Back tester (BT).

Introduction

Back testing(BT) is most of the time a crime. Forex brokers offer via MT4 and MT5 back testing opportunities. But you never really know if you have full history and / or if I trade multi instruments, are the bars synchronized. MT4 back testing is nice looking but tells you a little about the real trading on live account. MT5 is better, also suitable for multi instrument, multi time frame strategies. But in both cases the mathematical possibilities in mq4 or mq5 are limited. All most all has to be coded by yourself. Python has a lot of math libraries. Also for python all kind of libraries are offered for back testing. I have little or no experience with this. They are somehow woven with your code. But back testing in python and live trading will be different.

So my idea can I make a BT which behaves like a MT4/MT5 terminal (*more or less outside your code/orbit*) and do forward testing. From Dukascopy I downloaded M1 bars for the 28 basic instruments over the last 5.5 years. I noticed that the amount of bars for the different instruments are not exactly the same. Over the 5.5 years the difference can be over 2000 M1 bars, for multi instrument trading this is not good. So I synchronised the M1 bars, by taking 1 instrument as master(GBPUSD, has most bars) and for the other instruments filling the holes(duplicating) or deleting the additional bars. This process has as output 28 .csv files all with the same amount of records/bars with same timestamps. The day starts at 00.00, so did some date offset and summer/winter time correction too. For each instrument around 2.000.000 bars. I thought about tick data. But tick data for 1 instrument for 1 year is about 20-50 million ticks, so decided not to do for the time being. Synchronization between instruments with tick data is not possible anymore. 5 seconds bars could be a good alternative for tick data/bars.

Every 3 months bar data will be updated.

For the BT I want to have a same interface as with the Pytrader_API. The BT is realized as a Socket server written in C# and the new Pytrader_BT_API for the BT will make calls to the Socket server as a client. Also the functions should be the same if they are applicable. This will make the switch to live trading smooth.

So there will be a Socket server app. with the needed M1 bar .csv files. For licensed Pytrader_API customers the BT will work for all instruments. For others the BT will work for limited number of instruments (EURUSD, USDJPY, AUDNZD).

In the BT also trades can be opened, closed, etc.

At the moment I'm extending the BT with renko bars, later volume bars will follow. Also red news will be incorporated in a next release.

Date: 01-09-2021, BT version 1.02

Table of Contents

Introduction 1

Changes 5

Functions. 6

1. Set up the BT. 6
2. Instantiation. 7
3. Connect to server 7
4. Check connection. 8
5. Preparation of the BT for back testing 8
6. Reset BT 10
7. Reset BT pointers 10
8. Set the spread and commission 11
9. Get data factory info 11
10. Set bar date ascending or descending 11
11. Set the bar index for start back testing 12
12. Go X increments forwards 12
13. Change time out value 12
14. Retrieve broker server time. 12
15. Get dynamic account information 13
16. Get instrument information 13
17. Get last tick information 13
18. Get actual bar information 14
19. Get last x bars from now 14
20. Get a specific bar for list of instruments 14
21. Open order 15
22. Set SL and TP for position 15
23. Reset SL and TP for position 16
24. Set SL and TP for order (pendings) 16
25. Reset SL and TP for order 16
26. Get all orders 16
27. Get all (open) positions 16
28. Get all closed positions 17
29. Get last x closed positions 17
30. Close position by ticket 17
31. Delete order by ticket 17
32. Set BT for renko bars 18

33. Set brick size for renko bars 18
34. Retrieve amount of renko bars 18
35. Get actual renko bar info..... 18
36. Get last x renko bars from now 19
37. Get a specific renko bar for list of instruments..... 19

Changes

Date	Version	Changes
18-08-2021	V1_01	Initial version
01-09-2021	V1_02	Added renko bars for 3 different brick sizes in parallel. Added some general functions Extended error messages

Functions.

General

i. The "**Pytrader_BT_API_V1_01.py**" is coded as a class. In this documentation the instantiation will be named "**MT_Back**"

ii. After the execution of a function, the *MT_Back.command_OK* property will be set to *True* or *False*.

1. Set up the BT.

The BT and the instrument M1 bar datafiles have to be downloaded:

- The BT app has to be downloaded from git hub and to be installed.
- The datafiles for the M1 bars must also be downloaded from git hub. Due to size it will be zip files, which have to be unpacked. There will be 28 instrument files, the basic 28. The names of the files can't be changed.
- Of course the BT app. Has to be started

Next the Pytrader script "**Pytrader_BT_API_V1_01.py**" has to be downloaded from git hub and saved in your python project.
There is also a demo script (Demo_BT.py) available for download.

Remarks.

- The BT can work also with other programs, it does not need to be python. It is just the case that for python an API is coded.
- The BT works in full function mode without license for the following instruments: EURUSD, USDCAD and GBPJPY.
- For people who have a Pytrader_API license can use the full 28 instruments.
    - These will be extended in the future.
    - For license check a separate MT4/MT5 EA has to be downloaded from git hub. This EA only checks some license indicators.
    - The name of this EA is EA_License_check_V1.01.ex4/ex5.
    - This EA uses socket communication, default port = 5555

2. Instantiation.

declaration

from utils.Pytrader_BT_API_V1_01 import Pytrader_BT_API

instantiate
MT_Back = Pytrader_BT_API()

Utils is a subfolder in my python project. Can be any other sub folder, or just the main folder

3. Connect to server

At connection time a broker instrument dictionary has to be passed as a parameter. This dictionary is a lookup table for translating general instrument/symbol names into specific broker instrument/symbol names. This is for compatibility with *Pytrader_API*.

Instrument lookup dictionary, key=general instrument/symbol name, value=broker instrument/symbol name

brokerInstrumentsLookup = {'EURUSD':'EURUSD.ecn', 'GOLD':'XAUUSD', 'DAX':'GER30'}

connect to server local or to computer in same local network
Connected = MT_Back.Connect(server='127.0.0.1', port=10014,
instrument_lookup=brokerInstrumentsLookup)
or
Connected = MT_Back.Connect(server='192.168.0.103', port=10014,
instrument_lookup=brokerInstrumentsLookup)

Brokers often use different names for their instruments/symbols. Another way is to use config files. Example at the end of this document. *For the BT use the 6 character instrument names.*

Connected = bool, *True* or *False*.
If connection is made the *MT_Back.connected* property will be set to *True*. If no connection *MT_Back.connected* property will be set to *False*.

4. Check connection.

CheckAlive= MT_Back.Check_connection()

CheckAlive = bool, *True* or *False.*

5. Preparation of the BT for back testing

Before we can start back testing the BT has to be initialized. The BT has no account values. Some information like tick value, digits, point we have to retrieve from a broker with the Pytrader_API or define manually.

Before you can use the BT, like a MT terminal, the following functions/commands have to executed:

- Set the account parameters.
- Set the directory/folder where the BT can find the M1 bar data files
- Set comma or dot for reals
- Set the instruments, you want to trade
- Set the instrument parameters for the above instruments.
- Set the time frames you want to trade

i. Account settings.

The BT has no account data, some have to be set/created. At the moment margin calls are not implemented

Result = MT_Back.Set_account_parameters(balance=10000.0, max_pendings=100, margin_call=50)

Result = bool, *True* or *False*

Remark.
Margin call not yet implemented

ii. Set data directory

The BT must know where he can find the M1 data files

SetDir = MT_Back.Set_data_directory(data_directory="C:\\Temp")

SetDir = bool, *True* or *False*

The existence of the data directory is checked

iii. Set dot or comma, culture/country setting.

Depending on the country you are, float values are written with a dot or a comma.

Result = MT_Back.Set_comma_or_dot(mode="comma")

mode = "comma" or "dot"

Result = bool, *True* or *False*

iv. Set the instruments the BT will be using

Result = MT_Back.Set_instrument_list(['EURUSD', 'GBPUSD', 'AUDUSD'])

EURUSD, GBPUSD and AUDUSD are the instruments. The BT will load the data for(M1 bars) for the instruments in the list. So in this case only this 3 pairs can be back tested. For the time being only the 28 basic instruments can be used. If you specify more instruments then used, it will spoil performance.

Result = bool, *True* or *False*

Remark:

- This function needs only to be executed once as long the same instruments are used again and again. If many instruments then this loading of M1 bars can take some time, roughly 10 seconds for 1 instrument (of course machine dependent). So specify the instruments you really need.
- The existence of the datafiles will be checked

v. Instrument info parameters

For trading the BT needs information about the instruments like tick value to calculate profit, tradable lot sizes, …. for a trade. These information can be retrieved from a normal MT4 or MT5 terminal.

Get symbol info by normal broker via *Pytrader_API*
symbol_result = MT.Get_instrument_info(instrument='EURUSD'), this is Pytrader_API call

Set symbol info for BT
```
Result = MT_Back.Set_instrument_parameters(instrument=symbol_result ['instrument'],
        digits=symbol_result ['digits'], max_lotsize=symbol_result ['max_lotsize'],
        min_lotsize= symbol_result ['min_lotsize'], lot_stepsize=symbol_result 'lot_step'],
        point=symbol_result ['point'], tick_size=symbol_result ['tick_size'],
        tick_value=symbol_result ['tick_value'], swap_long= symbol_result [swap_long],
        swap_short= symbol_result [swap_short],)
```

or

Result = MT_Back.Set_instrument_parameters(instrument="EURUSD", digits=5, max_lotsize=200.0, min_lotsize=0.01,lot_stepsize=0.01, point=1e_05, tick_size=1e-05, tick_value=1.0, swap_long=-5.23, swap_short=-0.47)

Result = bool, *True* or *False*

vi. Time frame settings.

The BT uses M1 bars as base. From the base the following timeframe bars can be generated:

- M1, M5, M10, M15, M20, M30, H1, H2, H3, H4, H6, H8, H12 and D1

declare time frame list
timeframe_list = ['M1', 'M15', 'H1', 'H6']

OK = MT_Back.Set_timeframes(timeframe_list=timeframe_list)

OK = bool, *True* or *False*

Generating bars for all timeframes, is time consuming. Specify only the time frames, that are needed for testing your strategy.

6. Reset BT

This function will reset the BT in same way as starting the BT. F.i. if you want to add or change instrument(s), first you have to do a full reset.

Full_reset = MT_Back.Reset_backtester()

Full_reset = bool, *True* or *False*

7. Reset BT pointers

This function will reset the BT internal pointers. If you want to start back testing again with same instruments by using this function the M1 datafiles will not be loaded again.

Reset_pointers = MT_Back.Reset_backtester_pointer()

Reset_pointers = bool, *True* or *False*

8. Set the spread and commission

For every instrument the spread and commission can be set in pips.

For the spread a low level and high level spread value can be set. At opening and closing of trades the spread will be calculated as a random value between low and high level.

Result = MT_Back.Set_spread_and_commission_in_pips(instrument="EURUSD",
low_spread_in_pips=0.6, high_spread_in_pips=1.2, commission = 1.0)

Result= bool *True* or *False*

9. Get data factory info

Datafactory_info = MT_Back.Get_datafactory_info(instrument='EURUSD')

Datafactory_info: is a dictionary, with the following information:

instrument = 'EURUSD'
number_of_bars = total amount of bars in data factory
start_date = date of first bar
end_date = date of last bar

10. Set bar date ascending or descending

In MT4/5 the actual bar has index 0([0]). So the BT will do the same as default. You can do the opposite by using this function.

Result = MT_Back.Set_date_asc_or_desc(asc_desc=True)

asc_desc = True, row [0] is oldest bar
asc_desc = False, row [0] is actual bar

Result = bool, *True*, always

11. Set the bar index for start back testing

Result = MT_back.Set_index_for_start(index=30000)

All defined timeframe pointers are set to 0 at start(bar[0]. If you need as example in your strategy the last 20 D1 bars, then these have to be build. By setting the index to f.i. 30000 the first 30000 M1 bars will be processed and the defined other time frame bars will be calculated in the history of the BT. So in this example 20 D1 bars, 500 H1 bars, etc.

Result = bool, *True* or *False*

12. Go X increments forwards

Result = MT_Back.Go_x_increments_forwards(increments=x)

Result = bool, *True* or *False*

This function increments the pointer by x bars. So for M1 a new bar will be added (x times), For M5 the M5 bar will be updated or when the time is a multiple of 5, a new bar will be added to the M5 list/history, this process will be executed up to D1 bars. This way the BT steps through the bar data.

Remarks:

- For every increment forwards pending orders and active positions will be evaluated.
- If for positions sl and tp are set. There will always be first a check on sl and next on tp. So if we have a big M1 bar which hits sl and tp, it will be a loss.

13. Change time out value

Result = MT5_Back.Set_timeout(timeout_in_seconds=120)

120 = time out value in seconds.

Result = bool, always *True*

14. Retrieve broker server time.

ServerTime = MT_back.Get_broker_server_time()

ServerTime = broker server time. The returned time is the time of the actual M1 bar.

15. Get dynamic account information

DynamicInfo = MT_back.Get_dynamic_account_info()

DynamicInfo = dictionary with the following information:

balance = float, 3400.0
equity = float, 3350.0
profit = float, -50.0

16. Get instrument information

InstrumentInfo = MT_back.Get_instrument_info(instrument='EURUSD')

'EURUSD' = instrument.
InstrumentInfo = dictionary with the following information(if instrument not known, result is "none"):

Instrument = EURUSD
digits = 5
max_lotsize = 200.0
min_lotsize = 0.01
lot_step = 0.01
point = 1e-05
tick_size = 1e-05
tick_value = 1.0

17. Get last tick information

LastTick = MT_Back.Get_last_tick_info(instrument='EURUSD')

LastTick = dictionary with the following information:

instrument=EURUSD
date=1591401419
ask=1.12907
bid=1.129
last=0.0
volume=123

Remarks.

- This function can be used for live streaming of tick data.
- For the BT the smallest info unit is M1 bar. So the values will be the last/actual M1 bar values

18. Get actual bar information

```
ActualBar = MT_Back.Get_actual_bar_info(instrument='EURUSD',
        timeframe=MT_back.get_timeframe_value('H4'))
```

MT_back.get_timeframe_value('H4') = timeframe/period.

ActualBar = dictionary with the following information:

instrument = EURUSD
date = 1591315200
open = 1.13369
high = 1.13838
low = 1.12784
close = 1.129
volume = 98291

This function can be used for live streaming of actual bar data. Remember smallest increment is M1 bar.

19. Get last x bars from now

```
LastBars = MT_back.Get_last_x_bars_from_now(instrument='EURUSD',
        timeframe=MT_back.get_timeframe_value('M1'), nbrofbars=1000)
```

LastBars = array with the following bar info:
Date, open, high, low, close and volume

20. Get a specific bar for list of instruments

```
Specific_bars = MT.Get_specific_bar(instrument_list = instrument_list,
        specific_bar_index=1, timeframe = MT.get_timeframe_value('H1'))
```

Index = bar index, 0= actual bar, 1= last closed bar, etcetera

Specific_bars = Dictionary with for every instrument a dictionary with (d, o, h, l, c, v)

21. Open order

```
NewOrder = MT_back.Open_order(instrument='EURUSD', ordertype='buy', volume=0.01,
        openprice=0.0, slippage=10, magicnumber=2000, stoploss=0.0, takeprofit=0.0,
        comment='Test', open_log='sma cross')
```

open pending order

```
NewOrder = MT_back.Open_order(instrument='EURUSD', ordertype='buy_stop', volume=0.04,
        openprice=1.0870, slippage=10, magicnumber=2000, stoploss=1.0830, takeprofit=1.0950,
        comment='Test', open_log='fibo 0.5')
```

'EURUSD' = instrument.
'buy' = order type ('buy', 'sell', 'buy_stop', 'sell_stop', 'buy_limit', 'sell_limit').
0.02 = volume/lot size.
0.0 = open price. For market orders price will be zero (0.0), for pending orders price must have an appropriate value.
10 = slippage.
1000 = magic number.
1.0830 = stop loss. The stop loss value is a market price (not in delta pips), of 0.0 then no stop loss set.
1.0950 = take profit. The take profit is a market price (not in delta pips), if 0.0 then no take profit set.
Test = comment. The comment may not contain the characters !#$, these are used internally
open_log = additional comment. F.i. open conditions, why open order

NewOrder = ticket, if ticket has the value -1, placing of the order failed.

Remark:
If a ticket has the value -1, the following properties can be checked:

- *MT_Back.order_return_message*. It is a string with the reason for fail.
- *MT_Back.order_error*. It is an integer with MT4 error code.

22. Set SL and TP for position

```
ChangePosition = MT_back.Set_sl_and_tp_for_position(ticket=53136604, stoploss=0.0,
            takeprofit=1.11001)
```

ChangePosition = bool, *True* or *False*, *MT_Back.order_return_message* and *MT_Back.order_error* give more information

23. Reset SL and TP for position

Reset = MT_Back.Reset_sl_and_tp_for_position(ticket=53136604)

Reset = bool, *True* or *False*, *MT_Back.order_return_message* and *MT_Back.order_error* give more information

24. Set SL and TP for order (pendings)

ChangeOrder = MT_back.Set_sl_and_tp_for_order(ticket=53136804, stoploss=0.0, takeprofit=1.12001)

ChangeOrder = bool, *True* or *False*, *MT_Back.order_return_message* and *MT_Back.order_error* give more information

25. Reset SL and TP for order

Reset = MT_Back.Reset_sl_and_tp_for_order(ticket=53136604)

Reset = bool, *True* or *False*, *MT_Back.order_return_message* and *MT_Back.order_error* give more information

26. Get all orders

AllOrders = MT_back.Get_all_orders()

AllOrders = data frame with the following info(only pending orders):
ticket, instrument, order_type, magic_number, volume, open_price, stop_loss, take_profit, comment;

27. Get all (open) positions

AllPositions = MT_back.Get_all_open_positions()

AllPositions = data frame with the following info per position:
ticket, instrument, position_type, magic_number, volume, open_price, open_time, stop_loss, comment, take_profit, profit, swap, commission, dd_min, dd_plus;

Remarks:

- dd_min, is the lowest value of the position
- dd_plus, is the highest value of the position

28. Get all closed positions

AllClosedPositions = MT_back.Get_all_closed_positions()

AllClosedPositions = data frame with the following infoper position:
position_ticket, instrument, order_ticket, position_type, magic_number, volume, open_price, open_time, close_price, close_time, comment, profit, swap, commission, open_log, close_log, dd_min, dd_plus

29. Get last x closed positions

Last_x = MT_Back.Get_last_x_closed_positions(amount=3)

Last_x = data frame with the following info per position:
position_ticket, instrument, order_ticket, position_type, magic_number, volume, open_price, open_time, close_price, close_time, comment, profit, swap, commission, open_log, close_log, dd_min, dd_plus

30. Close position by ticket

ClosePosition = MT_back.Close_position_by_ticket(ticket=597318718)

.
ClosePosition = bool, *True* or *False*.

If ok = False, the properties *MT_Back.order_return_message* and *MT_Back.order_error* can be checked for the reason.

31. Delete order by ticket

DeleteOrder = MT_back.Delete_order_by_ticket(ticket=49988037)

DeleteOrder = bool, *True* or *False*.

If ok = False, the properties. *MT_Back.order_return_message* and *MT_Back.order_error* can be checked for the reason.

32. Set BT for renko bars

The BT can work with OHLC bars or Renko bars, so not both at same time. You can also not switch from OHLC bars into Renko bars and vice versa, when the BT has already build up some history. First a reset (*reset BT pointers*) has to be executed. Default are OHLC bars.
Result = MT_Back.Set_bar_type(bar_type='RENKO')

Result = bool, *True* or *False*

33. Set brick size for renko bars

Result = MT_Back.Set_bartype_parameters-for_renko(brick_list_in_pips=[8.0, 10.0, 12.0]
Brick_list_in_pips = List with brick sizes. A maximum of 3 brick sizes can be defines.

Result = bool, *True* or *False*

34. Retrieve amount of renko bars

With this function it can be checked how many renko bars are in history

Amount = MT_Back.Get_amount_of_renko_bars(instrument='EURUSD', brick_code='B1')

Amount = integer
-1, error, else amount of renko bars

35. Get actual renko bar info

This function is analogue to '*MT_Back.Get_actual_bar_info()*'

Actual_renko_bar = MT_Back. Get_actual_renko_bar_info(instrument='EURUSD', brick_code='B1')

Brick_code = 'B1'or 'B2'or 'B3'

Actual_renko_bar = dictionary with the following information:
instrument = EURUSD
date = 1591315200
open = 1.13369
high = 1.13838
low = 1.12784
close = 1.129
volume = 98291

36. Get last x renko bars from now

```
LastBars = MT_back.Get_last_x_renko_bars_from_now(instrument='EURUSD',
        brick_code='B1', nbrofbars=1000)
```

LastBars = array with the following bar info:
date, open, high, low, close and volume

37. Get a specific renko bar for list of instruments

```
Specific_renko_bars = MT.Get_specific_renkobar(instrument_list = instrument_list,
        specific_bar_index=1, brick_code='B1'))
```

Index = bar index, 0= actual bar, 1= last closed bar, etcetera

Specific_renko_bars = Dictionary with for every instrument a dictionary with (d, o, h, l, c, v)