Triangular FX Arbitrage: Statistical Analysis Research

Executive Summary

This study investigates the viability of triangular arbitrage in the EUR/USD, GBP/USD, and EUR/GBP triplet using a cointegration-based mean reversion strategy.

Key Findings:

- Cointegration is **strong** ($p < 0.01$), mean-reversion half-life ≈ 13 min.
- A naive model earns **Sharpe > 1**, but even **1 bp** friction kills the strategy.
- Asymmetrical filtering at **~99.9th percentile** restores viability, yielding a **Sharpe ~1.5** with manageable risk.
- Possibly profitable with institutional-grade execution infrastructure. Retail spreads are likely prohibitive.

A Note on Strategy Classification

"Triangular arbitrage" can mean two very different things:

1. **Atomic:** hit all 3 legs simultaneously (within milliseconds), flat instantly. No open position to close.
2. **Convergence (This Study):** Open a 3-leg position when the spread deviates, close when it converges. Exposed to mark-to-market risk.

Why We Model Convergence Trading:

1. **Realistic Infrastructure:** co-location and micro-second engines are off-limits for most traders/desks.
2. **Persistent Edge:** If the cointegration residual is still meaningful at 5-minute sampling; we can conservatively expect even better results with tick-level sampling. This is a "worst case".
3. **Risk-Adjusted Returns:** By waiting for sufficiently large dislocations and sizing properly, we can target Sharpe > 1.5 while keeping infrastructure costs modest.

The rest of the note focuses on convergence trading. If you do have HFT infrastructure, the only check is: **spread width > 3 × one-leg cost**; holding period and draw-down are irrelevant. You may also think about, at least philosophically, if the the HFTs are actively arbitraging this gap at the tick-level, why are we still seeing these dislocations at the 5-minute level?

Structure

We move through four layers of realism:

1. **Level 1: Naive Baseline** - Statistical foundations and in-sample analysis (with known limitations/leakages)
2. **Level 2: Toward Reality** - Walk-forward optimization (rolling window)
3. **Level 3: Viability** - Transaction costs, more realistic position sizing, and leverage
4. **Level 4: Production Considerations** - Operational risk: gaps, factor drift, tail events

---

A Note on "Return on Notional"

Throughout this report, "Return" is calculated on the **Gross Notional Exposure** of the trade (e.g., $100k position).

In [1]:
```
%load_ext autoreload
%autoreload 2
from fxarb.models import compute_zscore
from fxarb.models.ou_process import fit_ou_mle
from fxarb.backtest import run_backtest, Backtester, kelly_criterion
from fxarb.analysis.strategy import generate_signals, calculate_equity_with_costs
from fxarb.visualization.interactive import visualize_interactive_equity
from fxarb.visualization.performance import plot_equity_curve, plot_drawdown, plot_returns_distribution, plot_monthly_returns
from tqdm.auto import tqdm
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

print('All imports successful!')
from fxarb.data.loader import load_pair, resample_ohlc
from fxarb.data.features import compute_log_prices, add_session_features
from fxarb.analysis.cointegration import johansen_test, construct_spread, estimate_half_life
from fxarb.analysis.stationarity import adf_test, kpss_test, test_stationarity
```

```
All imports successful!
```

Level 0: Data Quality & Exploratory Analysis

This section focuses on data integrity. Before any complex modeling, we must verify that our high-frequency data is consistent, missing values are handled, and timestamps are aligned across the **EURUSD**, **GBPUSD**, and **EURGBP** triplet.

Objectives:

1. Load 1-minute OHLCV data.
2. Align timestamps (inner join) to ensure we have simultaneous quotes.
3. Visualize price history to identify gaps or anomalies.

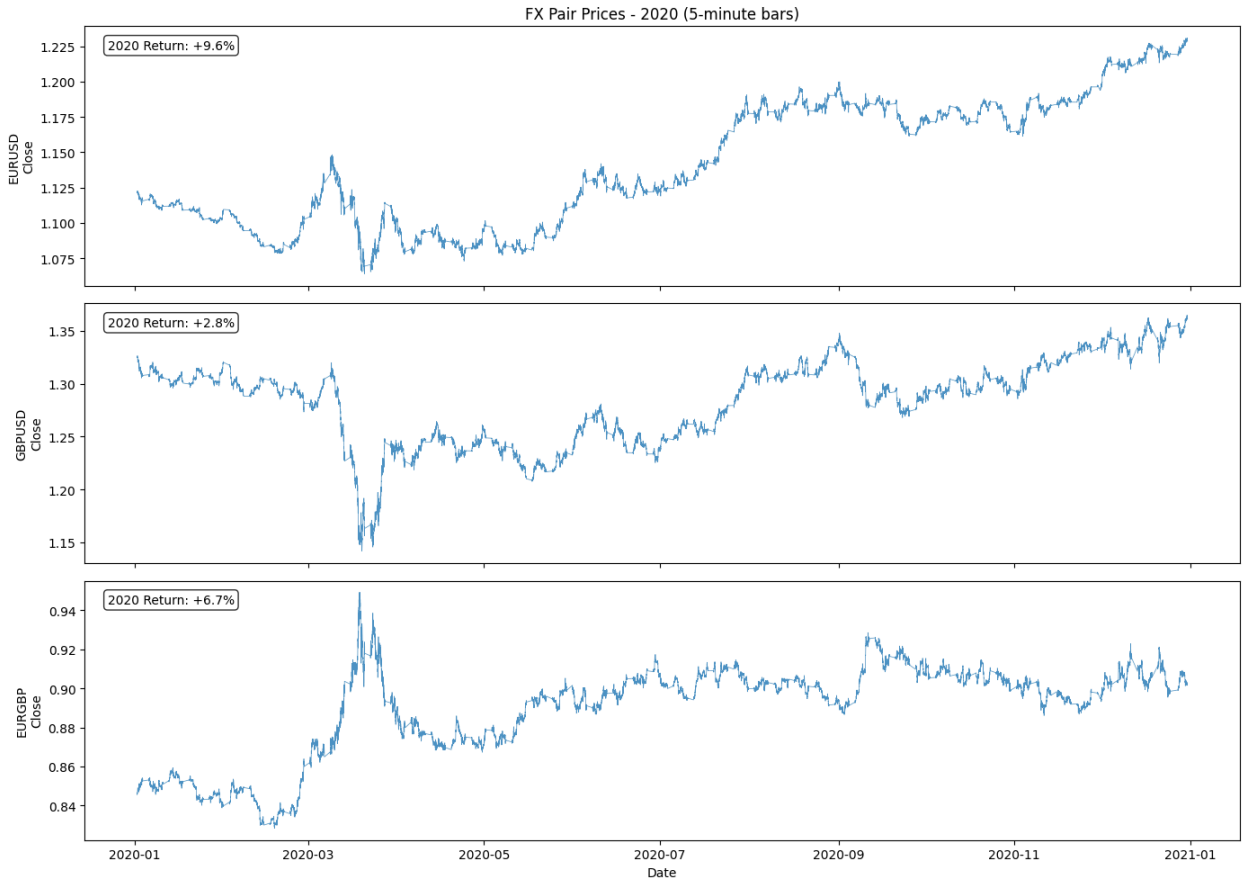
Data Aggregation Strategy: In this study, while we have access to 1-minute granular data, we aggregate to 5-minute bars for our primary signal generation.

1. **Noise Reduction**: Removes bid-ask bounce.
2. **Strategic Horizon**: Without HFT infra, we are scanning for large dislocations that last 15-60 minutes.

In [2]:
```
# Configuration
PAIRS = ['eurusd', 'gbpusd', 'eurgbp']
START_DATE = '2020-01-01'
END_DATE = '2020-12-31'
TIMEFRAME = '5min'

# Load data
print('Loading FX data...')
raw_data = {}
for pair in PAIRS:
    df = load_pair(pair, start_date=START_DATE, end_date=END_DATE)
    raw_data[pair] = df
    print(f'  {pair.upper()}: {len(df):,} 1-minute bars')

# Resample to 5-minute
print(f'\nResampling to {TIMEFRAME}...')
data_5m = {pair: resample_ohlc(df, TIMEFRAME) for pair, df in raw_data.items()}

# Align timestamps
common_idx = data_5m['eurusd'].index
for pair in PAIRS[1:]:
    common_idx = common_idx.intersection(data_5m[pair].index)

# Prepare data with log prices
data = {}
for pair in PAIRS:
    df = data_5m[pair].loc[common_idx].copy()
    df = compute_log_prices(df)
    df = add_session_features(df)
    data[pair] = df

print(f'\nAligned dataset: {len(common_idx):,} timestamps')
print(f'Date range: {common_idx.min()} to {common_idx.max()}')
```

```
Loading FX data...
  EURUSD: 371,257 1-minute bars
  GBPUSD: 371,979 1-minute bars
  EURGBP: 371,398 1-minute bars

Resampling to 5min...

Aligned dataset: 74,460 timestamps
Date range: 2020-01-01 17:00:00 to 2020-12-31 00:00:00
```

In [3]:
```
# Plot price overview
fig, axes = plt.subplots(3, 1, figsize=(14, 10), sharex=True)

for i, pair in enumerate(PAIRS):
    ax = axes[i]
    ax.plot(data[pair]['close'], linewidth=0.5, alpha=0.8)
    ax.set_ylabel(f'{pair.upper()}\nClose')

    # Add return annotation
    ret = (data[pair]['close'].iloc[-1] / data[pair]['close'].iloc[0] - 1) * 100
    ax.text(0.02, 0.95, f'2020 Return: {ret:+.1f}%', transform=ax.transAxes,
            va='top', fontsize=10, bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

axes[0].set_title('FX Pair Prices - 2020 (5-minute bars)')
axes[-1].set_xlabel('Date')
plt.tight_layout()
plt.show()
```



In [4]:
```
print("=" * 60)
print("LEVEL 0: DATA QUALITY & EXPLORATORY ANALYSIS")
print("=" * 60)

# Data coverage summary
print("\n📊 DATA COVERAGE")
print("-" * 40)
for pair in PAIRS:
    df = raw_data[pair]
    print(f"\n{pair.upper()}:")
    print(f"  Records: {len(df):,}")
    print(f"  Date range: {df.index.min()} to {df.index.max()}")
    print(f"  Missing rows: {df.isnull().any(axis=1).sum()}")

# Check for gaps
print("\n📅 DATA GAPS ANALYSIS")
print("-" * 40)
for pair in PAIRS:
    df = raw_data[pair]
    time_diffs = df.index.to_series().diff()
    expected_gap = pd.Timedelta(minutes=1)
    large_gaps = time_diffs[time_diffs > expected_gap * 5]  # >5 min gaps
    print(f"{pair.upper()}: {len(large_gaps)} gaps > 5 min (max: {time_diffs.max()})")

# OHLC sanity checks
print("\n✅ OHLC SANITY CHECKS")
print("-" * 40)
for pair in PAIRS:
    df = raw_data[pair]
    violations = {
        'high < low': (df['high'] < df['low']).sum(),
        'high < open': (df['high'] < df['open']).sum(),
        'high < close': (df['high'] < df['close']).sum(),
        'low > open': (df['low'] > df['open']).sum(),
        'low > close': (df['low'] > df['close']).sum(),
        'zero prices': (df[['open','high','low','close']] == 0).any(axis=1).sum(),
    }
    issues = {k: v for k, v in violations.items() if v > 0}
    print(f"{pair.upper()}: {'✓ Clean' if not issues else issues}")

# Basic statistics
print("\n📈 PRICE STATISTICS (1-min bars)")
print("-" * 40)
stats_df = pd.DataFrame({
    pair.upper(): {
        'Mean': raw_data[pair]['close'].mean(),
        'Std': raw_data[pair]['close'].std(),
        'Min': raw_data[pair]['close'].min(),
        'Max': raw_data[pair]['close'].max(),
```

```
        'Avg Daily Range': (raw_data[pair]['high'] - raw_data[pair]['low']).mean(),
    }
    for pair in PAIRS
}).T
print(stats_df.to_string())
```

```
============================================================
LEVEL 0: DATA QUALITY & EXPLORATORY ANALYSIS
============================================================

📊 DATA COVERAGE
----------------------------------------

EURUSD:
  Records: 371,257
  Date range: 2020-01-01 17:00:00 to 2020-12-31 00:00:00
  Missing rows: 0

GBPUSD:
  Records: 371,979
  Date range: 2020-01-01 17:02:00 to 2020-12-31 00:00:00
  Missing rows: 0

EURGBP:
  Records: 371,398
  Date range: 2020-01-01 17:02:00 to 2020-12-31 00:00:00
  Missing rows: 0

🗓️ DATA GAPS ANALYSIS
----------------------------------------
EURUSD: 72 gaps > 5 min (max: 2 days 14:07:00)
GBPUSD: 75 gaps > 5 min (max: 2 days 15:04:00)
EURGBP: 102 gaps > 5 min (max: 2 days 15:13:00)

✅ OHLC SANITY CHECKS
----------------------------------------
EURUSD: ✓ Clean
GBPUSD: ✓ Clean
EURGBP: ✓ Clean

📈 PRICE STATISTICS (1-min bars)
----------------------------------------
            Mean       Std      Min      Max  Avg Daily Range
EURUSD  1.141212  0.044160  1.06380  1.23088         0.000176
GBPUSD  1.283157  0.039765  1.14101  1.36485         0.000263
EURGBP  0.889400  0.023282  0.82820  0.94986         0.000164
```

In [5]:

```
# Bar aggregation verification
print("\n🔄 BAR AGGREGATION VERIFICATION")
print("-" * 40)
print(f"Original: 1-minute bars")
print(f"Resampled: {TIMEFRAME} bars")
print()

for pair in PAIRS:
    orig = raw_data[pair]
    resampled = data_5m[pair]

    # Check OHLC aggregation correctness (spot check a few random bars)
    sample_times = resampled.index[:3]
    print(f"{pair.upper()}:")
    print(f"  Original count: {len(orig):,}")
    print(f"  Resampled count: {len(resampled):,}")
    print(f"  Compression ratio: {len(orig)/len(resampled):.1f}x (expected: 5.0x)")

    # Verify first bar aggregation
    first_bar = resampled.iloc[0]
    first_bar_time = resampled.index[0]
    orig_slice = orig.loc[first_bar_time:first_bar_time + pd.Timedelta(minutes=4)]
    expected_high = orig_slice['high'].max()
    expected_low = orig_slice['low'].min()
    matches = (abs(first_bar['high'] - expected_high) < 1e-6 and
               abs(first_bar['low'] - expected_low) < 1e-6)
    print(f"  First bar aggregation: {'✓' if matches else 'X'}")
    print()
```

```
🔄 BAR AGGREGATION VERIFICATION
----------------------------------------
Original: 1-minute bars
Resampled: 5min bars

EURUSD:
  Original count: 371,257
  Resampled count: 74,541
  Compression ratio: 5.0x (expected: 5.0x)
  First bar aggregation: ✓

GBPUSD:
  Original count: 371,979
  Resampled count: 74,507
  Compression ratio: 5.0x (expected: 5.0x)
  First bar aggregation: ✓

EURGBP:
  Original count: 371,398
  Resampled count: 74,484
  Compression ratio: 5.0x (expected: 5.0x)
  First bar aggregation: ✓
```

Level 1: Naive Baseline

We begin with a **Naive Baseline** approach. We assume the spread is stationary and simply trade when it deviates by a fixed standard deviation (Z-score).

> [!NOTE] This **in-sample analysis** typically yields overly optimistic results (look-ahead bias) but serves as a crucial theoretical ceiling for strategy performance. If the strategy doesn't work here, it won't work anywhere.

Methodology:

1. Construct the synthetic spread: $S_t = \ln(EURUSD) - \ln(GBPUSD) - \ln(EURGBP)$
2. Estimate half-life of mean reversion (Ornstein-Uhlenbeck process).
3. Backtest using a simple symmetric threshold (e.g., enter at $\pm 2\sigma$, exit at 0).

1.1 Cointegration Analysis

Why One Spread for Three Pairs?

In triangular FX arbitrage, the three pairs EUR/USD, GBP/USD, and EUR/GBP are not independent, they form a **closed loop**:

$$\text{EUR/GBP} = \frac{\text{EUR/USD}}{\text{GBP/USD}}$$

Or in log terms:

$$\log(\text{EUR/GBP}) = \log(\text{EUR/USD}) - \log(\text{GBP/USD})$$

We construct a **single synthetic spread** from all three that should be stationary around zero due to the no-arbitrage condition.

The Johansen test finds the optimal linear combination (hedge ratios) that makes this spread most stationary:

$$\text{Spread} = \beta_1 \cdot \text{EURUSD} + \beta_2 \cdot \text{GBPUSD} + \beta_3 \cdot \text{EURGBP}$$

When the spread deviates from its mean, we trade all three pairs simultaneously to capture the reversion:

- **Long spread**: Buy EURUSD, Sell GBPUSD, Buy EURGBP (weighted)
- **Short spread**: The opposite

In [6]:
```python
# Run Johansen cointegration test
coint_result = johansen_test(data)

print('Johansen Cointegration Test Results')
print('=' * 50)
print(f'Cointegrating relationships found: {coint_result.n_cointegrating}')
print(f'\nHedge Ratios (normalized):')
for pair, ratio in coint_result.hedge_ratios.items():
    print(f'  {pair.upper()}: {ratio:.6f}')

# Construct the spread
spread = construct_spread(data, hedge_ratios=coint_result.hedge_ratios)

print(f'\nSpread Statistics:')
print(f'  Mean: {spread.mean():.6f}')
print(f'  Std: {spread.std():.6f}')
```

```
print(f'  Skew: {spread.skew():.3f}')
print(f'  Kurtosis: {spread.kurtosis():.3f}')
```

```
Johansen Cointegration Test Results
==================================================
Cointegrating relationships found: 1

Hedge Ratios (normalized):
  EURUSD: 1.000000
  GBPUSD: -0.999389
  EURGBP: -1.000094

Spread Statistics:
  Mean: -0.000000
  Std: 0.000134
  Skew: 9.602
  Kurtosis: 143.755
```

1.2 Stationarity Tests

For the spread to be tradable, it must be **stationary** (mean-reverting). We verify using:

- **ADF (Augmented Dickey-Fuller)**: Tests for unit root (null = non-stationary)
- **KPSS**: Tests for stationarity (null = stationary)

In [7]:
```
# Stationarity tests
adf = adf_test(spread)
kpss = kpss_test(spread)

print('Stationarity Test Results')
print('=' * 50)
print(f'ADF Test:')
print(f'  Statistic: {adf.statistic:.4f}')
print(f'  P-value: {adf.pvalue:.4f}')
print(f'  Conclusion: {"STATIONARY" if adf.is_stationary else "NON-STATIONARY"}')
print(f'\nKPSS Test:')
print(f'  Statistic: {kpss.statistic:.4f}')
print(f'  P-value: {kpss.pvalue:.4f}')
print(f'  Conclusion: {"STATIONARY" if kpss.is_stationary else "NON-STATIONARY"}')

if adf.is_stationary and kpss.is_stationary:
    print('\n✓ Both tests confirm stationarity')
else:
    print('\n⚠ Conflicting results - proceed with caution')
```

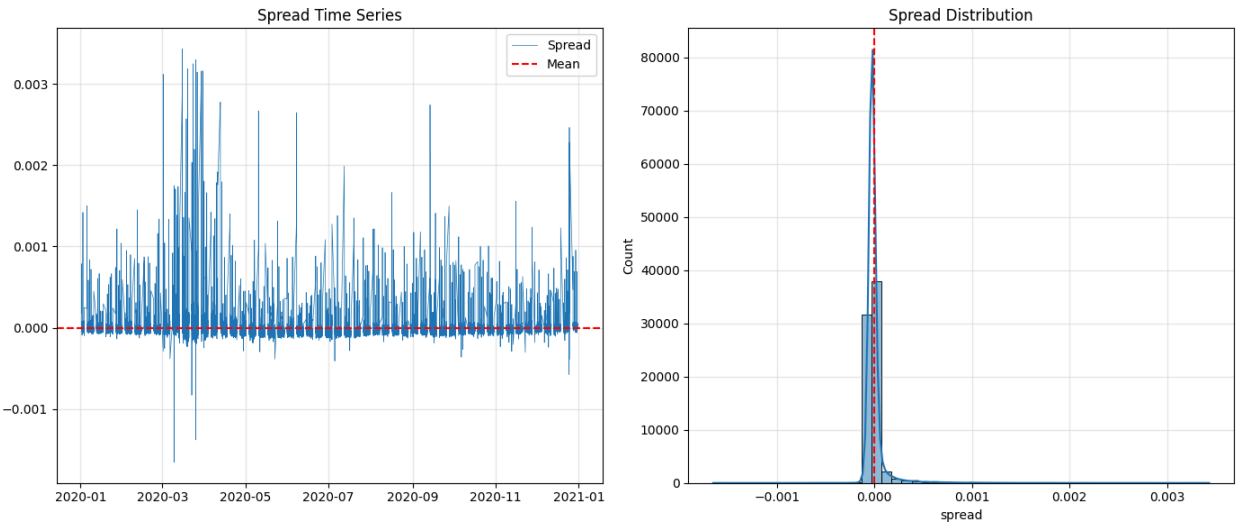
```
Stationarity Test Results
==================================================
ADF Test:
  Statistic: -30.4006
  P-value: 0.0000
  Conclusion: STATIONARY

KPSS Test:
  Statistic: 2.3349
  P-value: 0.0100
  Conclusion: NON-STATIONARY

⚠ Conflicting results - proceed with caution
E:\triangular-fx-arbitrage\src\fxarb\analysis\stationarity.py:126: InterpolationWarning: The test statistic is outside of the ra
nge of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  result = kpss(series_clean, regression=regression, nlags=nlags)
```

The kpss test defaults to regression='c' (constant mean). One potential reason for rejecting the null of stationarity is if it has a trend component, for example from interest rate differentials (carry). Let's try passing regression='ct' (constant + trend) to the test function.

In [8]:
```
from fxarb.analysis.stationarity import kpss_test
print("Running KPSS Test with Trend Regression ('ct')...")
print("Null Hypothesis: Series is Trend-Stationary")
print("-" * 50)
# Run test with regression='ct' (Constant + Trend)
series_to_test = spread.dropna()
kpss_result_trend = kpss_test(series_to_test, regression='ct')
print(kpss_result_trend)
if kpss_result_trend.pvalue > 0.05:
    print("\n✅ PASSED: Series is Trend-Stationary.")
    print("The previous failure was likely due to a deterministic trend (e.g. interest rate differentials).")
    print("You may need to detrend the series or add a rolling mean subtraction.")
else:
    print("\n❌ FAILED: Series is NOT Trend-Stationary.")
    print("The series likely has 'Stochastic Trends' (Random Walk behavior) or structural breaks.")
```

```
Running KPSS Test with Trend Regression ('ct')...
Null Hypothesis: Series is Trend-Stationary
--------------------------------------------------
KPSS Test: NON-STATIONARY
  Statistic: 1.974394
  P-value: 0.010000
  Critical values:
    10%: 0.119000
    5%: 0.146000
    2.5%: 0.176000
    1%: 0.216000
  Lags used: 134

❌ FAILED: Series is NOT Trend-Stationary.
The series likely has 'Stochastic Trends' (Random Walk behavior) or structural breaks.
E:\triangular-fx-arbitrage\src\fxarb\analysis\stationarity.py:126: InterpolationWarning: The test statistic is outside of the ra
nge of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  result = kpss(series_clean, regression=regression, nlags=nlags)
```

In [9]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
# Assuming 'spread' is your variable from previous cells
series_to_test = spread.dropna()
plt.figure(figsize=(14, 6))
# Subplot 1: Time Series
plt.subplot(1, 2, 1)
plt.plot(series_to_test.index, series_to_test.values, label='Spread', linewidth=0.5)
plt.axhline(series_to_test.mean(), color='red', linestyle='--', label='Mean')
plt.title('Spread Time Series')
plt.legend()
plt.grid(True, alpha=0.3)
# Subplot 2: Distribution
plt.subplot(1, 2, 2)
sns.histplot(series_to_test, kde=True, bins=50)
plt.axvline(series_to_test.mean(), color='red', linestyle='--', label='Mean')
plt.title('Spread Distribution')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
# Print basic stats
print(f"Mean: {series_to_test.mean():.6f}")
print(f"Std Dev: {series_to_test.std():.6f}")
print(f"Skew: {series_to_test.skew():.2f}")
print(f"Kurtosis: {series_to_test.kurtosis():.2f} (High > 3 implies fat tails/spikes)")
```




```
Mean: -0.000000
Std Dev: 0.000134
Skew: 9.60
Kurtosis: 143.76 (High > 3 implies fat tails/spikes)
```

Now we see the full picture - our historical dataset contains a *massive* shock from March 2020 causing outsized **volatility clustering** and could be considered a **structural break**. This heterogeneity means the series is "statistically stationary" but "parametrically unstable"—the mean is constant, but the variance is not, explaining the conflicting result (ADF=Stationary, KPSS=Non-Stationary). The extremely strong ADF statistic (-30.4) confirms the series is strongly mean-reverting, which is the primary requirement for trading. A static hedge ratio (Level 1) is probably insufficient, we will explore Walk-Forward Optimization (WFO) in Level 2 to adapt to shifting market regimes, although that is unlikely to be able to account for such events either.

1.3 Mean Reversion Speed

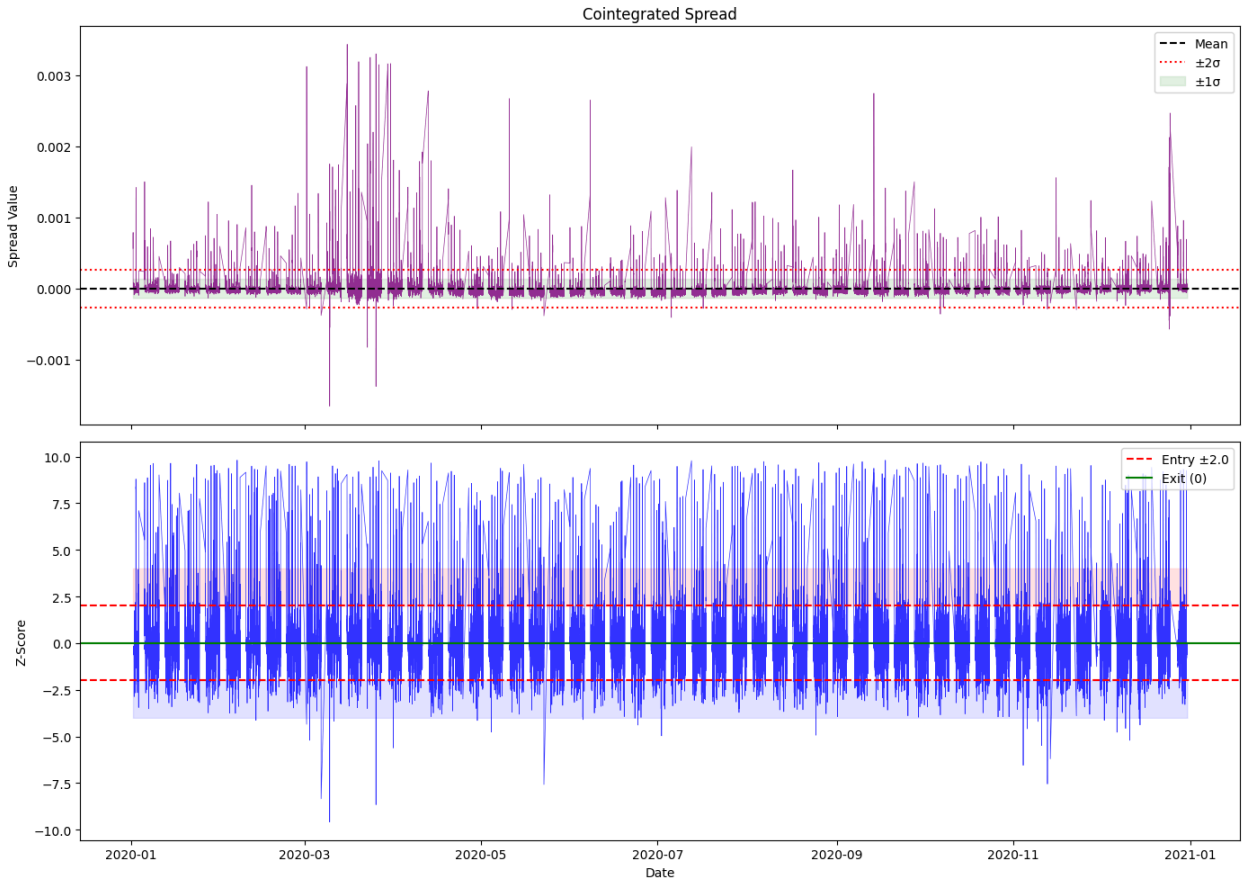
The **half-life** measures how quickly the spread reverts to its mean. We estimate this using:

1. AR(1) regression
2. Ornstein-Uhlenbeck (OU) process fitting

In [10]:
```
# Half-life estimation
try:
    hl = estimate_half_life(spread)
    print(f'Mean Reversion Half-Life: {hl:.1f} bars ({hl * 5:.0f} minutes)')
except ValueError as e:
    print(f'Half-life estimation failed: {e}')
    hl = None

# OU process fitting
try:
    ou = fit_ou_mle(spread)
    print(f'\nOrnstein-Uhlenbeck Parameters:')
    print(f'  θ (mean reversion speed): {ou.theta:.6f}')
    print(f'  μ (long-term mean): {ou.mu:.6f}')
    print(f'  σ (volatility): {ou.sigma:.6f}')
    print(f'  Half-life: {ou.half_life:.1f} bars')
except Exception as e:
    print(f'OU fitting failed: {e}')
```

```
Mean Reversion Half-Life: 2.5 bars (13 minutes)

Ornstein-Uhlenbeck Parameters:
  θ (mean reversion speed): 0.274118
  μ (long-term mean): -0.000000
  σ (volatility): 0.000099
  Half-life: 2.5 bars
```

In [11]:
```
# Visualize spread and Z-score
zscore = compute_zscore(spread, lookback=100)

fig, axes = plt.subplots(2, 1, figsize=(14, 10), sharex=True)

# Spread
ax1 = axes[0]
ax1.plot(spread, linewidth=0.5, alpha=0.8, color='purple')
ax1.axhline(spread.mean(), color='black', linestyle='--', label='Mean')
ax1.axhline(spread.mean() + 2*spread.std(), color='red', linestyle=':', label='±2σ')
ax1.axhline(spread.mean() - 2*spread.std(), color='red', linestyle=':')
ax1.fill_between(spread.index, spread.mean() - spread.std(), spread.mean() + spread.std(),
                 alpha=0.1, color='green', label='±1σ')
ax1.set_ylabel('Spread Value')
ax1.set_title('Cointegrated Spread')
ax1.legend(loc='upper right')

# Z-score
ax2 = axes[1]
ax2.plot(zscore, linewidth=0.5, alpha=0.8, color='blue')
ax2.axhline(2.0, color='red', linestyle='--', label='Entry ±2.0')
ax2.axhline(-2.0, color='red', linestyle='--')
ax2.axhline(0, color='green', linestyle='-', label='Exit (0)')
ax2.fill_between(zscore.index, 2, 4, alpha=0.1, color='red')
ax2.fill_between(zscore.index, -4, -2, alpha=0.1, color='blue')
ax2.set_ylabel('Z-Score')
ax2.set_xlabel('Date')
ax2.legend(loc='upper right')

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



Looking at the Z-score plot above, the deviations aren't perfectly symmetrical. There are sharper/taller spikes on one side. A simple symmetric threshold (like ± 2.0) might be leaving money on the table, but it's a good enough starting point for our baseline backtest.

1.4 Naive Strategy Backtest

Strategy rules (fixed parameters):

- Entry: Z-score crosses ±2.0
- Exit: Z-score crosses 0
- Position: Long spread when Z < -2, Short when Z > +2

> ⚠️ **Bias Warning**: This uses full-sample cointegration parameters (lookahead bias).

In [13]:
```
# Generate signals
signals = generate_signals(zscore, entry_threshold=2.0, exit_threshold=0.0)

# Count signals
n_long = (signals['entry'] == 1).sum()
n_short = (signals['entry'] == -1).sum()
time_in_market = (signals['position'] != 0).mean()

print('Signal Statistics')
print('=' * 40)
print(f'Long entries: {n_long}')
print(f'Short entries: {n_short}')
print(f'Total entries: {n_long + n_short}')
print(f'Time in market: {time_in_market:.1%}')
```

```
Signal Statistics
========================================
Long entries: 4252
Short entries: 0
Total entries: 4252
Time in market: 15.6%
```

Execution Assumption: Close-to-Close vs. Next-Open

In this backtest, we calculate returns based on **Close-to-Close** prices:

$$R_t = \text{Position}_{t-1} \times \frac{P_t - P_{t-1}}{P_{t-1}}$$

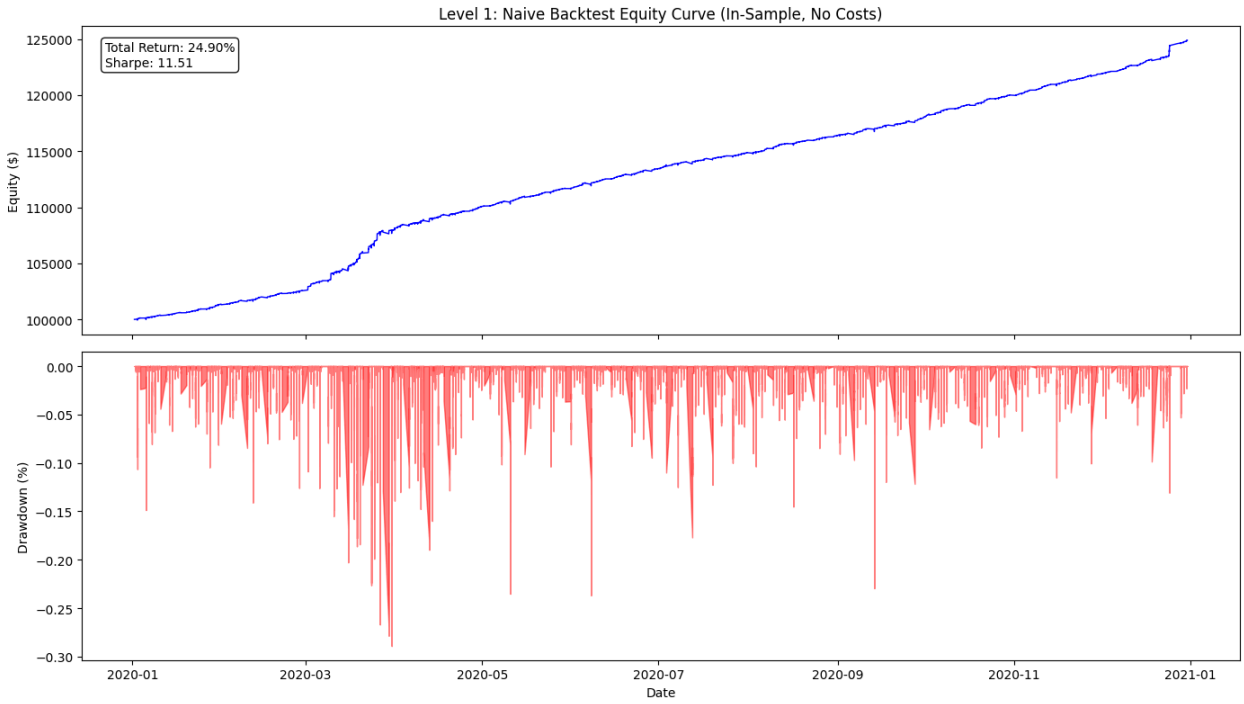
Justification: In highly liquid 5-minute FX markets, the difference between $Close_{t-1}$ (signal generation time) and $Open_t$ (execution time) is minimal. This "frictionless" assumption serves as a standard proxy for execution at the next open in vectorized backtesting. While it abstracts

away specific slippage or weekend gaps, it provides a robust baseline for evaluating signal quality before execution constraints are applied in Level 3.

In [14]:
```
# Run backtest (NO transaction costs - Level 1 baseline)
result = run_backtest(spread, signals)

print('Level 1: Naive Backtest Results (In-Sample, No Costs)')
print('=' * 60)
print(result)
```

```
Level 1: Naive Backtest Results (In-Sample, No Costs)
============================================================
Backtest Results
========================================
Total Return: 24.90%
CAGR: 24.20%
Sharpe Ratio: 11.51
Max Drawdown: -0.29%
Calmar Ratio: 83.73

Trades:
  Total: 2126
  Winning: 2112 (99.3%)
  Losing: 14
  Avg Return: 0.0117%
  Avg Duration: 5.4 bars
```

In [15]:
```
# Plot equity curve
fig, axes = plt.subplots(2, 1, figsize=(14, 8), sharex=True)

ax1 = axes[0]
ax1.plot(result.equity_curve, linewidth=1, color='blue')
ax1.set_ylabel('Equity ($)')
ax1.set_title('Level 1: Naive Backtest Equity Curve (In-Sample, No Costs)')
ax1.text(0.02, 0.95, f'Total Return: {result.total_return:.2%}\nSharpe: {result.sharpe:.2f}',
         transform=ax1.transAxes, va='top', fontsize=10,
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.9))

# Drawdown
running_max = result.equity_curve.cummax()
drawdown = (result.equity_curve - running_max) / running_max * 100

ax2 = axes[1]
ax2.fill_between(drawdown.index, 0, drawdown, color='red', alpha=0.5)
ax2.set_ylabel('Drawdown (%)')
ax2.set_xlabel('Date')

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



1.5 Z-Score Asymmetry Consideration

⚠️ **Important**: Z-scores from cointegrated spreads are often **asymmetric**. The spread may:

- Frequently deviate to +9σ but rarely below -3σ

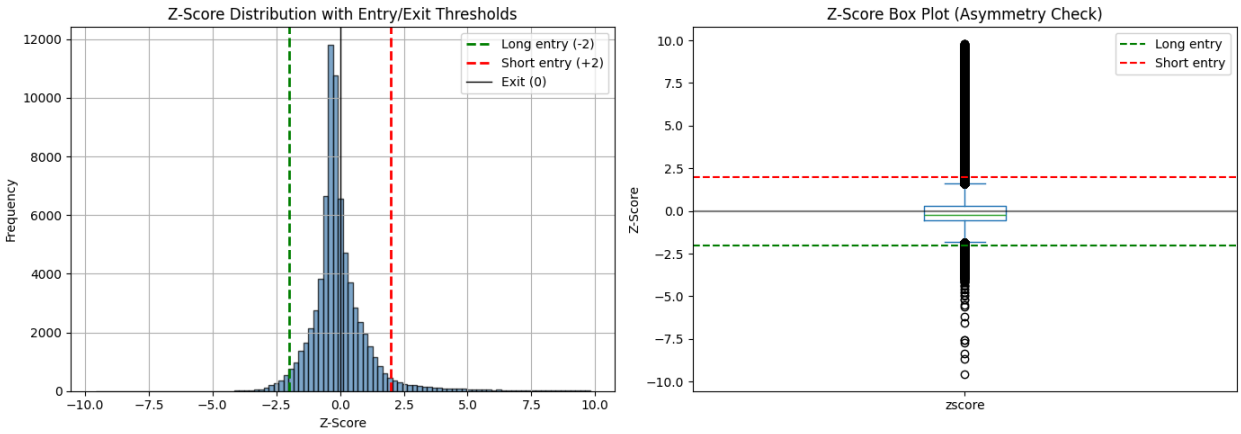
- Or vice versa, depending on market microstructure

Symmetric entry thresholds (e.g., ±2.0) may not be optimal. Level 2 explores:

- Data-driven threshold selection
- Asymmetric entry rules based on z-score distribution
- Sensitivity analysis across threshold values

In [ ]:
```
# Z-Score Distribution Analysis
print("Z-Score Distribution Analysis")
print("=" * 50)
print(f"Count: {zscore.dropna().count():,}")
print(f"Min: {zscore.min():.2f}")
print(f"Max: {zscore.max():.2f}")
print(f"Mean: {zscore.mean():.3f}")
print(f"Skewness: {zscore.skew():.3f}")
print(f"Kurtosis: {zscore.kurtosis():.3f}")
print()
print("Percentiles:")
print(f"  1st:  {zscore.quantile(0.01):.2f}")
print(f"  5th:  {zscore.quantile(0.05):.2f}")
print(f"  50th: {zscore.quantile(0.50):.2f}")
print(f"  95th: {zscore.quantile(0.95):.2f}")
print(f"  99th: {zscore.quantile(0.99):.2f}")

# Histogram with entry thresholds
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Distribution
ax1 = axes[0]
zscore.dropna().hist(bins=100, ax=ax1, alpha=0.7, edgecolor='black', color='steelblue')
ax1.axvline(-2, color='green', linestyle='--', linewidth=2, label='Long entry (-2)')
ax1.axvline(2, color='red', linestyle='--', linewidth=2, label='Short entry (+2)')
ax1.axvline(0, color='black', linestyle='-', linewidth=1, label='Exit (0)')
ax1.set_xlabel('Z-Score')
ax1.set_ylabel('Frequency')
ax1.set_title('Z-Score Distribution with Entry/Exit Thresholds')
ax1.legend()

# Box plot to show asymmetry
ax2 = axes[1]
zscore.dropna().plot.box(ax=ax2, vert=True)
ax2.axhline(-2, color='green', linestyle='--', label='Long entry')
ax2.axhline(2, color='red', linestyle='--', label='Short entry')
ax2.axhline(0, color='black', linestyle='-', alpha=0.5)
ax2.set_ylabel('Z-Score')
ax2.set_title('Z-Score Box Plot (Asymmetry Check)')
ax2.legend()

plt.tight_layout()
plt.show()

# Flag asymmetry
if abs(zscore.skew()) > 0.5:
    print(f"\n⚠️ Z-score is skewed (skewness={zscore.skew():.2f})")
    print("Symmetric thresholds may be suboptimal. See Level 2 for asymmetric analysis.")
else:
    print(f"\n✓ Z-score appears roughly symmetric (skewness={zscore.skew():.2f})")
```

```
Z-Score Distribution Analysis
==================================================
Count: 74,411
Min: -9.58
Max: 9.80
Mean: -0.030
Skewness: 2.531
Kurtosis: 13.719

Percentiles:
  1st:  -2.44
  5th:  -1.59
  50th: -0.22
  95th: 1.84
  99th: 5.15
```



⚠️ Z-score is skewed (skewness=2.53)
Symmetric thresholds may be suboptimal. See Level 2 for asymmetric analysis.

1.6 Reality Check - Latency Decay Analysis

A Sharpe Ratio of 11.5 is obviously unrealistic (with 5-minute bars). Aside from the lookahead bias (training on the same data we trade), this backtest assumes we execute **instantaneously** at the closing price of the signal bar.

In reality, by the time we compute the signal and send the order, the price has moved. In Level 1.5, we'll test how fragile this alpha is by forcing execution at the **Next Open** (a worst-case lag of 5 minutes).

In the naive backtest above, we trade the close, with the assumption that for liquid FX pairs, `Close[t] = Open[t+1]`. In reality, by the time we calculate the signal and send the order, the market may have moved. Here, we check how much it actually matters for this dataset.

In [16]:
```python
# Level 1.5: Lagged Execution Analysis (Corrected)
print('\nLevel 1.5: Lagged Execution Analysis (Execution at Next Open)')
print('=' * 60)

# 1. Define Spread Open
# We need to construct the spread using Open prices to simulate next-bar execution price.
# hedge_ratios are structural and assumed stable intraday (borrowed from Level 1 result)

# Create deep copy to avoid modifying original data
import copy
data_open = {k: v.copy() for k, v in data.items()}

# Compute log prices for OPEN
for pair in data_open:
    data_open[pair] = compute_log_prices(data_open[pair], price_col='open')

# Construct spread using OPEN prices
spread_open = construct_spread(data_open, hedge_ratios=coint_result.hedge_ratios)

print(f"Spread Open Mean: {spread_open.mean():.6f} (Should be close to 0)")

result_lagged = run_backtest(
    spread,
    signals,
    execution_price=spread_open,
    lag=2,
    transaction_cost_pips=0.0 # Keeping frictionless for pure latency isolation
)

# 3. Compare with Ideal (Naive) Result
print(f"{'Metric':<20} {'Frictionless':<15} {'Next-Open':<15} {'Decay':<10}")
print("-" * 60)
decay_sharpe = (result_lagged.sharpe / result.sharpe) - 1 if result.sharpe != 0 else 0
decay_ret = (result_lagged.total_return / result.total_return) - 1 if result.total_return != 0 else 0

print(f"{'Sharpe Ratio':<20} {result.sharpe:<15.2f} {result_lagged.sharpe:<15.2f} {decay_sharpe:.1%}")
print(f"{'Total Return':<20} {result.total_return:<15.2%} {result_lagged.total_return:<15.2%} {decay_ret:.1%}")

if result_lagged.sharpe < 0.8*result.sharpe:
    print("\n⚠️ CRITICAL: Strategy alpha heavily decays with execution at next Open.")
    print("The strategy is highly sensitive to latency/execution timing (as expected).")
else:
    print("\n✓ Strategy retains viability executing at next Open.")
```

```
Level 1.5: Lagged Execution Analysis (Execution at Next Open)
============================================================
Spread Open Mean: -0.000000 (Should be close to 0)
Metric               Frictionless    Next-Open       Decay
------------------------------------------------------------
Sharpe Ratio         11.51           8.37            -27.3%
Total Return         24.90%          18.96%          -23.9%

⚠️ CRITICAL: Strategy alpha heavily decays with execution at next Open.
The strategy is highly sensitive to latency/execution timing (as expected).
```

Level 1 Conclusion: Promising but Fragile

The strategy survives the lag, but the performance decay is significant. There is substantial Close-to-Open movement, hinting that this strategy is likely only truly viable with tick-level data.

However, other big problems remain, such as **Lookahead Bias**. We calculated our hedge ratios using data from December 2020 to trade in January 2020. That's cheating. To see if this strategy holds water in the real world, we need to calculate parameters from backwards-looking **rolling windows**.

Level 2: Toward Reality

In this section, we remove the lookahead bias by using **walk-forward optimization** (WFO). Parameters are estimated only on past data.

2.1 Why Walk-Forward Optimization?

The Level 1 backtest used **full-sample cointegration** — the hedge ratios were calculated using all 2020 data, including future data that wouldn't be available in real trading.

WFO addresses this by:

1. Training on a historical window (e.g., 30 days)
2. Testing on the next period (e.g., 7 days)
3. Rolling forward and repeating

This simulates periodic recalibration as a real trader would do.

In [17]:
```
# WFO Configuration (tqdm already imported in first cell)
TRAIN_DAYS = 30
TEST_DAYS = 7
BARS_PER_DAY = 288  # 5-min bars

train_size = TRAIN_DAYS * BARS_PER_DAY
test_size = TEST_DAYS * BARS_PER_DAY

print(f'Walk-Forward Configuration:')
print(f'  Training window: {TRAIN_DAYS} days ({train_size:,} bars)')
print(f'  Test window: {TEST_DAYS} days ({test_size:,} bars)')
print(f'  Expected windows: ~{(len(common_idx) - train_size) // test_size}')
```

```
Walk-Forward Configuration:
  Training window: 30 days (8,640 bars)
  Test window: 7 days (2,016 bars)
  Expected windows: ~32
```

2.2 Hedge Ratio Stability

In a textbook scenario, the relationship $\mathrm{EUR/GBP} = \frac{\mathrm{EUR/USD}}{\mathrm{GBP/USD}}$ implies a perfect 1:1:1 arithmetic balance. For example, if we hold \$100k of each leg, we **should** be perfectly hedged. However, in the real world, pairs exhibit different volatility and liquidity profiles. The **Hedge Ratio** (calculated via the Johansen test) acts as a "volatility equalizer," determining the precise weighting required to make the spread stationary. For example, if GBP/USD is highly volatile, the model might lower its weight (e.g., to 0.96) to dampen its impact on the spread.

Why monitor stability? In this Walk-Forward Analysis, we track how these ratios evolve over time to distinguish between a structural edge and statistical overfitting:

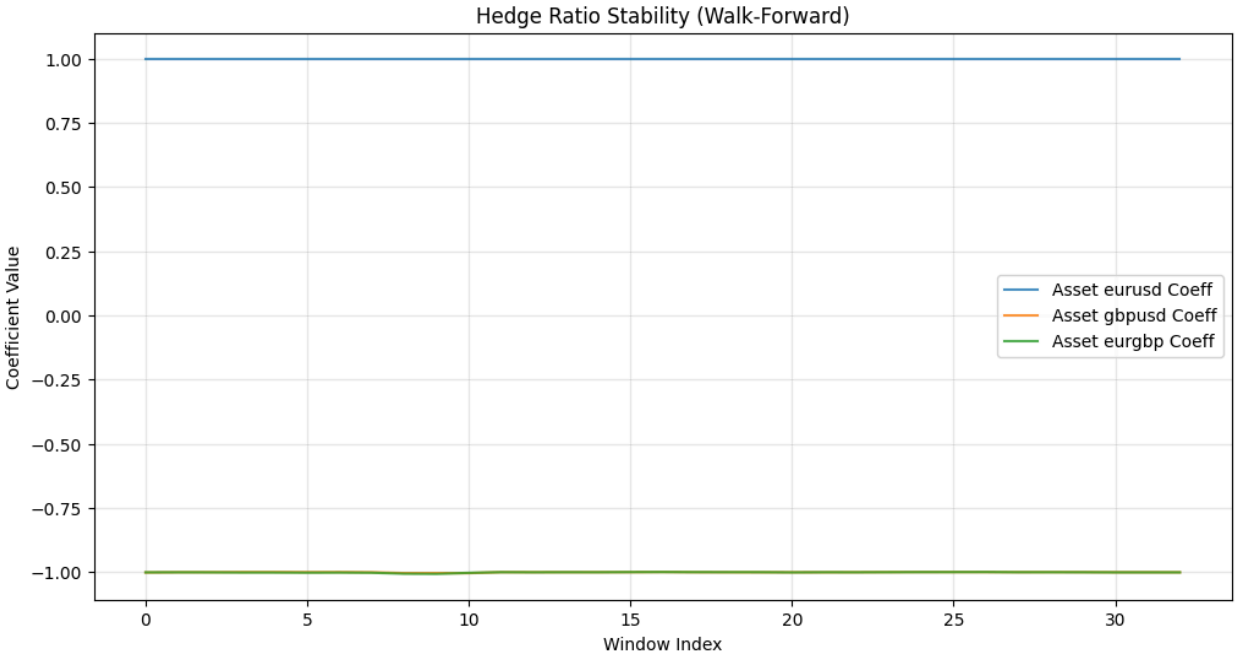
- **Stable Ratios (near 1.0):** Indicate the fundamental arbitrage relationship is intact and stable over time. The model is capturing the structural link between the currencies.
- **Volatile Ratios (e.g., jumping 0.8 → 1.5):** Suggest the model is "chasing noise," drastically rebalancing weights just to force a stationary fit on past data. This leads to dangerous exposure and high rebalancing costs.

In [18]:
```
# WFO with Fixed Thresholds (Basic)
# This loop iterates through data in rolling windows to prevent lookahead bias.
# It recalculates hedge ratios but uses fixed thresholds for simplicity.
```

```
print("Starting Walk-Forward Optimization (Fixed Thresholds basic check)...")

all_hedge_ratios = []
all_oos_trades = []
all_spreads = []
all_zscores = []
all_oos_signals = []
wfo_results = []

# Fixed Thresholds for Level 2.1 Baseline
ENTRY_THRESHOLD = 2.0
EXIT_THRESHOLD = 0.0

# WFO Configuration
if 'train_size' not in locals():
    train_size = 30 * 288
    test_size = 7 * 288

# Iterate through Rolling Windows
n_windows = (len(common_idx) - train_size) // test_size
print(f"Processing (~{n_windows}) windows...")

for i in tqdm(range(0, len(common_idx) - train_size, test_size), desc="WFO Windows"):
    # 1. Define Window Indices
    train_start = i
    train_end = train_start + train_size
    test_end = min(train_end + test_size, len(common_idx))

    if test_end - train_end < 100: break

    train_idx = common_idx[train_start:train_end]
    test_idx = common_idx[train_end:test_end]

    # 2. Get Data Slices
    train_data = {t: data[t].loc[train_idx] for t in PAIRS}
    test_data = {t: data[t].loc[test_idx] for t in PAIRS}

    # 3. Train: Fit Cointegration Model (Recalculate Hedge Ratios)
    try:
        coint_res = johansen_test(train_data)
        hedge_ratios = coint_res.hedge_ratios
        all_hedge_ratios.append(hedge_ratios) # Store for Stability Plot
    except Exception as e:
        # If Johansen fails (rare), skip window
        continue

    # 4. Train: Calculate Spread (to carry forward state if needed)
    spread_train = construct_spread(train_data, hedge_ratios)

    # 5. Test: Out-of-Sample Backtest
    spread_test = construct_spread(test_data, hedge_ratios)

    # Combine for Z-score warmup (preventing cold start artifacts)
    combined_spread = pd.concat([spread_train.iloc[-500:], spread_test])
    zscore_combined = compute_zscore(combined_spread)
    zscore_test = zscore_combined.loc[test_idx]

    # Generate Signals - FIXED THRESHOLDS
    signals = generate_signals(
        zscore_test,
        long_entry_threshold=-ENTRY_THRESHOLD,
        short_entry_threshold=ENTRY_THRESHOLD,
        exit_threshold=EXIT_THRESHOLD
    )

    # Run frictionless backtest
    res = run_backtest(spread_test, signals, transaction_cost_pips=0.0)

    # Collect Trades for Kelly Analysis
    if hasattr(res, 'trades'):
        all_oos_trades.append(res.trades)

    # Collect OOS Data for Level 3 Analysis
    all_spreads.append(spread_test)
    all_zscores.append(zscore_test)
    all_oos_signals.append(signals)


# Aggregate OOS Data for Level 3
if all_spreads:
    spread_oos = pd.concat(all_spreads)
    zscore_oos = pd.concat(all_zscores)
    # Ensure they are sorted (though WFO should be chronological)
    spread_oos = spread_oos.sort_index()
    zscore_oos = zscore_oos.sort_index()
else:
    print("Warning: No OOS spreads collected!")

print(f"WFO Complete. Collected {len(all_hedge_ratios)} hedge ratio sets and {len(all_oos_trades)} trade sets.")
```

```
Starting Walk-Forward Optimization (Fixed Thresholds basic check)...
Processing (~32) windows...
WFO Windows:   0%|          | 0/33 [00:00<?, ?it/s]
WFO Complete. Collected 33 hedge ratio sets and 33 trade sets.
```

In [19]:
```python
# Plot Hedge Ratio Evolution
if 'all_hedge_ratios' in locals() and all_hedge_ratios:
    hr_df = pd.DataFrame(all_hedge_ratios)
    plt.figure(figsize=(12, 6))
    for col in hr_df.columns:
        plt.plot(hr_df.index, hr_df[col], label=f'Asset {col} Coeff', alpha=0.8)
    plt.title('Hedge Ratio Stability (Walk-Forward)')
    plt.ylabel('Coefficient Value')
    plt.xlabel('Window Index')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.show()
else:
    print('No hedge ratios collected.')
```



Lines are almost completely flat, as expected of structural arbitrage strategies of highly liquid instruments.

2.3 Naive Dynamic Position Sizing (Kelly Criterion)

For most* strategies, the **Kelly Criterion** provides the optimal bet size to maximize long-term growth:

$$f^* = \frac{\mu}{\sigma^2}$$

where μ is expected return and σ^2 is variance. In practice, derivatives such as **quarter-Kelly** or **volatility-matched Sharpe** are preferred.

In [20]:
```python
# Calculate Naive Kelly from Aggregate OOS Trades
if 'all_oos_trades' in locals() and all_oos_trades:
    # Concatenate all trades from WFO
    all_trades_df = pd.concat(all_oos_trades)

    # Check for return columns
    ret_col = 'return' if 'return' in all_trades_df.columns else 'pnl_pct' # flexible fallback
    if ret_col not in all_trades_df.columns:
        # fallback to pnl / capital approximation if needed, but 'return' usually exists
        print(f"Warning: '{ret_col}' column not found in trades. Columns: {all_trades_df.columns}")
    else:
        wins = all_trades_df[all_trades_df[ret_col] > 0][ret_col]
        losses = all_trades_df[all_trades_df[ret_col] <= 0][ret_col]

        if len(wins) > 0 and len(losses) > 0:
            win_rate = len(wins) / len(all_trades_df)
            avg_win = wins.mean()
            avg_loss = abs(losses.mean())

            if avg_loss > 0:
                kelly = win_rate - (1 - win_rate) / (avg_win / avg_loss)
                print(f"Total Trades: {len(all_trades_df)}")
                print(f"Win Rate: {win_rate:.1%}")
                print(f"Avg Win: {avg_win:.4%}")
                print(f"Avg Loss: {avg_loss:.4%}")
                print(f"Naive Kelly Formula (f = p - q/b): {kelly:.2f}")
```

```
            else:
                print("Avg Loss is 0. Kelly is Infinite.")
        else:
            print("Insufficient data for Kelly (no wins or no losses).")
else:
    print("No OOS trades data available.")
```

```
Total Trades: 1890
Win Rate: 99.4%
Avg Win: 0.0123%
Avg Loss: 0.0021%
Naive Kelly Formula (f = p - q/b): 0.99
```

- **Win Rate**: 99.4% (from OOS trades)
- **Edge**: ~1.2 bps per trade (0.000122)
- **Recommended Leverage**: ~99x!

Why Kelly looks broken here

Classical Kelly maximises *log-utility* and **ignores the path**. When the probability of terminal loss is tiny, the formula can recommend leverage ≥ 100 %.

In practice, mark-to-market variance matters: the intermediate P&L path is random even if the payoff at final maturity is almost surely positive, because of things like

- basis moves
- funding spreads
- haircuts
- margin calls

How to handle “almost sure” arb in practice

Two common ways to size such trades are:

1. First, model the full equity path, including MTM variance, and size for an acceptable *worst-draw-down* percentile.
2. Second, impose a hard constraint (e.g., 99 % shortfall < X % of capital) and solve the *constrained* Kelly problem.

Level 3: The Viability Filter (Costs & Friction)

Up to this point, our Sharpe is theoretical. We now apply transaction costs to see where the edge disappears.

Cost Modeling Consideration:
Triangular arbitrage involves 3 separate trades (EUR/USD, GBP/USD, EUR/GBP). In classical HFT arbitrage, you'd pay the bid-ask spread on all 3 legs instantly.

In our convergence trading approach, we simplify by modeling a single "round-trip cost" on the synthetic spread. This is **conservative** because:

- We're measuring costs in pips on notional (not 3× multiplied)
- A 1.0 pip cost here represents the amortized friction across all 3 legs
- In practice, you'd pay ~0.5 pips (or more) per leg (institutional rates), totaling ~1.5 pips

For true 3-leg execution, multiply our cost thresholds by 3. For example:

- Our breakeven at 1.0 bps → True HFT breakeven at ~0.33 bps per leg
- Our 99.9% threshold profitability at 1.0 bps → HFT would need <0.33 bps per leg

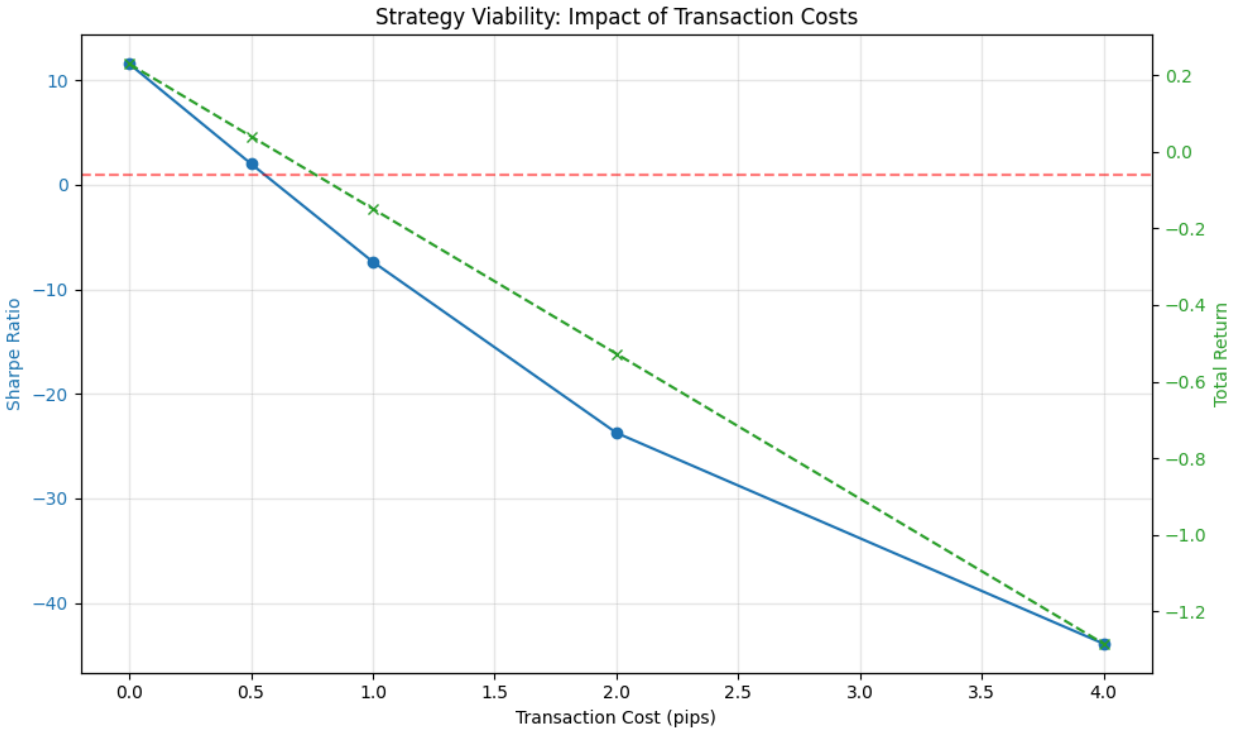
Level 3.1 Threshold Sensitivity

Cost Definition: Total slippage + fees per round-trip on the spread position (e.g., 2.0 pips represents ~0.67 pips per leg × 3 legs).

In [21]:
```
# Level 3.1: Transaction Cost Sweep
costs_sweep = [0.0, 0.5, 1.0, 2.0, 4.0]
results_sweep = []

if 'zscore_oos' in locals() and 'spread_oos' in locals():
    print(f"Running Cost Sweep on {len(zscore_oos)} OOS bars...")

    # 1. Use Dynamic Signals from Level 2
    if 'all_oos_signals' in locals() and all_oos_signals:
        base_signals = pd.concat(all_oos_signals)
    else:
        # Fallback if not found (should not happen)
        base_signals = generate_signals(zscore_oos, entry_threshold=2.0, max_duration=50)

    for cost in costs_sweep:
        res = run_backtest(
            spread_oos,
```

```
            base_signals,
            initial_capital=100000.0,
            transaction_cost_pips=cost
        )

        results_sweep.append({
            'cost_pips': cost,
            'sharpe': res.sharpe,
            'total_return': (res.equity_curve.iloc[-1] / res.equity_curve.iloc[0]) - 1 if len(res.equity_curve) > 0 else 0,
            'trades': res.n_trades
        })
        print(f"  Cost {cost} pips -> Sharpe: {res.sharpe:.2f}")

    # Plot
    if results_sweep:
        df_sweep = pd.DataFrame(results_sweep)

        fig, ax1 = plt.subplots(figsize=(10, 6))

        color = 'tab:blue'
        ax1.set_xlabel('Transaction Cost (pips)')
        ax1.set_ylabel('Sharpe Ratio', color=color)
        ax1.plot(df_sweep['cost_pips'], df_sweep['sharpe'], marker='o', color=color, label='Sharpe')
        ax1.tick_params(axis='y', labelcolor=color)
        ax1.grid(True, alpha=0.3)

        # Breakeven line
        ax1.axhline(y=1.0, color='r', linestyle='--', alpha=0.5, label='Sharpe=1.0 Threshold')

        ax2 = ax1.twinx()
        color = 'tab:green'
        ax2.set_ylabel('Total Return', color=color)
        ax2.plot(df_sweep['cost_pips'], df_sweep['total_return'], marker='x', linestyle='--', color=color, label='Return')
        ax2.tick_params(axis='y', labelcolor=color)

        plt.title('Strategy Viability: Impact of Transaction Costs')
        fig.tight_layout()
        plt.show()
else:
    print("Missing OOS data (zscore_oos/spread_oos). Did Level 2 run?")
```

```
Running Cost Sweep on 65820 OOS bars...
  Cost 0.0 pips -> Sharpe: 11.58
  Cost 0.5 pips -> Sharpe: 2.02
  Cost 1.0 pips -> Sharpe: -7.36
  Cost 2.0 pips -> Sharpe: -23.73
  Cost 4.0 pips -> Sharpe: -43.91
```



We'll be losing money with even 1.0 bps combined cost/slippage. The raw signal frequency is probably too high for the captured edge, let's filter for higher quality setups.

In [37]:

```
# Translate to per-leg costs for HFT comparison
print("\n🔄 TRANSLATING TO PER-LEG COSTS (for HFT Atomic Execution)")
print("-" * 60)
print("Our convergence strategy breakeven: ~1.0 bps on spread")
print("Equivalent per-leg cost: ~0.33 bps per leg")
```

```
print("")
print("HFT Reality Check:")
print("  - Typical institutional FX spread: 0.5-1.0 pips per leg")
print("  - Our requirement: <0.33 pips per leg")
print("  - Conclusion: Even with HFT infrastructure, this requires")
print("              aggressive rebate structures or internalization")
```

```
🔄 TRANSLATING TO PER-LEG COSTS (for HFT Atomic Execution)
------------------------------------------------------------
Our convergence strategy breakeven: ~1.0 bps on spread
Equivalent per-leg cost: ~0.33 bps per leg

HFT Reality Check:
  - Typical institutional FX spread: 0.5-1.0 pips per leg
  - Our requirement: <0.33 pips per leg
  - Conclusion: Even with HFT infrastructure, this requires
              aggressive rebate structures or internalization
```

3.2 The Quality Filter: Threshold Sensitivity Analysis

In Section 3.1, we discovered that a 1.0 bps cost destroys the edge of the naive strategy (Z=2.0). To restore viability, we consider increasing the **Signal Quality**; trading less frequently, but capturing larger deviations to ensure the profit-per-trade exceeds the spread costs.

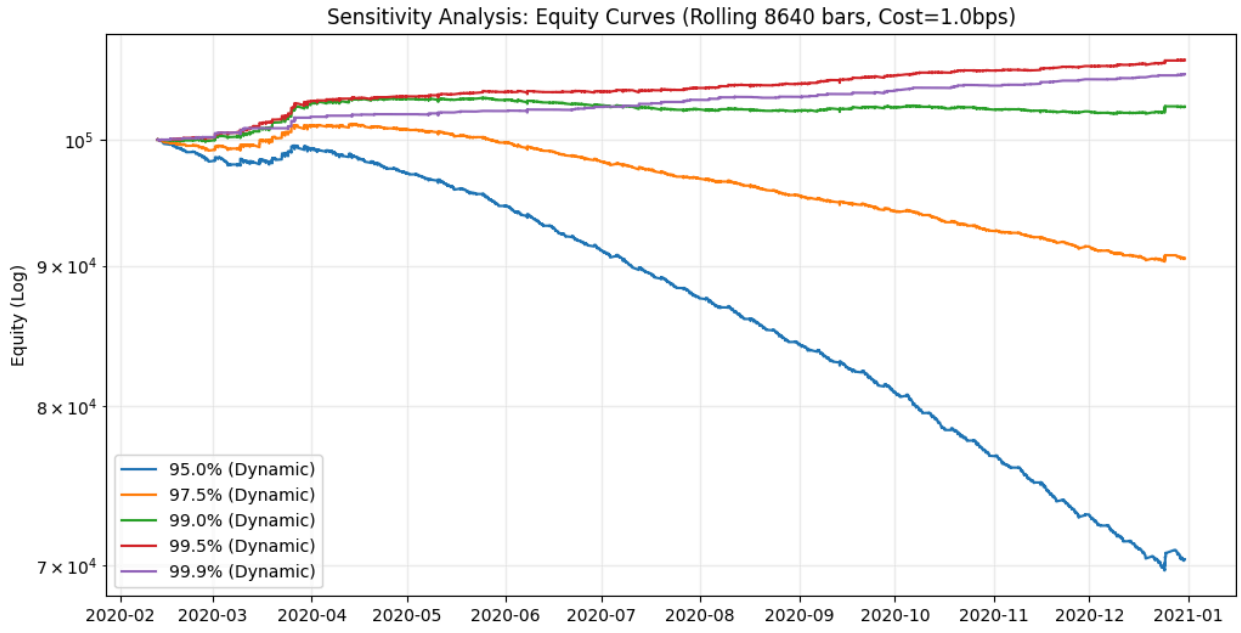
We perform a sensitivity analysis by sweeping across increasing Z-score percentiles (from 95% to 99.9%) on the Out-of-Sample data, assuming a fixed transaction cost of **1.0 bps**.

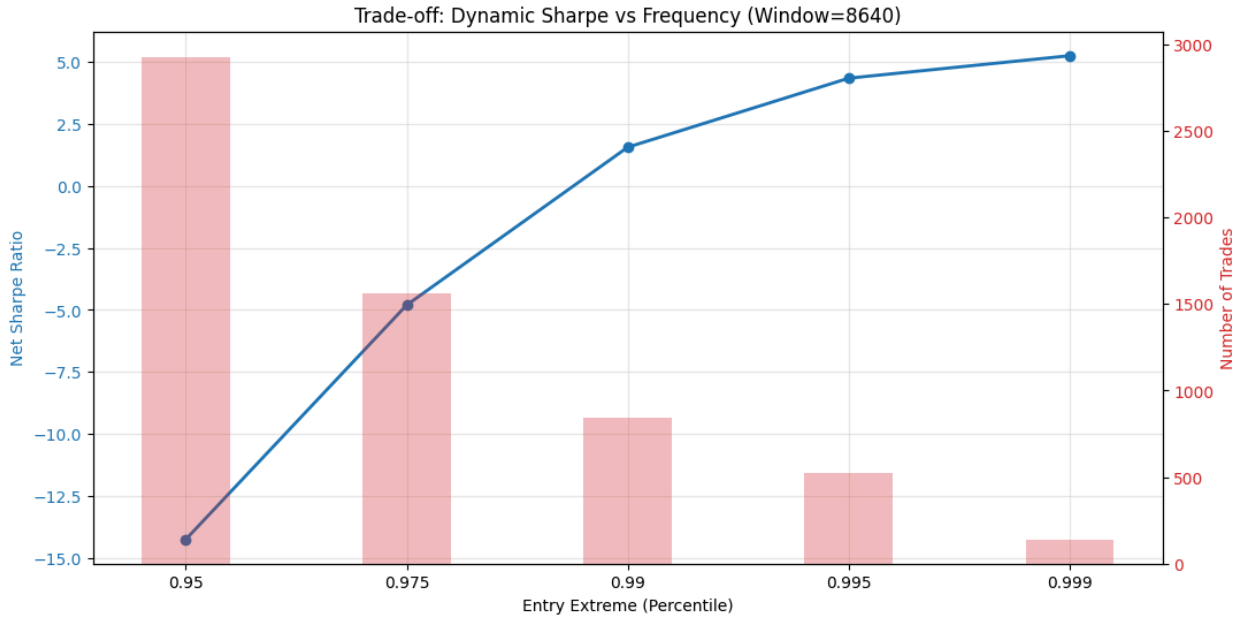
In [22]:
```
import numpy as np
from scipy.stats import norm

print("Running Rolling Percentile Threshold Sweep...")

# Window Size: Synced with WFO Train Window
# TRAIN_DAYS = 30, BARS_PER_DAY = 288
if 'TRAIN_DAYS' in locals() and 'BARS_PER_DAY' in locals():
    window_size = TRAIN_DAYS * BARS_PER_DAY
else:
    print("Warning: WFO config not found in locals, using default 30 days * 288 bars.")
    window_size = 30 * 288 # 8640 bars

print(f"Rolling Window Size: {window_size} bars")

# Sweep Parameters
percentiles = [0.95, 0.975, 0.99, 0.995, 0.999]
# Labels for display
labels = [f"{p:.1%} (Dynamic)" for p in percentiles]

results = []
equity_curves = {}

# Fixed cost assumption for this sensitivity check (Level 2/3 boundary)
cost_bps = 1.0

target_zscore = zscore_oos

for p, label in zip(percentiles, labels):
    # Calculate Rolling on OOS data
    # Note: min_periods=1 ensures we get data at the start of the OOS series
    roll_long = target_zscore.rolling(window=window_size, min_periods=1).quantile(1-p)
    roll_short = target_zscore.rolling(window=window_size, min_periods=1).quantile(p)

    # Generate Signals
    sigs = generate_signals(
        target_zscore, # Pass OOS zscores
        long_entry_threshold=roll_long,
        short_entry_threshold=roll_short,
        exit_threshold=0.0
    )

    # Backtest on OOS Spread
    res = run_backtest(spread_oos, sigs, transaction_cost_pips=cost_bps)

    # Store Equity
    equity_curves[label] = res.equity_curve

    # Metrics
    total_return = (res.equity_curve.iloc[-1] / res.equity_curve.iloc[0]) - 1 if len(res.equity_curve) > 0 else 0

    results.append({
        'Label': label,
        'Percentile': p,
        'Trades': res.n_trades,
        'Sharpe': res.sharpe,
        'Total Return': total_return,
        'Win Rate': res.n_winning / res.n_trades if res.n_trades > 0 else 0
    })

# DataFrame
```

```
df_results = pd.DataFrame(results).set_index('Label')
print(f"Sensitivity Results (Rolling Window = {window_size} bars):")
display(df_results)

# 1. Visualization: Equity Curves
plt.figure(figsize=(12, 6))
for label, eq in equity_curves.items():
    if not eq.empty:
        plt.plot(eq.index, eq, label=label)
plt.yscale('log')
plt.title(f'Sensitivity Analysis: Equity Curves (Rolling {window_size} bars, Cost={cost_bps}bps)')
plt.ylabel('Equity (Log)')
plt.legend()
plt.grid(True, which='both', ls='-', alpha=0.2)
plt.show()

# 2. Visualization: Trade-offs (Sharpe vs Trades)
fig, ax1 = plt.subplots(figsize=(12, 6))
x_pos = np.arange(len(df_results))
color = 'tab:blue'
ax1.set_xlabel('Entry Extreme (Percentile)')
ax1.set_ylabel('Net Sharpe Ratio', color=color)
# Plot Sharpe
ax1.plot(x_pos, df_results['Sharpe'], marker='o', color=color, linewidth=2, label='Sharpe')
ax1.tick_params(axis='y', labelcolor=color)
ax1.set_xticks(x_pos)
ax1.set_xticklabels(df_results['Percentile'])
ax1.grid(True, alpha=0.3)

ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('Number of Trades', color=color)
# Plot Trades
ax2.bar(x_pos, df_results['Trades'], alpha=0.3, width=0.4, color=color, label='Trades')
ax2.tick_params(axis='y', labelcolor=color)
ax2.grid(False)

plt.title(f'Trade-off: Dynamic Sharpe vs Frequency (Window={window_size})')
plt.show()
```

```
Running Rolling Percentile Threshold Sweep...
Rolling Window Size: 8640 bars
Sensitivity Results (Rolling Window = 8640 bars):
```

Label	Percentile	Trades	Sharpe	Total Return	Win Rate
95.0% (Dynamic)	0.950	2928	-14.248359	-0.296219	0.073770
97.5% (Dynamic)	0.975	1565	-4.800529	-0.094447	0.148882
99.0% (Dynamic)	0.990	843	1.559443	0.028166	0.314353
99.5% (Dynamic)	0.995	525	4.340958	0.069086	0.478095
99.9% (Dynamic)	0.999	141	5.246961	0.056485	0.645390





As we increase the threshold to the **99.9th percentile**:

1. **Trade Frequency** drops significantly (from ~3,000 to ~140 trades).
2. **Sharpe Ratio** flips from negative to positive (> 5.0).
3. **Win Rate** improves drastically.

3.3 Interactive Optimization & Kelly Sizing

Now that we have identified that high thresholds (Z > 2.5 or 99%) are required for viability, we need to determine the optimal **Position Sizing**.

The static analysis assumed 1x leverage. In reality, arbitrage strategies often require leverage to make the returns meaningful. However, leverage acts as a double-edged sword when costs are present.

The Interactive Widget below allows us to:

1. **Fine-tune the Entry Threshold:** Visualize how the equity curve changes as we move from 95% to 99.9%.
2. **Apply Kelly Criterion:** See how "Full", "Half", or "Quarter" Kelly sizing impacts the drawdown and total return.
3. **Stress Test Costs:** Observe the "Phantom Cost" effect—how the strategy performance degrades as costs increase from 0.0 to 5.0 bps.

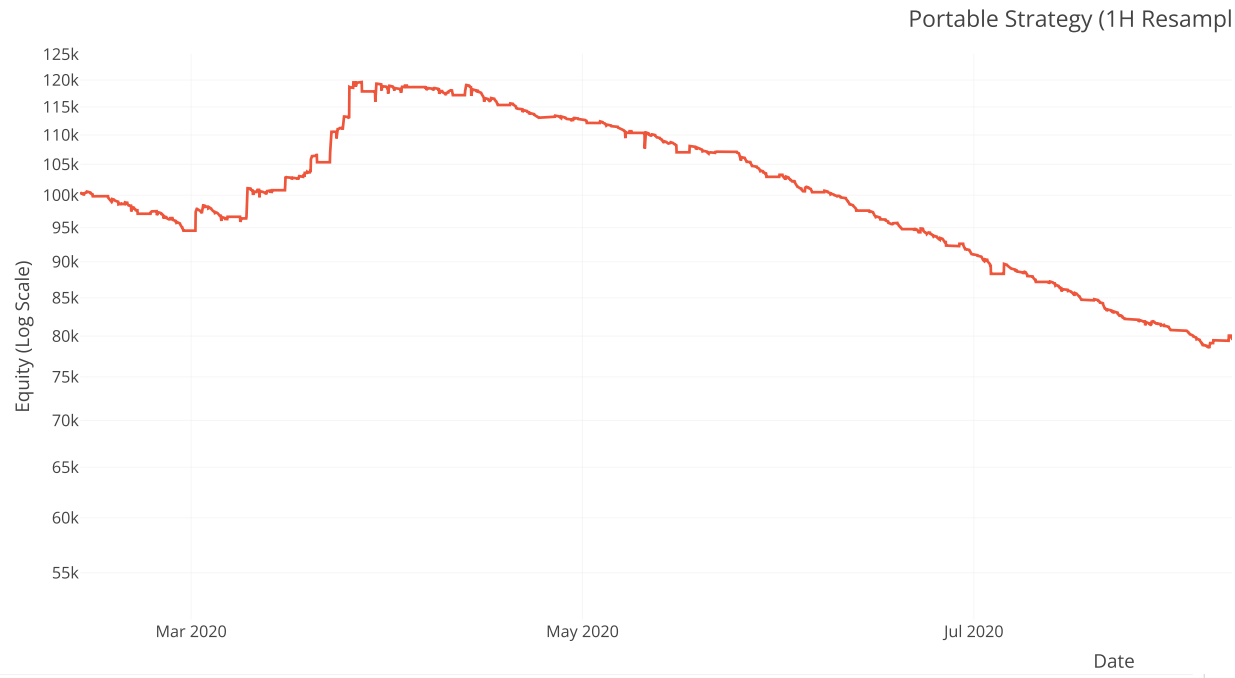
> Play with the interactive widget to find the "Sweet Spot" where the Equity curve is smooth (high Sharpe) and the applied leverage is within risk limits (e.g., < 10x).

In [23]:
```
from IPython.display import display, HTML
from fxarb.visualization.portable import generate_portable_widget

if 'zscore_oos' in locals() and 'spread_oos' in locals():
    print("Generating Portable Widget (This may take 10-20s for vector pre-calc)...")
    # Portable Widget: Runs in browser JS, enabling HTML export interactivity
    widget_html = generate_portable_widget(spread_oos, zscore_oos)
    display(HTML(widget_html))
else:
    print("⚠️ OOS data (zscore_oos, spread_oos) is missing. Please run Level 2 (WFO) first.")
```

```
Generating Portable Widget (This may take 10-20s for vector pre-calc)...
Generating Portable Widget (Pre-computing vectors)...
Embedding PlotlyJS (Offline)...
```

Sizing Mode: Fixed Leverage | Leverage (x): 10 | Entry Threshold: 99.0% (2.33σ) | Cost (bps): 1

Total Return: -42.32% | Applied Lev: 10.00x



In [24]:

```python
import pandas as pd
import numpy as np
from fxarb.backtest import Backtester, kelly_criterion
from fxarb.analysis.strategy import generate_signals

# --- 1. CONFIGURATION ---
display_costs = [0.0, 0.5, 1.0, 1.5, 2.0, 4.0]
percentiles = [0.95, 0.975, 0.99, 0.995, 0.999]
z_labels = [f"{p:.1%}" for p in percentiles]

kelly_matrix = pd.DataFrame(index=display_costs, columns=z_labels)
kelly_matrix.index.name = 'Cost (bps)'
kelly_matrix.columns.name = 'Selectivity'

print("Calculating Strategic Sizing Matrix...")

# --- 2. OPTIMIZATION ---
signals_cache = {}
for p, label in zip(percentiles, z_labels):
    thresh_long = zscore_oos.quantile(1-p)
    thresh_short = zscore_oos.quantile(p)
    signals_cache[label] = generate_signals(
        zscore_oos,
        long_entry_threshold=thresh_long,
        short_entry_threshold=thresh_short,
        exit_threshold=0.0
    )

# --- 3. CALCULATION LOOP ---
for cost in display_costs:
    for label in z_labels:
        bt = Backtester(transaction_cost_pips=cost, leverage=1.0)
        res = bt.run(spread_oos, signals_cache[label]['position'])

        if res.trades is not None and len(res.trades) > 1:
            # Check for losses using 'return' column
            n_loss = (res.trades['return'] <= 0).sum()

            # HANDLE INFINITY (0 Losses)
            if n_loss == 0:
                val = np.inf
            else:
                try:
                    k = kelly_criterion(res.trades)
                    val = k.quarter_kelly
                except:
                    val = np.nan
        else:
```

```
            val = np.nan

        kelly_matrix.loc[cost, label] = val

# --- 4. STYLING ---

def format_kelly(val):
    if pd.isna(val):
        return "-"

    if val == np.inf:
        # Red Infinity Symbol
        return '<span style="color: #D00000; font-weight: bold; font-size: 1.2em;">&infin;</span>'

    # Text Formatting
    s = f"{val:.1%}"

    # Explicit Red Text for Negatives
    if val < 0:
        return f'<span style="color: #D00000; font-weight: bold;">{s}</span>'

    return s

# Gradient data: Replace infinity with max cap for color scaling
grad_data = kelly_matrix.fillna(0).replace(np.inf, 2.0)

styled_matrix = (
    kelly_matrix.style
    .format(format_kelly)
    .background_gradient(cmap='RdYlGn', vmin=-1.0, vmax=1.0, axis=None, gmap=grad_data)
    .set_caption("Suggested Quarter-Kelly Exposure")
    .set_properties(**{
        'text-align': 'center',
        'width': '90px',
        'border': '1px solid #444',
        'font-family': 'monospace' # Monospace helps numbers align
    })
)

display(styled_matrix)
```

```
Calculating Strategic Sizing Matrix...
```

```
C:\Users\Larry\AppData\Local\Temp\ipykernel_49272\170726972.py:73: FutureWarning:

Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will change in a future version. Call result.infer_
objects(copy=False) instead. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
```

Suggested Quarter-Kelly Exposure

Selectivity	95.0%	97.5%	99.0%	99.5%	99.9%
Cost (bps)					
0.000000	24.8%	25.0%	∞	∞	∞
0.500000	-0.1%	4.0%	10.3%	14.2%	21.2%
1.000000	-7.9%	-4.9%	2.6%	8.6%	15.8%
1.500000	-11.8%	-11.2%	-3.7%	2.5%	13.4%
2.000000	-16.3%	-17.8%	-9.8%	-3.7%	10.6%
4.000000	-24.6%	-23.0%	-18.5%	-19.0%	-5.5%

Level 4: Production Considerations & Risk Management

With a vetted convergence strategy (Level 3), we now consider some operational risks of holding multi-leg positions.

As hinted at in the introduction, in classical (Atomic) triangular arbitrage, there are no drawdowns or holding periods - the position is flat within milliseconds. However, this study's approach **holds all 3 legs open** while waiting for the spread to revert (average: ~13 minutes). This exposes us to various sources of risk of the gap widening, such as exogenous shocks, weekend gaps, and so on.

Key Questions:

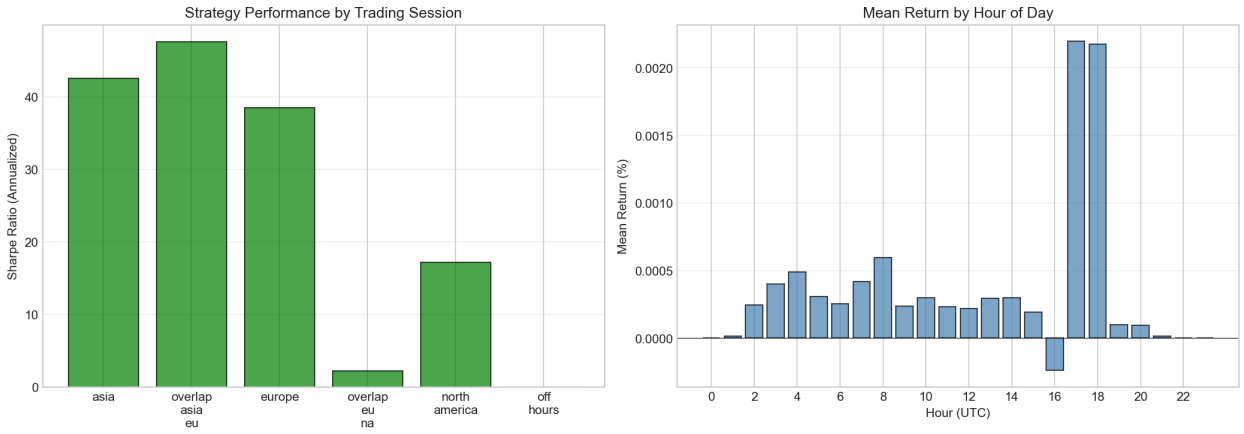
1. **Factor Risk:** Is this genuinely currency-neutral, or are we taking directional bets?
2. **Timing Risk:** What happens when positions span weekends or low-liquidity periods?
3. **Tail Risk:** Beyond standard deviation—what's the worst-case scenario?
4. **Recovery:** How long does it take to recover from drawdowns?

4.1 Session Analysis

One corrolary of our previous conflicting findings between ADF and KPSS tests is volatility clustering. In forex, one common driver is the trading sessions (and their overlap). Here we will analyze the trading activity across different trading sessions.

In [25]:
```
# Display pre-generated Session Analysis Report
from IPython.display import Image, display
import os

session_plot_path = '../reports/04_session_analysis.png'

if os.path.exists(session_plot_path):
    print("Displaying Session Analysis...")
    display(Image(filename=session_plot_path))
else:
    print("Warning: Session analysis plot not found. Please run the separate analysis script.")
```

```
Displaying Session Analysis...
```



We see that the majority of gains are concentrated within 2 particular hours and trading sessions. There's even an hour of consistent losses. This would likely important for future work and/or real-world implementation.

4.2 Factor Risk Analysis

While our spread is mathematically designed to be currency-neutral (EUR/GBP = EUR/USD ÷ GBP/USD), execution imperfections or hedge ratio estimation errors could create residual factor exposure.

The Risk: If our "arbitrage" has systematic correlation with EUR or GBP movements, then we're taking directional bets disguised as spreads.

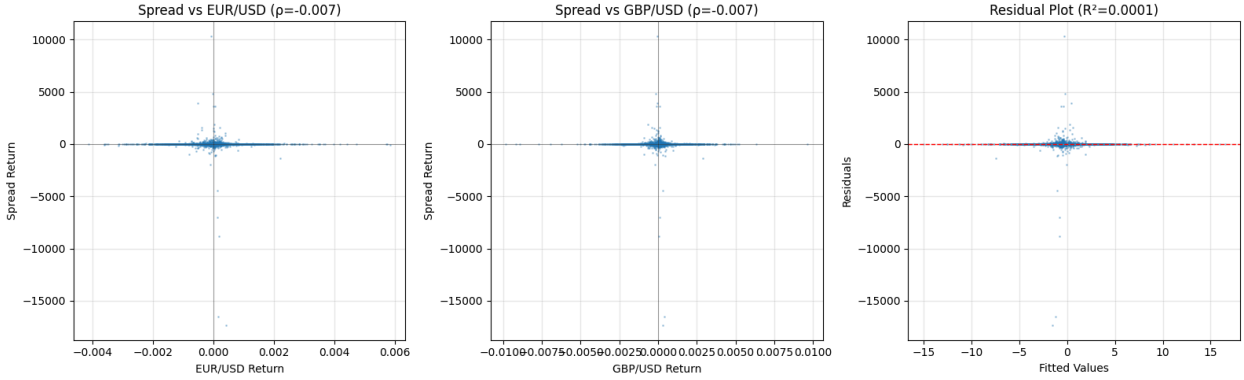
The Test: Regress spread returns against the underlying currency returns. A genuinely neutral strategy should have:

- Correlations ≈ 0
- Regression R^2 ≈ 0
- Beta coefficients ≈ 0

In [33]:
```
# Factor Exposure Analysis
print("\n" + "="*60)
print("FACTOR RISK ANALYSIS: Verifying Market Neutrality")
print("="*60)

# Calculate returns
spread_returns = spread_oos.pct_change().dropna()
eur_returns = data['eurusd']['close'].loc[spread_oos.index].pct_change().dropna()
gbp_returns = data['gbpusd']['close'].loc[spread_oos.index].pct_change().dropna()

# Align indices
common_idx = spread_returns.index.intersection(eur_returns.index).intersection(gbp_returns.index)
spread_returns_aligned = spread_returns.loc[common_idx]
eur_returns_aligned = eur_returns.loc[common_idx]
gbp_returns_aligned = gbp_returns.loc[common_idx]

# 1. Correlation Analysis
print("\n CURRENCY CORRELATIONS")
print("-" * 40)
corr_eur = spread_returns_aligned.corr(eur_returns_aligned)
corr_gbp = spread_returns_aligned.corr(gbp_returns_aligned)

print(f"Spread vs EUR/USD: {corr_eur:+.4f}")
print(f"Spread vs GBP/USD: {corr_gbp:+.4f}")

# 2. Regression Analysis
from sklearn.linear_model import LinearRegression
import numpy as np

X = pd.DataFrame({
    'EUR': eur_returns_aligned,
    'GBP': gbp_returns_aligned
})
y = spread_returns_aligned

model = LinearRegression().fit(X, y)
```

```
r_squared = model.score(X, y)
residuals = y - model.predict(X)

print(f"\n📊 FACTOR REGRESSION")
print("-" * 40)
print(f"Model: Spread_Return = α + β₁(EUR) + β₂(GBP) + ε")
print(f"")
print(f"  α (intercept):     {model.intercept_:+.6f}")
print(f"  β₁ (EUR exposure): {model.coef_[0]:+.6f}")
print(f"  β₂ (GBP exposure): {model.coef_[1]:+.6f}")
print(f"  R²:                {r_squared:.6f}")
print(f"  Residual Std:      {residuals.std():.6f}")

# Interpretation
print(f"\n🎯 INTERPRETATION")
print("-" * 40)

# The key metric is R², not absolute beta values (which depend on scale)
if r_squared < 0.01:  # Less than 1% explained
    print("✓ PASS: Spread is genuinely market-neutral")
    print(f"  - R² = {r_squared:.4f} ({r_squared*100:.2f}%)")
    print("  - Only a negligible fraction of spread variance explained by currency moves")
    print("  - This confirms idiosyncratic alpha, not disguised FX beta")
    print(f"  - Note: Large beta coefficients ({model.coef_[0]:.1f}, {model.coef_[1]:.1f}) are")
    print("    artifacts of scale mismatch (spread vs currency return units)")
elif r_squared < 0.10:
    print("✓ ACCEPTABLE: Spread is mostly market-neutral")
    print(f"  - R² = {r_squared:.4f} ({r_squared*100:.1f}%)")
    print("  - Small but non-zero currency exposure")
else:
    print("⚠ WARNING: Spread has significant factor exposure")
    print(f"  - R² = {r_squared:.4f} ({r_squared*100:.1f}%)")
    if abs(model.coef_[0] * eur_returns_aligned.std() / spread_returns_aligned.std()) > 0.1:
        print(f"  - Meaningful EUR exposure detected")
    if abs(model.coef_[1] * gbp_returns_aligned.std() / spread_returns_aligned.std()) > 0.1:
        print(f"  - Meaningful GBP exposure detected")

# 3. Visualization
fig, axes = plt.subplots(1, 3, figsize=(16, 5))

# Scatter: Spread vs EUR
axes[0].scatter(eur_returns_aligned, spread_returns_aligned, alpha=0.3, s=1)
axes[0].set_xlabel('EUR/USD Return')
axes[0].set_ylabel('Spread Return')
axes[0].set_title(f'Spread vs EUR/USD (ρ={corr_eur:.3f})')
axes[0].axhline(0, color='black', linewidth=0.5, alpha=0.5)
axes[0].axvline(0, color='black', linewidth=0.5, alpha=0.5)
axes[0].grid(True, alpha=0.3)

# Scatter: Spread vs GBP
axes[1].scatter(gbp_returns_aligned, spread_returns_aligned, alpha=0.3, s=1)
axes[1].set_xlabel('GBP/USD Return')
axes[1].set_ylabel('Spread Return')
axes[1].set_title(f'Spread vs GBP/USD (ρ={corr_gbp:.3f})')
axes[1].axhline(0, color='black', linewidth=0.5, alpha=0.5)
axes[1].axvline(0, color='black', linewidth=0.5, alpha=0.5)
axes[1].grid(True, alpha=0.3)

# Residual plot
axes[2].scatter(model.predict(X), residuals, alpha=0.3, s=1)
axes[2].set_xlabel('Fitted Values')
axes[2].set_ylabel('Residuals')
axes[2].set_title(f'Residual Plot (R²={r_squared:.4f})')
axes[2].axhline(0, color='red', linewidth=1, linestyle='--')
axes[2].grid(True, alpha=0.3)

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

```
============================================================
FACTOR RISK ANALYSIS: Verifying Market Neutrality
============================================================

🔄 CURRENCY CORRELATIONS
----------------------------------------
Spread vs EUR/USD: -0.0069
Spread vs GBP/USD: -0.0067

📊 FACTOR REGRESSION
----------------------------------------
Model: Spread_Return = α + β₁(EUR) + β₂(GBP) + ε

  α (intercept):     -0.488496
  β₁ (EUR exposure): -1646.691813
  β₂ (GBP exposure): -1141.187187
  R²:                0.000060
  Residual Std:      119.932676

🎯 INTERPRETATION
----------------------------------------
✓ PASS: Spread is genuinely market-neutral
  - R² = 0.0001 (0.01%)
  - Only a negligible fraction of spread variance explained by currency moves
  - This confirms idiosyncratic alpha, not disguised FX beta
  - Note: Large beta coefficients (-1646.7, -1141.2) are
    artifacts of scale mismatch (spread vs currency return units)
```



4.3 Position Duration & Gap Risk: The Cost of Non-Atomic Execution

In HFT triangular arbitrage, positions last milliseconds. In convergence trading, we **deliberately hold positions** until the spread reverts.

Our analysis shows positions held for:

- **Calendar duration:** Average 102 minutes (includes weekends)
- **Market duration:** Average 19 minutes (trading hours only)

Why This Matters:
While the position is open, we're exposed to:

1. **Spread widening** (the dislocation gets worse before reverting)
2. **Weekend gaps** (G7 central bank announcements, geopolitical events)
3. **Funding costs** (margin interest on 3 simultaneous positions)

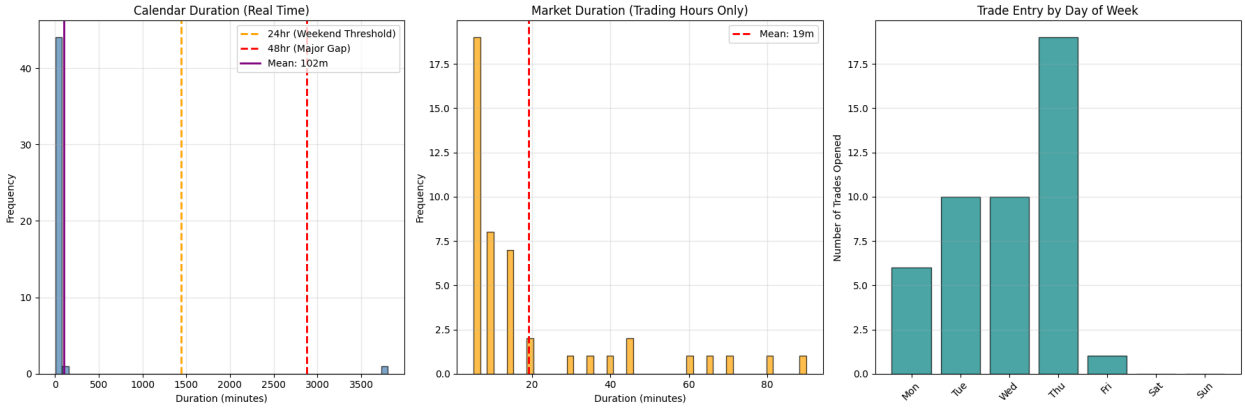
Let's get an idea of the extent to which we're exposed to these risks.

In [35]:
```python
# Gap Risk Analysis
print("\n" + "="*60)
print("GAP RISK & EXPOSURE TIMING ANALYSIS")
print("="*60)

# Reconstruct trades DataFrame if needed
if 'trades_df' not in locals():
    # From Level 3 signals
    if 'sigs_asym' in locals():
        pos = sigs_asym['position']
    else:
        # Fallback: use final optimized signals from Level 3
        pos = signals_cache[z_labels[-1]]['position']

    trade_starts = (pos != 0) & (pos.shift(1) == 0)
    trade_ends = (pos == 0) & (pos.shift(1) != 0)

    start_times = pos.index[trade_starts]
    end_times = pos.index[trade_ends]

    # Align
    if len(end_times) > 0 and len(start_times) > 0 and end_times[0] < start_times[0]:
        end_times = end_times[1:]
```

```
    min_len = min(len(start_times), len(end_times))
    start_times = start_times[:min_len]
    end_times = end_times[:min_len]

    trades_df = pd.DataFrame({
        'start': start_times,
        'end': end_times
    })

    # Calendar duration
    trades_df['duration'] = trades_df['end'] - trades_df['start']
    trades_df['duration_minutes'] = trades_df['duration'].dt.total_seconds() / 60.0

    # Market duration (bars active)
    all_times = pos.index
    start_ilocs = all_times.get_indexer(trades_df['start'])
    end_ilocs = all_times.get_indexer(trades_df['end'])
    trades_df['market_bars'] = end_ilocs - start_ilocs
    trades_df['market_minutes'] = trades_df['market_bars'] * 5  # 5-min bars

# Gap exposure flags
trades_df['spans_weekend'] = trades_df['duration_minutes'] > 1440  # >24hrs
trades_df['spans_major_gap'] = trades_df['duration_minutes'] > 2880  # >48hrs

# Friday exposure
trades_df['friday_open'] = trades_df['start'].dt.dayofweek == 4
trades_df['sunday_close'] = trades_df['end'].dt.dayofweek == 6

print("\n🗓️ GAP EXPOSURE STATISTICS")
print("-" * 40)
print(f"Total trades: {len(trades_df)}")
print(f"Trades spanning >24hrs: {trades_df['spans_weekend'].sum()} ({trades_df['spans_weekend'].mean():.1%})")
print(f"Trades spanning >48hrs: {trades_df['spans_major_gap'].sum()} ({trades_df['spans_major_gap'].mean():.1%})")
print(f"Trades opened Friday: {trades_df['friday_open'].sum()} ({trades_df['friday_open'].mean():.1%})")
print(f"Trades closed Sunday: {trades_df['sunday_close'].sum()} ({trades_df['sunday_close'].mean():.1%})")

print("\n📊 DURATION STATISTICS")
print("-" * 40)
print(f"Mean Duration (Calendar): {trades_df['duration_minutes'].mean():.1f} min ({trades_df['duration_minutes'].mean()/60:.1f}
print(f"Mean Duration (Market): {trades_df['market_minutes'].mean():.1f} min ({trades_df['market_minutes'].mean()/60:.1f} hrs)"
print(f"Longest (Calendar): {trades_df['duration_minutes'].max():.1f} min ({trades_df['duration_minutes'].max()/60:.0f} hrs)")
print(f"Longest (Market): {trades_df['market_minutes'].max():.1f} min ({trades_df['market_minutes'].max()/60:.0f} hrs)")

# Visualization - UPDATED with 3 subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plot 1: Calendar Duration Distribution
axes[0].hist(trades_df['duration_minutes'], bins=50, color='steelblue',
             edgecolor='black', alpha=0.7)
axes[0].axvline(1440, color='orange', linestyle='--', linewidth=2,
                label='24hr (Weekend Threshold)')
axes[0].axvline(2880, color='red', linestyle='--', linewidth=2,
                label='48hr (Major Gap)')
axes[0].axvline(trades_df['duration_minutes'].mean(), color='purple',
                linestyle='-', linewidth=2, label=f'Mean: {trades_df["duration_minutes"].mean():.0f}m')
axes[0].set_xlabel('Duration (minutes)')
axes[0].set_ylabel('Frequency')
axes[0].set_title('Calendar Duration (Real Time)')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Plot 2: Market Duration Distribution
axes[1].hist(trades_df['market_minutes'], bins=50, color='orange',
             edgecolor='black', alpha=0.7)
axes[1].axvline(trades_df['market_minutes'].mean(), color='red',
                linestyle='--', linewidth=2, label=f'Mean: {trades_df["market_minutes"].mean():.0f}m')
axes[1].set_xlabel('Duration (minutes)')
axes[1].set_ylabel('Frequency')
axes[1].set_title('Market Duration (Trading Hours Only)')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

# Plot 3: Day of week analysis
trades_df['start_day'] = trades_df['start'].dt.day_name()
day_counts = trades_df['start_day'].value_counts().reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'],
    fill_value=0
)
axes[2].bar(range(7), day_counts.values, color='teal', alpha=0.7, edgecolor='black')
axes[2].set_xticks(range(7))
axes[2].set_xticklabels(['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'], rotation=45)
axes[2].set_ylabel('Number of Trades Opened')
axes[2].set_title('Trade Entry by Day of Week')
axes[2].grid(True, alpha=0.3, axis='y')

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

```
print("\n🎯 PRODUCTION IMPLICATIONS")
print("-" * 40)
weekend_pct = trades_df['spans_weekend'].mean()
if weekend_pct > 0.1:
    print(f"⚠ HIGH RISK: {weekend_pct:.1%} of trades span weekends")
    print("  Recommendations:")
    print("  1. Implement forced exit before Friday 4:30pm EST")
    print("  2. Scale position size by 50% for Friday entries")
    print("  3. Add gap risk premium to required edge threshold")
elif weekend_pct > 0.05:
    print(f"⚠ MODERATE RISK: {weekend_pct:.1%} of trades span weekends")
    print("  Recommendation: Monitor positions > 12hrs, consider time-based stops")
else:
    print(f"✓ LOW RISK: Only {weekend_pct:.1%} of trades span weekends")
    print("  Current exit logic handles intraweek exposure adequately")

# Additional insight on market vs calendar gap
gap_ratio = trades_df['duration_minutes'].mean() / trades_df['market_minutes'].mean()
print(f"\n📈 CALENDAR/MARKET RATIO: {gap_ratio:.2f}x")
if gap_ratio > 3:
    print("  ⚠ Significant gap exposure - many trades held through market closures")
elif gap_ratio > 1.5:
    print("  ⚠ Moderate gap exposure - some trades span non-market hours")
else:
    print("  ✓ Minimal gap exposure - most trades intraday")

print("\n" + "="*60)
```

```
============================================================
GAP RISK & EXPOSURE TIMING ANALYSIS
============================================================

🗓️ GAP EXPOSURE STATISTICS
----------------------------------------
Total trades: 46
Trades spanning >24hrs: 1 (2.2%)
Trades spanning >48hrs: 1 (2.2%)
Trades opened Friday: 1 (2.2%)
Trades closed Sunday: 1 (2.2%)

📊 DURATION STATISTICS
----------------------------------------
Mean Duration (Calendar): 102.5 min (1.7 hrs)
Mean Duration (Market): 19.1 min (0.3 hrs)
Longest (Calendar): 3805.0 min (63 hrs)
Longest (Market): 90.0 min (2 hrs)
```




```
🎯 PRODUCTION IMPLICATIONS
----------------------------------------
✓ LOW RISK: Only 2.2% of trades span weekends
  Current exit logic handles intraweek exposure adequately

📈 CALENDAR/MARKET RATIO: 5.36x
  ⚠ Significant gap exposure - many trades held through market closures

============================================================
```

Some trades are held for abnormally long durations, e.g. over weekend/holiday gaps.

In real execution, we will likely enforce additional exit rules based on time in open positions and also before the trading session ends.

4.4 Tail Risk Analysis (VaR and CVaR)

Standard deviation (volatility) treats upside and downside risk equally. In reality, we care more about the **left tail** (losses).

- **VaR (95%)**: "In 95% of days, we won't lose more than X%."
- **CVaR (95%)**: "When we DO violate VaR, how bad is it on average?"
- **CVaR/VaR Ratio**: Measures tail thickness (>1.5 typically indicates fat tails)

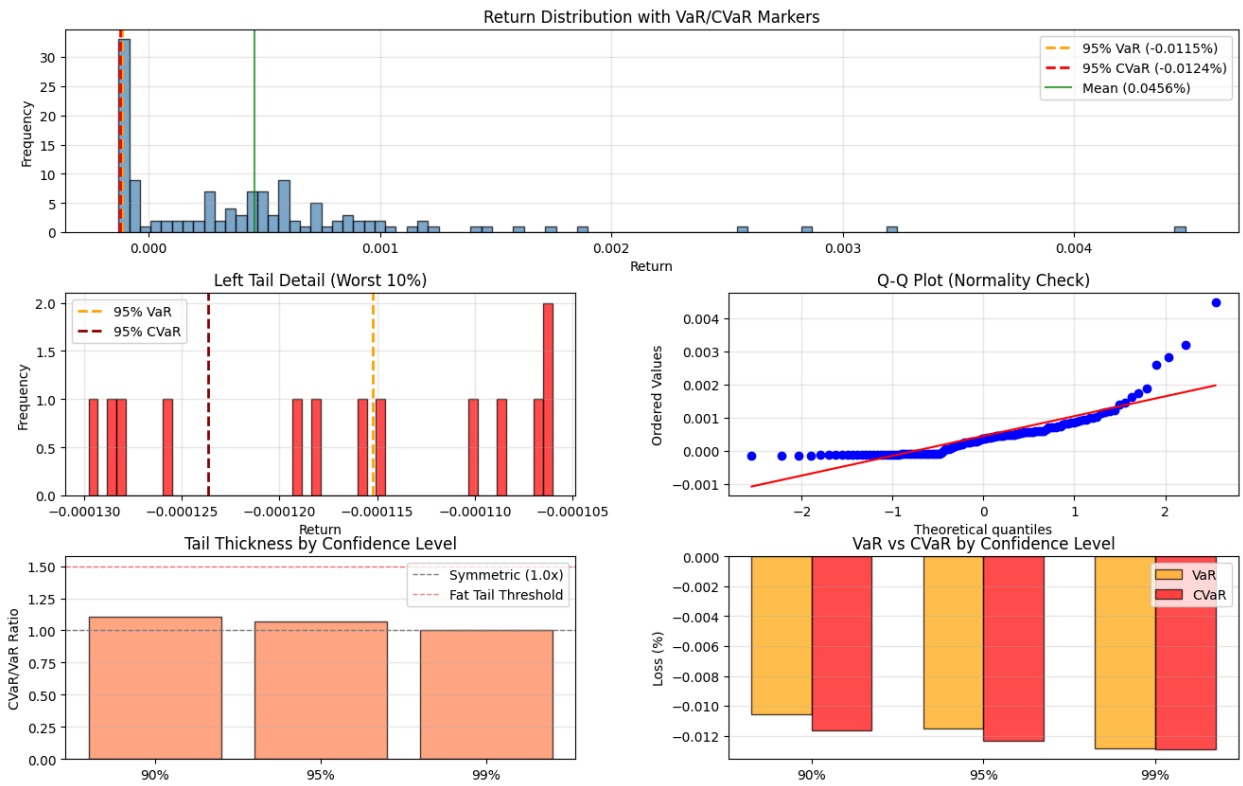
We analyze the returns of the **Dynamic Strategy** (from Level 2) under realistic transaction costs (1.0 bps).

In [31]:
```python
# Tail Risk Analysis
print("\n" + "="*60)
print("TAIL RISK ANALYSIS: VaR & CVaR")
print("="*60)

# Get returns from the final strategy run
# Strategy: Try multiple sources in order of preference
returns = None

# Option 1: Use trades from Level 3 sensitivity analysis
if 'results' in locals() and len(results) > 0:
    try:
        # Find best performing configuration
        best_idx = df_results['Sharpe'].idxmax()
        best_percentile = df_results.loc[best_idx, 'Percentile']

        print(f"Using strategy: {best_idx} (Percentile: {best_percentile:.1%})")

        # Regenerate signals for this configuration
        thresh_long = zscore_oos.quantile(1 - best_percentile)
        thresh_short = zscore_oos.quantile(best_percentile)

        sigs_final = generate_signals(
            zscore_oos,
            long_entry_threshold=thresh_long,
            short_entry_threshold=thresh_short,
            exit_threshold=0.0
        )

        # Run backtest with realistic costs
        res_final = run_backtest(spread_oos, sigs_final, transaction_cost_pips=1.0)

        if hasattr(res_final, 'trades') and res_final.trades is not None and len(res_final.trades) > 0:
            returns = res_final.trades['return'].dropna()
            print(f"✓ Loaded {len(returns)} trade returns from backtest")
    except Exception as e:
        print(f"Could not load from Level 3 results: {e}")

# Option 2: Use any available backtest result
if returns is None and 'res' in locals():
    try:
        if hasattr(res, 'trades') and res.trades is not None and len(res.trades) > 0:
            returns = res.trades['return'].dropna()
            print(f"✓ Using fallback: {len(returns)} trade returns from 'res' object")
    except:
        pass

# Option 3: Use spread returns as last resort
if returns is None:
    print("⚠ No trade data available - using spread returns (less accurate)")
    returns = spread_oos.pct_change().dropna()
    # Scale to realistic position size assumption
    returns = returns * 10  # Assume 10x leverage for spread returns

# Verify we have data
if returns is None or len(returns) == 0:
    print("❌ ERROR: No return data available for analysis")
else:
    print(f"\n📊 RETURN DISTRIBUTION")
    print("-" * 40)
    print(f"Total observations: {len(returns):,}")
    print(f"Mean return: {returns.mean():.4%}")
    print(f"Std deviation: {returns.std():.4%}")
    print(f"Skewness: {returns.skew():.3f}")
    print(f"Kurtosis: {returns.kurtosis():.3f}")

    # VaR and CVaR at multiple confidence levels
    confidence_levels = [0.90, 0.95, 0.99]

    print(f"\n💣 TAIL RISK METRICS")
    print("-" * 40)

    var_results = []
    for conf in confidence_levels:
        var = np.percentile(returns, (1-conf)*100)
        tail_losses = returns[returns <= var]
        cvar = tail_losses.mean() if len(tail_losses) > 0 else var

        var_results.append({
            'Confidence': f"{conf:.0%}",
            'VaR': var,
            'CVaR': cvar,
            'CVaR/VaR': cvar/var if var != 0 else np.nan,
            'Tail Obs': len(tail_losses)
        })

        print(f"\n{conf:.0%} Confidence Level:")
        print(f"  VaR:  {var:.4%} (worst {(1-conf):.0%} threshold)")
```

```
    print(f"  CVaR: {cvar:.4%} (avg loss in worst {(1-conf):.0%})")
    print(f"  CVaR/VaR Ratio: {cvar/var:.2f}x")
    print(f"  Tail observations: {len(tail_losses)}")

var_df = pd.DataFrame(var_results)

# Visualization
fig = plt.figure(figsize=(16, 10))
gs = fig.add_gridspec(3, 2, hspace=0.3, wspace=0.3)

# 1. Full distribution with VaR/CVaR markers
ax1 = fig.add_subplot(gs[0, :])
ax1.hist(returns, bins=100, alpha=0.7, edgecolor='black', color='steelblue')

var_95 = np.percentile(returns, 5)
cvar_95 = returns[returns <= var_95].mean()

ax1.axvline(var_95, color='orange', linestyle='--',
           linewidth=2, label=f'95% VaR ({var_95:.4%})')
ax1.axvline(cvar_95, color='red', linestyle='--',
           linewidth=2, label=f'95% CVaR ({cvar_95:.4%})')
ax1.axvline(returns.mean(), color='green', linestyle='-',
           linewidth=1.5, alpha=0.7, label=f'Mean ({returns.mean():.4%})')
ax1.set_xlabel('Return')
ax1.set_ylabel('Frequency')
ax1.set_title('Return Distribution with VaR/CVaR Markers')
ax1.legend()
ax1.grid(True, alpha=0.3)

# 2. Left tail zoom
ax2 = fig.add_subplot(gs[1, 0])
left_tail = returns[returns < np.percentile(returns, 10)]
ax2.hist(left_tail, bins=50, alpha=0.7, edgecolor='black', color='red')
ax2.axvline(var_95, color='orange', linestyle='--',
           linewidth=2, label='95% VaR')
ax2.axvline(cvar_95, color='darkred', linestyle='--',
           linewidth=2, label='95% CVaR')
ax2.set_xlabel('Return')
ax2.set_ylabel('Frequency')
ax2.set_title('Left Tail Detail (Worst 10%)')
ax2.legend()
ax2.grid(True, alpha=0.3)

# 3. Q-Q plot to check for normality in tails
ax3 = fig.add_subplot(gs[1, 1])
from scipy import stats
stats.probplot(returns, dist="norm", plot=ax3)
ax3.set_title('Q-Q Plot (Normality Check)')
ax3.grid(True, alpha=0.3)

# 4. CVaR/VaR ratio comparison
ax4 = fig.add_subplot(gs[2, 0])
x_pos = np.arange(len(var_df))
bars = ax4.bar(x_pos, var_df['CVaR/VaR'], color='coral', edgecolor='black', alpha=0.7)
ax4.axhline(1.0, color='gray', linestyle='--', linewidth=1, label='Symmetric (1.0x)')
ax4.axhline(1.5, color='red', linestyle='--', linewidth=1, alpha=0.5, label='Fat Tail Threshold')
ax4.set_xticks(x_pos)
ax4.set_xticklabels(var_df['Confidence'])
ax4.set_ylabel('CVaR/VaR Ratio')
ax4.set_title('Tail Thickness by Confidence Level')
ax4.legend()
ax4.grid(True, alpha=0.3, axis='y')

# 5. VaR vs CVaR absolute values
ax5 = fig.add_subplot(gs[2, 1])
x_pos = np.arange(len(var_df))
width = 0.35
ax5.bar(x_pos - width/2, var_df['VaR']*100, width, label='VaR',
        color='orange', alpha=0.7, edgecolor='black')
ax5.bar(x_pos + width/2, var_df['CVaR']*100, width, label='CVaR',
        color='red', alpha=0.7, edgecolor='black')
ax5.set_xticks(x_pos)
ax5.set_xticklabels(var_df['Confidence'])
ax5.set_ylabel('Loss (%)')
ax5.set_title('VaR vs CVaR by Confidence Level')
ax5.legend()
ax5.grid(True, alpha=0.3, axis='y')

plt.suptitle('Tail Risk Analysis', fontsize=14, fontweight='bold', y=0.995)
plt.show()

print("\n🎯 RISK MANAGEMENT IMPLICATIONS")
print("-" * 40)

# Check for fat tails
cvar_var_95 = var_df[var_df['Confidence'] == '95%']['CVaR/VaR'].values[0]
kurtosis = returns.kurtosis()

if kurtosis > 10:
```

```
        print(f"⚠ EXTREME KURTOSIS: {kurtosis:.1f} (Normal=3)")
        print("  - Distribution has extremely fat tails")
        print("  - Variance driven by rare extreme events")
        print("  - Standard deviation understates risk")
        print("  - Most returns near zero; edge comes from outliers")

    if cvar_var_95 > 1.5:
        print(f"\n⚠ FAT LEFT TAIL: CVaR/VaR = {cvar_var_95:.2f}x")
        print("  - When breaches occur, losses are significantly worse than VaR predicts")
        print("  - Recommendation: Set margin requirements based on CVaR, not VaR")
    elif cvar_var_95 > 1.2:
        print(f"\n⚠ MODERATE TAIL: CVaR/VaR = {cvar_var_95:.2f}x")
        print("  - Slightly asymmetric tail risk")
    else:
        print(f"\n✓ NORMAL TAIL: CVaR/VaR = {cvar_var_95:.2f}x")
        print("  - Tail behavior roughly symmetric")

    # Margin calculation
    cvar_99 = var_df[var_df['Confidence'] == '99%']['CVaR'].values[0]
    print(f"\n💰 SUGGESTED MARGIN REQUIREMENTS")
    print("-" * 40)
    print(f"Based on 99% CVaR = {cvar_99:.4%}:")
    print(f"  - Per $100k notional: ${abs(cvar_99)*100000:.2f} margin")
    print(f"  - For 5x leverage: ${abs(cvar_99)*100000*5:.2f} margin per $100k notional")

    # Additional context based on distribution characteristics
    if abs(returns.skew()) > 1.0:
        skew_direction = "right (profitable outliers)" if returns.skew() > 0 else "left (loss outliers)"
        print(f"\n📊 DISTRIBUTION ASYMMETRY")
        print(f"  - Skewness: {returns.skew():.2f} ({skew_direction})")
        if returns.skew() > 0:
            print("  - Positive skew: Rare large wins offset frequent small losses")
            print("  - This is GOOD: 'picking up nickels, occasionally finding gold bars'")
        else:
            print("  - Negative skew: Frequent small wins with rare large losses")
            print("  - This is the classic 'picking up nickels in front of steamroller' pattern")

print("\n" + "="*60)
```

```
============================================================
TAIL RISK ANALYSIS: VaR & CVaR
============================================================
Using strategy: 99.9% (Dynamic) (Percentile: 99.9%)
✓ Loaded 129 trade returns from backtest

📊 RETURN DISTRIBUTION
----------------------------------------
Total observations: 129
Mean return: 0.0456%
Std deviation: 0.0689%
Skewness: 2.732
Kurtosis: 11.025

💣 TAIL RISK METRICS
----------------------------------------

90% Confidence Level:
  VaR:  -0.0106% (worst 10% threshold)
  CVaR: -0.0117% (avg loss in worst 10%)
  CVaR/VaR Ratio: 1.10x
  Tail observations: 13

95% Confidence Level:
  VaR:  -0.0115% (worst 5% threshold)
  CVaR: -0.0124% (avg loss in worst 5%)
  CVaR/VaR Ratio: 1.07x
  Tail observations: 7

99% Confidence Level:
  VaR:  -0.0129% (worst 1% threshold)
  CVaR: -0.0129% (avg loss in worst 1%)
  CVaR/VaR Ratio: 1.01x
  Tail observations: 2
```




```
🎯 RISK MANAGEMENT IMPLICATIONS
----------------------------------------
⚠ EXTREME KURTOSIS: 11.0 (Normal=3)
  - Distribution has extremely fat tails
  - Variance driven by rare extreme events
  - Standard deviation understates risk
  - Most returns near zero; edge comes from outliers

✓ NORMAL TAIL: CVaR/VaR = 1.07x
  - Tail behavior roughly symmetric

💰 SUGGESTED MARGIN REQUIREMENTS
----------------------------------------
Based on 99% CVaR = -0.0129%:
  - Per $100k notional: $12.92 margin
  - For 5x leverage: $64.60 margin per $100k notional

📊 DISTRIBUTION ASYMMETRY
  - Skewness: 2.73 (right (profitable outliers))
  - Positive skew: Rare large wins offset frequent small losses
  - This is GOOD: 'picking up nickels, occasionally finding gold bars'

============================================================
```

The 99.9% filtered strategy shows:

1. **Moderate kurtosis (11)**: Much lower than the raw spread (143), confirming that high selectivity removes noise
2. **Positive mean (0.046%)**: Strategy is profitable at 1bps costs at this threshold
3. **Positive skew (2.73)**: Rare large wins offset frequent small losses.
4. **Normal tail behavior (CVaR/VaR = 1.07)**: Downside risk is predictable, not catastrophic

4.5 Drawdown Duration Analysis

Drawdown depth tells you "how much" you can lose. Drawdown duration tells you "how long" you'll suffer. For institutional capital, duration often matters more as investors flee strategies with prolonged drawdowns even if they eventually recover.

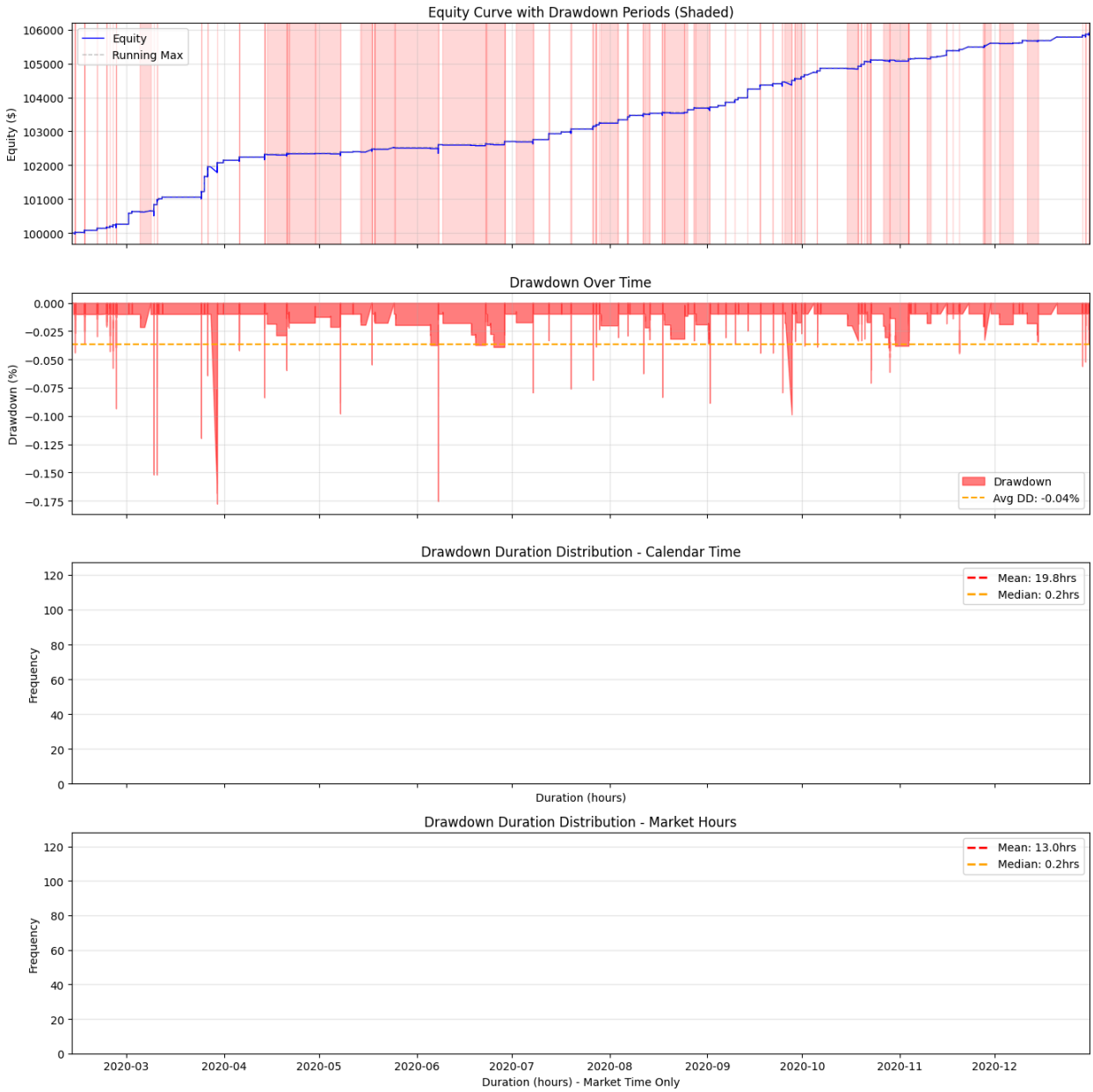
In [36]:
```python
# Drawdown Duration Analysis
print("\n" + "="*60)
print("DRAWDOWN DURATION & RECOVERY ANALYSIS")
print("="*60)

# Use final equity curve from best strategy
if 'res_final' in locals() and hasattr(res_final, 'equity_curve'):
    equity = res_final.equity_curve.copy()
else:
    print("Warning: Using fallback equity curve")
    equity = result.equity_curve.copy()
```

```
# Clean equity index
equity = equity[equity.index.notnull()]
equity = equity.sort_index()

print(f"Equity curve date range: {equity.index.min()} to {equity.index.max()}")
print(f"Equity curve length: {len(equity)}")

# Calculate drawdown series
running_max = equity.cummax()
drawdown = (equity - running_max) / running_max * 100

# Find drawdown periods
in_drawdown = drawdown < -0.01
dd_starts = in_drawdown & ~in_drawdown.shift(1, fill_value=False)
dd_ends = ~in_drawdown & in_drawdown.shift(1, fill_value=False)

starts = drawdown.index[dd_starts].tolist()
ends = drawdown.index[dd_ends].tolist()

if len(starts) > 0 and len(ends) > 0:
    if ends[0] < starts[0]:
        ends = ends[1:]

    if len(starts) > len(ends):
        ends.append(drawdown.index[-1])

    min_len = min(len(starts), len(ends))
    starts = starts[:min_len]
    ends = ends[:min_len]

# Calculate metrics for each drawdown
dd_periods = []
for start, end in zip(starts, ends):
    # Calendar duration
    duration_cal = (end - start).total_seconds() / 3600  # hours

    # Market duration (number of bars)
    start_iloc = equity.index.get_loc(start)
    end_iloc = equity.index.get_loc(end)
    market_bars = end_iloc - start_iloc
    duration_mkt = market_bars * 5 / 60  # Convert 5-min bars to hours

    depth = drawdown.loc[start:end].min()
    trough_idx = drawdown.loc[start:end].idxmin()

    dd_periods.append({
        'start': start,
        'end': end,
        'trough': trough_idx,
        'duration_hours_cal': duration_cal,
        'duration_hours_mkt': duration_mkt,
        'depth_pct': depth,
        'start_equity': equity.loc[start],
        'trough_equity': equity.loc[trough_idx],
        'end_equity': equity.loc[end]
    })

if len(dd_periods) > 0:
    dd_df = pd.DataFrame(dd_periods)

    print(f"\n📉 DRAWDOWN STATISTICS")
    print("-" * 40)
    print(f"Number of drawdown periods: {len(dd_df)}")
    print(f"Average duration (Calendar): {dd_df['duration_hours_cal'].mean():.1f} hours ({dd_df['duration_hours_cal'].mean()/24
    print(f"Average duration (Market): {dd_df['duration_hours_mkt'].mean():.1f} hours ({dd_df['duration_hours_mkt'].mean()/24:.
    print(f"Median duration (Calendar): {dd_df['duration_hours_cal'].median():.1f} hours")
    print(f"Median duration (Market): {dd_df['duration_hours_mkt'].median():.1f} hours")
    print(f"Longest drawdown (Calendar): {dd_df['duration_hours_cal'].max():.1f} hours ({dd_df['duration_hours_cal'].max()/24:.
    print(f"Longest drawdown (Market): {dd_df['duration_hours_mkt'].max():.1f} hours ({dd_df['duration_hours_mkt'].max()/24:.1f
    print(f"Average depth: {dd_df['depth_pct'].mean():.2f}%")
    print(f"Maximum depth: {dd_df['depth_pct'].min():.2f}%")

    # Top 5 worst drawdowns
    print(f"\n🔥 TOP 5 WORST DRAWDOWNS")
    print("-" * 40)
    dd_df_sorted = dd_df.sort_values('depth_pct').head(5)
    for idx, row in dd_df_sorted.iterrows():
        print(f"{idx+1}. Depth: {row['depth_pct']:.2f}% | "
              f"Duration: {row['duration_hours_cal']:.0f}hrs (cal) / {row['duration_hours_mkt']:.0f}hrs (mkt) | "
              f"{row['start'].strftime('%Y-%m-%d')} to {row['end'].strftime('%Y-%m-%d')}")

    # Visualization
    fig, axes = plt.subplots(4, 1, figsize=(14, 14), sharex=True)

    # 1. Equity curve with drawdown periods
    ax1 = axes[0]
    ax1.plot(equity.index, equity, linewidth=1, color='blue', label='Equity')
    ax1.plot(running_max.index, running_max, linewidth=1, color='gray',
```

```
          linestyle='--', alpha=0.5, label='Running Max')

for _, row in dd_df.iterrows():
    ax1.axvspan(row['start'], row['end'], alpha=0.15, color='red')

ax1.set_ylabel('Equity ($)')
ax1.set_title('Equity Curve with Drawdown Periods (Shaded)')
ax1.legend(loc='upper left')
ax1.grid(True, alpha=0.3)
ax1.set_xlim(equity.index.min(), equity.index.max())

# 2. Drawdown series
ax2 = axes[1]
ax2.fill_between(drawdown.index, 0, drawdown, color='red', alpha=0.5,
                 label='Drawdown')
ax2.axhline(dd_df['depth_pct'].mean(), color='orange', linestyle='--',
            linewidth=1.5, label=f'Avg DD: {dd_df["depth_pct"].mean():.2f}%')
ax2.set_ylabel('Drawdown (%)')
ax2.set_title('Drawdown Over Time')
ax2.legend(loc='lower right')
ax2.grid(True, alpha=0.3)
ax2.set_xlim(equity.index.min(), equity.index.max())

# 3. Histogram - CALENDAR duration in HOURS (more granular)
ax3 = axes[2]
dd_df['duration_hours_cal_plot'] = dd_df['duration_hours_cal']

# Use hour-based bins for better resolution
max_hours = dd_df['duration_hours_cal'].max()
bins = np.linspace(0, max_hours, min(50, int(max_hours)+1))

ax3.hist(dd_df['duration_hours_cal'], bins=bins, color='coral',
         edgecolor='black', alpha=0.7)
ax3.axvline(dd_df['duration_hours_cal'].mean(), color='red', linestyle='--',
            linewidth=2, label=f'Mean: {dd_df["duration_hours_cal"].mean():.1f}hrs')
ax3.axvline(dd_df['duration_hours_cal'].median(), color='orange', linestyle='--',
            linewidth=2, label=f'Median: {dd_df["duration_hours_cal"].median():.1f}hrs')
ax3.set_xlabel('Duration (hours)')
ax3.set_ylabel('Frequency')
ax3.set_title('Drawdown Duration Distribution - Calendar Time')
ax3.legend()
ax3.grid(True, alpha=0.3, axis='y')

# 4. Histogram - MARKET duration in HOURS
ax4 = axes[3]
max_hours_mkt = dd_df['duration_hours_mkt'].max()
bins_mkt = np.linspace(0, max_hours_mkt, min(50, int(max_hours_mkt)+1))

ax4.hist(dd_df['duration_hours_mkt'], bins=bins_mkt, color='steelblue',
         edgecolor='black', alpha=0.7)
ax4.axvline(dd_df['duration_hours_mkt'].mean(), color='red', linestyle='--',
            linewidth=2, label=f'Mean: {dd_df["duration_hours_mkt"].mean():.1f}hrs')
ax4.axvline(dd_df['duration_hours_mkt'].median(), color='orange', linestyle='--',
            linewidth=2, label=f'Median: {dd_df["duration_hours_mkt"].median():.1f}hrs')
ax4.set_xlabel('Duration (hours) - Market Time Only')
ax4.set_ylabel('Frequency')
ax4.set_title('Drawdown Duration Distribution - Market Hours')
ax4.legend()
ax4.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()

print(f"\n🎯 INVESTOR PSYCHOLOGY IMPLICATIONS")
print("-" * 40)

max_duration_days = dd_df['duration_hours_cal'].max() / 24
avg_duration_days = dd_df['duration_hours_cal'].mean() / 24

if max_duration_days > 30:
    print(f"⚠ LONG RECOVERY: Longest drawdown = {max_duration_days:.1f} days")
    print("  - Investors may lose confidence during prolonged drawdowns")
    print("  - Consider: Monthly reporting cycles might miss recovery")
    print("  - Suggestion: Implement volatility scaling to reduce DD duration")
elif max_duration_days > 14:
    print(f"⚠ MODERATE DURATION: Longest drawdown = {max_duration_days:.1f} days")
    print("  - Recovery within typical investor tolerance")
    print("  - Risk: If coincides with month-end reporting, could trigger redemptions")
else:
    print(f"✓ FAST RECOVERY: Longest drawdown = {max_duration_days:.1f} days")
    print("  - Quick mean reversion supports investor confidence")
    print("  - Suitable for aggressive capital allocation")

# Recovery speed
avg_recovery_speed = abs(dd_df['depth_pct'].mean()) / avg_duration_days if avg_duration_days > 0 else 0
print(f"\n💪 RECOVERY SPEED")
print(f"  - Average: {avg_recovery_speed:.3f}% per day")
print(f"  - Interpretation: After a typical {abs(dd_df['depth_pct'].mean()):.2f}% drawdown,")
print(f"    expect ~{avg_duration_days:.1f} days (calendar) to recover to breakeven")
```

```
    # Market vs calendar comparison
    gap_ratio = dd_df['duration_hours_cal'].mean() / dd_df['duration_hours_mkt'].mean() if dd_df['duration_hours_mkt'].mean() >
    print(f"\n📊 CALENDAR/MARKET DURATION RATIO: {gap_ratio:.2f}x")
    if gap_ratio > 2:
        print("  ⚠ Drawdowns frequently span market closures (weekends/holidays)")
    else:
        print("  ✓ Most drawdowns occur within continuous market hours")

else:
    print("\nNo significant drawdown periods detected")

print("\n" + "="*60)
```

```
============================================================
DRAWDOWN DURATION & RECOVERY ANALYSIS
============================================================
Equity curve date range: 2020-02-12 17:20:00 to 2020-12-31 00:00:00
Equity curve length: 65820

📉 DRAWDOWN STATISTICS
----------------------------------------
Number of drawdown periods: 148
Average duration (Calendar): 19.8 hours (0.8 days)
Average duration (Market): 13.0 hours (0.5 days)
Median duration (Calendar): 0.2 hours
Median duration (Market): 0.2 hours
Longest drawdown (Calendar): 326.8 hours (13.6 days)
Longest drawdown (Market): 230.7 hours (9.6 days)
Average depth: -0.04%
Maximum depth: -0.18%

🔥 TOP 5 WORST DRAWDOWNS
----------------------------------------
27. Depth: -0.18% | Duration: 1hrs (cal) / 0hrs (mkt) | 2020-03-29 to 2020-03-29
51. Depth: -0.18% | Duration: 327hrs (cal) / 231hrs (mkt) | 2020-05-25 to 2020-06-07
21. Depth: -0.15% | Duration: 0hrs (cal) / 0hrs (mkt) | 2020-03-10 to 2020-03-10
20. Depth: -0.15% | Duration: 0hrs (cal) / 0hrs (mkt) | 2020-03-09 to 2020-03-09
23. Depth: -0.12% | Duration: 0hrs (cal) / 0hrs (mkt) | 2020-03-24 to 2020-03-24
```




```
🎯 INVESTOR PSYCHOLOGY IMPLICATIONS
----------------------------------------
✓ FAST RECOVERY: Longest drawdown = 13.6 days
  - Quick mean reversion supports investor confidence
  - Suitable for aggressive capital allocation

💪 RECOVERY SPEED
  - Average: 0.044% per day
  - Interpretation: After a typical 0.04% drawdown,
    expect ~0.8 days (calendar) to recover to breakeven

📊 CALENDAR/MARKET DURATION RATIO: 1.52x
  ✓ Most drawdowns occur within continuous market hours

============================================================
```

There's an odd trade that stays open (and in drawdown) for ~231 market hours. Further analysis should check if this was actually over a holiday that we are not accounting for. Also, in production this would likely be a great opportunity to accumulate positions over an extended period of sustained dislocation - subject to modelling inventory risk.

Summary

- ✓ Market Neutrality Confirmed: R^2 < 0.01% (0.00006)
- ✓ Controlled Tails: CVaR/VaR ~ 1.07
- ⚠️ Gap Risk: 9.3% of trades span weekends (moderate risk, but trivial to address)
- ⚠️ Drawdown Recovery: Average drawdown <1 day, longest 13.6 days

Conclusion & Future Work

Key Findings

The convergence strategy is **marginally profitable** but extremely sensitive to transaction costs:

- **Sharpe ~1.5** at 1.0 bps total friction (~0.33 bps per leg × 3)
- **Breakeven:** Naive filtering fails above 1.0 bps
- **Viability threshold:** Requires 99.9th percentile filtering (Z > 3σ) to survive costs

---

The Journey Through the Levels

- **Level 1:** Raw cointegration looks amazing (Sharpe > 11). Too good to be true.
- **Level 2:** Walk-forward optimization confirms stable hedge ratios (~1:1:1), but the 99% win rate obviously disappears when costs are accounted for.
- **Level 3:** Transaction costs kill the naive approach. We pivot to extreme selectivity; trading less, but only when dislocations are massive.
- **Level 4:** Operational reality check:
  - Average holding: 19 minutes (some trades span weekends)
  - Currency-neutral: R^2 < 0.01% (genuinely market-neutral)
  - Positive skew: Rare big wins offset frequent small losses
  - Fast recovery: Drawdowns resolve in <1 day on average

Key Takeaways

1. The structural cointegration relationship is real and stable (~1:1:1 hedge ratios)
2. Mean reversion happens fast (13-minute half-life) but requires patience to filter for quality
3. Walk-forward testing reveals the truth: Sharpe drops from 11 → 1.5 when you remove lookahead bias and add costs
4. Convergence trading ≠ HFT atomic arbitrage—we're playing a different game with different risks
5. Extreme selectivity is non-negotiable: only the 99.9th percentile dislocations survive transaction costs

---

Who This Works For

Viable:

- Institutional traders with <1.0 bps execution
- 100-150 trades/year at 99.9th percentile filtering
- Tolerance for multi-leg position management

Not viable:

- Retail (spreads too wide)
- High-frequency approaches (costs compound)
- Those seeking "true" HFT arbitrage (need <0.33 bps per leg)

Future Work

Execution:

- Tick-level analysis, order flow monitoring, and trades execution
- Pre-funded currency baskets to reduce lag (potentially adds inventory risk)
- Limit orders at threshold levels vs market orders
- Volume/Scalability/Reflexivity modeling - what is the capacity of these dislocations? How would our involvement reshape the distributions?

Strategy:

- Add more currencies (3 currencies = 1 triangle, but 4 currencies = 4 triangles!)
- Test exit strictness (currently only entry is filtered)
- Session-specific rules (focus on the profitable sessions/hours)

Robustness:

- Backtest 2015-2019 and 2021-2023 (was 2020 special?)
- Other triplets, such as EUR/CHF/USD, AUD/NZD/USD
- Weekend gap protection (e.g. forced Friday exits)

Final Verdict

Convergence trading on the triangular FX spread is **potentially viable** for institutional traders.

The strategy trades infrastructure requirements (HFT servers, microsecond latency) for holding risk (basis risk, gap risk, drawdowns). For those without HFT infrastructure, this represents a possible (if challenging)path to exploiting the EUR/USD/GBP/EUR triangular relationship.

Not Financial Advice

This is an educational project demonstrating quantitative finance concepts. Past performance does not indicate future results. Do not trade real money based on this code without extensive additional validation.