

How to find and categorize resistance/support levels for back testing and online trading efficiently?

In this document you will find a basic and efficient solution for a trading strategy based on resistance/support levels. The complete source code can be found in GitHub in this [link](#). It is an object-oriented code, so you can use each of the classes simply inside your code.

How can find resistance/support levels efficiently?

Assuming radius = 240, a resistance level is a point where its price is higher than that of its 240 neighbors on either side. And a support level is a point where its price is lower than that of its 240 neighbors on either side.

```
def is_level_support(back_data, i, radius):
    """ check if a level is SUPPORT level through fractal identification """
    support=True
    for r in range(radius):
        support = support and back_data.Low.iloc[i] < back_data.Low.iloc[i +r+1]
        support = support and back_data.Low.iloc[i] < back_data.Low.iloc[i -r-1]
    return support

def is_level_resistance(back_data, i, radius):
    """ check if a level is RESISTANCE level through fractal identification """
    resistance=True
    for r in range(radius):
        resistance = resistance and back_data.High.iloc[i] > back_data.High.iloc[i +r+1]
        resistance = resistance and back_data.High.iloc[i] > back_data.High.iloc[i -r-1]
    return resistance
```

However, for each point, 480 back points need to be checked. Assuming you have large amounts of minute data, processing 5 years of data will take a long time. We have a data stream, so by defining two data history window frames, we need to check only 2 back points for each current point, so the processing time is significantly reduced. The implemented code is:

```
#find the max/min in first Radius
cur_high=cur_data.High
if self.level_window_max<cur_high:
    self.level_window_max=cur_high
    self.level_window_max_counter=0

cur_low=cur_data.Low
if cur_low<self.level_window_min:
    self.level_window_min=cur_low
    self.level_window_min_counter=0

#find next resistance/support level
if i>2*self.radius:#at the start of the program, we must have enough data
    if self.level_window_high[self.radius-1]==self.level_window_max and self.level_window_max_counter==self.radius:
        level=self.level_window_max
        self.level_window_max=0
    if self.level_window_low[self.radius-1]==self.level_window_min and self.level_window_min_counter==self.radius:
        level=self.level_window_min
        self.level_window_min=10**10#a big number

#shift right and insert new prices
self.level_window_high=self.level_window_high[:-1] #shift right
self.level_window_high.insert(0,cur_high)
self.level_window_low=self.level_window_low[:-1] #shift right
self.level_window_low.insert(0,cur_low)

self.level_window_max_counter+=1
self.level_window_min_counter+=1
```

Save to newfound levels.

Then we can update our levels arrays: 'levels' as sorted levels and 'levels_shape' as sequential levels for drawing purpose.

```
#update levels history
if level:#if found new level

    if self.levels_shape.size>0:
        plevel=self.levels_shape[-1,2] #previous level
        positive_signal=level*1.003<plevel<cur_close/1.003<plevel*1.01 #can be used for trading

    # Update data for displaying Support/Resistance levels
    self.levels_shape = np.append(self.levels_shape, [[cur_data.Time, back_data.Time, level, positive_signal]], axis=0

    # Find new level position in Dataframe levels
    i_level = np.searchsorted(self.levels['level_max'].to_numpy(dtype='float'), level)
    level_distance_min= cur_close*fee_rate
    num_levels=len(self.levels)

    #look back and check if near to previous level
    if i_level!=0 and level-self.levels.level_min.iloc[i_level-1]<level_distance_min\
    and i_level!=num_levels and level-self.levels.level_min.iloc[i_level-1]<self.levels.level_max.iloc[i_level]-level:
        if self.levels.level_max.iloc[i_level-1]<level:
            self.levels.at[i_level-1,'level_max']=level #change the previous level max

            self.levels.at[i_level-1,'occurrences'+=1
            self.levels.at[i_level-1,'last_time']=back_data.Time

    elif i_level!=num_levels and self.levels.level_max.iloc[i_level]-level<level_distance_min: #look forward and check
        if level<self.levels.level_min.iloc[i_level]:
            self.levels.at[i_level,'level_min']=level #change the next level min

            self.levels.at[i_level,'occurrences'+=1
            self.levels.at[i_level,'last_time']=back_data.Time

    else:#new channel of levels
        level_df=pd.DataFrame({'level_max':[level], 'level_min':[level], 'occurrences':1, \
                               'first_time':[back_data.Time], 'last_time':[back_data.Time] })
        if num_levels:#if levels is not empty
            self.levels=pd.concat([self.levels.loc[:i_level-1], level_df, self.levels.loc[i_level:]], ignore_index=True) #
        else:
            self.levels=level_df

return positive_signal #levels
```

In the code above, you can even check the new found level to see if it's a good time to buy. You can expand it later:

```
if self.levels_shape.size>0:
    plevel=self.levels_shape[-1,2] #previous level
    positive_signal=level*1.003<plevel<cur_close/1.003<plevel*1.01 #can be used for trading
```

Show the results

Then we can draw the levels using plotly module including a range slider:

```

class Plot():
    """ plot Close/Time + Resitance/Support levels"""
    def __init__(self, time, close, levels, title):
        figure = go.Figure()
        figure.add_trace(go.Scatter(x=list(time), y=list(close), line=dict(color='Blue', width=1)))#Display Close Prices

        for i in range(1,len(levels)):# Display Resistance/Support Leels
            if levels[i,3]:
                color='Green'
            else:
                color="Red"
            figure.add_shape(type='line', x0=levels[i,1], y0=levels[i,2], x1=levels[i,0], y1=levels[i,2], line=dict(color=color, width=1))

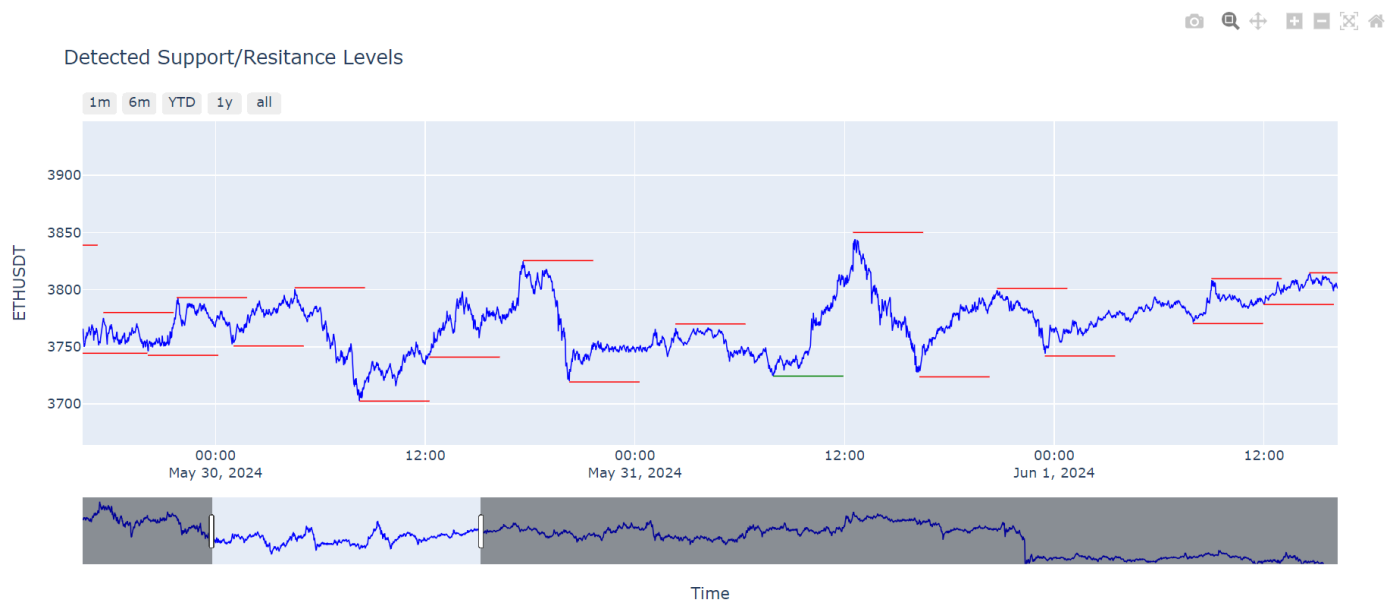
        # Set the title text of figure
        figure.update_layout(title_text="Detected Support/Resitance Levels", xaxis_title="Time", yaxis_title=title)

        # Add range slider
        figure.update_layout(
            xaxis=dict(rangeslider=dict(
                buttons=list([dict(count=1, label="1m", step="month", stepmode="backward"),
                             dict(count=6, label="6m", step="month", stepmode="backward"),
                             dict(count=1, label="YTD", step="year", stepmode="todate"),
                             dict(count=1, label="1y", step="year", stepmode="backward"),
                             dict(step="all") ])),
                rangeslider=dict(visible=True), type="date"))

        figure.update_layout(yaxis=dict(autorange=True, fixedrange=False))
        figure.show()

```

The result will be:



As you can see, the potential valuable levels for trading are shown in green color.

Inside this code I defined the following classes:

- class **FindLevels()** for finding support/resistance levels
- class **BackData()** for loading and preparing back data from a file or directly from Binance exchange
- class **Plot()** for drawing prices and also support/resistance levels.

I defined the following pandas and numpy variables:

- Pandas sorted **levels** in separated channels and Numpy **levels_shape** for drawing purpose:

```

self.levels_shape = np.empty(shape=[0, 4]) #CurrentTime, FindTime, Level
self.levels = pd.DataFrame({'level_max':[], 'level_min':[], 'occurrences':[], 'first_time':[], 'last_time':[]})
self.levels=self.levels.astype({'level_max': 'float', 'level_min': 'float', 'occurrences': 'int', 'first_time': 'str', 'last_time': 'str'})

```

- pandas **trades** for storing trade results.

```
trades = pd.DataFrame({'buy_price':[], 'sell_price':[], 'pattern':[], 'buy_time':[], 'sell_time':[], 'pL0':[], 'pL1':[], 'pL2':[], 'pL3':[], 'pL4':  
trades=trades.astype({'buy_price': 'float', 'sell_price': 'float', 'pattern': 'str', 'buy_time': 'str', 'sell_time': 'str', 'pL0': 'float', 'pL1': 'f1
```

It is use full for later analysis and tuning the code performance.

Disclaimer: This document is intended for educational purposes only and shall not to be considered as trading tools

Author = Ar. Manafi