A Jump-Diffusion Momentum Strategy with Python and Backtrader

PyQuantLab Following 7 min read · Jun 18, 2025

Financial asset prices rarely move purely in a smooth, continuous “diffusion” process. Instead, they often exhibit jumps: sudden, large, and discontinuous movements that can significantly impact returns. These jumps, typically driven by unexpected news or events, often exhibit short-term momentum or mean-reversion characteristics depending on the market and the nature of the shock. This article explores a trading strategy that attempts to capitalize on these jump-diffusion dynamics, combining statistical detection with momentum and trend-

volatility filtering. So, the question is can sudden jumps indicate breakouts?

🔥 Unlock the Power of Profitable Trading with Code

🚀 The Ultimate Algorithmic Trading Bundle

> ✅ Over 80 ready-to-run strategies
> ✅ Across 5 powerful categories: momentum, trend-following, mean reversion, ML-based, and volatility
> ✅ Includes step-by-step PDF guides with code explanations, visuals & real-world insights
> ✅ Perfect for traders, quant enthusiasts, and developers

📦 Everything you need to go from zero to confident algo trader — in one complete package.

👉 Start building & backtesting proven strategies today:

🛒 Get the Ultimate Bundle

💡 Just Starting Out? Grab the Algo Trading Value Pack

> ⚡ 3 concise eBooks
> 📘 30+ Python-coded strategies

> 🧠 Simple, practical, and actionable — ready to use in Backtrader and real trading platforms

🎯 Ideal if you're looking to learn fast and see results in just a few days.

🛒 Start with the Value Pack

The Method & Theory: Deconstructing Price Movements

The strategy's foundation lies in distinguishing between two primary types of price movements:

1. Diffusion (Normal Fluctuations): Small, continuous changes driven by regular market activity.
2. Jumps (Abnormal Fluctuations): Large, discrete changes that exceed normal volatility, often caused by new information.

This distinction is crucial because the market's behavior *after* a jump can be systematically different from its behavior during normal diffusion. A significant price jump might attract follow-through momentum or trigger a short-term reversal.

Jump Detection:
The core of jump detection involves comparing a single period's

return to the asset's historical volatility. A common approach uses Z-scores:

$$Z_t = \frac{|R_t|}{\sigma_t}$$

where R is the current return and σ is the estimated volatility over a lookback period. If Z_t exceeds a predefined `jump_threshold` (e.g., 2.5 standard deviations), combined with a `min_jump_size` (e.g., 2.5% move), a jump is flagged.

Post-Jump Momentum/Diffusion Analysis:
Once a jump is detected, the strategy investigates the subsequent price action.

- Momentum: Does the price continue in the jump's direction? This is measured by the consistency of returns over a `momentum_period`.
- Diffusion Trend: Even without a clear jump, is there a sustained, albeit smaller, directional bias? This is assessed using a smoothed average of recent returns.

Trend & Volatility Filtering (ADX & ATR):
To enhance signal quality and avoid trading in unsuitable market conditions, the strategy incorporates filters based on:

- Average Directional Index (ADX): Measures the strength of a trend. A rising ADX indicates increasing trend strength, while an ADX below a `min_adx_level` suggests a non-trending market.
- Average True Range (ATR): Measures market volatility. A rising ATR can indicate increasing market participation and conviction, potentially supporting momentum.
  The strategy requires sustained rising conditions for both ADX and ATR over a `rising_lookback` period, ensuring that confirmed trends are developing alongside supportive volatility.

The Indicators: Building Blocks

The strategy employs three custom indicators to capture these dynamics:

1. `JumpDiffusionDetector`:
   This indicator calculates daily returns, estimates rolling volatility (standard deviation of returns), and then computes a Z-score to identify jumps. It also provides a smoothed `diffusion_trend` based on recent returns.

```
class JumpDiffusionDetector(bt.Indicator):
    lines = ('jump_signal', 'jump_magnitude', 'diffusion_trend')
    params = (('lookback', 20), ('jump_threshold', 2.5), ('min_jump
```

```
        def __init__(self):
            self.returns = bt.indicators.PctChange(self.data.close, per
            self.vol_estimator = bt.indicators.StandardDeviation(self.r
            self.change_buffer = deque(maxlen=self.params.lookback)

        def next(self):
            current_return = self.returns[0]
            current_vol = self.vol_estimator[0]
            # ... (Z-score calculation, jump detection, jump_signal and
            # Diffusion trend: smoothed recent returns
            self.change_buffer.append(current_return)
            if len(self.change_buffer) >= 5:
                recent_trend = np.mean(list(self.change_buffer)[-5:])
                self.lines.diffusion_trend[0] = np.tanh(recent_trend /
```

The `jump_signal` is for an upward jump, for a downward jump, and decays over time if no new jump occurs.

2. `MomentumAfterJump`:

This indicator quantifies the consistency of price movement (`momentum_strength`) and its direction (`momentum_direction`) over a `momentum_period`. It's designed to confirm whether the market is following through on a jump or a general directional bias.

```
class MomentumAfterJump(bt.Indicator):
    lines = ('momentum_strength', 'momentum_direction')
    params = (('momentum_period', 5), ('momentum_threshold', 0.6),)

    def __init__(self):
        self.returns = bt.indicators.PctChange(self.data.close, per
        self.momentum_buffer = deque(maxlen=self.params.momentum_pe
```

```
    def next(self):
        self.momentum_buffer.append(self.returns[0] if not np.isnan
        if len(self.momentum_buffer) >= self.params.momentum_period
            returns_array = np.array(list(self.momentum_buffer))
            positive_returns = np.sum(returns_array > 0)
            negative_returns = np.sum(returns_array < 0)
            total_returns = len(returns_array)

            if total_returns > 0:
                strength = max(positive_returns, negative_returns)
                direction = 1 if positive_returns > negative_return
                self.lines.momentum_strength[0] = strength
                self.lines.momentum_direction[0] = direction if str
```

3. `TrendVolatilityFilter`:

This indicator measures ADX and ATR, but crucially, it also counts how many periods they have been *rising* over a `rising_lookback` window. This provides a "sustained rising" condition, ensuring that the market is not just trending or volatile, but that these characteristics are actively strengthening.

```
class TrendVolatilityFilter(bt.Indicator):
    lines = ('adx_rising', 'atr_rising', 'adx_strength', 'atr_stren
    params = (('adx_period', 7), ('atr_period', 7), ('min_adx_level

    def __init__(self):
        self.adx = bt.indicators.DirectionalMovementIndex(period=se
        self.atr = bt.indicators.AverageTrueRange(period=self.param
        self.adx_buffer = deque(maxlen=self.params.rising_lookback
        self.atr_buffer = deque(maxlen=self.params.rising_lookback

    def next(self):
        current_adx = self.adx[0]
```

```
            current_atr = self.atr[0]
            self.adx_buffer.append(current_adx)
            self.atr_buffer.append(current_atr)

            # Count rising periods
            adx_rising_count = sum(1 for i in range(1, len(self.adx_buf
            atr_rising_count = sum(1 for i in range(1, len(self.atr_buf

            # Check for sustained rising and min ADX level
            adx_sustained_rising = (adx_rising_count >= self.params.min
            atr_sustained_rising = (atr_rising_count >= self.params.min

            self.lines.adx_rising[0] = 1 if adx_sustained_rising else 0
            self.lines.atr_rising[0] = 1 if atr_sustained_rising else 0
```

he Strategy: `JumpDiffusionMomentumStrategy`

This strategy combines the signals from the three indicators. It seeks to enter trades when a significant jump occurs, *followed by* or *confirmed by* positive momentum/diffusion, and only if the underlying market conditions (ADX and ATR) show a sustained increase in trend strength and volatility, respectively. This layered approach aims to filter out false signals and capitalize on strong, confirmed movements.

```
class JumpDiffusionMomentumStrategy(bt.Strategy):
    params = (
        ('jump_threshold', 2.0), ('min_jump_size', 0.05), # Jump de
        ('momentum_threshold', 0.7), # Strength required for moment
        ('diffusion_weight', 0.5),   # How much diffusion trend mat
        ('hold_periods', 7),         # Minimum time to hold positio
        ('min_adx_level', 20),       # Minimum ADX for trade
```

```
        ('require_adx_rising', True), ('require_atr_rising', True),
        ('trailing_stop_pct', 0.05), ('stop_loss_pct', 0.1), # Risk
        ('printlog', False),
    )

    def __init__(self):
        self.jump_detector = JumpDiffusionDetector(...) # Initializ
        self.momentum_detector = MomentumAfterJump(...) # Initializ
        self.trend_vol_filter = TrendVolatilityFilter(...) # Initia

    def next(self):
        # ... (Fetch indicator values and check for data validity)

        # Entry signals (if no position)
        if not self.position:
            # Filter condition: ADX & ATR must show sustained risin
            trend_vol_filters_passed = ( (not self.params.require_a
                                         ( (not self.params.require_a

            # Long entry logic: Strong positive jump + momentum/dif
            if (self.jump_detector.jump_signal[0] > 0.8 and # A cle
                (self.momentum_detector.momentum_direction[0] > 0 o
                trend_vol_filters_passed): # Filter conditions met
                self.order = self.buy()

            # Short entry logic: Strong negative jump + momentum/di
            elif (self.jump_detector.jump_signal[0] < -0.8 and # A
                  (self.momentum_detector.momentum_direction[0] < 0
                  trend_vol_filters_passed): # Filter conditions me
                self.order = self.sell()

        # ... (Position management, trailing stop, and fixed stop-l
```

The strategy is deliberately conservative, requiring multiple layers of confirmation. A “jump” alone isn’t enough; it must be followed by momentum (or strong diffusion) *and* occur within a

market environment that shows strengthening trend and volatility.

Backtesting and Analysis

The strategy is backtested on `ETH-USD` data over a 3-year period. The `run_jump_diffusion_strategy()` function sets up the Backtrader environment, including initial capital, commissions, and a suite of analyzers (Returns, Trades, Sharpe Ratio, Drawdown, VWR) to provide a comprehensive performance overview.

```
def run_jump_diffusion_strategy():
    print("Downloading data for ETH-USD...")
    data = yf.download('ETH-USD', period='3y', auto_adjust=False).d

    cerebro = bt.Cerebro()
    cerebro.addstrategy(JumpDiffusionMomentumStrategy, ...) # Add s
    cerebro.adddata(bt.feeds.PandasData(dataname=data))
    cerebro.broker.setcash(10000.0)
    cerebro.broker.setcommission(commission=0.001)
    cerebro.addsizer(bt.sizers.PercentSizer, percents=95)

    # Add various analyzers for comprehensive metrics
    cerebro.addanalyzer(bt.analyzers.Returns, _name='returns')
    cerebro.addanalyzer(bt.analyzers.TradeAnalyzer, _name='trades')
    cerebro.addanalyzer(bt.analyzers.SharpeRatio, _name='sharpe')
    cerebro.addanalyzer(bt.analyzers.DrawDown, _name='drawdown')
    cerebro.addanalyzer(bt.analyzers.VWR, _name='vwr')

    starting_value = cerebro.broker.getvalue()
    print(f'Starting Value: ${starting_value:,.2f}')
    results = cerebro.run()
    final_value = cerebro.broker.getvalue()
```

```
        # ... (Print detailed performance summary including total retur
        #      risk metrics, and Buy & Hold comparison.)

        # Plot results
        plt.rcParams['figure.figsize'] = [12, 8]
        cerebro.plot(style='line', iplot=False, figsize=(12, 6))[0][0]
        plt.tight_layout()
        plt.show()

if __name__ == '__main__':
    run_jump_diffusion_strategy()
```

The performance metrics provide a quantitative assessment, while the plot offers a visual depiction of the strategy's equity curve against the asset's price.

Conclusion: Navigating Market Discontinuities

The Jump-Diffusion Momentum Strategy offers an advanced approach to understanding and reacting to distinct price behaviors in financial markets. By systematically identifying significant price jumps and then confirming subsequent directional bias with momentum and market environment filters, it aims to capture potentially profitable opportunities that arise from these discontinuities. This approach moves beyond simple trend following or mean reversion, seeking to exploit specific market dynamics. As with all complex quantitative strategies, careful parameter calibration, robust out-of-sample testing, and continuous monitoring are essential to ensure its effectiveness in live trading environments.

Python | Algorithmic Trading | Crypto Trading | Trading Strategy | Bitcoin

Written by PyQuantLab

655 followers · 6 following

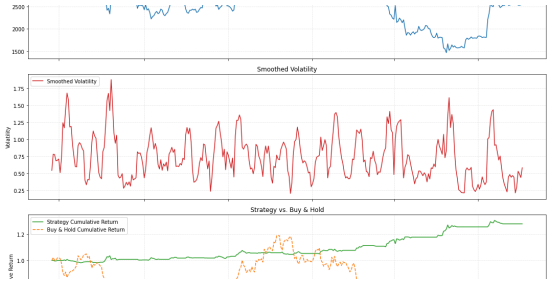
Your go-to place for Python-based quant tutorials, strategy deep-dives, and reproducible code. For more visit our website: www.pyquantlab.com

No responses yet

Steven Feng CAI

What are your thoughts?

More from PyQuantLab



PyQuantLab

Volatility Clustering Trading Strategy with Python

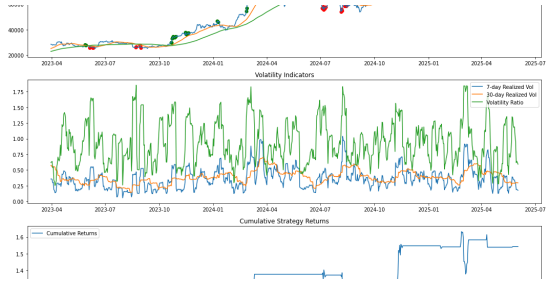
Ultimate Algorithmic Strategy Bundle has you covered with over 80 Python...

PyQuantLab

An Algorithmic Exploration of Volume Spread Analysis...

Note: You can read all articles for free on our website: pyquantlab.com

Jun 3 32 3



PyQuantLab

Trend-Volatility Confluence Trading Strategy

The Ultimate Algorithmic Strategy Bundle has you covered with over 80...

Jun 3 60

Jun 9 54 1

PyQuantLab

Building an Adaptive Trading Strategy with Backtrader: A...

Note: You can read all articles for free on our website: pyquantlab.com

Jun 4 64

See all from PyQuantLab

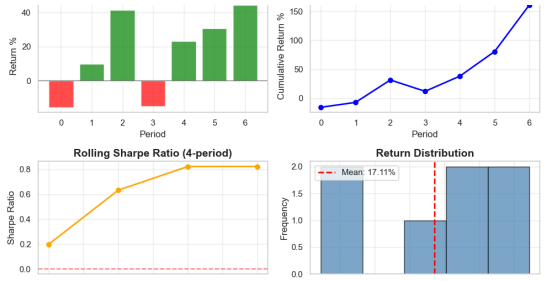
Recommended from Medium

MarketMuse

“I Let an AI Bot Trade for Me for 7 Days—It Made $8,000...

Subtitle: While you’re analyzing candlestick patterns, AI bots are fron...

Jun 3 75 3



PyQuantLab

Precision Trading with Volume Profile: An Enhance...

In the intricate landscape of financial markets, understanding where...

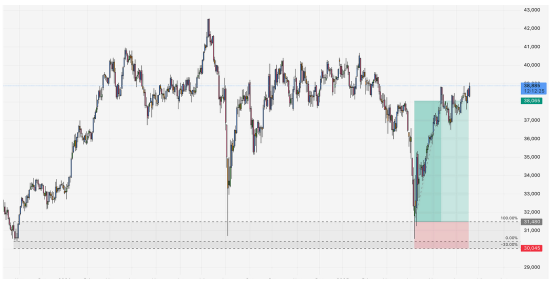
Jun 20 11

In DataDrivenInvestor by Mr. Q

Why You Should Ignore Most Backtested Trading...

To be more precise, while the title is limited in length, what I truly mean ar...

Jun 18 133 2

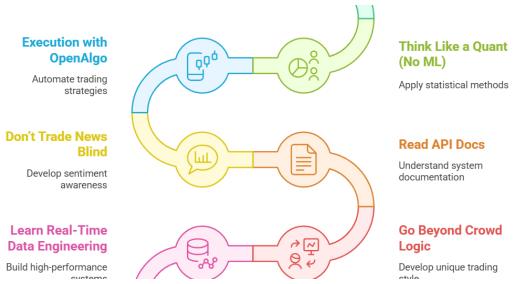


Candence

Exposing Bernd Skorupinski Strategy: How I Profited ove...

I executed a single Nikkei Futures trade that banked $16,400 with a...

Jun 19 2



Rajandran R (Creator - OpenAlgo)

Algorithmic Trading Roadmap 2025: From Curio...

The dream of algorithmic trading is simple: let code take over the...

Jun 22 23 3

In Python in Plain Engli... by Nayab Bhut...

This Python Stop-Loss Strategy Multiplied My Gain...

Every trader talks about risk management, but few actually build i...

Jun 18 51 4

See more recommendations