Trend Finder Method Using Wavelet Transform

Ehsan Khademolama

June 12, 2024

1 Introduction

In this report, we present a novel method for identifying trends in financial time series data using wavelet transforms. The method applies Discrete Wavelet Transform (DWT) to the closing prices of a stock to determine the underlying trend.

2 Wavelet Transform

Wavelet Transform is a mathematical technique that decomposes a time series into different frequency components, making it possible to analyze various aspects of the data at different scales. Unlike traditional methods like the Exponential Moving Average (EMA), wavelet transform provides better localization in both time and frequency domains.

2.1 Advantages Over EMA

One of the primary advantages of using wavelet transform over EMA is the reduced delay. EMA, while smoothing out short-term fluctuations, introduces a lag between the signal and its smoothed version. This lag can be detrimental for real-time trend identification and trading decisions.

Wavelet transform, on the other hand, captures both high and low-frequency components, allowing for more accurate and timely trend detection. By reconstructing the approximation component from the wavelet transform, our method can detect trend changes with less delay compared to EMA, providing a more responsive and accurate trend identification.

3 Methodology

The core of our methodology is implemented in Python, leveraging several libraries including `numpy`, `pandas`, `yfinance`, `pywt`, and `mplfinance`. The steps involved in the trend identification process are detailed below.

3.1 Class Definition

The `TrendIdentifier` class encapsulates the wavelet-based trend identification logic. The constructor allows the user to specify the wavelet type, with 'db2' as the default.

Listing 1: TrendIdentifier Class

```
import numpy as np
import pandas as pd
import pywt

class TrendIdentifier:
def __init__(self, wavelet='db2'):
self.wavelet = wavelet

def identify_trend(self, df):
df['Trend'] = 'sideways'
df['Trend'] = df['Trend'].astype(object)

data = df['Close'].values
if len(data) % 2 != 0:
data = np.pad(data, (0, 1), 'constant',
    constant_values=(data[-1],))

cA, cD = pywt.dwt(data, self.wavelet, 'smooth')
approx = pywt.idwt(cA, np.zeros_like(cD), self.
    wavelet)

if len(approx) > len(df):
approx = approx[:len(df)]
elif len(approx) < len(df):
approx = np.pad(approx, (0, len(df) - len(approx)),
     'edge')

df['approx'] = approx

for i in range(3, len(df)):
if (df['approx'].iloc[i] > df['approx'].iloc[i-1]
    and
df['approx'].iloc[i] > df['approx'].iloc[i-2] and
df['approx'].iloc[i] > df['approx'].iloc[i-3]):
df.loc[df.index[i], 'Trend'] = 'up'
elif (df['approx'].iloc[i] < df['approx'].iloc[i-1]
     and
df['approx'].iloc[i] < df['approx'].iloc[i-2] and
df['approx'].iloc[i] < df['approx'].iloc[i-3]):
```

```
df.loc[df.index[i], 'Trend'] = 'down'
else:
df.loc[df.index[i], 'Trend'] = 'sideways'

return df
```

3.2 Fetching Historical Data

We use the `yfinance` library to fetch historical stock data. The following function retrieves the data for a given symbol, interval, and period.

Listing 2: Fetching Historical Data

```
import yfinance as yf

def fetch_historical_data(symbol, interval='1d',
    period='1y'):
df = yf.download(symbol, period=period, interval=
    interval)
df.dropna(inplace=True)
return df
```

3.3 Applying the Trend Identifier

An instance of `TrendIdentifier` is created and used to identify trends in the fetched data. Additionally, the trends are visualized using `mplfinance`.

Listing 3: Applying the Trend Identifier

```
import matplotlib.pyplot as plt
import mplfinance as mpf

# Fetch historical data
df = fetch_historical_data("AAPL", "1d", "6mo")
trend_identifier = TrendIdentifier(wavelet='db24')
result_df = trend_identifier.identify_trend(df)

# Add arrows for trends
arrows_up = np.nan * np.ones(len(result_df))
arrows_down = np.nan * np.ones(len(result_df))
for i in range(len(result_df)):
if result_df['Trend'].iloc[i] == 'up':
arrows_up[i] = result_df['High'].iloc[i] + 0.1
elif result_df['Trend'].iloc[i] == 'down':
arrows_down[i] = result_df['Low'].iloc[i] - 0.1
```

```
# Convert arrows to pandas Series with datetime
    index
arrows_up_series = pd.Series(arrows_up, index=
    result_df.index)
arrows_down_series = pd.Series(arrows_down, index=
    result_df.index)

# Plotting the candlestick chart along with trends
apds = [
mpf.make_addplot(result_df['approx'], color='blue',
     panel=0, ylabel='Approximation'),
mpf.make_addplot(arrows_up_series, scatter=True,
    markersize=20, marker='^',
alpha=0.35, color='green', panel=0),
mpf.make_addplot(arrows_down_series, scatter=True,
    markersize=20, marker='v',
alpha=0.35, color='red', panel=0)
]

# Color coding trends
mc = mpf.make_marketcolors(up='g', down='r',
    inherit=True)
s = mpf.make_mpf_style(marketcolors=mc)

# Save the plot as an SVG file
mpf.plot(result_df, type='candle', addplot=apds,
    style=s,
title='Candlestick␣Chart␣with␣Wavelet␣Trends',
    volume=False,
savefig='candlestick_chart_with_wavelet_trends.svg'
    )
```

4 Results

The generated candlestick chart, along with the identified trends, is saved as a figure. The trends are marked with arrows, indicating upward and downward movements. Below is the illustration of the chart 2.

5 Conclusion

The Trend Finder method presented in this report successfully identifies and visualizes trends in financial time series data using wavelet transforms. This approach provides a robust way to analyze market trends, aiding in better decision-making for traders and analysts. Compared to traditional methods like

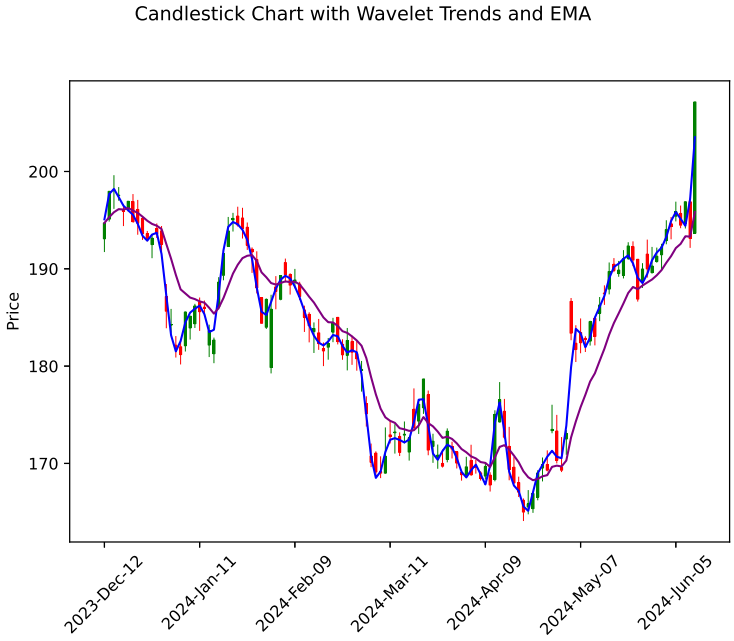

Figure 1: Candlestick Chart with Wavelet and EMA (10)

EMA, the wavelet-based approach offers improved responsiveness and accuracy in trend detection.

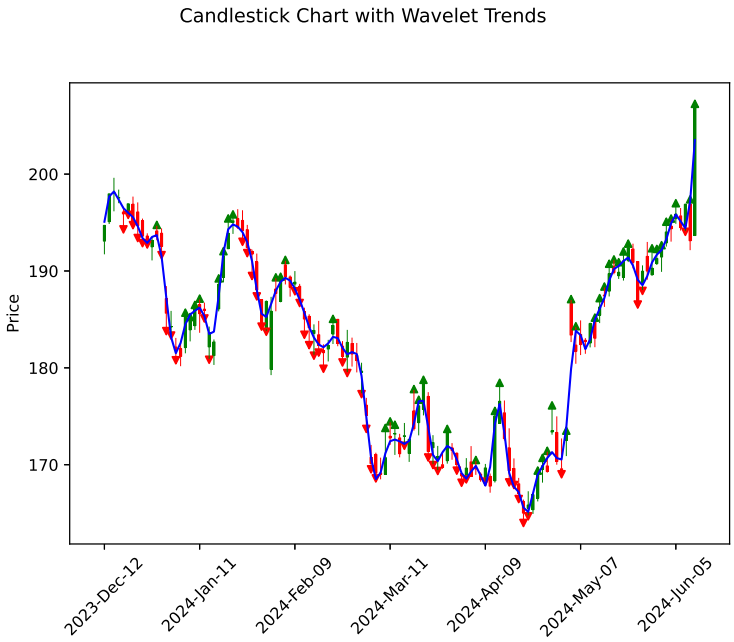

Figure 2: Candlestick Chart with Wavelet Trends arrows