# IMPORTANT DISCLOSURE

THE CODE IN THIS BOOK IS OUTDATED. I HIGHLY RECOMMEND YOU FOCUS ON THE THEORETICAL CONCEPTS AND NOT ON THE CODE.

COPYRIGHT © 2021 KAABAR SOFIEN

To my parents for always believing in me.

To my computer for putting up with my constant need for performance.

To my body for handling so much food to refrain from sleeping at nights writing this book.

To my eyes for burdening them with the constant need to stay awake during the nights.

To my wife who supported my absence because I was writing this book.

To everyone who reads this and gains something…

*Make it **quick**, make it **clear**, make it **practical**.*

# Q&A

**What is this book all about?**

This book is a modest attempt at presenting a more modern version of technical analysis based on objective measures rather than subjective ones. A sizeable chunk of this beautiful type of analysis revolves around technical indicators which is what this book covers. I believe it is time to be creative with indicators. The following chapters present trend-following indicators and how to code/use them. The code included in the book is available in the GitHub repository. A QR code link will be provided in the book.

**What am I going to gain?**

You will gain exposure to many new indicators and strategies that will change the way you think about trading, and you will find yourself busy experimenting and choosing the strategy that suits you the best.

**How is it organized?**

The order of the chapter is not very important, although reading the introductory Python chapter is helpful. The book is divided into four parts: Part 1 deals with different types of moving averages, Part 2 deals with trend-following indicators, Part3 deals with market regime detection techniques, and finally, Part 4 will present many different trend-following technical strategies.

**What level of knowledge do I need to follow this book?**

Although a basic or a good understanding of trading and coding is considered very helpful, it is not necessary. At the beginning of the book, I have included a chapter that deals with some Python concepts, but this book is not about Python.

# Table of Contents

INTRODUCTION TO DATA ANALYSIS IN PYTHON..... 4

**PART 1 MOVING AVERAGES** ..... 17

CHAPTER 1 THE SIMPLE MOVING AVERAGE ..... 18

CHAPTER 2 THE EXPONENTIAL MOVING AVERAGE ..... 24

CHAPTER 3 THE SMOOTHED MOVING AVERAGE ..... 27

CHAPTER 4 THE LINEAR-WEIGHTED MOVING AVERAGE ..... 30

CHAPTER 5 THE HULL MOVING AVERAGE ..... 34

CHAPTER 6 THE VOLATILITY MOVING AVERAGE ..... 38

CHAPTER 7 THE FIBONACCI MOVING AVERAGE..... 44

CHAPTER 8 THE ADAPTIVE MOVING AVERAGE..... 49

CHAPTER 9 THE FRACTAL ADAPTIVE MOVING AVERAGE ..... 54

CHAPTER 10 THE TRIANGULAR MOVING AVERAGE ..... 60

**PART 2 TREND-FOLLOWING INDICATORS**..... 62

CHAPTER 11 THE MACD OSCILLATOR..... 63

CHAPTER 12 THE AWESOME OSCILLATOR..... 67

CHAPTER 13 ICHIMOKU KINKO HYO ..... 70

CHAPTER 14 THE TREND INTENSITY INDEX ..... 82

CHAPTER 15 THE SUPERTREND INDICATOR..... 85

CHAPTER 16 THE DONCHIAN CHANNEL..... 90

CHAPTER 17 THE AVERAGE DIRECTIONAL INDEX..... 93

CHAPTER 18 THE SQUEEZE MOMENTUM INDICATOR ..... 98

CHAPTER 19 THE AROON OSCILLATOR ..... 105

CHAPTER 20 THE PARABOLIC STOP-AND-REVERSE..... 109

CHAPTER 21 THE VERTICAL HORIZONTAL FILTER..... 113
CHAPTER 22 THE TRIX INDICATOR..... 117
**PART 3 REGIME DETECTION TECHNIQUES**..... 120
CHAPTER 23 THE FIBONACCI RANGE..... 122
CHAPTER 24 THE GOPALAKRISHNAN RANGE INDEX..... 127
CHAPTER 25 HEIKIN ASHI ..... 132
CHAPTER 26 THE MARKET'S SLOPE..... 136
CHAPTER 27 THE ELDER IMPULSE SYSTEM..... 142
**PART 4 HANDS-ON TRADING STRATEGIES**..... 147
STRATEGY #1 BUYING THE DIPS & SELLING THE STRENGTHS..... 155
STRATEGY #2 MOVING AVERAGE CROSSOVERS ..... 159
STRATEGY #3 THE RSI & MOVING AVERAGES..... 162
STRATEGY #4 THE MACD CROSSOVER ..... 166
STRATEGY #5 THE MOVING AVERAGE SLOPE..... 169
STRATEGY #6 THE SUPERTREND FLIP..... 171
STRATEGY #7 THE PARABOLIC SAR & MOVING AVERAGES ..... 174
STRATEGY #8 PSYCHOLOGICAL LEVELS & MOVING AVERAGES .... 177
STRATEGY #9 INTENSITY MOVING AVERAGES..... 182
STRATEGY #10 HIDDEN DIVERGENCES ..... 185
STRATEGY #11 THE RSI NEUTRALITY CROSS..... 193
STRATEGY #12 THE VAMA FLIP ..... 196
STRATEGY #13 THE RSI & THE SUPERTREND ..... 199
STRATEGY #14 THE CATAPULT ..... 202
STRATEGY #15 VALHALLA..... 208
**CONCLUSION** ..... 211

# INTRODUCTION

Trend-following is a non-forecasting trading strategy that aims to profit from the aggregate direction of the current market. We tend to say riding the trend or jumping on the trend to signify trades that aim to be in line with the overall direction. In a bull (Rising) market, a trend-following trader would go long (Buy) and hold the position as long as she thinks the market will continue to go up. In parallel, in a bear (Falling) market, a trend-following trader would short (Sell) and ride the trend lower as long as he thinks the market will continue to drift downwards. Trend-following can be based on fundamental analysis, technical analysis, and even quantitative analysis. This book will discuss a wide array of technical indicators and strategies that aim to follow and exploit the current direction. It is composed of four different parts with **part 1** reserved for the well-known moving averages and their many different types, we will learn how to code and chart them. **Part 2** will present different trend-following indicators with their advantages and limitations. This part can be used as a mini encyclopedia of trend-following indicators. **Part 3** will discuss market regime detection techniques. Knowing whether the market is trending, or ranging is crucial for any trader, and this is precisely what we will cover through some state-detection indicators. And finally, **part 4** will show 15 trend-following strategies created after combining different indicators with new and exotic rules. The aim is to provide decent techniques based on a trade-off between signal frequency and above average relative performance when compared to classical indicators. With this being said, we need to understand that the strategies presented are not recommendations and must be back-tested by the reader himself while adding the personal risk management part and the necessary optimization. **Get your code on!**

# INTRODUCTION TO DATA ANALYSIS IN PYTHON

In research, we need the idea generation phase to have enough time to deliver good results and this cannot be done if we have a long process of transforming the ideas into visible results. Automating the process of importing historical data is valuable as it allows us to focus on more important non-repetitive tasks. By creating a few simple functions, we will be able to call thousands of OHLC[1] data in a matter of seconds. For this, we need to download a library and a software described below. But first, we need to download a Python interpreter such as SPYDER[2], PyCharm, or Jupyter.

## METATRADER5 SOFTWARE & LIBRARY

One of the most famous trading platforms in the retail community is the MetaTrader5 software. It is a powerful tool that comes with its own programming language and its huge online community support. Most importantly, it offers the possibility to export its historical short-term and long-term FX data. The first thing we need to do is to simply download the software from the official website. Then, after creating the unlimited demo account, we are ready to import the library in Python that allows us to import the OHLC data from MetaTrader5. A library is a group of structured functions that can be imported into the Python interpreter from where we can call and use the ones we want. The easiest way to install the MT5 library is to go to the Python prompt on our computer and type:

```
pip install MetaTrader5
```

[1] Open, High, Low, and Close values are the most basic form of financial historical data.
[2] https://www.anaconda.com/

This should install the library in our local Python. Now, we want to import it to the Python interpreter such as SPYDER so that we can use it. Let us import all the libraries we will be using for this.

```
import datetime # Date acquiring
import pytz # Time zone management
import pandas as pd # Mostly for Data frame manipulation
import MetaTrader5 as mt5 # Importing OHLC data
import matplotlib.pyplot as plt # Plotting charts
import numpy as np # Mostly for array manipulation
```

Anything that comes after "as" is a shortcut. The plt shortcut is there so that each time we want to call a function from that library we do not have to type the full matplotlib.pyplot statement. The official documentation for the library can be found in the official website[3].

**CREATING THE IMPORTING FUNCTIONS STEP-BY-STEP**

Our aim is to have OHLC arrays of the currency pairs we choose. The first thing we can do is to select which time frame we want to import. Let us suppose that there are only two frames, the 30-minute and the hourly bars. We can therefore create variables that hold the statement to tell the MetaTrader5 library which frame we want.

```
# Choosing the 30-minute time frame
frame_M30  = mt5.TIMEFRAME_M30
# Choosing the hourly time frame
frame_H1   = mt5.TIMEFRAME_H1
```

Then, by staying in the spirit of importing variables, we can define the variable

[3] https://www.mql5.com/en/docs/integration/python_metatrader5

that states what date is it now. This helps the algorithm know the stopping date of the import. We can do this by the simple line of code below.

```
# Defining the variable now to give out the current date
now = datetime.datetime.now()
```

Note that these code snippets are better used chronologically, hence, I encourage you to copy them in order and then execute them one by one so that you understand the evolution of what you are doing. The below is a function that holds whichever assets we want. Generally, I use 10 or more but for simplicity, let us consider that there are only two currency pairs: EURUSD and USDCHF.

```
def asset_list(asset_set):
  if asset_set == 1:
    assets = ['EURUSD', 'USDCHF']
  return assets
```

The below establishes a connection to MetaTrader5, applies the current date, and extracts the needed data. Notice the arguments year, month, and day. These will be filled by us to select from when we want the data to start. Note, I have inputted Europe/Paris as my time zone, you should use your time zone to get more accurate data.

```
def get_quotes(time_frame, year = 2005, month = 1, day = 1, asset = "EURUSD"):
   # Establish connection to MetaTrader 5
   if not mt5.initialize():
      print("initialize() failed, error code =", mt5.last_error())
      quit()
   timezone = pytz.timezone("Europe/Paris")
   utc_from = datetime.datetime(year, month, day, tzinfo = timezone)
   utc_to = datetime.datetime(now.year, now.month, now.day + 1, tzinfo = timezone)
   rates = mt5.copy_rates_range(asset, time_frame, utc_from, utc_to)
   rates_frame = pd.DataFrame(rates)
   return rates_frame
```

And finally, the last function we will use is the one that uses the get_quotes function and then cleans the results so that we have a nice array. We have selected data since January 2019 as shown.

```
def mass_import(asset, horizon):
 if horizon == 'M30':
   data = get_quotes(frame_M30, 2019, 1, 1, asset = assets[asset])
   data = data.iloc[:, 1:5].values
   data = data.round(decimals = 5)
 return data
```

Finally, we are done building the blocks necessary to import the data. To import EURUSD OHLC historical data, we simply use the below code line:

```
# Choosing the horizon
horizon = 'M30'
# Creating an array called EURUSD having M30 data since 2019
EURUSD = mass_import(0, horizon)
```

Now, we have a variable called EURUSD with historical OHLC data inside. The

first phase is done and now, we will proceed to data analysis. The utility of coding is bigger than ever nowadays. A trader or an analyst must know at least the basics of how algorithms work and must also know some functions and the logic behind the calculations. Many analysts use indicators but do not know how they are calculated nor how they can be coded. As this book presents many indicators, we need to first understand the basics, and prepare the necessary functions to analyze the data.

**BASIC SYNTAX AND REQUIRED FUNCTIONS**

Python is rapidly growing to be the go-to language when it comes to creating and testing strategies. It is a fast, versatile, and easy language to learn that will allow you to perform many operations. Basically, for the most part of the book, we will be using 3 known libraries to handle, manipulate, and plot our data. Let us start with the two similar libraries, numpy[4] and pandas[5] (which will be used to a much lesser extent than numpy). Every python developer must imperatively know a thing or two about them. We use numpy to manipulate arrays and perform operations and other useful mathematical functions, it is the go-to package when you are dealing with time series or any other dataset as it is fast due to vectorization (Not quite used in this book). As for pandas, it is also similar to numpy with some differences and is mainly for reading and slicing data frames and excel files among other functions such as calculating exponential moving averages. Of course, these are simplistic definitions, and the libraries can do much more than the described tasks.

[4] https://docs.scipy.org/doc/numpy/user/whatisnumpy.html
[5] https://pandas.pydata.org/pandas-docs/stable/

In python, we import a library using the import statement followed by the name of the library and an optional shortcut or nickname for that library so you can call it later.

Remember, a library can be considered as a box of functions where you can just load them and use whichever ones you want. If the field of mathematics was a library, then addition and multiplication would be functions that are inside of it.

The word np is the standard numpy shortcut while pd is the standard pandas shortcut. Also, plt is the standard matplotlib shortcut.

```
# Importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

After importing the libraries, we now have the necessary tools to start analyzing the time series. Generally, with financial time series, we will have what we call OHLC data. This is short for Open, High, Low, and Close. The data should be in the form of 4 columns. This will be our basic data structure. Our job in the full research process is to add columns where we place technical indicators and buy/sell orders as well as trading results.

Let us start by adding a column. Remember, we have four columns containing OHLC data and we want to add a fifth column containing zero values so that we populate it with values later. To add a new empty column, we can use the following function.

```
# The function to add a certain number of columns
def adder(Data, times):
    for i in range(1, times + 1):
        z = np.zeros((len(Data), 1), dtype = float)
        Data = np.append(Data, z, axis = 1)
    return Data
# Using the function to add 5 columns
my_data = adder(my_data, 5)
```

Sometimes, we need to remove columns when we have more than enough or when we want to clean out the data.

```
# The function to delete a certain number of columns
def deleter(Data, index, times):
    for i in range(1, times + 1):
        Data = np.delete(Data, index, axis = 1)
    return Data
# Using the function to delete 2 columns starting from column 5
my_data = deleter(my_data, 5, 2)
```

It is worth mentioning that Python starts indexing at zero, therefore column 5 is chronologically the sixth column and not the fifth column. What about rows? Sometimes, we will calculate some indicators that start from the 100$^{th}$ line, therefore, the first 100 lines will have no use. The next page shows a function to delete several rows from an array that is descending in order. This means that the first values are the earliest ones with the latest values below being the latest ones.

```
# The function to delete a certain number of rows
def jump(Data, jump):
   Data = Data[jump:, ]
   return Data
# Using the function to delete the first 10 rows of an array
my_data = jump(my_data, 10)
```

## PLOTTING DATA AND VISUALIZATION

How about we try visualizing the data now? This is very easy in Python and does not require any sophisticated code. Let us consider as usual that we have an OHLC array, and we want to visualize the latest 500 closing prices in black with a legend that describes what we are seeing. We also want to see a grid on the chart to facilitate the visual interpretation.

```
# Plotting
plt.plot(my_data[-500:, 3], color = 'black', label = 'Closing Price')
# Creating the grid
plt.grid()
# Calling the legend Function
plt.legend()
```

The above function says that we want to plot the latest 500 values (hence, the negative sign) of the fourth column (hence, the index 3 as Python starts indexing from zero). We also want the color to be black which is why the argument of color equals what we want it to be. The label is a string (text) of what we want to describe. This is what will appear on the chart in the form of a text box called later by the legend function. Finally, the grid function is self-explanatory. It is optional but it adds a visual effect to the chart.

*Figure 1 EURUSD line chart.*

## CODING THE SIGNALS

We want to create trading strategies out of technical indicators. This means we have to do two things after importing the OHLC data:

- Create the technical indicators in the columns next to the OHLC data.
- Create the conditions of buy and sell orders and populate the required columns.

Let us code the most basic of technical indicators, the simple moving average (Which we will cover in greater depth later). We will use the adder function to add three new columns, the column where the moving average is located and then the buy column (bullish signal) followed by the sell column (bearish signal).

```
def ma(Data, lookback, what, where):
    for i in range(len(Data)):
        try:
            Data[i, where] = (Data[i - lookback + 1:i + 1, what].mean())
        except IndexError:
            pass
    return Data

# Calling a 20-period Moving Average function based on closing prices (index = 3) and to be populated on the fifth column (index = 4)

my_data = ma(my_data, 20, 3, 4)

# The signal function that populates the buy/sell columns following the crossovers

def signal(Data, closing, moving_average, buy, sell):
    for i in range(len(Data)):
        if Data[i, closing] > Data[i, moving_average and Data[i - 1, closing] < Data[i - 1, moving_average]:
            Data[i, buy] = 1
        if Data[i, closing] < Data[i, moving_average and Data[i - 1, closing] > Data[i - 1, moving_average]:
            Data[i, sell] = -1

# Using the signal function

signal(my_data, 4, 5, 6)
```

The above signal function is what will be used to populate the sixth (index = 5) and seventh (index = 6) columns. The proxy for a buy signal or order will be 1 while the proxy for a sell (short) signal or order will be -1. Note that the strategies are all long/short, meaning that we can either buy and sell higher or sell short now and buy back lower hence, having two possible ways of benefitting from the asset's direction.

At the moment, we should have a data array with the following elements:

- OHLC data populating the first four columns.
- A 20-period moving average populating the fifth column.
- Buy orders with the value of 1 populating the column 6.
- Sell orders with the value of -1 populating column 7.

This is the basic strategy of when the market price surpasses its moving average, a buy signal is generated because it is implied that a new bullish trend has started. Similarly, when the market price breaks its moving average, a sell signal is generated because it is implied that a new bearish trend has started. A quick and easy way to plot the signals is to place upward pointing arrows whenever we have a buy signal and to place downward pointing arrows whenever we have a sell signal. We have to define the function and then apply it on the signals generated in the columns.

```
def ohlc_plot_candles(Data, window):
    Chosen = Data[-window:, ]
    for i in range(len(Chosen)):
        plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'black', linewidth = 1)
        if Chosen[i, 3] > Chosen[i, 0]:
            plt.vlines(x = i, ymin = Chosen[i, 0], ymax = Chosen[i, 3], color = 'green', linewidth = 3)
        if Chosen[i, 3] < Chosen[i, 0]:
            plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0], color = 'red', linewidth = 3)
        if Chosen[i, 3] == Chosen[i, 0]:
            plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0] + 0.00001, color = 'black', linewidth = 6)
    plt.grid()
def signal_chart(Data, close, what_bull, what_bear, window = 500):
    Plottable = Data[-window:, ]
    fig, ax = plt.subplots(figsize = (10, 5))
    ohlc_plot_candles(Data, window)
    for i in range(len(Plottable)):
        if Plottable[i, what_bull] == 1:
            x = i
            y = Plottable[i, close]
            ax.annotate(' ', xy = (x, y),
                        arrowprops = dict(width = 9, headlength = 11, headwidth = 11, facecolor = 'green', color =
'green'))
        elif Plottable[i, what_bear] == -1:
            x = i
            y = Plottable[i, close]
            ax.annotate(' ', xy = (x, y),
                        arrowprops = dict(width = 9, headlength = -11, headwidth = -11, facecolor = 'red', color =
'red'))
```

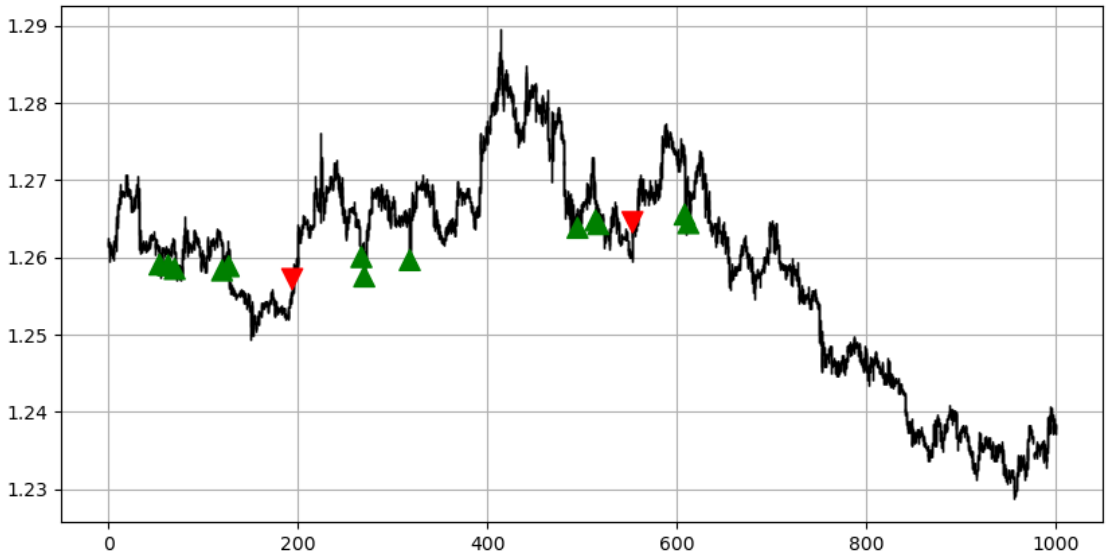



*Figure 2 - Signal chart example.*

The above plot shows signals generated using a hypothetical function. Unfortunately, to make the book accessible to more people by lowering the price, I had to write it in black and white. However, the colors will be visible when you run them in Python. Let us now start discussing trend-following indicators with moving averages before moving on to more complex indicators and strategies.

# PART 1 MOVING AVERAGES

Moving averages are probably the most known technical overlay indicators. This is due to their quantitative intuition, unbelievable simplicity, and agreed-upon reliability. They are also versatile as we can use them in trend-following or in contrarian strategies. Therefore, it is essential to acquire a strong working knowledge on moving averages. Before we move on, we need to start from scratch. What are moving averages? And where do they come from? A moving average is a mean that is calculated at each time step with a defined lookback period also known as a window. Here is a detailed example:

| Time Step | Value | 3-period Moving Average |
|---|---|---|
| 1 | 10 | |
| 2 | 12 | |
| 3 | 10 | 10.67 |
| 4 | 8 | 10.00 |
| 5 | 10 | 9.33 |
| 6 | 14 | 10.67 |
| 7 | 16 | 13.33 |
| 8 | 14 | 14.67 |
| 9 | 18 | 16.00 |
| 10 | 18 | 16.67 |

The first column is the time step, the second column contains chronologically ordered values, and the third column contains the moving average at each time step where we calculate the mean of the last 3 values. The first value 10.67 is found by averaging 10, 12, and 10. As mentioned above, the window moves with the values, meaning that the mean at time step 10 refers to the average of 14, 18, and 18. The next chapters present different types of moving averages in detail as well as their full code.

# CHAPTER 1 THE SIMPLE MOVING AVERAGE

This is the normal plain vanilla mean calculation that we have seen in the introductory part of moving averages. We tend to use it everywhere and it forms one of the pillars of descriptive statistics. It follows the basic formula below:

$$\mu = \frac{X_1 + X_2 + \cdots + X_n}{n}$$

The formula above states that the current average of a set of values equals their sum divided by their quantity. The mean is the best expectation or approximation we have to describe the data. This is to say that the expected next value can be the average of the previous values if we expect the same characteristics to persist. Now, we can apply the simple moving average onto market data. Here is how we will do it:

- We will have a prepared OHLC data in the form of an array where we will target the last column that deals with the closing prices.
- We will calculate the moving average of that column using a specified lookback period. The output (Moving average values) will be printed in an empty column next to the closing prices' column. Therefore, giving us five columns (Open, high, low, close, moving average).

The main advantages of simple moving averages are that they are widely used by traders. This has the advantage of improving its results since a lot of market participants will act on its signals thus pushing the prices towards the expected value in a self-fulfilling prophecy event. Another advantage is that they are simple to use. The strategies formed around moving averages are extremely intuitive and easy to use. The main disadvantages of simple moving averages are that they are lagging and slow to react to recent movements

especially when the lookback period is high. They are also likely to provide many false signals with crossover strategies, a type of strategies we will cover in this book. The way to code a simple moving average can be through an algorithm that loops around the data and at each time step (Row), calculates the mean using a specified lookback period (Window).

```
def ma(Data, lookback, close, where):
  Data = adder(Data, 1)
  for i in range(len(Data)):
   try:
    Data[i, where] = (Data[i - lookback + 1:i + 1, close].mean())
       except IndexError:
          pass
  return Data
```

A simple and quick way to go around the index error is to use the try and except technique as shown in the code above. Remember to define the adder function so that it works properly by adding one extra column populated with the moving average values next to the OHLC data. The next chart shows the hourly candlestick chart with the 200-hour moving average alongside it.

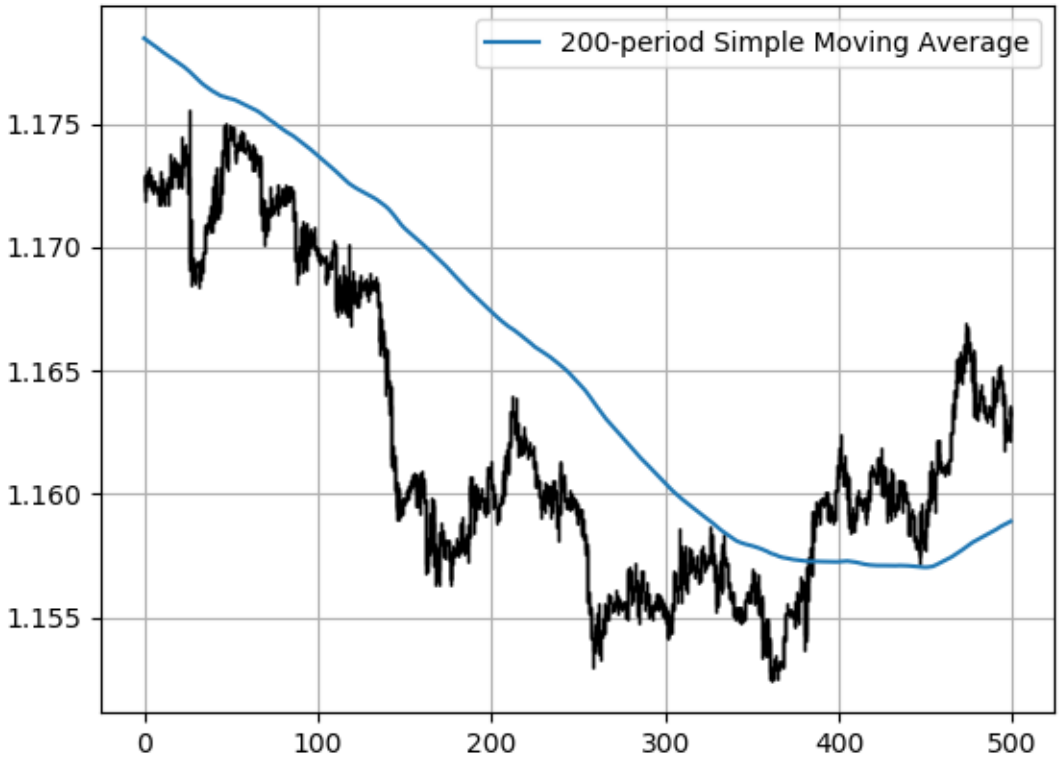



*Figure 3 - Hourly EURUSD values with the 200-hour simple moving average.*

Moving averages are mainly used in multiple ways to follow a trend:

- **Simple Crossovers**: Whenever the market surpasses its moving average, it can be a sign that the overall trend is becoming bullish if we do not account for the lag effect. Similarly, whenever the market breaks its moving average, it can be a sign that the overall trend is becoming bearish if we do not account for the lag effect
- **Double Crossovers**: Whenever a short-term moving average surpasses a long-term moving average, it can be a sign that the market is starting to drift upwards. This can also be a bullish trend confirmation and therefore, some traders look for long opportunities. In parallel, whenever a short-term moving average breaks a long-term moving

average, it can be a sign that the market is starting to drift downwards. This can also be a bearish trend confirmation and therefore, some traders look for short opportunities.

- **Triple Crossovers**: A less common form of crossovers which relies on a short-term moving average, a medium-term moving average, and a long-term moving average. A bullish signal is generated whenever the short-term moving average surpasses the medium-term moving average which itself is above the long-term moving average. In contrast, A bearish signal is generated whenever the short-term moving average breaks the medium-term moving average which itself is below the long-term moving average.
- **Optimal Entry Levels**: Whenever there is an established bullish trend with a moving average following the market, the trader can choose to buy the dips and corrections close to the area of the moving average in anticipation of a continuation higher. Similarly, whenever there is an established bearish trend with a moving average following the market, the trader can choose to sell the strengths and corrections close to the area of the moving average in anticipation of a continuation lower. This is also known as a dynamic support and resistance strategy.
- **Slope Technique**: Whenever the slope of the moving average changes, we can consider that it can be an early signal of a change in the trend.

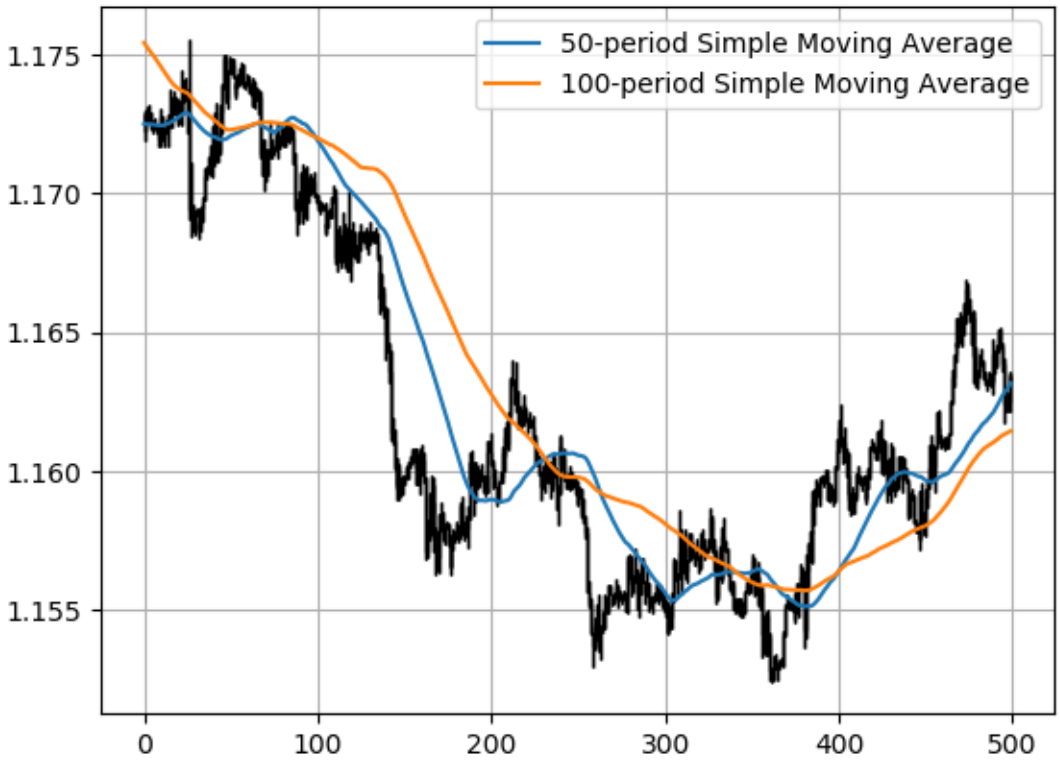



*Figure 4 - Hourly EURUSD values with 50-hour and 100-hour simple moving averages.*

The above chart shows an example of two moving averages, a short-term and a long-term one where their crossovers signal a change in trend and possibly trading triggers. It is very important to include moving averages in the trading framework as they are heavily followed and thus do have an impact on the market price. Even though there is research that suggests their predictability is lower than when they were first introduced, the trading framework is not complete without them. My recommendation is to use them as dynamic support and resistance levels after having optimized the lookback period. I also recommend using a zone of moving average rather than one moving average applied to the closing price. This means that we can apply the same moving

average on the highs and lows so that we get a moving zone that filters the noise caused by randomness and volatility.

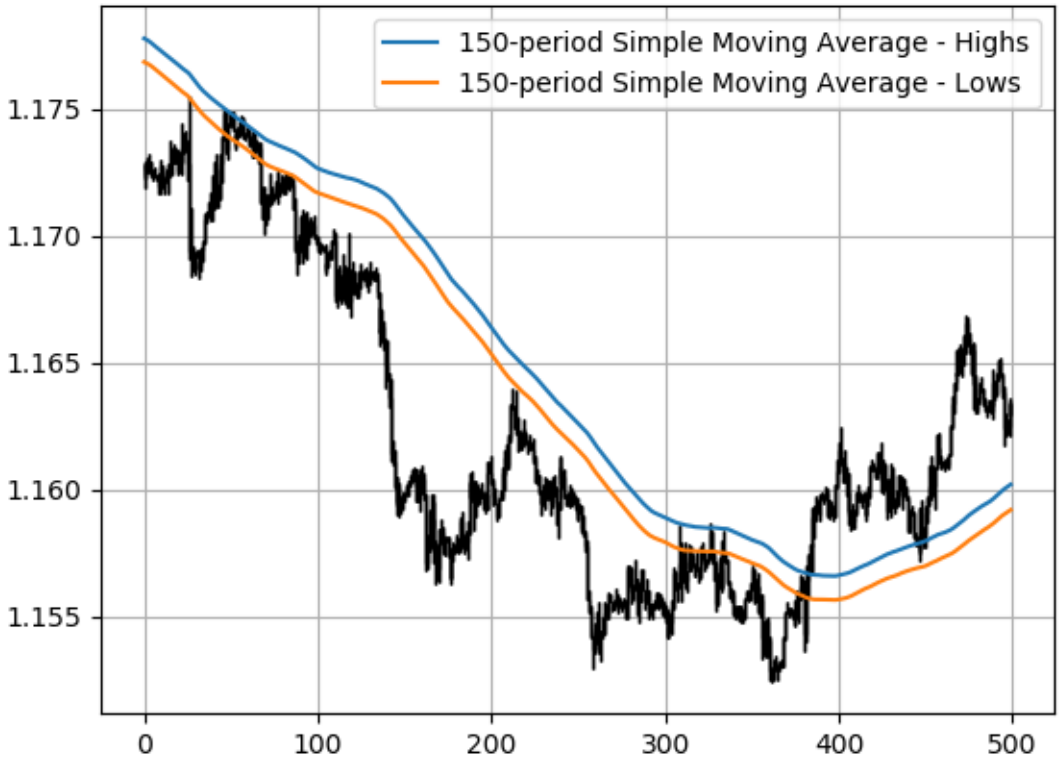



*Figure 5 - Hourly EURUSD values with a 150-hour moving average zone.*

The above chart shows an example of a 150-hour simple moving average applied on both price extremes, highs, and lows. We can see how well it acts as a dynamic support and resistance zone. The way to trade this set-up is to have a contrarian view each time we approach the zone. Whenever the market clearly surpasses the zone, we can say that the trend has changed, and we might see a new trend emerging. Therefore, we will use it as a simple crossover strategy. Notice how in the above chart, the market having surpassed the zone, shaped a pull-back around it before continuing higher. It has become a dynamic support after being a dynamic resistance.

# CHAPTER 2 THE EXPONENTIAL MOVING AVERAGE

Lag is an important limitation of moving averages and even though they might be useful sometimes in finding dynamic support and resistance levels, we need to confirm the change of the trend as fast as we can, and this is done by minimizing the lag factor. Surely, we can just reduce the lookback period on the simple moving average, but the question remains: Is equal weighting the correct approach? We are not sure, but we can try to minimize the lag by giving more weight to the recent values. This is done by adding an exponential factor following this formula below:

$$EMA_{\text{Today}} = \left(\text{Value}_{\text{Today}} * \left(\frac{\text{Smoothing}}{1+\text{Days}}\right)\right) + EMA_{\text{Yesterday}} * \left(1 - \left(\frac{\text{Smoothing}}{1+\text{Days}}\right)\right)$$

The main advantages of exponential moving averages are that even though they are less used than simple moving averages, they have less lag than them. Another advantage is that they are simple to use. The smoothing factor is often 2. Note, that if we increase the smoothing factor (also known as alpha) then, the more recent observations will have more weight. The next function calculates the exponential moving average using the moving average function since the first average is calculated in the simple way and hence, we need it. In Python language, we can define a function that calculates the EMA as follows.

```
def ema(Data, alpha, lookback, what, where):
    alpha = alpha / (lookback + 1.0)
    beta  = 1 - alpha
    # First value is a simple SMA
    Data = ma(Data, lookback, what, where)
    # Calculating first EMA
    Data[lookback + 1, where] = (Data[lookback + 1, what] * alpha) + (Data[lookback, where] * beta)
    # Calculating the rest of EMA
    for i in range(lookback + 2, len(Data)):
        try:
            Data[i, where] = (Data[i, what] * alpha) + (Data[i - 1, where] * beta)
        except IndexError:
            pass
    return Data
```

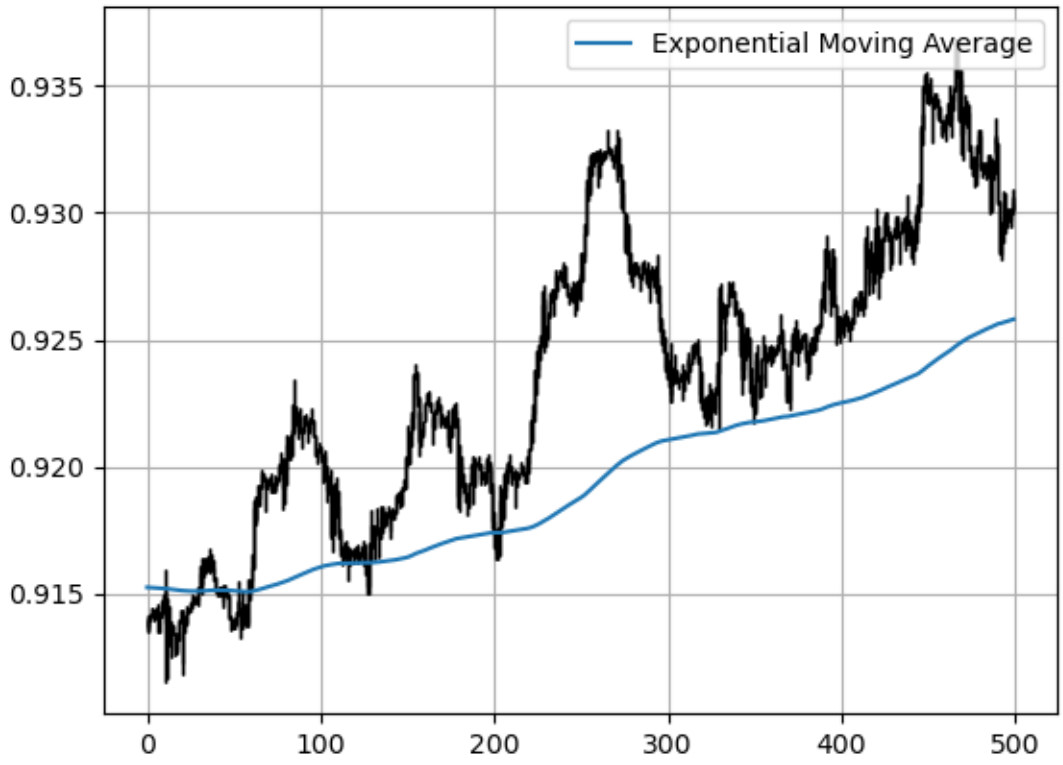



*Figure 6 - USDCHF hourly values with the 400-period exponential moving average.*

The next chart shows the difference between a 400-period simple moving average and a 400-period exponential moving average. We can notice the difference as the exponential version is adjusted to the recent price action since more weight is placed on it.

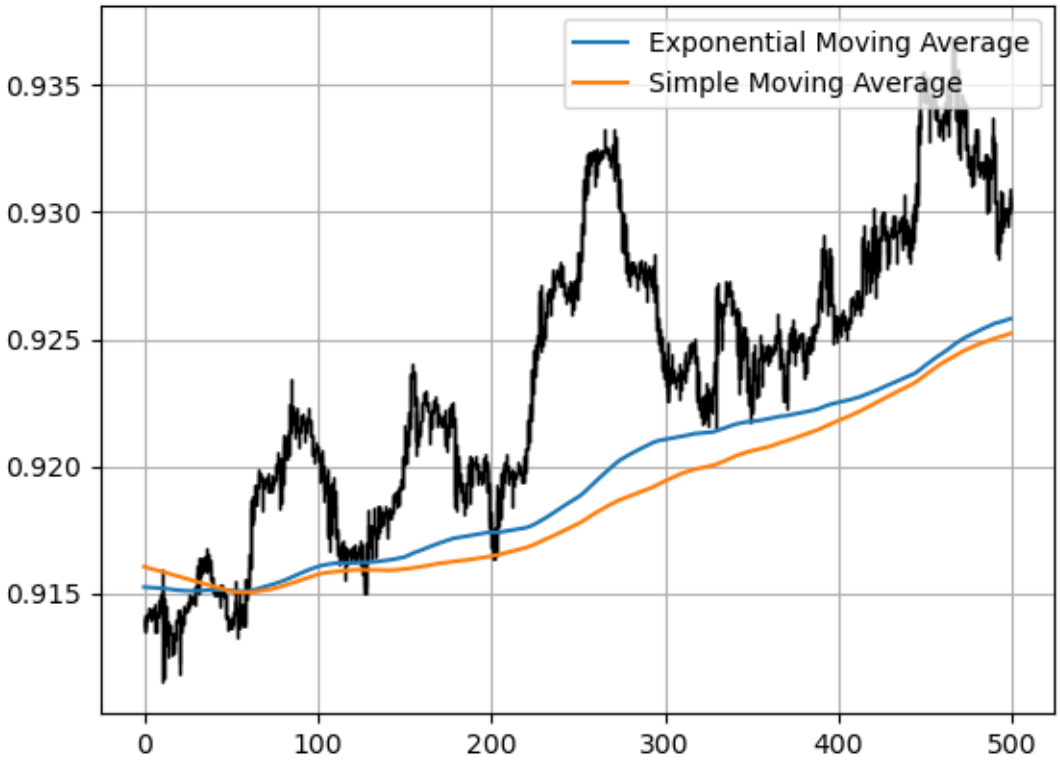



*Figure 7 - USDCHF hourly values with the 400-period exponential and simple moving averages.*

Exponential moving averages are the second most widely used moving averages after the simple type. They can be a form of diversification or confirmation to add to a trading strategy.

# CHAPTER 3 THE SMOOTHED MOVING AVERAGE

Welles Wilder, the inventor of many iconic technical indicators uses this type of moving average, which is as the name suggests, a smoothed version that reduces noise. We can calculate it easily through a simple transformation of the lookback period of the exponential moving average.

$$Smoothed\ Moving\ Average = (2\ .Exponential\ Moving\ Average) - 1$$

What the above formula tells us is that a 20-period smoothed moving average is a 39-period exponential moving average. Similarly, a 70-period smoothed moving average is 139-period exponential moving average. Keeping in mind the function of the exponential moving average, we can simply recycle it and add a lookback variable that automatically transforms the exponential into smoothed.

```
# Choosing the lookback period
lookback = 10
smoothed_lookback = (2 * lookback) – 1
# Calculating a 10-period smoothed moving average
my_data = ema(my_data, 2, smoothed_lookback, 3, 4)
```

Choosing between exponential and smoothed is a matter of preference and depends on the trader's experience. Personally, I use them equally and in different situations. I prefer to use the exponential moving average as an ingredient in other technical indicators but when it comes to finding support and resistance lines, I tend to have a small preference for the smoothed type. The next graph shows the difference between a 100-period exponential moving average and a 100-period smoothed moving average.

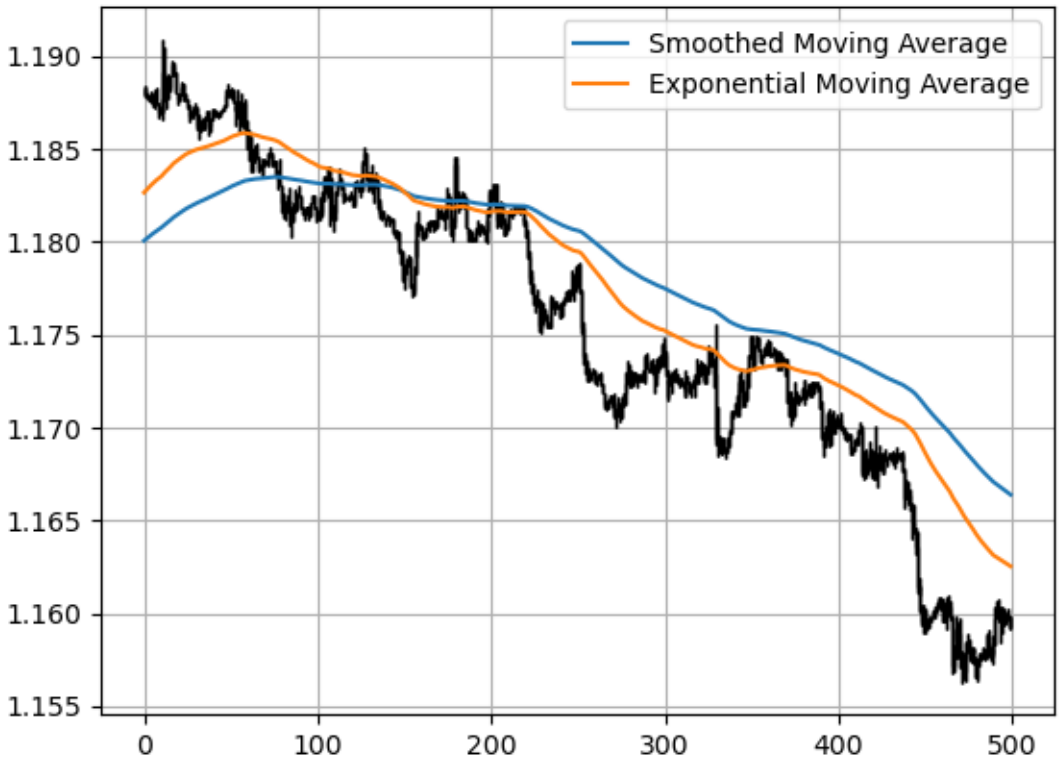



*Figure 8 - EURUSD hourly values versus smoothed and exponential moving averages.*

Remember that this is basically the same as a 100-period and a 199-period exponential moving averages which is why we see the line in orange sticking closer to the market while the line in blue is slightly more distant.

The next graph shows the hourly values of the USDCHF pair with the 150-period smoothed moving average (The equivalent of a 299-period exponential moving average).

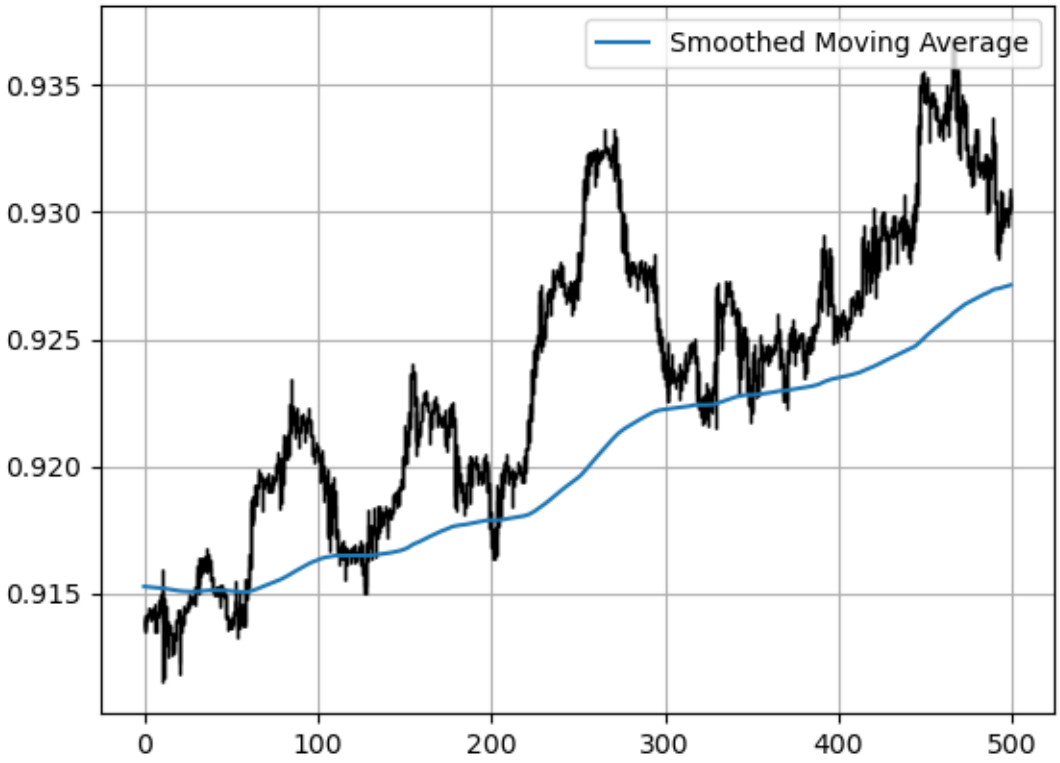



*Figure 9 - USDCHF hourly values with its 150-period smoothed moving average.*

Some of the best ways to use smoothed moving averages are the following:

- **Dynamic support and resistance levels**: Whenever the market approaches the smoothed moving average from the below, we can see it as a resistance level and whenever the market approaches the smoothed moving average from the above, we can see it as support. This technique requires optimization and constant change in the lookback period as the market's properties change.
- **Components of technical indicators**: As mentioned before, the smoothed moving average can be found as a component of the RSI. We can also find it in the average true range indicator as well as the average directional index, two indicators covered in this book.

# CHAPTER 4 THE LINEAR-WEIGHTED MOVING AVERAGE

Another extension to the idea of exponential smoothing is the linear-weighted moving average where, as the name suggests, different weights are given to each row which corresponds to a linearly adjusted factor. The idea is that by time, the weight of the values must diminish to account for the recent ones.

$$LWMA = \frac{(P_n \,.\, W_1) + (P_{n-1} \,.\, W_2) + \cdots + (P_{n-x} \,.\, W_n)}{\sum W}$$

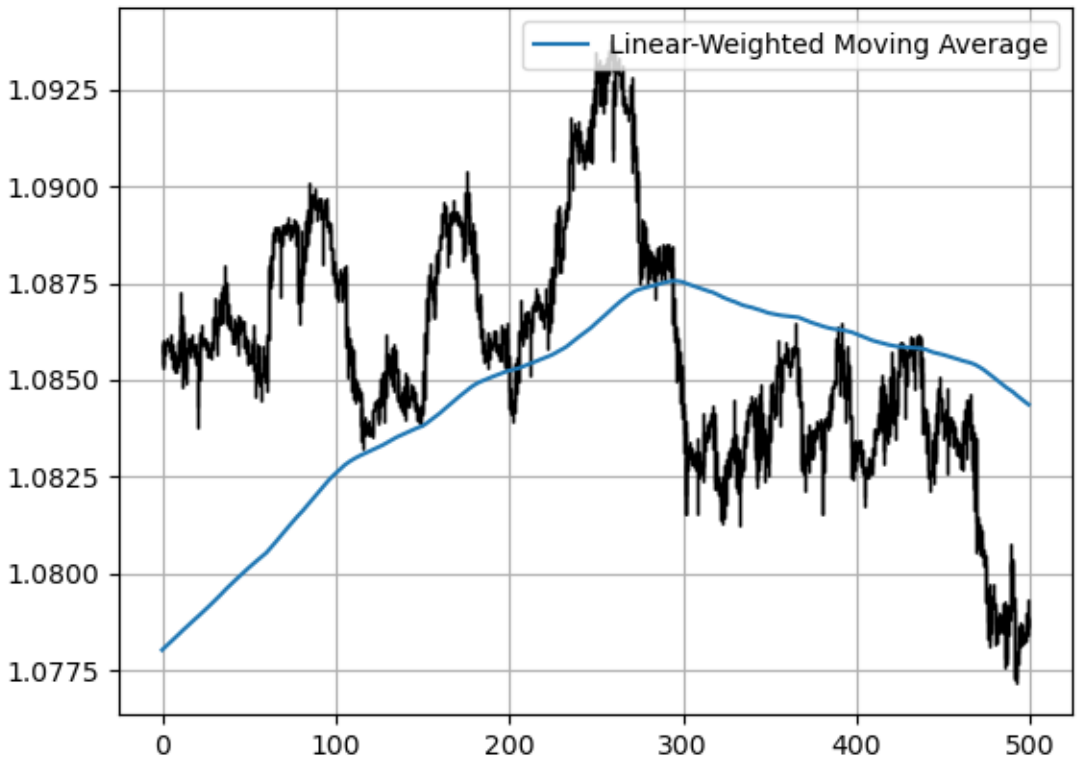



*Figure 10 - Hourly values of the EURCHF pair with the 100-hour linear-weighted moving average.*

Let us discuss the intuition and the advantage of the weighted moving average through an example.

Consider the following example on the AUDCAD pair. The below are the closing prices of hypothetical and chronologically ordered days:

- Day 1: 0.9060
- Day 2: 0.9045
- Day 3: 0.9050
- Day 4: 0.9000
- Day 5: 0.9075

Normally, a simple moving average on the fifth day would be calculated this way:

$$Simple\ Moving\ Average_5 = \frac{0.9060 + 0.9045 + 0.9050 + 0.9000 + 0.9075}{5} = 0.9046$$

If we want to use the linear-weighted method, we would follow this framework:

- Day 1: 0.9060 x 1 = 0.9060
- Day 2: 0.9045 x 2 = 1.8090
- Day 3: 0.9050 x 3 = 2.7150
- Day 4: 0.9000 x 4 = 3.6000
- Day 5: 0.9075 x 5 = 4.5375

$$Weighted\ Moving\ Average_5 = \frac{0.9060 + 1.8090 + 2.7150 + 3.6000 + 4.5375}{15} = 0.9045$$

We have divided by 15 rather than 5 because with this type of moving average we must use the number of coefficients which we also call weights. Therefore, by multiplying the most recent value by the number of values and by multiplying the previous value by the number of values minus one and so on, we will find 15 coefficients. Since this is a short-term analysis, the two moving averages are pretty close.

What if we take a look at a chart showing the two types of moving averages using the same lookback period and try to come up with a conclusion?

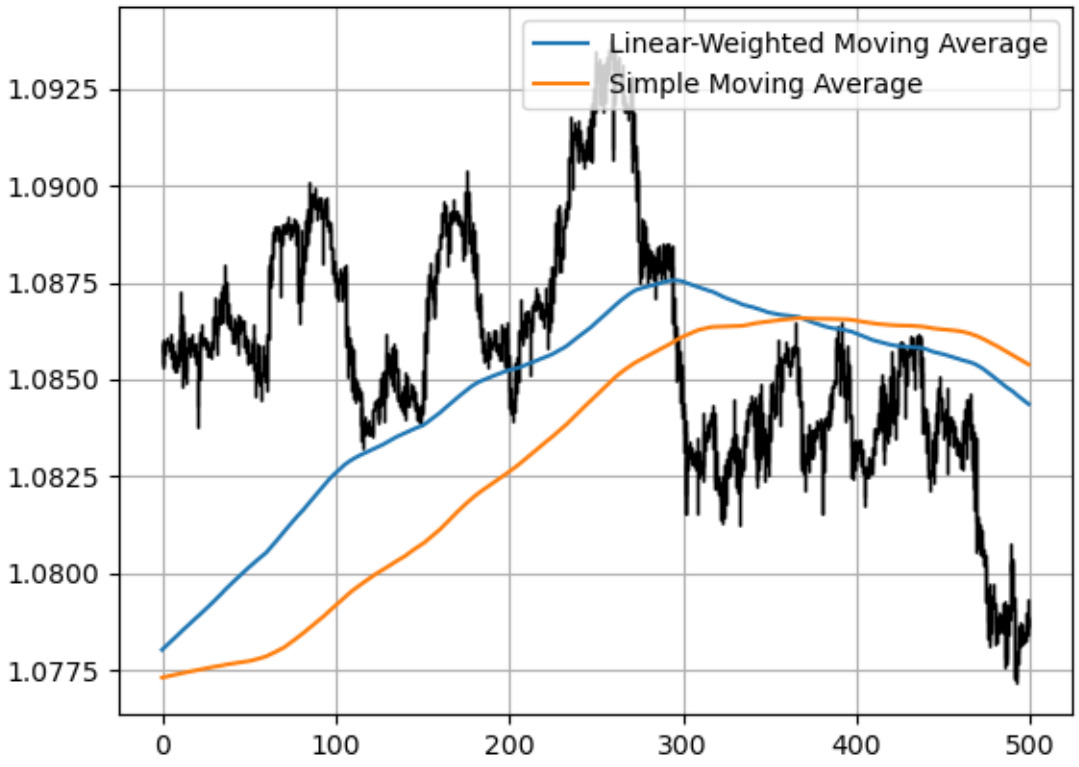



*Figure 11 – Hourly EURCHF values with the 100-hour simple and linear-weighted moving averages.*

It can be seen that the weighted moving average follows more closely the market's price since it places more weight on the more recent values. To code the weighted moving average, we can follow the syntax as shown next. All we need is an array of OHLC data and the function will do the rest.

```
def lwma(Data, lookback, what):
    weighted = []
    for i in range(len(Data)):
        try:
            total = np.arange(1, lookback + 1, 1)
            matrix = Data[i - lookback + 1: i + 1, what:what + 1]
            matrix = np.ndarray.flatten(matrix)
            matrix = total * matrix
            wma = (matrix.sum()) / (total.sum())
            weighted = np.append(weighted, wma)
        except ValueError:
            pass
    Data = Data[lookback - 1:, ]
    weighted = np.reshape(weighted, (-1, 1))
    Data = np.concatenate((Data, weighted), axis = 1)
    return Data
```

What is the best way to use the weighted moving average? We can basically use it for all the reasons listed in the first chapter covering moving averages. It is important to understand that each market has different characteristics, and our back-tests will give us different results from time to time. For example, consider a simple crossover strategy where we buy whenever the market crosses over the 50-period weighted moving average and we sell short whenever the market crosses below the 50-period weighted moving average. The results could indicate that this strategy works well on a certain market, but it may also indicate that it works horribly on another market even if the two markets are related. This is to say that the low signal-to-noise ratio combined with complex market dynamics can give us infinite possibilities but also opportunities.

# CHAPTER 5 THE HULL MOVING AVERAGE

Developed by Alan Hull, it is a very smooth and fast-moving average with the goal of rapidly determining the trend through lag reduction often wrongly described as lag elimination. I say wrongly because when you eliminate lag, you get a line that moves exactly with the market price which is pretty useless. The Hull moving average uses the weighted moving average as a building block and it is calculated through the below steps:

- Choose a lookback period such as 20 or 100 and calculate the weighted moving average of the closing price following the intuition seen in the previous chapter.
- Halve the lookback period found in the first step and calculate the weighted moving average of the closing price using this new lookback period. If the number cannot by divided by two, then take the closest number before the comma (e.g. a lookback of 15 can be 7 or 8 as the second lookback).
- Multiply the second weighted moving average by two and subtract from it the first weighted moving average.
- As a final step, take the square root of the first lookback period (e.g. if you have chosen a lookback of 100, then the third lookback period is 10) and calculate the weighted moving average on the latest result we had in the third step. Be careful not to calculate it on market prices.

Therefore, if we choose a lookback period of 100, we will calculate on 100 lookback period, then on 50, and finally, on 10 applied to the latest result. The method sure does look complex and it is but the result is actually pretty smooth and reduces significantly the lag inflicted by the averaging factor. It is worth mentioning that the Hull moving average is purely used to detect changes in

the trend and it is not to be used as a dynamic support and resistance level. Remember that the technique of dynamic support and resistance levels is all about awaiting the approach of the market's price to its moving average before initiating a contrarian position in anticipation that we have found a good reactionary level.

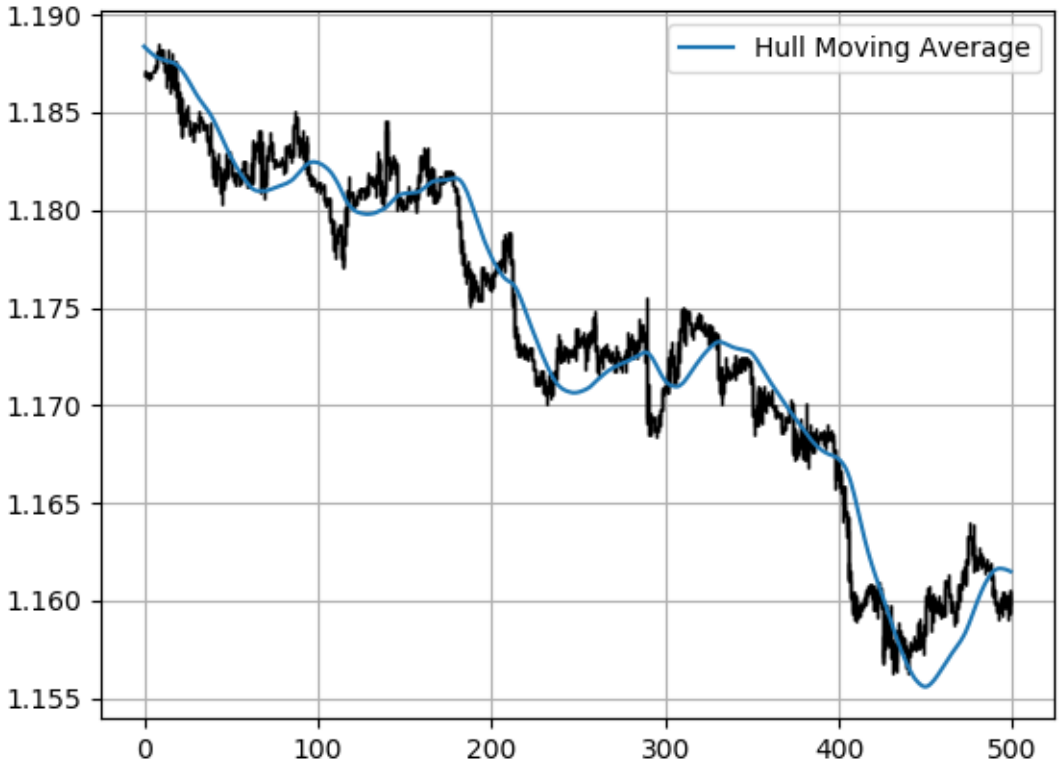



*Figure 12 - Hourly EURUSD values with the 100-hour Hull moving average.*

The figure above shows an example of a 100-period Hull moving average applied on the hourly values of the EURUSD. We can notice how close it is to the market price. The code for the moving average can be written down as follows.

```
def hull_moving_average(Data, what, lookback, where):
    # Calculating the weighted moving average
    Data = lwma(Data, lookback, what)
    # Halving the first lookback period
    second_lookback = round((lookback / 2), 1)
    second_lookback = int(second_lookback)
    # Calculating the second weighted moving average
    Data = lwma(Data, second_lookback, what)
    Data = adder(Data, 1)
    Data[:, where + 2] = ((2 * Data[:, where + 1]) - Data[:, where])
    # Calculating the third lookback period
    third_lookback = round(np.sqrt(lookback), 1)
    third_lookback = int(third_lookback)
    # Calculating the Hull moving average
    Data = lwma(Data, third_lookback, where + 2)
    Data = deleter(Data, where, 3)
    return Data
```

The code is very intuitive and simple unlike the reasoning of the Hull moving average. Let us check whether it is really more reactive or not by comparing it to the weighted moving average. The next chart shows a 100-period Hull and weighted moving averages. We can easily notice how much closer and more reactive the Hull moving average is.

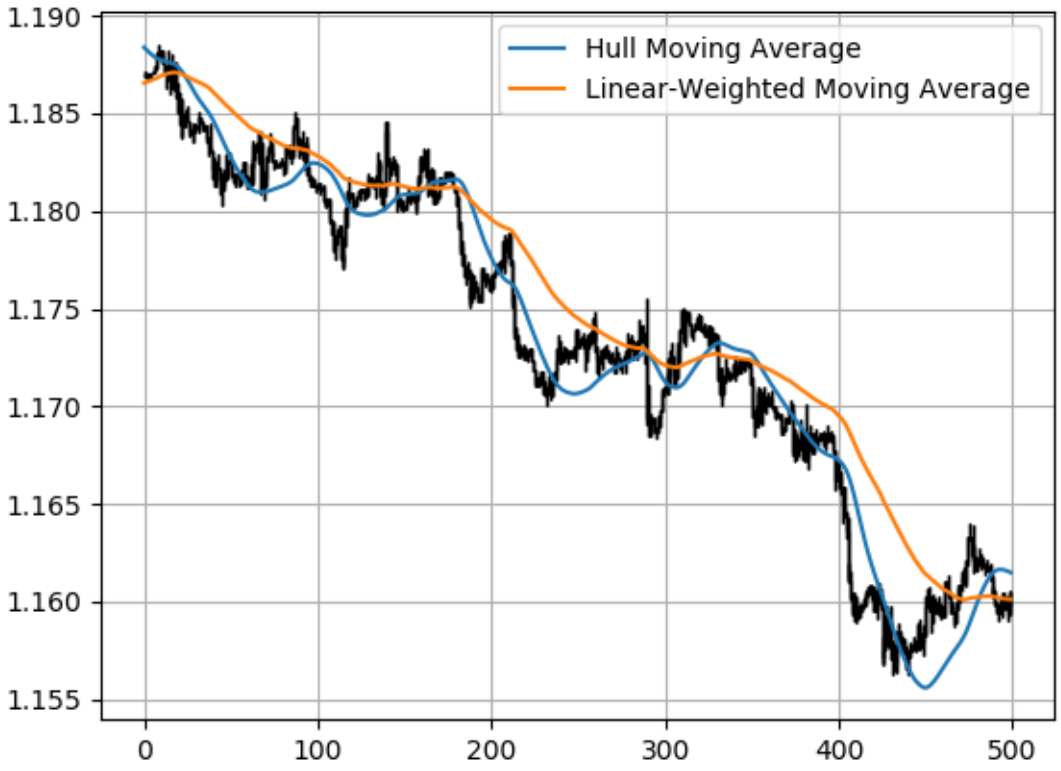



*Figure 13 – Hourly EURUSD values with the 100-hour linear-weighted and Hull moving average.*

Compared to the linear-weighted moving average, the Hull moving average does a very good job at detecting early the turnarounds. Why do we say that the Hull moving average reduces lag? The answer comes mainly from the building blocks which are the weighted moving averages. They place more weights on more recent values.

Furthermore, the lag is also reduced by offsetting one weighted moving average with the one that has half the lookback period. Finally, we take the square root of the lookback period and apply it on the moving average itself using the weighting method. This gives us a moving average close to the market price and serves as an early trend reversal indicator all else equal.

# CHAPTER 6 THE VOLATILITY MOVING AVERAGE

Volatility-weighting gives us the advantage of incorporating a valuable element of the financial markets. The characteristics of price action change with regards to many variables, one of them is volatility. Most moving averages fail to adjust to volatility and thus they miss a part of the picture. Surely, they are useful on their own, but every moving average has its own flaws. We must first understand the concept of volatility. It is a key concept in finance, whoever masters it holds a tremendous edge in the markets. Unfortunately, we cannot always measure and predict it with accuracy. Even though the concept is more important in options trading, we need it pretty much everywhere else. Traders cannot trade without volatility nor manage their positions and risk. Quantitative analysts and risk managers require volatility to be able to do their work. Before we discuss the different types of volatility, why not look at a graph that sums up the concept? Check out the below image to get you started.

*Figure 14 – A high volatility time series in red versus a low volatility time series in green.*

You can code the previous chart in Python yourself using the following snippet.

```
# Importing the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
# Creating high volatility noise
hv_noise = np.random.normal(0, 1, 250)
# Creating low volatility noise
lv_noise = np.random.normal(0, 0.1, 250)
# Plotting
plt.plot(hv_noise, color = 'red', linewidth = 1.5, label = 'High Volatility')
plt.plot(lv_noise, color = 'green', linewidth = 1.5, label = 'Low Volatility')
plt.axhline(y = 0, color = 'black', linewidth = 1)
plt.grid()
plt.legend()
```

The different types of volatility around us can be summed as follows:

- **Historical volatility**: It is the realized volatility over a certain period. Even though it is backward looking, historical volatility is used more often than not as an expectation of future volatility. One example of a historical measure is the standard deviation, which we will use to create the volatility-adjusted moving average.
- **Implied volatility**: In its simplest definition, implied volatility is the measure that when inputted into the Black-Scholes equation, gives out the option's market price. It is considered as the expected future actual volatility by market participants. It has one time scale, the option's expiration.
- **Forward volatility**: It is the volatility over a specific period in the future.

- **Actual volatility**: It is the amount of volatility at any given time. Also known as local volatility, this measure is hard to calculate and has no time scale.

The most basic type of volatility is our old friend "the standard deviation". It is one of the pillars of descriptive statistics and an important element in some technical indicators such as the Bollinger bands. But first, let us define what variance is before we find standard deviation. Variance is the squared deviations from the mean (a dispersion measure), we take the square deviations so as to force the distance from the mean to be non-negative. Finally, we take the square root to make the measure have the same units as the mean, in a way we are comparing apples to apples (mean to standard deviation). Variance is calculated through this formula:

$$\sigma^2 = \frac{\sum(x - \bar{x})^2}{n - 1}$$

Following our logic, standard deviation is therefore:

$$\sigma = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$$

Therefore, if we want to understand the concept in layman's terms, we can say that standard deviation is the average distance away from the mean that we expect to find when we analyze the different components of the time series. Now, let us code a function that calculates a rolling standard deviation of a certain time series.

```
def volatility(Data, lookback, what, where):
    # Adding an extra column
    Data = adder(Data, 1)
    for i in range(len(Data)):
        try:
            Data[i, where] = (Data[i - lookback + 1:i + 1, what].std())
        except IndexError:
            pass
    # Cleaning
    Data = jump(Data, lookback)
    return Data
```

The volatility-adjusted moving average — VAMA — is a technical indicator created by Tushar S. Chande. Commonly referred to as the variable index moving average, it is a powerful indicator that uses two standard deviation periods to account for recent volatility. I have taken the liberty to rename it into the volatility-adjusted moving average because there is a newer version with the same name that uses the Chande momentum oscillator instead of the standard deviation, hence it is important to distinguish the difference between the two. The first step to calculate the VAMA is to measure the alpha which can be found through this formula below:

$$\alpha_n = \frac{\sigma_{Short}}{\sigma_{Long}} \; x \; 0.20$$

Therefore, the alpha is calculated as a ratio between a short-term volatility calculation and a long-term volatility calculation. The result is multiplied by 0.20. Then, to calculate the VAMA, we follow these next steps.

$$VAMA = \alpha_n \, x \, Close_n + (1 - \alpha_n) \, x \, VAMA_{n-1}$$

The first VAMA value is simply the closing price, and then the algorithm can take the form of the above formula. The next chart shows the GBPUSD hourly values with the VAMA using a 3-period standard deviation and a 144-period standard deviation.

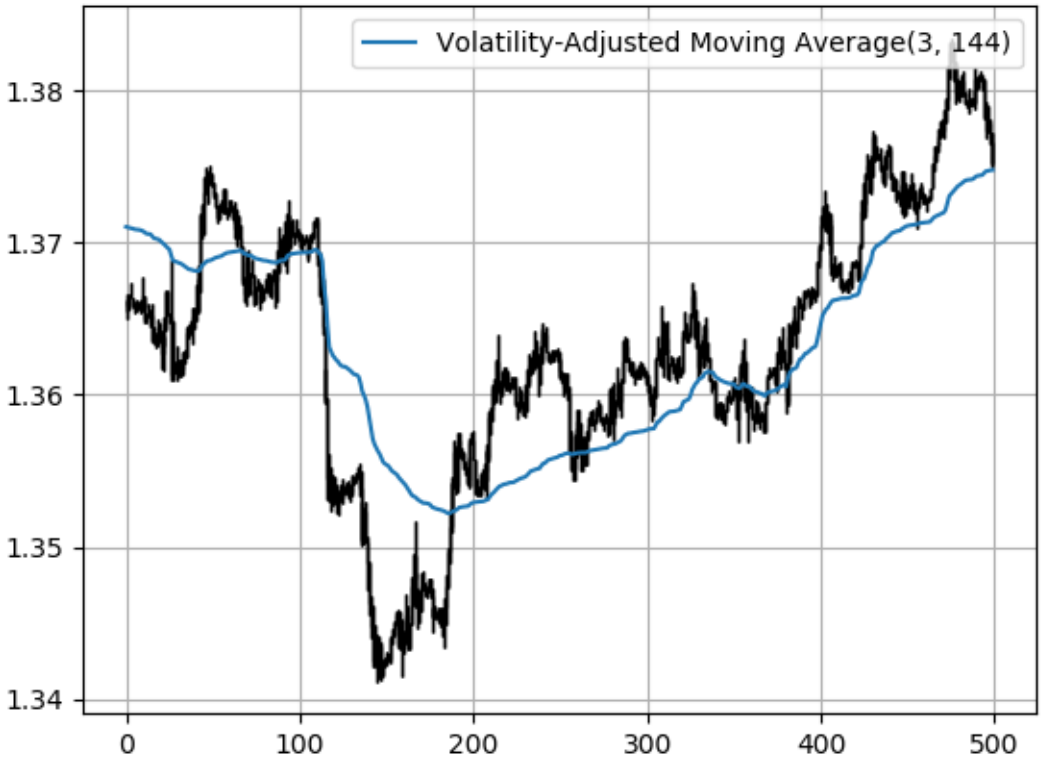



*Figure 15 – Hourly GBPUSD values with the VAMA(3, 144).*

The VAMA adjusts to volatility in a way that reflects the current market conditions making it one of the most up-to-date moving averages. Compared to a simple moving average, the VAMA has the advantage of including much more information in its calculation. However, sometimes, simplicity may work better in the markets. We need to answer just one question, how many

people are following the volatility-adjusted moving average and how does that number compare to the ones following the simple moving average? The answer could already give us a clue that financial markets never give a straight answer. We are not trying to compare because we can actually use both which is recommended. Therefore, do not hesitate to use the two moving averages in order to confirm a certain support or resistance level. To code the VAMA, we can use the below function.

```
def volatility_adjusted_moving_average(Data, lookback_volatility_short, lookback_volatility_long, close, where):
    # Adding Columns
    Data = adder(Data, 2)
    # Calculating Standard Deviations
    Data = volatility(Data, lookback_volatility_short, close, where)
    Data = volatility(Data, lookback_volatility_long, close, where + 1)
    # Calculating Alpha
    for i in range(len(Data)):
        Data[i, where + 2] = 0.2 * (Data[i, where] / Data[i, where + 1])
    # Calculating the First Value of VAMA
    Data[1, where + 3] = (Data[1, where + 2] * Data[1, close]) + ((1 - Data[1, where + 2]) * Data[0, close])
    # Calculating the Rest of VAMA
    for i in range(2, len(Data)):
        Data[i, where + 3] = (Data[i, where + 2] * Data[i, close]) + ((1 - Data[i, where + 2]) * Data[i - 1, where + 3])
    # Cleaning
    Data = deleter(Data, where, 3)
    Data = jump(Data, 1)
    return Data
```

# CHAPTER 7 THE FIBONACCI MOVING AVERAGE

Before we discuss this wonderful indicator, we need to understand the concept of the Fibonacci sequence. Leonardo Bonacci also known as Leonardo Fibonacci (which is a nickname to say son of Bonacci), has discovered one of the most fascinating series in our universe using simple addition techniques while observing rabbit populations. Now, to be fair, there is some evidence that suggest Indian mathematicians knew this sequence beforehand, we will stick to the widely accepted fact that Fibonacci came up with the sequence. Fibonacci numbers are simply obtained using the following simple formula for n > 2:

$$F_n = F_{n-1} - F_{n-2}$$

This gives us the following sequence that goes to infinity:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, \ldots.$$

We will use the Fibonacci sequence as lookback periods in the Fibonacci moving average. Here are the steps we will take to create the indicator:

- Calculate exponential moving averages using the following lookbacks {5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181} on the highs and the lows.
- For each time step (Row), we take the average of all 15 exponential moving averages. Note that the second step will use the simple moving average function as opposed to the exponential one.

Therefore, we will basically be calculating a simple moving average on the exponential moving averages which are based on the Fibonacci sequence. Now, another particularity with the Fibonacci moving average is that we calculate it as a zone. This means that the calculation is done on the highs and the lows

instead of just the closing prices. By doing so, we will have two lines representing dynamic support and resistance zones.

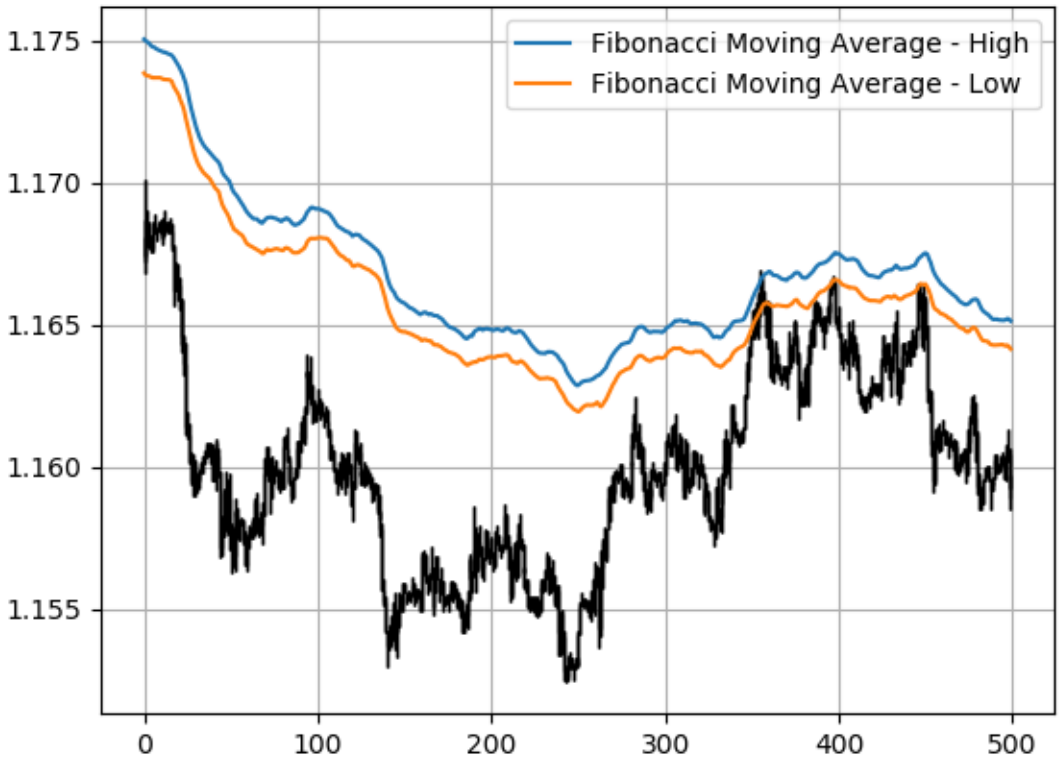



*Figure 16 – Hourly EURUSD values with the Fibonacci moving average.*

The chart shows the market's reaction close to the Fibonacci moving average. The idea is to anticipate contrarian reactions every time the market enters the zone. For instance, the chart shows three bearish reactions each time the market approaches or enters the zone from the below, this is why we refer to it as a dynamic support and resistance zone. The code for the Fibonacci moving average can be as follows.

```
def fma(Data, where):
    # Adding Columns
    Data = adder(Data, 2)
    # Calculating Different Moving Averages
    Data = ema(Data, 2, 5,    1, where)
    Data = ema(Data, 2, 8,    1, where + 1)
    Data = ema(Data, 2, 13,   1, where + 2)
    Data = ema(Data, 2, 21,   1, where + 3)
    Data = ema(Data, 2, 34,   1, where + 4)
    Data = ema(Data, 2, 55,   1, where + 5)
    Data = ema(Data, 2, 89,   1, where + 6)
    Data = ema(Data, 2, 144,  1, where + 7)
    Data = ema(Data, 2, 233,  1, where + 8)
    Data = ema(Data, 2, 377,  1, where + 9)
    Data = ema(Data, 2, 610,  1, where + 10)
    Data = ema(Data, 2, 987,  1, where + 11)
    Data = ema(Data, 2, 1597, 1, where + 12)
    Data = ema(Data, 2, 2584, 1, where + 13)
    Data = ema(Data, 2, 4181, 1, where + 14)
    # Calculating the High FMA
    Data[:, where + 15] = (Data[:, where]     + \
                           Data[:,  where + 1]  + \
                           Data[:,  where + 2]  + \
                           Data[:,  where + 3]  + \
                           Data[:,  where + 4]  + \
                           Data[:,  where + 5]  + \
                           Data[:,  where + 6]  + \
                           Data[:,  where + 7]  + \
                           Data[:,  where + 8]  + \
                           Data[:,  where + 9]  + \
```

```
                Data[:,  where + 10] + \
                Data[:,  where + 11] + \
                Data[:,  where + 12] + \
                Data[:,  where + 13] + \
                Data[:,  where + 14]) / 15
Data = deleter(Data, where, 15)
# Calculating Different Moving Averages
Data = ema(Data, 2, 5,    2, where + 1)
Data = ema(Data, 2, 8,    2, where + 2)
Data = ema(Data, 2, 13,   2, where + 3)
Data = ema(Data, 2, 21,   2, where + 4)
Data = ema(Data, 2, 34,   2, where + 5)
Data = ema(Data, 2, 55,   2, where + 6)
Data = ema(Data, 2, 89,   2, where + 7)
Data = ema(Data, 2, 144,  2, where + 8)
Data = ema(Data, 2, 233,  2, where + 9)
Data = ema(Data, 2, 377,  2, where + 10)
Data = ema(Data, 2, 610,  2, where + 11)
Data = ema(Data, 2, 987,  2, where + 12)
Data = ema(Data, 2, 1597, 2, where + 13)
Data = ema(Data, 2, 2584, 2, where + 14)
Data = ema(Data, 2, 4181, 2, where + 15)
# Calculating the High FMA
Data[:, where + 16] = (Data[:, where + 1]  + \
                Data[:,  where + 2]  + \
                Data[:,  where + 3]  + \
                Data[:,  where + 4]  + \
                Data[:,  where + 5]  + \
                Data[:,  where + 6]  + \
                Data[:,  where + 7]  + \
                Data[:,  where + 8]  + \
```

```
                Data[:, where + 9]  + \
                Data[:, where + 10] + \
                Data[:, where + 11] + \
                Data[:, where + 12] + \
                Data[:, where + 13] + \
                Data[:, where + 14] + \
                Data[:, where + 15]) / 15
    Data = deleter(Data, where + 1, 15)
    return Data
```

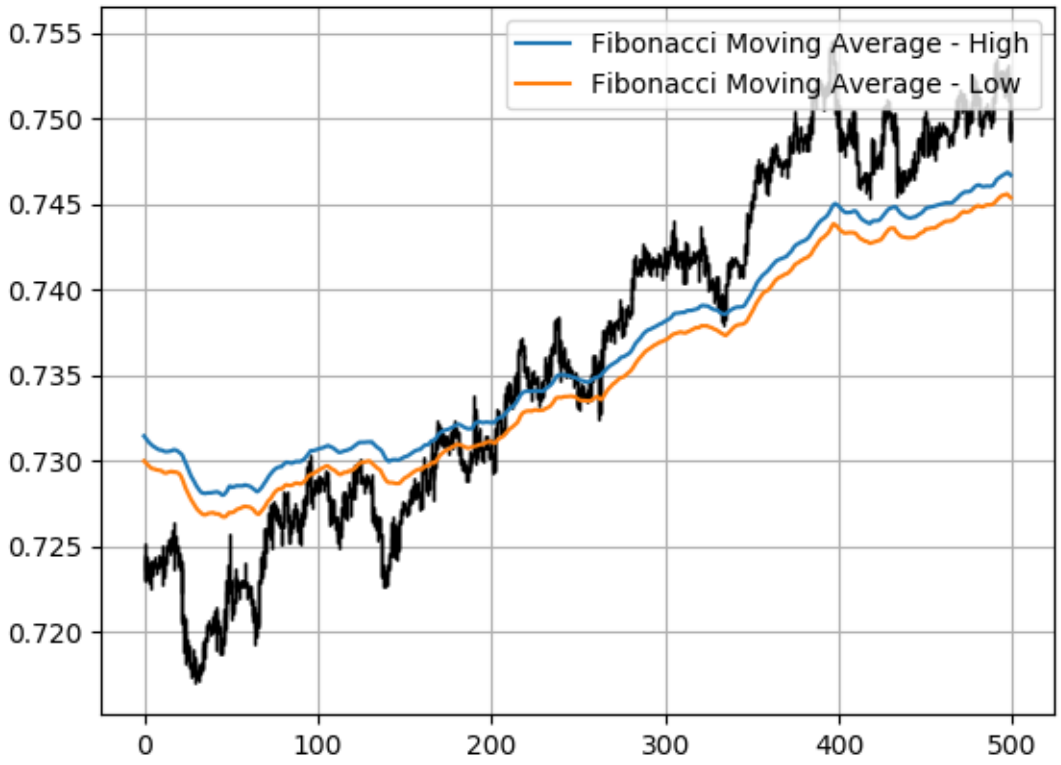



*Figure 17 – Hourly AUDUSD values with the Fibonacci moving average.*

As a break of the neutrality protocol, I should mention that the Fibonacci moving average is my favorite overlay indicator due to its powerful reactivity levels.

# CHAPTER 8 THE ADAPTIVE MOVING AVERAGE

The adaptive moving average (and specifically Kaufman's adaptive moving average) has been created to reduce noise and whipsaw effects. It works the same way as other moving averages and follows the same intuition. The generation of false trading signals is one of the problems with moving averages and this is due to short-term sudden fluctuations that bias the calculation. The KAMA's primary objective is to reduce as much noise as possible.

The first concept we should measure is the efficiency ratio which is the absolute change of the current close relative to the change in the past 10 periods (by default) divided by a type of volatility calculated in a special manner. We can say that the efficiency ratio is the change divided by volatility.

$$Efficiency\ Ratio = \frac{|Close_n - Close_{n-10}|}{Sum\ of\ |Close_n - Close_{n-1}|}$$

Then we calculate a smoothing constant based on the following formula:

$$Smoothing\ Constant = (Efficiency\ Ratio\ x\ 0.6667)^2$$

Finally, to calculate the KAMA we use the below formula:

$$KAMA = KAMA_{n-1} + (Smoothing\ Constant\ .\ (Closing\ Price\ .\ KAMA_{n-1}))$$

```
def kama(Data, what, where, lookback):
    Data = adder(Data, 10)
    # Lookback from previous period
    for i in range(len(Data)):
        Data[i, where] = abs(Data[i, what] - Data[i - 1, what])
    Data[0, where] = 0
```

```
    # Sum of lookbacks
    for i in range(len(Data)):
        Data[i, where + 1] = (Data[i - lookback + 1:i + 1, where].sum())
    # Volatility
    for i in range(len(Data)):
        Data[i, where + 2] = abs(Data[i, what] - Data[i - lookback, what])
    Data = Data[lookback + 1:, ]
    # Efficiency Ratio
    Data[:, where + 3] = Data[:, where + 2] / Data[:, where + 1]
    for i in range(len(Data)):
        Data[i, where + 4] = np.square(Data[i, where + 3] * 0.6666666666666666667)
    for i in range(len(Data)):
        Data[i, where + 5] = Data[i - 1, where + 5] + (Data[i, where + 4] * (Data[i, what] -
Data[i - 1, where + 5]))
        Data[11, where + 5] = 0
    Data = deleter(Data, where, 5)
    Data = jump(Data, lookback * 2)
    return Data
```

The next chart illustrates what the above function can give us. Notice the more stable tendency of the KAMA relative to other moving averages. It is an improvement with regards to whipsaws and false breaks. Surely, there will be some false breakouts but when we compare apples to apples, we will find that the adaptive moving average does a good job in lowering the number of false signals. Also, when it is flat, it signals that the market is choppy.

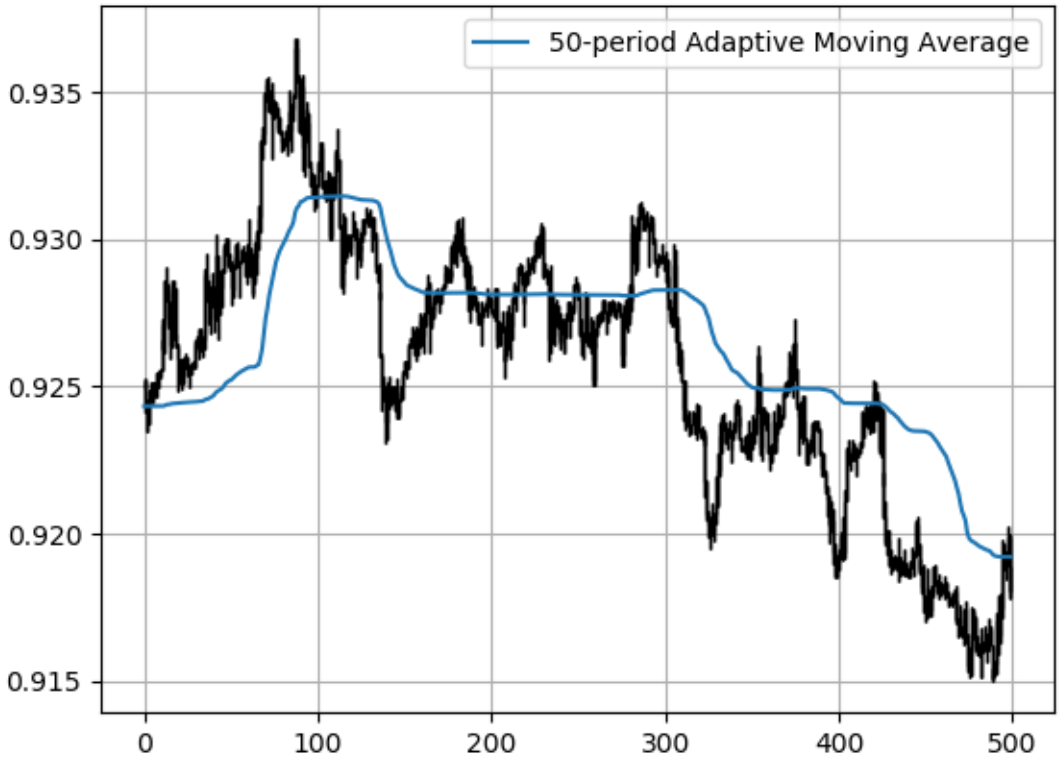



*Figure 18 – Hourly USDCHF values with the 50-hour adaptive moving average.*

The KAMA can also be used to detect the start of new trends. For instance, we can plot a long-term KAMA with a short-term KAMA and trade on their crossovers. If the default settings are 10-period, we can try a 30-period KAMA and see what it gives us. Therefore, an algorithm based on this strategy will have the following trading rules:

- **A bullish signal is generated whenever the 10-period KAMA crosses over the 30-period KAMA.**
- **A bearish signal is generated whenever the 10-period KAMA crosses under the 30-period KAMA.**

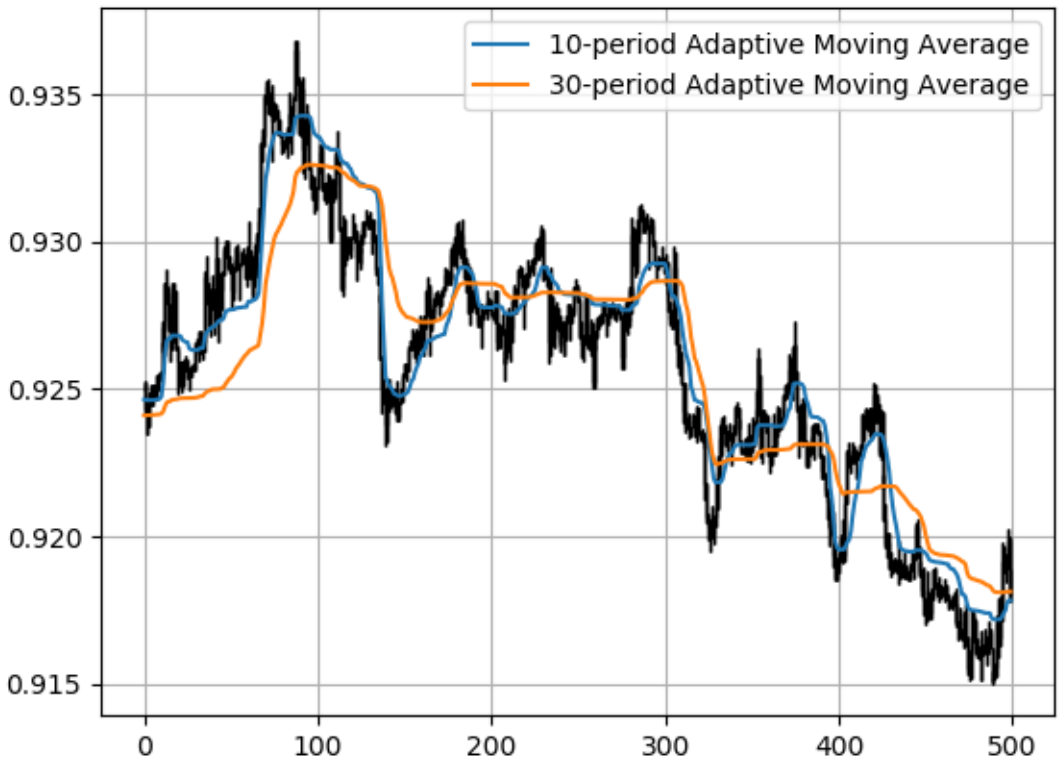



*Figure 19 – Hourly USDCHF values with the 10-hour and 30-hour adaptive moving averages.*

The KAMA provides valuable support and resistance levels that can be used to complement your trading system. It is not meant to be used as a sole contributor to generate trading signals, but it is valuable in confirming them. Remember that a dynamic support and resistance level is one of the best ways to find reactionary levels. Combined with other strategies, these could be extremely powerful.

The next chart shows the EURUSD's hourly values with the 70-period KAMA. We can notice how it fully caps the market price where good short opportunities were present each time the market approaches the moving average.

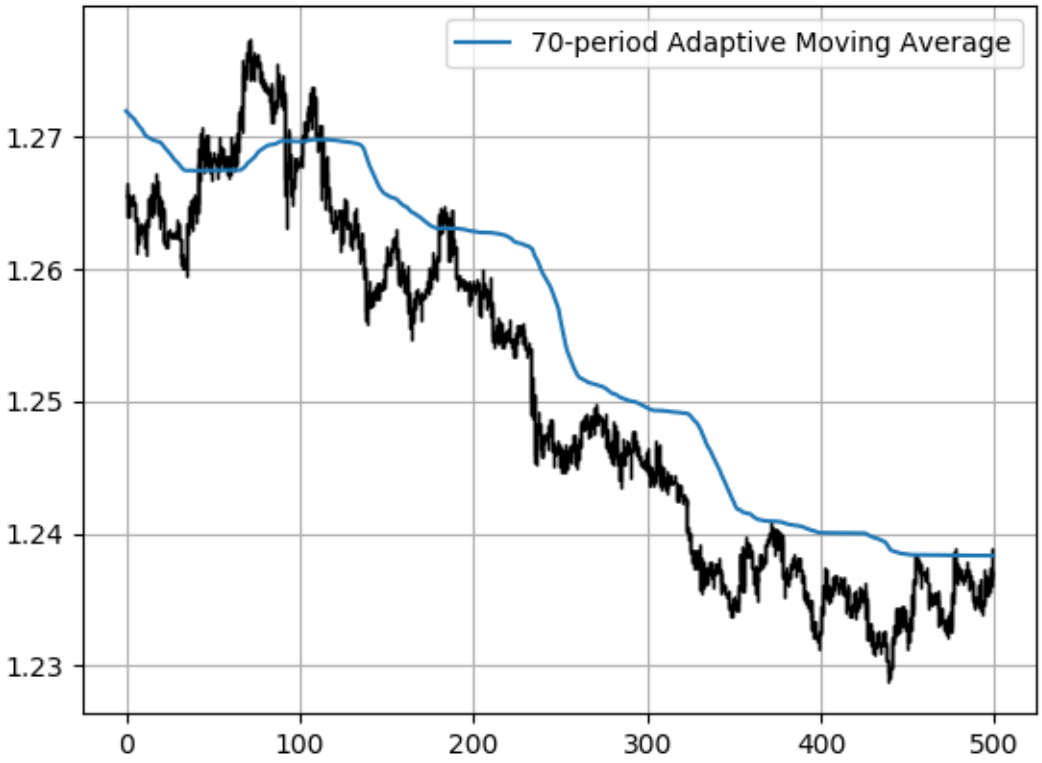



*Figure 20 – Hourly USDCAD values with the 70-hour adaptive moving average.*

To conclude, is the KAMA a good moving average? Definitely, this type of self-adjusting moving averages is on the way of becoming the most used kind. At the moment, the simple moving average has only one advantage, popularity. When market participants drop it for more reactive moving averages, the simple type will lose its appeal. I recommend using both to confirm the signal.

# CHAPTER 9 THE FRACTAL ADAPTIVE MOVING AVERAGE

Developed by John Ehlers, the FRAMA is a special type of moving average that uses exponential smoothing based on the current Fractal dimension index calculated on the price. It is also adaptive in that it is able to follow strong moves and at the same time takes into account corrections and consolidations. However, to understand the FRAMA, we must at least have an idea on the rational and the code of the Fractal dimension index.

The index measures how irregular the time series is. When there is a strong trend going on, naturally, the market price is approaching a one-dimensional line and the Fractal dimension index approaches 1.0 as per mathematical definition. When the market is choppy and moving sideways, the Fractal dimension index approaches 2.0. The below chart shows the EURUSD's hourly values against the 13-period Fractal dimension index.

*Figure 21 – Hourly EURUSD values with the 13-hour Fractal dimension index.*

The chart shows that generally, when the market is trending, the FDI shows a value below 1.50 and when the market is ranging, the FDI shows a value above 1.50. No measure is perfect and the FDI is also not intended to be a signal generator, but merely a current state market analysis.

```
def fractal_dimension(Data, lookback, high, low, close, where):
    Data = adder(Data, 10)
    for i in range(len(Data)):
        try:
            # Calculating N1
            Data[i, where]     = max(Data[i - (2 * lookback):i - lookback, high])
            Data[i, where + 1] = min(Data[i - (2 * lookback):i - lookback, low])
            Data[i, where + 2] = (Data[i, where] - Data[i, where + 1]) / lookback
            # Calculating N2
            Data[i, where + 3] = max(Data[i - lookback:i, high])
            Data[i, where + 4] = min(Data[i - lookback:i, low])
            Data[i, where + 5] = (Data[i, where + 3] - Data[i, where + 4]) / lookback
            # Calculating N3
            Data[i, where + 6] = max(Data[i, where], Data[i, where + 3])
            Data[i, where + 7] = min(Data[i, where + 1], Data[i, where + 4])
            Data[i, where + 8] = (Data[i, where + 6] - Data[i, where + 7]) / (2 * lookback)
            # Calculating the Fractal Dimension Index
            if Data[i, where + 2] > 0 and Data[i, where + 5] > 0 and Data[i, where + 8] > 0:
                Data[i, where + 9] = (np.log(Data[i, where + 2] + Data[i, where + 5]) -
np.log(Data[i, where + 8])) / np.log(2)
        except ValueError:
            pass
    Data = deleter(Data, where, 9)
    Data = jump(Data, lookback * 2)
    return Data
```

The first thing we need to calculate with the FRAMA is the alpha factor. The alpha factor deals with the fractal smoothing part of the formula and can be calculated as follows.

$$\alpha_n = e^{(-4.6\ x\ (FDI_n - 1))}$$

The above formula means that the alpha factor at any point in time equals the exponential function raised to the power of -4.6 multiplied by the current Fractal dimension index reading minus 1. Let us assume that we have defined the code of the FDI and that we have an OHLC data array. To calculate a new column populated by the alpha factor, we follow this code:

```
# Indicator Parameters
lookback = 13
# Calculating a 13-period Fractal Dimension Index
my_data = fractal_dimension(my_data, lookback, 1, 2, 3, 4)
# Adding a Few Columns
my_data = adder(my_data, 1)
# Calculating the alpha factor
for i in range(len(my_data)):
   my_data[i, 5] = np.exp(-4.6 * (my_data[i, 4] - 1))
```

The FRAMA is calculated following this below function.

$$FRAMA_n = \alpha_n\ x\ Price_n + (1 - \alpha_n)\ x\ FRAMA_{n-1}$$

```
# Calculating the First Value of FRAMA
my_data[1, 6] = (my_data[1, 5] * my_data[1, 3]) + ((1 - my_data[1, 5]) * my_data[0, 3])
# Calculating the Rest of FRAMA
for i in range(2, len(Data)):
    try:
        my_data[i, 6] = (my_data[i, 5] * my_data[i, 3]) + ((1 - my_data[i, 5]) * my_data[i - 1,
6])
except IndexError:
        pass
```

The below chart shows the values of EURUSD with the 13-period FRAMA.

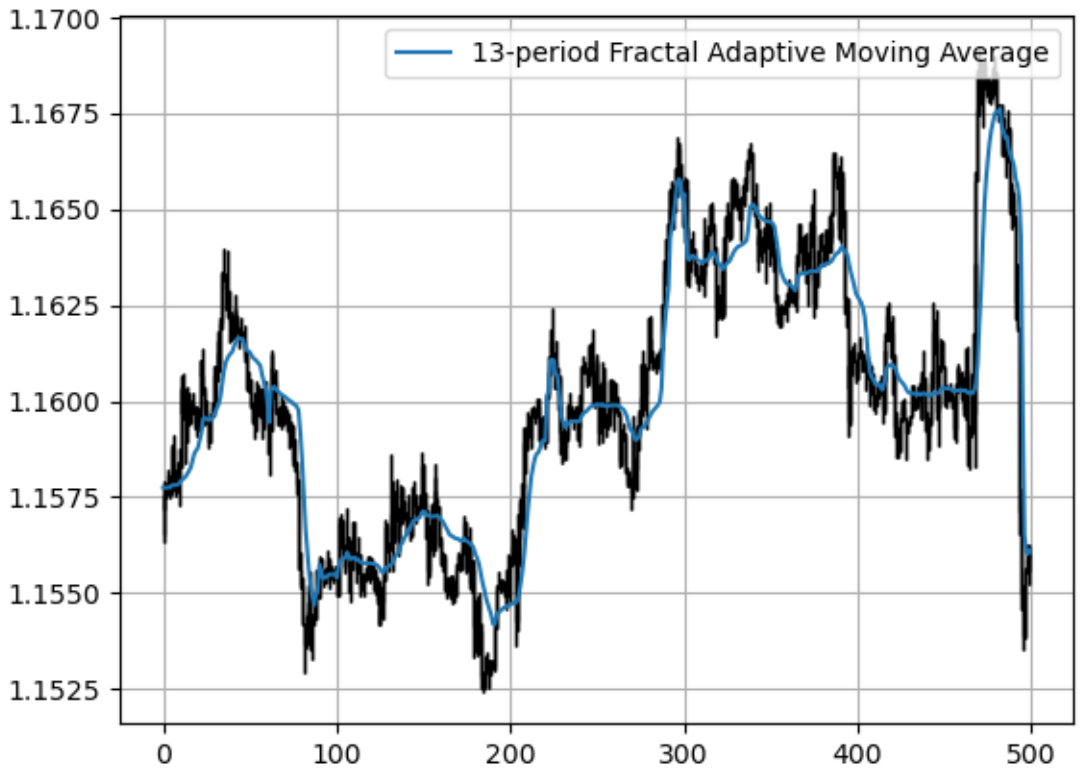



*Figure 22 – Hourly EURUSD values with the 13-hour FRAMA.*

We can see how the moving average changes its behavior depending on the change in the market regime and dynamics. This property makes the FRAMA good for detecting trend changes but not necessarily good for contrarian trading based on dynamic support and resistance levels. Remember, a dynamic support or resistance level is based on a moving average that provides up-to-date levels. Another particularity specific to the FRAMA is that changing the lookback period does not behave linearly as we would expect when we change the period on a simple or exponential moving average. Generally, a 200-period simple moving average is more stable than a 20-period simple moving average, however, as the FRAMA depends on the Fractal dimension index, this relationship does not hold.

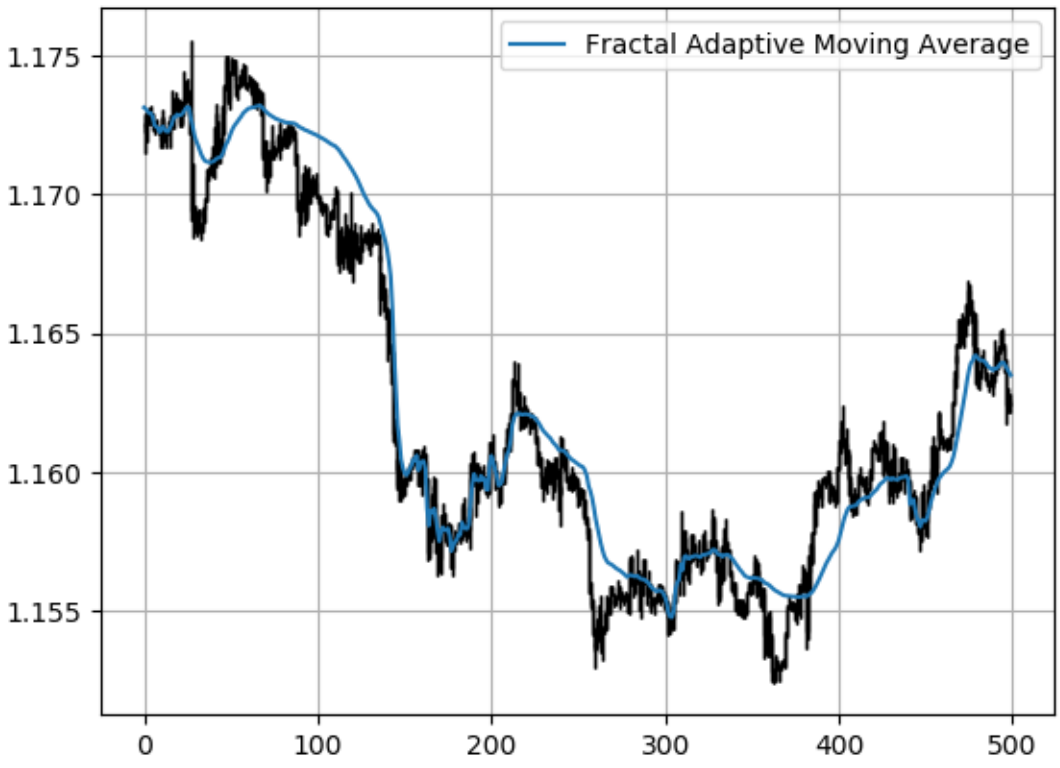



*Figure 23 – Hourly EURUSD values with the 55-hour FRAMA.*

As can be seen from the previous chart where we applied a 55-period FRAMA rather than a 13-period, the FRAMA is not always far from the price and is even closer to the price than the 13-period FRAMA sometimes. The recommended strategies with the FRAMA are the below:

- **A simple crossover of the price**: This is the simplest strategy on moving averages, and it still works on the FRAMA as well. Whenever the market price surpasses its FRAMA, a bullish signal is generated and whenever the market price breaks its FRAMA, a bearish signal is generated. For added accuracy and confirmation, we can use the highs and lows to confirm the signal.
- **A double crossover from two FRAMA's**: Although it needs optimization and tweaking, a crossover between two FRAMA's may give out bullish and bearish signals.

# CHAPTER 10 THE TRIANGULAR MOVING AVERAGE

We have reached the final chapter on moving averages before we move on to discussing other indicators. Moving averages come in all shapes and forms and each type has its advantages and limitations. This is a simple moving average applied on a simple moving average. Therefore, we are just using the function twice but on different time series.

**# Indicator Parameters**

lookback = 60

my_data = ma(my_data, lookback, 3, 4)

my_data = ma(my_data, lookback, 4, 5)

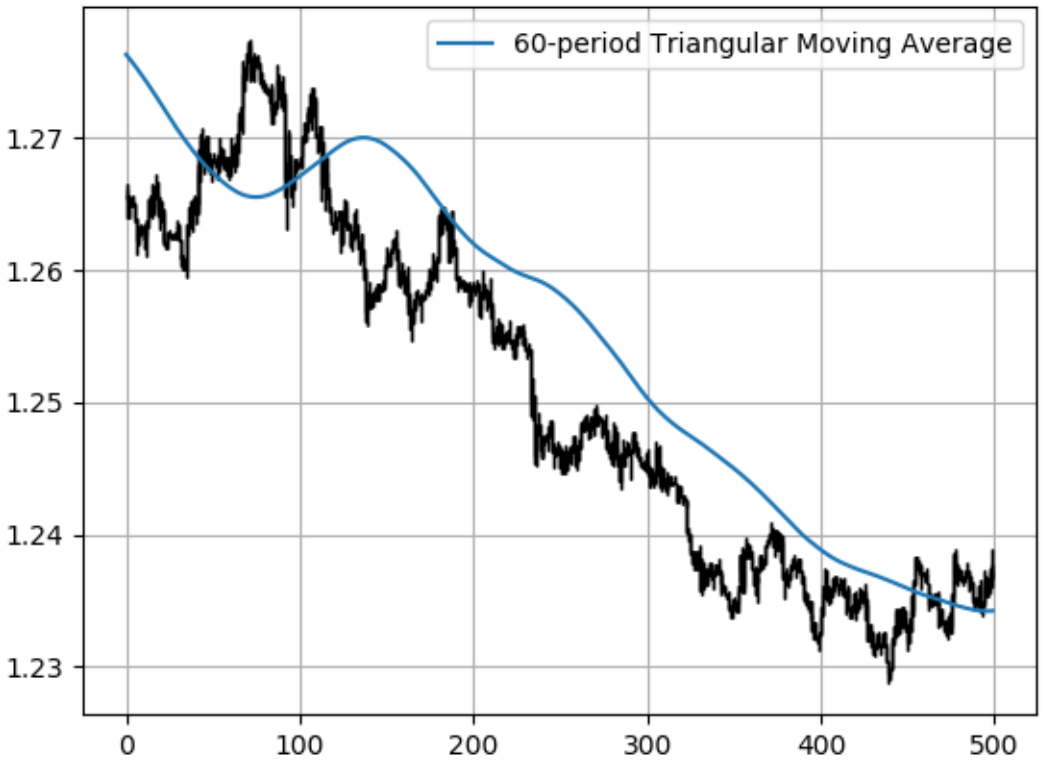



*Figure 24 – Hourly USDCAD values with the 60-hour triangular moving average.*

The above chart shows the USDCAD's hourly values with the 60-period triangular moving average which seems to be more lagging. The strategies

used by this type of moving average are the same as the common ones such as crossovers and dynamic support/resistance levels.

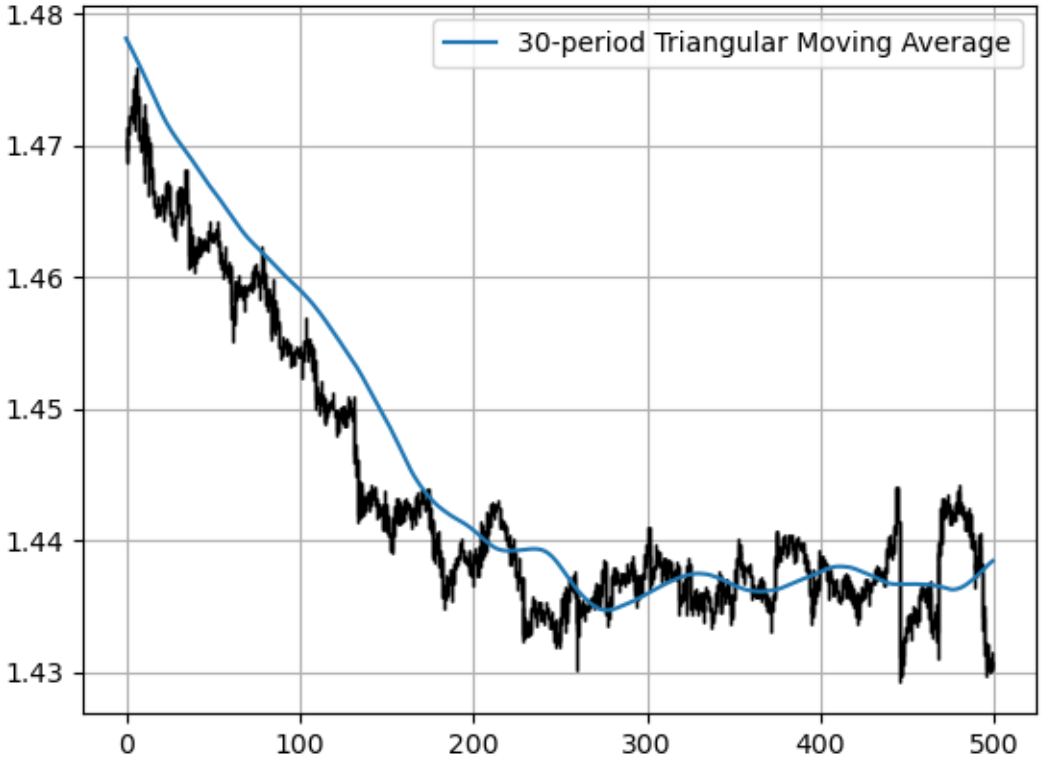



*Figure 25 - Hourly USDCAD values with the 30-hour triangular moving average.*

It is worth mentioning that you do not need to copy the code from the book as there is a dedicated GitHub which harbors every line of code seen in the book organized properly. Focus on understanding the concepts while moving forward.

# PART 2 TREND-FOLLOWING INDICATORS

Aside from moving averages, we have other technical indicators that are calculated differently and have the same purpose, that is, trend-following. As a reminder, an overlay indicator is one that is charted alongside the market in the same panel, just like moving averages while a regular indicator is mostly shown in another panel. This is because it has a different axis and scale. This part will deal with other well-known technical indicators, whether overlay or regular, and we will see how to code them along the historical OHLC data that we have imported.

Remember that the purpose here is to enlarge our horizons and be exposed to more indicators so that we become much more comfortable with what we use. Unfortunately, most traders using technical indicators are not even aware of how they are calculated.

This is a major problem as you will not be able to understand the limitations of the indicator and therefore fall in more traps. We must keep in mind that indicators generally help us confirm our biases and only in rare cases form strategies on their own, which is generally not recommended. Therefore, setting our expectations right, we will commence the part knowing that we will need a lot of tweaking on our own to find that right balance between risk and reward.

# CHAPTER 11 THE MACD OSCILLATOR

An abbreviation for Moving Average Convergence Divergence, the MACD oscillator is one of the most common technical indicators and is one of the most known oscillators in technical analysis. Many people also consider it a trend-following indicator, but others use graphical analysis on it to find reversal points, making the MACD a versatile indicator. How is the MACD calculated? It is the difference between the 26-period exponential moving average applied to the closing price and the 12-period exponential moving average also applied to the closing price.

The value found after taking the difference is called the MACD line. The 9-period exponential moving average of that calculation is called the MACD signal. Therefore, the MACD oscillator is composed of two lines, the MACD line and the MACD signal.

Both have their own strategies, and their behavior together is also analyzed so that we better understand the trend. Let us start first by creating the MACD oscillator in python and then we can discuss its strategies and how to chart it. In the meantime, we can visualize it in the next chart.

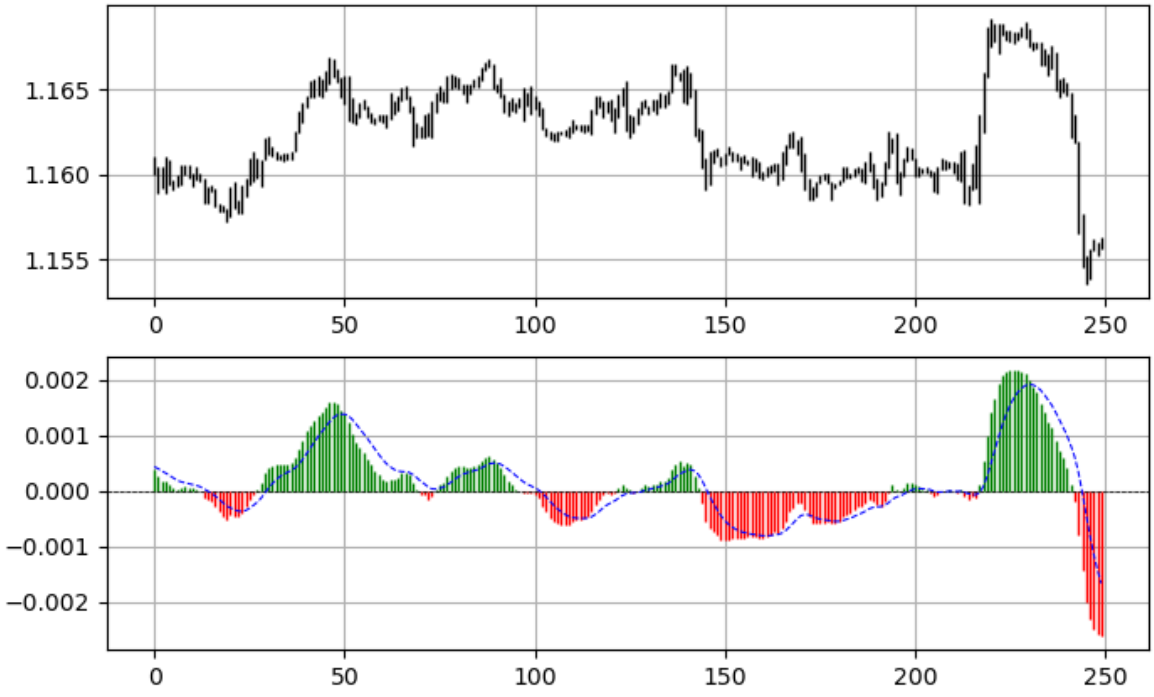



*Figure 26 - Hourly EURUSD values with the MACD.*

```
def macd(Data, what, long_ema, short_ema, signal_ema, where):
    Data = adder(Data, 1)
    # Calculating the two exponential moving averages
    Data = ema(Data, 2, long_ema,  what, where)
    Data = ema(Data, 2, short_ema, what, where + 1)
    # Calculating the difference between the two exponential moving averages
    Data[:, where + 2] = Data[:, where + 1] - Data[:, where]
    Data = jump(Data, long_ema)
    # Calculating the MACD signal
    Data = ema(Data, 2, signal_ema, where + 2, where + 3)
    Data = deleter(Data, where, 2)
    Data = jump(Data, signal_ema)
    return Data
```

```
def indicator_plot_double_macd(Data, MACD_line, MACD_signal, window = 250):
    fig, ax = plt.subplots(2, figsize = (8, 5))
    Chosen = Data[-window:, ]
    for i in range(len(Chosen)):
        ax[0].vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'black', linewidth = 1)
    ax[0].grid()
    for i in range(len(Chosen)):
        if Chosen[i, MACD_line] > 0:
            ax[1].vlines(x = i, ymin = 0, ymax = Chosen[i, MACD_line], color = 'green', linewidth = 1)
        if Chosen[i, MACD_line] < 0:
            ax[1].vlines(x = i, ymin = Chosen[i, MACD_line], ymax = 0, color = 'red', linewidth = 1)
        if Chosen[i, MACD_line] == 0:
            ax[1].vlines(x = i, ymin = Chosen[i, MACD_line], ymax = 0, color = 'black', linewidth = 1)
    ax[1].grid()
    ax[1].axhline(y = 0, color = 'black', linewidth = 0.5, linestyle = '--')
    ax[1].plot(Data[-window:, MACD_signal], color = 'blue', linewidth = 0.75, linestyle = 'dashed')
```

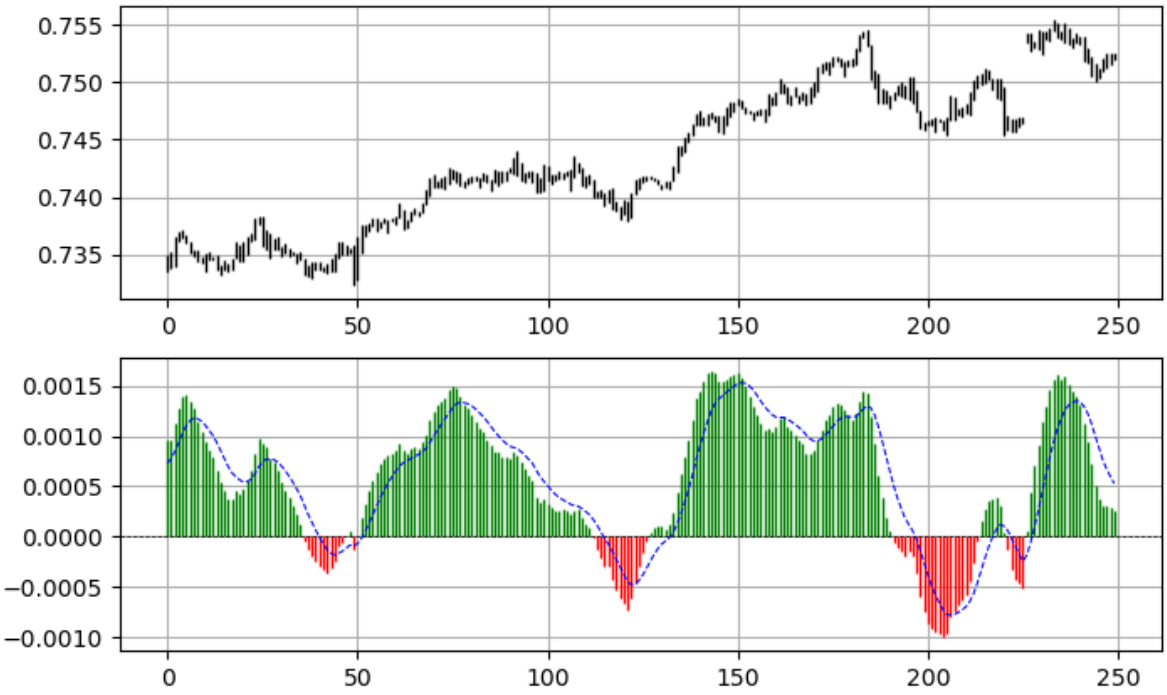



*Figure 27 - Hourly AUDUSD values with the MACD.*

The MACD is very useful as it is heavily followed by traders and can give good trend-following signals. Two strategies are famous with this indicator, the first is a simple crossover where the signals are generated based on the following conditions:

- **A long (Buy) signal is generated whenever the MACD surpasses the zero line.**
- **A short (Sell) signal is generated whenever the MACD breaks the zero line.**

Remember that a long position is synonymous to a buy position, but a short position is not really a sell order as it is merely a position on the other side where you borrow the asset from a party, sell it to a third party, and then buy it back at the market price to return it to the original party. This way, you can profit from a decreased price. The second strategy relies on crossovers between the MACD line and its signal line and has the following conditions:

- **A long (Buy) signal is generated whenever the MACD surpasses the signal line.**
- **A short (Sell) signal is generated whenever the MACD breaks the signal line.**

Remember that the signal line is simply a 9-period exponential moving average applied on the MACD. The oscillator may suffer from lag due to the smoothing, but it remains nevertheless one of the most used indicators of all time.

# CHAPTER 12 THE AWESOME OSCILLATOR

The awesome oscillator has been created by Larry Williams and is simply another variation of the MACD oscillator. The main differences are:

- The MACD oscillator uses an exponential moving average while the Awesome oscillator uses a simple moving average.
- The MACD oscillator calculates the moving average on the actual prices while the Awesome oscillator calculates it on the mid-point price (the median between the high and low).
- The MACD uses 12 and 26 as lookback periods while the Awesome oscillator uses 5 and 34 as lookback periods.

We typically use the awesome oscillator just as we use the MACD. To code the oscillator, we can use the below Python function.

```
def awesome_oscillator(Data, high, low, long_ma, short_ma, where):
    # Adding columns
    Data = adder(Data, 2)
    # Mid-point Calculation
    Data[:, where] = (Data[:, high] + Data[:, low]) / 2
    # Calculating the short-term Simple Moving Average
    Data = ma(Data, short_ma, where, where + 1)
    # Calculating the long-term Simple Moving Average
    Data = ma(Data, long_ma, where, where + 2)
    # Calculating the Awesome Oscillator
    Data[:, where + 3] = Data[:, where + 1] - Data[:, where + 2]
    # Removing Excess Columns/Rows
    Data = deleter(Data, where, 3)
    return Data
```

The Awesome oscillator has a color condition that states:

- Whenever the current value is greater than the previous value, the bar must have the color green.
- Whenever the current value is lower than the previous value, the bar must have the color red.

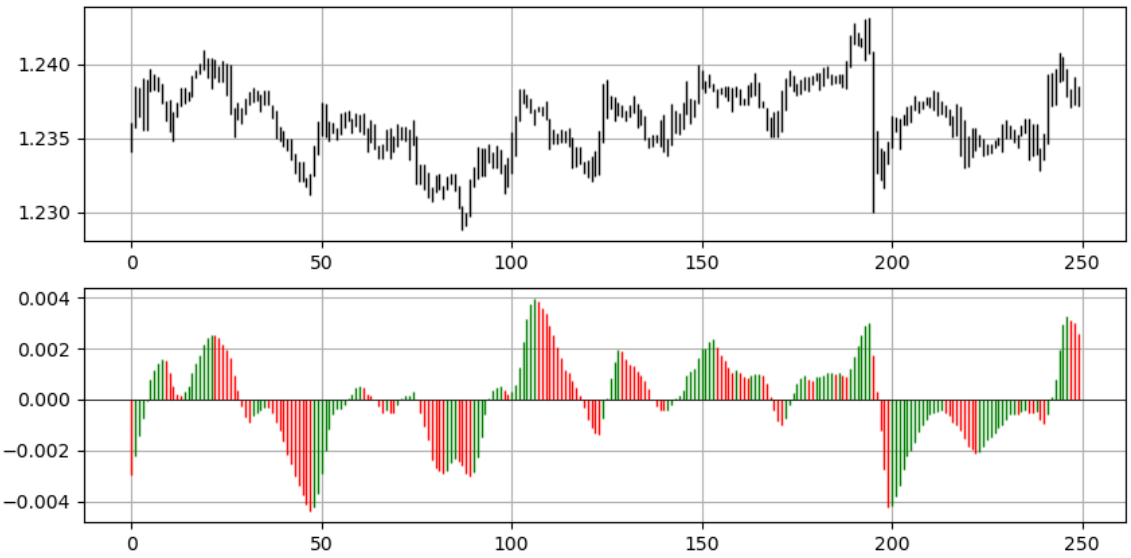



*Figure 28 - Hourly USDCAD values with the awesome oscillator.*

The awesome oscillator is less known to trader which may hinder its usefulness but is still very interesting. Two strategies are famous with this indicator, the first is a simple crossover where the signals are generated based on the following conditions:

- **A long (Buy) signal is generated whenever the oscillator surpasses the zero line.**
- **A short (Sell) signal is generated whenever the oscillator breaks the zero line.**

The second strategy relies on the color switch of the histograms:

- **A long (Buy) signal is generated whenever the histogram switches from red to green.**
- **A short (Sell) signal is generated whenever the histogram switches from green to red.**

It is recommended to confirm the signals of the MACD oscillator with the ones of the awesome oscillator. This could give us a confirmation factor that increases the conviction of our trade idea.

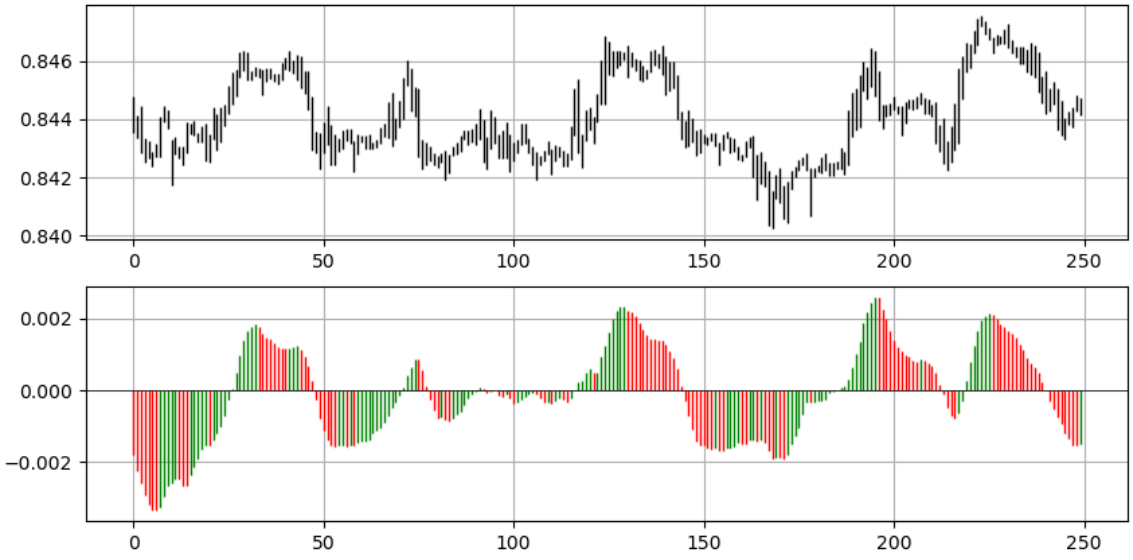



*Figure 29 - Hourly EURGBP values with the awesome oscillator.*

# CHAPTER 13 ICHIMOKU KINKO HYO

Also referred to as a trading system and not just an indicator, the Ichimoku Kinko Hyo is a Japanese phrase that translates to one glance equilibrium chart where multiple overlay indicators are plotted alongside the chart. It is generally thought of as a complex and noisy indicator due to the way it is shown on the charts. Ichimoku Kinko Hyo or Ichimoku is not just a strategy, but a full trading system. It builds on price bars to generate its signals. Developed around the late 1930's by Goichi Hosoda, the system grew to be widely popular in Japan and other locations around the world. It has been quoted as being hard to understand due to its many elements on the chart, but we will soon see that approaching it step by step will make it a crystal-clear indicator.

Before we start, we will briefly present the different parts of the Ichimoku system.

- **Kijun-sen** (The base line): A confirmation line that can act as a trailing stop line.
- **Tenkan-sen** (The conversion line): A signal line that can also act as a minor graphical line.
- **Chikou-span** (The lagging line): A 26-period lagged line of the actual price. Mainly used for confirmation. This is the least useful part of the system due to its huge lagging properties.
- **Ichimoku cloud**: This is the core of the system, and it is a combination of two lines that will form a future support or resistance zone. The lines are called Senkou span A and Senkou span B.

### Kijun-sen

Also called the Base line, the Kijun-sen line is used for medium-term support or resistance level. A straight line is an indication of a flat market and thus prices crossing a straight line should not be interpreted as a trading signal, however, if prices cross a trending line, a trading signal is generated. Also, a flat Kijun-sen line can act as a price-magnet. The formula of the line is:

$$Kijun\ sen = \frac{Highest\ high + Lowest\ low}{2}$$

The formula is for the last 26 periods, hence the highest highs and lowest lows of the last 26 periods. When the market price crosses above the Kijun-sen, it is considered as a bullish signal. When the market price crosses below the Kijun-sen, it is considered as a bearish signal.

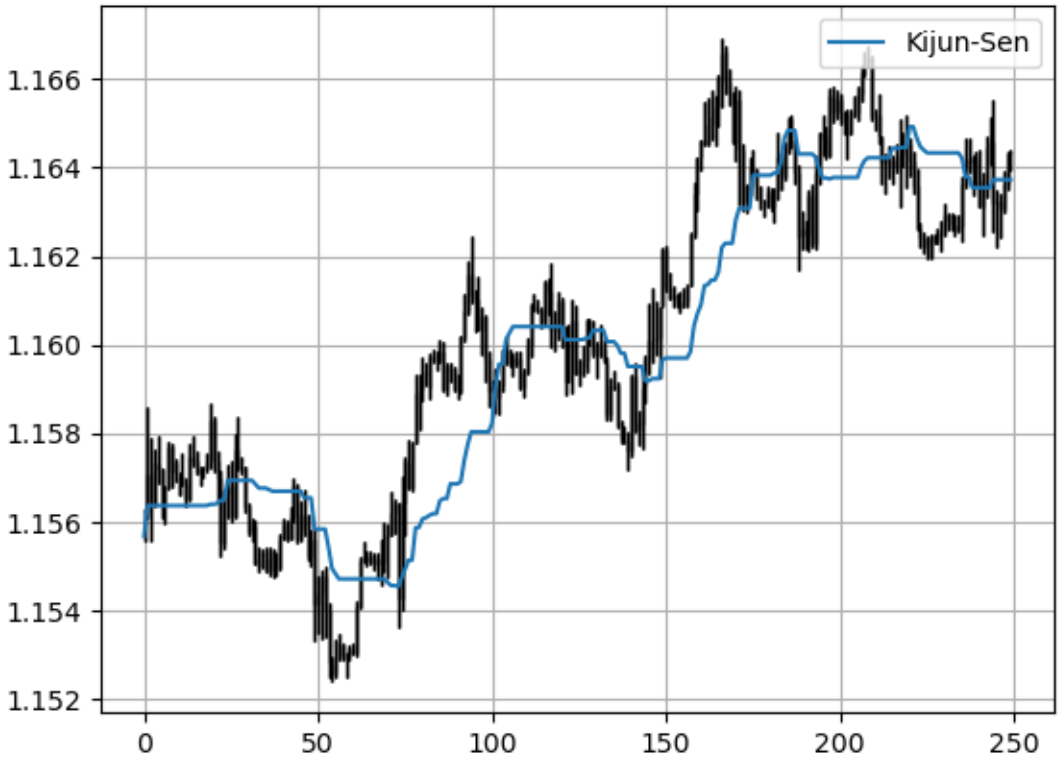



*Figure 30 - Hourly EURUSD values with the Kijun-sen.*

### Tenkan-sen

Also called the Conversion line, this line is the same as the previous one, only it takes the last 9 days (or periods) of the past.

$$Tenkan\ sen = \frac{Highest\ high + Lowest\ low}{2}$$

It can be used to find short-term support and resistance zones and it is conveniently used with the Kijun-sen line for crosses. In ranging markets, we can see many fake signals and hence it is better to seek out crosses in trending markets. When the market price crosses above the Tenkan-sen, it is considered as a bullish signal. When the market price crosses below the Tenkan-sen, it is considered as a bearish signal.

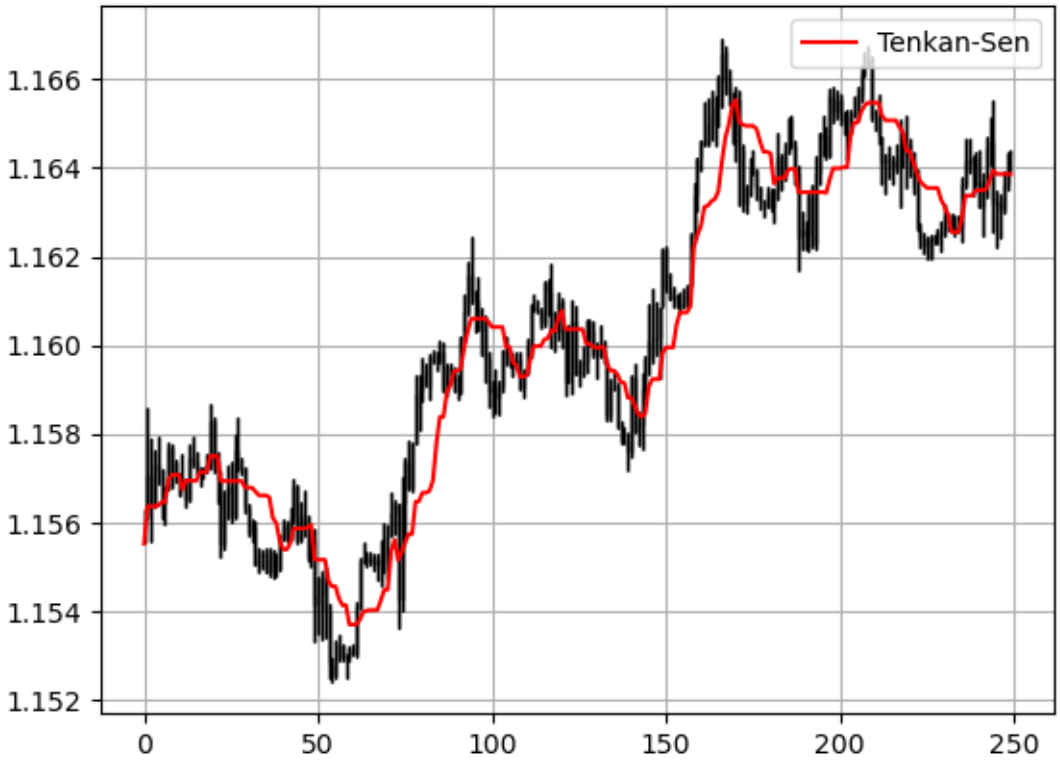



*Figure 31 - Hourly EURUSD values with the Tenkan-sen.*

We actually have a first strategy that can be constructed using some separate parts of the Ichimoku. Before we continue building the system, how about we present a trading strategy relying on the cross between the two lines?

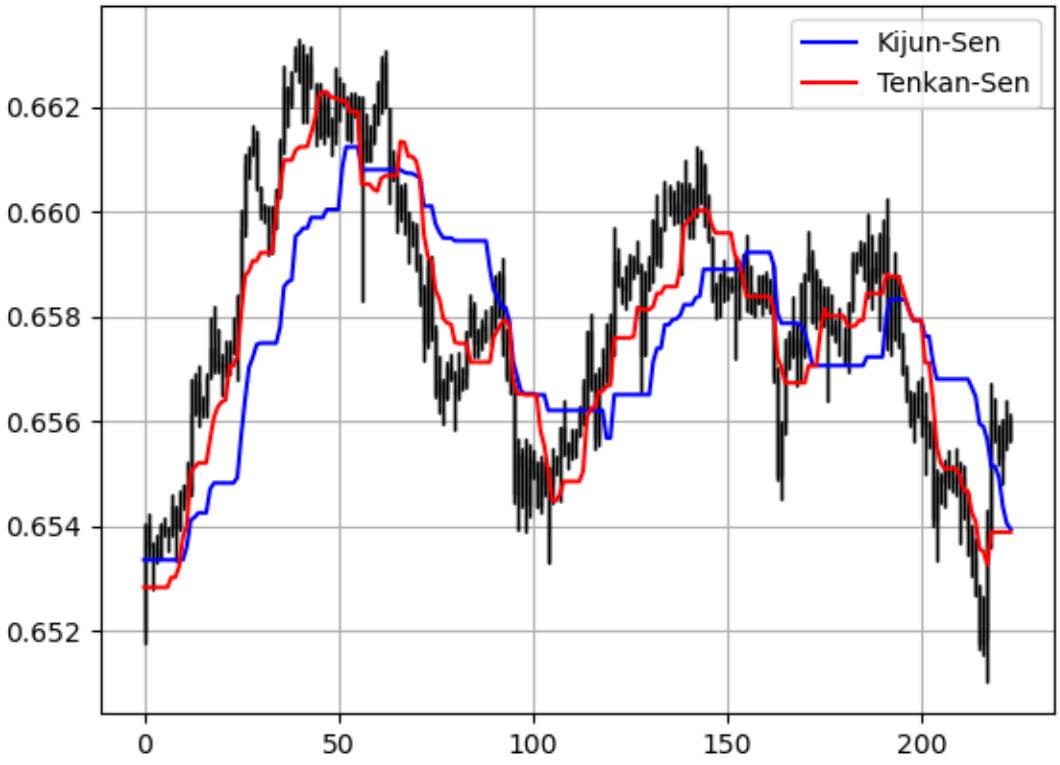



*Figure 32 - Hourly NZDCHF values with the Kijun-sen and the Tenkan-sen.*

Since the Tenkan-sen is shorter-term in nature than the Kijun-sen line, the cross between the two give us trading signals just like a cross between moving averages. The trading rules are as follows:

- **A long (Buy) signal is generated whenever the Tenkan-sen crosses over the Kijun-sen.**
- **A short (Sell) signal is generated whenever the Tenkan-sen crosses below the Kijun-sen.**

### Chikou-span

Also called the Lagging line, Chikou-span is the lagging indicator in the Ichimoku system. It is simply the price today projected back 26 periods. It only uses the closing price. Although its lagging nature, the Chikou-span is a good confirmation signal. It can be used in the following ways:

- When Chikou-span crosses above the market price from 26 periods ago, a bullish confirmation signal is generated.
- When Chikou-span crosses below the market price from 26 periods ago, a bearish confirmation signal is generated.

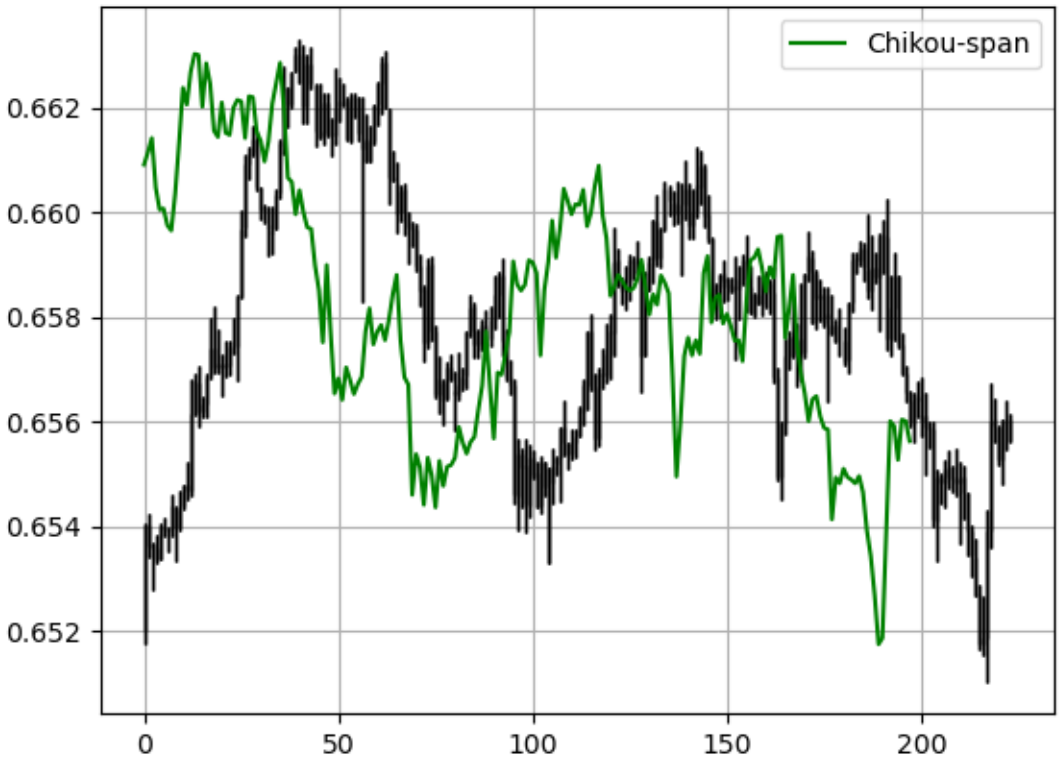



*Figure 33 - Hourly NZDCHF values with the Chikou-span.*

We have another strategy that can be constructed using the three described lines. the Kijun-sen, the Tenkan-sen, and the Chikou-span lines.

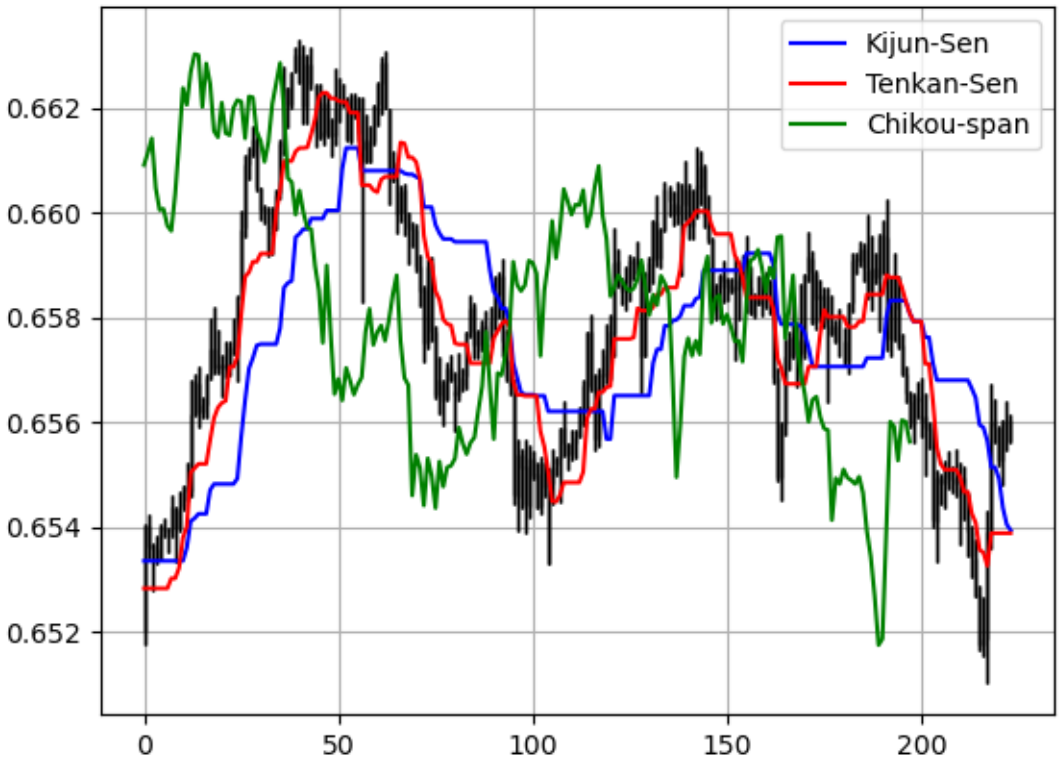



*Figure 34 - Hourly NZDCHF values with the three lines.*

The strategy is similar to the one presented earlier but it adds a confirmation factor, that is, the Chikou-span. Here are the trading rules:

- **A long (Buy) signal is generated whenever the Tenkan-sen crosses over the Kijun-sen. At the same time, the Chikou-span must be above the market price 26 periods ago.**
- **A short (Sell) signal is generated whenever the Tenkan-sen crosses below the Kijun-sen. At the same time, the Chikou-span must be below the market price 26 periods ago.**

In the beautiful Japanese language, a cloud is referred to as Kumo, just as this part of the system. The true innovation of the whole Ichimoku system is the cloud. It is composed of two lines or envelopes and the cloud is the area between them. We will use that area as a support and resistance zone and the usefulness here lies in the fact that it is plotted 26 periods ahead (as opposed to the Chikou-span which is plotted 26 periods in the past). The two lines that form the cloud are called Senkou span A and Senkou span B. The formulas for both lines are simple:

$$Senkou\ span\ A = \frac{Tenkan\ sen + Kijun\ sen}{2}$$

$$Senkou\ span\ B = \frac{Highest\ high + Lowest\ low}{2}$$

For the Senkou span B, the highest high and lowest low are calculated for the past 52 periods and just as the Senkou span A, it is projected 26 days into the future, so that the Ichimoku cloud is always leading the prices by 26 periods (or days if you are using a daily chart). Larger volatility forms larger (denser) clouds, and they tend to be stronger when they are large as opposed to being thin. The trend is considered bullish when Senkou span A is above Senkou span B and the opposite is true.

The angle is also helpful with determining the strength of the cloud. Steeper angles reinforce the conviction. Some rules of thumb regarding the Ichimoku trading system are that it is not recommended to initiate shorts above the cloud and to initiate longs below the cloud. When a breakout occurs with a thick cloud, it can be inferred that the trend has reversed. Another way of

determining the reversal or stability of the trend is by monitoring the levels of the Senkou span A and Senkou span B, if they switch (which is 26 periods ahead), a new trend may commence. Now it is time to back-test the Ichimoku Kinko Hyo system.

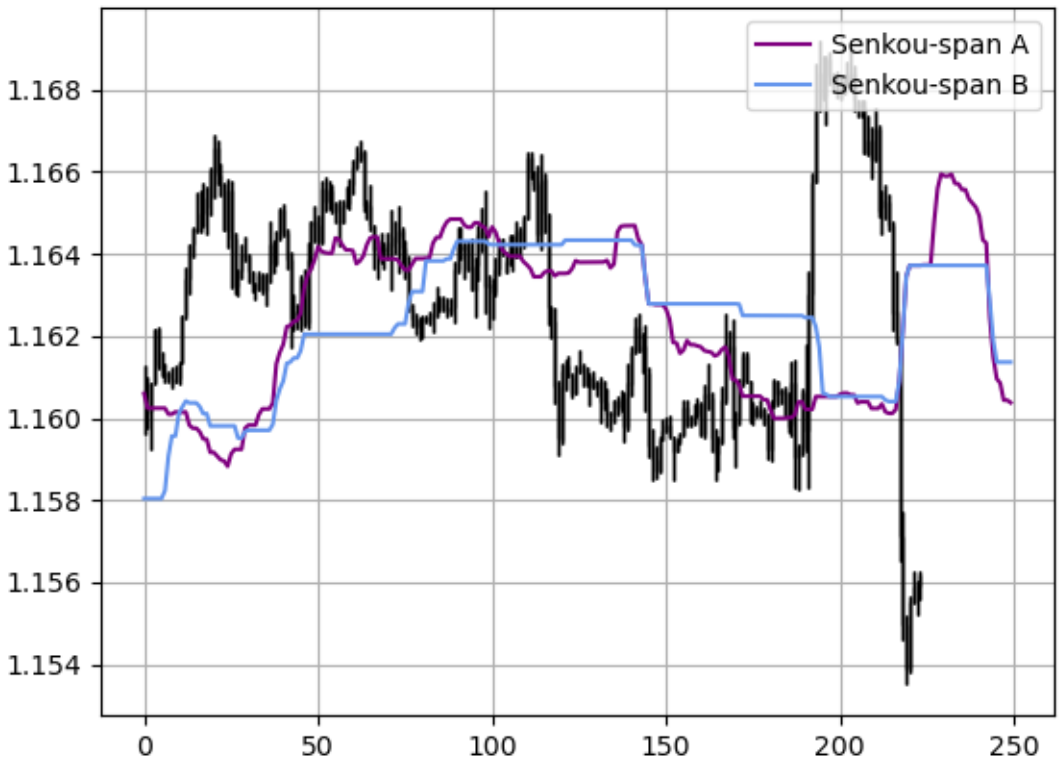



*Figure 35 - Hourly NZDCHF values with the Ichimoku cloud.*

We have already said that Ichimoku Kinko Hyo is a full system composed of mini-indicators, and so, to get a strong signal from it, we need to find a situation where all the indicators confirm each other so that we feel comfortable entering the trade. Teamwork is essential here. But first, here is the full Ichimoku function in Python.

```
def ichimoku(Data, close, high, low, kijun_lookback,
             tenkan_lookback,
             chikou_lookback,
             senkou_span_projection,
             senkou_span_b_lookback,
             where):
    Data = adder(Data, 3)
    # Kijun-sen
    for i in range(len(Data)):
        try:
            Data[i, where] = max(Data[i - kijun_lookback:i + 1, high]) + min(Data[i - kijun_lookback:i + 1, low])
        except ValueError:
            pass
    Data[:, where] = Data[:, where] / 2
    # Tenkan-sen
    for i in range(len(Data)):
        try:
            Data[i, where + 1] = max(Data[i - tenkan_lookback:i + 1, high]) + min(Data[i - tenkan_lookback:i + 1, low])
        except ValueError:
            pass
    Data[:, where + 1] = Data[:, where + 1] / 2
    # Senkou-span A
    senkou_span_a = (Data[:, where] + Data[:, where + 1]) / 2
    senkou_span_a = np.reshape(senkou_span_a, (-1, 1))
    # Senkou-span B
    for i in range(len(Data)):
        try:
            Data[i, where + 2] = max(Data[i - senkou_span_b_lookback:i + 1, high]) + min(Data[i - senkou_span_b_lookback:i + 1, low])
```

```
    except ValueError:
        pass
Data[:, where + 2] = Data[:, where + 2] / 2
senkou_span_b = Data[:, where + 2]
senkou_span_b = np.reshape(senkou_span_b, (-1, 1))
kumo = np.concatenate((senkou_span_a, senkou_span_b), axis = 1)
Data = deleter(Data, where + 2, 1)
Data = np.concatenate((Data, kumo), axis = 1)
Data = Data[senkou_span_b_lookback:, ]
for i in range (1, 7):
    new_array = shift(Data[:, 0], -senkou_span_projection, cval = 0)
    new_array = np.reshape(new_array, (-1, 1))
    Data = np.concatenate((Data, new_array), axis = 1)
    Data = deleter(Data, 0, 1)
kumo = Data[:, 0:2]
Data = deleter(Data, 0, 2)
Data = np.concatenate((Data, kumo), axis = 1)
Data = adder(Data, 1)
for i in range(len(Data)):
    try:
        Data[i, 8] = Data[i + chikou_lookback, 3]
    except IndexError:
        pass
Data[-senkou_span_projection:, 0] = Data[-senkou_span_projection:, 0] / 0
Data[-senkou_span_projection:, 1] = Data[-senkou_span_projection:, 1] / 0
Data[-senkou_span_projection:, 2] = Data[-senkou_span_projection:, 2] / 0
Data[-senkou_span_projection:, 3] = Data[-senkou_span_projection:, 3] / 0
Data[-senkou_span_projection:, 4] = Data[-senkou_span_projection:, 4] / 0
Data[-senkou_span_projection:, 5] = Data[-senkou_span_projection:, 5] / 0
Data[-52:, 8] = Data[-52:, 8] / 0
return Data
```

If we have a short signal coming from a cross between the Tenkan-sen and the Kijun-sen and prices are above the cloud, then we should not act on it because the trend is still bullish. Also, we have to be careful that the Kijun-sen is not flat when searching for trade opportunities. Although lagging in nature, we can confirm a reversal of trend if the Chikou-span crosses above prices even though 26 periods ago. Here are the conditions of a trade based on the Ichimoku system:

- **A long (Buy) signal is generated whenever the Tenkan-sen crosses the Kijun-sen from the below to the above while the market price is above the Ichimoku cloud. And finally, the Chikou span's last value must be higher than the corresponding market price in the same point in time.**
- **A sell (Short) signal is generated whenever the Tenkan-sen crosses the Kijun-sen from the above to the below while the market price is below the Ichimoku cloud. And finally, the Chikou span's last value must be lower than the corresponding market price in the same point in time.**

Another strategy that relies purely on the cloud is by expecting reactions whenever the market price enters it. We should buy whenever the market is approaching the cloud from the above in case it is dense enough, but this is a subjective strategy and sometimes the cloud can get so big that the risk-reward ratio becomes flawed. Let us visualize the Ichimoku system and try to interpret the recent values.

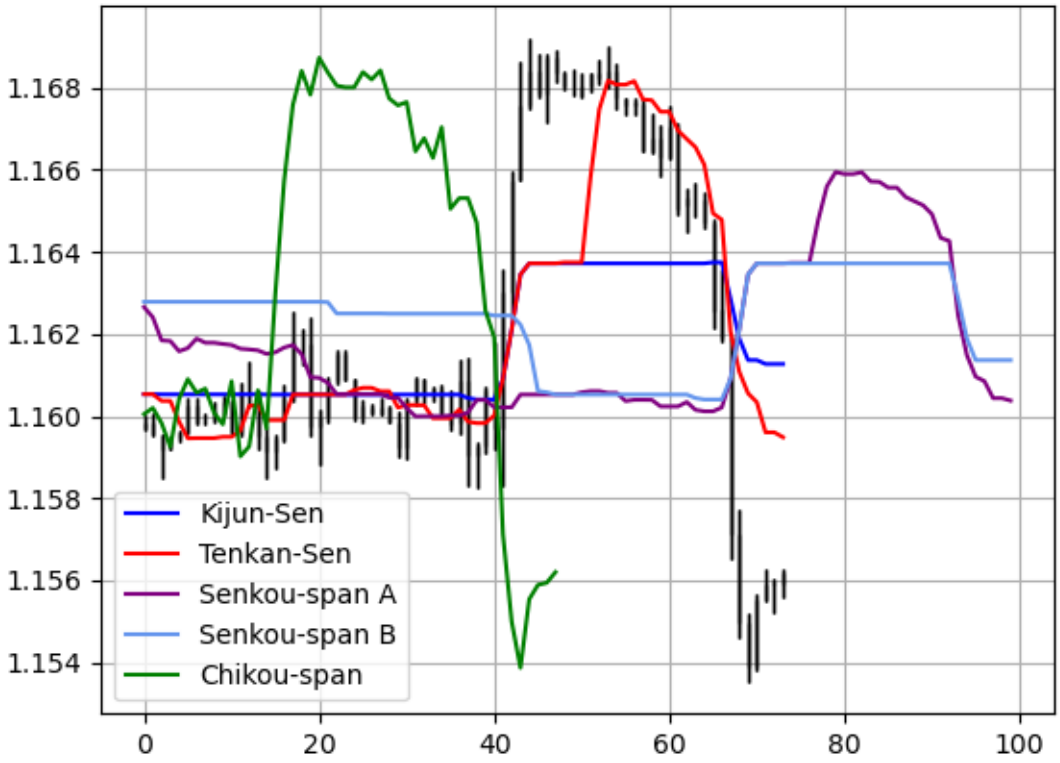



*Figure 36 - Hourly EURUSD values with the Ichimoku system.*

The above chart shows the EURUSD breaking the cloud and all three lines arguing that a planetary alignment has occurred for a short position closer to the red and blue lines for an enhanced risk-reward ratio.

# CHAPTER 14 THE TREND INTENSITY INDEX

The trend intensity index is a measure of the strength of the trend. It is a relatively simple calculation to make. The indicator is created out of a 10-period moving average and price deviations around it, then, we will count the number of the up periods relative to the total number of periods. Let us build the indicator step-by-step:

- Define the lookback period which is set at 10 and calculate the moving average on the market price using this lookback period.
- Calculate the deviations of the market price from the moving average. This is done by doing so on two columns. If the current market price is greater than its 10-period moving average, then the first column will be filled by the difference between the two (Market price minus moving average). If the current market price is lower than its 10-period moving average, then the second column will be filled by the difference between the two (Moving average minus market price).
- Now, we want to count the values where the market was above its 10-period moving average and where it was below it. This can be done using the numpy function count_nonzero():

And finally, we can calculate the Trend Intensity Index using the below formula with the Python code below it:

$$Trend\ Intensity\ Indicator = \left(\frac{Total\ Up}{(Total\ Up + Total\ Down)}\right) x\ 100$$

The code summed up in a function can be written down as follows.

```
def trend_intensity_index(Data, lookback, what, where):
    # Calculating the Moving Average
    Data = ma(Data, lookback, what, where)
    # Deviations
    for i in range(len(Data)):
        if Data[i, what] > Data[i, where]:
            Data[i, where + 1] = Data[i, what] - Data[i, where]
        if Data[i, what] < Data[i, where]:
            Data[i, where + 2] = Data[i, where] - Data[i, what]
    # Trend Intensity Index
    for i in range(len(Data)):
        Data[i, where + 3] = np.count_nonzero(Data[i - lookback + 1:i + 1, where + 1])
    for i in range(len(Data)):
        Data[i, where + 4] = np.count_nonzero(Data[i - lookback + 1:i + 1, where + 2])
    for i in range(len(Data)):
        Data[i, where + 5] = ((Data[i, where + 3]) / (Data[i, where + 3] + Data[i, where + 4])) * 100
    Data = deleter(Data, where, 5)
    return Data
```

*Figure 37 - Hourly EURUSD values with the 10-hour trend intensity index.*

Two strategies are to be used with this indicator, the first one relies on the confirmation of the new trend and has the following rules:

- **A long (Buy) signal is generated whenever the 10-period index surpasses 20 in a confirmation of the emerging bullish trend.**
- **A short (Sell) signal is generated whenever the 10-period index breaks 80 in a confirmation of the emerging bearish trend.**

The second strategy is very similar to the first one and is basically a confirmation strategy relying on the surpass or break of the neutrality level 50 to generate the signals.

It is worth noting that there is a third strategy to be used with the trend intensity index, but it is more contrarian in nature and relies on reaching the extreme levels using high lookback periods. For instance, we generally have a bullish signal whenever the 20-period trend intensity index reaches 5 or below.

# CHAPTER 15 THE SUPERTREND INDICATOR

The first concept we should understand before creating the SuperTrend indicator is volatility. We sometimes measure volatility using the Average True Range.

Although the ATR is considered a lagging indicator, it gives some insights as to where volatility is right now and where has it been last period (day, week, month, etc.). But before that, we should understand how the True Range is calculated (the ATR is just the average of that calculation). The true range is simply the greatest of the three price differences:

- High — Low
- | High — Previous close |
- | Previous close — Low |

Once we have gotten the maximum out of the above three, we simply take an average of n periods of the true ranges to get the average true range.

Generally, since in periods of panic and price depreciation we see volatility go up, the ATR will most likely trend higher during these periods, similarly in times of steady uptrends or downtrends, the ATR will tend to go lower. One should always remember that this indicator is lagging and therefore must be used with extreme caution.

The next page shows the function that calculates the ATR. Make sure you have an OHLC array of historical data.

```
def atr(Data, lookback, high, low, close, where, genre = 'Smoothed'):
    Data = adder(Data, 2)
    # True Range Calculation
    for i in range(len(Data)):
        try:
            Data[i, where] =   max(Data[i, high] - Data[i, low],
                                   abs(Data[i, high] - Data[i - 1, close]),
                                   abs(Data[i, low]  - Data[i - 1, close]))
        except ValueError:
            pass
    Data[0, where] = 0
    # Average True Range Calculation
    Data = ema(Data, 2, (lookback * 2) - 1, where, where + 1)
    Data = deleter(Data, where, 1)
    Data = jump(Data, lookback)
    return Data
```

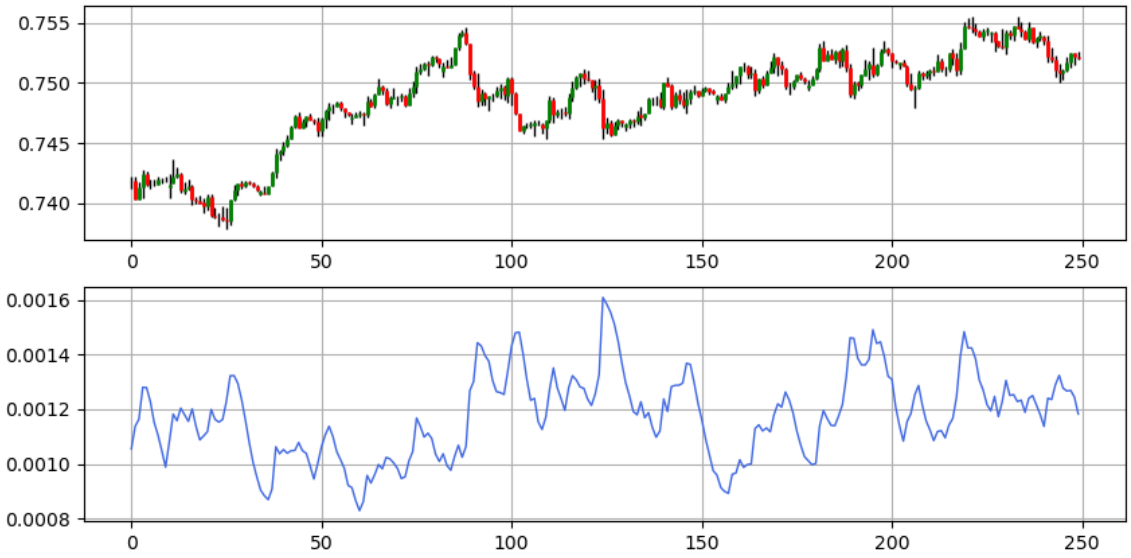



*Figure 38 - Hourly NZDUSD values with the 10-period ATR.*

Now that we have understood what the ATR is and how to calculate it, we can proceed further with the SuperTrend indicator. The indicator seeks to provide entry and exit levels for trend followers. You can think of it as a moving average or an MACD. Its uniqueness is its main advantage and although its calculation method is much more complicated than the other two indicators, it is intuitive in nature and not that hard to understand. Basically, we have two variables to choose from: The ATR lookback and the multiplier's value. The former is just the period used to calculate the ATR while the latter is generally an integer (usually 2 or 3).

$$Basic\ Upperband = \left(\frac{High + Low}{2}\right) + (multiplier\ .\ ATR)$$

$$Basic\ Lowerband = \left(\frac{High + Low}{2}\right) - (multiplier\ .\ ATR)$$

The first thing to do is to average the high and low of the price bar, then we will either add or subtract the average with the selected multiplier multiplied by the ATR as shown in the above formulas. This will give us two arrays, the basic upper band and the basic lower band which form the first building blocks in the SuperTrend indicator. The next step is to calculate the final upper band and the final lower band using the below formulas.

$$Final\ Upperband = If\ Current\ Basic\ Upperband\ < Previous\ Final\ Upperband\ OR$$
$$Previous\ Close > Previous\ Final\ Upperband\ Then$$
$$Current\ Basic\ Upperband\ Else\ Previous\ Final\ Upperband$$

$$Final\ Lowerband = If\ Current\ Basic\ Lowerband > Previous\ Final\ Lowerband\ OR$$
$$Previous\ Close < Previous\ Final\ Lowerband\ Then$$
$$Current\ Basic\ Lowerband\ Else\ Previous\ Final\ Lowerband$$

It may seem complicated but most of these conditions are repetitive and in any case, I will provide the Python code so that you can play with the function and optimize it to your trading preferences. Finally, using the previous two calculations, we can find the SuperTrend.

$$SuperTrend = If\big((Previous\ SuperTrend = Previous\ Final\ Upperband)and\ (Current\ Close <= Current\ Final\ Upperband)\big)Then\ Current\ Final\ Upperband$$

$$If\big((Previous\ SuperTrend = Previous\ Final\ Upperband)and\ (Current\ Close > Current\ Final\ Upperband)\big)\ Then\ Current\ Final\ Lowerband$$

$$Else$$

$$If\big((Previous\ SuperTrend = Previous\ Final\ Lowerband)and\ (Current\ Close >= Current\ Final\ Lowerband)\big)Then\ Current\ Final\ Lowerband$$

$$Else$$

$$If\big((Previous\ SuperTrend = Previous\ Final\ Lowerband)and\ (Current\ Close < Current\ Final\ Upperband)\big)Then\ Current\ Final\ Upperband$$

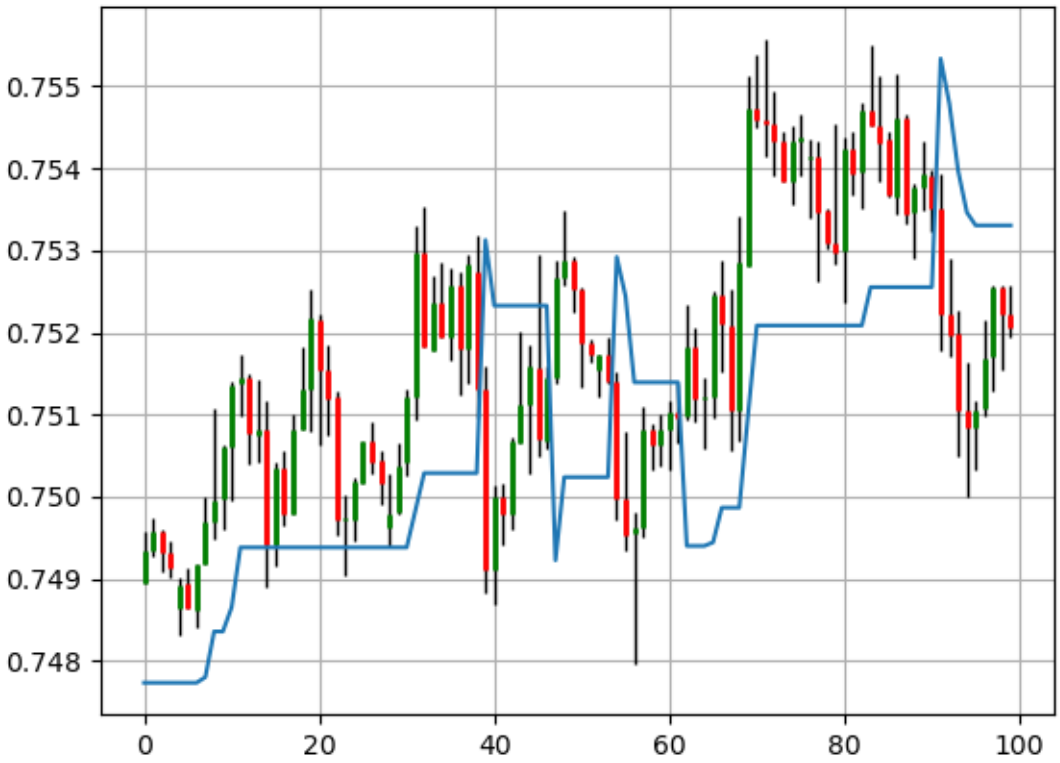



*Figure 39 - Hourly AUDUSD values with the SuperTrend.*

The strategy that relies on the SuperTrend uses the following conditions:

- **A long (Buy) signal is generated whenever the market surpasses the SuperTrend where now it becomes support to follow the rise.**
- **A short (Sell) signal is generated whenever the market breaks the SuperTrend where now it becomes resistance to follow the drop.**

I should point out that I regularly use the word “surpass” to signify a breach from the below to the above and “break” to signify a breach from the above to the below. The SuperTrend is a great tool to follow the trend and to determine the market regime. We will come back to this indicator in the fourth part to back-test its strategies.

# CHAPTER 16 THE DONCHIAN CHANNEL

Created by Richard Donchian, this simple and great indicator is used to identify breakouts and reversals. The goal is to determine objectively a range exit by the surpass or break of any of the barriers. The way it is formed is by first calculating the maximum of the last n-period highs and the minimum of the last n-period lows, then calculating the average of them both. This gives us three measures:

- The Donchian upper band as the tracking highest high of the latest n periods.
- The Donchian lower band as the tracking lowest low of the latest n periods.
- the Middle band as the average between the two bands. We will be omitting this band from the research as it does not bring much to the table.

$$Upper\ channel\ =\ Highest\ High\ in\ Last\ N\ Periods$$

$$Lower\ channel\ =\ Lowest\ Low\ in\ Last\ N\ periods$$

The Donchian channel is mainly used for breakouts, and we generally use the following conditions to generate the trades:

- **If the current close surpasses the last upper Donchian channel, a bullish breakout has occurred, and a long (Buy) signal is generated.**
- **If the current close breaks the last lower Donchian channel, a bearish breakout has occurred, and a short (Sell) signal is generated.**

```
def donchian(Data, low, high, lookback, where, median = 1):
    Data = adder(Data, 3)
    for i in range(len(Data)):
        try:
            Data[i, where] = max(Data[i - lookback:i + 1, high])
        except ValueError:
            pass
    for i in range(len(Data)):
        try:
            Data[i, where + 1] = min(Data[i - lookback:i + 1, low])
        except ValueError:
            pass
    if median == 1:
        for i in range(len(Data)):
            try:
                Data[i, where + 2] = (Data[i, where] + Data[i, where + 1]) / 2
            except ValueError:
                pass
    Data = jump(Data, lookback)
    return Data
```

Traders like this indicator because it is objective as opposed to just tracing trend lines manually to detect breakouts.

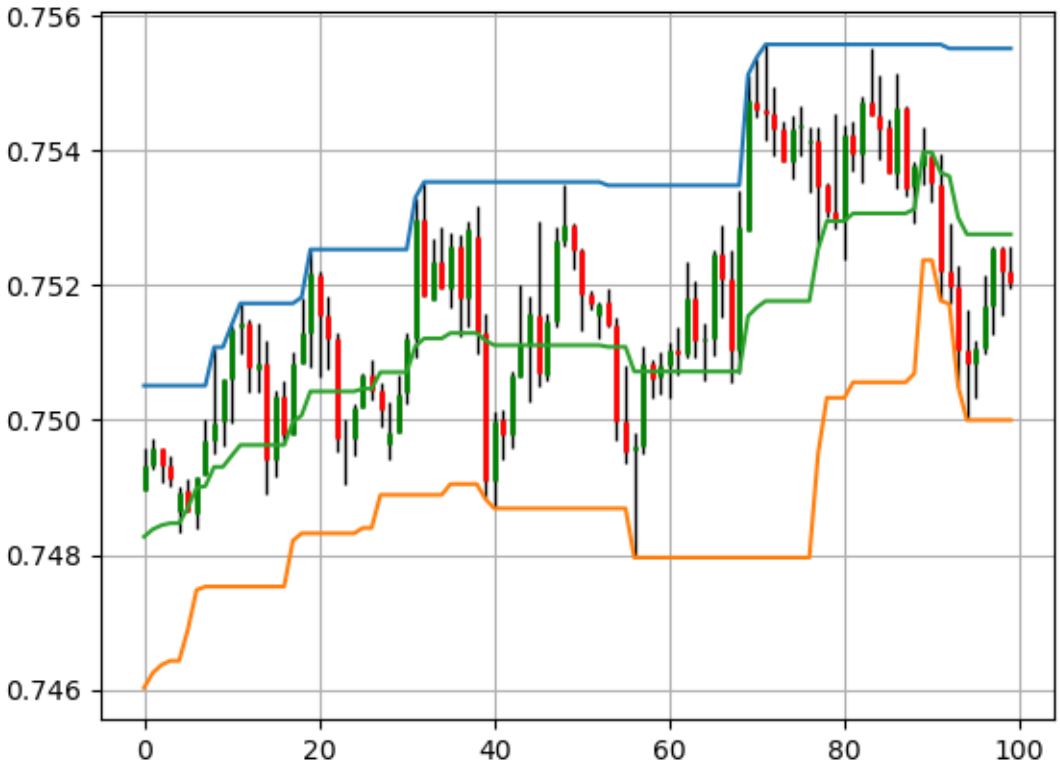



*Figure 40 - Hourly AUDUSD values with the 20-period Donchian channel.*

```
my_data = donchian(my_data, 2, 1, lookback, 4, median = 1)
ohlc_plot_candles(my_data, 100)
plt.plot(my_data[-100:, 4])
plt.plot(my_data[-100:, 5])
plt.plot(my_data[-100:, 6])
```

It is recommended to use this indicator with other technical tools so that we get an extra confirmation factor as the back-testing strategies relying solely on the Donchian channel are not profitable and this is to be expected from a simple indicator.

# CHAPTER 17 THE AVERAGE DIRECTIONAL INDEX

How do we know whether we are ranging or trending? And for that matter, how can we measure the strength of the trend? Sure, we can take a look at the chart and maybe overlay some moving averages but that still does not tell us much. Created by Welles Wilder, the average directional index — ADX is a complex indicator that is used to determine the strength or absence of a trend, be it bullish or bearish. Generally, it is accompanied by a negative directional index (-DI) and a positive directional index (+DI). We will see how to calculate and code the ADX step-by-step.

The first concept we need to understand is the positive directional movement (+DM) and the negative directional movement (-DM). These two are the building blocks of this complex indicator. The +DM is the first measure in determining the bullish trend's strength and is found using the following formula:

$$+DM = Current\ High - Previous\ High$$

With that being said, the -DM is also the first measure in determining the bearish trend's strength and below we can see its formula:

$$-DM = Previous\ Low - Current\ Low$$

After calculating the two directional movements, we have to calculate an independent indicator called the average true range — ATR, which we have seen in a previous chapter. The ATR is a gauge for volatility and is heavily used in strategies to measure risk and place stops and targets. Now, we have to calculate the smoothed moving average of the +DM and the -DM. Then, after calculating the moving average of the two, we will simply divide the result by the ATR. Hence, for every point in time we will have our recent average of the

+DM and the -DM which are both divided by the current ATR. This gives us two lines following the market price. The lines are called the positive directional index (+DI) and the negative directional index (-DI)

$$+DI = \frac{Smoothed(+DM)}{ATR}$$

$$-DI = \frac{Smoothed(-DM)}{ATR}$$

The below plot shows the EURUSD with the Positive Directional Index and the Negative Directional Index using the standard 14-period parameter. This means that the moving average has been calculated on the last 14 values including the current value.

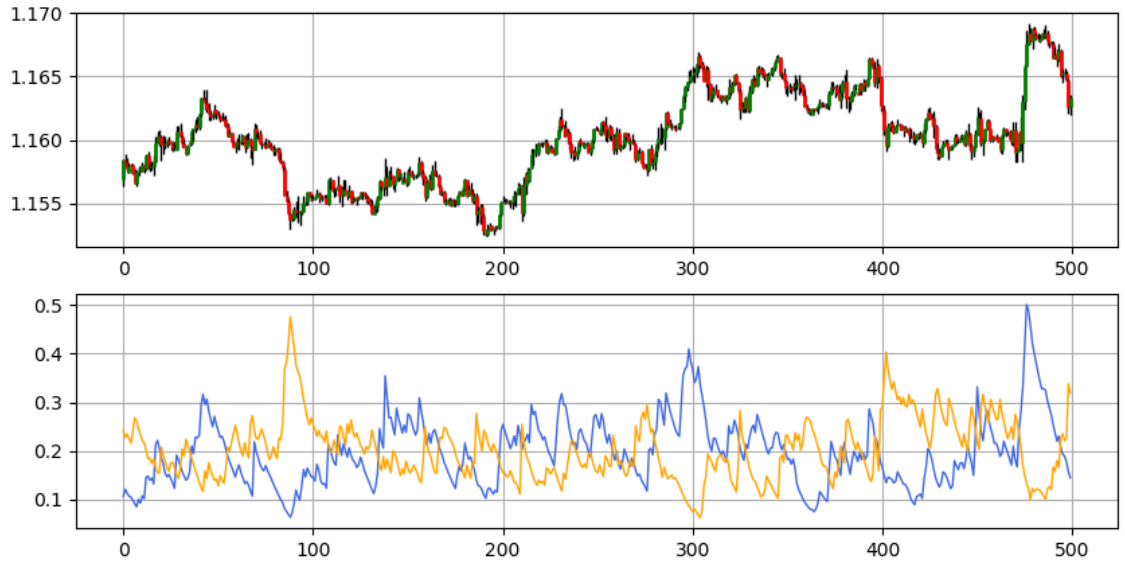



*Figure 41 – Hourly EURUSD values with the 14-period +DI and -DI..*

The final step to finding the Average Directional Index is to use this formula:

$$ADX = \left(\frac{|+DI - (-DI)|}{|+DI + (-DI)|}\right)$$

Giving us the final ADX plot that resembles something like this:

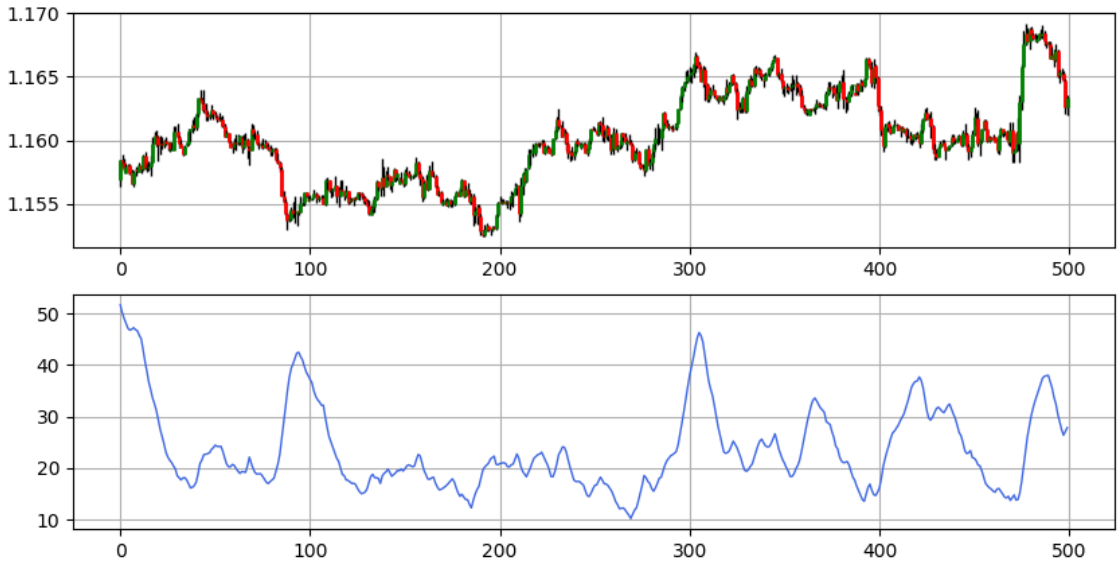



*Figure 42 - Hourly EURUSD values with the 14-period ADX.*

Hence, we can write the whole ADX in a clear and concise Python function.

```
def adx(Data, high, low, close, lookback, where):
    Data = adder(Data, 10)
    # DM+
    for i in range(len(Data)):
        if (Data[i, high] - Data[i - 1, high]) > (Data[i - 1, low] - Data[i, low]):
            Data[i, where] = Data[i, high] - Data[i - 1, high]
        else:
            Data[i, where] = 0
```

```
    # DM-
    for i in range(len(Data)):
        if (Data[i, high] - Data[i - 1, high]) < (Data[i - 1, low] - Data[i, low]):
            Data[i, where + 1] = Data[i - 1, low] - Data[i, low]
        else:
            Data[i, where + 1] = 0
    # Smoothing DI+
    Data = ema(Data, 2, (lookback * 2 - 1), where, where + 2)
    # Smoothing DI-
    Data = ema(Data, 2, (lookback * 2 - 1), where + 1, where + 3)
    # Smoothing ATR
    Data = atr(Data, (lookback * 2 - 1), high, low, close, where + 4)
    Data = jump(Data, lookback)
    # DI+
    Data[:, where + 5] = Data[:, where + 2] / Data[:, where + 4]
    # DI-
    Data[:, where + 6] = Data[:, where + 3] / Data[:, where + 4]
    # ADX
    for i in range(len(Data)):
        Data[i, where + 7] = abs(Data[i, where + 5] - Data[i, where + 6]) / abs(Data[i,
where + 5] + Data[i, where + 6]) * 100
```

The function is to be used with an OHLC array. It is commonly believed that when the ADX is showing values less than 25, the market is in sideways, and no trend is apparent. The average directional index — ADX is a good addition to our trading arsenal but only as an informational indicator not as signal generator.

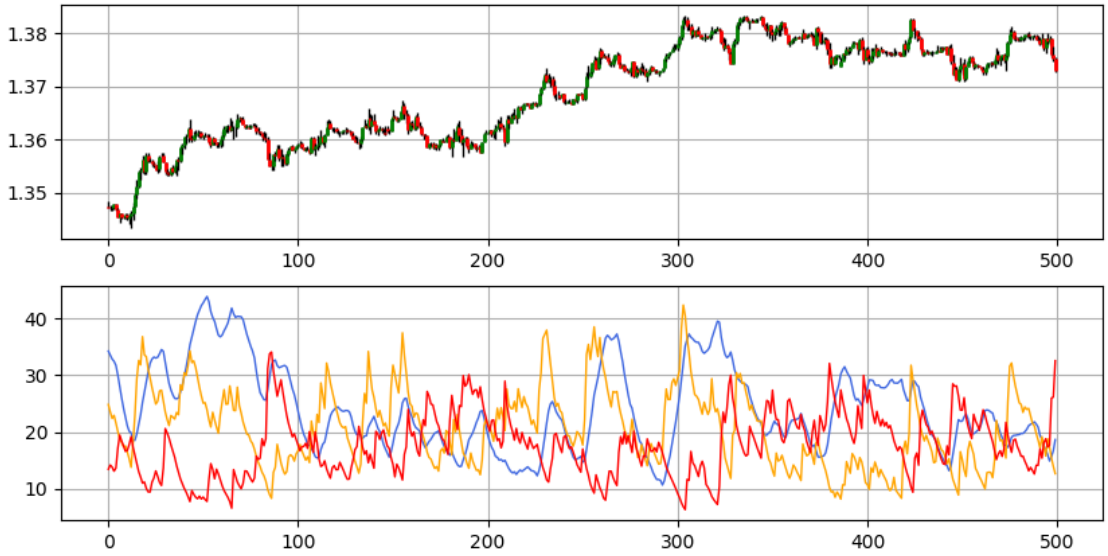



*Figure 43 – Hourly GBPUSD values with the full ADX.*

```
indicator_plot_double(my_data, 0, 1, 2, 3, 7, window = 500)
plt.plot(my_data[-500:, 4] * 100, color = 'orange', linewidth = 1)
plt.plot(my_data[-500:, 5] * 100, color = 'red', linewidth = 1)
```

The chart shows the 14-period ADX in blue with the +DI and -DI multiplied by 100 to have the same scale. It is also generally believed that a buy signal is generated whenever the +DI crosses the -DI from the below to the above and a sell signal is generated whenever the +DI crosses the -DI from the above to the below, but this has proven to be a poor strategy from personal back-tests. I recommend using this indicator in tandem with other trend-following indicators to confirm the bias.

# CHAPTER 18 THE SQUEEZE MOMENTUM INDICATOR

The squeeze momentum indicator is a volatility-momentum oscillator created by John Carter to measure and trade breakouts. It uses multiple indicators fused together to deliver buy and sell signals. This chapter will discuss how to use this amazing tool. but first, we discuss the main ingredients that will lead us to the squeeze momentum indicator.

In its most basic form, the indicator is composed of two famous technical tools:

- A 20-period Bollinger bands with a standard deviation of 2.
- A 20-period Keltner channel with a multiplier of 2.
- A custom-made momentum oscillator.

The word basic is a bit misleading I admit, as this indicator is complex and its interpretation is also not very straightforward, however, it shows decent results that exceed some classical indicators. Let us start by understanding what Bollinger bands are and what do they represent. Having defined moving averages in the previous chapters, we already have an idea on the bands. Now, with the information below, we are ready to start creating the Bollinger Bands indicator:

- Financial time series data can have a moving average that calculates a rolling mean window. For example, a 20-period moving average calculates each time a 20-period mean that refreshes each time a new bar is formed.
- On this rolling mean window, we can calculate the standard deviation of the same lookback period on the moving average.

What are Bollinger bands? When prices move, we can calculate a moving average (mean) around them so that we better understand their position

regarding their mean. By doing this, we can also calculate where do they stand statistically. Some say that the concept of volatility is the most important one in the financial markets industry. Trading volatility bands is using statistical properties to aid you in the decision-making process; hence, you know you are in good hands. The idea of Bollinger bands is to form two barriers calculated from a constant multiplied by the rolling standard deviation. They are in essence, barriers that give out a probability that the market price should be contained within them. The lower Bollinger band can be considered as a dynamic support while the upper Bollinger Band can be considered as a dynamic resistance. Hence, Bollinger bands are simply a combination of a moving lines that follow prices. To calculate the two Bands, we use the following relatively simple formulas:

$$Upper\ Band = Moving\ Average + Constant\sqrt{\frac{\sum_{i=1}^{n}(y_i - Moving\ Average)^2}{n}}$$

$$Lower\ Band = Moving\ Average - Constant\sqrt{\frac{\sum_{i=1}^{n}(y_i - Moving\ Average)^2}{n}}$$

With the constant being the number of standard deviations that we choose to envelop prices with. By default, the indicator calculates a 20-period simple moving average and two standard deviations away from the price, then plots them together to get a better understanding of any statistical extremes. This means that on any time, we can calculate the mean and standard deviations of the last 20 observations we have and then multiply the standard deviation by the constant. Finally, we can add and subtract it from the mean to find the upper and lower band. To create the Bollinger bands in Python, we need to

define the moving average function, the standard deviation (volatility) function, and then the Bollinger bands function which will use the former two functions.

```
def volatility(Data, lookback, what, where):
    Data = adder(Data, 1)
    for i in range(len(Data)):
        try:
            Data[i, where] = (Data[i - lookback + 1:i + 1, what].std())
        except IndexError:
            pass
    # Cleaning
    Data = jump(Data, lookback)
    return Data
def bollinger_bands(Data, boll_lookback, standard_distance, what, where):
    Data = adder(Data, 2)
    Data = ma(Data, boll_lookback, what, where)
    Data = volatility(Data, boll_lookback, what, where + 1)
    Data[:, where + 2] = Data[:, where] + (standard_distance * Data[:, where + 1])
    Data[:, where + 3] = Data[:, where] - (standard_distance * Data[:, where + 1])
    Data = jump(Data, boll_lookback)
    Data = deleter(Data, where, 2)
    return Data
```

Now, we can discuss the second component of the squeeze momentum indicator. The Keltner channel is a volatility-based technical indicator that resembles the Bollinger bands, only it uses an exponential moving average as the mean calculation and the average true range as a volatility proxy. Hence, here are the main two differences between the two:

- **The Bollinger bands**: A simple moving average with bands based on historical Standard Deviation.
- **The Keltner channel**: An exponential moving average with bands based on the Average True Range.

Now, we calculate the Keltner Channel using an exponential moving average with the ATR of the price. Here is the formula:

$$Upper\ Band = Exponential\ Moving\ Average + (Constant\ .ATR)$$

$$Lower\ Band = Exponential\ Moving\ Average - (Constant\ .ATR)$$

```
def keltner_channel(Data, ma_lookback, atr_lookback, multiplier, what, where):
    # Adding a few columns
    Data = adder(Data, 3)
    Data = ema(Data, 2, ma_lookback, what, where)
    Data = atr(Data, atr_lookback, 2, 1, 3, where + 1)
    Data[:, where + 2] = Data[:, where] + (Data[:, where + 1] * multiplier)
    Data[:, where + 3] = Data[:, where] - (Data[:, where + 1] * multiplier)
    Data = deleter(Data, where, 2)
    Data = jump(Data, ma_lookback)
    return Data
```

And finally, we will build the third component of the indicator. This calculation will be used to show the expected directional. This oscillator will resemble the MACD, an indicator we have seen in this part about trend-following indicators. We start by taking the average between the highest highs and lowest lows of a certain lookback period, then we apply a transformation formula before smoothing it again using a moving average.

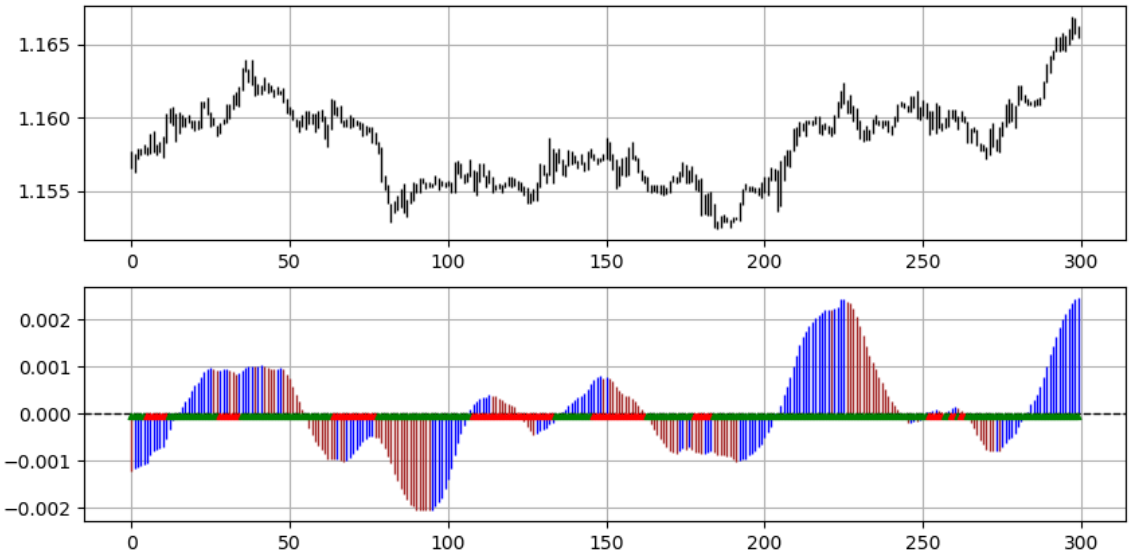



*Figure 44 - Hourly EURUSD values with the squeeze momentum indicator.*

To sum up the steps needed to create the squeeze momentum indicator are:

- Step 1: Calculate the Bollinger bands on the market price.
- Step 2: Calculate the Keltner channel on the market price.
- Step 3: Calculate the highest high in the last 20 periods.
- Step 4: Calculate the lowest low in the last 20 periods.
- Step 5: Find the mean between the two above results.
- Step 6: Calculate a 20-period simple moving average on the closing price
- Step 7: Calculate the difference between the closing price and the mean between the result from step 5 and 6.

Alternatively, we will calculate a 20-period simple moving average on the result from step 7 as opposed to the original technique where a linear regression line is drawn. This is to simplify things as the indicator is already complex as it is.

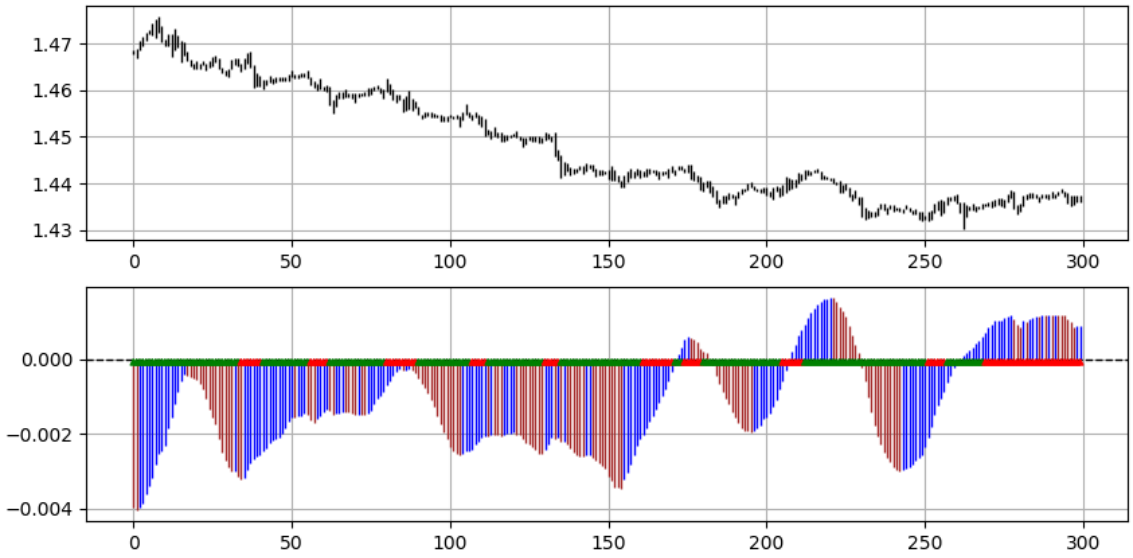



*Figure 45 - Hourly GBPUSD values with the squeeze momentum indicator.*

The next code snippet shows the full code for the Squeeze momentum indicator.

```
def squeeze(Data, boll_lookback, boll_vol, kelt_lookback, multiplier, close, where):
    Data = adder(Data, 10)
    # Adding Bollinger Bands
    Data = bollinger_bands(Data, boll_lookback, boll_vol, close, where)
    # Adding Keltner Channel
    Data = keltner_channel(Data, kelt_lookback, kelt_lookback, multiplier, close, where +
2)
    # Donchian Middle Point
    for i in range(len(Data)):
        try:
            Data[i, where + 4] = max(Data[i - boll_lookback + 1:i + 1, 1])
        except ValueError:
            pass
```

```
for i in range(len(Data)):
    try:
        Data[i, where + 5] = min(Data[i - boll_lookback + 1:i + 1, 2])
    except ValueError:
        pass
Data[:, where + 6] = (Data[:, where + 4] + Data[:, where + 5]) / 2
Data = deleter(Data, where + 4, 2)
Data = ma(Data, boll_lookback, close, where + 5)
for i in range(len(Data)):
    Data[i, where + 6] = Data[i, close] - ((Data[i, where + 4] + Data[i, where + 5]) / 2)
Data = ma(Data, boll_lookback, where + 6, where + 7)
Data = deleter(Data, where + 4, 3)
# Squeeze Detection
for i in range(len(Data)):
    if Data[i, where] < Data[i, where + 2] and Data[i, where + 1] > Data[i, where + 3]:
        Data[i, where + 5] = 0.001
return Data
```

The way to use the indicator can be based on the below steps:

- **Whenever the Bollinger bands are inside the Keltner channel, we say that the squeeze is on, and we are waiting for a breakout. In parallel, we must keep an eye on the oscillator. If the squeeze turns off by having any of the Bollinger bands; legs go beyond the Keltner channel while the oscillator is above zero, then, a buy signal is generated.**
- **Whenever the Bollinger Bands are inside the Keltner channel, we say that the squeeze is on, and we are waiting for a breakout. In parallel, we must keep an eye on the oscillator. If the squeeze turns off by having any of the Bollinger Bands; legs go beyond the Keltner channel while the oscillator is below zero, then, a sell signal is generated.**

# CHAPTER 19 THE AROON OSCILLATOR

The Aroon Oscillator is created from two sub-indicators called Aroon Up and Aroon Down. They measure the trend's strength using the highs and lows in a way that is unique and unlike other regular indicators. To calculate the Aroon Up and Down, we use the following formula:

$$Aroon\ Up = 100\ x\ \left(\frac{(25 - Periods\ Since\ Highest\ High\ of\ 25\ Periods\ Ago)}{25}\right)$$

$$Aroon\ Down = 100\ x\ \left(\frac{(25 - Periods\ Since\ Lowest\ Low\ of\ 25\ Periods\ Ago)}{25}\right)$$

Let us look at an example on what is meant by Aroon Up and Aroon Down. Consider an asset where it has reached a new low last time period. For a 25-period Aroon Down, the value will be 96 because we will be subtracting 1 from 25, dividing it by 25 and multiplying by 100. Similarly, for a 13-period Aroon Up, where an asset has reached a new high one period ago, the Aroon Up will show a reading of 92.30. It is understood that the higher the value of the Aroon, the stronger the trend. Now, let us code the Aroon Oscillator.

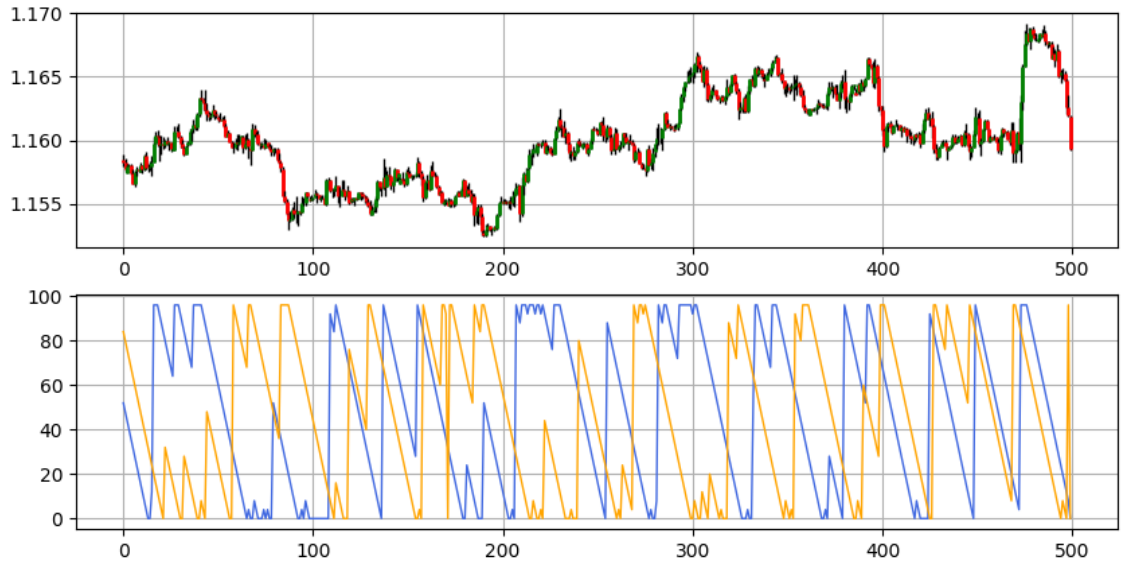



*Figure 46 Hourly EURUSD values with the Aroon indicator.*

```
def aroon(Data, period, close, where):
    # Adding Columns
    Data = adder(Data, 10)
    # Max Highs
    for i in range(len(Data)):
        try:
            Data[i, where] = max(Data[i - period + 1:i + 1, 1])
        except ValueError:
            pass
    # Max Lows
    for i in range(len(Data)):
        try:
            Data[i, where + 1] = min(Data[i - period + 1:i + 1, 2])
        except ValueError:
            pass
    # Where the High Equals the Highest High in the period
    for i in range(len(Data)):
        if Data[i, 1] == Data[i, where]:
            Data[i, where + 2] = 1
    # Where the Low Equals the Lowest Low in the period
    for i in range(len(Data)):
        if Data[i, 2] == Data[i, where + 1]:
            Data[i, where + 3] = 1
    # Jumping Rows
    Data = jump(Data, period)
    # Calculating Aroon Up
    for i in range(len(Data)):
        try:
```

```
        try:
            x = max(Data[i - period:i, 1])
            y = np.where(Data[i - period:i, 1] == x)
            y = np.array(y)
            distance = period - y
            Data[i - 1, where + 4] = 100 *((period - distance) / period)except ValueError:
            pass
    except IndexError:
        pass
# Calculating Aroon Down
for i in range(len(Data)):
    try:
        try:
            x = min(Data[i - period:i, 2])
            y = np.where(Data[i - period:i, 2] == x)
            y = np.array(y)
            distance = period - y
            Data[i - 1, where + 5] = 100 *((period - distance) / period)except ValueError:
            pass
    except IndexError:
        pass
# Cleaning
Data = deleter(Data, 4, 4)
return Data
```

Created by Tushar Chande, the Aroon Oscillator is a popular trend-following indicator which is calculated as seen in the above code snippet. It is worth knowing that the word Aroon most likely means dawn's early light in Sanskrit.

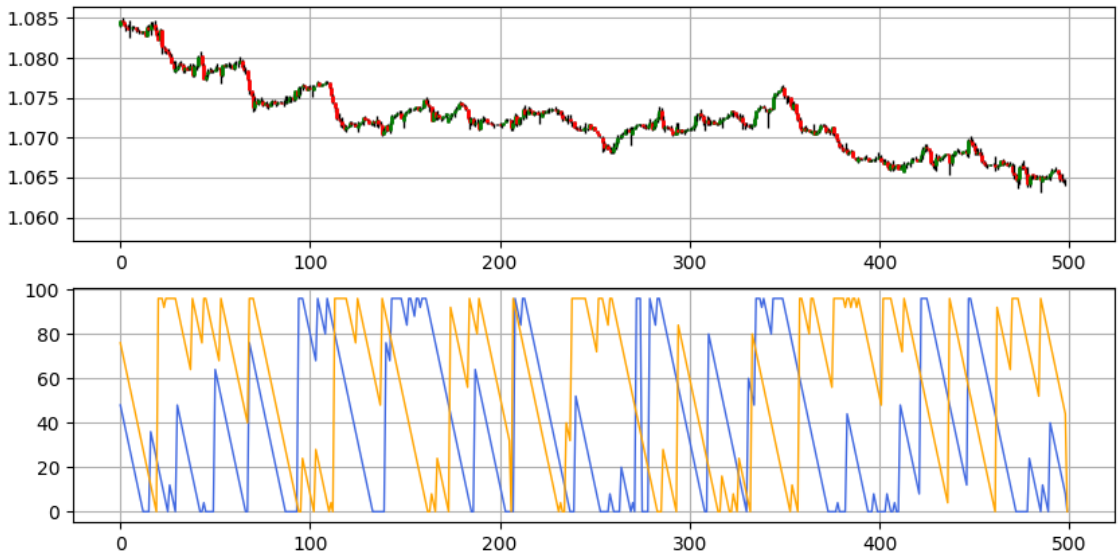



*Figure 47 - Hourly EURUSD values with the Aroon indicator.*

The way to trade the Aroon Oscillator is to wait for the crossovers between the two sub-oscillators:

- **A long (Buy) signal is generated whenever the Aroon Up crosses over the Aroon Down.**
- **A short (Sell) signal is generated whenever the Aroon Up crosses under the Aroon Down.**

I highly discourage you from using the Aroon indicator in solo mode as it is not very useful alone.

# CHAPTER 20 THE PARABOLIC STOP-AND-REVERSE

The Parabolic Stop-and-Reverse is an interesting trend-following indicator created by Welles Wilder. This indicator is mostly used as a trailing stop that tracks the trend as it develops but there is no harm in testing it as a trading strategy. It is worth noting that it performs relatively well in steady trends but just as any other indicator, it has its weakness, in this case, ranging markets.

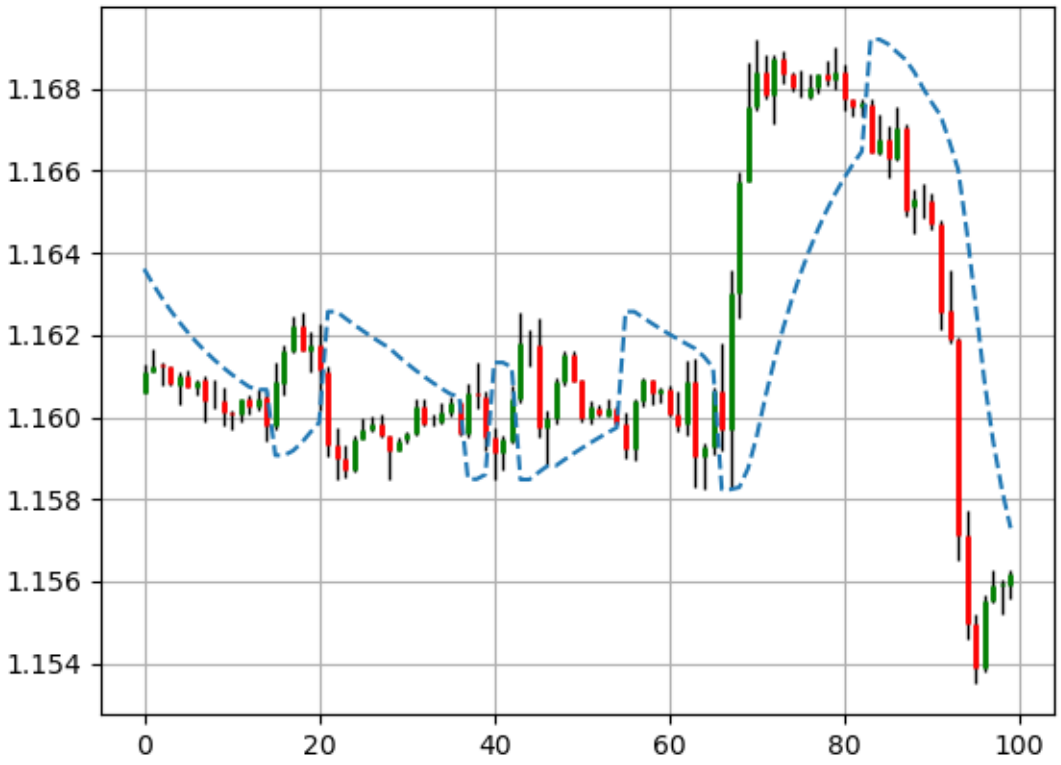



*Figure 48 - Hourly EURUSD values with the Parabolic SAR.*

The basic understanding is that when the Parabolic SAR (the dashes around the market price) is under the current price, then the outlook is bullish and when it is above the current price, then the outlook is bearish.

I will refer to a python library called talib from where the user could import the sar function that uses a data frame and calculates the values. Having modified that function, you can refer to the one below (I do not take credit for it as I merely just changed some lines as opposed to the other functions which have been coded by me):

```
def sar(s, af = 0.02, amax = 0.2):
    high, low = s.high, s.low
    sig0, xpt0, af0 = True, high[0], af
    sar = [low[0] - (high - low).std()]
    for i in range(1, len(s)):
        sig1, xpt1, af1 = sig0, xpt0, af0
        lmin = min(low[i - 1], low[i])
        lmax = max(high[i - 1], high[i])
        if sig1:
            sig0 = low[i] > sar[-1]
            xpt0 = max(lmax, xpt1)
        else:
            sig0 = high[i] >= sar[-1]
            xpt0 = min(lmin, xpt1)
        if sig0 == sig1:
            sari = sar[-1] + (xpt1 - sar[-1])*af1
            af0 = min(amax, af1 + af)
            if sig0:
                af0 = af0 if xpt0 > xpt1 else af1
                sari = min(sari, lmin)
```

```
        else:
            af0 = af0 if xpt0 < xpt1 else af1
            sari = max(sari, lmax)
        else:
            af0 = af
            sari = xpt0
        sar.append(sari)
    return sar
```

To add the Parabolic SAR to your OHLC array, use the following steps:

```
# If you have an array
my_data = pd.DataFrame(my_data)
# Renaming the columns to fit the function
my_data.columns = ['open','high','low','close']
# Calculating the Parabolic SAR
Parabolic = sar(my_data, 0.01, 0.2)
# Converting the Parabolic values back to an array
Parabolic = np.array(Parabolic)
# Reshaping
Parabolic = np.reshape(Parabolic, (-1, 1))
# Concatenating with the OHLC Data
my_data = np.concatenate((my_data, Parabolic), axis = 1)
```

The basic strategy on the Parabolic SAR is the same one used on the SuperTrend indicator where we buy whenever the market surpasses the dashes and sell short whenever the market breaks the dashes.

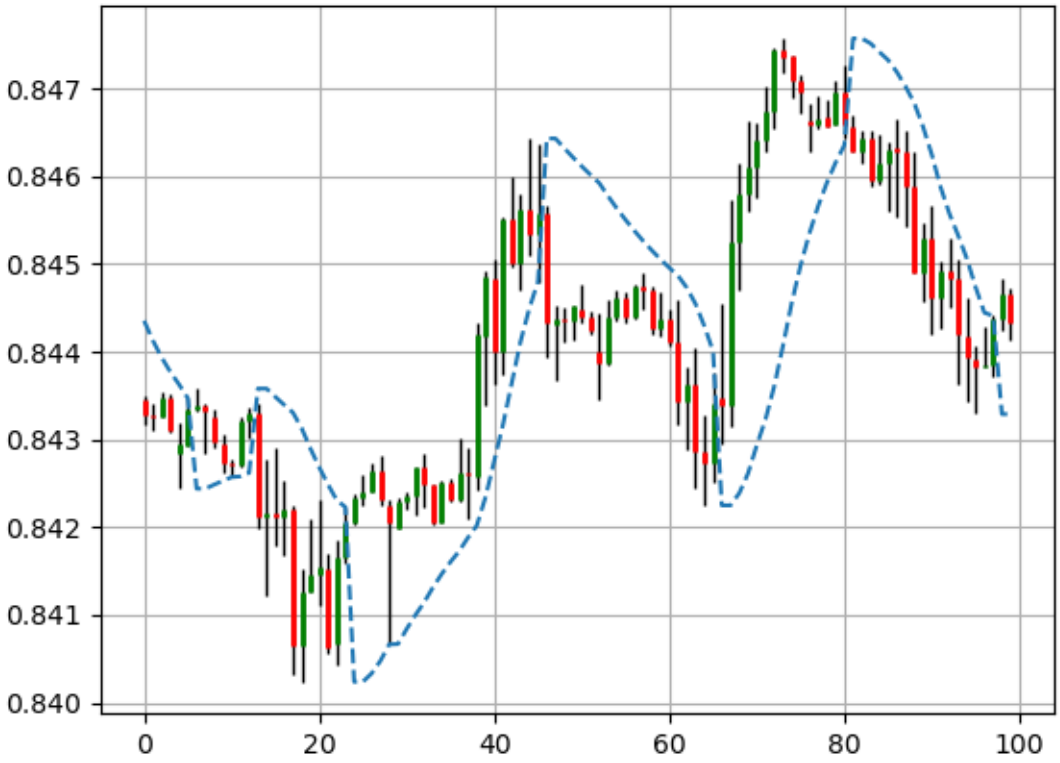



*Figure 49 - Hourly EURGBP values with the Parabolic SAR.*

# CHAPTER 21 THE VERTICAL HORIZONTAL FILTER

A huge chunk of the trading time is spent searching for the answer to this famous question: "What will the market do next? Range? Trend?". The answer to this question is a form of holy grail because if we know what will happen then we would be able to choose the right strategy.

Many good strategies fail at some point because the market regime has switched and is no longer representative. Imagine a hypothetical scenario where we know the market will continue to move sideways. A strategy relying on buying the dips and selling the highs should do well, however, the moment the market decides to go in one direction, the strategy will underperform. The Vertical Horizontal Filter was introduced by Adam White and is a measure how strong or weak the markets are trending. This is to be able to choose the appropriate strategy. It is calculated using the following steps:

- Calculate the absolute sum of price differences from one period to another. For example, if the current price is $99 and the last price was $100, then the absolute price column should be $1.
- Calculate the maximum values in a certain lookback period and subtract it from the minimum value. This means that if the lookback period is 5, then we should look to subtract the highest close from the lowest close in that time period.
- Divide the numerator by the denominator to get the Vertical Horizontal Filter.

```
def vertical_horizontal_indicator(Data, lookback, what, where):
    # Adding a few columns
    Data = adder(Data, 10)
    for i in range(len(Data)):
        Data[i, where] = Data[i, what] - Data[i - 1, what]
    Data = jump(Data, 1)
    Data[:, where] = abs(Data[:, where])
for i in range(len(Data)):
        Data[i, where + 1] = Data[i - lookback + 1:i + 1, where].sum()
    for i in range(len(Data)):
        try:
            Data[i, where + 2] = max(Data[i - lookback + 1:i + 1, what]) - min(Data[i -
lookback + 1:i + 1, what])
        except ValueError:
            pass
    Data = jump(Data, lookback)
    Data[:, where + 3] = Data[:, where + 2] / Data[:, where + 1]
    Data = deleter(Data, where, 3)
    return Data
```

The issue however is that it is not really used for trading nor is it clear how it is used to follow the trend. The idea of the indicator is that rising values represent a trending market (Bullish or bearish) while falling values represent a ranging market, but we need to find our own thresholds and limits.

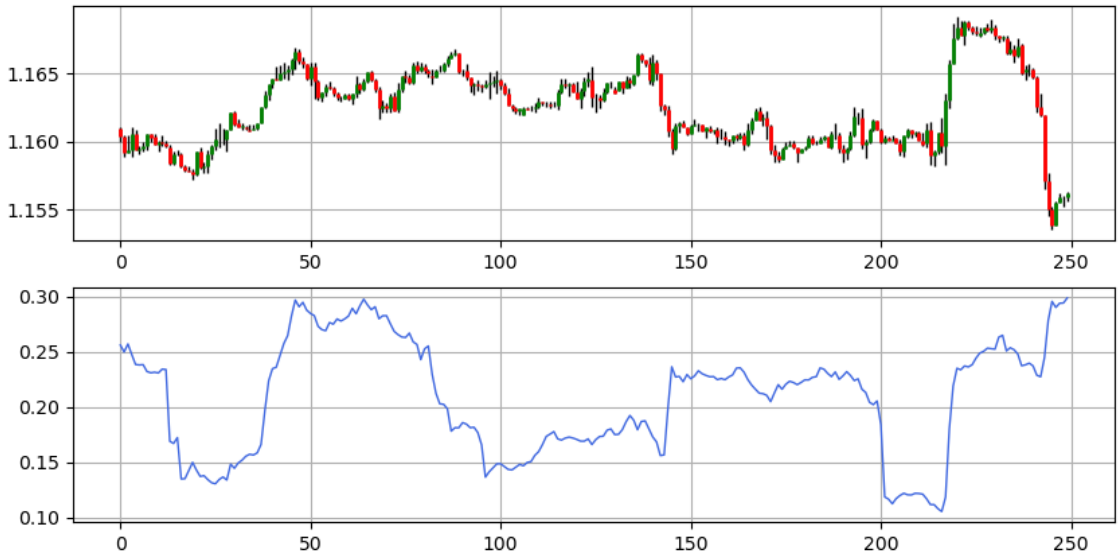



*Figure 50 - Hourly EURUSD values with the 60-hour vertical horizontal filter.*

The previous chart shows the hourly EURUSD with the 60-period Vertical Horizontal Filter. On average, we can notice that rising values of the indicator correspond to a trending market and therefore, we can probably form a type of mean that follows the indicator's values and derive hypothetical interpretations from the cross such as the below:

- Whenever the indicator is below its moving average, the market's regime is unclear / ranging.
- Whenever the indicator is above its moving average, the market's regime is trending.

The next chart features the addition of a 60-period moving average. The trading conditions are getting clearer now. Whenever the filter is above its moving average, a trending market is in progress and whenever the filter is below its moving average, a ranging market is in progress.

*Figure 51 - Hourly EURUSD values with the 60-hour vertical horizontal filter.*

Let us try to formulate a simple trading strategy using the filter.

- **Whenever the 60-hour vertical horizontal filter is above its 60-hour moving average and the MACD oscillator is above zero, a long (Buy) signal is generated.**
- **Whenever the 60-hour vertical horizontal filter is below its 60-hour moving average and the MACD oscillator is below zero, a short (Sell) signal is generated.**

Creativity is encouraged with such indicators. Technical analysis is like cooking, we need ingredients that match together to be able to combine them.

# CHAPTER 22 THE TRIX INDICATOR

A famous technical indicator called the TRIX is often mentioned by experienced and novice traders. It belongs to the family of momentum indicators. The TRIX stands for the Triple Exponential Moving Average and is a simple indicator that can be easily coded interpreted. We have already seen what an exponential moving average is, so now, we will use it as the main ingredient to create the protagonist of this chapter. The TRIX is a momentum indicator like its peers MACD and the awesome oscillator. It is composed of a bundle of exponential moving averages. Developed by Jack Hutson, the TRIX indicator can be considered a smooth version of the MACD. To create the indicator, we can follow the following steps:

- Calculate the exponential moving average of the market price using a selected lookback period.
- Calculate another exponential moving average on the first average using the same lookback period.
- Calculate another exponential moving average on the second average using the same lookback period.
- Divide the last exponential moving average by the same one 1 period ago and subtract one. This gives us the TRIX value.

Hence, the TRIX is the rate of return of the third exponential moving average.

$$TRIX_n = \frac{Last\ EMA_n}{Last\ EMA_{n-1}} - 1$$

We have already seen how to code the exponential moving average. The below code snippet shows the function of the TRIX.

```
def trix(Data, lookback, what, where):
    # First EMA
    Data = ema(Data, 2, lookback, what, where)
    Data = jump(Data, lookback)
    # Second EMA
    Data = ema(Data, 2, lookback, where, where + 1)
    Data = jump(Data, lookback)
    # Third EMA
    Data = ema(Data, 2, lookback, where + 1, where + 2)
    Data = jump(Data, lookback)
    # TRIX
    for i in range(len(Data)):
        Data[i, where + 3] = (Data[i, where + 2] / Data[i - 1, where + 2]) - 1
    Data = deleter(Data, where, 3)
    return Data
```

*Figure 52 - Hourly USDCAD values with the TRIX.*

The TRIX Indicator can be used just as we use the MACD. This means that two main strategies can already be established:

- **The crossover strategy**: Whenever the TRIX switches from negative to positive, it can be a signal that the trend is changing from bearish to bullish and whenever the TRIX switches from positive to negative, it can be a signal that the trend is changing from bullish to bearish.
- **The divergence strategy**: Whenever the indicator is making higher lows while the market price is making lower lows, a bullish divergence can be detected and this generally signals that the trend may be switching from bearish to bullish and whenever the indicator is making lower highs while the market price is making higher highs, a bearish divergence can be detected, and this generally signals that the trend may be switching from bullish to bearish.

I have to note that this indicator is quite lagging and must only be used as a confirmation and not really a signal generator.

# PART 3 REGIME DETECTION TECHNIQUES

The market's regime is its current state and can be divided into:

- **Bullish trend**: The market tends to make higher highs. The aggregate direction is upwards.
- **Bearish trend**: The market tends to make lower lows. The aggregate direction is downwards.
- **Sideways**: The market tends to range while remaining within established zones.

Many tools attempt to detect the trend and most of them do give the answer but we cannot really say we can predict the next state. The best way to solve this issue is to assume that the current state will continue and then trade any reactions in tandem with the direction of the trend. For example, if the EURUSD is above its moving average and is shaping higher highs, then it makes sense to wait for the dips before buying and expecting more bullish momentum. This is also known as a trend-following strategy.

Regime detection techniques can be as simple as the position of the moving average relative to the market price or can be as complex as sophisticated machine learning algorithms analyzing patterns. In this part, we will look at some regime detection techniques and see how to use them to understand the current state. Remember, regime detection has a powerful filtering purpose which is to confirm the trade idea by telling us whether the trend is on our side or not. Most of the techniques function also as technical indicators but I have found that they serve a better purpose when they are used only for regime detection. It is worth mentioning that even contrarian trade ideas can benefit from the overall trend, hence, this part can also be used for contrarian trading. This is what I like to refer to as the global framework. If we have a strategy that

buys whenever a technical indicator becomes oversold, then we can lower its lookback period so that it becomes short-term in nature and then we can follow the overall trend with a long-term moving average where we only buy whenever the first condition is satisfied and whenever the market is above its moving average, therefore, by benefitting from a contrarian signal with a trend-following framework. The strategy described will be seen in the next part and is under the label Strategy #3.

It is worth mentioning that you do not need to copy the code from the book as there is a dedicated GitHub which harbors every line of code seen in the book organized properly. Focus on understanding the concepts while moving forward.

# CHAPTER 23 THE FIBONACCI RANGE

We have already discussed the Fibonacci sequence in a previous chapter. The strength of Fibonacci-based techniques is still debated to this day, but many use it and claim that it adds value. Personally, I think its self-fulfilling prophecy properties are more than enough to make them one of the pillars of technical analysis. I also use them as lookback periods in many technical indicators. Hence, I believe they add value but is it a direct added value on profitability? That requires a complicated answer and most importantly, that is not the question to be asked here. This part discusses regime detection as opposed to signal generation.

As outlined in the introductory paragraph of this part, the market's regime can be trending upwards, downwards, or ranging. Regime detection techniques seek an answer to the following questions:

- What is the current direction of the market?
- Since when did the market start its move?
- Would mean-reversion strategies work in this market regime?
- When will this trend likely end?

Of course, having an accurate answer to each question above is easier said than done, but our job is to try to approach reality as much as we can and as realistically as possible given a certain amount of data. Generally, we say that the past cannot predict the future, but the past is most likely the best we have to predict the future.

Mainly created for short-term regime detection, the Fibonacci Range method revolves around the idea of using highs and lows combined with the Fibonacci sequence to understand the current momentum and to give an approximate idea of whether the market is trending up, down, or flat.

*Figure 53 – Hourly USDCHF values.*

How do we create the Fibonacci Range? And how is it different from other trend indicators? To answer the first question, we can follow the below steps:

- For a bullish regime bar, the current high must be greater than the low from two periods ago, three periods ago, five periods ago, eight periods ago, and thirteen periods ago.

- For a bearish regime bar, the current low must be lower than the high from two periods ago, three periods ago, five periods ago, eight periods ago, and thirteen periods ago.

For a flat regime bar, any one of the conditions mentioned above must be unsatisfied. And to answer the second question, there is basically no big difference except for the fact that the Fibonacci Range uses pure price action rather than mean values (Moving averages) or volatility (SuperTrend). Theoretically, this should reduce lag but practically, it is hard to prove. The previous chart is a color-coded candlestick graph where the values in red imply an on-going bearish regime, the values in green imply an on-going bullish regime, and the values in black imply a flat or neutral on-going regime. What we are trying to do is simply confirm our trading entries. For instance, if we have a bullish signal from our strategy, we can check the market's regime and see if it is confirming our signal or not. Notably, we must have the following alignments:

- When we have a bullish signal from our trading strategy, it is recommended to act on it only if we are in a bullish regime.
- When we have a bearish signal from our trading strategy, it is recommended to act on it only if we are in a bearish regime.

```
def fibonacci_range(Data, high, low, where):
    Data = adder(Data, 1)
    for i in range(len(Data)):
        if Data[i, high] > Data[i - 2, low] and \
            Data[i, high] > Data[i - 3, low] and \
            Data[i, high] > Data[i - 5, low] and \
            Data[i, high] > Data[i - 8, low] and \
            Data[i, high] > Data[i - 13, low]:
                Data[i, where] = 1
        elif Data[i, low] < Data[i - 2, high] and \
            Data[i, low] < Data[i - 3, high] and \
            Data[i, low] < Data[i - 5, high] and \
            Data[i, low] < Data[i - 8, high] and \
            Data[i, low] < Data[i - 13, high]:
             Data[i, where] = -1
        else:
            Data[i, where] = 0
    return Data
```

The above function shows the syntax needed to generate a column populated by values that indicate the market regime. For instance, the first loop will scan each row and detect whether the current high, represented by the letter i, is greater than the low two periods ago, the low three periods ago, the low five periods ago, the low eight periods ago, and the low thirteen periods ago. If the conditions are satisfied, a value of 1 will be inputted in the column labelled as where which can be 4 in case we have an OHLC data array composed of four columns. Similarly, the loop will continue to search for bearish and flat conditions and inputs -1 and 0 respectively. This will give us a column populated by -1's, 0's, and 1's where each number represents the implied current regime

```
def ohlc_plot_candles(Data, window):
    Chosen = Data[-window:, ]
    for i in range(len(Chosen)):
        if Chosen[i, 4] > 0:
            plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'green', linewidth = 1)
            plt.vlines(x = i, ymin = Chosen[i, 0], ymax = Chosen[i, 3], color = 'green', linewidth = 1)
        if Chosen[i, 4] < 0:
            plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'red', linewidth = 1)
            plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0], color = 'red', linewidth = 1)
        if Chosen[i, 4] == 0:
            plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'black', linewidth = 1)
            plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0], color = 'black', linewidth = 1)
    plt.grid()
```

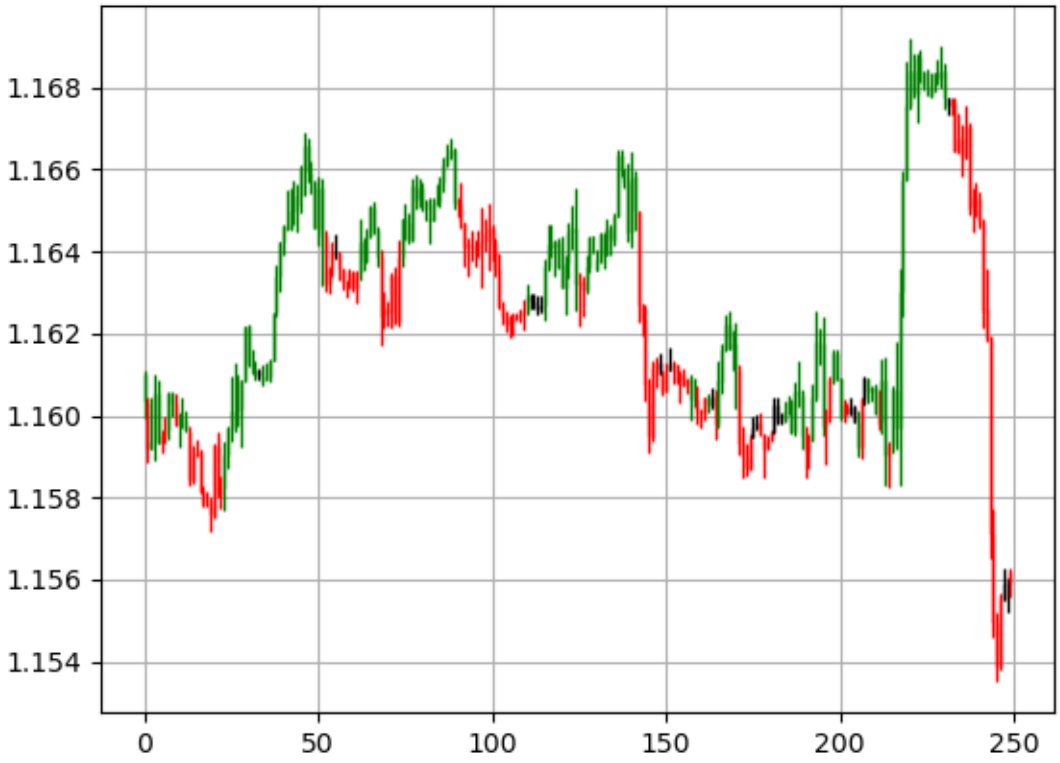



*Figure 54 – Hourly EURUSD signal.*

# CHAPTER 24 THE GOPALAKRISHNAN RANGE INDEX

Some indicators are specifically used for detecting regimes and volatility as opposed to directional indicators. As we advance in these chapters, we will encounter yet another indicator of the sort, the Gopalakrishnan range index, also known as the GRI. The index calculates the relative strength between the highs and lows divided by their lookback period.

$$GRI\ Index_n = \frac{\log\left(Highest\ High_n - Lowest\ Low_n\right)}{\log\left(n\right)}$$

If in a certain lookback period, the difference between the highest highs and the lowest lows is increasing, then the indicator will rise. If the index is showing a flat reading, it means that the market has failed to make a higher high and a lower low, therefore, the reading remains stable. Any time a new high or a new low is made, the indicator's value will increase to adjust to this, which is why it is a trending indicator.

```
def gri_index(Data, lookback, high, low, where):
  Data = adder(Data, 1)
  for i in range(len(Data)):
    try:
      Data[i, where] = abs(np.log(max(Data[i - lookback + 1:i + 1, high]) - min(Data[i - lookback
+ 1:i + 1, low])) / np.log(lookback))
    except ValueError:
      pass
  return Data
```

The above function outputs the GRI given an OHLC data array. It starts by looping around the rows, taking the highest highs of the selected window minus the lowest lows of the same window divided by the log of the window.

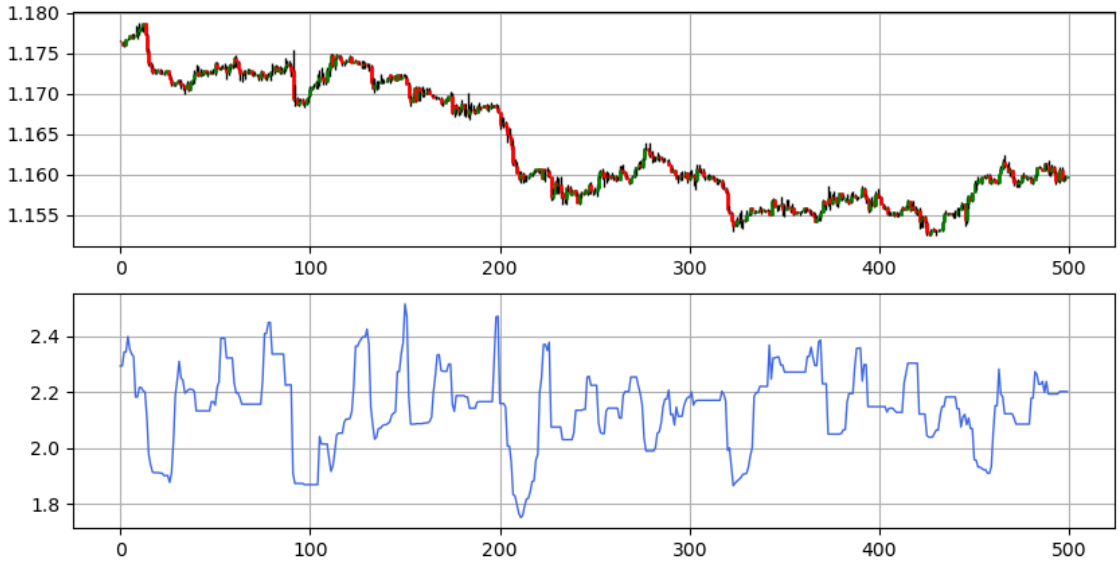



*Figure 56 – Hourly EURUSD values with the 13-period GRI.*

One limitation of the GRI is that it does not tell you the direction of the trend. You must detect it yourself using other techniques such as moving averages or even momentum indicators like the RSI.

The GRI index can be used in many trend-following strategies. Due to its nature of confirming new highs and new lows, we can list a few strategies below:

- The market is trending if the GRI is going up, therefore, we can set thresholds to surpass so that we understand when we can call the market a trending one.
- If the GRI is flat, then we can use the highest point in the market as a resistance level and the lowest point in the market as a support level.
- We can use the GRI with the MACD to confirm the trend. A suggested strategy is that whenever we see a MACD cross and a rising GRI, then we can have a better conviction.

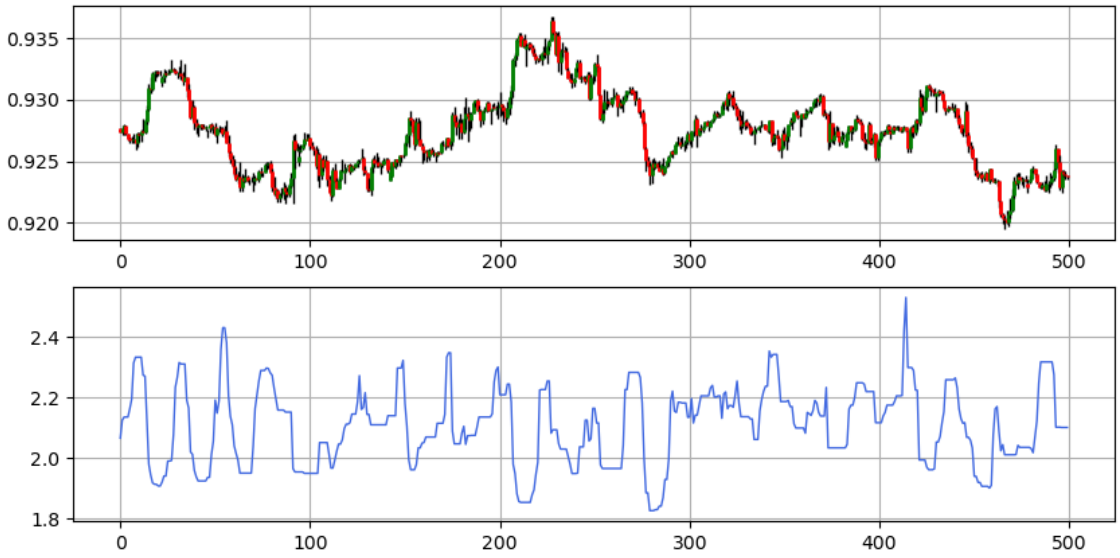



*Figure 57 - Hourly USDCHF values with the 13-period GRI.*

However, since we are in the part that discusses regimes, we can simply elaborate a way that tells us about the current market state using the GRI and a simple moving average of 30 periods. Here is the intuition of this method:

- If the GRI is flat, then, regardless of the position of the moving average relative to the current market price, the regime is flat.
- If the GRI has changed and the current market price is above the 30-period moving average, the regime is bullish.
- If the GRI has changed and the current market price is below the 30-period moving average, the regime is bearish.

When we say "The GRI has changed", we are talking about the difference between the current GRI reading and the previous reading. This is based on the idea that flat readings refer to neutral markets while readings that change (Preferably rise) refer to a trending market.

Let us code the previous three-point intuition and create a color-coded chart where the bar is painted black in case the market is flat, green in case the market is bullish, and red in case the market is bearish.

```
def ohlc_plot_candles(Data, window):
  Chosen = Data[-window:, ]
  for i in range(len(Chosen)):
    if Chosen[i, 4] == Chosen[i - 1, 4]:
      plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'black', linewidth = 1)
      plt.vlines(x = i, ymin = Chosen[i, 0], ymax = Chosen[i, 3], color = 'black', linewidth = 1.5)
    if Chosen[i, 4] != Chosen[i - 1, 4] and Chosen[i, 3] > Chosen[i, 5]:
      plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'green', linewidth = 1)
      plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0], color = 'green', linewidth = 1.5)
    if Chosen[i, 4] != Chosen[i - 1, 4] and Chosen[i, 3] < Chosen[i, 5]:
      plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'red', linewidth = 1)
      plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0], color = 'red', linewidth = 1.5)
  plt.grid()
```

The code is very simple. We start by selecting a specified window to plot and create a new array, then we loop around each row and check whether the value in column 4 hence the GRI, is equal to the previous value, print a vertical line spanning from the low to high in black. The loop then continues to search for the times where the GRI value does not equal the previous value and at the same time, the current close is above the value in column 5 which contains the moving average. In case the conditions are satisfied, the bar will be printed in green. The same reasoning goes for bearish bars. The last line of code calls the grid function which simply plots a grid to enhance the visual interpretation.

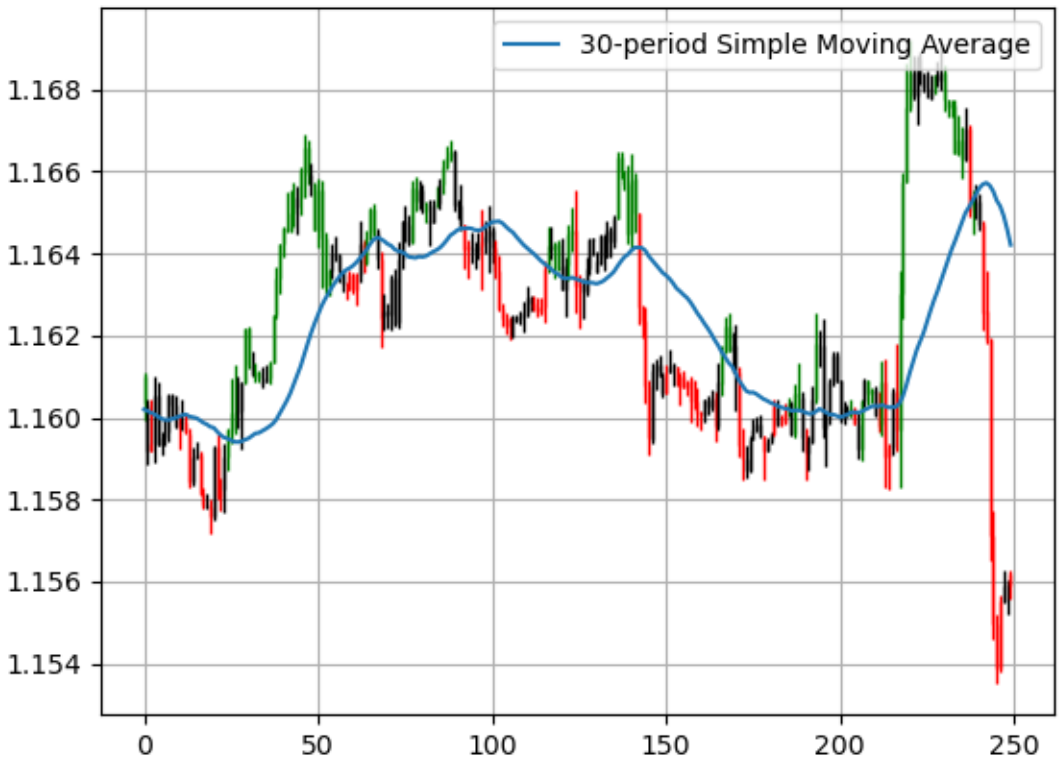



*Figure 58 – Hourly EURUSD values with the 30-hour simple moving average.*

Before we run the previous code, we need to create a column that contains the 30-period simple moving average.

```
# Calculating the GRI
my_data = gri_index(my_data, lookback, 1, 2, 4)
# Calculating the MA
my_data = ma(my_data, 30, 3, 5)
# Plotting the Color-coded chart
ohlc_plot_candles(my_data, 250)
```

The indicator is helpful whenever we want to confirm our trend-following ideas, but it must be smoothed as it looks noisy with only a 30-period moving average.

# CHAPTER 25 HEIKIN ASHI

Candlesticks are a quick way to understand the data and to detect patterns. It is very straightforward and easy to interpret. A bullish (typically green) candle is when the market closes above its opening price. A bearish (typically red) candle is when the market closes below its opening price. This is the real price data we are all aware of. Heikin-Ashi candlesticks seek to clean out the picture and show a clearer trend by smoothing out the OHLC data. Take a look at the below candlestick chart on the EURUSD. We can notice some noise between alternating colors.

*Figure 59 – Hourly EURUSD values.*

The aim is to smooth it out so that we understand the underlying momentum beneath the noise. Heikin-Ashi plotting answers this question.

We will transform the price data into a smoothed version and then chart it:

$$Close_n = \frac{Open_n + High_n + Low_n + Close_n}{4}$$

$$Open_n = \frac{Open_{n-1} + Close_{n-1}}{2}$$

$$High_n = \max\left(Open_n, High_n, Close_n\right)$$

$$Low_n = \min\left(Open_n, Low_n, Close_n\right)$$

By applying the above formulas to the OHLC array we have, it will give us a chart that has less color fluctuations. This means that the Heikin-Ashi chart will have less probability for a bar to switch from one color to another than a normal candlestick chart.

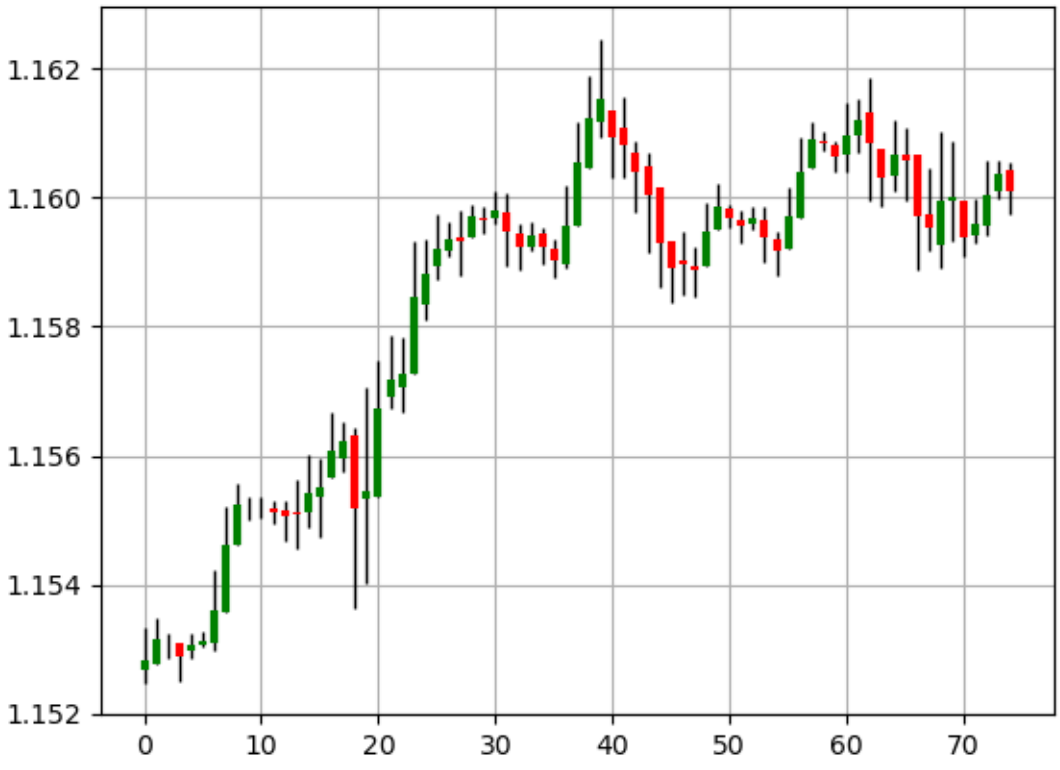



*Figure 60 Hourly EURUSD values with Heikin-Ashi chart.*

The two charts can be compared together and obviously, the Heikin-Ashi chart delivers a clearer view on the momentum and regime of the market.

```
def heikin_ashi(Data, opening, high, low, close, where):
  Data = adder(Data, 4)
  # Heikin-Ashi Open
  try:
    for i in range(len(Data)):
      Data[i, where] = (Data[i - 1, opening] + Data[i - 1, close]) / 2
  except:
    pass
  # Heikin-Ashi Close
  for i in range(len(Data)):
    Data[i, where + 3] = (Data[i, opening] + Data[i, high] + Data[i, low] + Data[i, close]) / 4
  # Heikin-Ashi High
  for i in range(len(Data)):
    Data[i, where + 1] = max(Data[i, where], Data[i, where + 3], Data[i, high])
  # Heikin-Ashi Low
  for i in range(len(Data)):
    Data[i, where + 2] = min(Data[i, where], Data[i, where + 3], Data[i, low])
  # Cleaning
  Data = jump(Data, 1)
  return Data
```

The above code is used as always with an OHLC data array. The formulas are applied in a recursive way to deliver four new columns with the Heikin-Ashi values inside. The technique not only gives some insights on the market's regime, but it also gives trading signals whenever there is a change in the color of the bar or whenever we detect a candlestick pattern. We must note however, that the Heikin-Ashi chart does not reflect the true price.

```
def ohlc_plot_candles(Data, window):
    Chosen = Data[-window:, ]
    for i in range(len(Chosen)):
        plt.vlines(x = i, ymin = Chosen[i, 6], ymax = Chosen[i, 5], color = 'black', linewidth = 1)
        if Chosen[i, 7] > Chosen[i, 4]:
            plt.vlines(x = i, ymin = Chosen[i, 4], ymax = Chosen[i, 7], color = 'green', linewidth = 3)
        if Chosen[i, 7] < Chosen[i, 4]:
            plt.vlines(x = i, ymin = Chosen[i, 7], ymax = Chosen[i, 4], color = 'red', linewidth = 3)
        if Chosen[i, 7] == Chosen[i, 4]:
            plt.vlines(x = i, ymin = Chosen[i, 7], ymax = Chosen[i, 4] + 0.00001, color = 'black', linewidth =
6)
    plt.grid()
```

# CHAPTER 26 THE MARKET'S SLOPE

Color-coding charts can be helpful to determine the market's state when we want to take a quick glance at the charts. It is true that we might be able to determine the trend visually or by using moving averages, but we can also use other indicators and transform the chart so that it reflects the indicator's values. This chapter discusses the slope (Steepness) of the market as a possible indication of the regime.

The steepness of two or more points is measured by the slope. A greater slope means a steeper line while a lower slope means a flatter line. We can measure the slope in geometry by using a simple formula. Basically, the slope can be found by measuring the ratio of the vertical change to the horizontal change.

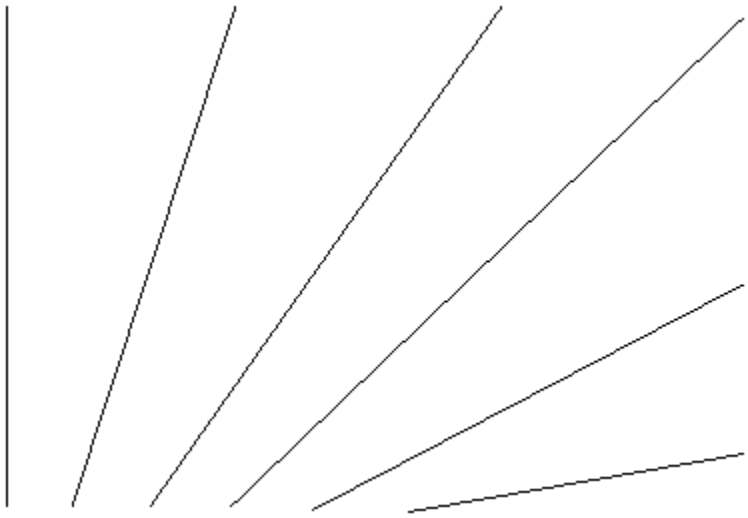


*Figure 61 From left to right, the line is getting flatter (Less Steep).*

In time series, when we are dealing with the slope of two successive points in time, the slope function shortens to just subtracting the last value by the one preceding it.

In case we want the slope of a wider interval, say 5 points in time, we must use the below formula:

$$Slope = \frac{\Delta Price}{\Delta Time}$$

For example, consider the following information on the EURUSD:

- Current closing price = 1.2300
- Closing price 5 periods ago = 1.2200
- Time interval selected = 5 periods

We can find that the slope for this period is the change in the closing prices which is 0.0100 divided by 5, giving us a slope of 0.002. We will apply a pure approach where we simply calculate the price slope and derive signals from it. This should give us an uncorrelated technical indicator that can help us diversify our framework. This slope indicator is a good addition to a systematic trading framework to identify the market's regime. Let us now code the indicator using the below function which loops around the OHLC data, takes the current closing price subtracts it from a closing price of a certain period ago that we choose (Lookback period), and divides by the length of time.

```
def slope_indicator(Data, lookback, close, where):
    Data = adder(Data, 1)
    for i in range(len(Data)):
        Data[i, where] = (Data[i, close] - Data[i - lookback, close]) / lookback
    Data = jump(Data, lookback)
    return Data
```

When we apply the function on the hourly values of EURCHF using a 10-period

parameter, we get the following graph. Notice the oscillation around the zero-level done by the indicator.

*Figure 62 Hourly EURCHF values with the 10-period Slope indicator*

The main idea here is that whenever the indicator surpasses the zero level, the market turns bullish and whenever the indicator breaks the zero level, the market turns bearish. To avoid whipsaws, we can also try increasing the lookback period, however that comes at the price of more lag. The indicator detects more movements and is more reactive whenever the lookback period is set to a low number as opposed to a high lookback period where more smoothing will occur, it is all a trade-off between volatility and smoothness. The next code snippet shows how to plot the previous chart.

```
def indicator_plot_double(Data, opening, high, low, close, second_panel, window = 250):
  fig, ax = plt.subplots(2, figsize = (10, 5))
  Chosen = Data[-window:, ]
  for i in range(len(Chosen)):
    ax[0].vlines(x = i, ymin = Chosen[i, low], ymax = Chosen[i, high], color = 'black', linewidth =
1)
    if Chosen[i, close] > Chosen[i, opening]:
      color_chosen = 'green'
      ax[0].vlines(x = i, ymin = Chosen[i, opening], ymax = Chosen[i, close], color = 'green',
linewidth = 2)
    if Chosen[i, close] < Chosen[i, opening]:
      ax[0].vlines(x = i, ymin = Chosen[i, close], ymax = Chosen[i, opening], color =’red’,
linewidth = 2)
    if Chosen[i, close] == Chosen[i, opening]:
      ax[0].vlines(x = i, ymin = Chosen[i, close], ymax = Chosen[i, opening], color = 'black',
linewidth = 2)
  ax[0].grid()
  ax[1].plot(Data[-window:, second_panel], color = 'royalblue', linewidth = 1)
  ax[1].grid()
```

Color-coding techniques reduce the noise seen in charts and does not take space therefore, it is ideal as a complementary tool. We will do the same thing as we have been doing in this regime detection part; whenever the market regime is bull, the bar is painted in green and whenever the market regime is bear, the bar is painted in red. The assumption we will be making is that the Slope is a trend-following indicator where values above zero point to a bullish regime and values below zero point to a bearish regime. For this, we will try to reduce the noise and use a lookback period of 50.

```
# Indicator Parameters
lookback = 50
my_data = slope_indicator(my_data, lookback, 3, 4)

def ohlc_plot_candles(Data, window):
    Chosen = Data[-window:, ]
    for i in range(len(Chosen)):
        if Chosen[i, 4] > 0:
            plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'green', linewidth = 1)
            plt.vlines(x = i, ymin = Chosen[i, 0], ymax = Chosen[i, 3], color = 'green', linewidth = 1)
        if Chosen[i, 4] < 0:
            plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'red', linewidth = 1)
            plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0], color = 'red', linewidth = 1)
    plt.grid()
```

*Figure 63 Hourly EURCHF values.*

Let us list a few of them but be careful, they are not to be used directly in trading as you must do your own back-tests before applying your ideas:

- Combining the slope with moving averages can be used to further confirm the trend. Therefore, we can buy whenever the slope is positive and the market crosses over its moving average. Similarly, we can sell short whenever the slope is negative and the market crosses below its moving average.
- Combining the slope with the Ichimoku Kinko Hyo can also be used to further confirm the trend. We can buy the market is within the cloud support and the slope is positive and vice versa for a short trade.

# CHAPTER 27 THE ELDER IMPULSE SYSTEM

The system combines a 13-period exponential moving average and a MACD oscillator. It color-codes the candlesticks following these conditions:

- A green bar is printed when the price is above the 13-period exponential moving average while simultaneously the MACD line is positive above zero.
- A red bar is printed when the price is below the 13-period exponential moving average while simultaneously the MACD line is negative below zero.
- A neutral black bar is printed when the price is above the 13-period exponential moving average while simultaneously the MACD line is negative below zero.
- A neutral black bar is printed when the price is below the 13-period exponential moving average while simultaneously the MACD line is positive above zero.

Therefore, to code the system, we need to prepare the 13-period exponential moving average and the MACD. We already know the functions.

```
# Calling the 13-period EMA
my_data = ema(my_data, 2, 13, 3, 4)
# Calling the MACD function using the default parameters
my_data = macd(my_data, 3, 26, 12, 9, 5)
# Removing the signal line as we do not need it in the Elder System
my_data = deleter(my_data, 6, 1)
```

Now, we are all set to code the conditions and transform the chart into an Elder impulse system chart. Note that this regime detection technique has the same issue with every backward-looking measure, lag. However, one person's lag is

another's confirmation and therefore, we can actually think of lag from a positive angle by assuming that the view we are expecting will last for a while and therefore, a form of confirmation can go a long away and help us to have more confidence in the trade. The next snippet shows how to code the Elder impulse system.

```
def elder_system_plot(Data, window):
  Chosen = Data[-window:, ]
  for i in range(len(Chosen)):
    plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'black', linewidth = 1)
    if Chosen[i, 3] > Chosen[i, 4] and Chosen[i, 5] > 0:
      color_chosen = 'green'
      plt.vlines(x = i, ymin = Chosen[i, 0], ymax = Chosen[i, 3], color = color_chosen, linewidth =
3)
    if Chosen[i, 3] < Chosen[i, 4] and Chosen[i, 5] < 0:
      color_chosen = 'red'
      plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0], color = color_chosen, linewidth =
3)
    if Chosen[i, 3] < Chosen[i, 4] and Chosen[i, 5] > 0:
      color_chosen = 'black'
      plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0] + 0.00001, color = color_chosen,
linewidth = 3)
    if Chosen[i, 3] > Chosen[i, 4] and Chosen[i, 5] < 0:
      color_chosen = 'black'
      plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0] + 0.00001, color = color_chosen,
linewidth = 3)
  plt.grid()
```

Remember to define the exponential moving average and the MACD functions prior to applying the previous code. With the code above defined, we need to call the function. Let us do that and define a lookback period of 250. This means that we will chart the latest 250 hourly values on the EURUSD.

```
elder_system_plot(my_data, 250)
```

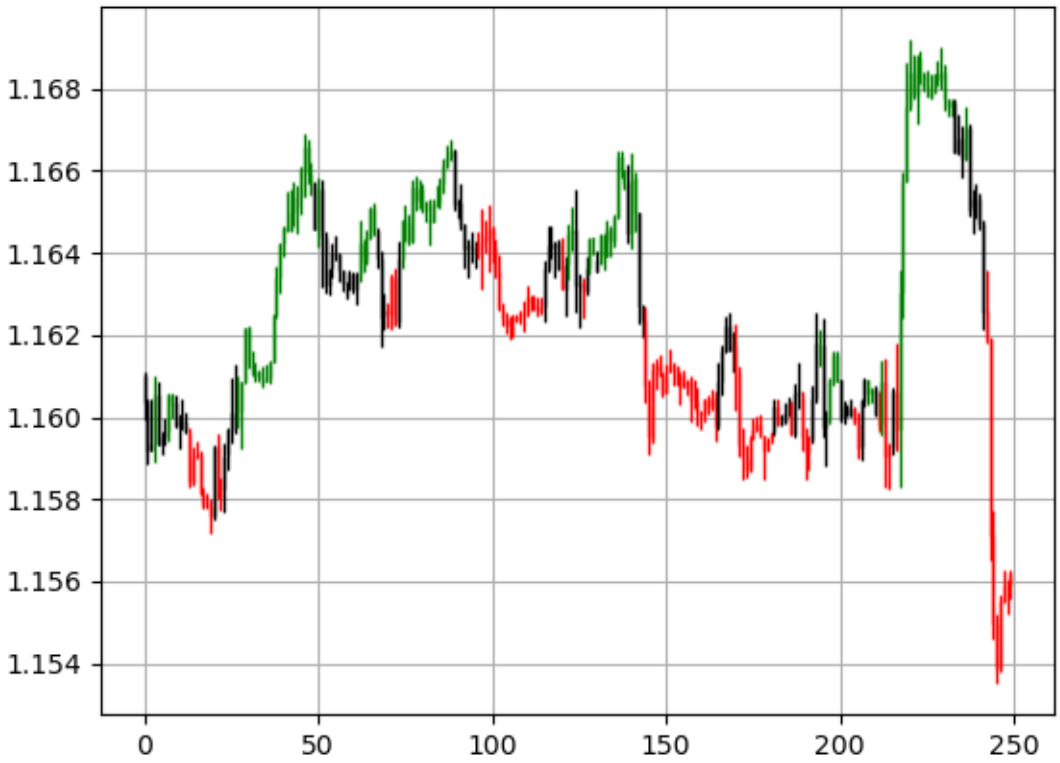



*Figure 64 – Hourly EURUSD values.*

We can note however the imperfections of the indicator due to the lag effect and of course, market randomness that imposes on us the impossibility of having a perfect prediction and diagnostic at any point of time. The simplest trading strategy with this indicator is to assume that a color change is a trading signal. Therefore:

- Whenever the market switches to green, a buy signal is generated.
- Whenever the market switches to red, a sell short signal is generated.
- Whenever the market switches to black, the position is closed.

The previous trading conditions are easy to code and view on a chart.

```
for i in range(len(my_data)):
    if my_data[i, 3] > my_data[i, 4] and my_data[i, 5] > 0 and my_data[i - 1, 5] < 0:
        my_data[i, 6] = 1
    elif my_data[i, 3] < my_data[i, 4] and my_data[i, 5] < 0 and my_data[i - 1, 5] > 0:
        my_data[i, 7] = -1
```

The function states that whenever the current market price is greater than the exponential moving average in column indexed at 4 while simultaneously, the MACD is above zero and no buy signal generated previously, a bullish signal is generated.

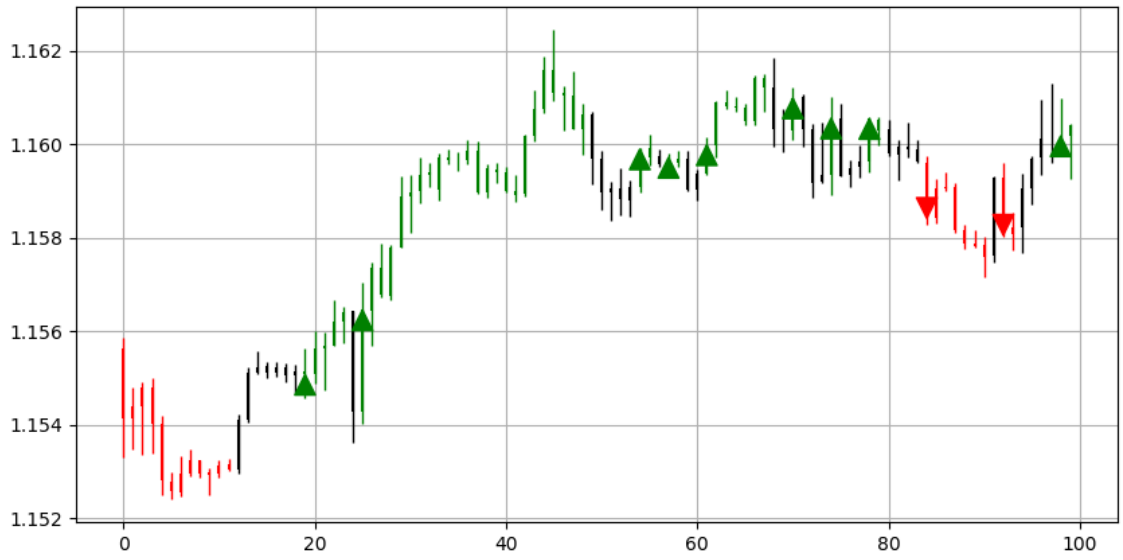



*Figure 65 – EURUSD Signal chart.*

The previous chart shows the signals generated using the function. Each time there is a color change, a signal is sent. Of course, this is just a regime detection system and not really a signal generator, which is why we need to look at it from a global perspective and not to find accurate trading positions

```
def signal_chart_bars(Data, close, what_bull, what_bear, window = 500):
    fig, ax = plt.subplots(figsize = (10, 5))
    Chosen = Data[-window:, ]
    for i in range(len(Chosen)):
        if Chosen[i, 3] > Chosen[i, 4] and Chosen[i, 5] > 0:
            plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'green', linewidth = 1)
            plt.vlines(x = i, ymin = Chosen[i, 0], ymax = Chosen[i, 3], color = 'green', linewidth = 1.5)
        if Chosen[i, 3] < Chosen[i, 4] and Chosen[i, 5] < 0:
            plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'red', linewidth = 1)
            plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0], color = 'red', linewidth = 1.5)
        if Chosen[i, 3] < Chosen[i, 4] and Chosen[i, 5] > 0:
            plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'black', linewidth = 1)
            plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0] + 0.00001, color = 'black',
linewidth = 1.5)
        if Chosen[i, 3] > Chosen[i, 4] and Chosen[i, 5] < 0:
            plt.vlines(x = i, ymin = Chosen[i, 2], ymax = Chosen[i, 1], color = 'black', linewidth = 1)
            plt.vlines(x = i, ymin = Chosen[i, 3], ymax = Chosen[i, 0] + 0.00001, color = 'black',
linewidth = 1.5)
    plt.grid()
    for i in range(len(Chosen)):
        if Chosen[i, what_bull] == 1:
            x = i
            y = Chosen[i, close]
            ax.annotate(' ', xy = (x, y),
                        arrowprops = dict(width = 9, headlength = 11, headwidth = 11, facecolor = 'green',
color = 'green'))
        elif Chosen[i, what_bear] == -1:
            x = i
            y = Chosen[i, close]
            ax.annotate(' ', xy = (x, y),
                        arrowprops = dict(width = 9, headlength = -11, headwidth = -11, facecolor = 'red',
color = 'red'))
```

# PART 4 HANDS-ON TRADING STRATEGIES

We have finally reached the interesting part, although the previous chapters have introduced a crude amount of knowledge which is defining, understanding, coding, and charting the indicators, we now need to see them in action by combining some of them in structured trading strategies that have the aim of filtering and improving the signals gotten from individual indicators.

Before we start, let us take a look at how the strategies will be presented. First, we will not need to reintroduce the indicators used all over again, therefore, we will just refer to them before presenting the trading rules that seem to deliver better than random results with an acceptable number of trades over the last 10 years.

It is important to understand that these are not ready-to-be-deployed strategies, they are merely raw suggestions on which you can choose to build your system on. A good trading framework based on one of the presented strategies will likely look completely different but will just have its roots coming from them. **Always do your own back-tests** because only you can know exactly what your risk and reward appetite is. Remember, applying strategies you find online or in books is unlikely to lead to sustainable profitable results but transforming these strategies by absorbing their main ideas will give you enough experience to maneuver whenever there is some turbulence in the markets. In other words, strategies do not work forever, and you must understand when to switch your methods. Before we move on to the strategies, we will take a look at how to evaluate the strategies. The below sections discuss very simple performance evaluation measures that will serve as general evaluators but of course, more detailed measures must be done to

ensure the viability of the strategy. Also, for every strategy, there will be only one back-test to show how to interpret the measures, but it is not indicative of anything as it only tests one currency pair using no transaction costs. Many platforms propose back-testing services with real life simulation. For the mean time, we can code our strategies in Python and then check out how to export them to more real-life environments.

**The Holding Period**

A variable holding period is based on the concept of exiting upon the next signal. This means that when we buy, we will hold until getting another signal whether bullish or bearish and then we will close the position. Similarly, when we get a sell (Short) signal, we will remain bearish until getting a new signal whether bullish or bearish. Therefore, the result will be based on the difference between the close at exit and the entry price which is generally the closing price at the signal assuming perfect execution. Meanwhile, a fixed holding period assumes an exit after a specified amount of time. This is generally not recommended but we will use it as a comparison with the variable holding period.

**Number of Trades**

The number of trades reflects how frequent are the signals generated by the strategy. This gives us an idea on how many trades to expect which is a direct clue to transaction costs and the statistical significance of the back-testing results. There is no minimum or maximum number for this measure, it depends on the trader's appetite to take on trades.

### The Hit Ratio

The hit ratio is simply the number of winning trades over the number of the trades taken in total. For example, if we have 1359 trades over the course of 5 years and we have been profitable in 711 of them, then our hit ratio is 711/1359 = 52.31%. This means that on average, for every 100 trades taken, we tend to be winners in 52 of them. This measure is useless without risk management, and we will see in why in the section dealing with the risk-reward ratio.

### Profit Factor

This is a straightforward method that measures the total profit as a ratio of the total loss. Hence, we can say that the profit factor is how much we have gained compared to a $1 loss. If we consider that a strategy earned us $50,000 this year in gross profits and made us $20,000 in losses, then our profit factor is calculated as follows.

$$Profit\ Factor = \frac{50{,}000}{20{,}000} = 2.5$$

Therefore, for every $1 lost on the strategy, we have made $2.5 which is a sign of a good strategy in case we have adequate risk management.

### Expectancy

Expectancy is a flexible measure composed of the average gain per trade, the average loss per trade, and the hit ratio. It provides the expected profit or loss on a dollar figure weighted by the hit ratio. I became aware of this method following Laurent Bernut's amazing advice a few years ago.

$$Expectancy = (Average\ Gain\ x\ Hit\ Ratio) - (Average\ Loss\ x\ (1 - Hit\ Ratio))$$

Let us consider the following trading results on a basic strategy:

- Average gain in 2020: $125.00
- Average loss in 2020: $75.00
- Hit ratio in 2020: 45.00%

Our expectancy following this strategy is therefore:

$$Expectancy = (125\ x\ 0.45) - \big(75\ x\ (1 - 0.45)\big) = \$15.00$$

**Realized Risk-Reward Ratio**

The realized risk-reward ratio measures how much risk we are taking for every trade realized. This means that we will divide the average gain by the average loss. A good strategy will have a realized risk-reward ratio of at least 1.80 – 2.00 so that we have enough margin with the hit ratio of the strategy.

$$Realized\ RiskReward\ Ratio = \frac{Average\ Gain}{Average\ Loss}$$

Trading is not about perfectly predicting the markets; success is dependent on how well we manage risk. Let us consider the below example:

- Average gain in 2020: $125.00
- Average loss in 2020: $75.00

$$Realized\ RiskReward\ Ratio = \frac{125}{75} = 1.67$$

The above results show that we are probably taking a little more risk than is generally acceptable. This does not mean that it is a bad strategy. The hit ratio and the risk-reward ratio go hand in hand and successful strategies rely on a healthy trade-off between the two.

Consider these two examples below:

- A strategy with a 90.00% hit ratio and a risk-reward ratio of 0.05 will unlikely result in any profits. This is because with a risk-reward ratio of 0.05, we will need 95.23% hit ratio just to break-even assuming the same monetary size trades.
- A strategy with a 30.00% hit ratio and a risk-reward ratio of 4 will likely be profitable. This is because with a risk-reward ratio of 4, we will need 20.00% hit ratio just to break-even assuming the same monetary size trades.

To calculate the break-even hit ratio, we can follow this below formula:

$$Breakeven\ Hit\ Ratio = \frac{1}{(1 + RiskReward\ Ratio)}$$

As weird as it looks, but sometimes winning 9 trades out 10 is less profitable than winning 3 trades out of 10 trades. This is a basic of risk management, knowing how much to risk and what to target. The back-tests in this chapter will assume a $100,000 balance with standard lots traded every time on the EURUSD hourly values since 2011.

Let us create two master functions (One for the variable holding period and the other for the fixed holding period) that englobe the discussed performance metrics which we will later use for each strategy. Remember to do your own research and tweaks before you decide to adopt any strategy you see here or online and to always include risk management tools such as stops and targets in your framework.

```
def performance_variable_period(Data, close, buy, sell, long_result_col, short_result_col,
total_result_col):
  # Adding a few columns
  Data = adder(Data, 10)
  # Variable Holding Period
  for i in range(len(Data)):
    try:
      if Data[i, buy] == 1:
        for a in range(i + 1, i + 1000):
          if Data[a, buy] == 1 or Data[a, sell] == -1:
            Data[a, long_result_col] = Data[a, close] - Data[i, close]
            break
          else:
            continue
      else:
        continue
    except IndexError:
      pass
  for i in range(len(Data)):
    try:
      if Data[i, sell] == -1:
        for a in range(i + 1, i + 1000):
          if Data[a, buy] == 1 or Data[a, sell] == -1:
            Data[a, short_result_col] = Data[i, close] - Data[a, close]
            break
          else:
            continue
      else:
        continue
    except IndexError:
      pass
```

```
def performance_fixed_period(Data, close, buy, sell, period, long_result_col, short_result_col,
total_result_col):
    # Adding a few columns
    Data = adder(Data, 10)
    # Fixed Period Holding
    for i in range(len(Data)):
        try:
            if Data[i, buy] == 1:
                Data[i + period, long_result_col] = Data[i + period, close] - Data[i, close]
            elif Data[i, sell] == -1:
                Data[i + period, short_result_col] = Data[i, close] - Data[i + period, close]
        except IndexError:
            pass
    # Aggregating the Long & Short Results Into One Column
    Data[:, total_result_col] = Data[:, long_result_col] + Data[:, short_result_col]
    # Profit Factor
    total_net_profits = Data[Data[:, total_result_col] > 0, total_result_col]
    total_net_losses  = Data[Data[:, total_result_col] < 0, total_result_col]
    total_net_losses  = abs(total_net_losses)
    profit_factor     = round(np.sum(total_net_profits) / np.sum(total_net_losses), 2)
    # Hit Ratio
    hit_ratio         = len(total_net_profits) / (len(total_net_losses) + len(total_net_profits))
    hit_ratio         = round(hit_ratio, 2) * 100
    # Risk Reward Ratio
    average_gain          = total_net_profits.mean()
    average_loss          = total_net_losses.mean()
    realized_risk_reward  = average_gain / average_loss
    # Expectancy
    expectancy    = (average_gain * (hit_ratio / 100)) - ((1 - (hit_ratio / 100)) * average_loss)
    expectancy    = round(expectancy, 4)
```

```
# Number of Trades
trades = len(total_net_losses) + len(total_net_profits)
print('Hit Ratio        = ', hit_ratio)
print('Average Gain     = ', average_gain * 100000)
print('Average Loss     = ', average_loss * 100000)
print('Expectancy       = ', expectancy * 100000)
print('Profit factor    = ', profit_factor)
print('Realized RR      = ', round(realized_risk_reward, 3))
print('Number of Trades  = ', trades)
return Data
```

It is worth mentioning that you do not need to copy the code from the book as there is a dedicated GitHub which harbors every line of code seen in the book organized properly. Focus on understanding the concepts while moving forward.

## STRATEGY #1 BUYING THE DIPS & SELLING THE STRENGTHS

As a reminder, an overlay indicator is one that is charted alongside the market in the same panel. Moving averages provide excellent dynamic support and resistance levels. This strategy is mainly used to find optimal entries to ride the trend or add positions to an existing trade. We should use long-term moving averages such as the Fibonacci moving average, or any other type of moving average with a high lookback period (Typically 200 and above). Another important aspect to add is noise management where we should calculate a zone based on highs and lows rather than just one line calculated on closing prices. Let us test the below strategy:

- **A long position may be opened whenever the market price enters the 800-period moving average zone from the above.**
- **A short position may be opened whenever the market price enters the 800-period moving average zone from the below.**

We first have to calculate two 800-period moving averages on the highs and lows of the OHLC data.

```
lookback = 800
my_data = ma(my_data, lookback, 1, 4)
my_data = ma(my_data, lookback, 2, 5)
```

Remember that in an OHLC data array, the column indexed at 1 refers to the highs and the column indexed at 2 refers to the lows. The next chart shows the signals generated following the above strategy. The code for the signal will follow right after.

*Figure 67 – Signal chart on the EURUSD.*

We can see that every time the market enters the zone, a trade is triggered. The above chart shows that the strategy is unlikely to show frequent signals due to the long-term nature of the moving average that tends to stay far from the market price.

*Figure 68 – Signal chart on the EURCAD.*

```
def signal(Data, high, low, ma_high_col, ma_low_col, buy, sell):
    Data = adder(Data, 10)
    for i in range(len(Data)):
        if Data[i, low] > Data[i, ma_low_col] and \
            Data[i, low] < Data[i, ma_high_col] and \
            Data[i - 1, low] > Data[i, ma_high_col] and \
            Data[i - 10, low] > Data[i, ma_high_col]:
                Data[i, buy] = 1
        if Data[i, high] < Data[i, ma_high_col] and \
            Data[i, high] > Data[i, ma_low_col] and \
            Data[i - 1, high] < Data[i, ma_low_col] and \
            Data[i - 10, high] < Data[i, ma_high_col]:
                Data[i, sell] = -1
    return Data
```

The last figure shows an interesting set-up on EURCAD where the market validated a strong bearish trend after pulling back to the broken moving average. The algorithm detected this opportunity and printed a sell signal at the closing price within the moving average zone. The below results are the output of an hourly back-test on the EURUSD pair with a 15-period hold for the fixed holding period strategy.

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 38.00% | 53.00% |
| Average Gain | $ 364.25 | $ 370.97 |
| Average Loss | $ 292.96 | $ 378.53 |
| Profit Factor | $ 0.76 | $ 1.12 |
| Expectancy | $ -40.00 | $ 20.00 |
| Risk-Reward | $ 1.24 | $ 0.98 |
| Trades | 531 | 531 |

We can clearly see that the fixed period strategy has outperformed the variable period strategy. The expectancy metric is showing $20.00 per trade as opposed to -$40.00 per trade for the variable period strategy. However, the risk-reward ratio for the fixed period strategy is in the red zone by being less than 1.00 which is generally not recommended unless the strategy dictates it and generally, trend-following strategies should have high risk-reward ratios with low volatility returns.

## STRATEGY #2 MOVING AVERAGE CROSSOVERS

Moving average crosses are among the most known strategies in trend-following. Who among us has not heard of the terms "Golden Cross and "Death Cross"? These two events are the pillars of moving average crossovers. Here are the trading rules of the strategy:

- **A long position may be opened whenever the 50-period moving average crosses over the 100-period moving average from the below. This is referred to as a Golden Cross.**
- **A short position may be opened whenever the 50-period moving average crosses under the 100-period moving average from the above. This is referred to as a Death Cross.**

A cross signals a confirmed change in the trend. The confirmation part is due to the inherent lag with the moving averages.

```
lookback_short = 50
lookback_long  = 100
my_data = ma(my_data, lookback, 3, 5)
my_data = ma(my_data, lookback, 3, 6)
```

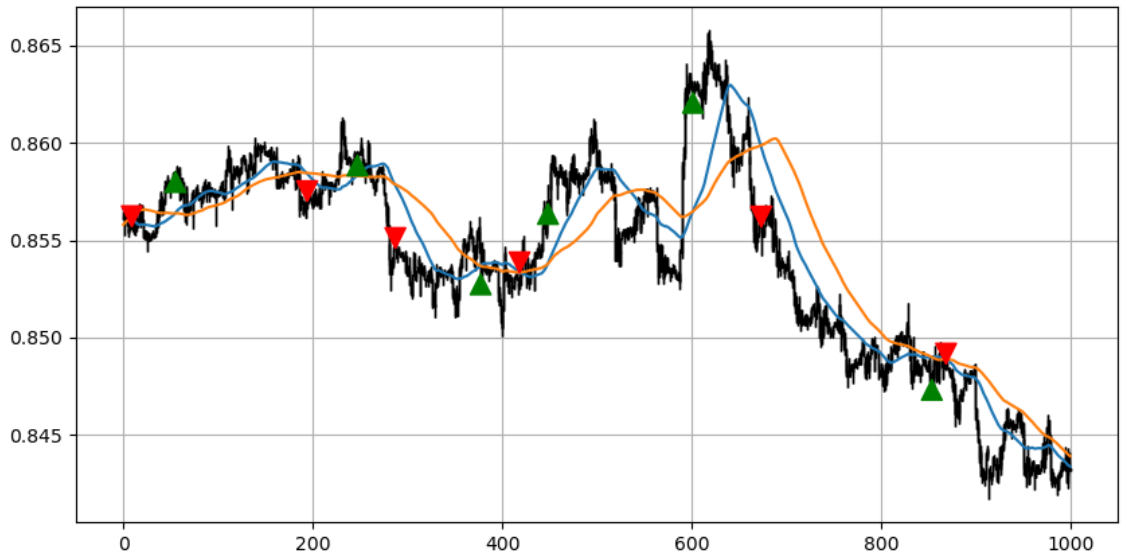



*Figure 69 – Signal chart on the EURGBP.*

```
def signal(Data, close, ma_short, ma_long, buy, sell):
    Data = adder(Data, 10)
    for i in range(len(Data)):
        if Data[i, ma_short] > Data[i, ma_long] and \
            Data[i - 1, ma_short] < Data[i - 1, ma_long]:
                Data[i, buy] = 1
        if Data[i, ma_short] < Data[i, ma_long] and \
            Data[i - 1, ma_short] > Data[i - 1, ma_long]:
                Data[i, sell] = -1
    return Data
```

The last figure shows an interesting set-up on EURGBP where the market is showing good crossover signals.

```
my_data = performance(my_data)
```

The below results are the output of an hourly back-test on the EURUSD pair.

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 42.00% | 51.00% |
| Average Gain | $ 1,002.26 | $ 366.29 |
| Average Loss | $ 649.12 | $ 392.02 |
| Profit Factor | $ 1.13 | $ 0.99 |
| Expectancy | $ 40.00 | $ -10.00 |
| Risk-Reward | $ 1.54 | $ 0.93 |
| Trades | 701 | 701 |

The results show that the variable holding period strategy is mildly performing well on the EURUSD but right over the edge and hence, nothing significant. The fixed holding period strategy has underperformed as can be seen from the negative expectancy.

## STRATEGY #3 THE RSI & MOVING AVERAGES

This is one of the simplest trend-following strategies which involve two of the most known technical indicators. First, let us understand and code the relative strength index – RSI before presenting the strategy. The RSI is the most famous momentum indicator, and this is to be expected as it has many strengths especially in ranging markets. It is also bounded between 0 and 100 which makes it easier to interpret. Also, the fact that it is famous, contributes to its potential. This is because the more traders and portfolio managers look at the RSI, the more people will react based on its signals and this in turn can push market prices.. The RSI is calculated using a rather simple way. We first start by taking price differences of one period. This means that we must subtract every closing price from the one before it. Then, we will calculate the smoothed average of the positive differences and divide it by the smoothed average of the negative differences. The last calculation gives us the relative strength which is then used in the RSI formula to be transformed into a measure between 0 and 100.

*Figure 70 – Hourly USDCHF values with the 14-period RSI.*

```
def rsi(Data, lookback, close, where, width = 1):
    # Adding a few columns
    Data = adder(Data, 5)
    # Calculating Differences
    for i in range(len(Data)):
        Data[i, where] = Data[i, close] - Data[i - width, close]
    # Calculating the Up and Down absolute values
    for i in range(len(Data)):
        if Data[i, where] > 0:
            Data[i, where + 1] = Data[i, where]
        elif Data[i, where] < 0:
            Data[i, where + 2] = abs(Data[i, where])
    # Calculating the Smoothed Moving Average on Up and Down absolute values
    lookback = (lookback * 2) - 1 # From exponential to smoothed
    Data = ema(Data, 2, lookback, where + 1, where + 3)
    Data = ema(Data, 2, lookback, where + 2, where + 4)
    # Calculating the Relative Strength
    Data[:, where + 5] = Data[:, where + 3] / Data[:, where + 4]
    # Calculate the Relative Strength Index
    Data[:, where + 6] = (100 - (100 / (1 + Data[:, where + 5])))
    # Cleaning
    Data = deleter(Data, where, 6)
    Data = jump(Data, lookback)
    return Data
```

The trading strategy is based on a hybrid between trend-following and contrarian trading. We first detect the directional bias using moving averages and then time the entries using the relative strength index.

The trading rules will be straightforward and simple and although the below parameters must be optimized for each market, the idea remains the same:

- **A long position may be opened whenever the market is above its 50-period moving average while the 2-period RSI is below 5.**
- **A short position may be opened whenever the market is below its 50-period moving average while the 2-period RSI is above 95.**

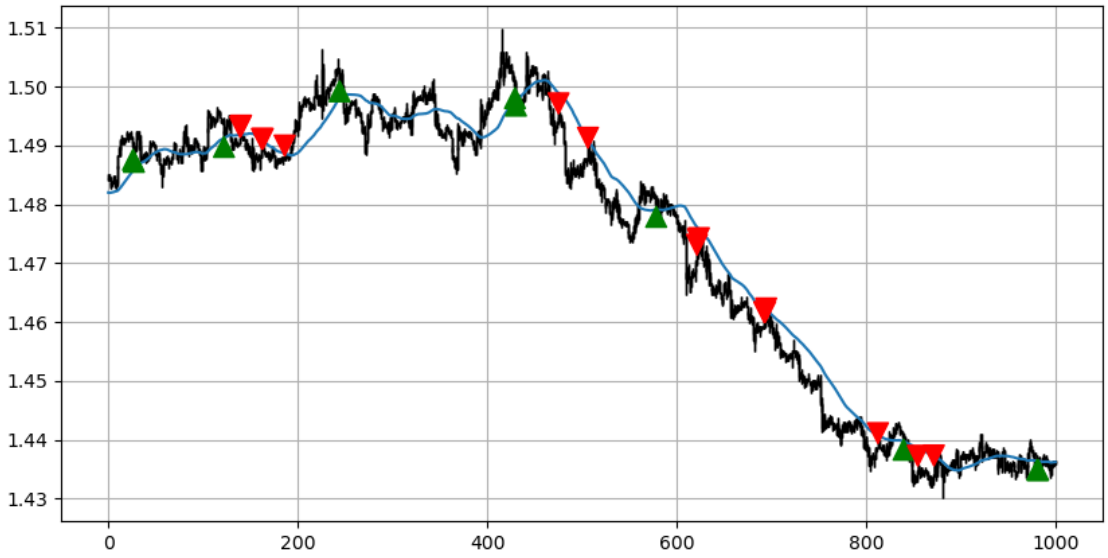



*Figure 71 – Signal chart on the EURCAD.*

```
def signal(Data, close, ma_col, rsi_col, buy, sell):
    Data = adder(Data, 10)
    for i in range(len(Data)):
        if Data[i, close] > Data[i, ma_col] and Data[i, rsi_col] < lower_barrier:
            Data[i, buy] = 1
        elif Data[i, close] < Data[i, ma_col] and Data[i, rsi_col] > upper_barrier:
            Data[i, sell] = -1
    return Data
```

The back-testing results based on the simple performance metrics presented on the EURUSD are as follows.

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 32.00% | 50.00% |
| Average Gain | $ 743.07 | $ 367.11 |
| Average Loss | $ 362.14 | $ 379.58 |
| Profit Factor | $ 0.94 | $ 0.98 |
| Expectancy | $ -10.00 | $ -10.00 |
| Risk-Reward | $ 2.05 | $ 0.97 |
| Trades | 1532 | 1532 |

The results show that the variable strategy is showing around the same profitability as the fixed period strategy when taking into account the trade-off between the hit ratio and the risk-reward ratio. These low metrics must never deter us from pursuing the strategy and digging deeper as no optimization nor risk management is included in the results. The two form an inseparable part of any trading system.

We should also remember that we are back-testing the strategies on just one currency pair through the hourly time frame, thus our conclusions are very limited. I encourage you to perform the back-tests on a wide variety of assets using multiple risk management tools. Only then will you see the potential.

## STRATEGY #4 THE MACD CROSSOVER

The MACD is an interesting oscillator created from the difference of two exponential moving averages. Many strategies can be formed around it with the simplest being one of the best. Whenever the MACD crosses the zero line, we get a trend change confirmation signal. The conditions for the trade are below:

- **A long position may be opened whenever the MACD surpasses the zero line.**
- **A short position may be opened whenever the MACD breaks the zero line.**

*Figure 72 – Signal chart on the EURUSD.*

It is relatively easy to code the above conditions as we search for rows where the MACD is above zero and check for the previous value if it is below zero or not. In case it is, the bullish signal is validated and 1 is inputted in the buy column, and vice versa for the bearish case.

```
def signal(Data, macd_col, buy, sell):
    Data = adder(Data, 10)
    for i in range(len(Data)):
        if Data[i, macd_col] > 0 and Data[i - 1, macd_col] < 0:
            Data[i, buy] = 1
        elif Data[i, macd_col] < 0 and Data[i - 1, macd_col] > 0:
            Data[i, sell] = -1
    return Data
```

*Figure 73 – Signal chart on the NZDUSD.*

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 30.00% | 51.00% |
| Average Gain | $ 714.55 | $ 362.33 |
| Average Loss | $ 318.82 | $ 359.56 |
| Profit Factor | $ 0.96 | $ 1.03 |
| Expectancy | $ -10.00 | $ 10.00 |
| Risk-Reward | $ 2.24 | $ 1.00 |
| Trades | 2234 | 2234 |

The results show that the variable holding period strategy not performing well but is showing superior risk-reward ratio. Of course, this is useless if the hit ratio is very low. The fixed holding period strategy is showing positive expectancy but not by much since the hit ratio is only 1% higher than random and with a risk-reward of $1.00, it is mostly a gambling strategy.

## STRATEGY #5 THE MOVING AVERAGE SLOPE

Having seen the concept of slopes in a previous chapter, we will now apply it in a trading strategy but instead of calculating the slope of the market price, we will calculate the slope of its moving average to generate trading signals. Here is how:

- **A long position may be opened whenever the slope of the 20-period moving average turns positive.**
- **A short position may be opened whenever the slope of the 20-period moving average turns negative.**

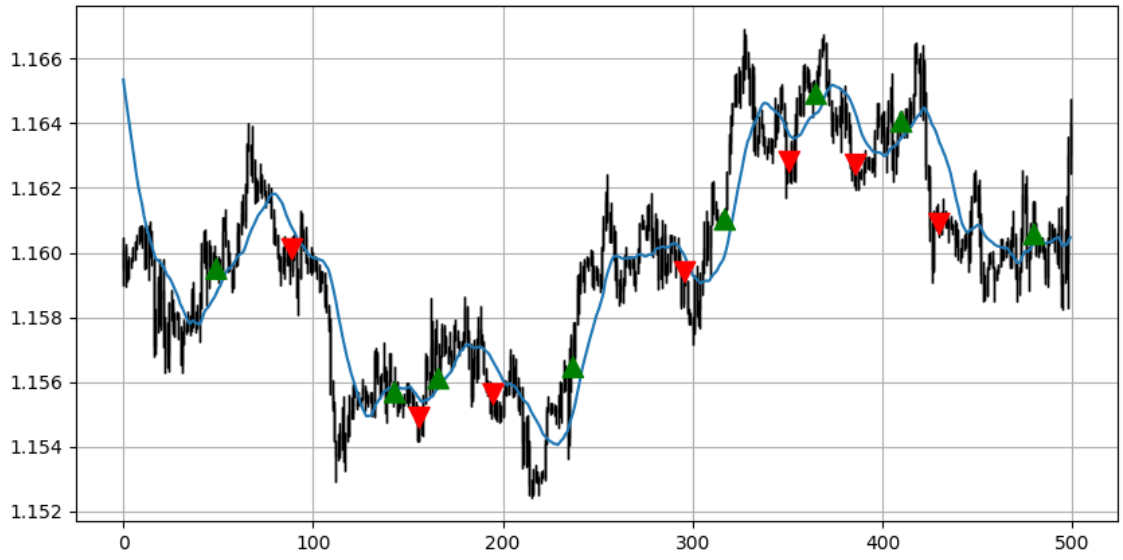



*Figure 74 – Signal chart on the EURNZD.*

```
lookback = 20
my_data = ma(my_data, lookback, 3, 4)
my_data = slope_indicator(my_data, lookback, 4, 5)
my_data = signal(my_data, 5, 6, 7)
```

The code snippet defines the 20-period lookback, calculates the moving average and then applies the slope function we have already defined from an earlier chapter.

```
def signal(Data, ma_slope_col, buy, sell):
  Data = adder(Data, 10)
  for i in range(len(Data)):
    if Data[i, ma_slope_col] > 0 and Data[i - 1, ma_slope_col] < 0:
        Data[i, buy] = 1
    elif Data[i, ma_slope_col] < 0 and Data[i - 1, ma_slope_col] > 0:
        Data[i, sell] = -1
  return Data
```

Let us take a quick look at the simplistic back-testing results we have so far.

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 38.00% | 50.00% |
| Average Gain | $ 655.46 | $ 359.34 |
| Average Loss | $ 425.71 | $ 352.62 |
| Profit Factor | $ 0.93 | $ 1.01 |
| Expectancy | $ -10.00 | $ 0.00 |
| Risk-Reward | $ 1.54 | $ 1.01 |
| Trades | 1774 | 1774 |

It looks like the fixed holding period strategy does slightly better than the variable holding period strategy but overall, both do not seem to add value.

## STRATEGY #6 THE SUPERTREND FLIP

As previously seen in an earlier chapter, the SuperTrend indicator is a trend-following technique used to detect the changes in the market's dynamics. We usually use the basic flip strategy to generate trades. The following are the trading rules used as trading triggers:

- **A long position may be opened whenever the market turns above the SuperTrend indicator.**
- **A short position may be opened whenever the market turns below the SuperTrend indicator.**

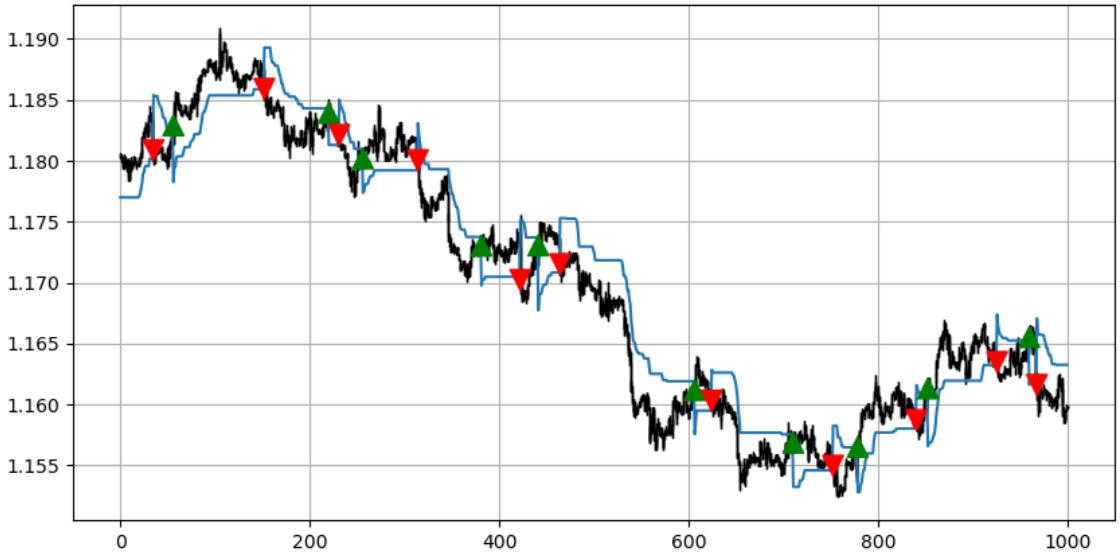



*Figure 75 – Signal chart on the EURUSD.*

```
def signal(Data, close, supertrend_col, buy, sell):
    Data = adder(Data, 10)
    for i in range(len(Data)):
        if Data[i, close] > Data[i, supertrend_col] and Data[i - 1, close] < Data[i - 1, supertrend_col]:
            Data[i, buy] = 1
        elif Data[i, close] < Data[i, supertrend_col] and Data[i - 1, close] > Data[i - 1, supertrend_col]:
            Data[i, sell] = -1
    return Data
lookback = 10
multiplier = 4
my_data = atr(my_data, lookback, 1, 2, 3, 4)
my_data = supertrend(my_data, multiplier, 4, 3, 1, 2, 5)
my_data = signal(my_data, 3, 5, 6, 7)
```

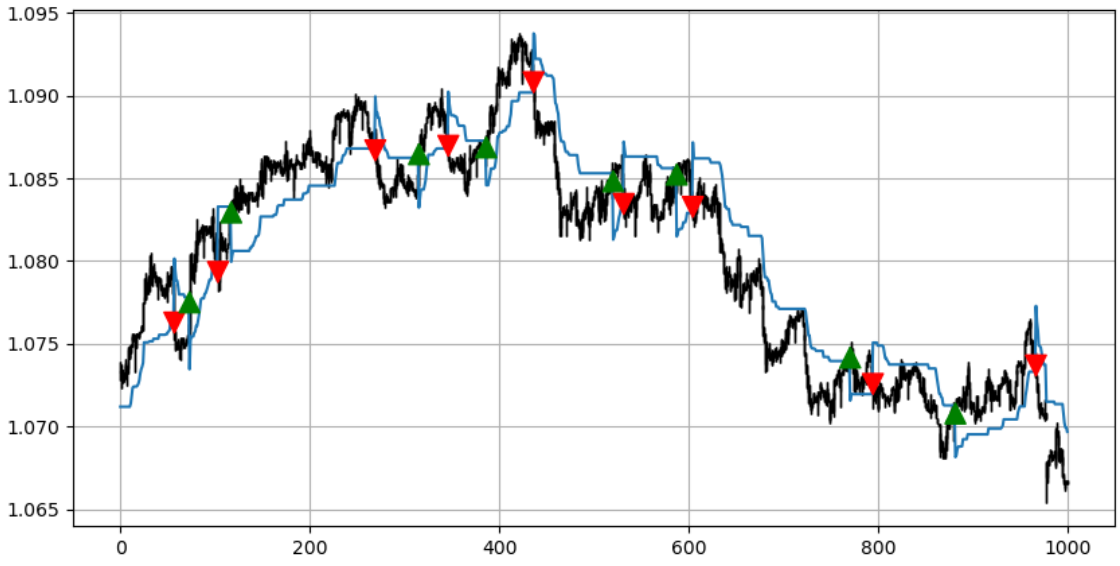



*Figure 76 – Signal chart on the EURCHF.*

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 37.00% | 50.00% |
| Average Gain | $ 869.71 | $ 352.25 |
| Average Loss | $ 556.50 | $ 358.87 |
| Profit Factor | $ 0.93 | $ 0.97 |
| Expectancy | $ -30.00 | $ -0.00 |
| Risk-Reward | $ 1.16 | $ 0.98 |
| Trades | 1180 | 1180 |

The variable holding period does not provide much added-value to the strategy as with only 37.00% hit ratio and a risk-reward ratio at 1.16, we are unlikely to make any money. The fixed holding period shows a slight improvement but without risk management, we cannot have a definitive answer. The reason I do not include risk management in these back-tests is because it is a function of each trader, and I cannot dictate how much risk should you take. I prefer to give out raw back-tests to show how to evaluate the performance and to give an idea on the general statistics of strategies in their basic format so that the reader builds upon this framework and adds her own secret sauce which makes the strategy stable and profitable, that is if it is possible.

## STRATEGY #7 THE PARABOLIC SAR & MOVING AVERAGES

We have previously discussed the Parabolic SAR indicator's flips may point to a change in the short-term trend. We will try to apply that idea here by adding a moving average to filter the signals. As simple as moving averages are, they provide a great filter to any strategy. The trading rules will be as follows:

- **A long position may be opened whenever the market surpasses the Parabolic SAR while being above its 50-period simple moving average.**
- **A short position may be opened whenever the market breaks the Parabolic SAR while being below its 50-period simple moving average.**

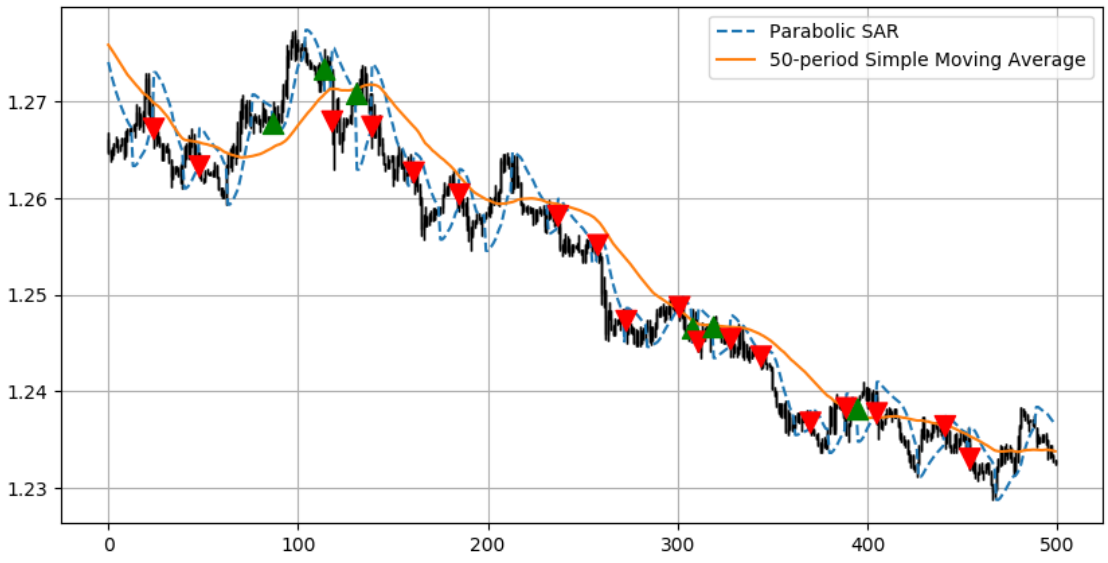



*Figure 77 – Signal chart on the GBPUSD.*

The above chart shows the signals generated from the strategy on the USDCAD pair. We can clearly see that every time there is a switch between the price and

the Parabolic SAR while the moving average satisfies its condition, a position is triggered.

```
def signal(Data, close, psar_col, ma_col, buy, sell):
  Data = adder(Data, 10)
  for i in range(len(Data)):
    if Data[i, close] > Data[i, psar_col] and \
      Data[i, close] > Data[i, ma_col] and \
      Data[i - 1, close] < Data[i - 1, psar_col]:
        Data[i, buy] = 1
    if Data[i, close] < Data[i, psar_col] and \
      Data[i, close] < Data[i, ma_col] and \
      Data[i - 1, close] > Data[i - 1, psar_col]:
        Data[i, sell] = -1
  return Data
```

To prepare the field for the strategy, we must call the Parabolic SAR and the moving average's functions.

```
lookback = 50
my_data = pd.DataFrame(my_data)
my_data.columns = ['open','high','low','close']
Parabolic = sar(my_data, 0.02, 0.2)
Parabolic = np.array(Parabolic)
Parabolic = np.reshape(Parabolic, (-1, 1))
my_data = np.concatenate((my_data, Parabolic), axis = 1)
my_data = ma(my_data, lookback, 3, 5)
my_data = signal(my_data, 3, 4, 5, 6, 7)
```

The above syntax assumes we have an array which is why it switches it to a data frame to do some manipulation and output the Parabolic SAR. After

converting back to an array, we calculate the 50-period moving average on the closing price. Finally, we call the signal function previously defined.

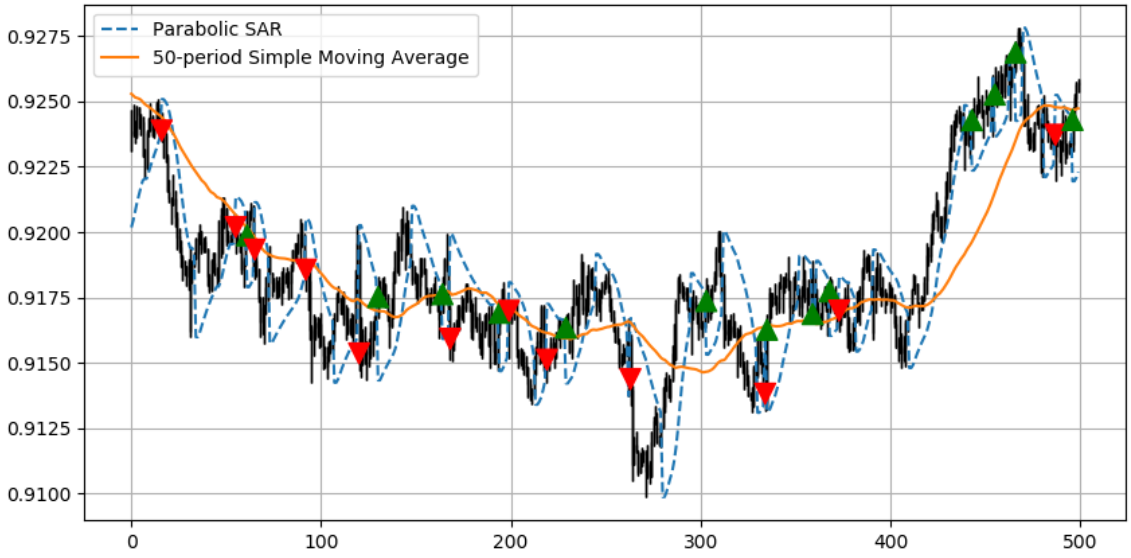



*Figure 78 - Signal chart on the USDCHF.*

Let us check the back-testing results following this strategy.

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 45.00% | 49.00% |
| Average Gain | $ 492.45 | $ 373.27 |
| Average Loss | $ 397.51 | $ 373.95 |
| Profit Factor | $ 1.01 | $ 0.96 |
| Expectancy | $ 0.00 | $ -10.00 |
| Risk-Reward | $ 1.23 | $ 0.99 |
| Trades | 2982 | 2982 |

## STRATEGY #8 PSYCHOLOGICAL LEVELS & MOVING AVERAGES

Detecting support & resistance levels is a highly subjective field of analysis and this explains the complete discrepancy between results. However, it should not be this way because we have many ways of making the subjective part rather objective. We can do this in many ways, among them are psychological levels, an interesting phenomenon based on human psychology and anchoring.

At its purest form, a support level is an area supposed to keep prices from going down further. Naturally, when the market price approaches a support area, the right decision is to have a bullish bias. A resistance level is an area supposed to keep prices from going up further. Naturally, when the market price approaches a resistance area, the right decision is to have a bearish bias.

Psychological levels form an important part in any analysis. The reason for this is because mentally, they are given more attention than other levels. For instance, which price would you retain more in your mind if you come across it? 1.1500 on the EURUSD or 1.3279 on the GBPUSD? By Psychological Levels, we are talking about either round number (such as 1.2000 on the EURUSD) or simply par numbers (such as 1.000 USDCHF or $100 on Apple stock). Clearly, round numbers are the first part of a psychological price. The other part is simply significance. For example, the par-level such as 1.000 on the USDCHF or 100.00 on the USDJPY is considered a par-level. The basic idea is that around these levels, market participants may choose to place their orders or stops, hence, a form of reaction may happen theoretically. Our aim is to develop an algorithm that trades whenever the market reaches a psychological level with a trend filter based on a moving average.

Therefore, our trading conditions will be as follows.

- **A long position may be opened whenever the market closes at a psychological level while being above its 50-period simple moving average.**
- **A short position may be opened whenever the market closes at a psychological level while being below its 50-period simple moving average.**

To develop a loop across numpy arrays to find the needed levels, we can follow these steps:

- Loop across all rows in the fourth column which harbors the closing prices.
- Mark the times when there is a closing price ending in a hundred pips position such as 1.1000 or 2.300.
- Increase the interval by 100 pips (0.0100) each time so that we scan for all possible levels.

```
def psychological_levels_scanner(Data, close, where):
    # Adding buy and sell columns
    Data = adder(Data, 10)
    # Rounding for ease of use
    Data = rounding(Data, 4)
    # Threshold
    level    = 0.00
    # Scanning for Psychological Levels
    for i in range(len(Data)):
        for i in range(len(Data)):
            if Data[i, close] == level:
                Data[i, where] = 1
        level = round(level + 0.01, 2)
        if level > 5:
            break
    return Data
# You can choose your starting level. I recommend starting at zero
```

Now, let us write the code for the 50-period moving average followed by the signal function that will satisfy the conditions of the trading rules stated previously.

```
def signal(Data, high, low, close, psychological_signal, ma_col, buy, sell):
    Data = adder(Data, 10)
    for i in range(len(Data)):
        if Data[i, psychological_signal] == 1 and \
            Data[i, high] > Data[i, ma_col] and \
            Data[i - 1, close] > Data[i, close]:
                Data[i, buy] = 1
        if Data[i, psychological_signal] == 1 and \
            Data[i, low] < Data[i, ma_col] and \
            Data[i - 1, close] < Data[i, close]:
                Data[i, sell] = -1
    return Data
lookback = 100
my_data = psychological_levels_scanner(my_data, 3, 4)
my_data = ma(my_data, lookback, 3, 5)
my_data = signal(my_data, 1, 2, 3, 4, 5, 6, 7)
```

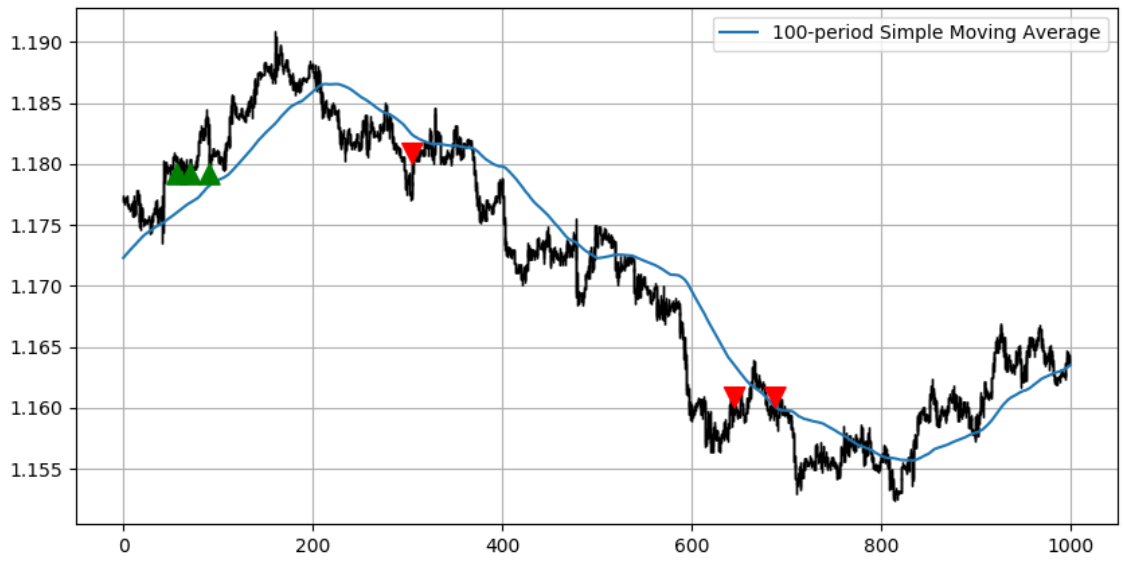



*Figure 79 – Signal chart on the EURUSD.*

Let us check some results from a back-test. Remember, we are always following the same rules as in the other strategies.

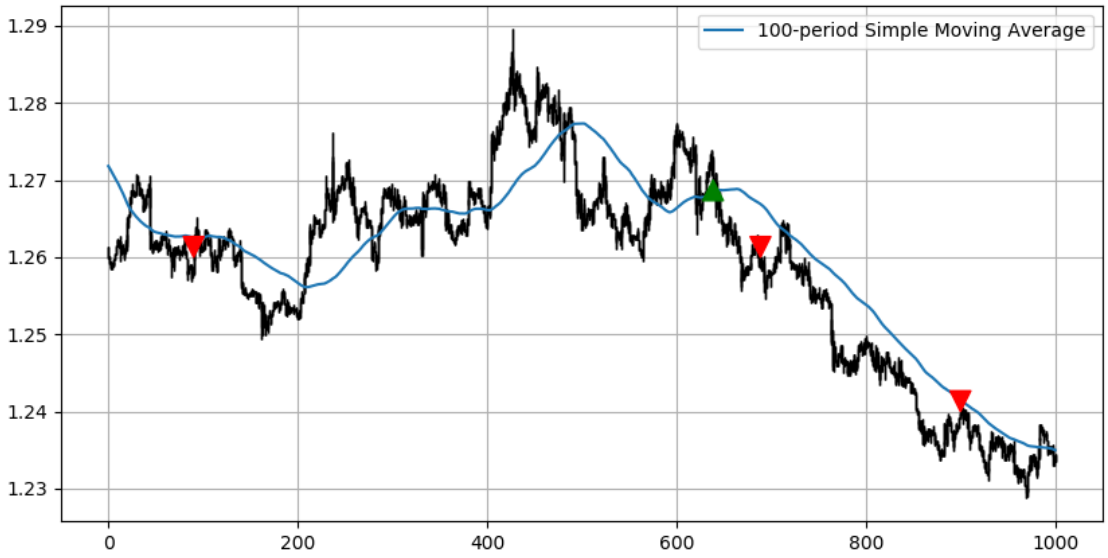



*Figure 80 – Signal chart on the USDCAD.*

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 56.00% | 50.00% |
| Average Gain | $ 1,745.28 | $ 408.32 |
| Average Loss | $ 1,880.95 | $ 278.30 |
| Profit Factor | $ 1.17 | $ 1.49 |
| Expectancy | $ 150.00 | $ 70.00 |
| Risk-Reward | $ 0.93 | $ 1.46 |
| Trades | 190 | 190 |

Psychological levels have a higher hit ratio than other levels and this is evidenced by the table above. However, when it comes to profitability, we have yet to prove their worth. The table shows a 56.00% hit ratio with a 0.93 risk-reward ratio suggesting a break-even hit ratio of 52.00%.

## STRATEGY #9 INTENSITY MOVING AVERAGES

We have previously discussed the trend intensity index and how it is used. We will try to apply the strategy here by adding a moving average to filter the signals. The trading rules will be as follows:

- **A long position may be opened whenever the 14-period trend intensity index surpasses 60 while the market price is above its 100-period simple moving average.**
- **A short position may be opened whenever the 14-period trend intensity index breaks 40 while the market price is below its 100-period simple moving average.**

```
def signal(Data, close, tii_col, ma_col, buy, sell):
  Data = adder(Data, 10)
  for i in range(len(Data)):
    if Data[i, tii_col] > 60 and \
      Data[i - 1, tii_col] < 60 and \
      Data[i, close] > Data[i, ma_col]:
        Data[i, buy] = 1
    if Data[i, tii_col] < 40 and \
      Data[i - 1, tii_col] > 40 and \
      Data[i, close] < Data[i, ma_col]:
        Data[i, sell] = -1
  return Data
```

As a reminder, the trend intensity index measures the strength of the trend and even though it can also be used as a contrarian indicator in some strategies, its true use is confirming the strength of the trend by looking at high and low values. In this strategy, we have chosen a 14-period trend intensity index with

60 as the bullish trend validation level and 40 as the bearish trend validation level.

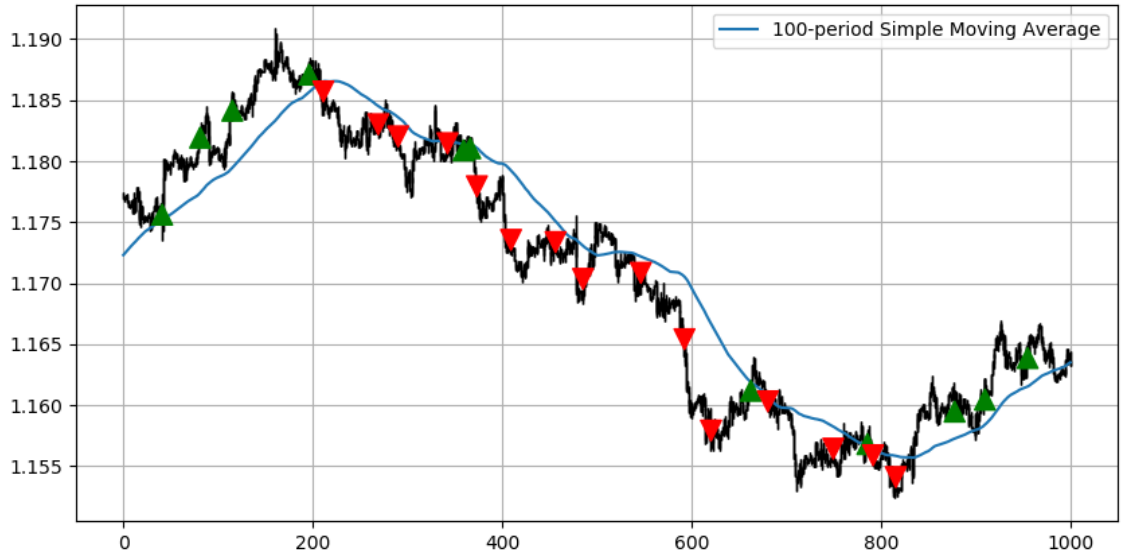



*Figure 81 – Signal chart on the EURUSD.*

```
lookback_ma  = 100
lookback_tii = 14
my_data = trend_intensity_indicator(my_data, lookback_tii, 3, 4)
my_data = ma(my_data, lookback_ma, 3, 5)
```

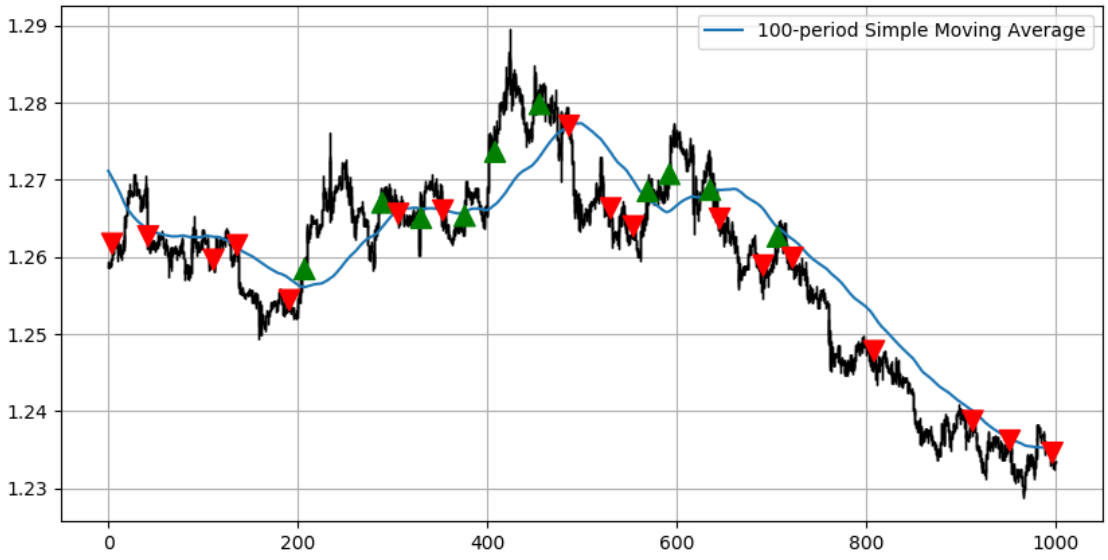



*Figure 82 – Signal chart on the USDCAD.*

To generate the signals, we can use the follow snippet.

```
my_data = signal(my_data, 3, 4, 5, 6, 7)
```

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 47.00% | 49.00% |
| Average Gain | $ 595.94 | $ 337.22 |
| Average Loss | $ 552.67 | $ 352.79 |
| Profit Factor | $ 0.95 | $ 0.91 |
| Expectancy | $ -10.00 | $ -10.00 |
| Risk-Reward | $ 1.08 | $ 0.96 |
| Trades | 1657 | 1657 |

The results show that with the chosen parameters, the strategy is unlikely to provide profitable results. However, optimization is possible, and the addition of risk management rules will make us master this strategy. At the moment, any outperformance of the fixed period relative to the variable period caused by the hit ratio is reversed by the risk-reward ratio.

## STRATEGY #10 HIDDEN DIVERGENCES

Divergences are known to be counter trend signals that show an exhaustion in the current trend. There exists a type of divergences that signal a trend continuation. These divergences are referred to as hidden divergences and are more common on the stochastic oscillator, a type of price-based contrarian indicator. The anecdote of hidden divergences is that they are a trend-following technique that is based on a contrarian indicator. First, let us understand and code the stochastic oscillator.

The stochastic oscillator is a famous contrarian indicator based on 3 pieces of data, the close, the high, and the low. It is based on the normalization function where the output is a value bound between 0 and 100. The concept revolves around subtracting the minimum value in a certain lookback period from the current value and dividing by the maximum value in the same lookback period minus the minimum value (the same in the nominator).

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

The Stochastic Oscillator takes this into another level and uses more information. The formula is as follows:

$$x = \frac{Closing\ Price - Lowest\ Low\ in\ n\ period}{Highest\ High\ in\ n\ period - Lowest\ Low\ in\ n\ period}$$

```
def stochastic(Data, lookback, close, where, genre = 'High-Low'):
    # Adding a column
    Data = adder(Data, 1)
    # Stochastic Oscillator
    if genre == 'High-Low':
        for i in range(len(Data)):
            try:
                Data[i, where] = (Data[i, close] - min(Data[i - lookback + 1:i + 1, 2])) /
(max(Data[i - lookback + 1:i + 1, 1]) - min(Data[i - lookback + 1:i + 1, 2]))
            except ValueError:
                pass
        Data[:, where] = Data[:, where] * 100
        Data = jump(Data, lookback)
    # Simple Normalization
    if genre == 'Normalization':
        for i in range(len(Data)):
            try:
                Data[i, where] = (Data[i, close] - min(Data[i - lookback + 1:i + 1, close])) /
(max(Data[i - lookback + 1:i + 1, close]) - min(Data[i - lookback + 1:i + 1, close]))
            except ValueError:
                pass
        Data[:, where] = Data[:, where] * 100
        Data = jump(Data, lookback)
    return Data
```

The above function gives a choice to either calculate the rolling Stochastic Oscillator given a period that we choose or to calculate the rolling simple normalization which uses only closing prices in the function. To simplify things, we can say that normalization only uses the closing price where values of 100 signify a highest close since a specified lookback period and values of 0

signify a lowest close since a specified lookback period. Similarly, the Stochastic Oscillator's function incorporates the highs and the lows in its function so that a better version is given.

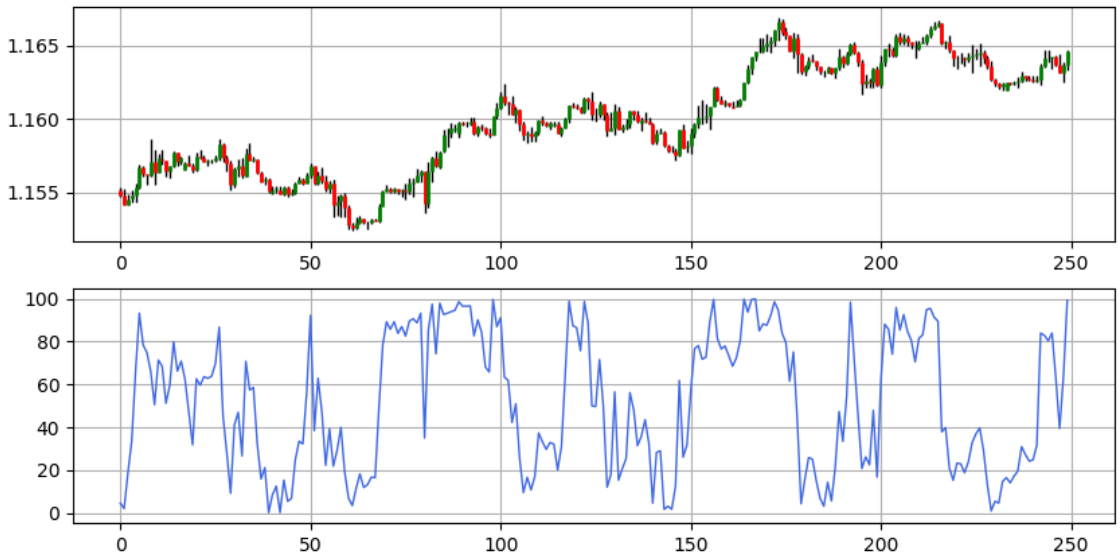



*Figure 83 – Hourly EURUSD values with the 14-hour stochastic oscillator.*

Naturally, the stochastic oscillator is used like the famous relative strength index where we have oversold and overbought levels. An oversold level implies too much selling activity and is generally a buy signal at its simplest form while an overbought level implies too much buying activity and is generally a sell signal at its simplest form. Let us now discuss the divergence method which is another way of using the indicator.

When prices are rising and making new tops while a price-based indicator is making lower tops, a weakening is occurring and a possibility to change the bias from long to short can present itself. That is what we call a normal divergence. We know that:

- When prices are making higher highs while the indicator is making lower highs, it is called a bearish divergence, and the market might stall.
- When prices are making lower lows while the indicator is making higher lows, it is called a bullish divergence, and the market might show some upside potential.

A hidden divergence can be seen as the mirror view of a normal divergence:

- When prices are making lower highs while the indicator is making higher highs, it is called a bearish hidden divergence, and the market might stall and continue lower.
- When prices are making higher lows while the indicator is making lower lows, it is called a bullish hidden divergence, and the market might show some upside potential.

*Figure 84 – Hourly USDCAD values with the 14-period Stochastic oscillator.*

Let us try to code a simple algorithm that tries to detect hidden divergences. Note that we need an OHLC array of historical data as always. The hidden divergence technique is a local trend-confirmation strategy, as opposed to the normal divergence which is a contrarian strategy. They confirm the recent trend (not the overall direction). Hidden divergences tend to occur with the stochastic oscillator more than others.

```
def hidden_divergence(Data, indicator, lower_barrier, upper_barrier, width):
    Data = adder(Data, 10)
    for i in range(len(Data)):
        try:
            if Data[i, indicator] < lower_barrier and Data[i - 1, indicator] > lower_barrier:
                for a in range(i + 1, i + width):
                    if Data[a, indicator] > lower_barrier:
                        for r in range(a + 1, a + width):
                            if Data[r, indicator] < lower_barrier and \
                            Data[r, indicator] < Data[i, indicator] and Data[r, 3] > Data[i, 3]:
                                for s in range(r + 1, r + width):
                                    if Data[s, indicator] > lower_barrier:
                                        Data[s, 6] = 1
                                        break
                                    else:
                                        break
                            else:
                                break
                        else:
                            break
                else:
                    break
        except IndexError:
            pass
    for i in range(len(Data)):
        try:
            if Data[i, indicator] > upper_barrier  and Data[i - 1, indicator] < upper_barrier:
                for a in range(i + 1, i + width):
                    if Data[a, indicator] < upper_barrier:
```

```
                for r in range(a + 1, a + width):
                    if Data[r, indicator] > upper_barrier and \
                    Data[r, indicator] > Data[i, indicator] and Data[r, 3] < Data[i, 3]:
                        for s in range(r + 1, r + width):
                            if Data[s, indicator] < upper_barrier:
                                Data[s, 7] = -1
                                break
                            else:
                                break
                    else:
                        break
                else:
                    break
            else:
                break
    except IndexError:
        pass
return Data
```

*Figure 85 – Signal chart on the EURUSD.*

This technique is recommended to be used only as a trend-confirmation factor and should not be the main protagonist of the trading system. The signals generated vary from asset to asset and from time to time and thus dubbed as not highly reliable.

*Figure 86 – Hourly EURUSD values with the 14-period stochastic oscillator.*

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 51.00% | 46.00% |
| Average Gain | $ 1,165.03 | $ 371.36 |
| Average Loss | $ 1,041.11 | $ 369.28 |
| Profit Factor | $ 1.18 | $ 0.85 |
| Expectancy | $ 80.00 | $ -30.00 |
| Risk-Reward | $ 1.12 | $ 1.00 |
| Trades | 314 | 314 |

The results show that the variable holding period strategy is mildly performing well on the EURUSD but right over the edge and hence, nothing significant. The fixed holding period strategy has underperformed suggesting that possibly divergences should not be constrained by time.

## STRATEGY #11 THE RSI NEUTRALITY CROSS

We have already seen the RSI as a contrarian indicator where reversal signals are given around the extremes. We can also use the RSI for trend-following even though it is recommended to use it the first way. The neutrality level on the RSI is 50 and the hypothesis is that any reading above it hints to a bullish momentum while any reading below it hints to a bearish momentum. Therefore, the trading rules are easy:

- **A long position may be opened whenever the 21-period RSI surpasses the 50 level plus a constant.**
- **A short position may be opened whenever the 21-period RSI breaks the 50 level minus a constant.**

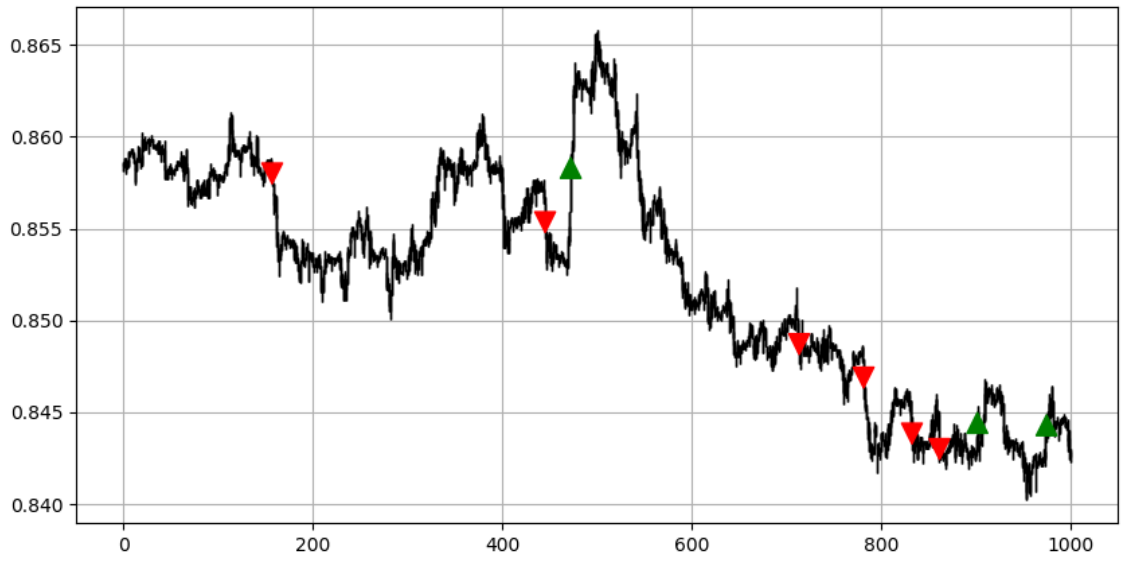



*Figure 87 – Signal chart on the EURGBP.*

The constant is a confirmation level that finally validates the change in the trend. For example, when we see the 21-period RSI of the EURUSD at 51 after

surpassing the 50, we might prefer to wait for a surpass of 53 to validate the signal. Therefore, the confirmation constant is 3.

```
def signal(Data, rsi_col, confirmation, buy, sell):
  Data = adder(Data, 10)
  for i in range(len(Data)):
    if Data[i, rsi_col] > 50 + confirmation and \
      Data[i - 1, rsi_col] > 50 and \
      Data[i - 2, rsi_col] < 50 and \
      Data[i - 3, rsi_col] < 50 and \
      Data[i - 4, rsi_col] < 50:
        Data[i, buy] = 1
    elif Data[i, rsi_col] < 50 - confirmation and \
      Data[i - 1, rsi_col] < 50 and \
      Data[i - 2, rsi_col] > 50 and \
      Data[i - 3, rsi_col] > 50 and \
      Data[i - 4, rsi_col] > 50:
        Data[i, sell] = -1
  return Data
```

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 43.00% | 49.00% |
| Average Gain | $ 1,072.00 | $ 342.28 |
| Average Loss | $ 982.00 | $ 414.46 |
| Profit Factor | $ 0.81 | $ 0.78 |
| Expectancy | $ -100.00 | $ -40.00 |
| Risk-Reward | $ 1.09 | $ 0.83 |
| Trades | 412 | 412 |

The table shows the results of using a 21-period RSI with a constant of 7. The results show that the variable strategy does not function well on the EURUSD and while the fixed period showed less average gain, it did compensate by having a higher hit ratio and overall profitability as evidenced by the expectancy metric. This indicates that the surpass or break of the neutrality level is probably short-lived and not sustainable.

## STRATEGY #12 THE VAMA FLIP

This is a simple crossover strategy on the volatility-adjusted moving average where we assume a change of trend whenever there is a cross between the market price and the VAMA. The trading rules will be as follows:

- **A long position may be opened whenever the market surpasses its VAMA(3, 233).**
- **A short position may be opened whenever the market breaks its VAMA(3, 233).**

```
def signal(Data, close, vama_col, distance, buy, sell):
  Data = adder(Data, 10)
  for i in range(len(Data)):
    if Data[i, close] > Data[i, vama_col] and Data[i - 1, close] < Data[i - 1, vama_col]:
      for a in range(i + 1, len(Data)):
        if Data[a, close] > Data[a, vama_col] + distance:
          Data[a, buy] = 1
          break
        elif Data[a, close] < Data[a, vama_col]:
          break
    elif Data[i, close] < Data[i, vama_col] and Data[i - 1, close] > Data[i - 1, vama_col]:
      for a in range(i + 1, len(Data)):
        if Data[a, close] < Data[a, vama_col] - distance:
          Data[a, sell] = -1
          break
        elif Data[a, close] > Data[a, vama_col]:
          break
  return Data
```

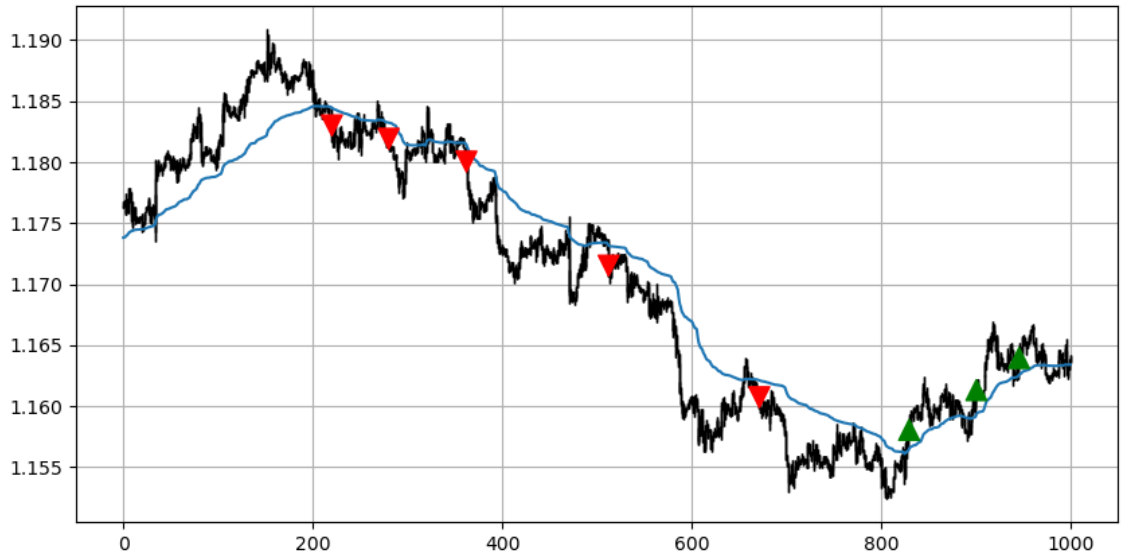



*Figure 88 – Signal chart on the EURUSD.*

The signal function is different than regular crossovers. It is based on the concept of confirmation before validating a signal. Therefore, it will wait until the market surpasses or breaks the VAMA and then will also wait until the market is a specified distance away from the VAMA before triggering the signal. An extra condition is imposed where the market should not re-integrate the VAMA while awaiting to traverse the minimum distance. Here is an example of a bullish crossover:

- The EURUSD pair is at 1.2000 while the VAMA is at 1.1210, the next price bar closes at 1.1220 while the VAMA is at 1.1215. The first condition is satisfied and now we are awaiting a minimum difference (Let us say 20 pips) between the two to be able to proceed. Let us consider that the next price bar closes at 1.2240 while the VAMA is at 1.1220. This constitutes a valid bullish signal if during the whole process the EURUSD did not dip below the VAMA after surpassing it the first time at 1.1220.

```
lookback_volatility_short = 3

lookback_volatility_long  = 233

my_data = volatility_adjusted_moving_average(my_data, lookback_volatility_short,
lookback_volatility_long, 3, 4)

my_data = signal(my_data, 3, 4, 0.0020, 6, 7)
```

The above creates the signals using a VAMA(3, 233) and a minimum distance of 20 pips. Let us check the back-testing results on the EURUSD using the strategy.

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 43.00% | 51.00% |
| Average Gain | $ 632.67 | $ 378.01 |
| Average Loss | $ 510.69 | $ 402.25 |
| Profit Factor | $ 0.92 | $ 0.96 |
| Expectancy | $ -20.00 | $ -0.00 |
| Risk-Reward | $ 1.23 | $ 0.94 |
| Trades | 1344 | 1344 |

## STRATEGY #13 THE RSI & THE SUPERTREND

Having already seen the two behemoths of technical analysis, we can easily combine them together in order to follow the trend. Naturally, the protagonist here will be the SuperTrend indicator since it will be used to signal the trend flip. The RSI will act as a filter to confirm the flip. The trading conditions are as follows:

- **A long position may be opened whenever the market surpasses its SuperTrend indicator while the RSI is above 50.**
- **A short position may be opened whenever the market breaks its SuperTrend indicator while the RSI is below 50.**

As a reminder, the SuperTrend indicator uses the average true range and a few conditions to follow the price and to detect changes in the trend, while the RSI is a contrarian indicator that uses overbought and oversold levels to detect consolidations and accelerations.

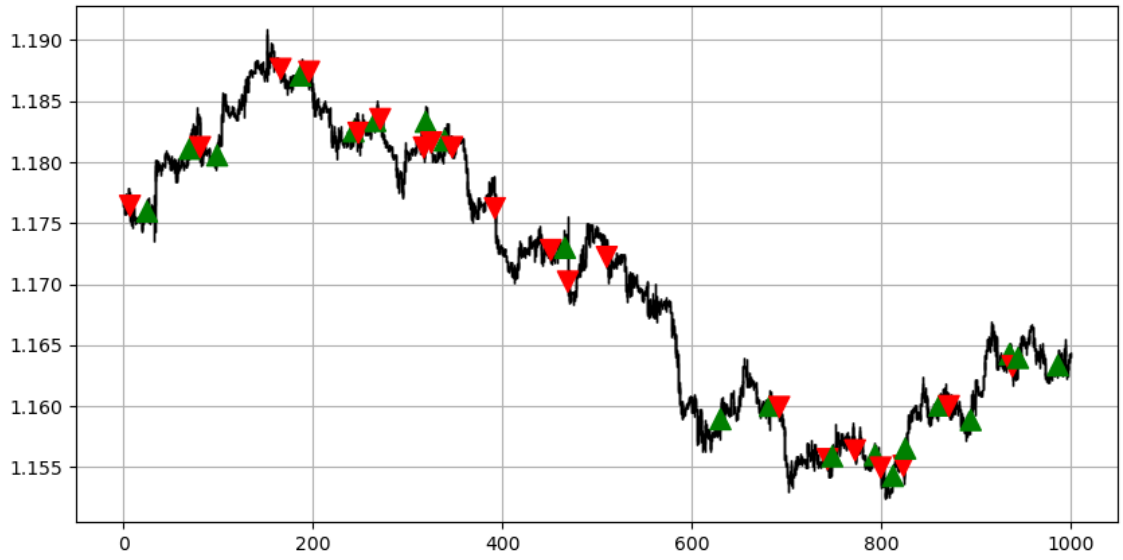



*Figure 89 – Signal chart on the EURUSD.*

The function used to generate the signals can be written down as the below.

```
def signal(Data, close, super_trend_col, rsi_col, buy, sell):
  Data = adder(Data, 10)
  for i in range(len(Data)):
    if Data[i, close] > Data[i, super_trend_col] and \
      Data[i - 1, close] < Data[i - 1, super_trend_col] and \
      Data[i, rsi_col] > 50:
        Data[i, buy] = 1
    elif Data[i, close] < Data[i, super_trend_col] and \
      Data[i - 1, close] > Data[i - 1, super_trend_col] and \
      Data[i, rsi_col] < 50:
        Data[i, sell] = -1
  return Data
```

The previous chart was created based on a 14-period RSI and a 10-period SuperTrend indicator with a multiplier of 2. We can tweak these parameters to optimize our strategy. For example, the next chart has the following parameters.

```
lookback_rsi = 14
lookback_supertrend = 20
multiplier = 3
```

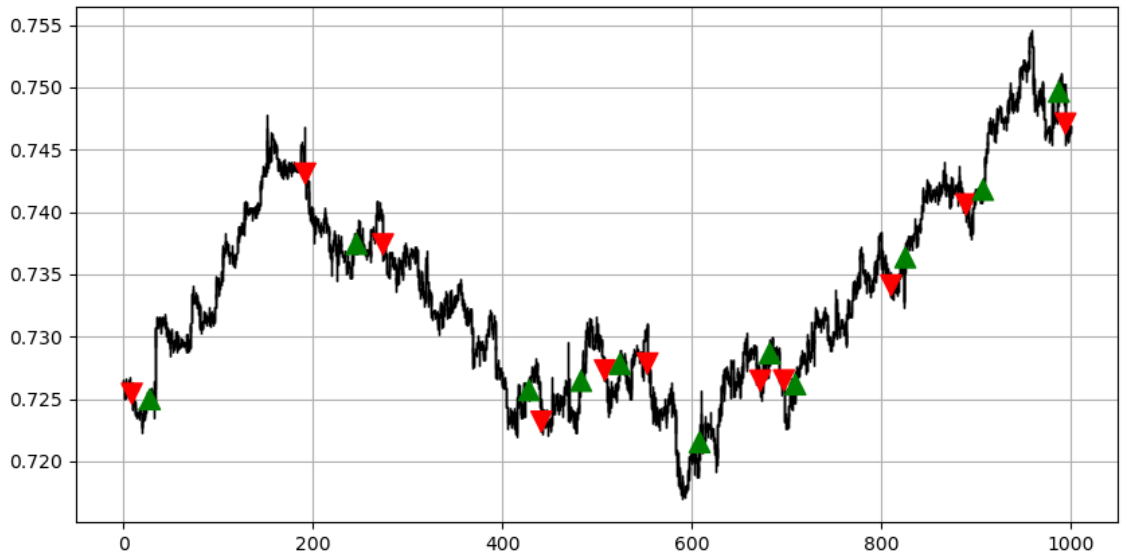



*Figure 90 – Signal chart on the AUDUSD.*

Now, it is time to check the results from trading this strategy.

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 37.00% | 51.00% |
| Average Gain | $ 741.62 | $ 353.89 |
| Average Loss | $ 439.40 | $ 349.20 |
| Profit Factor | $ 1.01 | $ 1.04 |
| Expectancy | $ -0.00 | $ 10.00 |
| Risk-Reward | $ 1.68 | $ 1.01 |
| Trades | 1665 | 1665 |

The results show that the variable holding period strategy does not function well on the EURUSD and while the fixed period showed less average gain, it did compensate by having a higher hit ratio and overall profitability as evidenced by the expectancy metric which is not very far either from the expectancy of the variable period strategy. The risk-reward of the variable period was superior to the one with fixed period.

## STRATEGY #14 THE CATAPULT

Just like a catapult, this technical indicator is based on three systems which deal with Volatility (Acting as the fulcrum), Momentum (Acting as a payload propeller, and a Directional Filter (Acting as the support). The aim is to get a signal that predicts an acceleration of volatility based on previous patterns as well as the direction following this volatility. Therefore, we are aiming to answer an important question, when will the market move? and where?

For the volatility part, the Catapult uses the 21-period relative volatility index to predict the big moves. There are many ways to measure volatility such as the average true range, the mean absolute deviation, and the standard deviation. We will use standard deviation to create the relative volatility index. The most basic type of volatility is the standard deviation. It is one of the pillars of descriptive statistics and an important element in some technical indicators such as the famous Bollinger bands. But first, let us define what variance is before we find the standard deviation. Variance is the squared deviations from the mean (a dispersion measure), we take the square deviations so as to force the distance from the mean to be non-negative, finally we take the square root to make the measure have the same units as the mean, in a way we are comparing apples to apples (mean to standard deviation standard deviation). Variance is calculated through this formula:

$$\sigma^2 = \frac{\sum(x - \bar{x})^2}{n - 1}$$

Following what we have said, standard deviation is therefore:

$$\sigma = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$$

The exciting job now is to turn the historical volatility otherwise known as the historical standard deviation into a measure that allows us to predict whether there will be an outburst of volatility or not. This will simply be done by applying the formula of the Relative Strength Index — RSI on the values of the standard deviation.

```
def relative_volatility_index(Data, lookback, close, where):
    # Calculating Volatility
    Data = volatility(Data, lookback, close, where)
    # Calculating the RSI on Volatility
    Data = rsi(Data, lookback, where, where + 1)
    # Cleaning
    Data = deleter(Data, where, 1)
    return Data
```

For the momentum part, the Catapult uses the 14-period Relative Strength Index to predict the likelihood direction of the move. As a reminder, the RSI is bounded between 0 and 100 where we can identify two levels from where contrarian signals are generated:

- The oversold level 30 is where the market is perceived to have gone down too much and a bullish reaction is to be expected.
- The overbought level 70 is where the market is perceived to have gone up too much and a bearish reaction is to be expected.

Another interesting way to use the RSI is by comparing the current reading to the 50 level. If the RSI is above 50, then a likely bullish momentum is in place and if it is below 50, then a likely bearish momentum is in place.

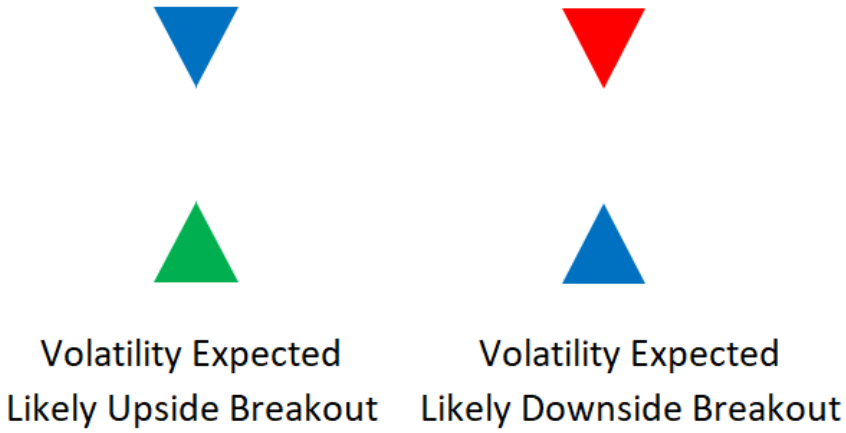
For the directional filter part, the Catapult uses the 200-period simple moving average to keep us in the direction of the trend. This serves as a sanity check and to stack more odds with us. Now, we will build the Catapult indicator.

The indicator is based on a healthy mix of the above three indicators where:

- The 21-period relative volatility index will give the first trigger by stating that as of now, there will be above average volatility and therefore, possible a directional move.
- The 14-period relative strength index will give the likelihood direction of the move by saying that if the reading is above 50, then the move is likely bullish while if the reading is below 50, then the move is likely bearish.
- The 200-period simple moving average will add conviction to the likelihood direction of the move. If the market is above the 200-period moving average, then we understand that there is bullish pressure and quite likely a continuation to the upside. Similarly, if the market is below the 200-period moving average then we understand that there is bearish pressure and quite likely a continuation to the downside.

```
lookback_rvi = 21
lookback_rsi = 14
lookback_ma  = 200
my_data = ma(my_data, lookback_ma, 3, 4)
my_data = rsi(my_data, lookback_rsi, 3, 5)
my_data = relative_volatility_index(my_data, lookback_rvi, 3, 6)
```

The Catapult is an overlay indicator with two handles. The first shows blue and green arrows which signify a buy signal and the second shows blue and red arrows which signify a sell signal.




The below chart shows a real-life example on the most recent values on the hourly EURUSD. Whenever we see a blue arrow, it means that volatility is on the verge of exploding. The green or red arrow point to the direction of the move.

*Figure 91 - Signal chart on the EURUSD.*

```
def signal(Data, rvi_col, signal):
    Data = adder(Data, 10)
    for i in range(len(Data)):
        if Data[i,     rvi_col] < 30 and \
            Data[i - 1, rvi_col] > 30 and \
            Data[i - 2, rvi_col] > 30 and \
            Data[i - 3, rvi_col] > 30 and \
            Data[i - 4, rvi_col] > 30 and \
            Data[i - 5, rvi_col] > 30:
                Data[i, signal] = 1
    return Data
```

The previous function shows that whenever the RVI is showing a value of less than 30, we should expect volatility. Then, we can form our strategy by adding the RSI and the moving average. Let us take a look at the simple back-testing results on the hourly values of the EURUSD since 2011.

| | Variable Period | Fixed Period |
|---|---|---|
| Hit Ratio | 47.00% | 49.00% |
| Average Gain | $ 1,117.78 | $ 412.03 |
| Average Loss | $ 1,133.76 | $ 376.50 |
| Profit Factor | $ 0.88 | $ 1.06 |
| Expectancy | $ -80.00 | $ 10.00 |
| Risk-Reward | $ 0.99 | $ 1.09 |
| Trades | 386 | 386 |

The results show that the strategy does not function well on the EURUSD and while the fixed period showed less average gain, it did compensate by having a slightly higher hit ratio and overall profitability as evidenced by the expectancy metric. This indicates that the blast moves from the volatility squeeze do not last forever. The fixed period also acted as a shield as shown in

the reduction of the average loss between the two strategies. The recommendation here is to tweak the parameters and to add more sophisticated risk management.

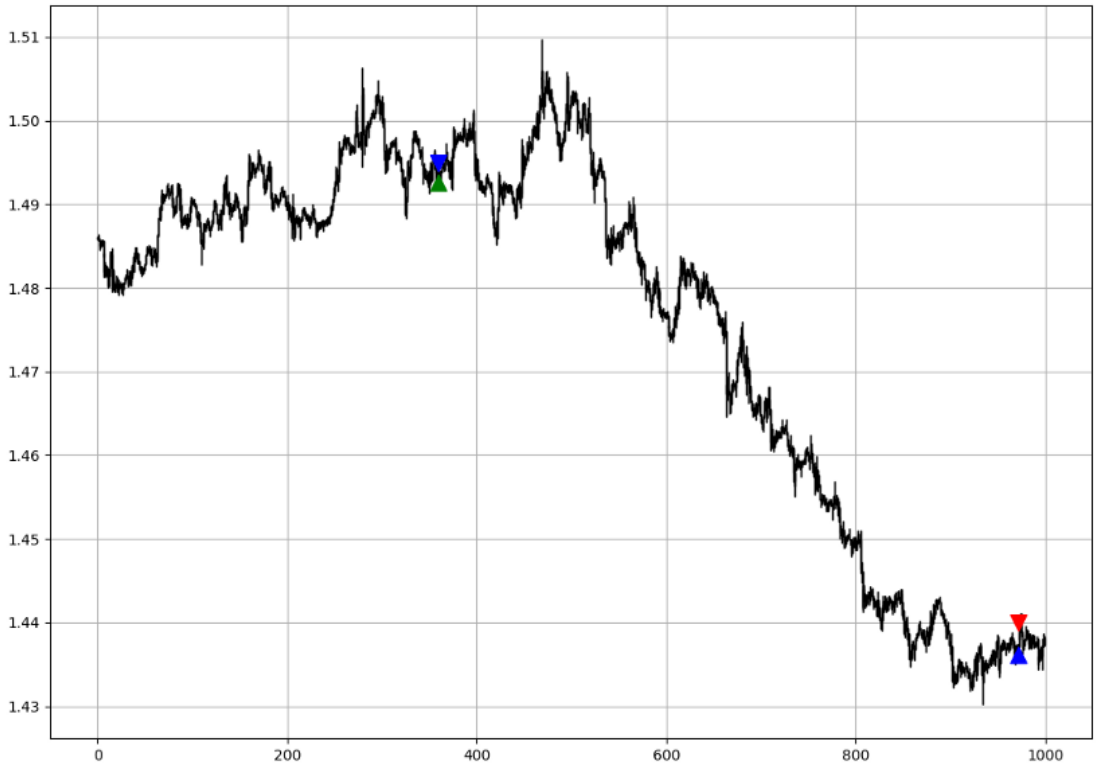



*Figure 92 - Signal chart on the EURCAD.*

## STRATEGY #15 VALHALLA

Discretionary trading is done manually without algorithms and with flexible rules chosen subjectively by the trader. In other words, the trader based on her own past experiences and knowledge accumulation, chooses to follow a set of changeable trading rules that are not necessarily objective without feeling the need to do aby Back-testing due to the historical success of the strategy.

Scientifically, this is a flawed method but we all either do it or did it at some point in time. However, some traders who act on their personal judgement are successful, but the reality is that this is a number's game, and we must stack all the odds with us so that we do not fall inside the 95% who lose money eventually. Stacking the odds is research-based that rely on a huge set of data and validation.

Personally, I have used in the past many old folk discretionary strategies and while some have shown good results, it is impossible to know whether it was due to luck or due to superior rules. I will present the one that made an impact on me; however, I will break the convention of this book by not coding the strategy but presenting its rules. Feel free to code it if you like, but its subjective parts make it unlikely to be 100% back-testable.

So, what is the use of presenting a strategy to the public without hard data? Well, we would all be lying if we say we have never typed "Top Trading Strategies" or "The Best Trading Strategy" in Google and spent hours visualizing these strategies in our charts. This is one strategy that we are unlikely to find online even if it is simple and straightforward. It is to be used on the FX market.

The tools necessary for this strategy are:

- **A 200-period exponential or simple moving average (optional).**
- **A short-term Stochastic indicator set at a lookback period of 5.**
- **A mid-term Stochastic indicator set at a lookback period of 14.**
- **A Fibonacci expansion tool to measure targets and potentials.**

Notice that in the last step, we have mentioned expansion and not retracement. Indeed, the Fibonacci expansion tool is different from the conventional retracement tool in that they help us project the amplitudes. So, naturally, one can use the Fibonacci retracements for determining until what level prices may react and then use the expansion tool to see until where the trend may continue. The common Fibonacci expansion levels used are 61.8%, 100%, and 161.8%. The 200-period EMA is only used to confirm the trend if there are any doubts with the direction as we will only initiate buy orders during up trends and vice versa.

The first step is to look for a trending series either through a moving average or through visual interpretation. I tend to prefer long-term moving averages such as 100 and 200 so as to be sure of the pre-established trends. During lucky times, the moving averages can provide an extra layer of support/resistance level from where we initiate our trigger. As long as we are in a bullish trend, we will only be looking at bullish opportunities. The next step seeks to find the end of the corrections/consolidations by establishing intervals. How are we going to do that? Simple, by using two Stochastic indicators, we will find the points where both indicators show extreme values. Each two successive extremes form the interval that we will use to initiate our orders, calculate the potential target, and measure the risk (where to put our stop).

Now, we can move on to the next step after having found two successive Stochastic extremes that gave us the bullish signal (Remember: we are only looking for bullish signals within this bullish trend). This is where the Fibonacci expansion tool comes in. How is the potential calculated? Well, we take the interval, starting from the low of the first candle, project it to the high of the highest candle inside the interval by using the low of the last candle in the interval. A bullish trend will have the low of the first candle coinciding with the first oversold zone of the Stochastic, the final part of the Expansion tool will also coincide with the low of the last candle occurring with the second oversold zone. The middle part is the highest high inside the whole Fibonacci expansion pattern. The 61.8% projection can be thought of as the first target. A first target can be where we choose to close out 50% of our position in order to secure some profits. As I am a fan of the simple basic risk management principle that says never to risk more than you expect, I like to use a 2:1 risk-reward ratio. This means that if I expect to gain $2, I would risk losing $1.

The best word to describe this strategy is that it is 'improvised'. Understandably, a pure technical strategy that combines two or more indicators together in man-made measures and rules does not make much sense. In investing and trading we are all looking for something that not only works, but it also has to make economic sense. In technical analysis, this can pose a problem sometimes. However, the strategy per se does not make much sense as technical analysis can at least be explained by the psychological behaviors and market participants.

# CONCLUSION

Trend-following remains an art as valuable as contrarian trading and while this book discussed many indicators and strategies, it is far from being exhaustive. Undoubtably, the point of the book was not to present all trend-following topics but to simply have a general view which can help the reader get ideas and apply them. I must point out that the strategies presented are not to be used in their raw form because they must be optimized for every market and must have solid risk management rules so that they become more stable by time. We are mostly concerned by the ability to profit from the on-going trend not really by timing it exactly as time in the market is more important than timing the market. The next edition of the book will cover contrarian strategies and will have the same aim of this book: Stimulate ideas and creative thinking so as to give technical analysis a new face based on objective measures and filtered indicators.

***- Sofien Kaabar***