Member-only story

VIDYA with Adaptive Smoothing via Chande Momentum Oscillator

PyQuantLab Following 6 min read · May 30, 2025

Before we dive in, make sure you check out the pack I put together to get you started with python and algo trading in no time. It includes **three in-depth books** covering everything from Backtrader basics to advanced machine-learning strategies — **plus** a **code pack of 30+ Python algorithms.** Whether you're mastering SMA/EMA crossovers, Bollinger Band mean-reversion, ADX trend filters, or Kalman-filtered neural nets, this **Algo Trading Value Pack** delivers **step-by-step guides** and **ready-to-run scripts** so you can:

- 🚀 **Get up and running in minutes** with Backtrader essentials
- 📊 **Execute pro-grade strategies** without reinventing the wheel
- 💡 **Discover advanced techniques** like HMM regime shifts and adaptive volatility

👉 Grab it now at a **limited-time price** and transform your trading:

Now, let's get to our strategy. Traditional moving averages use a fixed lookback period, which can make them lag in fast-moving markets or produce too many signals in choppy conditions. The Variable Index Dynamic Average (VIDYA), developed by Tushar Chande, addresses this by dynamically adjusting its smoothing factor. In this exploration, we'll construct a VIDYA whose smoothing is modulated by the magnitude of the Chande Momentum Oscillator (CMO), causing it to "auto-accelerate" in strong momentum and slow down in weak momentum. We'll

walk through its Python implementation for `ETH-USD` and discuss how to analyze its behavior.

Core Concepts: CMO, Adaptive Alpha, and VIDYA

1. Chande Momentum Oscillator (CMO):

The CMO, also developed by Tushar Chande, measures pure momentum. It's calculated as:

$$CMO = 100 \times (Su - Sd) / (Su + Sd)$$

where Su is the sum of upward price movements over a period (e.g., 14 days), and Sd is the sum of downward movements. CMO values range from -100 (strongest bearish momentum) to +100 (strongest bullish momentum), with values near 0 indicating low momentum or a balance between up/down moves.

For VIDYA, we use the absolute magnitude of CMO (abs(CMO)) as an indicator of momentum strength, irrespective of direction. abs(CMO) ranges from 0 to 100.

2. Adaptive Smoothing for VIDYA:

The core idea is to make the VIDYA (which is a type of Exponential Moving Average) faster when momentum is strong and slower when it's weak.

- Strong Momentum (High `abs(CMO)`): The VIDYA will use a shorter effective EMA period, making it more sensitive to recent price changes.
- Weak Momentum (Low `abs(CMO)`): The VIDYA will use a longer effective EMA period, providing more smoothing and filtering out noise. To achieve this, we normalize `abs(CMO)` (from its 0-100 range to a 0-1 scale) and use this normalized value to interpolate an effective EMA period between a predefined minimum (`period_vidya_min`) and maximum (`period_vidya_max`).

3. VIDYA Calculation:

VIDYA is calculated iteratively like an EMA:

$$VIDYA_t = \alpha_t \cdot Price_t + (1 - \alpha_t) \cdot VIDYA_{t-1}$$

The crucial difference is that the smoothing factor $\alpha_t = 2/(AdaptivePeriod_t + 1)$ changes daily, based on the AdaptivePeriod_t derived from the previous day's abs(CMO).

4. Trading Strategy & Risk:

We'll use a standard price crossover with this adaptive VIDYA. Trades are managed with an ATR-based trailing stop loss, and commissions are factored in.

Python Implementation Highlights

Let's examine key code sections.

Snippet 1: Calculating CMO and the Adaptive VIDYA Period

This snippet shows how CMO is calculated and its absolute magnitude is then normalized and mapped to an adaptive EMA period for VIDYA.

Python

```
import pandas_ta as ta # For Chande Momentum Oscillator

# --- Parameters (example for ETH-USD) ---
# cmo_period = 14
# period_vidya_min = 10  # Faster EMA period for high abs(CMO)
# period_vidya_max = 60  # Slower EMA period for low abs(CMO)

# --- Column Names ---
# cmo_col_name = f"CMO_{cmo_period}"
# norm_abs_cmo_col = "Normalized_Abs_CMO"
# adaptive_vidya_period_col = "Adaptive_VIDYA_Period"

# --- Indicator Calculation (within pandas DataFrame 'df') ---
# Assume df has 'Close' column.

# 1. Calculate Chande Momentum Oscillator (CMO)
# pandas_ta might need lowercase 'close' or you can pass df['Close'
df[cmo_col_name] = ta.momentum.cmo(df['Close'], length=cmo_period)
df[cmo_col_name].fillna(0, inplace=True) # Fill initial NaNs with 0

# 2. Normalized Absolute CMO (0 to 1 scale)
# abs(CMO) ranges from 0 to 100.
abs_cmo = df[cmo_col_name].abs()
```

```
df[norm_abs_cmo_col] = (abs_cmo / 100.0).clip(0, 1)

# 3. Adaptive VIDYA Period (based on lagged normalized absolute CMO
# High NormAbsCMO (1.0, strong momentum) -> period_vidya_min (faste
# Low NormAbsCMO (0.0, weak momentum) -> period_vidya_max (slower)
df[adaptive_vidya_period_col] = period_vidya_max - df[norm_abs_cmo_
# Fill NaNs (e.g., first row due to shift) with a mid-range period
df[adaptive_vidya_period_col] = np.round(df[adaptive_vidya_period_c
    (period_vidya_min + period_vidya_max) / 2
).astype(int)
# Ensure period is within defined min/max bounds
df[adaptive_vidya_period_col] = np.clip(df[adaptive_vidya_period_co
```

The `adaptive_vidya_period_col` now dictates the responsiveness of our VIDYA filter each day.

Snippet 2: Iterative Calculation of VIDYA (Adaptive EMA)

The VIDYA is then calculated day by day, using the adaptive period to derive a dynamic smoothing factor α.

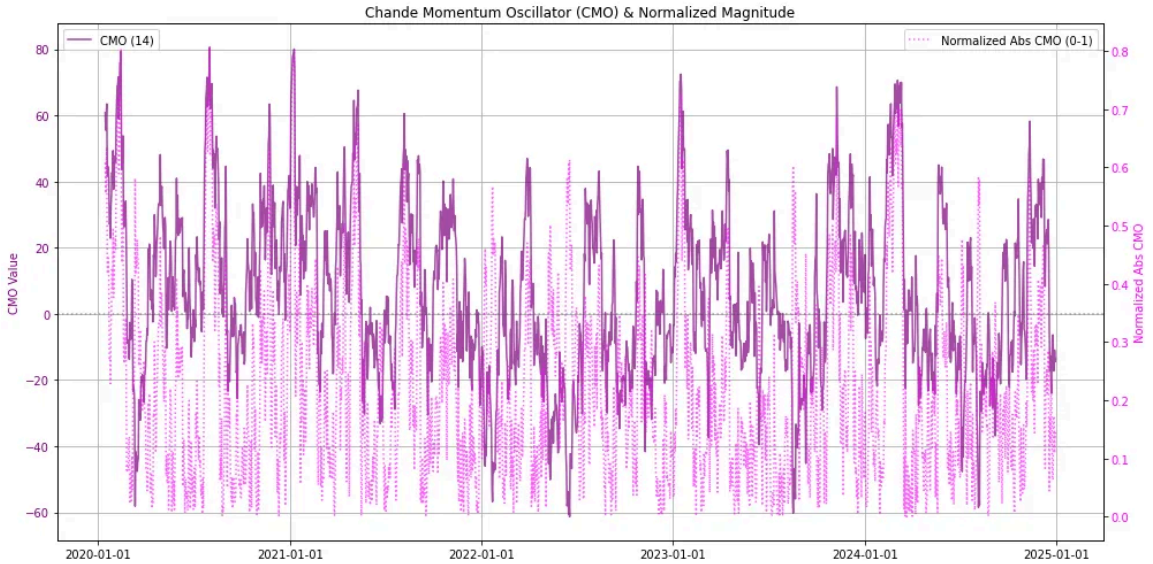
Python

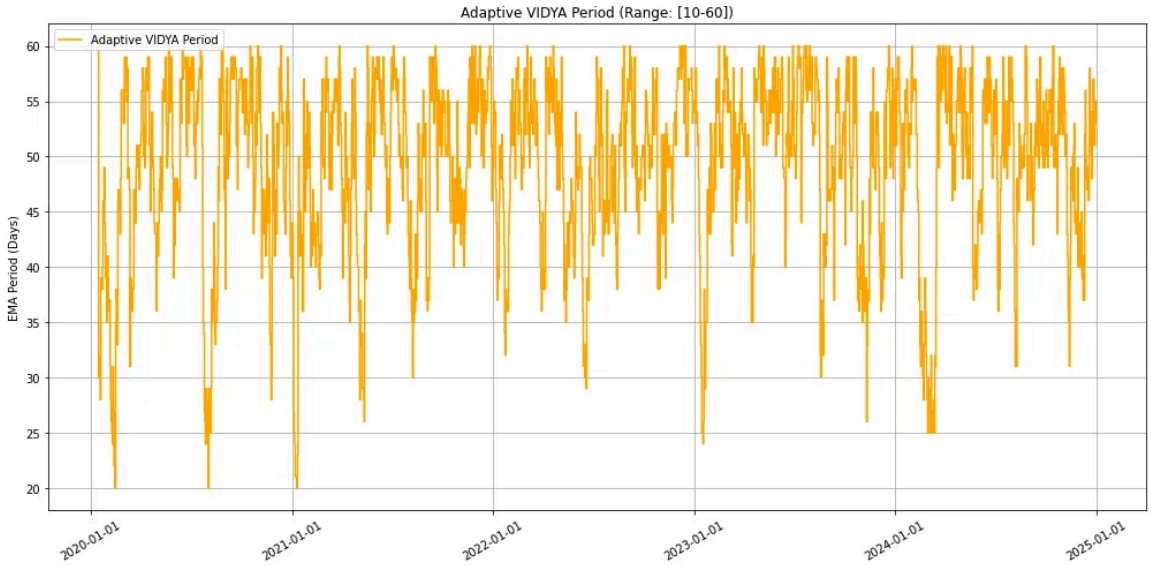
```
# --- Column Names ---
# adaptive_vidya_period_col = "Adaptive_VIDYA_Period" (calculated a
# vidya_filter_col = "VIDYA_Filter"

# --- Iterative VIDYA Calculation (within pandas DataFrame 'df') --
# Alpha for today's VIDYA is based on today's adaptive_vidya_period
# (which itself was based on yesterday's CMO magnitude)
df['Alpha_Adaptive_VIDYA'] = 2 / (df[adaptive_vidya_period_col] + 1
```

```
df[vidya_filter_col] = np.nan # Initialize column

# Seed the first VIDYA value (e.g., with the first close price wher
first_valid_alpha_idx = df['Alpha_Adaptive_VIDYA'].first_valid_inde
if first_valid_alpha_idx is not None:
    df.loc[first_valid_alpha_idx, vidya_filter_col] = df.loc[first_

    start_loc_for_ema_loop = df.index.get_loc(first_valid_alpha_idx

    for i_loop in range(start_loc_for_ema_loop + 1, len(df)):
        idx_today = df.index[i_loop]
        idx_prev = df.index[i_loop-1]

        alpha_val = df.loc[idx_today, 'Alpha_Adaptive_VIDYA']
        current_close_val = df.loc[idx_today, 'Close']
        prev_filtered_price_val = df.loc[idx_prev, vidya_filter_col

        if any(pd.isna(val) for val in [alpha_val, current_close_va
            # Carry forward previous filtered price if any componen
            df.loc[idx_today, vidya_filter_col] = prev_filtered_pri
        else:
            # Standard EMA formula with adaptive alpha
            df.loc[idx_today, vidya_filter_col] = alpha_val * curre

# Fill any remaining NaNs at the start
df[vidya_filter_col].fillna(method='ffill', inplace=True)
df[vidya_filter_col].fillna(method='bfill', inplace=True) # Ensure
```

This process generates the `vidya_filter_col`, our dynamically adapting trendline.


ETH-USD Price & VIDYA Filter
Close
VIDYA Filter
Trailing Stop
Price
5000
4000
3000
2000
1000
0
2020-01-01
2021-01-01
2022-01-01
2023-01-01
2024-01-01
2025-01-01

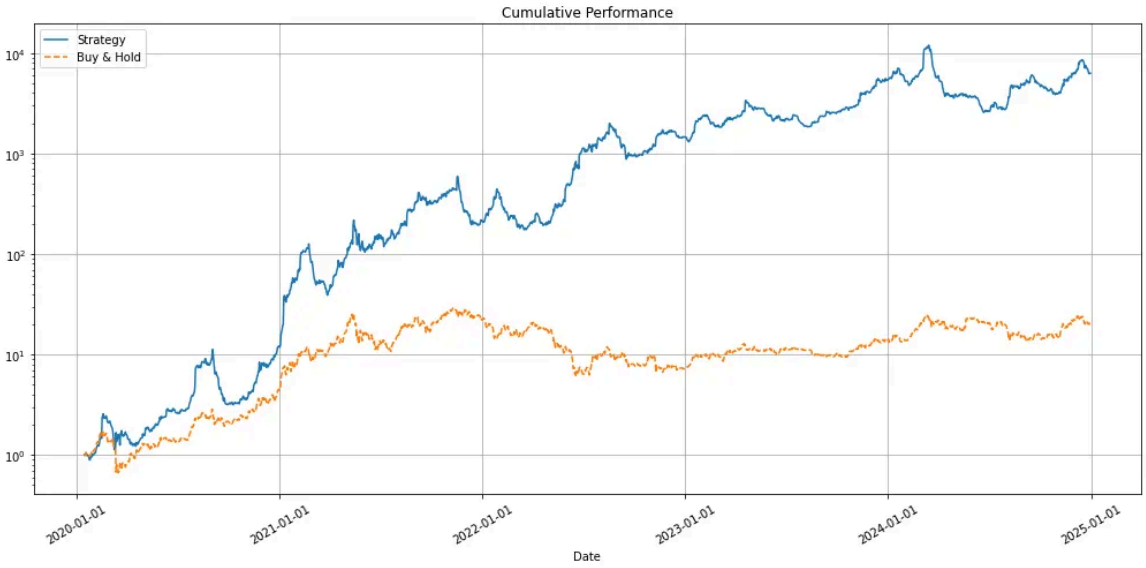

Chande Momentum Oscillator (CMO) & Normalized Magnitude
CMO (14)
Normalized Abs CMO (0-1)
CMO Value
Normalized Abs CMO
80
60
40
20
0
−20
−40
−60
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2020-01-01
2021-01-01
2022-01-01
2023-01-01
2024-01-01
2025-01-01



Strategy Execution, Backtesting, and Interpretation

The full Python script (structured with a main backtesting function like `run_vidya_crossover_backtest` and a separate plotting function `plot_vidya_results_separate_windows`) then implements:

- Trading Signals: Price (`Close[t-1]`) crossing `VIDYA_Filter[t-1]`, with trades entered at `Open[t]`.
- Risk Management: An ATR trailing stop loss and per-side commissions.
- Performance Evaluation: Standard metrics (Cumulative Return, Annualized Return/Volatility, Sharpe Ratio, Max Drawdown, Trade Count) are calculated and compared to a Buy & Hold benchmark.



Critical Considerations for Research:

- Parameter Sensitivity: How robust are the results to changes in `cmo_period`, the `period_vidya_min`/`max` range, and ATR settings?
- CMO Normalization: Using `abs(CMO)/100` is a direct approach. Other normalization or scaling methods for CMO magnitude could be explored.
- Transaction Costs: With frequent adaptations and potential crossovers, the impact of commissions (e.g., 2 bps per side in the example) on net profitability is vital.
- Out-of-Sample Validation: The strategy should be tested on data not used for development or parameter exploration to gauge its true robustness.

Conclusion

VIDYA, when adapted using the Chande Momentum Oscillator's magnitude, offers an intuitive method for creating a more dynamic moving average. By becoming more sensitive during periods of strong momentum and more placid during lulls, it aims to better align with the market's current tempo. The Python framework allows for a detailed examination of this concept. As with all adaptive strategies, while the allure of self-adjustment is strong, rigorous testing for parameter robustness, out-of-sample performance, and the impact of real-world trading frictions is essential for any practical consideration. This approach, however, provides a compelling framework for deeper research into momentum-adaptive filtering techniques.

Python | Algorithmic Trading | Quantitative Finance | Backtesting | Crypto

Written by PyQuantLab

655 followers · 6 following

Following

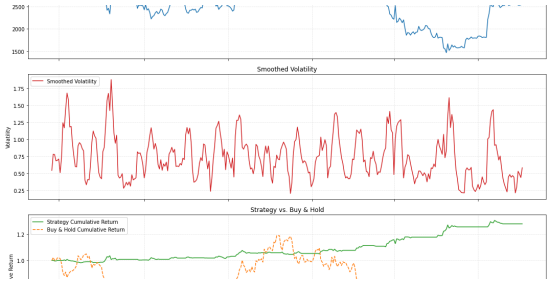
Your go-to place for Python-based quant tutorials, strategy deep-dives, and reproducible code. For more visit our website: www.pyquantlab.com

No responses yet

Steven Feng CAI

What are your thoughts?

More from PyQuantLab



PyQuantLab

Volatility Clustering Trading Strategy with Python

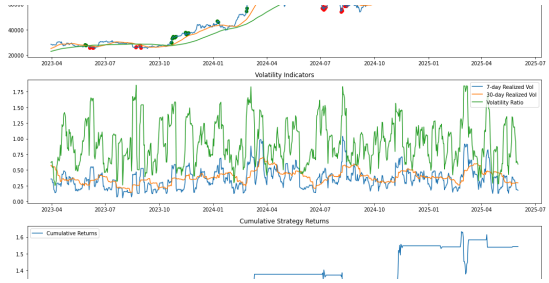
Ultimate Algorithmic Strategy Bundle has you covered with over 80 Python...

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



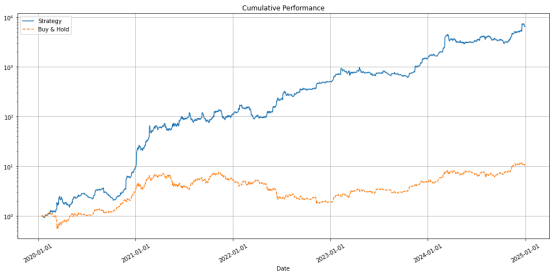
PyQuantLab

Trend-Volatility Confluence Trading Strategy

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

Jun 3 60

Jun 9 54 1



PyQuantLab

Filtering Through Market Noise: The Time-Decay...

Traditional Exponential Moving Averages (EMAs) are a staple in...

Jun 1 69 1

See all from PyQuantLab

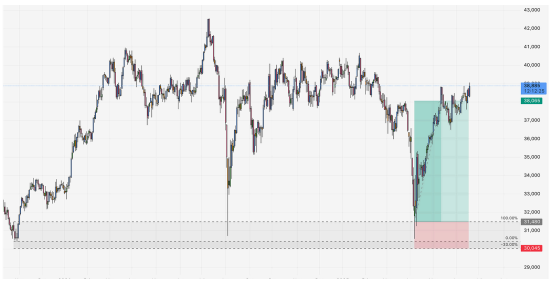
Recommended from Medium

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

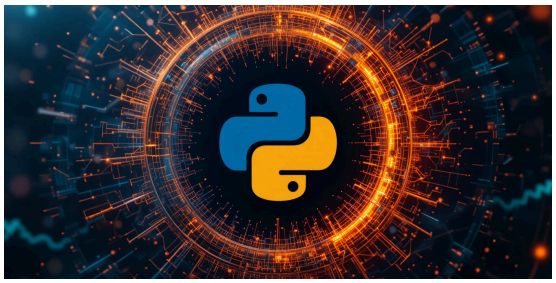


Candence

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

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

Jun 19 2



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

Navnoor Bawa

QOADRS Mathematical Implementation: A Step-by-...

How we achieved 90.7% accuracy processing 9,462 real options...

Jun 13 2

Remedy

How I Turned My Trading Strategy into a Professional...

By Chinedu Uzochukwu

6d ago 1 1

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