Member-only story

A Fourier Transform-Based Trading Strategy in Backtrader

PyQuantLab Following 13 min read · Jun 8, 2025

Traditional technical analysis often relies on indicators derived from moving averages, oscillators, or volume. While effective, these methods might not fully capture the underlying cyclical nature of financial markets. Fourier Transform (FT), a powerful mathematical tool from signal processing, offers a different lens. It decomposes a time series (like stock prices) into its constituent frequencies, allowing us to identify dominant cycles within the data.

This tutorial explores how to integrate Fast Fourier Transform (FFT) into a `backtrader` strategy. We'll build a strategy that attempts to identify market cycles and generate trading signals based on the reconstructed price signal derived from these dominant cycles. To enhance robustness, we'll combine this with a simple moving average trend filter and incorporate a crucial stop-loss mechanism.

Looking to supercharge your algorithmic trading research? The Ultimate Algorithmic Strategy Bundle has you covered with over 80 Python strategies, fully documented in comprehensive PDF manuals:

Ultimate Algorithmic Strategy Bundle

Understanding Fourier Transform in Trading

What is Fourier Transform?

At its core, the Fourier Transform converts a signal from its original domain (e.g., time, in our case price over time) into a frequency domain. It reveals which frequencies (or cycles) are present in the original signal and their respective amplitudes.

In trading, this means we can:

1. Identify Cycles: Discover recurring patterns or “periods” within price data (e.g., a 20-day cycle, a 50-day cycle).
2. Filter Noise: By selecting only the most dominant frequencies, we can reconstruct a “smoother” price signal, effectively filtering out high-frequency noise that might lead to false signals.
3. Predict Future Movement (Implicitly): If dominant cycles are identified, their continuation could offer insights into potential future price direction.

The Strategy Concept

Our Fourier Transform strategy will follow these steps:

1. Price History Collection: Store a rolling window of recent closing prices.
2. Detrending: Remove the overall linear trend from the price data. This is crucial because FFT works best on stationary data (data without a strong trend).
3. FFT Application: Perform FFT on the detrended price data.
4. Dominant Frequency Selection: Identify the frequencies with the highest amplitudes. These represent the strongest cycles.
5. Signal Reconstruction: Reconstruct a “cleaner” price signal using only the selected dominant frequencies. This

reconstructed signal represents the underlying cycles.

6. Trading Signal Generation:

- Buy Signal: When the reconstructed signal turns upward (its trend is positive) AND the actual price is above a longer-term moving average (confirming an uptrend).
- Sell Signal: When the reconstructed signal turns downward (its trend is negative) AND the actual price is below a longer-term moving average (confirming a downtrend).

7. Risk Management: A fixed percentage stop-loss will be applied to all trades.

Prerequisites

Ensure you have the necessary Python libraries installed:

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

Step-by-Step Implementation

We'll break down the strategy into its core components.

1. Initial Setup and Data Acquisition

First, we'll set up our environment and download Bitcoin (BTC-USD) historical data.

```
import backtrader as bt
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

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

# Download historical data for Bitcoin (BTC-USD)
# Remember the instruction: yfinance download with auto_adjust=Fals
print("Downloading BTC-USD data from 2021-01-01 to 2024-01-01...")
data = yf.download('BTC-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 historical cryptocurrency price data. `auto_adjust=False` is used as per our persistent instruction.
- `data.columns = data.columns.droplevel(1)` : Flattens the multi-level column index from `yfinance` .

- `bt.feeds.PandasData` : Converts our cleaned pandas DataFrame into a format `backtrader` can consume.

2. The Fourier Transform Strategy (`FourierStrategy`)

This is where the core logic of our strategy resides, including the custom Fourier analysis function.

```
class FourierStrategy(bt.Strategy):
    params = (
        ('lookback', 30),         # Window for FFT analysis (number
        ('num_components', 3),    # Number of dominant frequency co
        ('trend_period', 30),     # Period for the Simple Moving Av
        ('stop_loss_pct', 0.02),  # Percentage for the stop-loss (e
    )

    def __init__(self):
        # Store a rolling history of closing prices for FFT analysi
        self.price_history = []

        # A simple moving average to act as a long-term trend filte
        self.trend_ma = bt.indicators.SMA(self.data.close, period=s

        # Variables to store the current reconstructed FFT signal v
        self.fft_signal = 0
        self.fft_trend = 0

        # Variables to keep track of active orders to prevent multi
        self.order = None       # Holds a reference to any active b
        self.stop_order = None  # Holds a reference to any active s

    def fourier_analysis(self, prices):
        """
        Performs FFT analysis on a given price series, reconstructs
        using dominant frequencies, and returns the current signal
        """
```

```
# Ensure we have enough data for the lookback period
if len(prices) < self.params.lookback:
    return 0, 0 # Return zeros if not enough data

# 1. Detrend the data: Remove the linear trend to make the
x = np.arange(len(prices)) # Create an array for the x-axis
coeffs = np.polyfit(x, prices, 1) # Fit a 1st degree polyno
trend_line = np.polyval(coeffs, x) # Calculate the trend li
detrended_prices = prices - trend_line # Subtract the trend

# 2. Apply Fast Fourier Transform (FFT)
fft_values = np.fft.fft(detrended_prices) # Compute FFT of
# frequencies = np.fft.fftfreq(len(detrended_prices)) # Not

# 3. Identify and select dominant frequencies
# Calculate the magnitude (amplitude) of each frequency com
magnitude = np.abs(fft_values)
# Sort indices by magnitude and select the top 'num_compone
# We take the top 'num_components' from both positive and n
# selecting 'num_components' dominant cycles.
# Ensure we don't pick the 0th (DC) component for cycles if
# For simplicity, we sort and pick the largest `num_compone

# Exclude the DC component (index 0) if it's not a desired
# Create a temporary array, set magnitude at index 0 to -in
temp_magnitude = magnitude.copy()
if self.params.num_components > 0: # Only if we actually ne
    temp_magnitude[0] = -np.inf # Ignore DC component for c

dominant_indices = np.argsort(temp_magnitude)[-self.params.

# 4. Reconstruct signal using only the dominant frequencies
reconstructed_fft_spectrum = np.zeros_like(fft_values, dtyp
# Copy only the FFT values corresponding to dominant freque
reconstructed_fft_spectrum[dominant_indices] = fft_values[d

# Apply inverse FFT to convert back to time domain
reconstructed_signal = np.real(np.fft.ifft(reconstructed_ff

# 5. Extract current signal value and its directional trend
current_signal = reconstructed_signal[-1] # The last value
# The trend of the reconstructed signal (current value minu
signal_trend = reconstructed_signal[-1] - reconstructed_sig
```

```
        return current_signal, signal_trend

    def notify_order(self, order):
        # This method is called by Cerebro whenever an order's stat
        if order.status in [order.Completed]:
            # If a buy order was completed and we have a long posit
            if order.isbuy() and self.position.size > 0:
                # Set a stop-loss order below the entry price
                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.pri
            # If a sell order (for shorting) was completed and we h
            elif order.issell() and self.position.size < 0:
                # Set a stop-loss order above the entry price
                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.exe

        # If the order is completed, canceled, or rejected, clear t
        if order.status in [order.Completed, order.Canceled, order.
            self.order = None # Clear main order reference
            if order == self.stop_order: # If the completed order w
                self.stop_order = None # Clear stop-loss order refe

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

    def next(self):
        # Prevent new orders if there's already an active order pen
        if self.order is not None:
            return

        # Store current price in history
        self.price_history.append(self.data.close[0])

        # Keep only the 'lookback' most recent prices
        if len(self.price_history) > self.params.lookback:
            self.price_history = self.price_history[-self.params.lo

        # Ensure we have enough data for FFT analysis
        if len(self.price_history) < self.params.lookback:
            return # Not enough data yet, skip this bar
```

```
        # Perform Fourier analysis on the current price history
        signal, signal_trend = self.fourier_analysis(np.array(self.

        # Update internal FFT signal values for the next iteration
        prev_signal = self.fft_signal # Store the previous signal v
        self.fft_signal = signal      # Update with the current sig
        self.fft_trend = signal_trend # Update with the current sig

        # Trading logic:
        # Long Entry/Short Exit: Reconstructed signal turns upward
        if (signal_trend > 0 and prev_signal < 0 and
            self.data.close[0] > self.trend_ma[0]): # Check if curr

            if self.position.size < 0:  # If currently in a short p
                self.log(f'CLOSING SHORT POSITION (FFT Signal Up),
                if self.stop_order is not None:
                    self.cancel(self.stop_order) # Cancel any activ
                self.order = self.close() # Close the short positio
            elif not self.position:  # If not in any position
                self.log(f'OPENING LONG POSITION (FFT Signal Up & T
                self.order = self.buy() # Execute a buy order

        # Short Entry/Long Exit: Reconstructed signal turns downwar
        elif (signal_trend < 0 and prev_signal > 0 and
              self.data.close[0] < self.trend_ma[0]): # Check if cu

            if self.position.size > 0:  # If currently in a long po
                self.log(f'CLOSING LONG POSITION (FFT Signal Down),
                if self.stop_order is not None:
                    self.cancel(self.stop_order) # Cancel any activ
                self.order = self.close() # Close the long position
            elif not self.position:  # If not in any position
                self.log(f'OPENING SHORT POSITION (FFT Signal Down
                self.order = self.sell() # Execute a sell order
```

Explanation of `FourierStrategy`:

- `params` : Configurable parameters for the strategy:
- `lookback` : The window size (number of past bars) for the FFT analysis.
- `num_components` : The number of dominant frequency components to use for signal reconstruction. Choosing this value carefully is crucial. Too few might oversimplify, too many might include noise.
- `trend_period` : Period for the SMA used as a longer-term trend filter.
- `stop_loss_pct` : Percentage for the stop-loss.
- `__init__(self)` :
- `self.price_history` : A list to manually store recent closing prices, which will be passed to `fourier_analysis` .
- `self.trend_ma` : An instance of `bt.indicators.SMA` to act as a simple trend filter.
- `self.fft_signal` , `self.fft_trend` : Variables to store the reconstructed signal's value and its current trend.
- `self.order` , `self.stop_order` : Standard `backtrader` variables for managing orders.
- `fourier_analysis(self, prices)` : This is a custom helper function within the strategy that performs the FFT.

- Detrending: `np.polyfit` and `np.polyval` are used to fit a linear trend to the `prices` and then remove it. This is a common preprocessing step for FFT on financial data.
- FFT: `np.fft.fft(detrended_prices)` computes the Fast Fourier Transform.
- Dominant Frequency Selection: `np.abs(fft_values)` gives the magnitude (amplitude) of each frequency. `np.argsort` finds the indices of the largest magnitudes. `num_components` determines how many of these dominant cycles we'll use. *Crucially,* `temp_magnitude[0] = -np.inf` *is used to ignore the DC (0th frequency) component, which represents the average level and not a cycle.*
- Signal Reconstruction: `np.zeros_like(fft_values)` creates an empty array. Only the FFT values corresponding to the `dominant_indices` are copied. Then, `np.fft.ifft` (inverse FFT) is used to convert this filtered frequency spectrum back into a time-domain signal. `np.real` takes the real part, as FFT results can be complex numbers.
- Signal and Trend Extraction: It returns the last value of the `reconstructed_signal` (the current signal) and the difference between the current and previous reconstructed signal values (the signal's trend).
- `notify_order(self, order)` : This `backtrader` callback is standard for handling order updates and placing stop-loss

orders.

- `log(self, txt, dt=None)` : A utility for logging messages.
- `next(self)` : The main trading logic, called for each new data bar.
- It updates `self.price_history` with the current close price, maintaining a rolling window of `lookback` length.
- It calls `self.fourier_analysis` to get the `signal` and `signal_trend` from the reconstructed FFT signal.
- Trading Signals:
- Long: Enters a long position (or closes a short) if `signal_trend > 0` (reconstructed signal is rising), `prev_signal < 0` (reconstructed signal just turned positive from negative, indicating a potential reversal to uptrend), AND `self.data.close[0] > self.trend_ma[0]` (current price is above the long-term trend MA). This combination aims to catch upward turns in cycles when the overall market trend is also up.
- Short: Enters a short position (or closes a long) if `signal_trend < 0` (reconstructed signal is falling), `prev_signal > 0` (reconstructed signal just turned negative from positive), AND `self.data.close[0] < self.trend_ma[0]` (current price is below the long-term trend MA). This aims to catch downward turns in cycles when the overall market trend is also down.

- As in previous strategies, `self.cancel(self.stop_order)` is crucial when closing a position due to a signal.

3. Backtesting Setup and Execution

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

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

# Add the strategy
cerebro.addstrategy(FourierStrategy)

# 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
```

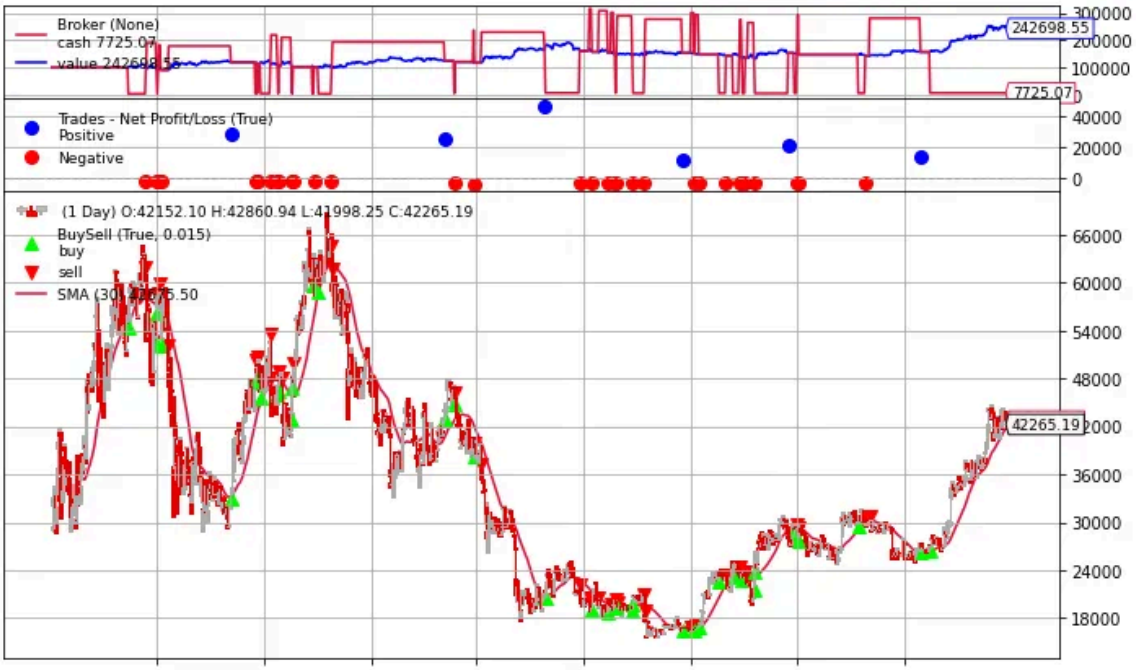
```
    # 5. System Quality Number (SQN) - Dr. Van Tharp's measure of syste
    sqn_analysis = strat.analyzers.sqn.get_analysis()
    print(f"System Quality Number (SQN): {sqn_analysis.get('sqn', 'N/A'


    # --- 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

        # You can also add reconstructed signal to plot, but requires m
        # or plotting outside Cerebro. For now, we rely on the trading

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

Explanation of Backtesting Setup:

- `cerebro.addanalyzer(...)` : We add several `backtrader` analysis modules to get a detailed statistical breakdown of the

strategy's performance, including risk-adjusted returns, drawdown, trade statistics, and the System Quality Number.

- Result Printing: The code then accesses the results from these analyzers and prints them, providing a comprehensive overview of the strategy’s profitability, risk, and trade characteristics.
- Plotting Enhancements:
- `plt.rcParams['figure.max_open_warning'] = 0` : Suppresses warnings for creating too many figures.
- `plt.rcParams['agg.path.chunksize'] = 10000` : Improves plotting performance for very large datasets by breaking down paths into smaller chunks.
- `cerebro.plot(iplot=False, style='candlestick', volume=False, plotreturn=True)` : Generates a static plot. `volume=False` is set to avoid plotting volume, which might not be directly interpreted with FFT signals, keeping the plot cleaner. `plotreturn=True` adds the equity curve.
- `try-except` block: Catches potential plotting errors, allowing the backtest results to still be displayed even if plotting fails.



Advantages and Challenges of FFT in Trading

Advantages:

- Cycle Detection: Can potentially identify underlying cyclical patterns that are not obvious with traditional indicators.
- Noise Reduction: By focusing on dominant frequencies, FFT can filter out high-frequency noise, leading to smoother signals.
- Unique Perspective: Offers a different way to analyze market data compared to standard time-domain indicators.

Challenges and Considerations:

1. Non-Stationarity of Financial Data: FFT assumes stationary data (mean, variance, and autocorrelation are constant over time). Financial time series are often non-stationary (e.g., strong trends). The detrending step helps but doesn't fully solve this.

2. Lookback Period (`lookback`): The choice of `lookback` period is critical. It determines the window of data used for FFT. A short window might miss longer cycles; a long window might include outdated information.

3. Number of Components (`num_components`): Deciding how many dominant frequencies to include for reconstruction is a major tuning parameter. Too few might oversimplify; too many might reintroduce noise. This often requires empirical optimization.

4. Lag: Despite detrending, there's still inherent lag in FFT due to requiring a history of data. The signal for the *current* bar is based on *past* data.

5. Reversal vs. Continuation: FFT can be good at detecting cycles, which inherently implies reversals. However, markets also trend strongly. Combining it with a trend filter (like our `trend_ma`) is a good approach to mitigate this.

6. Computational Cost: For very large datasets or very short timeframes, performing FFT on every `next` call can be

computationally intensive, though typically manageable for daily data.

7. Overfitting Risk: With many parameters (`lookback` , `num_components` , `trend_period` , `stop_loss_pct`), there's a significant risk of overfitting the strategy to historical data. Rigorous out-of-sample testing and optimization are paramount.

Further Enhancements

1. Dynamic `num_components` : Instead of a fixed number, adjust `num_components` based on the energy (total magnitude) in the frequency spectrum, or filter out frequencies above a certain noise threshold.
2. Adaptive Lookback: Make the `lookback` period adaptive, similar to your adaptive MA strategy, perhaps based on market volatility or autocorrelation.
3. Phase Analysis: Beyond magnitude, Fourier Transform also provides phase information. This could potentially be used to predict the “turning point” of a cycle with more precision.
4. Other Trend Filters: Experiment with more sophisticated trend filters (e.g., ADX, higher-timeframe MAs) to improve entries/exits.
5. Profit Taking: Implement trailing stops or profit targets in addition to the fixed stop-loss.

6. Optimization: Use `backtrader`'s `optstrategy` to systematically find the best `lookback`, `num_components`, `trend_period`, and `stop_loss_pct` parameters. This is crucial for validating the strategy.

7. Visualization of Reconstructed Signal: While not directly plotted by `cerebro.plot` in this example, you could modify `FourierStrategy` to expose the `fft_signal` as a `bt.Line` and then plot it. This would allow visual inspection of how the reconstructed signal tracks the price.

Conclusion

This tutorial has provided a detailed walkthrough of implementing a Fourier Transform-based trading strategy in `backtrader`. By leveraging FFT, we attempt to tap into the cyclical patterns of financial markets, combining this with a trend filter and robust risk management. While powerful, FFT-based strategies require careful parameter tuning and a deep understanding of their assumptions and limitations. This strategy serves as an excellent foundation for exploring the fascinating intersection of signal processing and quantitative trading.

Python | Algorithmic Trading | Quantitative Finance | Backtesting | Bitcoin

Written by PyQuantLab

655 followers · 6 following

Following

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

Responses (1)

Steven Feng CAI

What are your thoughts?

Swalk

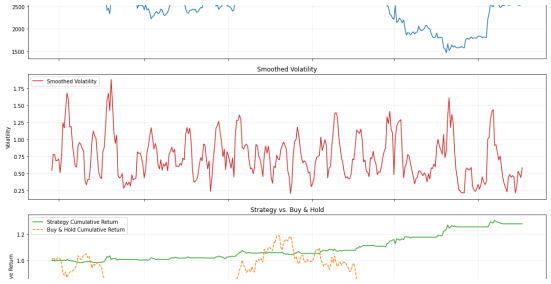
Jun 8

Nice

Could have been so much better with backtest results vs benchmarks

1 Reply

More from PyQuantLab



PyQuantLab

Volatility Clustering Trading Strategy with Python

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

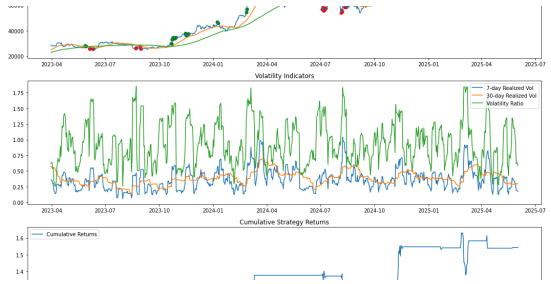
Jun 3 32 3

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

PyQuantLab

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

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

Jun 3 60

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

Jun 4 64

See all from PyQuantLab

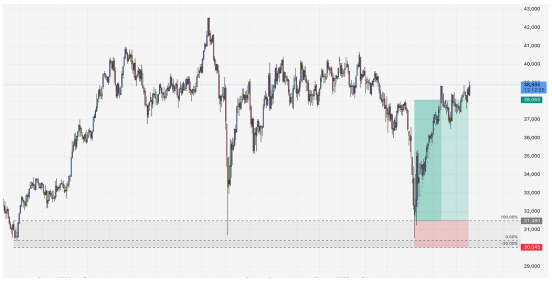
Recommended from Medium

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

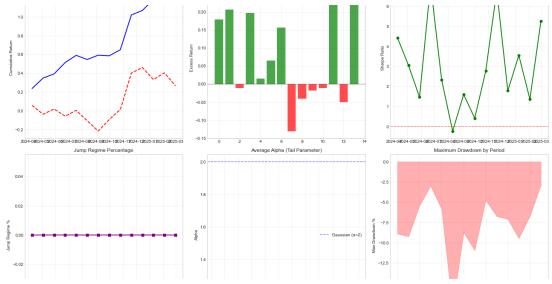


Candence

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

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

Jun 19 2



PyQuantLab

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

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

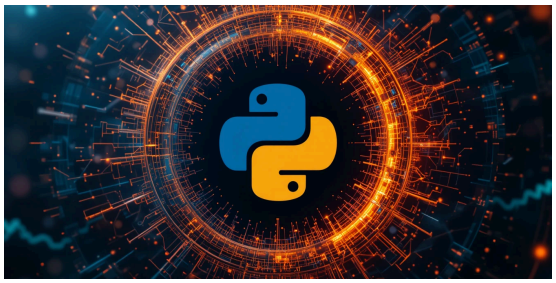
Jun 5 102

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



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

Trading Dude

Battle-Tested Backtesters: Comparing VectorBT, Ziplin...

When it comes to developing and validating trading strategies in Pytho...

Jun 13 13

See more recommendations