Leonard De Vinci Graduate School Of Engineering

Time series Analysis – Application to trading Dynamic Time Warping and applications

Students :
Nathan Bonneau
Ibrahim Chakchouk
Mohammed Drici
Thomas Robert
Maxime Varacca
Julien Zheng

Supervisor:
Léa Beudin

Contents

1 General introduction

1.1 Preamble

First introduced in 1960 in the field of voice recognition, and then in *M. Müller's* 2007 research paper, Dynamic Time Warping (often referred to as DTW) is a technique developed to find the optimal alignment between two time series. This technique is generally applied in the field of speech recognition, as well as in the simulation of movements by an individual. The aim of DTW is to measure the similarity between two time sequences (A and B, for example), which are similar but vary more or less in speed. DTW is useful for identifying which parts of sequence A are faster or slower than the other time sequence (B). Using DTW, we can visualize by compressing or decompressing certain parts of sequence A to try and create a new one (sequence C) that will be as close as possible to sequence B.

Various works have been carried out to apply DTW in the field of finance. For example, in 2014, *A. Tsinaslanidis* defined DTW as a similarity measure and studied its relationship with Spearman and Pearson correlation coefficients. In addition, *P. Puspita* developed a clustering method based on DTW in 2020. In our study, we propose a new approach by exploiting the particularities of DTW to develop a trading strategy applied to currency pairs on the FOREX market.

The aim of our study is to harness the power of DTW to measure the similarity between two time series and thus build buy or sell positions on currencies, while ensuring the profitability of the strategy. To our knowledge, there is no existing work on this subject, although this technique seems to be a good candidate for generating positions by studying the behavior of a time series in relation to another series that is highly correlated. We would therefore like to explore this approach in order to develop an innovative trading strategy based on DTW in the currency market.

We will begin by deepening our understanding of the DTW algorithm, implementing it and evaluating its effectiveness. We will then present our general idea for the strategy, setting up the basic functions needed to implement it. We will then detail our methodology, present the code for our strategy and propose an improvement using a well-known method. Finally, we will share the results obtained and the optimal parameters identified to maximize the strategy's performance.

1.2 DTW operation and rules

As mentioned earlier, DTW is an algorithm capable of measuring the similarity between two time series. However, it is not simply a classic Euclidean correspondence algorithm. It performs a kind of 'deformation' of the x-axis (which generally corresponds to time in the field of finance).

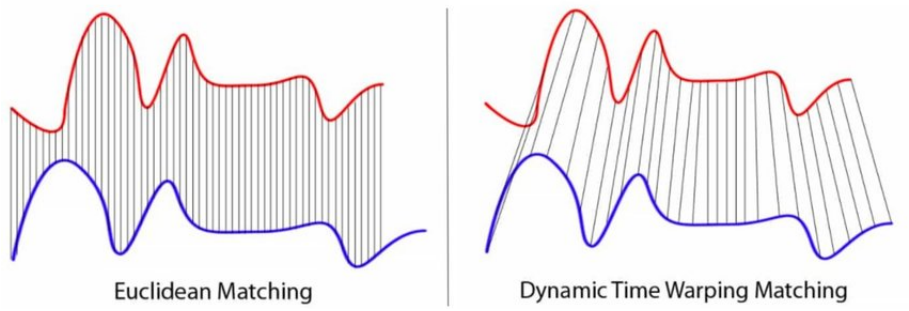

Figure 1: Matching euclidien

With DTW, we can even study the similarity between different time sequences that do not have the same length, unlike Euclidean matching. On the graph above, we can see that the DTW model associates each peak in the blue time series with those corresponding to the red time series, regardless of their position on the x-axis, while complying with a number of rules:

- Each index in the first sequence must be associated with one or more indices in the other sequence, and vice versa.
- The first (resp. last) index of the first sequence must be associated with the first (resp. last) index of the other sequence.
- The correspondence between the indices of the first sequence and the indices of the other sequence must be monotonic, i.e. increasing. These rules enable DTW to find the optimum alignment between the time sequences, taking into account variations in speed and delay between the two series

2 Implementing DTW on Python

2.1 Importing data from Yahoo Finance

```python
def if_data(pair) :
  sg = pair + "=X"
  data = yf.Ticker(sg)
  dataDF = data.history(period='1m', start='2023-11-20', end='2023-11-30', interval='1h')
  array10 = dataDF.index.to_numpy()  # Time
  array1 = dataDF['Close'].to_numpy()  # Price
  return array1

array1 = if_data("GPBUSD")
array2 = if_data("EURUSD")
```

Figure 2: Function for importing data from Yahoo Finance

We use the Yahoo Finance library to retrieve data between 2023/11/20 and 2023/11/30 over a period of 1 minute. We return only an array with the price values set as parameters.

2.2 Standardizing our data

```python
def normalize(array):
    min_val = np.min(array)
    max_val = np.max(array)
    normalized_array = (array - min_val) / (max_val - min_val) #min-max normalization
    return normalized_array
```

Figure 3: Function for normalizing a table

It is essential to normalize both sets of data before applying DTW. This ensures consistent peaks and puts the data on the same scale. In addition, normalization reduces noise and potential errors.

2.3 DTW Algorithm

2.3.1 Absolute distance calculation

```python
#Cost matrix which will have all the manhattan distance between our points
def cost_matrix_compute(a,b):
  cost_matrix = np.zeros((len(a), len(b)))
  for i in range(len(a)):
      for j in range(len(b)):
          cost_matrix[i, j] = abs(a[i] - b[j]) # Manhattan distance
  # Cost_matrix has a dimension of len(a)*len(b)
# We now create a matrix which will have the accumulated distance
  acc_cost_matrix = np.zeros((len(a), len(b)))
  acc_cost_matrix[0, 0] = cost_matrix[0, 0]
  for i in range(1, len(a)):
      acc_cost_matrix[i, 0] = cost_matrix[i, 0] + acc_cost_matrix[i-1, 0]
  for j in range(1, len(b)):
      acc_cost_matrix[0, j] = cost_matrix[0, j] + acc_cost_matrix[0, j-1]
  for i in range(1, len(a)):
      for j in range(1, len(b)):
          acc_cost_matrix[i, j] = cost_matrix[i, j]
          + min(acc_cost_matrix[i-1, j], acc_cost_matrix[i, j-1], acc_cost_matrix[i-1, j-1])

  # The idea is that acc_cost_matrix[i,j] will contain the minimum cumulative cost to reach point a[i] from point b[j]
```

Figure 4: Section for calculating the absolute distance and then the DTW cost matrix

The *cost_ matrix_ compute()* function is used to calculate the cost matrix associated with the DTW. This is a central element in our project and we'll illustrate it with an example to make it easier to understand. Consider two time series:

Time series A = [1,3,4,9,8,2,1,5,7,3] and Time series B = [1,6,2,3,0,9,4,3,6,3].

The purpose of the *cost_ matrix* is to contain the absolute distances needed to fill the final cost matrix: *acc_ cost_ matrix*. It has a number of rows equal to the number of elements in time series A (10 in this example) and a number of columns corresponding to the number of elements in time series B (also 10). For ease of understanding, it is simpler to visualize the *cost_ matrix* as a schematic representation where each row index corresponds to the elements of time series A, and each column index corresponds to the values of time series B. The first values are placed at the bottom left of the matrix. The

first step in the code is to fill this matrix using the absolute distance, which corresponds to the absolute value of the difference between the elements of time series A and those of time series B.

For example, for row i and column j : cost_matrix$[i, j] = |A[i] - B[j]|$ In this way, the *cost_ matrix* will contain the absolute distances required for the subsequent calculation of the *acc_ cost_ matrix*.

2.3.2 Calculation of the DTW cost matrix

We can now fill in the *acc_ cost_ matrix* as follows: simply take the absolute distance of the square calculated previously and add the minimum of the 3 neighboring squares (the neighboring square to its left, the neighboring square below it or the neighboring square diagonally below it). The calculation is different for squares in the first column or the last row.

Formula for filling in the first column:

$acc\_cost\_matrix[i, 0]$=$cost\_matrix[i, 0]$+$acc\_cost\_matrix[i - 1, 0]$

Formula for filling in the last line :

$acc\_cost\_matrix[0, j]$=$cost\_matrix[0, j]$+$acc\_cost\_matrix[0, j - 1]$

General formula :

$acc\_cost\_matrix[i, j]$=$cost\_matrix[i, j]$+min($acc\_cost\_matrix[i-1, j]$, $acc\_cost\_matrix[i, j-1]$, $acc\_cost\_matrix[i - 1, j - 1]$)

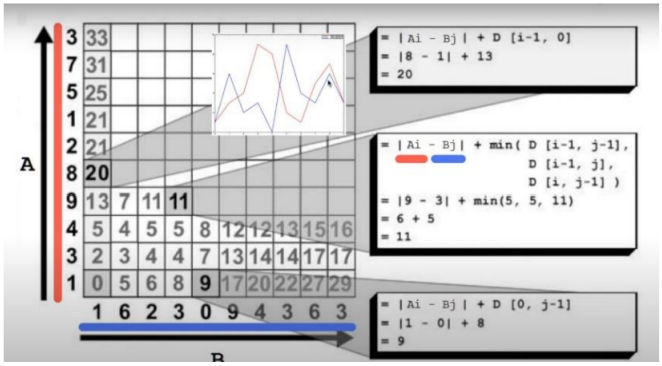

Figure 5: Method for filling *acc_ cost_ matrix*

2.3.3 Select the shortest path

```
path = [(len(a)-1, len(b)-1)]
i = len(a) - 1
j = len(b) - 1
while i > 0 or j > 0:
    if i == 0: # In this situation we no longer have a choice
        j -= 1
    elif j == 0: # In this situation we no longer have a choice
        i -= 1
    else:
        min_cost = min(acc_cost_matrix[i-1, j], acc_cost_matrix[i, j-1], acc_cost_matrix[i-1, j-1])
        if min_cost == acc_cost_matrix[i-1, j]:
            i -= 1
        elif min_cost == acc_cost_matrix[i, j-1]:
            j -= 1
        else:
            i -= 1
            j -= 1
    path.append((i, j)) # We add the couple (i,j) to our future shortest path
path.reverse() # As we started from the end, we must return our path
```

Figure 6: Code to find the shortest path

The *path* variable will contain the shortest path. To construct this variable, we start at the top right of the matrix, i.e. at the end of our two time series (corresponding to the cell *acc_cost_matrix[0, len(B)-1]*. As shown in the example below, from the cell at the top right of the matrix, we will select the smallest cell from among its three neighbours: its neighbour on the left, its neighbour below or its neighbour in the diagonal at the bottom left. Once the neighbouring cell has been determined, we add it to our path variable and repeat the same process. The aim is to reach the box at the bottom left of our matrix, corresponding to the first value in our two time series (i.e. the box *acc_cost_matrix[len(A)-1, 0]*).

At the end, all we have to do is invert the *path* variable to get the path in the right direction.

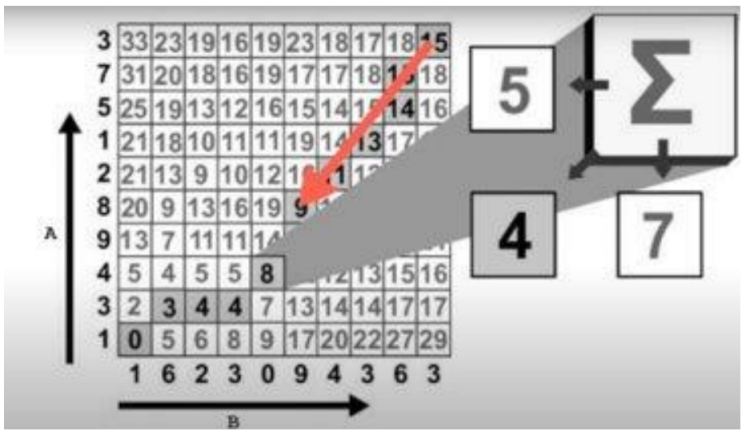

Figure 7: Method for finding the shortest path

2.3.4 Represent the deformed version graphically

```
deformed_a = [a[p[0]] for p in path[:]]
```

Figure 8: Representation of the deformed version of A using our code

Using *path*, we can now represent the deformed version of time series A. All we have to do is select the first element of each tuple in *path* and set it as an index of time series A. In this way, we obtain a new *deformed_ a* time series which corresponds to the time series compressed or decompressed in certain places so that it corresponds as closely as possible to time series B. For a very simplified example, we obtain the graph below:

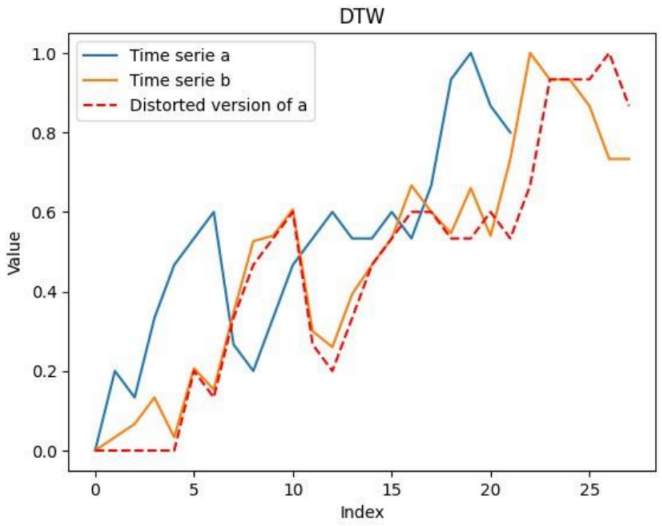

Figure 9: Writing the distorted version of A

The curve in red (*deformed_ a*) is a compressed/decompressed version of time series A to match time series B (orange) as closely as possible.

2.4 Evaluate the quality of DTW

Indeed, it is important to estimate the performance and success of our DTW. Correlation was our first choice, but given that prices are not considered to be stationary, it is ultimately meaningless.

The RMSE metric seemed a more appropriate choice for measuring the squared deviation between the distorted time series of A and the time series of B. A figure close to 0 indicates good phase alignment, while an RMSE close to 1 indicates poor DTW performance. The formula for RMSE is :

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(deformed\_a[i]-b[i])^2}{n}}$$

```
difference = b - deformed_a
squared_difference = difference ** 2
rmse = np.sqrt(np.mean(squared_difference))
print("RMSE:", rmse)
```

Figure 10: Calculation of the RMSE between B and the deformed version of A

3 Preamble to the DTW strategy

3.1 Introduction to the application of DTW in quantitative finance

In quantitative finance, it can be interesting to use Dynamic Time Warping to predict the price of an asset/currency, say EURUSD, from another highly correlated currency pair, say GBPUSD. We know that these two pairs have similar movements, but with a slight lag or lead over each other. If on 20/01/2024 at 10:07 a.m. we see a strong positive spike in the GBPUSD and no spike in the EURUSD, we can predict that there will be a strong move in the GBPUSD in the next few seconds/minutes.

3.2 General idea of the strategy

The DTW works on the basis of two input time series, in our case two currency pairs: for example the GBPUSD as time series A and the EURUSD as time series B. It is important that these two pairs are sufficiently linked/correlated for the DTW to make sense.

The general aim is to anticipate an upward or downward movement in the GBPUSD pair based on changes in its correlated pair (in our example, the EURUSD). In this way, we can take positions (short or long) on the GBPUSD by identifying whether it is ahead or behind the EURUSD curve.

Here are the main steps of our strategy:

1) Every minute, we update our metrics (average price variation, average positive variation, average negative variation, largest positive variation, smallest negative variation, etc.).

2) Calculation of the DTW matrix, deforming pair A (GBPUSD) to match pair B (EURUSD) as closely as possible.

3) From this matrix, we identify whether time series A (GBPUSD) is leading (decompression) or lagging (compression). With these elements, we can make a decision to place a

trade.

4) If the EURUSD has a positive movement above a certain threshold and the GBPUSD is lagging, I buy GBPUSD (LONG).

If the EURUSD experiences a negative movement above a certain threshold and the GBPUSD is lagging, I sell the GBPUSD (SHORT).

5) I set my take profits (stop gains) and stop losses using the metrics calculated in step 1).

3.3 Choice of work

The proposed trading strategy was created on the EURUSD and GBPUSD (or AUDUSD) currency pairs by distorting the GBPUSD (or AUDUSD) pair. Indeed, from our calculations with the RMSE and our discussions with professionals, we have learned that these two pairs undergo similar variations.

The data used is per-minute data. For the sake of simplicity, once a position has been taken (short or long GBPUSD), we cannot carry out other trades. Furthermore, we only work on buying or selling the GBPUSD. However, it would be possible, using the same logic as in the code, to add a buy or sell functionality for the EURUSD. We have assumed that there is no delay when buying or selling a currency pair.

3.4 Functions required for loading financial data

3.4.1 Import from Yahoo Finance

```
def yfv2(pair, i, years, months, day, h, min) :
  sg = pair + "=X"
  data = yf.Ticker(sg)
  start_datetime = datetime.datetime(years, months,day, h, min)
  end_datetime = start_datetime + datetime.timedelta(minutes=i)
  dataDF = yf.download(tickers= sg, start=start_datetime, end=end_datetime, interval='1m')
  array10 = dataDF.index.to_numpy()  # Time
  array1 = dataDF['Close'].to_numpy()  # Price
  return array1
```

Figure 11: Function for importing from Yahoo Finance

The *yfv2()* function is an improved version of our first *if_ data()* function, designed to return an array of currency pair prices from Yahoo Finance. It now takes several parameters as input:

- pair : a character string corresponding to the currency pair, e.g. "AUDUSD" etc.
- i : the number of minutes required.
- years : the year of the departure date.
- months : the month of the departure date.
- days : the day of departure.

- h : the time of departure.
- min : the minute of the departure date.

The function will return an array of i elements, corresponding to the rates of the specified currency pair at the start date and time specified in the parameters.

3.4.2 Import from Excel

Functions of type *get_day_close_pricesEUR()* can also be used to load financial data, but this time from an Excel file written in the form of a csv file, like this one:

Time	Open	High	Low	Close	Volume
2023-10-29 21:18:00	0.63374	0.63376	0.63324	0.63329	34
2023-10-29 21:19:00	0.63329	0.63395	0.63328	0.63395	23
2023-10-29 21:20:00	0.63396	0.63399	0.63342	0.63351	32
2023-10-29 21:21:00	0.6335	0.63351	0.6334	0.63351	12
2023-10-29 21:22:00	0.63351	0.63351	0.63351	0.63351	1
2023-10-29 21:23:00	0.63347	0.63358	0.63341	0.63348	34
2023-10-29 21:24:00	0.63348	0.6335	0.63348	0.63349	27
2023-10-29 21:25:00	0.63349	0.63349	0.63346	0.63346	13
2023-10-29 21:26:00	0.63346	0.63347	0.63346	0.63347	7

Figure 12: Sample Excel file for importing with the get day close prices function

4 Implementing the DTW strategy

4.1 Strategy function parameters

def *test_strategy()*(nbpoints, years, months, day, h, min, declenchement, decalage, nbpointsDTW, varMin, varMax, montant=1000, fees=0.0001):

The *test_strategy()* function has several parameters:

- nbpoints : corresponds to the number of minutes during which we will exercise the trading strategy.
- years : the year of the departure date.
- months : the month of the departure date.
- days : the day of departure.
- h : the time of the departure date.
- min : the minute of the departure date.
- declenchement : represents a percentage which will be useful for setting the value at which we consider that the variation is significant enough to trigger a trade.
- decalage : represents the number of minutes that time series A (GBPUSD) has in relation to time series B (EURUSD). When this number is too large, we consider that the DTW has malfunctioned, so we set this threshold above which the lag measured by the DTW is not significant.
- nbpointsDTW : represents the number of minutes we will take to carry out our DTW. If nbpointsDTW = 15, we will take the last 15 minutes of time series A (GBPUSD) and

time series B (EURUSD) to calculate the DTW cost matrix.

- varMin : represents a percentage that will be useful for setting the stop loss when going short or long on GBPUSD.
- varMax : represents a percentage which will be useful for setting the take profit when going short or long on GBPUSD.
- montant : initial amount invested in the strategy, by default $1000 .
- fees : transaction fees and bid ask spread, default 0.1 pips.

The beginning of the code defines the main Booleans that will be activated in the event of a short, a long or a trade in progress. We use the *yfv2()* function explained earlier to store our two currency pairs: in *array1vrai*, I store the currency I'm going to distort and take a short or long position on, and in *array2vrai*, the pair I'm going to use as a reference to distort the other.

Next, we use a 'for' loop to simulate the passing of the minutes step by step so that we can backtest our strategy, which is implemented directly in the function. The index (20+i-1) corresponds to the minute we are at in *array1vrai* and *array2vrai*.

array1 and *array2* are two arrays that store the prices of the two currency pairs over the last 20 minutes up to the current minute (20+i-1 in *array1vrai* and *array2vrai*). In *array1* and *array2*, the current minute corresponds to the last element of the array (len(*array1*)-1 and len(*array2*)-1).

```
array1 = array1vrai[i-1:20+i-1]
array2= array2vrai[i-1:20+i-1]
```

Figure 13: Definition of the tables containing the data

4.2 Calculation of metrics

```
# Metric calculation and update
pips_changesB = [(array2[i] - array2[i-1]) / 0.0001 for i in range(1, len(array2))] #1
pips_changesA = [(array1[i] - array1[i-1]) / 0.0001 for i in range(1, len(array1))] #2
positive_pips_changes = [change for change in pips_changesB if change > 0] #3
positive_pips_changes.sort() #4
index = int(len(positive_pips_changes) * declenchement) #5
if len(positive_pips_changes) != 0: #6
  max_value_pc = positive_pips_changes[index] #7
else : max_value_pc = 10 #8
negative_pips_changes = [change for change in pips_changesB if change < 0] #9
negative_pips_changes.sort() #10
index = int(len(negative_pips_changes) *(1-declenchement) #11
if len(negative_pips_changes) != 0: #12
  min_value_pc = negative_pips_changes[index] #13
else : min value pc = -10 #14
```

Figure 14: Calculation of metrics over the last 20 minutes

As explained earlier in the report, it's important to calculate metrics in order to set our order thresholds and measure whether it's worth trading in this situation. All our metrics will be calculated in terms of pips (10.e-4 in most currency pairs). We calculate these metrics over the last 20 minutes.

- (Line 1 and Line 2): We start by calculating the variation between each minute in number of pips and store them in *pips_changesB* (EURUSD) and *pips_changesA*

(GBPUSD).

• (Line 3 and Line 4) : The variable positive *pips_changes* stores all the positive variations of the EURUSD sorted in ascending order.

• (Line 5 to Line 8): Using the *declenchement* variable given as a parameter, we define *index* which corresponds to the index of the list of positive variations of the EURUSD based on a percentage (*declenchement*). This represents the index above which we will have "declenchement percent" of positive variations in the EURUSD. We store the value of this index in *max_value_pc*. The logic is the same but for negative variations, and we store the value of the index in *min_value_pc* (Line 9 to Line 14).

```
if(len(pips_changesA)!=0): #15
  meanA = sum(pips_changesA) / len(pips_changesA) #16
else : meanA =0 #17
positive_pipsA = [pips for pips in pips_changesA if pips > 0] #18
negative_pipsA = [pips for pips in pips_changesA if pips < 0] #19
mean_positiveA = sum(positive_pipsA) / len(positive_pipsA) if positive_pipsA else 0 #20
mean_negativeA = sum(negative_pipsA) / len(negative_pipsA) if negative_pipsA else 0 #21
MaxPositifA = max(positive_pipsA) if positive_pipsA else mean_positiveA #22
MinNegativeA = max(abs(pips) for pips in negative_pipsA) if negative_pipsA  else mean_negativeA #23
array1 = array1vrai[20-nbpointsDTW+i-1:20+i-1] #24
array2 = array2vrai[20-nbpointsDTW+i:20+i-1] #25
```

Figure 15: Continuation of metric calculations

• (Line 15 to Line 17) :*MeanA* corresponds to the average change in pips of time series A (GBPUSD).

• (Line 18 and Line 19): Similar to lines 3 and 4, but this time we store all positive GBPUSD variations in *positive_pipsA* and negative variations in *negative_pipsA*.

• (Line 20 and Line 21): *mean_positiveA* corresponds to the average of positive variations in pips of the GBPUSD. Similarly for *mean_negativeA*, but for negative variations.

• (Line 22 and Line 23): *MaxPositiveA* is the largest positive variation in the GBPUSD. *MinNegativeA* is the smallest negative variation of the GBPUSD in absolute value.

• (Line 24 and Line 25): We redefine *array1* and *array2* because we have just calculated the metrics over the last 20 minutes. However, the number defined to carry out the DTW corresponds to the function's input parameter: *nbpointsDTW*.

```
# Case where we don't have any trade
    if trade_en_cours == False:
      a = normalize(array1)
      b = normalize(array2)
      #plot_graph(array1, array2, normalized_array1, normalized_array2)
      pipsA = (array1[len(array1)-1] - array1[len(array1)-2]) / 0.0001
      pipsB = (array2[len(array2)-1] - array2[len(array2)-2]) / 0.0001
```

Figure 16: Check whether a trade is in progress

A first loop with the *trade_en_cours* boolean is used to check whether there is a trade in progress or not. We normalize our two time series to work on a common scale. If no trade is in progress, we can start calculating whether it is worth taking a short or long position on the GBPUSD. We calculate the last change in the number of pips in series A (resp. series B), which we store in *pipsA* (resp. *pipsB*).

4.3 Short Position

4.3.1 Trigger threshold and DTW calculations

```
# Short the pair GBPUSD
if pipsB < (min_value_pc): #26
  # Calculations related to DTW
  path, cost_matrix, acc_cost_matrix = cost_matrix_compute(a,b) #27
  deformed_a = [a[p[0]] for p in path[:]] #28 deformed_a
  x = [p[1] for p in path] #29 index of the deformed_b
  y = [p[0] for p in path] #30 index of the deformed_a
  #plot_deformed(a,b,deformed_a)
  time = len(array2)-1 #31
```

Figure 17: DTW calculation for short

In the case of a potential **short** of the GBPUSD, to start the first calculations, we first check whether the current variation in the EURUSD is sufficiently large (below the *min_value_pc* threshold defined here). If it is, then we perform DTW (Dynamic Time Warping) on our two normalized time series (a and b). Thus, *deformed_a* contains the shortest path (index of the cells in the matrix through which the shortest path passes, applied to our normalized time series A). In line 29, we define the indices of time series A where the shortest path passes, and the same for time series B.

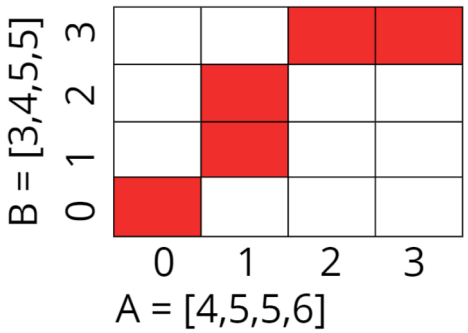

Figure 18: Shortest path diagram

Assume the DTW cost matrix above. We have two time series A and B which take the following values: A = [4,5,5,6] and B = [3,4,5,5]. For simplicity, we have not normalized A and B, but in our code, A and B are normalized. The path in red represents the shortest path in the DTW cost matrix.

path = [(0;0), (1,1),(1;2);(2;3),(3;3)]

x = [0,1,2,3,3]

y = [0,1,1,2,3]

deformed_a = [a[p[0]] for p in path] = [4,5,5,5,6]

Figure 19: Example of path

Path therefore represents the set of coordinates of the shortest path in our matrix. X represents the second coordinate of the shortest path, and Y the first coordinate. Deformed simply corresponds to the second coordinates of the path applied to our time series A.

4.3.2 Identification of DTW compression or decompression moments

```
if(x[time]-y[time]<decalage and x[time]-y[time]>-decalage): #32
  new_index = len(df) #33
  df.at[new_index, 'time'] = time #34
  df.at[new_index, '1 min'] = pipsB #35
  if y[time] < x[time]:  #36
      df.at[new_index,'Indication A'] = 'D' #37 array1 has been decompressed = it is ahead of array2
      df.at[new_index, 'Indication B'] = 'C' #38 array1 has been compressed = it is behind of array1
      df.at[new_index, 'delay'] = x[time] - y[time]#39 #number of minutes late
  elif y[time] > x[time]: #40
      df.at[new_index,'Indication A'] = 'C'#41 This means that array1 has been compressed = it is behind of array2
      df.at[new_index, 'Indication B'] = 'D' #42 This means that array2 has been decompressed = it is ahead of array1
      df.at[new_index, 'delay'] = x[time] - y[time] #43 The number of minutes late
  else:
      df.at[new_index,'Indication A'] = 'N' #44 Array1 and array2 are at the same speed
      df.at[new_index, 'Indication B'] = 'N'#45 Array1 and array2 are at the same speed
      df.at[new_index, 'delay'] = x[time] - y[time] #46
  last row = df.iloc[-1] #47
```

Figure 20: Evaluation of compression and decompression moments

What's important to understand is that when the first and second 'path' coordinates are the same, for example (0,0), (1,1), etc., this ultimately means that *array1* and *array2* are not behind each otheraccording to DTW. But as soon as there is a difference between the first and second coordinates, it means that one of the two time series is ahead of the other.

The difference between the first and second coordinates is measured as: |x[time] - y[time]|. If the difference is greater than the *decalage* parameter inserted at the input of the function, this means that the DTW has encountered a problem and is not relevant. If we think that the deviation |x[time] - y[time]| is relevant, then we will fill the *df* dataframe with the following information: *time*, *1 min*, *Indication A*, *Indication B* and *delay*:

- time : corresponds to the current time (minute i).
- 1min : corresponds to the variation in pips in the time series B at *time*.
- Indication A : is filled with a "D" when time series A has been shifted ahead of series B. Is filled with a "C" when time series A has lagged behind series B. Is filled with an "N" when series A and B have the same speed.
- Indication B : filled in the same way as "Indication A", but from the point of view of series B in relation to series A.
- Delay : represents the number of minutes of delay estimated by the DTW for the time series in question.

Once we have correctly filled in the dataframe line (*df*), we store it in the *last_row* variable using instruction 47.

4.3.3 Setting stop loss and stop gain

4.3.4 Identification of DTW compression or decompression moments

```
if(pipsA>= meanA) : #48 High array1 performance compared to average
    if(last_row['Indication A'] == 'C' and last_row['1 min'] < 0): #49 Array1 is late
      sell_price = array1[len(array1)-1] #50 I short A
      trade_en_cours = True #51
      stop_loss = sell_price+((MaxPositifA*varMin)*0.0001) #52 Add +0.0002 if we work with 15min data
      stop_gain = sell_price +((-MinNegativeA*varMax)*0.0001) #53 Add -0.0002 if we work with 15min data
      sum_trade +=1 #54 Number of trade
      print("le stop gain est ", stop_gain)
      print("le stop loss est ", stop_loss)
      print("le buy price est ", sell_price)
      trade_short = True #55
```

Figure 21: Code to fix a short order

So now, if the last change in pips for time series A is greater than the average change in pips for A, and according to our dataframe (*df*), time series A is lagging (Indication A = C) behind series B and series B has suffered a negative change (1min $<$ 0), then we estimate that series A will repeat a negative change in the next few minutes. We therefore need to short GBPUSD.

To put it more simply: *array1* (A) is behind *array2* (B) and *array2* (B)'s performance is negative, so I sell *array1* (A) short.

We set the booleans *trade_ en_ cours* and *trade_ short* to True to indicate that a position has been taken. We set our *stop_ gains* (take profit) and *stop_ loss*.

(*) We have defined them dynamically so that the difference between the sell price and the stop loss is less than the difference between the sell price and the stop gain. This way, when we win a trade, we reap more money than when we lose a trade.

Our *stop_ loss* is defined as the selling price (as a short position) added to the largest variation observed over the last 20 minutes for the GBPUSD pair, multiplied by a percentage *varMin* (function parameter).

Our *stop_ gain* is defined as the sell price (as a short position) subtracted from the absolute value of the minimum negative variation, multiplied by a percentage *varMax* (function parameter). It is important to note that *varMin* must be less than *varMax* to ensure that (*) is always true.

Here is an example to illustrate the methodology used to set our stop loss and take profit for a short position:

Figure 22: Explanatory example of the methodology for SHORT stop orders

4.4 Long Position

4.4.1 Trigger threshold and DTW calculations

In the case of a potential **long** position on GBPUSD, the idea is the same as for a short position, but certain conditions will be reversed. We start by checking whether the current variation in EURUSD is large enough (above the max-value-pc threshold). Then we apply the DTW and obtain the shortest path in the DTW cost matrix.

```
#Long the pair GBPUSD
  if pipsB > (max_value_pc) :

    # Calculations related to DTW
    path, cost_matrix, acc_cost_matrix = cost_matrix_compute(a,b)
    deformed_a = [a[p[0]] for p in path[:]]
    x = [p[1] for p in path]
    y = [p[0] for p in path]
    #plot_deformed(a,b,deformed_a)
    time = len(array2)-1
```

Figure 23: DTW calculation for a short position

In the same way, we complete the *df* dataframe. The idea behind the long position is as follows: when I know that the *array1* (GBPUSD) is lagging behind the EURUSD, and that there is a significant positive variation in the EURUSD, I anticipate a rise in the GBPUSD in the next few minutes: I buy the GBPUSD.

4.4.2 Setting the stop loss and stop gain

The part that will change concerns the definition of *stop_loss* and *stop_gain* :

```
# Buying GPBUSD
if(pipsA <= meanA):
  if(last_row['Indication A'] == 'C' and last_row['1 min'] > 0):
    percentofvariationB = array1[len(array2)-1] - array1[len(array2)-2] / ( array1[len(array2)-2])
    buy_price = array1[len(array1)-1]
    trade_en_cours = True
    stop_loss = buy_price +((varMin*(-MinNegativeA))*0.0001) # Add - 0.0002 for 15 min data
    stop_gain = buy_price +((varMax*MaxPositifA)*0.0001) # Add + 0.0002 for 15 min data
    sum_trade +=1
    print("le stop gain est ", stop_gain)
    print("le stop loss est ", stop_loss)
    print("le buy price est ", buy_price)
    trade_long = True
```

Figure 24: definition of stop order

The *stop_loss* is defined as the purchase price (of the GBPUSD) plus the smallest (negative) variation observed over the last 20 minutes, multiplied by a percentage *varMin* defined as a parameter to the function.

The *stop_gain* is defined as the purchase price (of the GBPUSD) plus the largest (positive) variation recorded over the last 20 minutes, multiplied by a percentage *varMax* defined as a parameter to the function.

Here is an example to illustrate the methodology used to set our *stop_loss* and *stop_gain* for a long position.

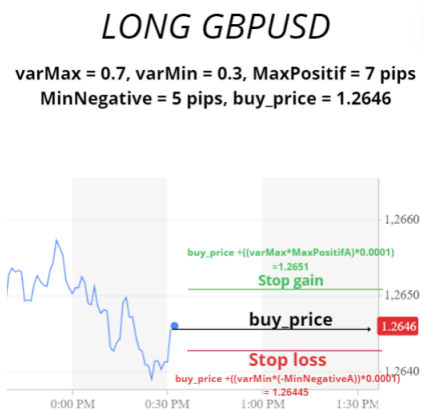

Figure 27 : : Explanatory example of the methodology for LONG stop orders

Figure 25: Explanatory example of the methodology for LONG stop orders

4.5 Closing a position

If a trade is in progress, simply monitor whether a *stop_loss* or *stop_gain* has been exceeded, then close the position and calculate the result. There are four possible cases:

1) If I trade GBPUSD long and the GBPUSD price exceeds my *stop_gain*, I cut my position. My position wins and my profit is equal to *stop_gain* - *buy_price*.
2) In the case of a long GBPUSD trade and the GBPUSD price falls below my *stop_loss*, I cut my position. I lose my position and my loss is equal to *stop_loss* - *buy_price*.
3) In the case of a short trade on GBPUSD and the GBPUSD price falls below my *stop_gain*, I cut my position. My position wins and my profit is equal to *sell_price* - *stop_gain*.
4) In the case of a short GBPUSD trade and the GBPUSD price rises above my *stop_loss*, I cut my position. I lose my position and my loss is equal to *sell_price* - *stop_loss*.

In these four situations, we determine the rate of return on the trade in the *trade_return* variable: *gain*/*buy_price* (for a purchased currency, in this case a pound sterling).

Once the situation has been identified, we need to take into account the transaction costs in order to determine the net amount at the end of the trade. I estimate the transaction costs/spread bid ask on entry to the position, which corresponds to line 56. We then subtract these costs from the initial amount to obtain the net capital inserted in the position (Line 57). To obtain the gross amount at the end of the trade, we multiply by 1+ *trade_return* determined earlier (Line 58). We now estimate the transaction costs/bid ask spread at the exit of the position (Line 59). And now, by subtracting these costs from our gross amount, we obtain the net amount at the end of the trade, which will correspond to our gross entry amount for the next position (Line 60 and Line 61).

Once the situation has been identified, simply enter the following information in the *results* dataframe: the result (*win* or *loss*), the amount (*gain*), the buy price (*buy_price*), the *stop_gain*, the *stop_loss*, type of trade (*Long* or *Short*), the trade day (*day*), the trade months (*months*), the hour of the trade, the minute of the trade, trade returns (*trade_return*), the begin brut trade amount (*brut_init_amount*), the begin net trade amount (*net_init_amount*), the end brut trade amount (*brut_end_amount*), the net end amount (*net_end_amount*), the initial fees (*init_fees*) and the end fees (*end_fees*).

```
if array1[len(array1)-1] >= stop_gain and trade_long == True: # Stop gain exceeded in case of long = WIN #56
  gain = stop_gain - buy_price

  trade_return =gain/buy_price #Calculate the return on the trade (in %)
  brut_init_amount = montant #Initial brut investment of the trade
  init_fees = brut_init_amount * fees #Initial amount of fees #56
  net_init_amount = brut_init_amount - init_fees #Initial net investment of the trade #57

  brut_end_amount = net_init_amount*(1 + trade_return) #58
  end_fees = brut_end_amount *fees #59
  net_end_amount = brut_end_amount - end_fees #60
  montant = net_end_amount #61

  print("END OF TRADE, YOU HAVE WON", gain)
  print("prix réalisé", array1[len(array1)-1], "réalisé en ", 20+i-1)
  trade_en_cours = False
  trade_long = False
  results = results._append({'Result': 'win', 'Montant': gain, 'Buy Price': buy_price, 'Stop Gain':
                             stop_gain, 'Stop Loss': stop_loss, 'Type': 'Long', 'Day': day, 'Months' : months,
                             'Hours': ((20+i-2)// 60), 'Min': (20+i-2)% 60 ,'Trade returns' : trade_return,'Begin brut trade amount':
                             brut_init_amount,'Begin net trade amount': net_init_amount, 'End brut trade amount': brut_end_amount,'End net trade amount':
                             net_end_amount, 'Begin Fees':init_fees, 'End Fees':end_fees}, ignore_index=True)
```

Figure 26: Closing the Long position for a win

```
if array1[len(array1)-1] <= stop_loss and trade_long == True :# Stop loss exceeded in case of long = LOSS
  loss = stop_loss - buy_price

  trade_return =loss/buy_price #Calculate the return on the trade (in %)
  brut_init_amount = montant #Initial brut investment of the trade
  init_fees = brut_init_amount * fees #Initial amount of fees
  net_init_amount = brut_init_amount - init_fees #Initial net investment of the trade

  brut_end_amount = net_init_amount*(1 + trade_return)
  end_fees = brut_end_amount *fees
  net_end_amount = brut_end_amount - end_fees
  montant = net_end_amount

  print("END OF TRADE, YOU HAVE LOST", loss)
  trade_en_cours = False
  trade_long = False
  #print("prix réalisé", array1[len(array1)-1], "réalisé en ",20+i-1)
  results = results._append({'Result': 'loose', 'Montant': loss, 'Buy Price': buy_price, 'Stop Gain':
                             stop_gain, 'Stop Loss': stop_loss, 'Type': 'Long', 'Day': day, 'Months' : months,
                             'Hours': ((min+20+i-2)// 60), 'Min': (20+i-2)% 60, 'Trade returns' : trade_return,'Begin brut trade amount':
                             brut_init_amount,'Begin net trade amount': net_init_amount, 'End brut trade amount': brut_end_amount,'End net trade amount'
                             net_end_amount, 'Begin Fees':init_fees, 'End Fees':end_fees}, ignore_index=True)
```

Figure 27: Closing the Long position for a loss

Our *test_strategy()* function returns the dataframe containing all the informations about the positions taken during the *nbpoints* minutes and the *montant* which correspond to the amount for the next brut amount for the next trade.

4.6 Improving strategy with mean reverting

4.6.1 How mean reverting works ?

This strategy is based on the assumption that the price of an asset will tend to converge towards the average price over time. In simple terms, when the current market price is below the past average price and below a certain threshold, it may be worth going long. Conversely, when the current market price is above the past average price and above a certain threshold, it would appear to be worth going short. In other words, deviations from the average price should return to the average: this is the basis of this strategy.

4.6.2 Implementing a DTW and Mean reverting strategy

We decided to add a basic mean reversion component to our DTW strategy to see if this improved the results.

```
dfmr = pd.DataFrame()
dfmr["Adj Close"] = array1
window=windows1
dfmr['ma_20'] = dfmr['Adj Close'].rolling(window=window).mean()
dfmr['std_20']  = dfmr['Adj Close'].rolling(window=window).std()
dfmr['zscore']  = (dfmr['Adj Close'] - dfmr['ma_20']) / dfmr['std_20']
n_std=1.25
dfmr['signal'] = np.where(dfmr['zscore'] < -n_std, 1, np.where(dfmr['zscore'] > n_std,-1, 0))
```

Figure 28: Mean reverting construction

We define a dataframe called *dfmr* which contains the mean reversion signals. In this dataframe, we store GBPUSD prices for the last *windows1* minutes in the *Adj Close* column. The *ma_20 column* calculates the moving average over a sliding window of *windows1* periods, while the *std_20* column calculates the standard deviation over the same sliding window.

The z-score is a statistical measure that quantifies the extent to which an individual value differs from the mean of a set of data, expressed in terms of standard deviation. It is calculated by subtracting the moving average from the standard deviation.

We define a threshold n-std equal to 1.25, which will be used as a reference to compare the z-score of each minute and assign a signal :

- When the z-score is below the threshold value *-n_std*, this can be interpreted as a signal to buy (long) the GBPUSD. We assign a value of 1 to the signal, which means that the price of the GBPUSD is very low compared with its usual average and that it is likely to rise again.
- When the z-score is above the threshold value *n_std*, this can be interpreted as a signal to sell short the GBPUSD. We assign a value of -1 to the signal, which means that the price of the GBPUSD is very high compared with its usual average and is likely to fall.

We use these signals to complete our DTW strategy, more specifically in the case of a short position: we add a condition which checks whether the signal of the current value (the last in the *array1* which contains the last 20 elements) is equal to -1. If these two conditions are met, this means that both the DTW indicates a short position on the GBPUSD and the mean reversion confirms this decision.

```
if(pips_changesA[len(pips_changesA)-1] >= exec) and dfmr.iloc[19]['signal']==-1:
```

Figure 29: New condition for shorts

In the case of a long position on GBPUSD, all we need to do is add a condition that checks whether the signal of the current value (the last one in *array1*, which contains the last 20 elements) is equal to 1. So, if these two conditions are met, it means that DTW agrees with the mean reversion to take a long position on GBPUSD.

```
if(pips_changesA[len(pips_changesA)-1] <= exec) and dfmr.iloc[19]['signal']==1:
```

Figure 30: New condition for shorts

In fact, after running a number of tests with different parameters, we've found that combining DTW and mean reverting gives much more interesting results than DTW alone. It is therefore essential to combine these two indicators in order to implement a more profitable strategy. For example, using our optimal parameters estimated in the next section, we obtained the following results for two weeks of trades:

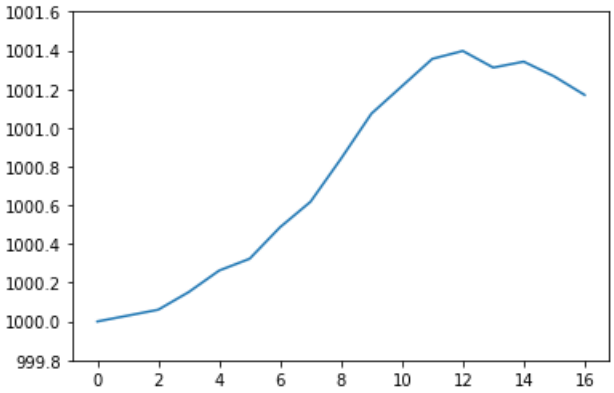

Figure 31: $1000 Strategy with DTW and Mean reverting during 2 weeks

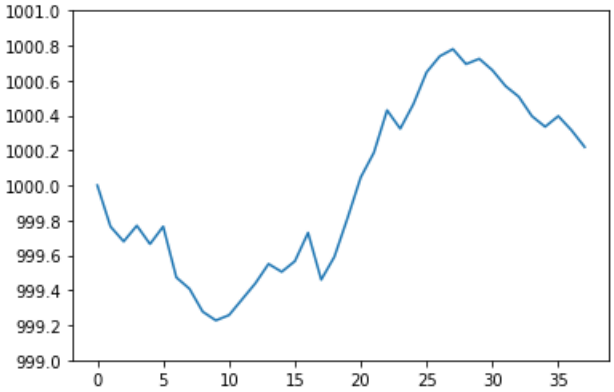

Figure 32: $1000 Strategy with DTW only during 2 weeks

5 Optimum parameters search

5.1 Strategy variables

The *test_strategy()* function has exactly 6 variables that we can vary in order to estimate the best combination of parameters. Let's recall the usefulness of these 5 parameters:

- declenchement: This is used to set the value at which the variation is deemed sufficiently significant to trigger a trade.
- decalage : This is the threshold above which the DTW is considered uninteresting, as it is too large and not representative of reality.
- nbpointsDTW: This is the number of minutes we take into account when constructing our DTW cost matrix.

• varMin: This represents a percentage used to set the *stop_loss* for a short or long GBPUSD position.

• varMax: This represents a percentage used to set the *stop_gain* for a short or long GBPUSD position.

• windows1 :corresponds to the number of days used to calculate the metrics for mean reverting.

• montant : initial amount invested in the strategy, by default $1000 .

• fees : transaction fees and bid ask spread, default 0.1 pips.

To select the combination that maximized our gains, we carried out tests on an initial set of training data. Above all, we needed to be able to export the results of each combination in a single file.

5.1.1 Function to export our results to Excel

```
def save_styled_dataframe_to_excel(styled_dataframe, merged_dataframe, filename, declenchement, decalage, nbpointsDTW, varMin, varMax,windows1):
    # Check if the file exists
    file_exists = False
    try:
        workbook = openpyxl.load_workbook(filename)
        file_exists = True
    except FileNotFoundError:
        workbook = openpyxl.Workbook()

    # Select the active sheet
    sheet = workbook.active

    # Convert the styled dataframe to a regular dataframe
    dataframe = styled_dataframe.data
```

Figure 33: Start of Excel export function

The save_styled_dataframe_to_excel() function takes as parameters the 6 variables mentioned above, as well as the future name of the Excel file (*filename)* and two dataframes (*styled_dataframe* and *merged_dataframe)*. The two dataframes are almost identical, except that *styled_dataframe* is an improved, stylized version of *merged_dataframe*. The first steps of the function are designed to define the active sheet and check the existence of the file to be created.

```
# Append a line of '/' as a separator if the file already exists
if file_exists:
    separator_row = ['/' for _ in range(len(dataframe.columns))]
    sheet.append(separator_row)
sheet.append(['Parameters (declenchement, decalage, nbpointsDTW, varMin, varMax, windows):', declenchement, decalage, nbpointsDTW, varMin, varMax,windows1])
# Append the dataframe to the sheet
for row in dataframe_to_rows(dataframe, index=False, header=True):
    sheet.append(row)
```

Figure 34: Display parameters

This part allows you to add each result of each combination on the same Excel sheet. To make the Excel sheet easier to read, we add a separating line between two results. Each result block is made up of the 6 variables and their values. Here's an example of how a result block is displayed in Excel:

Parameter	0,65	3	10	0,4	0,8	20										
Result	Montant	Day	Months	Hours	Min	Buy Price	Stop Gain	Stop Loss	Type	Trade retu	Begin brut	Begin net	End brut tr	End net tr	Begin Fee	End Fees
win	0,000139	14	2	14	44	1,25428	1,254419	1,254117	Long	0,000111	1000	999,99	1000,101	1000,091	0,01	0,010001
loose	-5,1E-05	15	2	21	23	1,259922	1,259846	1,259973	Short	-4E-05	1000,091	1000,081	1000,04	1000,03	0,010001	0,01
win	6,37E-05	20	2	21	15	1,262594	1,262531	1,262645	Short	5,05E-05	1000,03	1000,02	1000,071	1000,061	0,01	0,010001
win	0,00014	20	2	21	19	1,262738	1,262598	1,262865	Short	0,000111	1000,061	1000,051	1000,162	1000,152	0,010001	0,010002
win	0,000166	21	2	14	11	1,262802	1,262636	1,262897	Short	0,000131	1000,152	1000,142	1000,273	1000,263	0,010002	0,010003
win	0,000102	22	2	6	3	1,265759	1,265656	1,265861	Short	8,08E-05	1000,263	1000,253	1000,334	1000,324	0,010003	0,010003
win	0,000232	22	2	8	8	1,270196	1,269964	1,270261	Short	0,000183	1000,324	1000,314	1000,497	1000,487	0,010003	0,010005
win	0,000191	22	2	13	33	1,261432	1,261623	1,261279	Long	0,000151	1000,487	1000,477	1000,628	1000,618	0,010005	0,010006
win	0,000306	22	2	13	35	1,262929	1,262624	1,263375	Short	0,000242	1000,618	1000,608	1000,85	1000,84	0,010006	0,010009
win	0,000319	22	2	13	40	1,262547	1,262228	1,262993	Short	0,000253	1000,84	1000,83	1001,083	1001,073	0,010008	0,010011
win	0,000204	22	2	14	14	1,262563	1,262359	1,262728	Short	0,000161	1001,073	1001,063	1001,225	1001,215	0,010011	0,010012
win	0,000204	22	2	14	59	1,264254	1,264051	1,264459	Short	0,000161	1001,215	1001,205	1001,366	1001,356	0,010012	0,010014
win	7,69E-05	22	2	16	47	1,265935	1,265858	1,265999	Short	6,07E-05	1001,356	1001,346	1001,407	1001,397	0,010014	0,010014
loose	-8,3E-05	22	2	17	54	1,266368	1,266188	1,266451	Short	-6,6E-05	1001,397	1001,387	1001,321	1001,311	0,010014	0,010013
win	6,41E-05	22	2	18	54	1,265679	1,265743	1,265614	Long	5,06E-05	1001,311	1001,301	1001,352	1001,342	0,010013	0,010014
loose	-7,1E-05	26	2	13	4	1,269616	1,269732	1,269545	Long	-5,6E-05	1001,342	1001,332	1001,276	1001,266	0,010013	0,010013
loose	-9,6E-05	28	2	10	8	1,264334	1,264539	1,264238	Long	-7,6E-05	1001,266	1001,256	1001,18	1001,17	0,010013	0,010012
Number of	13															
Number of	4															
NET real R	0,00117															

Figure 35: Example of a block in Excel

```
# Count the 'win' values
counts = merged_dataframe['Result'].value_counts()
win_count = counts.get('win', 0)
loose_count = counts.get('loose',0)
# Add a row in the Excel file for the number of 'win' values and the total amount
sheet.append(['Number of wins:', win_count])
sheet.append(['Number of losses:', loose_count])
sheet.append(['NET real RETURN :', (merged_dataframe["End net trade amount"].iloc[-1]
                                    -merged_dataframe["Begin brut trade amount"].iloc[0])/(merged_dataframe["Begin brut trade amount"].iloc[0])])
# Apply the styles to the sheet
for cell in sheet["A1":"Z1000"]:
    cell_obj = cell[0]
    row_num, col_num = cell_obj.row, cell_obj.column_letter
    if row_num > 1 and col_num != 'A':  # Skip the header row and the first column
        style = styled_dataframe.data.loc[row_num-2, col_num-2]
        cell_obj._style = style


workbook.save(filename)
```

Figure 36: End of code to export to Excel

At the end of each result block, we calculate the total number of winning trades, losing trades and the net real return, which corresponds to a classic return calculation using the last amount in our dataframe compared with our initial amount (by default $1000). The final code is then used to save the Excel file correctly.

5.2 Results

Now that we're able to save our results cleanly in an Excel file, we can test our trading strategy based on the DTW and mean reverting approach on a first set of training data. We tried out a total of 596 different combinations by varying our different variables using for loops over two weeks of Forex data. That's over 25,000 rows in Excel! Thanks to a little VBA code, we were able to identify potentially interesting combinations for our subsequent tests. To do this, we have created a test function that loops through a series of days, capturing any errors. This function calls all the other functions presented in this report. It also displays a graph showing the variations in our strategy over the trades executed.

```python
def test(declenchement,decalage, nbpointsDTW, varMin, varMax,windows1):
    merged_dataframe = pd.DataFrame()
    montant = 1000

    for i in range(8, 31, 1):
        try:
            test_strategy1, montant= test_strategy(1200, 2024, 2,i , 1, 0, declenchement, decalage, nbpointsDTW, varMin, varMax,windows1,montant)
            merged_dataframe = pd.concat([merged_dataframe, test_strategy1])
        except Exception as e:
            print(f"Error for the day {i} of months 2 :", e)
            continue

    merged_dataframe = merged_dataframe.reset_index().rename(columns={'index': 'Strategy'})
    merged_dataframe = merged_dataframe.drop("Strategy", axis=1)
    styled_dataframe = merged_dataframe.style.apply(color_row, axis=1)
    clear_output(wait=True)
    print(styled_dataframe)
    save_styled_dataframe_to_excel(styled_dataframe,merged_dataframe, "file_jourforexoptimaux8_.xlsx", declenchement, decalage,nbpointsDTW,varMin, varMax,windows1)

    merged_dataframe.loc[0, 'End net trade amount'] = 1000
    plot_soussous = merged_dataframe['End net trade amount']
    plt.plot(plot_soussous)
    plt.grid(True)
    plt.show()
```

Figure 37: Back test with the function test

We then tested these few combinations on a new two-week data set. This allowed us to select the best combination of variables for GBPUSD vs EURUSD trades. So, for the best combination identified for trading using DTW on GBPUSD against EURUSD, here are the results obtained on a test dataset for two weeks:

The optimal parameters would therefore be :
$->$ $declenchement = 0.65$, $decalage = 3$, $nbpointsDTW = 10$, $varMin = 0.4$, $varMax = 0.8$, $windows1 = 20$. Net return for 1 month : 0.001549.

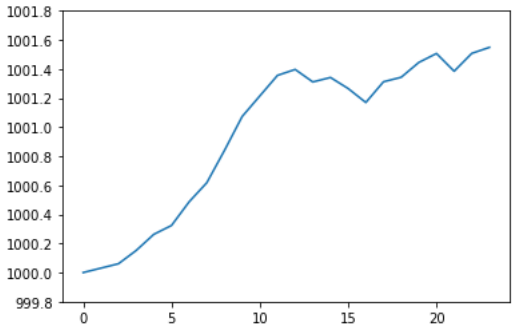

Figure 38: Evolution of 1 month with optimal parameters for $1000 portfolio

6 Conclusion and ideas for improving your trading strategy

The trading strategy we have developed is based on the study of the differences in variations between two highly correlated assets. The aim is to predict an increase or decrease in the price of asset A, taking into account the lag estimated by the DTW cost matrix in relation to a highly correlated asset B. In general, this correction does not take much time. If our hypothesis is correct, the variation of asset A must exceed the transaction costs and the bid-ask spread to be profitable. However, the problem with FOREX currencies is their low volatility and limited variation. This makes it difficult to be profitable when transaction costs and bid-ask spread are taken into account. There are several ways in which we can improve our performance:

- Improve the current mean reverting coupled with our DTW approach to obtain a more elaborate and efficient mean reverting.
- Have access to more data at a frequency of at least one minute and with sufficient precision. We used data from Yahoo Finance, which is accurate to 10e-10, but we were limited in the amount of data we could use. Bloomberg data was not precise enough for our needs (10e-4 accuracy). It might also be interesting to use data at a higher frequency, say every fifteen seconds, to take full advantage of price corrections.
- Another possibility is to use more volatile financial assets to ensure that transaction fees and bid-ask spread are exceeded. For example, crypto-currencies correlated with each other, such as Bitcoin versus Ethereum, could be good candidates.
- Finally, the method of setting stop-losses and stop-gains should be reviewed to achieve better results. Other approaches may prove more effective.

In summary, there are several avenues of improvement to enhance trading strategy performance, including improving mean reverting, accessing more accurate data and exploring more volatile financial assets. The setting of stop-losses and stop-gains could also be reviewed to achieve better results.