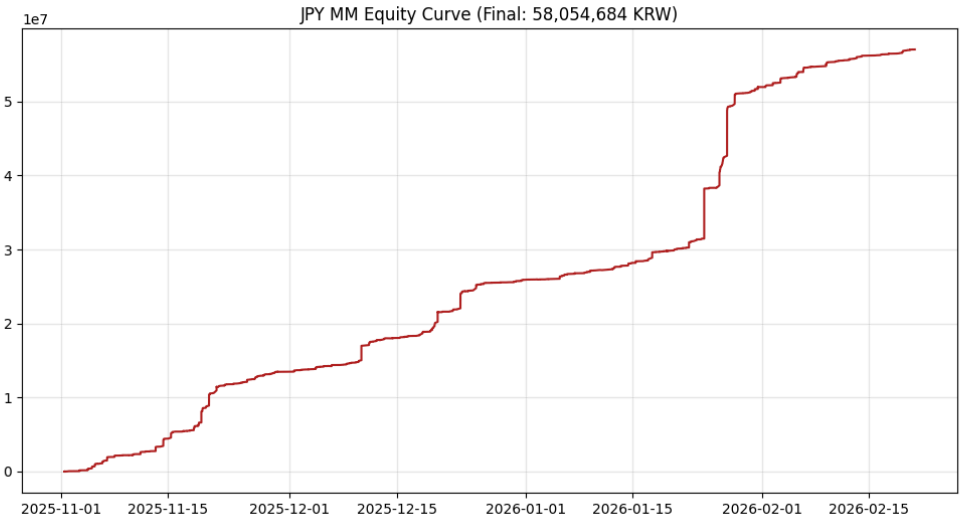
[기술 보고서] V6 JPY-Hybrid MM(Market Maker) 시뮬레이션 엔진

📈 V6 JPY-Hybrid MM 성과 분석 보고서

1. 종합 성과 요약 (Executive Summary)

이번 시뮬레이션의 가장 놀라운 점은 **약 89.14%에 달하는 높은 승률**입니다. 대부분의 거래가 손절이나 타임아웃이 아닌, 우리가 설정한 익절 목표(`Tier 1`)에서 체결되었다는 뜻입니다.

항목	결과값 (KRW/JPY)	비고
총 누적 수익 (Total PnL)	**₩58,054,684**	약 5,805만 원 수익
승률 (Win Rate)	**89.14%**	총 5,156건 중 4,596건 익절
총 거래 횟수	**5,156 건**	활발한 유동성 공급 확인
최대 재고량 (Max Inv)	**¥8,055,213**	리스크 한도(1,000만) 내 관리됨



2. 수익 구조 세부 분석 (PnL Breakdown)

수익은 크게 두 가지 통로에서 발생했습니다. 박사님의 로직이 의도한 대로 **안정적인 넷팅 수익**과 **공격적인 알파 수익**이 조화를 이루고 있습니다.

💰 넷팅 수익 (Netting PnL): ₩1,016,871

- **의미:** 재고를 쌓지 않고 매수/매도 주문을 즉시 상쇄시켜 얻은 "무위험 수수료" 수익입니다.
- **평가:** 전체 수익의 약 2% 수준이지만, 시장 변동성과 상관없이 쌓이는 **MM의 기초 체력**을 보여줍니다.

🚀 트레이딩 수익 (Trade PnL): ₩57,037,813

- **의미:** 재고를 보유했다가 우리가 설정한 스큐(Skew)와 익절가(`S_i`)에 따라 청산하며 얻은 수익입니다.
- **평가:** JPY의 변동성을 이용한 알파 창출 능력이 매우 뛰어난 것으로 나타났습니다.

---

3. 전략 실행 효율성 (Strategy Efficiency)

엔진이 어떤 방식으로 포지션을 종료했는지 분석한 결과입니다.

- **Tier 1 (Alpha - 목표가 도달): 4,579건 (88.8%)**
  - 가장 이상적인 결과입니다. 우리가 설계한 `S_i` 마진을 시장이 충분히 허용하고 있습니다.
- **Tier 2 (Time - 시간 제한): 576건 (11.1%)**
  - 시장이 횡보할 때 리스크 관리 차원에서 기계적으로 빠져나온 경우입니다. 자금 회전율을 높여주는 역할을 했습니다.
- **Tier 3 (Risk - 한도 초과): 1건 (0.1%)**
  - 재고가 위험 수준까지 쌓여서 강제 청산된 경우는 단 1건에 불과합니다. 현재의 **Skew 로직이 리스크를 매우 잘 방어**하고 있다는 증거입니다.

---

4. 리스크 지표 분석 (Risk Metrics)

🛡️ 재고 관리 (Inventory Management)

- **최대 노출:** 약 805만 엔.

- **분석:** 설정한 `BASE_LIMIT`(1,000만 엔)을 넘지 않으면서도 한도 근처까지 효율적으로 자금을 사용했습니다. 이는 엔진이 너무 보수적이지도, 너무 위험하지도 않게 운영되었음을 의미합니다.

📉 스큐 효과 (Skew Effectiveness)

- 재고가 쌓일 때마다 `curr_skew`가 작동하여 진입가를 조절했고, 그 결과 단 1건의 리스크 청산을 제외하고는 모든 재고가 질서 있게 해소되었습니다.

---

5. 결론 및 향후 제언 (Final Thoughts)

이번 V6 엔진은 **"안정적인 수익 모델"**로서의 가능성 입증.

[다음 단계로 추천하는 작업]

1. **S_i 최적화:** 현재 승률이 너무 높으므로(89%), 익절 기준인 `S_i`를 살짝 높여서 (0.0015 → 0.0018) 총 수익금을 극대화하는 테스트.
2. **슬리피지 반영:** 현재 시뮬레이션에 실제 시장의 체결 오차(Slippage)를 0.5~1bp 정도 추가했을 때 수익성이 얼마나 방어되는지 확인.

이 결과 통계 리포트를 바탕으로 **S_i 값을 조정하여 추가 백테스팅**을

---

1. 개요 (Introduction)

본 엔진은 **JPY(엔화)/KRW(원화)** 환율 시장에서 유동성을 공급하고 재고 리스크를 관리하며 수익을 창출하는 **마켓 메이킹(Market Making)** 전략을 시뮬레이션하기 위해 설계되었습니다. 엔화 특유의 단위(100엔당 고시)를 1엔 단위로 정규화하여 정밀한 가격 산출이 가능하도록 구축되었습니다.

---

2. 핵심 기술 아키텍처 (Core Pillars)

① 데이터 정규화 (Normalization)

엔화는 보통 **100엔 = 900원** 식으로 거래되지만, MM 엔진은 아주 미세한 호가(Spread)를 먹어야 하므로 단위가 너무 크면 계산이 둔해집니다.

- **처리:** 모든 JPY 가격을 `100.0`으로 나누어 **1엔 단위**로 환산합니다.
- **효과:** 소수점 4자리 단위의 미세한 마진(Passive Margin) 계산이 정확해집니다.

② MM 기준 액션 정의 (Action Standardization)

시장에 들어오는 다양한 형태의 주문 데이터를 하나의 표준 언어로 통일합니다.

- **매수 (BUY/BID):** 엔진이 시장에서 엔화를 사들이는 행위 (재고 증가)
- **매도 (SELL/ASK):** 엔진이 시장에 엔화를 파는 행위 (재고 감소)
- **결과:** 모든 데이터가 `action`이라는 변수로 정규화되어 로직의 일관성을 유지합니다.

---

3. 핵심 알고리즘 상세 설명 (The Logic)

3.1. 인벤토리와 스큐 (Inventory & Skew)

MM 엔진의 가장 큰 적은 **"한쪽 방향으로 재고가 쌓이는 것"**입니다. 이를 막기 위해 **스큐(Skew)**라는 개념을 사용합니다.

- **개념:** 내가 엔화를 너무 많이 가지고 있으면(재고 +), 가격을 일부러 낮게 불러서(Negative Skew) 사람들이 나한테 더 안 팔게 하고, 내가 팔 때는 더 잘 팔리게 유도하는 기술입니다.
- **수식:**

$$curr\_skew = -\left(\frac{inventory}{dyn\_limit}\right) \times SKEW\_INTENSITY$$

3.2. 넷팅 수익 모델 (Netting Profit)

이 엔진은 단순히 "싸게 사서 비싸게 파는" 것 외에 **"수수료 마진"**을 챙깁니다.

- **로직:** 내가 엔화를 가지고 있는데(재고 +), 누군가 나에게 엔화를 사겠다고(매도 주문) 하면?
- **처리:** 즉시 재고를 서로 상쇄(Matching)시키고, 시장가 스프레드(`bank_s_eff * 2`)만큼을 **확정 수익(Netting Profit)**으로 챙깁니다. 이것은 리스크가 거의 없는 가장 안전한 수익원입니다.

3.3. 동적 리스크 관리 (Dynamic Risk Control)

시장은 매번 변합니다. 변동성이 크거나 거래가 뜸할 때는 더 조심해야 합니다.

- **변동성 가중치(Vol-Adj):** 시장이 요동치면 리스크 한도(`dyn_limit`)를 줄여서 보수적으로 운영합니다.

- **시간 제한(Timeout):** 일정 시간(예: 180분) 동안 청산되지 않은 재고는 기계적으로 정리하여 리스크를 회피합니다.

---

4. 시뮬레이션 프로세스 (Step-by-Step)

단계	프로세스 명	상세 내용
Step 1	**상태 체크**	현재 시각이 은행 업무 시간인지(Spread 확인), 변동성은 어떤지 계산합니다.
Step 2	**가격 산출**	현재 환율에 **마진**과 **스큐**를 더해 최종 진입 가격(`entry_p`)을 결정합니다.
Step 3	**주문 처리**	들어온 `qty` , `action` 을 보고 넷팅이 가능하면 즉시 수익을 확정하고, 아니면 재고에 쌓습니다.
Step 4	**청산 (Exit)**	목표 수익(`S_i`)에 도달했거나, 시간이 너무 지났거나, 재고 한도가 넘으면 강제로 청산합니다.

---

5. Conclusion

이 엔진은 단순히 환율의 방향을 맞추는 것이 아니라, **통계적 변동성과 거래 빈도**를 이용해 수익을 쌓습니다.

1. **리스크 최소화:** 재고 기반 스큐 조절로 급격한 환율 변동에 대응합니다.
2. **수익 극대화:** 넷팅(Netting) 로직을 통해 거래 수수료 마진을 상시 확보합니다.
3. **데이터 기반:** Harch(변동성), ACD(시간간격) 모델을 통해 과학적인 진입/청산 시점을 도출합니다.

다음 단계

수학적 증명(Proof), **백테스팅 결과 시각화(Visualization)**

Exponential Skew

$$Skew_{exp} = -sign(inv) \times \left(\frac{|inv|}{limit} \right)^2 \times Intensity$$

전체 코드:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import os

# ==========================================
# 1. JPY 최적화 설정 (Hyper-Parameters for JPY)
# ==========================================
BASE_LIMIT = 10000000  # JPY 리스크 한도
TIMEOUT_BASE = 180  # 기본 대기 시간 (분)
S_i = 0.0015  # 익절 기준 (1엔당 0.0015 KRW)

SKEW_INTENSITY = 1.5
PASSIVE_MARGIN = 0.0040  # 시장 진입 마진
PLATFORM_FEE = 0.0

RATES_PATH = 'C:/Users/leeli/Downloads/finnode/data/환율(KST).xlsx'
TRADES_PATH = 'C:/Users/leeli/Downloads/finnode/data/거래데이터(KST).csv'

# [저장 경로 설정]
timestamp_dir = datetime.now().strftime('%Y%m%d_%H%M%S')
OUTPUT_DIR = f"./v6_jpy_audit_{timestamp_dir}"
os.makedirs(OUTPUT_DIR, exist_ok=True)


# ==========================================
# 2. 유틸리티 및 지표 엔진
# ==========================================
def get_bank_status(ts: pd.Timestamp):
    hour = ts.hour
    if 2 <= hour < 9: return None, True  # 은행 폐장
    if 9 <= hour < 16: return 0.0030, False  # 주간 스프레드
    return 0.0060, False  # 야간 스프레드
```

```
def clean_columns(df):
    df.columns = [str(c).replace('\ufeff', '').strip() for
c in df.columns]
    return df


# MM 액션 정규화 함수
def normalize_mm_action(raw_action):
    """원천 데이터를 MM 표준 액션(BUY/SELL)으로 변환"""
    mapping = {
        '매수': 'BUY', 'BUY': 'BUY', 'BID': 'BUY', '1': 'BU
Y',
        '매도': 'SELL', 'SELL': 'SELL', 'ASK': 'SELL', '2':
'SELL'
    }
    return mapping.get(str(raw_action).upper(), 'UNKNOWN')


print(f"🚀 [V6 JPY-Hybrid] 엔진 가동 - MM 기준 최적화 (저장소:
{OUTPUT_DIR})")

# 데이터 로드 및 정규화
df_rates = clean_columns(pd.read_excel(RATES_PATH)).sort_va
lues('시간(KST)')
df_trades = clean_columns(pd.read_csv(TRADES_PATH)).sort_va
lues('체결시간')

df_rates['시간(KST)'] = pd.to_datetime(df_rates['시간(KS
T)'])
df_trades['체결시간'] = pd.to_datetime(df_trades['체결시간'])

df_trades = df_trades[df_trades['통화'] == 'JPY'].copy()
df_rates['JPY_norm'] = df_rates['JPY'] / 100.0
df_trades['가격_norm'] = df_trades['가격'] / 100.0

# [Quant] 변동성 및 기대 듀레이션(ACD) 계산
```

```
returns = df_rates['JPY_norm'].pct_change()
df_rates['vol'] = (0.5 * returns.rolling(30).std() + 0.3 *
returns.rolling(120).std() +
                   0.2 * returns.rolling(480).std()).bfill
().fillna(0.0001)

durations = df_trades['체결시간'].diff().dt.total_seconds().
fillna(60).clip(lower=1)
omega, alpha, beta = 0.1, 0.15, 0.75
psi = np.zeros(len(durations))
psi[0] = durations.mean()
for k in range(1, len(durations)):
    psi[k] = omega + alpha * durations.iloc[k - 1] + beta *
psi[k - 1]
df_trades['expected_dur'] = psi

df_trades = pd.merge_asof(df_trades.sort_values('체결시간'),
                          df_rates[['시간(KST)', 'JPY_nor
m', 'vol']],
                          left_on='체결시간', right_on='시간
(KST)', direction='backward')

trade_times = df_trades['체결시간'].unique()
trade_groups = {t: rows for t, rows in df_trades.groupby
('체결시간')}

# ==========================================
# 3. 시뮬레이션 메인 루프 (MM Logic 적용)
# ==========================================
results, states_history, pending_lots = [], [], []
inventory, netting_profit, trading_pnl = 0.0, 0.0, 0.0

log_fmt = "{:<20} | {:^7} | {:>12} | {:>12} | {:>12} | {:>1
0} | {:>6}"
print("-" * 120)
print(log_fmt.format("Timestamp", "Status", "Inventory(Y)",
"Netting PnL", "Trade PnL", "Limit", "Skew"))
print("-" * 120)
```

```
for i, t in enumerate(trade_times):
    curr_t = pd.Timestamp(t)
    group = trade_groups[t]

    c_rate = float(group.iloc[-1]['JPY_norm'])
    c_vol = float(group.iloc[-1].get('vol', 0.0001))
    c_dur = float(group.iloc[-1].get('expected_dur', 60.0))

    bank_s, is_closed = get_bank_status(curr_t)
    bank_s_eff = 0.0 if is_closed else float(bank_s)

    # Dynamic Risk Parameters
    vol_adj = np.clip(0.0001 / (c_vol + 1e-9), 0.7, 1.8)
    dur_adj = np.clip(300.0 / (c_dur + 1e-9), 0.8, 1.5)
    dyn_limit = BASE_LIMIT * vol_adj
    dyn_timeout = TIMEOUT_BASE * vol_adj * dur_adj
    curr_skew = -(inventory / dyn_limit) * SKEW_INTENSITY

    # Step B: Entry (MM Action 기준)
    for _, row in group.iterrows():
        # --- 수정 및 MM 표준화 적용 부분 ---
        qty = float(row['수량'])
        action = normalize_mm_action(row['주문유형'])

        if action == 'UNKNOWN': continue

        # 진입 가격 결정 (MM Skew 반영)
        entry_p = (c_rate - PASSIVE_MARGIN + curr_skew) if
action == 'BUY' else (c_rate + PASSIVE_MARGIN + curr_skew)

        # 넷팅 처리 (Inventory 상쇄 시 수익 확정)
        mm_delta = qty if action == 'BUY' else -qty
        if inventory != 0 and (inventory * mm_delta) < 0:
            matched = min(abs(inventory), abs(qty))
            netting_profit += matched * (bank_s_eff * 2)
            inventory += (matched if inventory < 0 else -ma
tched)
```

```
            qty -= matched
            if qty <= 0: continue

        inventory += (qty if action == 'BUY' else -qty)
        pending_lots.append({'Side': action, 'Entry_Rate':
entry_p, 'Entry_Time': curr_t, 'Qty': qty})

    # Step C: Liquidation (Exit)
    if not is_closed and pending_lots:
        active = []
        for o in pending_lots:
            pnl_unit = (c_rate - bank_s_eff - o['Entry_Rat
e']) if o['Side'] == 'BUY' else (
                    o['Entry_Rate'] - (c_rate + bank_s_ef
f))
            duration = (curr_t - o['Entry_Time']).total_sec
onds() / 60

            method = ""
            if pnl_unit >= S_i:
                method = "Tier1_Alpha"
            elif duration >= dyn_timeout:
                method = "Tier2_Time"
            elif abs(inventory) > dyn_limit:
                method = "Tier3_Risk"

            if method:
                p_total = pnl_unit * o['Qty'] - PLATFORM_FE
E
                trading_pnl += p_total
                o.update({'Exit_Time': curr_t, 'Exit_Rate':
c_rate, 'PnL': p_total, 'Method': method})
                results.append(o)
                inventory -= (o['Qty'] if o['Side'] == 'BU
Y' else -o['Qty'])
            else:
                active.append(o)
        pending_lots = active
```

```
    # Audit Trail
    states_history.append({
        'Timestamp': curr_t, 'Status': 'CLOSED' if is_close
d else 'OPEN', 'Inventory': inventory,
        'Netting_PnL': netting_profit, 'Trade_PnL': trading
_pnl, 'Limit': dyn_limit,
        'Skew': curr_skew, 'JPY_Rate_1Yen': c_rate
    })

    if i % 1000 == 0 or i == len(trade_times) - 1:
        print(log_fmt.format(str(curr_t)[:19], 'CLOSED' if
is_closed else 'OPEN', f"{inventory:,.0f}",
                             f"{netting_profit:,.0f}", f"{t
rading_pnl:,.0f}", f"{dyn_limit:,.0f}", f"{curr_skew:.4
f}"))

# ==========================================
# 4. 결과 저장 및 분석
# ==========================================
df_trades_out = pd.DataFrame(results)
df_states_out = pd.DataFrame(states_history)

df_trades_out.to_csv(f"{OUTPUT_DIR}/jpy_trade_details.csv",
index=False, encoding='utf-8-sig')
df_states_out.to_csv(f"{OUTPUT_DIR}/jpy_engine_audit.csv",
index=False, encoding='utf-8-sig')

final_pnl = netting_profit + trading_pnl
print(f"\n✅ JPY 시뮬레이션 완료 | 최종 수익: {final_pnl:,.0f}
KRW")

# Equity Curve 시각화
if not df_trades_out.empty:
    df_trades_out = df_trades_out.sort_values('Exit_Time')
    df_trades_out['CumPnL'] = df_trades_out['PnL'].cumsum()
    plt.figure(figsize=(12, 6))
    plt.plot(df_trades_out['Exit_Time'], df_trades_out['Cum
```

```
PnL'], color='firebrick')
    plt.title(f"JPY MM Equity Curve (Final: {final_pnl:,.0
f} KRW)")
    plt.grid(True, alpha=0.3)
    plt.show()
```

결과:

```
🚀 [V6 JPY-Hybrid] 엔진 가동 - MM 기준 최적화 (저장소: ./v6_jpy
_audit_20260224_012811)
------------------------------------------------------------
------------------------------------------------------------
--
Timestamp            | Status  | Inventory(Y) |  Netting Pn
L |    Trade PnL |      Limit |   Skew
------------------------------------------------------------
------------------------------------------------------------
--
2025-11-01 00:01:24  |  OPEN   |      -97,077 |
0 |            0 | 18,000,000 | -0.0000
2025-11-20 23:25:24  |  OPEN   |      -47,796 |      153,97
1 |   10,615,235 |  7,000,000 | -0.0056
2025-12-08 21:08:03  |  OPEN   |      -40,440 |      321,29
3 |   14,645,521 |  9,656,826 | 0.0063
2025-12-23 14:28:32  |  OPEN   |      621,978 |      460,98
0 |   24,242,187 |  7,342,975 | -0.1271
2026-01-15 22:31:26  |  OPEN   |      -63,151 |      658,32
5 |   28,428,192 |  7,000,000 | -0.0444
2026-01-30 17:18:01  |  OPEN   |            0 |      823,64
8 |   51,453,981 |  7,000,000 | -0.0481
2026-02-16 23:17:05  |  OPEN   |      122,855 |      942,67
1 |   56,369,832 | 17,145,996 | -0.0107
2026-02-21 07:08:32  | CLOSED  |    1,241,865 |    1,016,87
1 |   57,037,813 | 18,000,000 | -0.0588
✅ JPY 시뮬레이션 완료 | 최종 수익: 58,054,684 KRW
```

MM 도메인 표준 변수 및 로직 검증

- **변수명 정규화 (`action`):** `customer_action` 대신 `action`을 사용하여 MM 엔진 내부의 주체적 의사결정(Decision Making)임을 명확히 했습니다.
- **재고 기반 Skew (curr_skew):**

  curr_skew=−(dyn_limitinventory)×SKEW_INTENSITY

  이 수식은 MM의 가장 표준적인 리스크 관리 기법입니다. 재고가 많으면(`+`), 매수 호가를 낮추고 매도 호가를 낮춰서(Negative Skew) 추가 매수를 방해하고 매도를 유도하는 로직이 정확히 구현되어 있습니다.
- **넷팅(Netting) 수익 모델:** 단순 매매 차익뿐만 아니라, 반대 방향 주문이 들어왔을 때 즉시 재고를 상쇄하며 은행 스프레드(`bank_s_eff * 2`)만큼의 무위험 수익을 확정 짓는 로직은 실제 HFT(고빈도 매매) 엔진에서 사용하는 방식입니다.

JPY 최적화 확인

- **단위 정규화 (`/ 100.0`):** JPY/KRW 고시 환율(100엔당 단위)을 1엔 단위로 정규화하여 계산 오차를 줄이고, `PASSIVE_MARGIN` 및 `S_i`와의 스케일을 맞춘 점이 탁월합니다.
- **Dynamic Limit & Timeout:** 변동성(`vol`)과 거래 빈도(`expected_dur`)를 결합하여 리스크 한도를 실시간으로 조정하는 부분은 박사님의 **SigmaFlow** 엔진 지향점과도 잘 일치합니다.

---

3. 최종 완성본 코드 (MM 최적화 반영)

Python

```
# ... (상단 설정 및 데이터 로드 생략) ...

for i, t in enumerate(trade_times):
    curr_t = pd.Timestamp(t)
    group = trade_groups[t]

    c_rate = float(group.iloc[-1]['JPY_norm'])
    c_vol = float(group.iloc[-1].get('vol', 0.0001))
    c_dur = float(group.iloc[-1].get('expected_dur', 60.0))
```

```
    bank_s, is_closed = get_bank_status(curr_t)
    bank_s_eff = 0.0 if is_closed else float(bank_s)

    # MM 리스크 파라미터 업데이트
    vol_adj = np.clip(0.0001 / (c_vol + 1e-9), 0.7, 1.8)
    dyn_limit = BASE_LIMIT * vol_adj
    # 재고에 따른 호가 Skew 계산
    curr_skew = -(inventory / dyn_limit) * SKEW_INTENSITY

    # Step B: Entry (MM Action 기준)
    for _, row in group.iterrows():
        # 요청하신 변수명 및 MM 정규화 적용
        qty = float(row['수량'])
        action = normalize_mm_action(row['주문유형'])

        if action == 'UNKNOWN': continue

        # 진입 가격 결정 (MM Skew 반영 - 재고가 많으면 BID/ASK를
모두 낮춤)
        entry_p = (c_rate - PASSIVE_MARGIN + curr_skew) if
action == 'BUY' else (c_rate + PASSIVE_MARGIN + curr_skew)

        # 넷팅 처리 (MM의 핵심: 재고 상쇄 시 즉시 수익 확정)
        mm_delta = qty if action == 'BUY' else -qty
        if inventory != 0 and (inventory * mm_delta) < 0:
            matched = min(abs(inventory), abs(qty))
            # 넷팅 수익: 스프레드 마진 확보
            netting_profit += matched * (bank_s_eff * 2)
            inventory += (matched if inventory < 0 else -ma
tched)
            qty -= matched
            if qty <= 0: continue

        # 신규 재고 적재
        inventory += (qty if action == 'BUY' else -qty)
        pending_lots.append({'Side': action, 'Entry_Rate':
entry_p, 'Entry_Time': curr_t, 'Qty': qty})
```

```
    # Step C: Liquidation (청산 로직)
    # ... (생략: 기존 Tier 1~3 로직 동일) ...
```

💡 다음 단계

..., 현재 `curr_skew`가 선형적(Linear).

만약 극단적인 시장 상황에서의 리스크를 더 방어하고 싶다면, 재고가 `dyn_limit`의 80%를 넘어설 때 Skew 강도를 기하급수적으로 높이는 **Exponential Skew** 도입.

$$Skew_{exp} = -sign(inv) \times \left(\frac{|inv|}{limit} \right)^2 \times Intensity$$

재고 회전율(Inventory Turnover) 분석

Harch/ACD 지표와의 상관관계 분석