Artificial Intelligence and Machine Learning in Financial Environments

-

Introduction

Prof. Paulo André L. de Castro
pauloac@ita.br
www.comp.ita.br/~pauloac
Sala 110, IEC-ITA
Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos-SP

Framework mt5b3

- The framework mt5b3 provides access to the B3 market to python programs through Metatrader and some Brazilian brokers (XP, Clear corretora, and others...)

- It allows access to price data (open, close, high, low) and book data (bid, ask) and it also allows order placement.

- mt5b3 provides an API and facilitates the creation of autonomous traders based on traditional algorithms or AI techniques

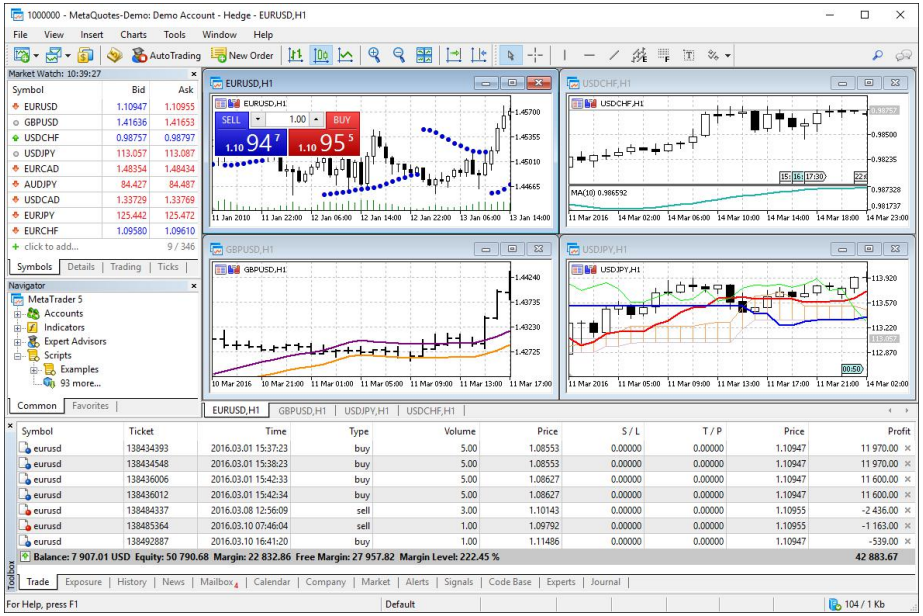
- Primary information source:

https://github.com/paulo-al-castro/mt5b3/

  - Notebooks, examples and tutorials

Metatrader

- MetaTrader 5 is a multi-asset platform offering trading possibilities and technical analysis tools, as well as enabling the use of automated trading systems (trading robots). MT5 is a product of **MetaQuotes Software Corp.**



- MT5 uses a proprietary language called MQL5, which is similar to C++. It is not an open source software, but it is free to use by retail investors and traders

- URL: https://www.metatrader5.com/

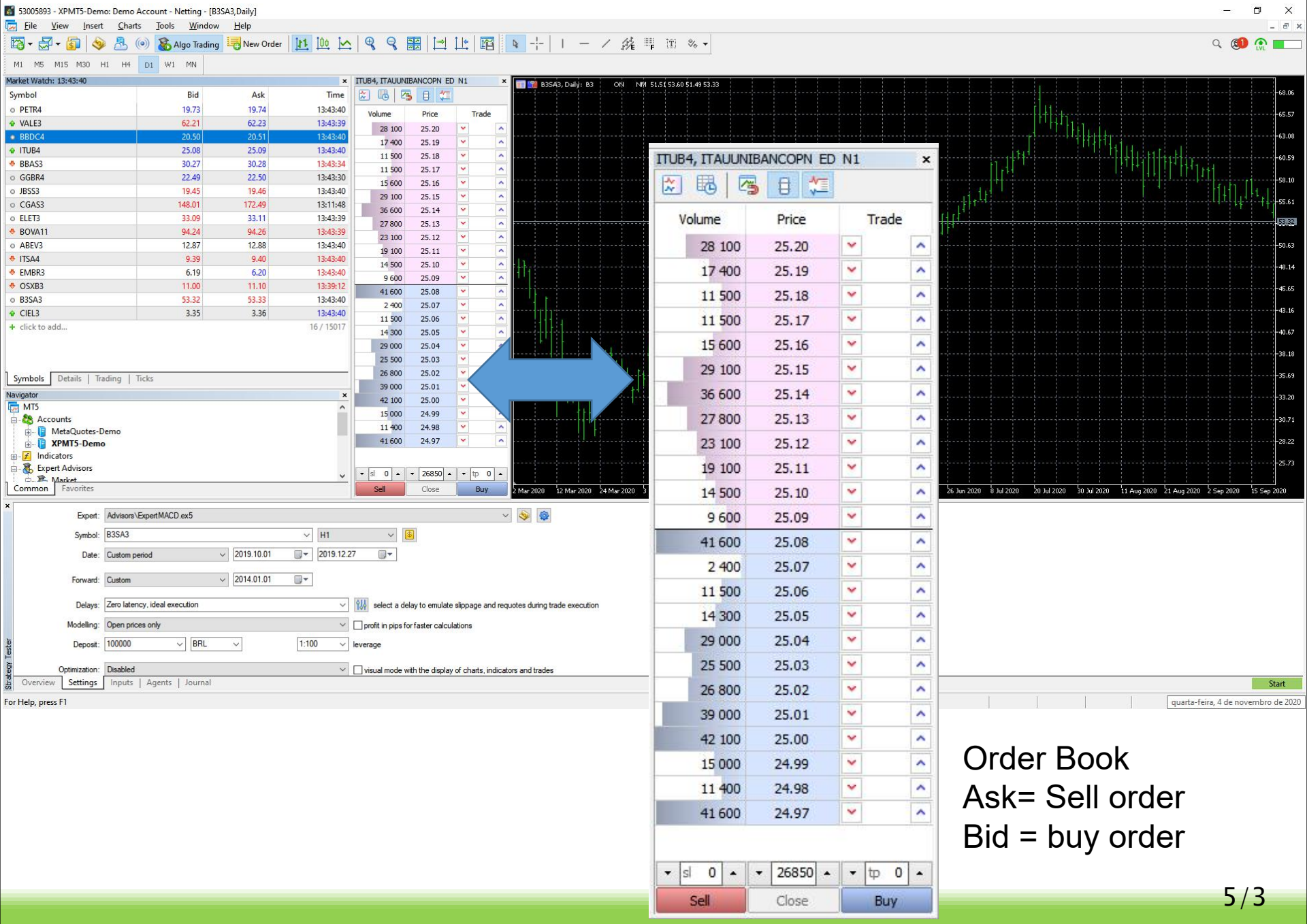
B3 – Brazilian Stock Exchange

- The Brazilian Stock Exchange was founded in 1890 and it had several names since then. It was nominated Bovespa (Bolsa de Valores de São Paulo) in 1967. Finally in 2017, The current name, B3, was adopted

- In 2020, More than 300 companies were listed on B3 and they had a market valuation of 3,6 trilhions of Reais (699 US$ bilhões).

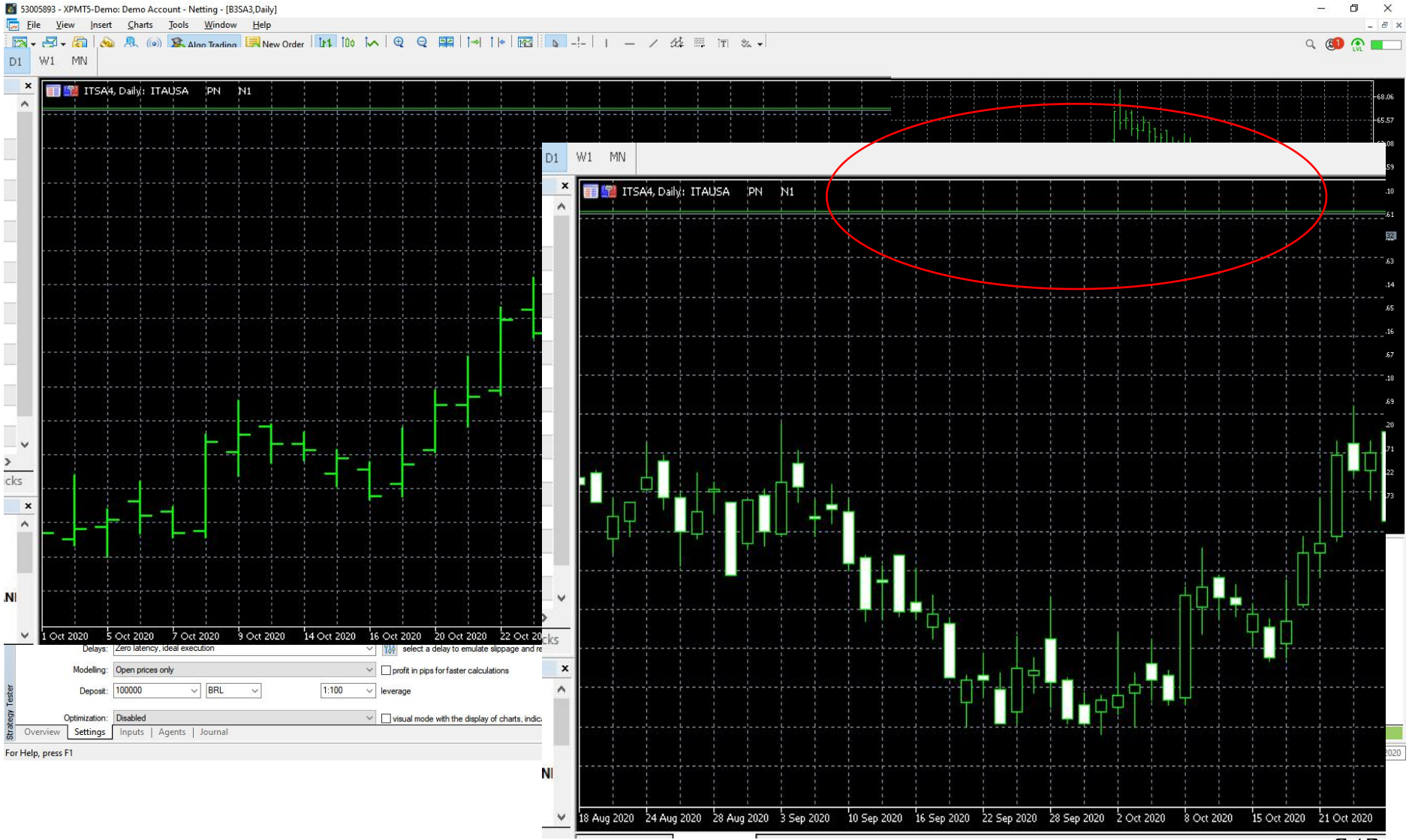
- The number of investors was about 2,9 milions in July/2020. Among those investors, the number of institutional investor was about 30 thousands and 2,82 millions de individual investors.

  - Source: http://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/consultas/mercado-a-vista/valor-de-mercado-das-empresas-listadas/bolsa-de-valores/

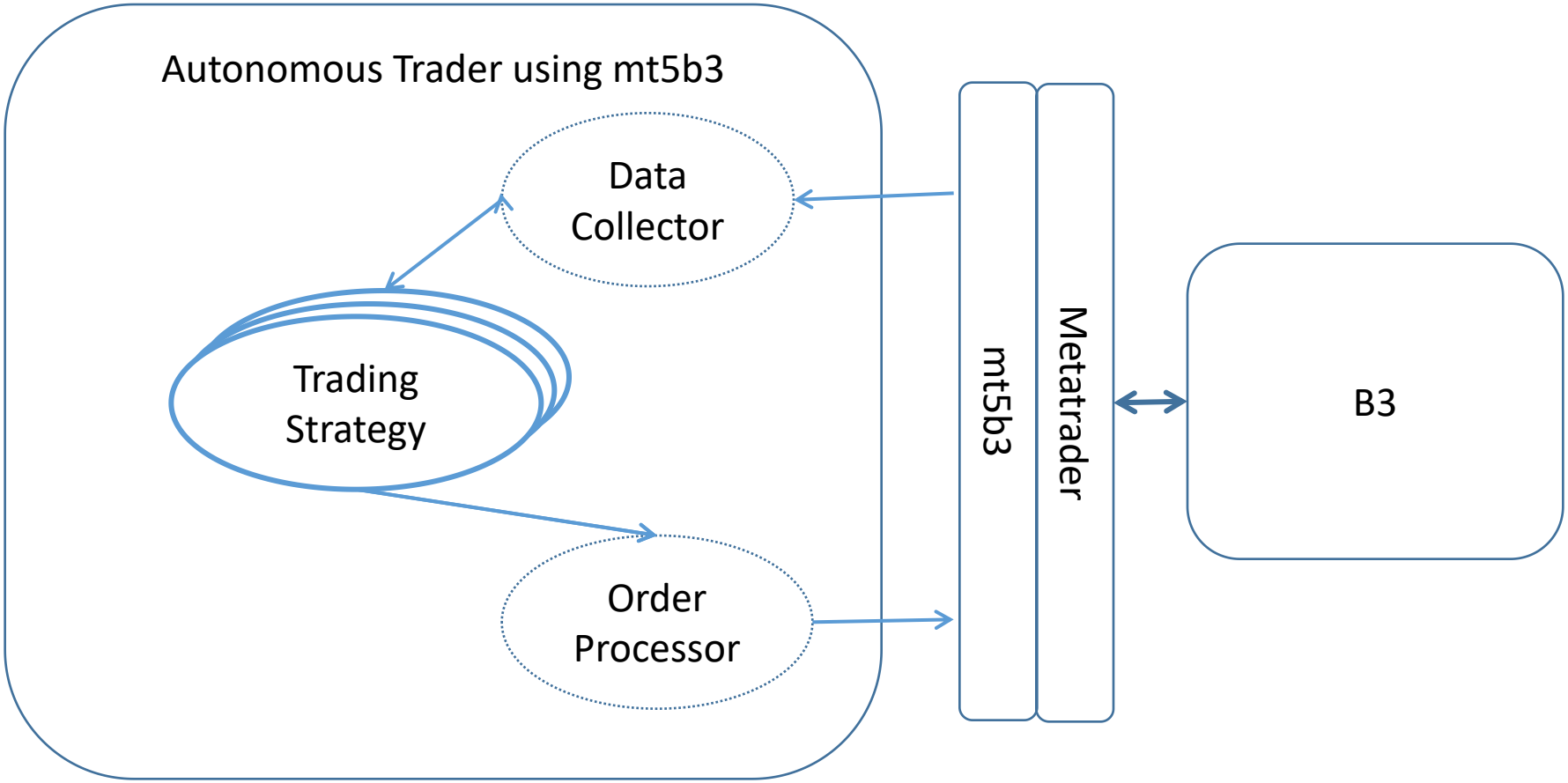


Order Book
Ask= Sell order
Bid = buy order

Bars (open, high,low,close)



Autonomos Trader and mt5b3 Architecture



Simple Trader agent's Components

- **Data Collector**: According to theStrategy, there are setof data points that needs to be collected and properly stored in theMarket informationin a continuous way. That is the role of the Data collector component.

- **Trading Strategy**: it is the implementation of the function that decides the Robot's actions according to current portfolio and Market situation. It can arbirtraly complex and it needs to deal with market prediction and risk management.

- **Order Processor**: Once the Strategy decides which orders to submit according to the observed Market information and current portfolio. These orders are dispatched to some Market (real orsimulated) by the Order processor component to be completely or partially executed or even not executed according to the market conditions. The order processor also updates the portfolio information according to the result of the submitted orders

Package mt5b3

- In order to install the package mt5b3, you need to install package MetaTrader5 as well. You may use pip:

> pip install MetaTrader5

>pip install mt5b3

- or within a jupyter notebook environment

>>import sys

>># python MetaTrader5

>>!{sys.executable} -m pip install Metatrader5

>>#mt5b3

>>!{sys.executable} -m pip install mt5b3

Connecting and getting account information

- import mt5b3 as b3

- if not b3.connect():
    print("Error on connection", b3.last_error())
    exit()

Getting information about the terminal

```
#Terminal info
ti=b3.terminalInfo()
ti.path  # Metatrader program file path
ti.datapath # Metatrader path to data folder
ti.commonDatapath# Metatrader common data path
```

Getting info about the account

```
 # it returns account's information
acc=b3.accountInfo()
print('login=',acc.login)  # Account id
# Account balance in the deposit currency  using buy price of assets
print('balance=',acc.balance)
# Account equity in the deposit currency using current price of assets (capital liquido)
print('equity=',acc.equity)
# Free margin ( balance in cash ) of an account in the deposit currency(BRL)
print('free margin=',acc.margin_free)
#Account margin used in the deposit currency
print('margin=',acc.margin)
print('client name=',acc.name) #Client name
print('Server =',acc.server) # Trade server name
print('Currency =',acc.currency) # Account currency, BRL for Brazilian Real
```

Getting Bars (a.k.a quotes, rates)

```
>>df=b3.getBars('PETR4',10) # it returns the last 10 days
>>df
       time   open   high    low  close  tick_volume  spread  real_volume
0 2020-08-05  17.99  18.89  17.91  18.52        54553       1     52845000
1 2020-08-06  18.55  18.62  17.92  18.06        35795       1     14643400
2 2020-08-07  17.84  18.17  17.27  17.61        35280       1     17218800
3 2020-08-10  17.70  18.32  17.64  18.26        30707       1     15233800
4 2020-08-11  18.46  18.55  18.02  18.15        26154       1     10598000
5 2020-08-12  18.21  18.59  17.75  18.11        29531       1     14836800
6 2020-08-13  18.13  18.37  17.75  17.86        21333       1      9629200
7 2020-08-14  17.85  17.99  17.59  17.99        21537       1      7838400
8 2020-08-17  17.99  18.49  17.75  18.02        34427       1     13390200
9 2020-08-18  18.40  19.70  18.19  19.51        46881       0     32077600
# bar=< time   open   high    low  close  tick_volume  spread  real_volume >
```

Getting Bars (a.k.a quotes, rates)

```
# getting bars in a given period
from datetime import datetime
import pandas as pd

df=b3.getBars('PETR4',datetime(2020,1,1),datetime(2020,3,31))

#  getting intraday (per minute) bars in a specific day
bars=b3.getIntradayBars("ITUB4",datetime(2020,8,17))

# or a period
df=b3.getBars('PETR4',datetime(2020,1,1),datetime(2020,2,1),b3.INTRADAY)

first=b3.getFirstPrice(bars)
last=b3.getLastPrice(bars)
max=b3.getMaxPrice(bars)
min=b3.getMinPrice(bars)
```

Getting info about current position

- b3.getPosition() # return the current list of assets

```
>> pos=getPosition()
>> print(pos)
  symbol   profit    volume  price_open  price_current
0  BBAS3  23750.0  154800.0   38.766576         38.92
1  BBDC4  23360.0   73000.0   27.400000         27.72
```

- b3.getPosition(symbol_id) # return the current position in a given asset (symbol_id)

```
>>getPosition('BBDC4')
  symbol   profit    volume  price_open  price_current
0  BBDC4  23360.0   73000.0   27.400000         27.72
```

- b3.getTotalPosition(symbolId=None) # returns the current total value of the position

```
>>b3.getTotalPosition('BBDC4')
 2023560.0
##  2023560.0 = 73000 * 27.72
>>b3.getTotalPosition()
8048376.0
```

Creating and sending orders

- **# Buying three hundred shares of BBDC4 !!**

```
symbol_id='BBDC4'
volume=300
price=27  # optional
sl=StopLoss=26.5  # optional
tp=TakeProfit=28.5  # optional
b=b3.buyOrder(symbol_id,volume, price, sl, tp )

if b3.checkOrder(b):
    if b3.sendOrder(b): #buying
            print('Buy order sent to B3')
    else:
            print('Error : ',b3.getLastError())
else:
    print('Error : ',b3.getLastError())
```

Creating and sending a Sell Order

```
# Selling !!
# price, sl and tp are optional
s=b3.sellOrder(symbol_id, price, sl, tp, volume)
if b3.checkOrder(s):
    if b3.sendOrder(s): #selling
            print('order sent to B3')
    else:
            print('Error : ',b3.getLastError())
else:
    print('Error : ',b3.getLastError())
```

More functions about creating orders

- b3.order(symbol_id, buyOrder,price, sl, tp, volume) # buy order, if argument buyOrder is True, or sell order if it is False

- b3.buyOrder(symbol_id,volume) # buy order at the current price, without sl (stop loss) or tp (take profit)

- b3.sellOrder(symbol_id,volume) # sell order at the current price, without sl (stop loss) or tp (take profit)

Managing orders

b3.numOrders() #returns the number of active orders

b3.getOrders() # returns a dataframe with all active orders

order_id | buy_sell | volume | price | sl | tp |

- You may cancel an order if you have its id and the order was not yet executed!!

```
if b3.cancelOrder(order_id):
    print('order ',order_id, 'cancelled')
else:
    print('Error when cancelling order',order_id, b3.getLastError())
```

Building mt5b3 robots

- There are two alternative approaches for building mt5b3 robots: direct control and control inversion,
  - Direct control: you can build them using the provided API and using any architecture and keep the coding under direct control ,
  - Control inversion: you use the provided code skeleton that reduces the required effort to build trading robots. You just have to include some functions in order to make your strategy to work. It is a control inversion framework.

Direct control mt5b3 robots

- In order to build a robot from scratch using mt5b3, you will need some like the following code, which is a very simple trading robot based on RSI (Relative Strength Index)(Indice de Força Relativa)

```
# To execute it, run:
if b3.connect()==False:
    print('Unable to connect to B3')
    exit()
else:
    run('PETR4') # trade asset PETR4
```

```
import mt5b3 as b3
import pandas as pd
import time
def run(asset):
    while b3.isMarketOpen():
        print("getting information")
        bars=b3.getBars(asset,14)
        curr_shares=b3.getShares(asset)
        # number of shares that you can buy
        free_shares=b3.getAfforShares(asset)
        rsi=b3.tech.rsi(bars)
        print("deliberating")
        if rsi>=70 and free_shares>0:
            order=b3.buyOrder(asset,free_shares)
        elif  rsi<70 and curr_shares>0:
            order=b3.sellOrder(asset,curr_shares)
        print("sending order")
        # check and send (it is sent only if check is ok!)
        if order!=None:
            if b3.checkOrder(order) and b3.sendOrder(order):
                print('order sent to B3')
            else:
                print('Error  : ',b3.getLastError())
        else:
            print("No order at the moment for asset=",asset )
        time.sleep(1)   # waits one second
    print('Market is now closed!')
```

Multiple asset Robot – 2

- Multiple asset Robot (Example), single strategy for multiple assets, where the resources are equally shared among the assets

```
def runMultiAsset(assets):
    while b3.isMarketOpen():
        for asset in assets:
            bars=b3.getBars(asset,14)    #get information
            curr_shares=b3.getShares(asset)
            money=b3.accountInfo().margin_free/len(assets)   # number of shares that you can buy
            free_shares=b3.getAfforShares(asset,money)
            rsi=b3.tech.rsi(bars)
            if rsi>=70 and free_shares>0:
                order=b3.buyOrder(asset,free_shares)
            elif  rsi<70 and curr_shares>0:
                order=b3.sellOrder(asset,curr_shares)
            else:
                order=None
            if order!=None: # check and send if it is Ok
                if b3.checkOrder(order) and b3.sendOrder(order):
                    print('order sent to B3')
                else:
                    print('Error  : ',b3.getLastError())
            else:
                print("No order at the moment for asset=",asset)
            time.sleep(1)
    print('Market is now closed!')
```

Financial Data processing

- You may process financial data using mt5b3/MetaTrader. You have access to daily or intraday (1-minute interval) for any assets traded in B3 stock exchange

- The data includes open, high, low, close and volume information for each time period (a.k.a bar)

- As shown before, you may get this data using functions getBars or getIntradayBars

```
df=b3.getBars('PETR4',datetime(2020,1,1),datetime(2020,3,31))

#  getting intraday (per minute) bars in a specific day
bars=b3.getIntradayBars("ITUB4",datetime(2020,8,17))
```

Financial Data processing – 2

- Function getIntradayBars returns a Pandas DataFrame with 1-minute bars for the specified day and asset (given it symbol_id)

```
df=b3.getIntradayBars(symbol_id, day)
```

- Function getBars also returns a Pandas Dataframe, therefore you can use any pandas method, for instance:

```
df.to_csv('data.csv') # save the dataframe as a CSV file
```

- There are several ways to use function getBars, it is also possible to get intraday data with it. Some examples:

```
# Returns bars from 1/jan/2020 to 31/Mar/2020
df=b3.getBars('PETR4',datetime(2020,1,1),datetime(2020,3,31))
# returns the last 15 (daily) bars
df=b3.getBars('PETR4',15)
#returns the Intraday bars for the period 1/jan/2020 to 31/Mar/2020
df=b3.getBars('PETR4',datetime(2020,1,1),datetime(2020,3,31),b3.INTRADAY)
```

- Warning: MetaTrader does not keep intraday data for very long, because it would so much data, but it does keep daily data for long periods

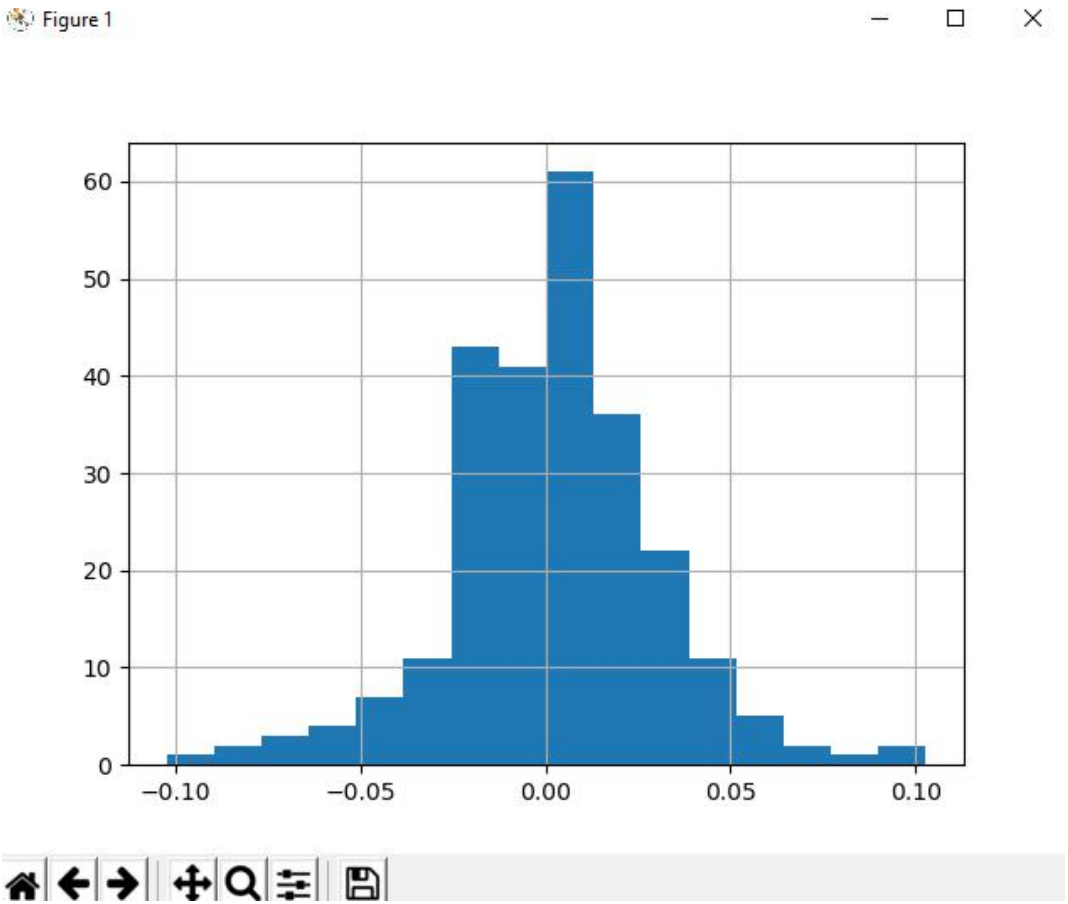
Financial Data processing – Histograms

- You may also create graphs using matplotlib with data from mt5b3/MetaTrader, below you see an example of code that creates a histogram of daily returns

```
import mt5b3 as b3
from datetime import datetime
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

b3.connect()
bars=b3.getBars('ggbr4',252) # 252 business days (basically one year)
x=b3.getReturns(bars) # calcuate daily returns given the bars
plt.hist(x,bins=16)  # creates a histogram graph with 16 bins
plt.grid()
plt.show()
```

Return histogram



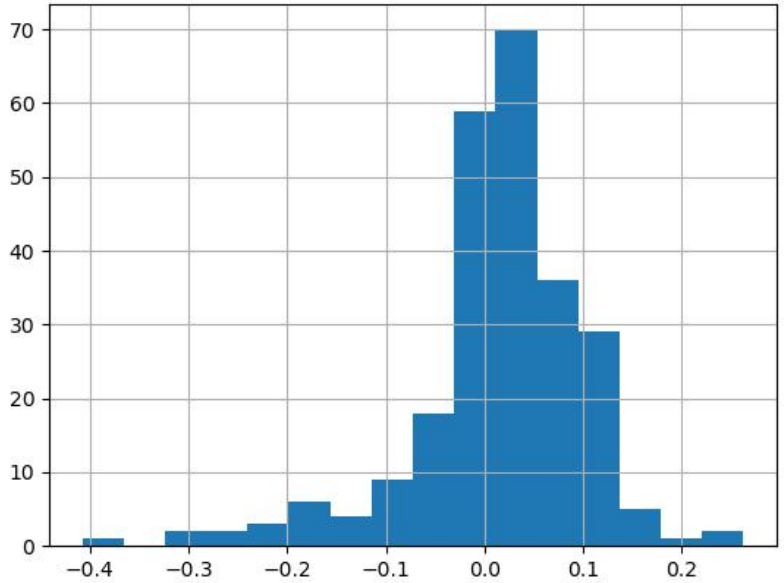
Weekly returns

- With a small change we can see the historgram of weekly returns

....

x=b3.getReturns(bars,offset=5)

...



Robots based on Inversion of control

- You may use an alternative method to build your robots, that may reduce your workload.

- It is called inverse control robots. You receive the most common information requrired by robots and returns your orders

- Let's some examples of Robots based on Inversion of Control including the multiasset strategy presented before in a inverse control implementation

Trader class

- Inversion of control Traders are classes that inherint from b3.Trader and they have to implement just one function:
  - **trade**: It is called at each moment, with dbars. It should returns the list of orders to be executed or None if there is no order at the moment

- Your trader may also implement two other function if required:
  - **setup**: It is called once when the operation starts. It receives dbars ('mem' bars from each asset) . See the operation setup, for more information

  - **ending**: It is called one when the sheculed operation reaches its end time.

- Your Trader class may also implement a constructor function

Example of a Very dummy trader!

```
import numpy.random as rand
import mt5b3 as b3
class DummyTrader(b3.Trader):
    def __init__(self):
            pass
    def setup(self,dbars):
        print('just getting started!')
    def trade(self,bts,dbars):
        orders=[]
        for asset in assets:
            if rand.randint(2)==1:
                order=b3.buyOrder(asset,100)
            else:
            order=b3.sellOrder(asset,100)
            orders.append(order)
        return orders
    def ending(self,dbars):
        print('Ending stuff')
```

Multiple asset Robot – (Inversion of control robot Example)

- Multiple asset Robot (Example), single strategy for multiple assets, implement as inversed control

```
import mt5b3 as b3
import pandas as pd
import time

if b3.connect()==False:
    print('Error when trying to connect to B3')
    exit()
else:
    assets=['PETR4', 'VALE3', 'ITUB4','B3SA3']
    runMultiAsset('PETR4') # trade asset PETR4
```

```
class MyTrader(b3.Trader):
    def trade(self,bts,dbars):
        assets=dbars.keys()
        orders=[]
        for asset in assets:
            curr_shares=bts['shares_'+asset]
            money=bts['capital']/len(assets)
             free_shares=b3.backtest.getAfforShares( \
                        asset,money,dbars)
             rsi=b3.tech.rsi(dbars[asset])
            if rsi>=70 and free_shares>0:
                order=b3.buyOrder(asset,free_shares)
            elif  rsi<70 and curr_shares>0:
                order=b3.sellOrder(asset,curr_shares)
            else:
                order=None
            if order!=None:
                if b3.backtest.checkOrder(order,bts,dbars[asset]):
                    orders.append(order)
                else:
                    print('Error  : ',b3.getLastError())
        return orders
```

Multiple asset Robot – (Direct Control)

- Multiple asset Robot (Example), single strategy for multiple assets, where the resources are equally shared among the assets

```
import mt5b3 as b3
import pandas as pd
import time

if b3.connect()==False:
    print('Error when trying to connect to B3')
    exit()
else:
    assets=['PETR4', 'VALE3', 'ITUB4','B3SA3']  #
    runMultiAsset('PETR4') # trade asset PETR4
```

```
def runMultiasset(assets):
    while b3.isMarketOpen():
        for asset in assets:
            #get information
            bars=b3.getBars(asset,14)
            curr_shares=b3.getShares(asset)
            money=b3.accountInfo().margin_free/len(assets)
            # number of shares that you can buy
            free_shares=b3.getAfforShares(asset,money)
            rsi=b3.tech.rsi(bars)
            #deliberate
            if rsi>=70 and free_shares>0:
                order=b3.buyOrder(asset,free_shares)
            elif  rsi<70 and curr_shares>0:
                order=b3.sellOrder(asset,curr_shares)
            else:
                order=None
            #send order
            # check and send (it is sent only if check is ok!)
            if order!=None:
                if b3.checkOrder(order) and b3.sendOrder(order):
                    print('order sent to B3')
                else:
                    print('Error  : ',b3.getLastError())
            else:
                print("No order at the moment for asset=",asset)
            time.sleep(1)
```

Backtesting

- Backtesting is a kind of evaluation for trading robots: A trading robot executes with historical price series , and its performance is computed.

- In fact, there are many (thousands in some some electronic platforms) robots that [allegedly] are able to be profitable in real markets. Those claims are almost always based on backtest results, and very often they are wrong.
  - many initiatives in Financial Artificial Intelligence misuse mathematical (and machine learning) tools to describe actual observations. Their models are overfit and fail when implemented.
  - The first step to avoid these mistakes is perform meaningful backtest

Backtesting

- In backtesting, time is discretized according with bars and the package mt5b3 controls the information access to the Trader according with the simulated time.

- To backtest one strategy, you just need to create a subclass of Trader and implement one function:
  - trade

- You may implement function setup, to prepare the Trader Strategy if it is required and a function ending to clean up after the backtest is done

- The simulation time advances and in function 'trade' the Trader class receives the new bar info and decides wich orders to send

Multiasset Strategy (Trader class)

```
class MultiAssetTrader(b3.Trader):
    def trade(self,bts,dbars):
        assets=dbars.keys()
        orders=[]
        for asset in assets:
            bars=dbars[asset]
            curr_shares=b3.backtest.getShares(bts,asset)
            # shares the money equally among assets
            money=b3.backtest.getBalance(bts)/len(assets)
            # number of shares that you can buy of asset
            free_shares=b3.backtest.getAfforShares(bts,dbars,asset,money)
            rsi=b3.tech.rsi(bars)
            if rsi>=70 and free_shares>0:
                order=b3.buyOrder(asset,free_shares)
            elif  rsi<70 and curr_shares>0:
                order=b3.sellOrder(asset,curr_shares)
            else:
                order=None
            if order!=None:
                orders.append(order)
        return orders
```

Backtesting

```
import mt5b3 as b3
import pandas as pd
b3.connect()
# sets backtest options
prestart=b3.date(2018,12,10)
start=b3.date(2019,1,10)
end=b3.date(2019,2,27)
capital=100000
results_file='data_equity_file.csv'
verbose=false # Use True if you want debug information for your Trader
#sets the backtest setup
period=b3.DAILY  # it may be b3.INTRADAY (one minute interval)
bts=b3.backtest.set(assets,prestart,start,end,period,capital,results_file,verbose)

# creates instance of your trader
trader=MultiAssetTrader()
#executes the backtest for the trader and returns a pandas dataframe with the results,
# which are also save on data_file
df= b3.backtest.run(trader,bts)
# evaluates the bactest results
b3.backtest.evaluate(df)
```

Evaluating Backtesting results

- The method backtest.run creates a data file with the name given in the backtest setup (bts)
- This will give you a report about the trader performance
- We need to note that it is hard to perform meaningful evaluations using backtest. There are many pitfalls to avoid and it may be easier to get trading robots with great performance in backtest, but that perform really badly in real operations.
- More about that in mt5b3 backtest evaluation chapter.
- For a deeper discussion, we suggest:
  - Is it a great Autonomous Trading Strategy or you are just fooling yourself Bernardini,M. and Castro, P.A.L
- In order to analyze the trader's backtest, you may use :

```
b3.backtest.evaluateFile(fileName)  #fileName is the name of file generated by the backtest
or
b3.backtest.evaluate(df)  # df is the dataframe returned by b3.backtest.run
```

Example of Backtest Report

---------- Backtest Report ----------

Total Return (%)=-3.21 in 33 business days

Annualized Return (%)=-22.06

Std Deviation of returns =0.0117

Sharpe Ratio=-0.0790

Annualized Sharpe Ratio=-1.2546

Probability that Annual Return is greater than 0.0%=45.45%

Probability that Annual Return is greater than 10.0%=42.42%

Probability that Annual Return is greater than 20.0%=36.36%

---------- End of Report ----------

- As you may see in the numbers, it is an example of Backtest Trader report showing a **bad** performance

Deploying Autonomous Traders

- In order, to execute an Autonomous Traders in real (or simulated) account all you need to do is command its executing using b3.operations.run

- Basically, the same code used for backtesting is going to be used in real or simulated execution.. some small changes may be required!

- In order to do connect to B3, you need an account in a broker authorized to operate in B3 stock exchange. For instance:
  - XP Inc., Rico corretora, Clear and others

Live operations with Autonomous Traders in B3 using mt5b3

- With your autonomous trader already built, all you need to do is :

1. Setup the operations parameters
2. Connect to B3 using your account information
3. Create an instance of your trader and execute it giving such parameters

Setup operation!

```
#trading data
assets=['PETR4','VALE3','ITUB4']
capital=100000
endTime=b3.operations.sessionEnd()  # it will run by the end of session!
endTime=b3.operations.sessionEnd()   # it will run to the end of session!
#endTime=b3.now(hourOffset=1,minOffset=30) # the trader will run for 1:30h after started!
waitForOpen=True  # if market is closed, it keeps waiting until it opens
# Trader information
data_file='data_equity_file.csv'
verbose=True
timeFrame=b3.INTRADAY
delay=1 # seconds to wait between trade calls
mem=10  # number of bars to take into account
# define operation setup (ops)  (no b3.backtest.set)
ops=b3.operations.set(assets,capital,endTime,mem,timeFrame,data_file,verbose,delay,waitForOpen)
```

Connect to B3 using your Account Info and Execute your Trader

```
#Connect to B3 using your Account Data
login=None #'your account Number!'
password=None # 'Guess what is it..'
 b3.connect(login,password)
# login and password are optional!! if they are not defined, mt5b3 uses
# the default account defined in MT5

# Create an instance of your trader
import MyTrader()
trader=MyTrader()

#Execute Your trader, according setup  (instead of b3.backtest.run)
 b3.operations.run(trader,ops)
```

Putting it All Together!

```
#trading data
assets=['PETR4','VALE3','ITUB4']
capital=100000
endTime=b3.operations.sessionEnd()  # it will run by the end of session!
waitForOpen=True
# Trader information
data_file='data_equity_file.csv'
verbose=True
timeFrame=b3.INTRADAY
delay=1 # seconds to wait between trade calls
mem=10  # number of bars to take into account
# define operation setup (ops)
ops=b3.operations.set(assets,capital,endTime, \
mem,timeFrame,data_file,verbose,delay,waitForOpen)
```

```
#Connect to B3 using your Account Data
login=None #'your account Number!'
password=None # 'Guess what is it..'
 b3.connect(login,password)
# login and password are optional!!
# b3.connect() : uses the default account defined in MT5

# Create an instance of your trader
import MyTrader()
trader=MyTrader()

#Execute the trader, according setup
 b3.operations.run(trader,ops)
```

Running your Autonomous Trader

- Let's say that the code shown before (Putting it All Together) is in a file named goTrader.py

- Then all you need to do is to run a python interpreter to execute your trader:

>>python goTrader.py

Next:
Artificial Intelligence based Trading Robots

machine learning, probabilistic reasoning, search based trading robots