Member-only story

A Chaos Theory-Based Trading Strategy in Backtrader

PyQuantLab Following 17 min read · Jun 8, 2025

Financial markets are often described as complex, non-linear systems. While traditional technical analysis provides valuable insights, some researchers delve into Chaos Theory to find patterns, or rather, the boundaries of predictability within seemingly random price movements. Chaos theory suggests that even complex, deterministic systems can exhibit seemingly random behavior due to extreme sensitivity to initial conditions (the “butterfly effect”). Key concepts like Lyapunov Exponents, Correlation Dimension, and the Hurst Exponent can provide insights into the underlying dynamics of a time series.

This tutorial will guide you through building a `backtrader` strategy that attempts to gauge the "chaotic" nature of market returns. The strategy will calculate these advanced metrics on a rolling basis and use them to identify potential regime changes – shifts from chaotic to more ordered behavior, or vice versa – as trading signals. We will combine this with a simple trend filter and essential risk management via a stop-loss.

Understanding Chaos Theory in Trading

Key Concepts

1. Phase Space Reconstruction (Time Delay Embedding):

- Financial data is univariate (e.g., a single price series). To analyze its chaotic properties, we need to reconstruct its

"phase space," which is a multi-dimensional representation of its state.

- Takens' Theorem states that we can reconstruct an attractor topologically equivalent to the original system's attractor using a single time series by forming vectors with time-delayed copies of the series, where is the embedding dimension and is the time delay.

2. Lyapunov Exponent (λ):

- Measures the rate at which nearby trajectories in phase space diverge or converge.
- A positive Lyapunov exponent is the hallmark of chaotic systems, indicating exponential divergence and thus unpredictability over longer time horizons.
- A negative exponent suggests convergence to a stable point (predictable).
- A zero exponent indicates periodic or quasi-periodic behavior.
- In trading, a high positive Lyapunov exponent might suggest extreme unpredictability, where trend-following or mean-reversion strategies struggle. A low or negative exponent could indicate periods of higher predictability.

3. Correlation Dimension (D_2):

- A measure of the fractal dimension of the attractor in phase space. It quantifies the “complexity” or “roughness” of the system.
- For a truly random (stochastic) series, D_2 tends towards infinity (or the embedding dimension).
- For a deterministic chaotic system, D_2 will be a fractional value.
- In trading, a lower, fractional D_2 might suggest a more deterministic, albeit chaotic, process, while a higher approaching the embedding dimension might point to randomness.

4. Hurst Exponent (H):

- Measures the long-term memory of a time series, quantifying its tendency to trend or mean-revert.
- H = 0.5 : Random walk (Brownian motion), no long-term memory. Future movements are independent of past ones.
- 0.5 < H < 1: Persistent or trending behavior. If the price moved up in the past, it’s more likely to move up in the future. Stronger persistence as H approaches 1.

- $0 < H < 0.5$: Anti-persistent or mean-reverting behavior. If the price moved up in the past, it's more likely to move down in the future. Stronger mean-reversion as H approaches 0.
- In trading, Hurst can help identify if a market is in a trending or mean-reverting regime.

Strategy Concept

Our Chaos Theory strategy will use these metrics to identify regime changes in market returns (since chaos is often more apparent in returns than in prices themselves).

1. Returns History: Collect a rolling window of daily percentage returns.
2. Phase Space Reconstruction: Create embedded vectors from the returns history using specified `embedding_dim` and `delay`.
3. Chaos Metric Calculation:

- Estimate the largest Lyapunov Exponent (using a simplified Wolf's algorithm).
- Estimate the Correlation Dimension (using a simplified Grassberger-Procaccia algorithm).
- Estimate the Hurst Exponent (using R/S analysis).

4. Trading Signals (Focus on Regime Change):

- “Order” Signal (Enter Trades): When the system transitions from a highly chaotic state (high Lyapunov, high Correlation Dimension) to a more ordered or predictable state (low Lyapunov, low Correlation Dimension). This suggests a shift to a potentially more trendable or mean-reverting regime. We will combine this with a simple trend filter.
- Buy: If chaos decreases (Lyapunov drops below threshold, Correlation Dimension is low) AND the market is in an uptrend.
- Sell: If chaos decreases (Lyapunov drops below threshold, Correlation Dimension is low) AND the market is in a downtrend.
- “Chaos Resumes” Signal (Exit Trades): When the system reverts to a highly chaotic state (Lyapunov exponent significantly increases), suggesting unpredictability and potential for whipsaws.
- Hurst-based Signals (Alternative/Complementary): Use the Hurst exponent to identify persistent (trending) or anti-persistent (mean-reverting) regimes.
- Buy: If Hurst suggests persistence in an uptrend.
- Sell: If Hurst suggests anti-persistence in a downtrend.

5. Risk Management: A fixed percentage stop-loss will be applied to all positions.

Prerequisites

Ensure you have the necessary Python libraries installed. Note that `scipy.spatial.distance` and `scipy.stats.linregress` are key for the chaos metrics.

```
pip install backtrader yfinance pandas numpy matplotlib scipy
```

Step-by-Step Implementation

We'll structure the code into components, with the most complex being the chaos metric calculations.

1. Initial Setup and Data Acquisition

First, we set up our environment and download Ethereum (ETH-USD) historical data.

```
import backtrader as bt
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist, squareform # For correlat
```

```
from scipy.stats import linregress # For correlation dimension and

# Set matplotlib style for better visualization
%matplotlib inline
plt.rcParams['figure.figsize'] = (10, 6)

# Download historical data for Ethereum (ETH-USD)
# Remember the instruction: yfinance download with auto_adjust=Fals
print("Downloading ETH-USD data from 2021-01-01 to 2024-01-01...")
data = yf.download('ETH-USD', '2021-01-01', '2024-01-01', auto_adju
data.columns = data.columns.droplevel(1) # Drop the second level of
print("Data downloaded successfully.")
print(data.head()) # Display first few rows of the data

# Create a Backtrader data feed from the pandas DataFrame
data_feed = bt.feeds.PandasData(dataname=data)
```

Explanation:

- `yfinance.download` : Fetches cryptocurrency data.
- `data.columns = data.columns.droplevel(1)` : Flattens the column index.
- `bt.feeds.PandasData` : Converts data to `backtrader` format.
- `scipy.spatial.distance` and `scipy.stats.linregress` : Imported for the specialized chaos metric calculations.

2. The Chaos Theory Strategy (`ChaosStrategy`)

This class contains the logic for calculating chaos metrics and generating trading signals.

```
class ChaosStrategy(bt.Strategy):
    params = (
        ('lookback', 100),            # Window for chaos analysis (
        ('embedding_dim', 5),         # Embedding dimension for pha
        ('delay', 3),                 # Time delay for embedding
        ('lyap_threshold', 0.05),     # Lyapunov exponent threshold
        ('corr_dim_threshold', 2.0),  # Correlation dimension thres
        ('trend_period', 30),         # Simple Moving Average perio
        ('stop_loss_pct', 0.02),      # 2% stop loss
    )

    def __init__(self):
        # Store returns history for chaos analysis
        self.returns_history = [] # Use returns, as chaotic dynamic

        # Calculate daily percentage returns
        self.returns = bt.indicators.PctChange(self.data.close, per

        # Trend filter (Simple Moving Average on close price)
        self.trend_ma = bt.indicators.SMA(self.data.close, period=s

        # Variables to store chaos theory measures for the current
        self.lyapunov_exponent = 0.0
        self.correlation_dimension = 0.0
        self.hurst_exponent = 0.5 # Default for random walk

        # Track orders
        self.order = None
        self.stop_order = None

    def time_delay_embedding(self, series, m, tau):
        """
        Create time-delay embedding vectors for phase space reconst
        Args:
            series (np.array): The time series data (e.g., returns)
            m (int): Embedding dimension.
            tau (int): Time delay.
        Returns:
            np.array: Embedded vectors.
        """
        N = len(series)
        # Ensure enough data points to form at least one embedded v
```

```
        if N < m * tau:
            return np.array([])

        # Create embedded vectors
        embedded = np.zeros((N - (m-1)*tau, m))
        for i in range(m):
            embedded[:, i] = series[i*tau : N-(m-1-i)*tau]
        return embedded

    def calculate_lyapunov_exponent(self, embedded_vectors):
        """
        Calculates a simplified largest Lyapunov exponent using Wol
        This is a computationally intensive and approximate impleme
        """
        if len(embedded_vectors) < 20: # Need sufficient points
            return 0.0

        try:
            N = len(embedded_vectors)
            lyap_sum = 0
            count = 0

            # Iterate through trajectory points to find nearest nei
            for i in range(N - 5): # Avoid index out of bounds for
                # Find the nearest neighbor to embedded_vectors[i]
                distances = np.linalg.norm(embedded_vectors[i] - em
                distances[i] = np.inf  # Exclude self from nearest
                nearest_idx = np.argmin(distances)

                # Ensure nearest neighbor is also within bounds for
                if nearest_idx < N - 5:
                    initial_dist = distances[nearest_idx]
                    if initial_dist > 0: # Avoid division by zero
                        # Track divergence over 'delay' steps (here
                        future_dist = np.linalg.norm(
                            embedded_vectors[i + 3] - embedded_vect
                        )

                        if future_dist > 0: # Avoid log(0)
                            lyap_sum += np.log(future_dist / initia
                            count += 1

            # Average divergence rate over all considered pairs
            # Divided by the number of steps over which divergence
```

```
            return lyap_sum / (3 * count) if count > 0 else 0.0

        except Exception as e:
            # print(f"Error calculating Lyapunov: {e}")
            return 0.0

    def calculate_correlation_dimension(self, embedded_vectors):
        """
        Estimates the correlation dimension using a simplified Gras
        This is a computationally intensive and approximate impleme
        """
        if len(embedded_vectors) < 20: # Needs sufficient points
            return 0.0

        try:
            # Calculate all pairwise Euclidean distances between em
            distances = pdist(embedded_vectors)

            if len(distances) == 0 or np.all(distances == 0):
                return 0.0

            # Define a range of radii (epsilon) values on a logarit
            # Ensure min_dist is greater than zero
            min_dist = np.min(distances[distances > 0])
            max_dist = np.max(distances)

            if min_dist >= max_dist: # Handle cases where all non-z
                return 0.0

            radii = np.logspace(np.log10(min_dist), np.log10(max_di
            correlation_integrals = []

            # Calculate the correlation integral C(r) for each radi
            # C(r) is the proportion of pairs of points whose dista
            for r in radii:
                count = np.sum(distances < r)
                correlation_integrals.append(count / len(distances)

            # Estimate dimension from the slope of log(C(r)) vs log
            # Take log of radii and integrals, add a small epsilon
            log_radii = np.log(radii[radii > 0]) # Ensure radii are
            log_integrals = np.log(np.array(correlation_integrals)[

            # Perform linear regression on the linear region of the
```

```
            if len(log_radii) > 3: # Need enough points for regress
                slope, _, _, _, _ = linregress(log_radii, log_integ
                # The slope is an estimate of the correlation dimen
                return max(0.0, min(self.params.embedding_dim, slop

            return 0.0

        except Exception as e:
            # print(f"Error calculating Correlation Dimension: {e}"
            return 0.0

    def calculate_hurst_exponent(self, series):
        """
        Calculate Hurst exponent using the rescaled range (R/S) ana
        This is a simplified, single-window R/S calculation.
        """
        if len(series) < 20: # Needs at least 20 points for a reaso
            return 0.5 # Default to random walk if not enough data

        try:
            # 1. Calculate the mean-adjusted series
            mean_series = np.mean(series)
            centered_series = series - mean_series

            # 2. Calculate the cumulative sum (deviations from mean
            cumsum = np.cumsum(centered_series)

            # 3. Calculate the range (R)
            R = np.max(cumsum) - np.min(cumsum)

            # 4. Calculate the standard deviation (S)
            S = np.std(series)

            if S == 0: # Avoid division by zero
                return 0.5

            # 5. Calculate the rescaled range (R/S)
            rs_ratio = R / S

            if rs_ratio <= 0: # Handle cases where R/S is non-posit
                return 0.5

            # 6. Estimate Hurst exponent: H = log(R/S) / log(N)
            # This is a simplified direct calculation. More robust
```

```
            # for multiple N and taking the slope.
            hurst = np.log(rs_ratio) / np.log(len(series))

            return max(0.0, min(1.0, hurst)) # Bound H between 0 an

        except Exception as e:
            # print(f"Error calculating Hurst: {e}")
            return 0.5

    def analyze_strange_attractor(self, series):
        """
        Combines all chaos metric calculations.
        """
        # Ensure series has enough data for embedding and subsequen
        if len(series) < self.params.lookback:
            return 0.0, 0.0, 0.5 # Return defaults if not enough da

        # 1. Phase space reconstruction
        embedded = self.time_delay_embedding(
            series, self.params.embedding_dim, self.params.delay
        )

        if len(embedded) < 20: # Need enough embedded points for ro
            return 0.0, 0.0, 0.5

        # 2. Calculate chaos measures
        lyap = self.calculate_lyapunov_exponent(embedded)
        corr_dim = self.calculate_correlation_dimension(embedded)
        hurst = self.calculate_hurst_exponent(series) # Hurst is us

        return lyap, corr_dim, hurst

    def notify_order(self, order):
        # Standard backtrader notify_order for managing stop-loss
        if order.status in [order.Completed]:
            if order.isbuy() and self.position.size > 0:
                stop_price = order.executed.price * (1 - self.param
                self.stop_order = self.sell(exectype=bt.Order.Stop,
                # self.log(f'BUY EXECUTED, Price: {order.executed.p
            elif order.issell() and self.position.size < 0:
                stop_price = order.executed.price * (1 + self.param
                self.stop_order = self.buy(exectype=bt.Order.Stop,
                # self.log(f'SELL EXECUTED (Short), Price: {order.e
```

```
        if order.status in [order.Completed, order.Canceled, order.
            self.order = None
            if order == self.stop_order:
                self.stop_order = None

    def log(self, txt, dt=None):
        ''' Logging function for the strategy '''
        dt = dt or self.datas[0].datetime.date(0)
        # print(f'{dt.isoformat()}, {txt}') # Commented out for cle

    def next(self):
        # Prevent new orders if one is already pending
        if self.order is not None:
            return

        # Store current returns in history list
        if not np.isnan(self.returns[0]):
            self.returns_history.append(self.returns[0])

        # Keep only the most recent 'lookback' * 2 history (arbitra
        # The actual `lookback` from params is used when slicing `r
        if len(self.returns_history) > self.params.lookback * 2:
            self.returns_history = self.returns_history[-self.param

        # Skip if not enough data for the lookback period
        if len(self.returns_history) < self.params.lookback:
            return

        # Perform chaos analysis on the recent returns history
        recent_returns = np.array(self.returns_history[-self.params
        lyap, corr_dim, hurst = self.analyze_strange_attractor(rece

        # Store previous values and update current values for compa
        prev_lyap = self.lyapunov_exponent
        prev_corr_dim = self.correlation_dimension

        self.lyapunov_exponent = lyap
        self.correlation_dimension = corr_dim
        self.hurst_exponent = hurst

        # --- Trading logic based on Chaos Theory metrics ---
        # Strategy: Attempt to trade when the system transitions fr
        # This implies a shift to a regime where patterns might be
```

```
        # Signal 1: Transition from Chaos to Order (Lyapunov drops
        # And align with the market trend using SMA.

        # Condition for potential order/predictability
        is_becoming_ordered = (lyap < self.params.lyap_threshold an
        is_simple_structure = (corr_dim < self.params.corr_dim_thre

        # Determine current market trend
        is_uptrend = self.data.close[0] > self.trend_ma[0]
        is_downtrend = self.data.close[0] < self.trend_ma[0]

        if is_becoming_ordered and is_simple_structure:
            if is_uptrend: # Enter long in an uptrend when predicta
                if self.position.size < 0:  # Close short first if
                    self.order = self.close()
                    if self.stop_order is not None: self.cancel(sel
                elif not self.position:  # Go long if not in positi
                    self.order = self.buy()
                    self.log(f'BUY SIGNAL (Chaos -> Order, Uptrend)
            elif is_downtrend: # Enter short in a downtrend when pr
                if self.position.size > 0:  # Close long first if e
                    self.order = self.close()
                    if self.stop_order is not None: self.cancel(sel
                elif not self.position:  # Go short if not in posit
                    self.order = self.sell()
                    self.log(f'SELL SIGNAL (Chaos -> Order, Downtre

        # Signal 2: Exit when system becomes chaotic again (predict
        # This acts as a protective mechanism, exiting when the mar
        elif lyap > self.params.lyap_threshold * 2 and self.positio
            self.log(f'CLOSING POSITION (Chaos Resuming), Lyap: {ly
            if self.stop_order is not None:
                self.cancel(self.stop_order)
            self.order = self.close()

        # Signal 3 (Alternative/Complementary): Hurst Exponent-base
        # Trade when Hurst suggests clear trending (H > 0.6) or mea
        # This part could be used in conjunction with or as an alte
        # For this tutorial, we will make it distinct to demonstrat
        # Note: If these signals are activated, they might override
        # You would typically choose one main set of entry criteria

        # Uncomment and adjust if you want to also trade based on H
        # elif self.position.size == 0: # Only enter new positions
```

```
#     if (hurst > 0.6 and is_uptrend): # Strong persistence
#         self.order = self.buy()
#         self.log(f'BUY SIGNAL (Hurst Trending), H: {hurst
#     elif (hurst < 0.4 and is_downtrend): # Strong anti-pe
#         # This interpretation can be tricky; sometimes an
#         # but here, in downtrend, could be interpreted as
#         # A better interpretation for Hurst < 0.5 + downt
#         # temporarily bounces up, or a strong trending si
#         # For simplicity, let's just use > 0.6 for long i
#         # If Hurst < 0.4 usually suggests mean-reversion,
#         pass # Currently no explicit Hurst short signal
```

Important Notes on Chaos Metric Implementations:

- Approximation: The implementations of Lyapunov Exponent (Wolf's algorithm) and Correlation Dimension (Grassberger-Procaccia) provided here are simplified and approximate for conceptual demonstration within `backtrader`. Real-world, robust implementations often involve more sophisticated techniques, parameter selection (e.g., optimal `delay` and `embedding_dim` often found via mutual information and false nearest neighbors methods), and averaging over multiple scales/neighbors. They are computationally intensive.
- Computational Cost: Calculating these metrics for every `next` bar, especially with large `lookback` periods, can be very slow. For practical applications, these might be calculated less frequently or optimized with C/Fortran extensions.

- Interpretation: The thresholds (`lyap_threshold` , `corr_dim_threshold`) are highly empirical and asset-specific. Their meaning can be subjective and vary greatly.

Explanation of `ChaosStrategy` :

- `params` : Defines parameters for `lookback` , `embedding_dim` , `delay` , `lyap_threshold` , `corr_dim_threshold` , `trend_period` , and `stop_loss_pct` .
- `__init__(self)` : Initializes history lists for returns, `PctChange` indicator for returns, an SMA for trend filtering, and variables to store chaos metrics.
- `time_delay_embedding(self, series, m, tau)` : Reconstructs the phase space. It takes a time series `series` , embedding dimension `m` , and time delay `tau` , returning a set of embedded vectors.
- `calculate_lyapunov_exponent(self, embedded_vectors)` : Implements a simplified version of Wolf's algorithm. It looks for nearest neighbors in phase space and tracks their exponential divergence over a few steps. A positive result suggests chaos.
- `calculate_correlation_dimension(self, embedded_vectors)` : Implements a simplified Grassberger-Procaccia algorithm. It calculates pairwise distances in phase space and counts how

many pairs are within a given radius (`r`). The slope of `log(C(r))` vs `log(r)` estimates the dimension.

- `calculate_hurst_exponent(self, series)` : Calculates the Hurst exponent using R/S (Rescaled Range) analysis. A value > 0.5 indicates persistence, < 0.5 indicates anti-persistence, and = 0.5 is random.
- `analyze_strange_attractor(self, series)` : A wrapper function that calls the three chaos metric calculations.
- `notify_order(self, order)` : Standard `backtrader` method for managing order status and placing stop-loss orders.
- `log(self, txt, dt=None)` : Simple logging utility.
- `next(self)` : The main trading logic:
- Appends current returns to `returns_history` and maintains its length.
- Calls `analyze_strange_attractor` on the recent returns.
- Updates `self.lyapunov_exponent`, `self.correlation_dimension`, `self.hurst_exponent` for the current bar, storing previous values for comparison.
- Trading Logic (Regime Change Detection):
- Entry Signal: The strategy looks for a transition from a more chaotic state (`prev_lyap >= self.params.lyap_threshold`) to a more ordered one (`lyap < self.params.lyap_threshold`) AND a

relatively simple underlying structure (`corr_dim < self.params.corr_dim_threshold`). If these conditions are met, and the market is in an uptrend (`data.close[0] > trend_ma[0]`), it buys. If in a downtrend, it sells.

- Exit Signal: If the Lyapunov exponent significantly increases (`lyap > self.params.lyap_threshold * 2`), indicating the system is becoming highly unpredictable again, the strategy exits any open position. This is a "safety" exit.
- Hurst-based Signals (Commented out): An alternative or complementary set of signals based on the Hurst exponent, where strong persistence (`H > 0.6`) might signal long trades in an uptrend, and anti-persistence (`H < 0.4`) might signal mean-reversion trades. This section is commented out to keep the primary logic focused on Lyap/CorrDim for this tutorial, but can be reactivated for experimentation.

3. Backtesting Setup and Execution

Finally, we configure the `backtrader` Cerebro engine, add our strategy, data, broker settings, and comprehensive performance analyzers.

```
# Create a Cerebro entity
cerebro = bt.Cerebro()

# Add the strategy
```

```
cerebro.addstrategy(ChaosStrategy)

# Add the data feed
cerebro.adddata(data_feed)

# Set the sizer: invest 95% of available cash on each trade
cerebro.addsizer(bt.sizers.PercentSizer, percents=95)

# Set starting cash
cerebro.broker.setcash(100000.0) # Start with $100,000

# Set commission (e.g., 0.1% per transaction)
cerebro.broker.setcommission(commission=0.001)

# --- Add Analyzers for comprehensive performance evaluation ---
cerebro.addanalyzer(bt.analyzers.SharpeRatio, _name='sharpe')
cerebro.addanalyzer(bt.analyzers.DrawDown, _name='drawdown')
cerebro.addanalyzer(bt.analyzers.Returns, _name='returns')
cerebro.addanalyzer(bt.analyzers.TradeAnalyzer, _name='tradeanalyze
cerebro.addanalyzer(bt.analyzers.SQN, _name='sqn') # System Quality

# Print starting portfolio value
print(f'Starting Portfolio Value: ${cerebro.broker.getvalue():,.2f}

# Run the backtest
print("Running backtest...")
results = cerebro.run()
print("Backtest finished.")

# Print final portfolio value
final_value = cerebro.broker.getvalue()
print(f'Final Portfolio Value: ${final_value:,.2f}')
print(f'Return: {((final_value / 100000) - 1) * 100:.2f}%') # Calcu

# --- Get and print analysis results ---
strat = results[0] # Access the strategy instance from the results

print("\n--- Strategy Performance Metrics ---")

# 1. Returns Analysis
returns_analysis = strat.analyzers.returns.get_analysis()
total_return = returns_analysis.get('rtot', 'N/A') * 100
annual_return = returns_analysis.get('rnorm100', 'N/A')
print(f"Total Return: {total_return:.2f}%")
```

```
print(f"Annualized Return: {annual_return:.2f}%")

# 2. Sharpe Ratio (Risk-adjusted return)
sharpe_ratio = strat.analyzers.sharpe.get_analysis()
print(f"Sharpe Ratio: {sharpe_ratio.get('sharperatio', 'N/A'):.2f}"

# 3. Drawdown Analysis (Measure of risk)
drawdown_analysis = strat.analyzers.drawdown.get_analysis()
max_drawdown = drawdown_analysis.get('maxdrawdown', 'N/A')
print(f"Max Drawdown: {max_drawdown:.2f}%")
print(f"Longest Drawdown Duration: {drawdown_analysis.get('maxdrawd

# 4. Trade Analysis (Details about trades)
trade_analysis = strat.analyzers.tradeanalyzer.get_analysis()
total_trades = trade_analysis.get('total', {}).get('total', 0)
won_trades = trade_analysis.get('won', {}).get('total', 0)
lost_trades = trade_analysis.get('lost', {}).get('total', 0)
win_rate = (won_trades / total_trades) * 100 if total_trades > 0 el
print(f"Total Trades: {total_trades}")
print(f"Winning Trades: {won_trades} ({win_rate:.2f}%)")
print(f"Losing Trades: {lost_trades} ({100-win_rate:.2f}%)")
print(f"Average Win (PnL): {trade_analysis.get('won',{}).get('pnl',
print(f"Average Loss (PnL): {trade_analysis.get('lost',{}).get('pnl
print(f"Ratio Avg Win/Avg Loss: {abs(trade_analysis.get('won',{}).g

# 5. System Quality Number (SQN) - Dr. Van Tharp's measure of syste
sqn_analysis = strat.analyzers.sqn.get_analysis()
print(f"System Quality Number (SQN): {sqn_analysis.get('sqn', 'N/A'


# --- Plot the results ---
print("\nPlotting results...")
# Adjust matplotlib plotting parameters to prevent warnings with la
plt.rcParams['figure.max_open_warning'] = 0
plt.rcParams['agg.path.chunksize'] = 10000 # Helps with performance

try:
    # iplot=False for static plot, style='candlestick' for candlest
    # plotreturn=True to show the equity curve in a separate subplo
    # Volume=False to remove the volume subplot as it might not be
    fig = cerebro.plot(iplot=False, style='candlestick',
                       barup=dict(fill=False, lw=1.0, ls='-', color='gree
                       bardown=dict(fill=False, lw=1.0, ls='-', color='re
                       plotreturn=True, # Show equity curve
```

```
                    numfigs=1, # Ensure only one figure is generated
                    volume=False # Exclude volume plot, as it can clut
                )[0][0] # Access the figure object to save/show

        plt.show() # Display the plot
    except Exception as e:
        print(f"Plotting error: {e}")
        print("Strategy completed successfully, but plotting was skippe
```

Advantages and Challenges of Chaos Theory-Based Strategies

Advantages:

- Novelty: Offers a unique, non-linear approach to market analysis, distinct from traditional indicators.

- Regime Detection: Potentially capable of identifying shifts in market behavior (e.g., from unpredictable to more structured, or trending to mean-reverting).
- Deep Understanding: Encourages a deeper, more scientific inquiry into market dynamics.

Challenges and Considerations:

1. Computational Complexity: Calculating Lyapunov Exponents and Correlation Dimension accurately is *extremely* computationally intensive and sensitive to parameters (`embedding_dim`, `delay`, `lookback`). The implementations provided here are simplified approximations for demonstration. Robust research-grade implementations are far more complex.
2. Parameter Sensitivity: The choice of `lookback`, `embedding_dim`, and `delay` is crucial and non-trivial. Incorrect parameter selection can lead to meaningless results. Optimal values often require specialized methods (e.g., mutual information for `delay`, false nearest neighbors for `embedding_dim`).
3. Data Requirements: These methods typically require a large amount of clean, high-quality data to produce reliable results.
4. Interpretation: The meaning of the numerical values for Lyapunov exponents and correlation dimension in a financial context is often debated and can be ambiguous. What

constitutes “chaotic enough” or “ordered enough” is subjective and prone to overfitting.

5. Non-Stationarity: Financial time series are inherently non-stationary, which violates the assumptions of many chaos theory tools. While using returns (which are more stationary than prices) helps, it doesn’t fully solve the problem.
6. Predictability vs. Randomness: Even if a system is deterministic and chaotic, its high sensitivity to initial conditions means it remains unpredictable beyond a very short horizon. The goal is often to identify *when* it’s less chaotic or to understand its underlying structure, rather than direct prediction.
7. Overfitting: With many parameters and the complex nature of the calculations, there’s a very high risk of overfitting this strategy to historical data.

Further Enhancements

1. Optimal Parameter Selection: Implement methods like average mutual information (for `delay`) and false nearest neighbors (for `embedding_dim`) to determine more appropriate parameters for phase space reconstruction.
2. More Robust Algorithms: For serious research, use dedicated libraries or more sophisticated implementations of chaos metrics.

3. Adaptive Thresholds: Make `lyap_threshold` and `corr_dim_threshold` dynamic based on the historical distribution of these metrics.
4. Multi-Scale Analysis: Apply chaos analysis at different timeframes or with different `lookback` periods to identify nested dynamics.
5. Combine Signals: Integrate these chaos metrics more deeply with other indicators (e.g., using a state-machine or machine learning model that takes chaos metrics as features).
6. Visualization of Chaos Metrics: Create custom `backtrader.indicators` to plot the `lyapunov_exponent`, `correlation_dimension`, and `hurst_exponent` in subplots to visually correlate them with price movements.
7. Performance Optimization: For real-time or higher-frequency trading, these calculations would need significant optimization (e.g., using Numba, Cython, or pre-computation).

Conclusion

This tutorial has ventured into the intriguing domain of Chaos Theory in quantitative finance, demonstrating how to implement a strategy based on Lyapunov Exponents, Correlation Dimension, and the Hurst Exponent in `backtrader`. While these advanced tools offer a unique perspective on market dynamics

and regime changes, their practical application in live trading is highly challenging due to computational complexity, parameter sensitivity, and the inherent unpredictability of financial markets. This strategy serves as an excellent starting point for academic exploration and understanding, highlighting the depth of quantitative analysis beyond conventional indicators. Remember, rigorous testing, caution, and a deep understanding of the underlying mathematics are essential when dabbling in such complex approaches.

Python | Algorithmic Trading | Quantitative Finance | Bitcoin | Backtesting

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

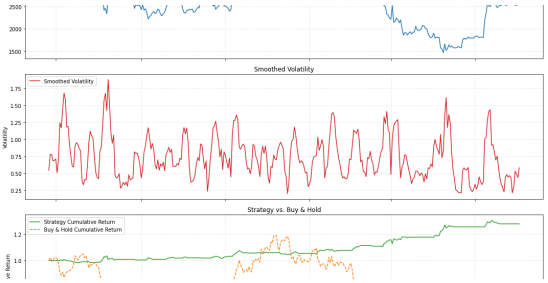
Following

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...

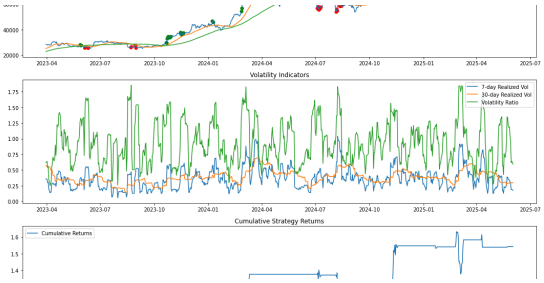
Jun 3 32 3

PyQuantLab

An Algorithmic Exploration of Volume Spread Analysis...

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

Jun 9 54 1



PyQuantLab

Trend-Volatility Confluence Trading Strategy

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

Jun 3 60

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

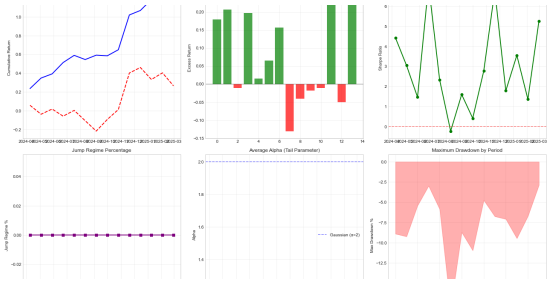
Recommended from Medium

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

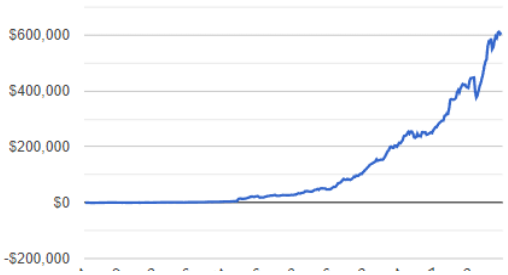


PyQuantLab

Capturing Market Momentum with Levy Flights: A Python...

Financial markets are complex systems, often exhibiting behaviors...

Jun 5 102



In DataDrivenInvestor by Ayrat Murtazin

$400k+ Profit, 20,000% Account Growth In 1.5 Years...

If I had read this when I first started off I would probably not believe it myself...

Jun 7 128 10

In InsiderFinance Wire by Nayab Bhutta

Don’t Trade Without This! Why I Switched from...

The Indicator Addiction (That Almost Ruined My Trading)

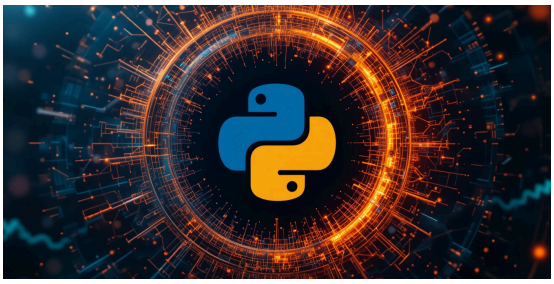
Jun 12 152 3

Swapnilphutane

How I Built a Multi-Market Trading Strategy That Pass...

When I first got into trading, I had no plans of building a full-blown system....

6d ago 16 1



Unicorn Day

The Quest for the Perfect Trading Score: Turning...

Navigating the financial markets... it feels like being hit by a tsunami of da...

3d ago 43

See more recommendations