229e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:21 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -2.997677033022720e-03 1.0e+00 1.80e-01  2e-01  2e-01 0:00.0
    2     50 -3.593271949710332e-03 1.5e+00 1.97e-01  2e-01  2e-01 0:00.0
    3     75 -3.608900004420507e-03 1.6e+00 2.29e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=87):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 3.7e+01
  100   2500 -3.672140466058089e-03 8.7e+00 1.35e-01  1e-05  3e-01 0:00.6
  150   3750 -3.672141505678101e-03 8.6e+02 8.56e-02  5e-08  3e-01 0:01.0
termination on maxiter=150 (Tue Jul 22 17:29:22 2025)
final/bestever f-value = -3.672142e-03 -3.672142e-03 after 3751/3737 evaluation
s
incumbent solution: [ 5.56688829e-07, -2.07040365e+00, -3.22006607e-09, 1.57949
724e+00, -1.10017138e+01, -4.01355772e+00, -7.76099365e+00]
std deviation: [5.21303949e-07, 7.89918017e-02, 5.30171903e-08, 5.33241414e-02,
2.98107157e-01, 1.41456639e-01, 2.40217975e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:22 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -9.282072939524409e-03 1.0e+00 1.85e-01  2e-01  2e-01 0:00.0
    2     50 -8.822617449722231e-03 1.5e+00 2.07e-01  2e-01  2e-01 0:00.0
    3     75 -9.314277465947447e-03 1.5e+00 2.84e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=90):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.9e+08 to 1.2e+01
  100   2500 -9.378594935186939e-03 5.2e+00 4.73e-01  3e-05  1e+00 0:00.6
/tmp/ipython-input-3-1096312254.py:108: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
```

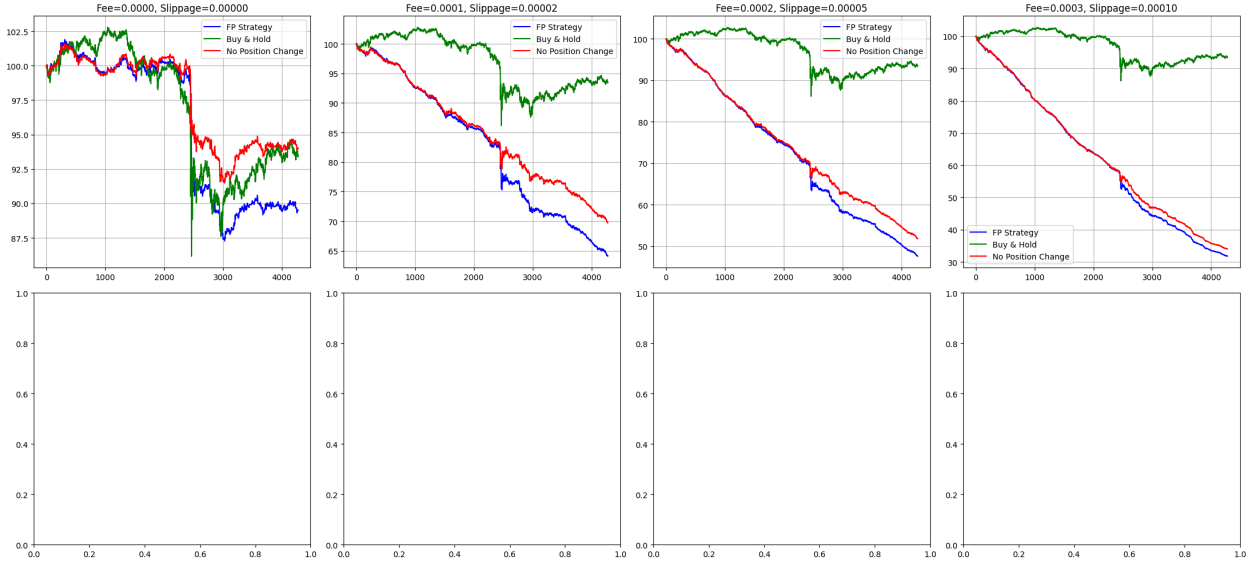
```
  150   3750 -9.378598406765979e-03 6.6e+02 1.57e-01  9e-08  5e-01 0:00.9
termination on maxiter=150 (Tue Jul 22 17:29:23 2025)
final/bestever f-value = -9.378598e-03 -9.378598e-03 after 3751/3651 evaluation
s
incumbent solution: [ 6.93367618e-07, -9.38710459e-01, 5.55783365e-09, 1.973810
94e+00, -1.22114923e+01, 6.85207816e+00, -5.49593341e+00]
std deviation: [6.12891058e-07, 5.98095154e-02, 9.26199195e-08, 1.86132695e-01,
4.81062255e-01, 3.73735861e-01, 3.06505660e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:23 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -7.011612237721095e-03 1.0e+00 1.68e-01  1e-01  2e-01 0:00.0
    2     50 -7.518549286287819e-03 1.5e+00 2.13e-01  2e-01  3e-01 0:00.0
    3     75 -7.550527561762524e-03 1.6e+00 2.50e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=86):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 2.4e+02
  100   2500 -7.613497774104766e-03 2.1e+01 3.02e-01  2e-05  7e-01 0:00.5
  150   3750 -7.613501844664640e-03 1.7e+03 1.04e-01  5e-08  4e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:24 2025)
final/bestever f-value = -7.613502e-03 -7.613502e-03 after 3751/3700 evaluation
s
incumbent solution: [ 5.53177788e-07, -4.46787923e+00, -4.89718075e-10, 9.19071
939e+00, -1.14293267e+01, -3.23081530e-01, -5.94213228e+00]
std deviation: [5.53760396e-07, 1.50440127e-01, 5.33808474e-08, 4.40777802e-01,
3.71063845e-01, 2.67744440e-01, 3.98652814e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:24 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 4.557484594831114e-03 1.0e+00 1.65e-01  1e-01  2e-01 0:00.0
    2     50 2.659317534984327e-03 1.4e+00 2.04e-01  2e-01  2e-01 0:00.0
    3     75 2.722570131250868e-03 1.7e+00 2.42e-01  2e-01  3e-01 0:00.0
   28    700 -0.000000000000000e+00 2.9e+01 1.53e+00  2e-01  3e+00 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 17:29:24 2025)
final/bestever f-value = -0.000000e+00 -0.000000e+00 after 701/414 evaluations
incumbent solution: [-0.83047049, 1.8505945, -0.07316134, -2.67192297, -1.50818
791, 4.12958252, -1.96051021]
std deviation: [0.66704301, 1.68314367, 0.20281353, 2.72848975, 1.36433318, 2.9
0765999, 2.98532626]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:24 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -6.830567971840822e-04 1.0e+00 1.84e-01  2e-01  2e-01 0:00.0
    2     50 -5.146100670839772e-03 1.5e+00 2.02e-01  2e-01  2e-01 0:00.0
    3     75 -5.213963680742954e-03 1.6e+00 2.38e-01  2e-01  3e-01 0:00.0
/usr/local/lib/python3.11/dist-packages/numpy/lib/_function_base_impl.py:1452:
RuntimeWarning: invalid value encountered in subtract
  a = op(a[slice1], a[slice2])
```

```
NOTE (module=cma, iteration=90):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.2e+08 to 4.3e+01
  100   2500 -5.465178702073414e-03 1.1e+01 1.34e-01  1e-05  6e-01 0:00.4
  150   3750 -5.465183068844795e-03 1.4e+03 5.89e-02  3e-08  3e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:25 2025)
final/bestever f-value = -5.465183e-03 -5.497239e-03 after 3751/200 evaluations
incumbent solution: [ 2.76371447e-07, -1.34907459e+00, -7.73413495e-09, 2.56178
250e+00, -9.73920655e+00, -4.94842554e+00, -9.10426366e+00]
std deviation: [2.44308498e-07, 5.22513104e-02, 3.24734489e-08, 8.07831253e-02,
1.56025005e-01, 2.20507358e-01, 3.19791295e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:25 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -4.692297712348501e-05 1.0e+00 1.79e-01  2e-01  2e-01 0:00.0
    2     50 -2.074957504047670e-03 1.4e+00 2.12e-01  2e-01  3e-01 0:00.0
    3     75 -2.039310873834525e-03 1.6e+00 2.42e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=87):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 2.4e+08 to 3.3e+02
  100   2500 -2.261057463812447e-03 1.7e+01 1.76e-01  1e-05  6e-01 0:00.4
  150   3750 -2.261064744656786e-03 7.8e+02 1.82e-01  1e-07  7e-01 0:00.6
termination on maxiter=150 (Tue Jul 22 17:29:26 2025)
final/bestever f-value = -2.261065e-03 -2.261065e-03 after 3751/3750 evaluation
s
incumbent solution: [ 6.34299353e-07, 1.83242893e+00, -1.77827593e-08, -6.46901
705e+00, -1.53026495e+01, -8.32073656e+00, -2.68120180e+00]
std deviation: [7.08896204e-07, 1.86131064e-01, 1.42176119e-07, 4.26217390e-01,
6.61945052e-01, 4.03437241e-01, 2.27903560e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:26 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -1.437828961315717e-03 1.0e+00 1.72e-01  1e-01  2e-01 0:00.0
    2     50 -1.333150781140008e-03 1.5e+00 1.74e-01  1e-01  2e-01 0:00.0
    3     75 -1.495397460701533e-03 1.6e+00 2.04e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=93):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.5e+08 to 5.1e+01
  100   2500 -1.672137534699789e-03 8.8e+00 1.82e-01  2e-05  5e-01 0:00.4
  150   3750 -1.672141490761000e-03 6.5e+02 1.28e-01  9e-08  6e-01 0:00.6
termination on maxiter=150 (Tue Jul 22 17:29:27 2025)
final/bestever f-value = -1.672141e-03 -1.672141e-03 after 3751/3617 evaluation
s
incumbent solution: [ 8.44874904e-07, -2.16173011e+00, -3.39239669e-09, 3.89199
637e+00, -1.60260122e+01, 1.57654552e+00, -2.66635046e+00]
std deviation: [7.41314726e-07, 1.02473510e-01, 8.87291777e-08, 2.24712205e-01,
5.72810705e-01, 2.11399951e-01, 1.24761542e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:27 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -6.576820135732578e-03 1.0e+00 1.78e-01  2e-01  2e-01 0:00.0
    2     50 -7.139216297365022e-03 1.5e+00 2.00e-01  2e-01  2e-01 0:00.0
    3     75 -7.269919253921724e-03 1.6e+00 2.56e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=88):
```

```
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.0e+08 to 6.4e+01
  100   2500 -7.378593995790033e-03 1.4e+01 1.37e-01  1e-05  5e-01 0:00.4
  150   3750 -7.378598371747883e-03 1.3e+03 5.62e-02  6e-08  3e-01 0:00.6
termination on maxiter=150 (Tue Jul 22 17:29:28 2025)
final/bestever f-value = -7.378598e-03 -7.378598e-03 after 3751/3745 evaluation
s
incumbent solution: [ 2.21820298e-07, 1.01724204e+00, 1.23985602e-08, -3.291418
38e+00, -1.12081811e+01, -1.59068732e+01, -5.54013687e+00]
std deviation: [2.68755609e-07, 5.44059558e-02, 5.79441742e-08, 6.08635086e-02,
8.22357715e-02, 2.56943210e-01, 8.50579841e-02]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:28 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -2.413794988232786e-03 1.0e+00 1.77e-01  2e-01  2e-01 0:00.0
    2     50 -5.405128678772939e-03 1.4e+00 2.08e-01  2e-01  2e-01 0:00.0
    3     75 -5.451933495819256e-03 1.6e+00 2.41e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=97):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.8e+08 to 3.0e+01
  100   2500 -5.613488473932896e-03 5.6e+00 2.97e-01  3e-05  5e-01 0:00.4
  150   3750 -5.613501763243031e-03 3.8e+02 2.31e-01  3e-07  5e-01 0:00.6
termination on maxiter=150 (Tue Jul 22 17:29:29 2025)
final/bestever f-value = -5.613502e-03 -5.613502e-03 after 3751/3720 evaluation
s
incumbent solution: [ 1.53978410e-06, -4.94871340e-01, 8.42061636e-09, 5.845655
89e-01, -1.45013093e+01, -5.79048613e-01, -2.15358940e+00]
std deviation: [1.74832552e-06, 1.35393677e-01, 2.66000998e-07, 3.39367441e-01,
5.02877791e-01, 2.63543485e-01, 3.37191084e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:29 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 7.136965226810571e-03 1.0e+00 1.64e-01  1e-01  2e-01 0:00.0
    2     50 4.895104999140996e-03 1.4e+00 1.94e-01  2e-01  2e-01 0:00.0
    3     75 4.893771665774153e-03 1.6e+00 2.16e-01  2e-01  3e-01 0:00.0
   12    300 -0.000000000000000e+00 3.1e+00 1.73e+00  1e+00  2e+00 0:00.1
termination on tolflatfitness=1 (Tue Jul 22 17:29:29 2025)
final/bestever f-value = -0.000000e+00 -0.000000e+00 after 301/166 evaluations
incumbent solution: [-1.84862673, 1.08815517, -1.63072812, 1.30871095, -4.54777
802, 2.46785746, 0.81213312]
std deviation: [1.62385969, 1.42340696, 1.34721701, 1.89876145, 1.8314069, 1.73
849437, 2.00450016]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:29 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 1.263106343464298e-02 1.0e+00 1.76e-01  2e-01  2e-01 0:00.0
    2     50 -2.965448736032520e-03 1.4e+00 1.91e-01  2e-01  2e-01 0:00.0
    3     75 -1.678374894984130e-03 1.5e+00 1.83e-01  2e-01  2e-01 0:00.0
/tmp/ipython-input-3-1096312254.py:134: RuntimeWarning: overflow encountered in
square
  volatility = np.sqrt(np.mean(np.diff(signal)**2))
```

```
NOTE (module=cma, iteration=87):
condition in coordinate system exceeded 1.4e+08, rescaled to 1.0e+00,
condition changed from 1.8e+08 to 8.3e+01
  100   2500 -3.465172219152100e-03 7.6e+00 2.53e-01  1e-05  7e-01 0:00.4
  150   3750 -3.465183049279102e-03 1.4e+03 8.06e-02  4e-08  3e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:30 2025)
final/bestever f-value = -3.465183e-03 -3.465183e-03 after 3751/3738 evaluation
s
incumbent solution: [ 3.25853562e-07, -4.75591524e+00, -3.24496297e-09, 1.00109
284e+01, -4.30836859e+00, -5.04512326e+00, -1.41296678e+01]
std deviation: [3.09338147e-07, 8.65557339e-02, 3.59806545e-08, 2.15308712e-01,
1.89696347e-01, 1.55526627e-01, 2.75522525e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:30 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 2.762596062547140e-04 1.0e+00 1.71e-01  2e-01  2e-01 0:00.0
    2     50 1.663448048295785e-04 1.5e+00 1.86e-01  2e-01  2e-01 0:00.0
    3     75 7.939571683639353e-05 1.5e+00 2.33e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=83):
condition in coordinate system exceeded 1.2e+08, rescaled to 1.0e+00,
condition changed from 1.4e+08 to 4.9e+01
  100   2500 -2.610585223339337e-04 1.6e+01 1.36e-01  9e-06  6e-01 0:00.4
  150   3750 -2.610647179035405e-04 1.9e+03 6.71e-02  4e-08  4e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:30 2025)
final/bestever f-value = -2.610647e-04 -2.610647e-04 after 3751/3748 evaluation
s
incumbent solution: [ 2.86436012e-07, -2.03533037e+00, -1.23832048e-08, 4.07712
337e+00, -1.77556587e+01, -9.37870098e+00, 1.35321668e-02]
std deviation: [3.10960818e-07, 4.96860625e-02, 4.41364187e-08, 1.01432938e-01,
3.97218541e-01, 2.47395733e-01, 1.15448947e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:30 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 7.823244741587167e-04 1.0e+00 1.72e-01  1e-01  2e-01 0:00.0
    2     50 8.044561198268033e-04 1.5e+00 1.74e-01  1e-01  2e-01 0:00.0
    3     75 8.015069572387197e-04 1.7e+00 1.76e-01  1e-01  2e-01 0:00.0
   57   1425 -0.000000000000000e+00 1.9e+02 2.15e+00  2e-02  4e+00 0:00.2
termination on tolflatfitness=1 (Tue Jul 22 17:29:31 2025)
final/bestever f-value = -0.000000e+00 -0.000000e+00 after 1426/1179 evaluation
s
incumbent solution: [-1.45689791e-01, 1.80570558e+00, -7.13384018e-03, 1.790112
40e+00, -8.28277462e+00, 4.75903705e+00, -9.22210322e-01]
std deviation: [0.17265872, 0.89413, 0.02368369, 1.52333982, 2.81016607, 2.3995
3918, 3.70470382]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:31 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -4.296957417265065e-03 1.0e+00 1.79e-01  2e-01  2e-01 0:00.0
    2     50 -4.932187397017430e-03 1.5e+00 2.09e-01  2e-01  2e-01 0:00.0
    3     75 -5.051247064739375e-03 1.6e+00 2.54e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=90):
condition in coordinate system exceeded 1.1e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 4.6e+01
  100   2500 -5.378593140840021e-03 8.2e+00 1.81e-01  2e-05  5e-01 0:00.4
```

```
  150   3750 -5.378598376460380e-03 7.4e+02 8.13e-02  5e-08  3e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:32 2025)
final/bestever f-value = -5.378598e-03 -5.378598e-03 after 3751/3729 evaluation
s
incumbent solution: [ 3.49108933e-07, 1.00855546e+00, -3.39619019e-09, -6.33107
634e+00, -5.65875669e+00, -2.52533167e-01, -1.23538684e+01]
std deviation: [5.02754420e-07, 4.21085176e-02, 5.43192038e-08, 1.43837178e-01,
1.45332978e-01, 1.14254777e-01, 2.87730039e-01]
(12_w,25)-aCMA-ES (mu_w=7.3,w_1=23%) in dimension 7 (seed=42, Tue Jul 22 17:2
9:32 2025)
Iterat #Fevals   function value  axis ratio  sigma  min&max std  t[m:s]
    1     25 -2.704633042538648e-03 1.0e+00 1.80e-01  2e-01  2e-01 0:00.0
    2     50 -3.183939640905381e-03 1.5e+00 2.19e-01  2e-01  3e-01 0:00.0
    3     75 -3.287010720529792e-03 1.7e+00 2.74e-01  2e-01  3e-01 0:00.0
NOTE (module=cma, iteration=85):
condition in coordinate system exceeded 1.0e+08, rescaled to 1.0e+00,
condition changed from 1.3e+08 to 5.9e+01
  100   2500 -3.613491455096055e-03 1.1e+01 2.28e-01  1e-05  6e-01 0:00.4
  150   3750 -3.613501823668871e-03 1.4e+03 6.09e-02  4e-08  2e-01 0:00.7
termination on maxiter=150 (Tue Jul 22 17:29:32 2025)
final/bestever f-value = -3.613502e-03 -3.613502e-03 after 3751/3702 evaluation
s
incumbent solution: [ 2.50158003e-07, -5.54036632e-01, -7.92427231e-09, -2.6762
5570e-01, -1.27629491e+01, -2.46566306e+00, -5.40859136e+00]
std deviation: [2.67890222e-07, 5.05026349e-02, 3.64664657e-08, 6.56633978e-02,
1.96028817e-01, 9.45639331e-02, 1.48490680e-01]
```



Final Portfolio Values and Returns for Different Fee/Slippage Configurations:

Fee	Slippage	FP Strategy ($)	FP Return (%)	Buy & Hold ($)	Buy & Hold Return (%)	NPC ($)	NPC Return (%)
0.0000	0.00000	89.51	-10.49	93.45	-6.55	94.01	-5.99
0.0001	0.00002	64.12	-35.88	93.45	-6.55	69.75	-30.25
0.0002	0.00005	47.64	-52.36	93.45	-6.55	51.84	-48.16
0.0003	0.00010	31.76	-68.24	93.45	-6.55	34.00	-66.00

In [ ]:

```python
import pandas as pd
import numpy as np
from skopt import gp_minimize
from skopt.space import Real
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from typing import Tuple, List, Dict

# Configuration
class Config:
    RANDOM_SEED = 42
    TRAIN_RATIO = 0.6
    CV_RATIO = 0.2
    TEST_RATIO = 0.2
    INITIAL_CAPITAL = 100
    FEE_SLIPPAGE_COMBOS = [
        (0, 0),
        (0.0002, 0.00005),
        (0.0004, 0.0001),
        (0.0006, 0.0003)
    ]
    WINDOW_SIZE = 3
    N_MODEL_SEGMENTS = 5

np.random.seed(Config.RANDOM_SEED)

# Data Preparation
def prepare_data(filepath: str) -> Tuple[pd.DataFrame, pd.DataFrame, pd.DataFr
    """Load and preprocess the data"""
    df = pd.read_csv(filepath)

    # Calculate price levels
    for j in range(15):
        df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
        df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

    # Calculate features
    bid_cols = [f"bids_notional_{i}" for i in range(15)]
    ask_cols = [f"asks_notional_{i}" for i in range(15)]

    df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (
        df[bid_cols].sum(axis=1) + df[ask_cols].sum(axis=1) + 1e-8)
    df['dobi'] = df['obi'].diff().fillna(0)
    df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
    df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']
    df['spread'] = df['ask_price_0'] - df['bid_price_0']

    # Log returns
    df['log_mid'] = np.log(df['midpoint'])
    df['returns'] = df['log_mid'].diff().fillna(0)

    # Train/Validation/Test split
    train_end = int(len(df) * Config.TRAIN_RATIO)
```

```
    cv_end = int(len(df) * (Config.TRAIN_RATIO + Config.CV_RATIO))

    df_train = df.iloc[:train_end].copy().reset_index(drop=True)
    df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
    df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

    # Feature scaling
    scaler = StandardScaler()
    scale_cols = ['obi', 'depth', 'queue_slope', 'spread']
    df_train[scale_cols] = scaler.fit_transform(df_train[scale_cols])
    df_cv[scale_cols] = scaler.transform(df_cv[scale_cols])
    df_test[scale_cols] = scaler.transform(df_test[scale_cols])

    return df_train, df_cv, df_test

# Trading Strategy Components
def trading_strategy(signal: np.ndarray, threshold: float) -> Tuple[np.ndarray
    """Generate positions from trading signals"""
    positions = np.zeros_like(signal)
    positions[signal > threshold] = 1
    positions[signal < -threshold] = -1
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(
    positions: np.ndarray,
    trades: np.ndarray,
    returns: np.ndarray,
    fee: float,
    slip: float,
    trade_sizes: np.ndarray = None
) -> np.ndarray:
    """Calculate PnL with realistic trading costs"""
    raw_pnl = positions[:-1] * returns[1:len(positions)]

    # Dynamic slippage based on trade size and liquidity
    if trade_sizes is None:
        costs = np.abs(trades[1:len(positions)]) * (fee + slip)
    else:
        liquidity_impact = 0.0001 * (trade_sizes / 1e6)  # Assume liquidity in
        costs = np.abs(trades[1:len(positions)]) * (fee + slip + liquidity_imp

    return raw_pnl - costs

# Signal Generation Model
def simulate_fp(
    mu_params: List[float],
    sigma_params: List[float],
    x0: float,
    obi: np.ndarray,
    timesteps: int,
    dt: float = 1.0
) -> np.ndarray:
```

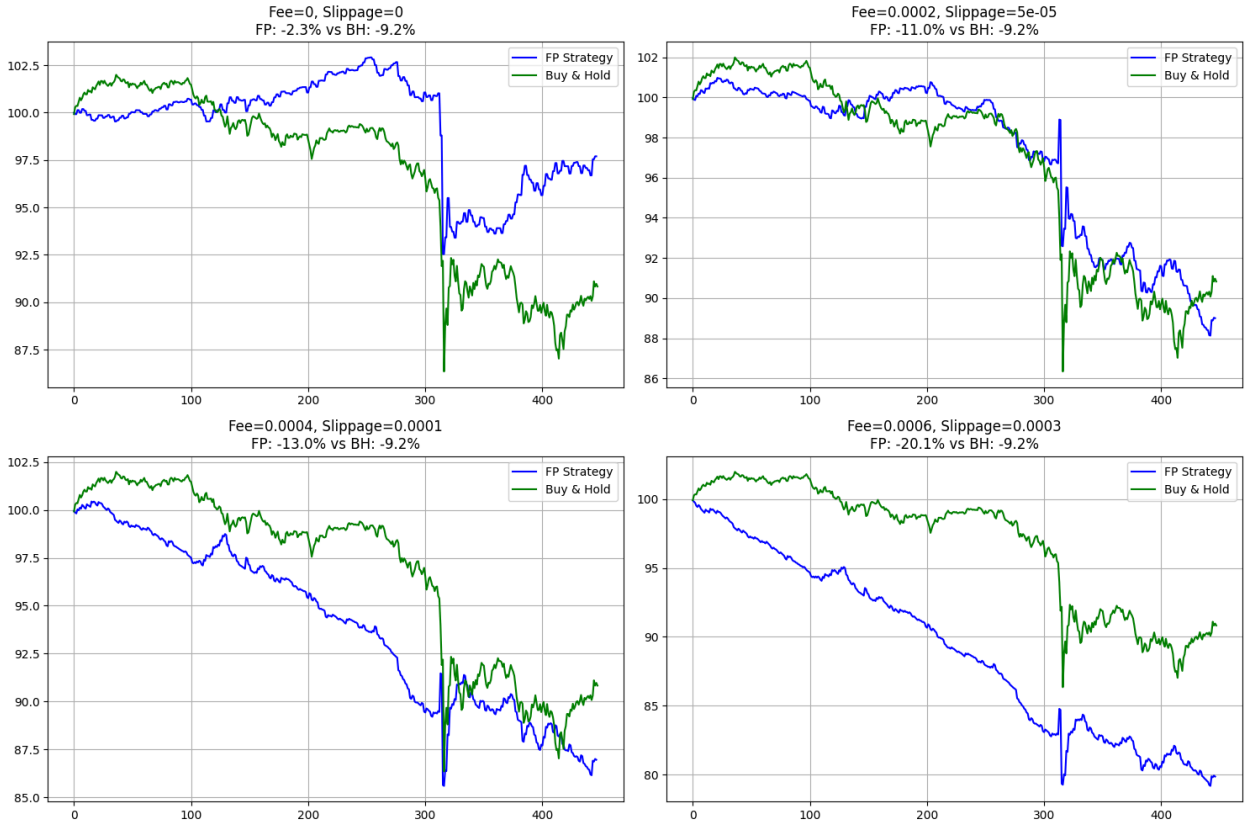
```
    """Fokker-Planck inspired signal generation"""
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params

    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(Config.RANDOM_SEED)

    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()

    return x

# Optimization
def optimize_threshold(
    signal: np.ndarray,
    returns: np.ndarray,
    fee: float,
    slip: float
) -> float:
    """Find optimal trading threshold"""
    thresholds = np.linspace(0.001, 0.01, 20)
    best_pnl = -np.inf
    best_thresh = 0.005

    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))

        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t

    return best_thresh

def train_fp_model(
    df_slice: pd.DataFrame,
    fee: float,
    slip: float
) -> Tuple[List[float], List[float]]:
    """Train model using Bayesian optimization"""
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0

    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
```

```
    space = [
        Real(-1, 1, name='a0'),
        Real(-1, 1, name='a1'),
        Real(-1, 1, name='a2'),
        Real(0.0001, 0.1, name='b0'),
        Real(0.0001, 0.1, name='b1')
    ]

    res = gp_minimize(objective, space, n_calls=50, random_state=Config.RANDOM
    return res.x[:3], res.x[3:]

# Backtest Framework
def run_backtest(
    df_train: pd.DataFrame,
    df_cv: pd.DataFrame,
    df_test: pd.DataFrame,
    fee: float,
    slip: float
) -> Dict:
    """Complete backtest pipeline for one fee/slippage combo"""
    # 1. Train multiple models on different segments
    segment_size = len(df_train) // Config.N_MODEL_SEGMENTS
    segment_models = []
    segment_thresholds = []

    for i in range(Config.N_MODEL_SEGMENTS):
        start = i * segment_size
        end = (i + 1) * segment_size
        if end > len(df_train):
            continue

        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0,
                             df_train.iloc[start:end]['obi'].values,
                             end - start)
        threshold = optimize_threshold(signal,
                                       df_train.iloc[start:end]['returns'].value
                                       fee, slip)
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    # 2. Model selection on CV data
    selected_models = []
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values

    for start in range(0, len(cv_returns) - Config.WINDOW_SIZE, Config.WINDOW_
        end = start + Config.WINDOW_SIZE
        best_pnl = -np.inf
        best_index = 0

        for i, (mu_p, sigma_p) in enumerate(segment_models):
```

```
            signal = simulate_fp(mu_p, sigma_p, 0.0,
                                 cv_obi[start:end],
                                 Config.WINDOW_SIZE)
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades,
                                             cv_returns[start:end],
                                             fee, slip))
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i

        selected_models.append(best_index)

    # 3. Test on out-of-sample data
    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []

    for i, start in enumerate(range(0, len(test_returns) - Config.WINDOW_SIZE
        end = start + Config.WINDOW_SIZE
        model_idx = selected_models[min(i, len(selected_models) - 1)]
        mu_p, sigma_p = segment_models[model_idx]
        threshold = segment_thresholds[model_idx]

        signal = simulate_fp(mu_p, sigma_p, 0.0,
                             test_obi[start:end],
                             min(Config.WINDOW_SIZE, len(test_returns) - start))
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    # Combine results
    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    # Calculate PnLs
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = Config.INITIAL_CAPITAL * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = Config.INITIAL_CAPITAL * np.exp(np.cumsum(bh_returns))

    # Calculate metrics
    def calculate_metrics(returns):
        total_return = (np.exp(np.sum(returns)) - 1) * 100
        sharpe = np.mean(returns) / np.std(returns) * np.sqrt(365*24*12)  # 5m
```

```
        max_drawdown = (np.exp(np.min(returns.cumsum())) - 1) * 100
        return total_return, sharpe, max_drawdown

    fp_metrics = calculate_metrics(fp_net_returns)
    bh_metrics = calculate_metrics(bh_returns)

    return {
        'fee': fee,
        'slippage': slip,
        'fp_pnl': fp_pnl,
        'bh_pnl': bh_pnl,
        'fp_return_pct': fp_metrics[0],
        'fp_sharpe': fp_metrics[1],
        'fp_drawdown_pct': fp_metrics[2],
        'bh_return_pct': bh_metrics[0],
        'bh_sharpe': bh_metrics[1],
        'bh_drawdown_pct': bh_metrics[2]
    }

# Main Execution
if __name__ == "__main__":
    # Load and prepare data
    df_train, df_cv, df_test = prepare_data("BTC_5min.csv")

    # Run backtests for all fee/slippage combinations
    results = []
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))
    axes = axes.flatten()

    for idx, (fee, slip) in enumerate(Config.FEE_SLIPPAGE_COMBOS):
        result = run_backtest(df_train, df_cv, df_test, fee, slip)
        results.append(result)

        # Plotting
        ax = axes[idx]
        ax.plot(result['fp_pnl'], label='FP Strategy', color='blue')
        ax.plot(result['bh_pnl'], label='Buy & Hold', color='green')
        ax.set_title(f"Fee={fee}, Slippage={slip}\n"
                     f"FP: {result['fp_return_pct']:.1f}% vs BH: {result['bh_re
        ax.grid(True)
        ax.legend()

    plt.tight_layout()
    plt.show()

    # Results table
    results_df = pd.DataFrame([{
        'Fee': r['fee'],
        'Slippage': r['slippage'],
        'FP Return (%)': r['fp_return_pct'],
        'FP Sharpe': r['fp_sharpe'],
        'FP Drawdown (%)': r['fp_drawdown_pct'],
        'BH Return (%)': r['bh_return_pct'],
```

```
        'BH Sharpe': r['bh_sharpe'],
        'BH Drawdown (%)': r['bh_drawdown_pct']
    } for r in results])

    print("\nPerformance Metrics Across Different Cost Scenarios:")
    print(results_df.to_string(index=False, float_format="%.2f"))
```




```
Performance Metrics Across Different Cost Scenarios:
 Fee  Slippage  FP Return (%)  FP Sharpe  FP Drawdown (%)  BH Return (%)  BH Sh
arpe  BH Drawdown (%)
0.00      0.00          -2.31      -4.22            -7.46          -9.17      -1
4.53          -13.65
0.00      0.00         -11.00     -21.12           -11.87          -9.17      -1
4.53          -13.65
0.00      0.00         -13.05     -25.29           -14.41          -9.17      -1
4.53          -13.65
0.00      0.00         -20.15     -40.55           -20.83          -9.17      -1
4.53          -13.65
```

In [ ]:

```
import pandas as pd
import numpy as np
from skopt import gp_minimize
from skopt.space import Real
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from typing import Tuple, List, Dict

# Configuration
class Config:
```

```
    RANDOM_SEED = 42
    TRAIN_RATIO = 0.6
    CV_RATIO = 0.2
    TEST_RATIO = 0.2
    INITIAL_CAPITAL = 100
    FEE_SLIPPAGE_COMBOS = [
        (0, 0),
        (0.0002, 0.00005),
        (0.0004, 0.0001),
        (0.0006, 0.0003)
    ]
    WINDOW_SIZE = 3
    N_MODEL_SEGMENTS = 5

np.random.seed(Config.RANDOM_SEED)

# Data Preparation
def prepare_data(filepath: str) -> Tuple[pd.DataFrame, pd.DataFrame, pd.DataFr
    """Load and preprocess the data"""
    df = pd.read_csv(filepath)

    # Calculate price levels
    for j in range(15):
        df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
        df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

    # Calculate features
    bid_cols = [f"bids_notional_{i}" for i in range(15)]
    ask_cols = [f"asks_notional_{i}" for i in range(15)]

    df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (
        df[bid_cols].sum(axis=1) + df[ask_cols].sum(axis=1) + 1e-8)
    df['dobi'] = df['obi'].diff().fillna(0)
    df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
    df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']
    df['spread'] = df['ask_price_0'] - df['bid_price_0']

    # Log returns
    df['log_mid'] = np.log(df['midpoint'])
    df['returns'] = df['log_mid'].diff().fillna(0)

    # Train/Validation/Test split
    train_end = int(len(df) * Config.TRAIN_RATIO)
    cv_end = int(len(df) * (Config.TRAIN_RATIO + Config.CV_RATIO))

    df_train = df.iloc[:train_end].copy().reset_index(drop=True)
    df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
    df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

    # Feature scaling
    scaler = StandardScaler()
    scale_cols = ['obi', 'depth', 'queue_slope', 'spread']
    df_train[scale_cols] = scaler.fit_transform(df_train[scale_cols])
```

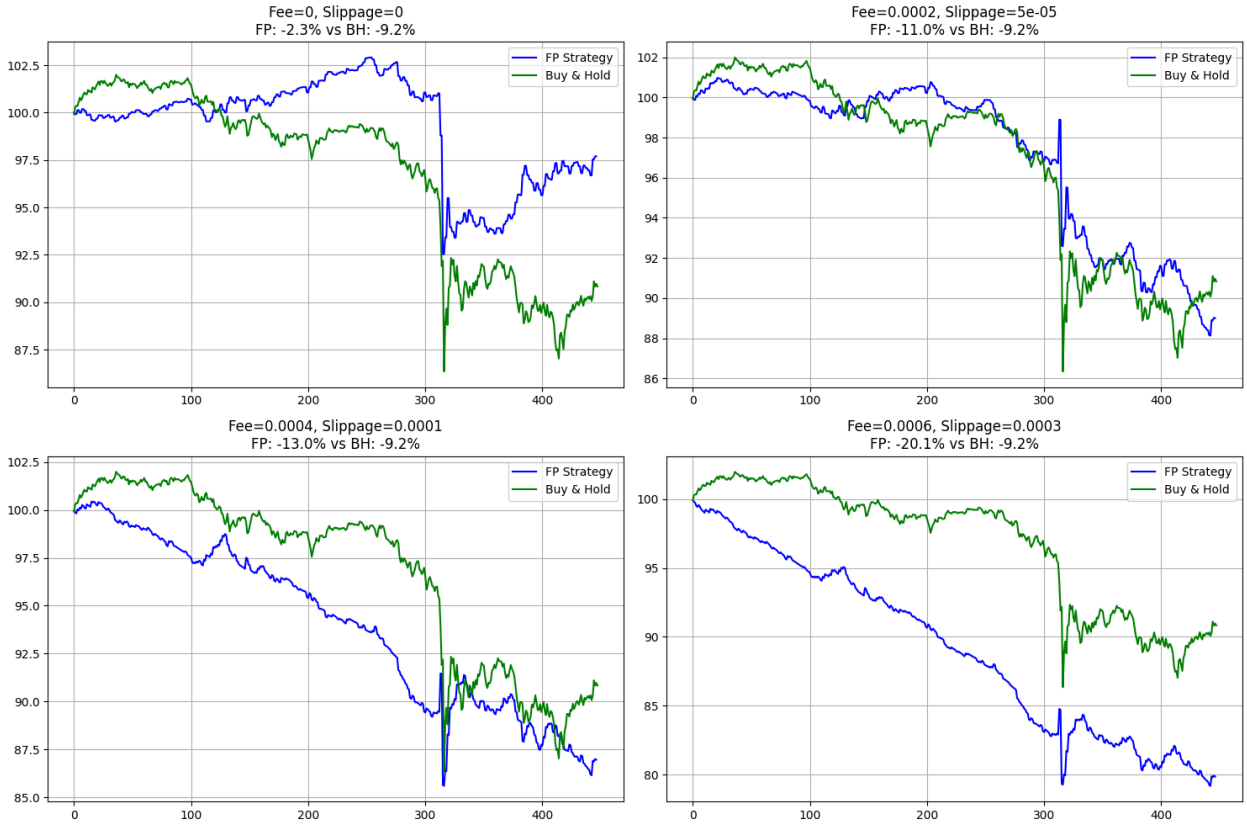
```python
    df_cv[scale_cols] = scaler.transform(df_cv[scale_cols])
    df_test[scale_cols] = scaler.transform(df_test[scale_cols])

    return df_train, df_cv, df_test

# Trading Strategy Components
def trading_strategy(signal: np.ndarray, threshold: float) -> Tuple[np.ndarray
    """Generate positions from trading signals"""
    positions = np.zeros_like(signal)
    positions[signal > threshold] = 1
    positions[signal < -threshold] = -1
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(
    positions: np.ndarray,
    trades: np.ndarray,
    returns: np.ndarray,
    fee: float,
    slip: float,
    trade_sizes: np.ndarray = None
) -> np.ndarray:
    """Calculate PnL with realistic trading costs"""
    raw_pnl = positions[:-1] * returns[1:len(positions)]

    # Dynamic slippage based on trade size and liquidity
    if trade_sizes is None:
        costs = np.abs(trades[1:len(positions)]) * (fee + slip)
    else:
        liquidity_impact = 0.0001 * (trade_sizes / 1e6)  # Assume liquidity ir
        costs = np.abs(trades[1:len(positions)]) * (fee + slip + liquidity_imp

    return raw_pnl - costs

# Signal Generation Model
def simulate_fp(
    mu_params: List[float],
    sigma_params: List[float],
    x0: float,
    obi: np.ndarray,
    timesteps: int,
    dt: float = 1.0
) -> np.ndarray:
    """Fokker-Planck inspired signal generation"""
    a0, a1, a2 = mu_params
    b0, b1 = sigma_params

    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(Config.RANDOM_SEED)

    for t in range(1, timesteps):
        mu = a0 + a1 * x[t-1] + a2 * obi[t-1]
```

```
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()

    return x

# Optimization
def optimize_threshold(
    signal: np.ndarray,
    returns: np.ndarray,
    fee: float,
    slip: float
) -> float:
    """Find optimal trading threshold"""
    thresholds = np.linspace(0.001, 0.01, 20)
    best_pnl = -np.inf
    best_thresh = 0.005

    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))

        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t

    return best_thresh

def train_fp_model(
    df_slice: pd.DataFrame,
    fee: float,
    slip: float
) -> Tuple[List[float], List[float]]:
    """Train model using Bayesian optimization"""
    returns = df_slice['returns'].values
    obi = df_slice['obi'].values
    x_init = 0.0

    def objective(params):
        mu_params = params[:3]
        sigma_params = params[3:]
        signal = simulate_fp(mu_params, sigma_params, x_init, obi, len(returns
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))

    space = [
        Real(-1, 1, name='a0'),
        Real(-1, 1, name='a1'),
        Real(-1, 1, name='a2'),
        Real(0.0001, 0.1, name='b0'),
        Real(0.0001, 0.1, name='b1')
    ]

    res = gp_minimize(objective, space, n_calls=50, random_state=Config.RANDOM
```

```
    return res.x[:3], res.x[3:]

# Backtest Framework
def run_backtest(
    df_train: pd.DataFrame,
    df_cv: pd.DataFrame,
    df_test: pd.DataFrame,
    fee: float,
    slip: float
) -> Dict:
    """Complete backtest pipeline for one fee/slippage combo"""
    # 1. Train multiple models on different segments
    segment_size = len(df_train) // Config.N_MODEL_SEGMENTS
    segment_models = []
    segment_thresholds = []

    for i in range(Config.N_MODEL_SEGMENTS):
        start = i * segment_size
        end = (i + 1) * segment_size
        if end > len(df_train):
            continue

        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0,
                             df_train.iloc[start:end]['obi'].values,
                             end - start)
        threshold = optimize_threshold(signal,
                                       df_train.iloc[start:end]['returns'].value
                                       fee, slip)
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    # 2. Model selection on CV data
    selected_models = []
    cv_returns = df_cv['returns'].values
    cv_obi = df_cv['obi'].values

    for start in range(0, len(cv_returns) - Config.WINDOW_SIZE, Config.WINDOW_
        end = start + Config.WINDOW_SIZE
        best_pnl = -np.inf
        best_index = 0

        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0,
                                 cv_obi[start:end],
                                 Config.WINDOW_SIZE)
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades,
                                             cv_returns[start:end],
                                             fee, slip))
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
```

```
        selected_models.append(best_index)

    # 3. Test on out-of-sample data
    test_returns = df_test['returns'].values
    test_obi = df_test['obi'].values
    test_positions = []
    test_trades = []

    for i, start in enumerate(range(0, len(test_returns) - Config.WINDOW_SIZE
        end = start + Config.WINDOW_SIZE
        model_idx = selected_models[min(i, len(selected_models) - 1)]
        mu_p, sigma_p = segment_models[model_idx]
        threshold = segment_thresholds[model_idx]

        signal = simulate_fp(mu_p, sigma_p, 0.0,
                             test_obi[start:end],
                             min(Config.WINDOW_SIZE, len(test_returns) - start))
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    # Combine results
    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    # Calculate PnLs
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = Config.INITIAL_CAPITAL * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = Config.INITIAL_CAPITAL * np.exp(np.cumsum(bh_returns))

    # Calculate metrics
    def calculate_metrics(returns):
        total_return = (np.exp(np.sum(returns)) - 1) * 100
        sharpe = np.mean(returns) / np.std(returns) * np.sqrt(365*24*12)  # 5m
        max_drawdown = (np.exp(np.min(returns.cumsum())) - 1) * 100
        return total_return, sharpe, max_drawdown

    fp_metrics = calculate_metrics(fp_net_returns)
    bh_metrics = calculate_metrics(bh_returns)

    return {
        'fee': fee,
        'slippage': slip,
        'fp_pnl': fp_pnl,
```

```
        'bh_pnl': bh_pnl,
        'fp_return_pct': fp_metrics[0],
        'fp_sharpe': fp_metrics[1],
        'fp_drawdown_pct': fp_metrics[2],
        'bh_return_pct': bh_metrics[0],
        'bh_sharpe': bh_metrics[1],
        'bh_drawdown_pct': bh_metrics[2]
    }

# Main Execution
if __name__ == "__main__":
    # Load and prepare data
    df_train, df_cv, df_test = prepare_data("BTC_5min.csv")

    # Run backtests for all fee/slippage combinations
    results = []
    fig, axes = plt.subplots(2, 2, figsize=(15, 10))
    axes = axes.flatten()

    for idx, (fee, slip) in enumerate(Config.FEE_SLIPPAGE_COMBOS):
        result = run_backtest(df_train, df_cv, df_test, fee, slip)
        results.append(result)

        # Plotting
        ax = axes[idx]
        ax.plot(result['fp_pnl'], label='FP Strategy', color='blue')
        ax.plot(result['bh_pnl'], label='Buy & Hold', color='green')
        ax.set_title(f"Fee={fee}, Slippage={slip}\n"
                     f"FP: {result['fp_return_pct']:.1f}% vs BH: {result['bh_re
        ax.grid(True)
        ax.legend()

    plt.tight_layout()
    plt.show()

    # Results table
    results_df = pd.DataFrame([{
        'Fee': r['fee'],
        'Slippage': r['slippage'],
        'FP Return (%)': r['fp_return_pct'],
        'FP Sharpe': r['fp_sharpe'],
        'FP Drawdown (%)': r['fp_drawdown_pct'],
        'BH Return (%)': r['bh_return_pct'],
        'BH Sharpe': r['bh_sharpe'],
        'BH Drawdown (%)': r['bh_drawdown_pct']
    } for r in results])

    print("\nPerformance Metrics Across Different Cost Scenarios:")
    print(results_df.to_string(index=False, float_format="%.2f"))
```



Performance Metrics Across Different Cost Scenarios:

Fee	Slippage	FP Return (%)	FP Sharpe	FP Drawdown (%)	BH Return (%)	BH Sharpe	BH Drawdown (%)
0.00	0.00	-2.31	-4.22	-7.46	-9.17	-14.53	-13.65
0.00	0.00	-11.00	-21.12	-11.87	-9.17	-14.53	-13.65
0.00	0.00	-13.05	-25.29	-14.41	-9.17	-14.53	-13.65
0.00	0.00	-20.15	-40.55	-20.83	-9.17	-14.53	-13.65

In [ ]:
```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("BTC_5min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
```

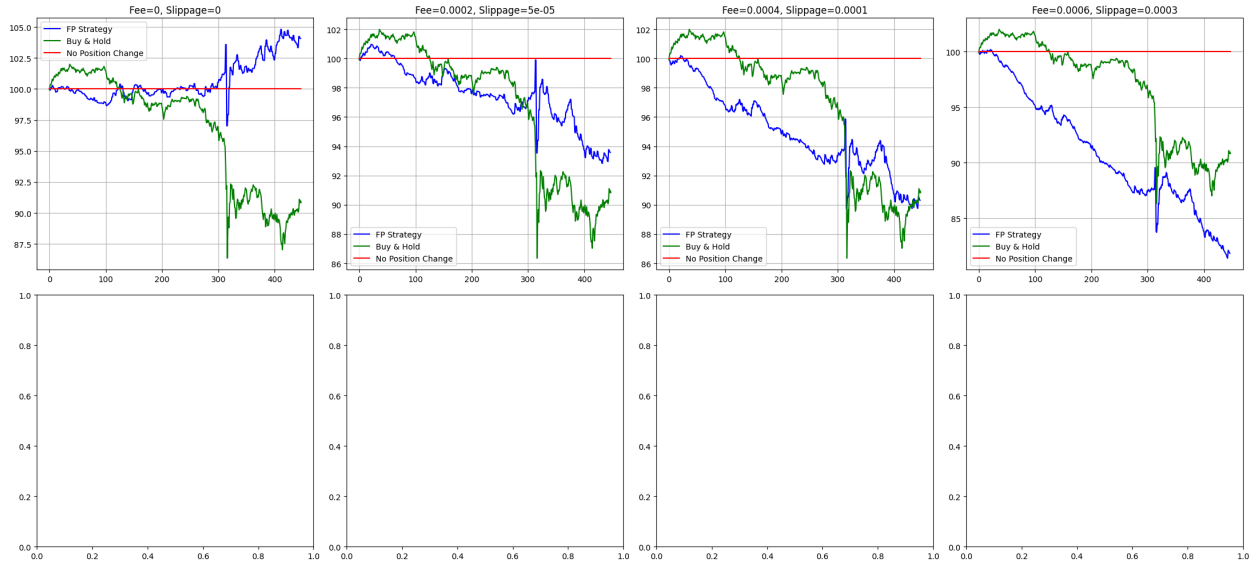
```
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        obi_t = features['obi'].iloc[t-1]
        dobi_t = features['dobi'].iloc[t-1]
        depth_t = features['depth'].iloc[t-1]
        slope_t = features['queue_slope'].iloc[t-1]
        spread_t = features['spread'].iloc[t-1]
        mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
```

```
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'queue_slope', 'spread']]
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:7]
        sigma_params = params[7:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*7 + [0.005, 0.005], sigma0=0.2, options={'seed':
    return res[0][:7], res[0][7:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+200) for i in range(0, len(df_train)-200, 200)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'queue_slope', 'spread']]
```

```
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
```

```
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-3-310034494.py:21: FutureWarning: Series.fillna with 'metho
d' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfi
ll() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000           104.07           4.07           90.83
-9.17    100.0             0.0
0.0002   0.00005            93.58          -6.42           90.83
-9.17    100.0             0.0
0.0004   0.00010            90.31          -9.69           90.83
-9.17    100.0             0.0
0.0006   0.00030            81.85         -18.15           90.83
-9.17    100.0             0.0
```

In [ ]:

```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("BTC_1min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']
```

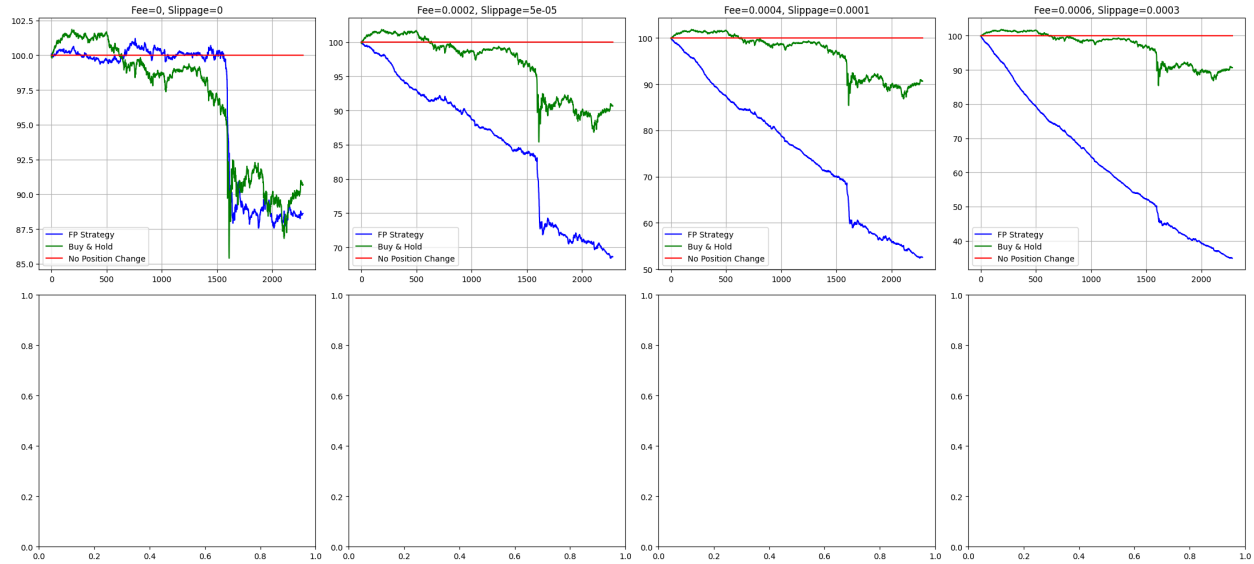
```
bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope'] = df['bids_notional_0'] - df['bids_notional_5']
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        obi_t = features['obi'].iloc[t-1]
        dobi_t = features['dobi'].iloc[t-1]
        depth_t = features['depth'].iloc[t-1]
        slope_t = features['queue_slope'].iloc[t-1]
        spread_t = features['spread'].iloc[t-1]
        mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
```

```
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'queue_slope', 'spread']]
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:7]
        sigma_params = params[7:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*7 + [0.005, 0.005], sigma0=0.2, options={'seed':
    return res[0][:7], res[0][7:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+500) for i in range(0, len(df_train)-500, 500)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
```

```
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'queue_slope', 'spread']]
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
```

```
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-7-3049133861.py:21: FutureWarning: Series.fillna with 'metho
d' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfi
ll() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
/tmp/ipython-input-7-3049133861.py:58: RuntimeWarning: overflow encountered in
scalar multiply
  mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5 * slop
e_t + a6 * spread_t)
/tmp/ipython-input-7-3049133861.py:59: RuntimeWarning: overflow encountered in
scalar multiply
  sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
/tmp/ipython-input-7-3049133861.py:60: RuntimeWarning: invalid value encountere
d in scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-3049133861.py:60: RuntimeWarning: overflow encountered in
scalar add
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
/tmp/ipython-input-7-3049133861.py:60: RuntimeWarning: overflow encountered in
scalar multiply
  x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
```



```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000            88.61         -11.39           90.67
-9.33    100.0             0.0
0.0002   0.00005            68.62         -31.38           90.67
-9.33    100.0             0.0
0.0004   0.00010            52.57         -47.43           90.67
-9.33    100.0             0.0
0.0006   0.00030            34.87         -65.13           90.67
-9.33    100.0             0.0
```

In [ ]:

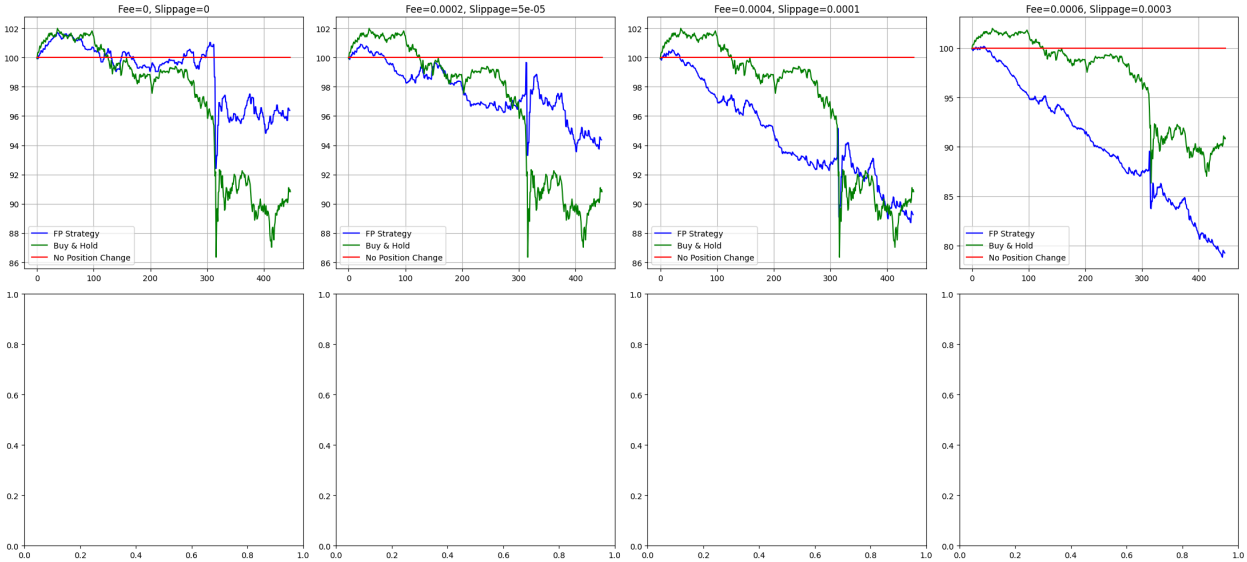
```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("BTC_5min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]

df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope_bid'] = df['bids_notional_0'] - df['bids_notional_5']
df['queue_slope_ask'] = df['asks_notional_0'] - df['asks_notional_5']
df['net_queue_slope'] = df['queue_slope_bid'] - df['queue_slope_ask']
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
df['depth_variance'] = df[bid_cols + ask_cols].std(axis=1)
df['abs_dobi'] = df['dobi'].abs()

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades
```

```
def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6, a7, a8, a9 = mu_params
    b0, b1, b2 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        obi = features['obi'].iloc[t-1]
        dobi = features['dobi'].iloc[t-1]
        depth = features['depth'].iloc[t-1]
        net_slope = features['net_queue_slope'].iloc[t-1]
        spread = features['spread'].iloc[t-1]
        depth_var = features['depth_variance'].iloc[t-1]
        abs_dobi = features['abs_dobi'].iloc[t-1]
        mu = (a0 + a1 * x[t-1] + a2 * obi + a3 * dobi + a4 * depth + a5 * net_
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]) + b2 * spread)
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'net_queue_slope', 'spread',
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:10]
        sigma_params = params[10:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*10 + [0.005, 0.005, 0.005], sigma0=0.2, options=
    return res[0][:10], res[0][10:]
```

```
fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+200) for i in range(0, len(df_train)-200, 200)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'net_queue_slope', 'sprea
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]
```

```
    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-2-3620304626.py:24: FutureWarning: Series.fillna with 'metho
d' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfi
ll() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000            96.39          -3.61           90.83
-9.17    100.0             0.0
0.0002   0.00005            94.37          -5.63           90.83
-9.17    100.0             0.0
0.0004   0.00010            89.26         -10.74           90.83
-9.17    100.0             0.0
0.0006   0.00030            79.26         -20.74           90.83
-9.17    100.0             0.0
```

In [ ]:

```python
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("BTC_1min.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['net_queue_slope'] = (df['bids_notional_0'] - df['bids_notional_5']) - (df[
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
df['depth_variance'] = df[bid_cols + ask_cols].std(axis=1)
df['abs_dobi'] = np.abs(df['dobi'])

train_end = int(len(df) * 0.6)
```

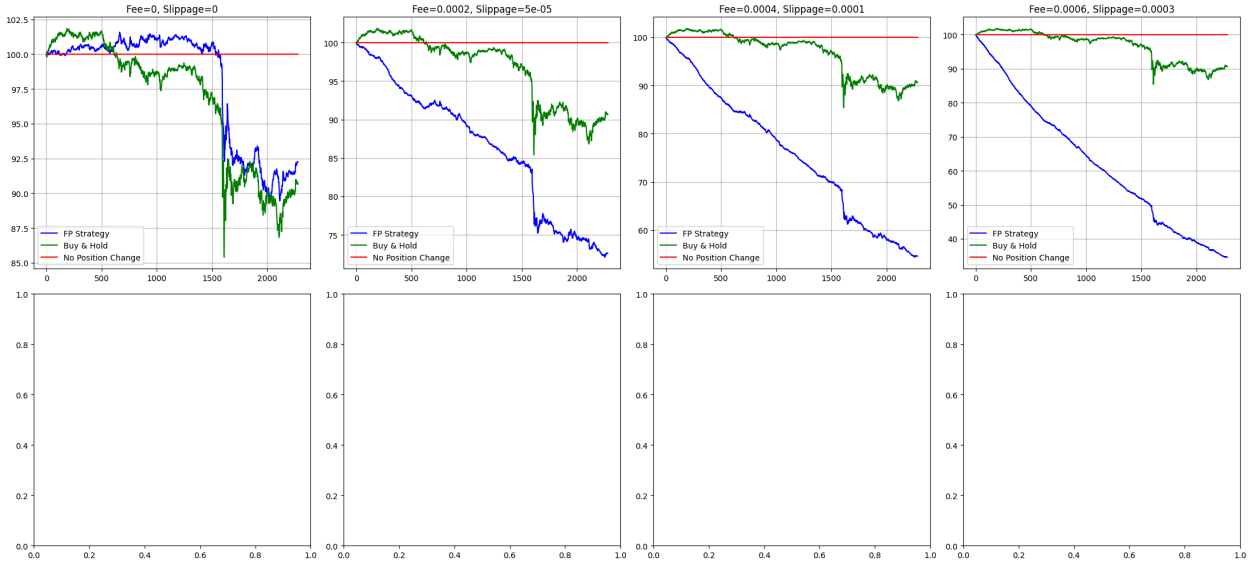
```
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        obi_t = features['obi'].iloc[t-1]
        dobi_t = features['dobi'].iloc[t-1]
        depth_t = features['depth'].iloc[t-1]
        slope_t = features['net_queue_slope'].iloc[t-1]
        spread_t = features['spread'].iloc[t-1]
        dv_t = features['depth_variance'].iloc[t-1]
        abs_dobi_t = features['abs_dobi'].iloc[t-1]
        mu = (a0 + a1 * x[t-1] + a2 * obi_t + a3 * dobi_t + a4 * depth_t + a5
        sigma = np.abs(b0 + b1 * (dv_t + abs_dobi_t))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
            best_thresh = t
    return best_thresh
```

```
def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'net_queue_slope', 'spread',
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:7]
        sigma_params = params[7:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*7 + [0.005, 0.005], sigma0=0.2, options={'seed':
    return res[0][:7], res[0][7:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+500) for i in range(0, len(df_train)-500, 500)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'net_queue_slope', 'sprea
    test_positions = []
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
```

```
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })

plt.tight_layout()
plt.show()
```

```
results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-3-97513049.py:21: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfil
l() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
```




```
Final Portfolio Values and Returns for Different Fee/Slippage Configurations:
   Fee  Slippage  FP Strategy ($)  FP Return (%)  Buy & Hold ($)  Buy & Hold Re
turn (%)  NPC ($)  NPC Return (%)
0.0000   0.00000            92.26          -7.74           90.67
-9.33    100.0             0.0
0.0002   0.00005            72.59         -27.41           90.67
-9.33    100.0             0.0
0.0004   0.00010            54.58         -45.42           90.67
-9.33    100.0             0.0
0.0006   0.00030            34.58         -65.42           90.67
-9.33    100.0             0.0
```

In [ ]:
```
import pandas as pd
import numpy as np
from cma import fmin
import matplotlib.pyplot as plt

np.random.seed(42)
random_seed = 42

df = pd.read_csv("BTC_1sec.csv")
for j in range(15):
    df[f'bid_price_{j}'] = df['midpoint'] - df[f'bids_distance_{j}']
    df[f'ask_price_{j}'] = df['midpoint'] + df[f'asks_distance_{j}']

bid_cols = [f"bids_notional_{i}" for i in range(15)]
ask_cols = [f"asks_notional_{i}" for i in range(15)]
df['obi'] = (df[bid_cols].sum(axis=1) - df[ask_cols].sum(axis=1)) / (df[bid_cc
```

```
df['dobi'] = df['obi'].diff().fillna(0)
df['depth'] = df[bid_cols + ask_cols].sum(axis=1)
df['queue_slope_bid'] = df['bids_notional_0'] - df['bids_notional_5']
df['queue_slope_ask'] = df['asks_notional_0'] - df['asks_notional_5']
df['net_queue_slope'] = df['queue_slope_bid'] - df['queue_slope_ask']
df['spread'] = np.where((df['asks_notional_0'] > 0) & (df['bids_notional_0'] >
df['spread'] = df['spread'].fillna(method='ffill').fillna(0)

train_end = int(len(df) * 0.6)
cv_end = int(len(df) * 0.8)
df_train = df.iloc[:train_end].copy().reset_index(drop=True)
df_cv = df.iloc[train_end:cv_end].copy().reset_index(drop=True)
df_test = df.iloc[cv_end:].copy().reset_index(drop=True)

for d in [df_train, df_cv, df_test]:
    d['log_mid'] = np.log(d['midpoint'])
    d['returns'] = d['log_mid'].diff().fillna(0)

def trading_strategy(signal, threshold):
    positions = np.where(signal > threshold, 1, np.where(signal < -threshold,
    trades = np.diff(positions, prepend=0)
    return positions, trades

def apply_trading_costs(positions, trades, returns, fee, slip):
    raw_pnl = positions[:-1] * returns[1:len(positions)]
    trade_mask = np.abs(trades[1:len(positions)]) > 0
    costs = np.zeros_like(raw_pnl)
    costs[trade_mask] = fee + slip
    net_pnl = raw_pnl - costs
    return net_pnl

def simulate_fp(mu_params, sigma_params, x0, features, timesteps, dt):
    a0, a1, a2, a3, a4, a5, a6, a7 = mu_params
    b0, b1 = sigma_params
    x = np.zeros(timesteps)
    x[0] = x0
    rng = np.random.RandomState(random_seed)
    for t in range(1, timesteps):
        f = features.iloc[t-1]
        mu = a0 + a1 * x[t-1] + a2 * f['obi'] + a3 * f['dobi'] + a4 * f['depth
        sigma = np.abs(b0 + b1 * np.abs(x[t-1]))
        x[t] = x[t-1] + mu * dt + sigma * np.sqrt(dt) * rng.randn()
    return x

def optimize_threshold(signal, returns, fee, slip):
    thresholds = np.linspace(0.001, 0.01, 15)
    best_pnl = -np.inf
    best_thresh = 0.005
    for t in thresholds:
        pos, trades = trading_strategy(signal, t)
        pnl = np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
        if pnl > best_pnl:
            best_pnl = pnl
```

```
            best_thresh = t
    return best_thresh

def train_fp_model(df_slice, fee, slip):
    returns = df_slice['returns'].values
    features = df_slice[['obi', 'dobi', 'depth', 'spread', 'queue_slope_bid',
    x_init = 0.0
    dt = 1.0
    def objective(params):
        mu_params = params[:8]
        sigma_params = params[8:]
        signal = simulate_fp(mu_params, sigma_params, x_init, features, len(re
        pos, trades = trading_strategy(signal, 0.005)
        return -np.sum(apply_trading_costs(pos, trades, returns, fee, slip))
    res = fmin(objective, [0]*8 + [0.005, 0.005], sigma0=0.2, options={'seed':
    return res[0][:8], res[0][8:]

fees = [0, 0.0002, 0.0004, 0.0006]
slippages = [0, 0.00005, 0.0001, 0.0003]
results = []
fig, axes = plt.subplots(2, 4, figsize=(22, 10))
axes = axes.flatten()

for idx, (fee, slip) in enumerate(zip(fees, slippages)):
    train_segments = [(i, i+200) for i in range(0, len(df_train)-200, 200)]
    segment_models = []
    segment_thresholds = []
    for start, end in train_segments:
        mu_p, sigma_p = train_fp_model(df_train.iloc[start:end], fee, slip)
        signal = simulate_fp(mu_p, sigma_p, 0.0, df_train.iloc[start:end][['ob
        threshold = optimize_threshold(signal, df_train.iloc[start:end]['retur
        segment_models.append((mu_p, sigma_p))
        segment_thresholds.append(threshold)

    window_size = 3
    cv_returns = df_cv['returns'].values
    selected_model_indices = []
    for start in range(0, len(cv_returns) - window_size, window_size):
        end = start + window_size
        best_pnl = -np.inf
        best_index = 0
        for i, (mu_p, sigma_p) in enumerate(segment_models):
            signal = simulate_fp(mu_p, sigma_p, 0.0, df_cv.iloc[start:end][['c
            pos, trades = trading_strategy(signal, segment_thresholds[i])
            pnl = np.sum(apply_trading_costs(pos, trades, cv_returns[start:end
            if pnl > best_pnl:
                best_pnl = pnl
                best_index = i
        selected_model_indices.append(best_index)

    test_returns = df_test['returns'].values
    test_features = df_test[['obi', 'dobi', 'depth', 'spread', 'queue_slope_bi
    test_positions = []
```

```
    test_trades = []
    for i, start in enumerate(range(0, len(test_returns) - window_size + 1, wi
        end = start + window_size
        model_index = selected_model_indices[min(i, len(selected_model_indices
        mu_p, sigma_p = segment_models[model_index]
        threshold = segment_thresholds[model_index]
        signal = simulate_fp(mu_p, sigma_p, 0.0, test_features.iloc[start:end]
        pos, trades = trading_strategy(signal, threshold)
        test_positions.append(pos)
        test_trades.append(trades)

    if not test_positions:
        continue

    fp_positions = np.concatenate([p[:-1] if len(p) > 1 else p for p in test_p
    fp_trades = np.concatenate([t[:-1] if len(t) > 1 else t for t in test_trad
    fp_returns = test_returns[1:len(fp_positions)+1]

    min_length = min(len(fp_positions), len(fp_returns))
    fp_positions = fp_positions[:min_length]
    fp_trades = fp_trades[:min_length]
    fp_returns = fp_returns[:min_length]

    initial_investment = 100
    fp_net_returns = apply_trading_costs(fp_positions, fp_trades, fp_returns,
    fp_pnl = initial_investment * np.exp(np.cumsum(fp_net_returns))

    bh_returns = test_returns[1:min_length+1]
    bh_pnl = initial_investment * np.exp(np.cumsum(bh_returns))

    first_position = fp_positions[0] if len(fp_positions) > 0 else 0
    initial_trade_cost = (fee + slip) if first_position != 0 else 0
    npc_returns = first_position * bh_returns - initial_trade_cost
    npc_pnl = initial_investment * np.exp(np.cumsum(npc_returns))

    ax = axes[idx]
    ax.plot(fp_pnl, label='FP Strategy', color='blue')
    ax.plot(bh_pnl, label='Buy & Hold', color='green')
    ax.plot(npc_pnl, label='No Position Change', color='red')
    ax.set_title(f"Fee={fee}, Slippage={slip}")
    ax.grid(True)
    ax.legend()

    results.append({
        "Fee": fee,
        "Slippage": slip,
        "FP Strategy ($)": round(fp_pnl[-1], 2),
        "FP Return (%)": round((fp_pnl[-1] - initial_investment) / initial_inv
        "Buy & Hold ($)": round(bh_pnl[-1], 2),
        "Buy & Hold Return (%)": round((bh_pnl[-1] - initial_investment) / ini
        "NPC ($)": round(npc_pnl[-1], 2),
        "NPC Return (%)": round((npc_pnl[-1] - initial_investment) / initial_i
    })
```

```
plt.tight_layout()
plt.show()

results_df = pd.DataFrame(results)
print("\nFinal Portfolio Values and Returns for Different Fee/Slippage Configu
print(results_df.to_string(index=False))
```

```
/tmp/ipython-input-4-247870957.py:23: FutureWarning: Series.fillna with 'metho
d' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfi
ll() instead.
  df['spread'] = df['spread'].fillna(method='ffill').fillna(0)
```