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

Summary

- AI and Finance
- Brazilian Stock Exchange (B3) and Metatrader
- Framework mt5b3

Can Machines trade [better than us]?

- This question reminds the one raised by Alan Turing in the paper "Can machines think?" [1]. Turing discuss many objections pointed out to reinforce the idea that machines will never be able to really think.

- Some of these objections could be raised against the idea of machines that can analyze investment…but not all of them. For instance, the theological objection. It seems unlikely that someone would argue that analyzing investments is a function of man's immortal soul.

- Let's briefly discuss what seems to us the most relevant objections to machine that can invest…

Objections to Machine that can trade

- **The Mathematical objection**: Trading or asset management is more than logic, it is kind of art, so it is beyond the limits of computability.
  - We use here Turing's short answer "...although it is established that there are limitations to the powers of any particular machine, it has only been stated without any sort of proof, that no such limitations apply to the human intellect..."

- **The Heads in the Sand objection**: The consequences of machines controlling investment would be too dreadful or : Machines controlling investment would steel jobs from real people or even creating catastrophic crises in global markets.
  - Turing answers such objection, stating that this argument is not sufficiently substantial to require refutation and consolation would be more appropriate. We would add that if AIA can be really efficient, perhaps it would be more likely that financial crises would become more rare

- **Other disabilities**: Machines could do significant part of the job, but no machine will ever be able to do X in investment analysis. Numerous features X can be pointed out, for instance: be intuitive, have common sense, be innovative, think something really new.
  - In fact, some of this features can be very hard to achieve, However, there is no hard evidence it is impossible. Furthermore, one may argue that would be possible the existence of an efficient AIA even without common sense, provided it has access to all relevant information

So, why not just give the money to any professional manager ?

- Choosing among professional managers is pretty much the same problem of choosing among investments. Good professionals are really **expensive**, but there is another issue:

- **Conflict of interests**: there are possible conflicts of interest among analysts and investors, when analysts have investments in the target assets themselves or are contracted by securities emitters.
  - In fact, SEC (U.S. Securities and Exchange Commission) has a long history of examining potential conflicts of interests among such roles (please, read the paper and citations to more info about that)

- **Interest of Machines**: Due to the fact that machine can have controlled or at least formally verifiable interests, possible conflict of interests can be avoided or at very least controlled in a more efficient way.

What happens if Algo Traders become ubiquitous?

- Would everybody become rich or at least present very high average returns on their investments?
- The short answer is **no**.
- We believe the scenario described by Eugene Fama [6] in his Efficient Market Hypothesis (EMH) would take place.
- The EMH states that financial markets are efficient in pricing assets. Asset prices would reflect all information publicly available and the collective beliefs of all investors over the foreseeable future.
- Thus, it would not be possible to overcome the performance of the market, using information that is known by the market, except for simple chance

Related Work

- There are **many** initiatives that deal with Algorithmic Trading or some closely related concept, such as autonomous analysis, high frequency trading, automated asset management, automated stock trading and some others terms.

- These initiatives (and also human analysts) can be combined in two big groups:
  - **Technical Analysis** that tries to identify patterns in time series (usually price and volume time series) to predict price trends.
  - **Fundamentalist analysis** that uses economic fundamental information (such as net profit, market share, revenues and so on) to define a "fair" price

- Strategies that deal with short time intervals, (few days or fractions of second) are (almost) always based on technical analysis. In fact, the fundamentalist analysis approach is less used in automated investment analysis regardless of time horizon and despite the fact it is widely used by human asset managers.
  - Probably, it happens because it is harder to deal with many fundamentalist concepts than time series.

- Despite all the work done, there is no system that could be pointed as the "*Deep blue*" of investment.
  - The first machine to win against a world champion

How hard is Autonomous rading?

- The environment faced by the autonomous investment analysts could be classified in a classic way as:
  - partially observable,
  - sequential,
  - stochastic,
  - dynamic,
  - continuous
  - and multiagent,
- Which is the most complex environment class
- However, it does not really represents the whole complexity of the problem.
  - More than stochastic, such environment is also a **non-stationary process** (Probability distributions do change along the time) and it is also strategic in the sense that two traders **compete** for a more accurate valuation of assets and their actions may change other agents' behavior.

Some pitfalls and Challenges in the way…

- The environment for Algo Trading present some key challenges, that sometimes are forgotten or not well understood :
  - It is a non-stationary process
    - Traditional Machine learning approaches usually take the stationarity for granted!!
    - Good performance in ***Back testing*** does not warrant good performance in the market!!
    - Stop loss mechanisms are needed all the time!!
  - It is hard to define what is the relevant information (and what is not relevant). It is not the same for each asset
  - A change in the horizon of investment may change the game!
  - People **do not** have the same acceptable level of risk and return
    - Algo Traders usually do not deal with risk assessment explicitly or Investor profiling!

Some exciting Opportunities and Technologies..

- Deep Learning
  - Convolutional neural networks tem apresentado execelentes resultados no processamento de imagens, mas também em análises de séries temporais (preço, volume,etc.) e podem ser muito úteis na construção de novos algoritmos
- The deep network approach has been tested in different tecnhiques with promising results...
  - Deep Reinforcemente Learning
  - Deep Bayesian Networks
- Agent based Simulations
- Cognitive Computing
  - Specially Natural language processing, Sentiment analysis and human-computer interaction
- A Small Market (like Brazil) may take more time to adjust to new information, it gives opportunities to Algo trading to explore it

Raio X da B3

- Criada em meados 1890, teve várias denomicações e em 1967 recebeu o nome de Bolsa de Valores de São Paulo (Bovespa). Em 2017, adotou a denominação atual B3

- Em 2020, a B3 tinha em torno de 330 empresas listadas com valor de mercado em 3,6 trilhões de reais (699 US$ bilhões).

- O número de investidores era de aproximadamente 2,85milhões, sendo 30 mil institucionais e 2,82 milhões de pessoas físicas, em julho de 2020

  - Data: 31/jul/2020.
  - Fonte: 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/

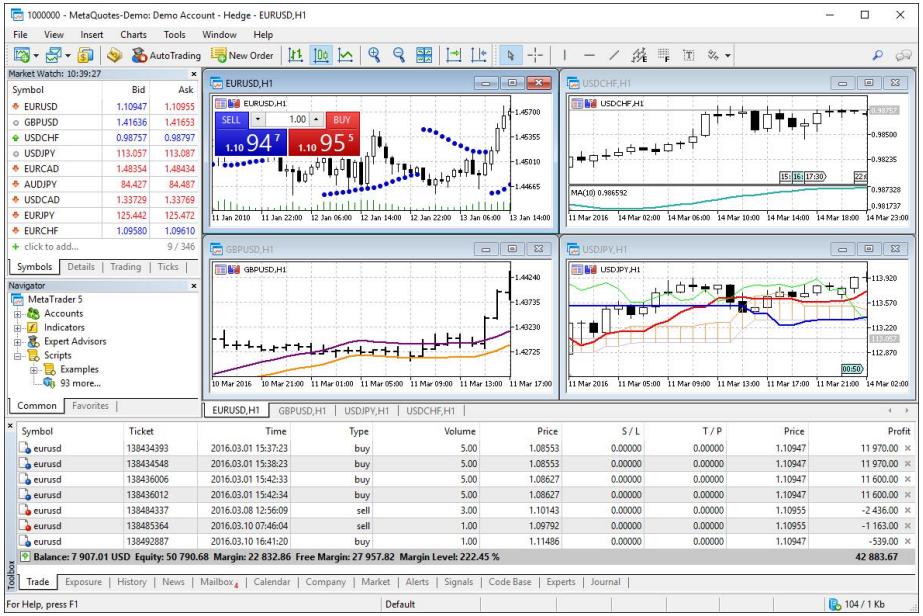
Metatrader

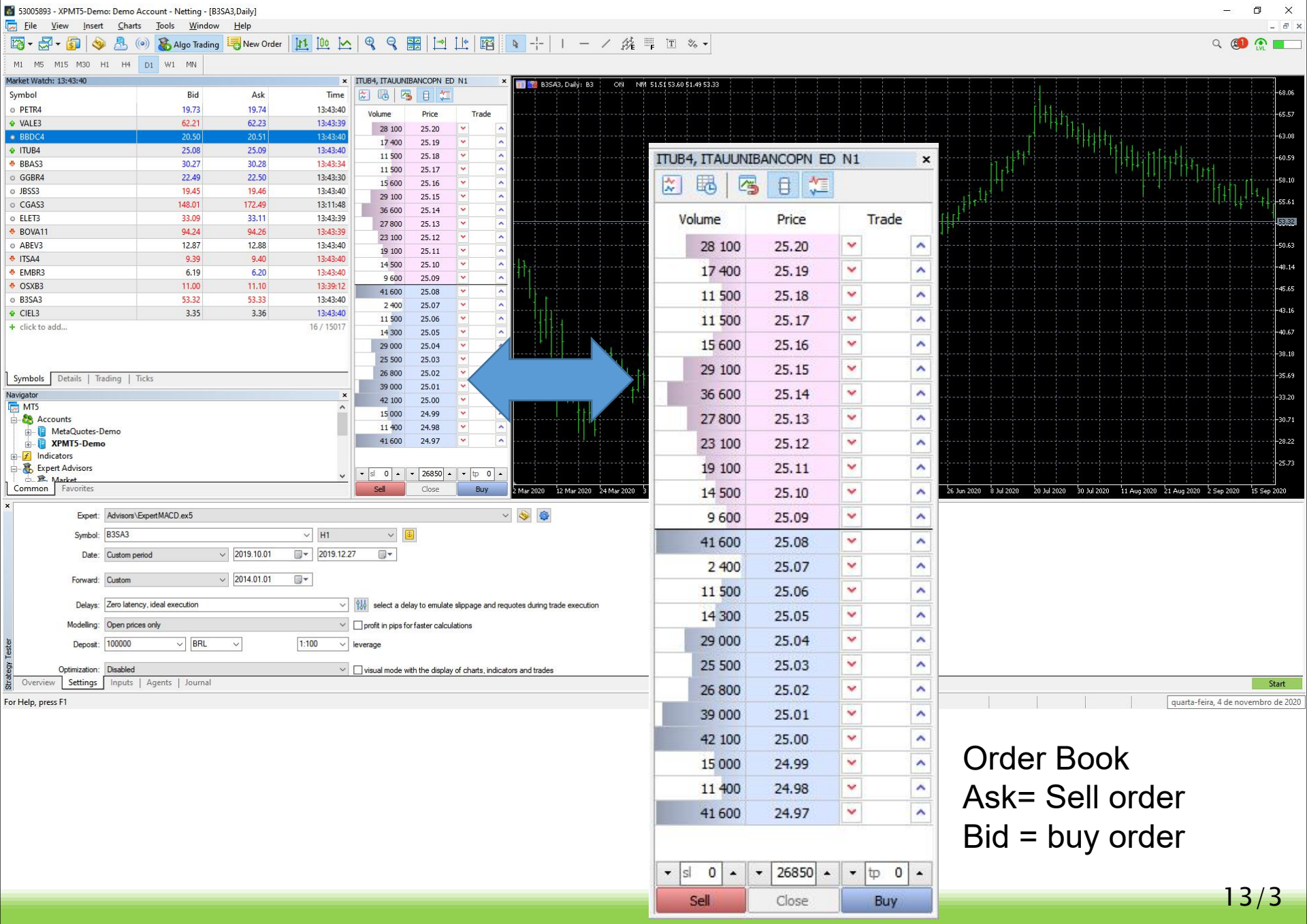
- MetaTrader 5 é uma plataforma gratuita multimercado para trading, análise técnica, uso de sistemas automáticos de negociação (robôs de negociação) da empresa **MetaQuotes Software Corp**.

- O MetaTrader 5 permite negociar nos mercados de câmbio (Forex), ações e futuros, através de interface gráfica ou através de API **e também ativos na B3**

- O MT5 e sua API são escritos na linguagem [proprietária] de programação MQL5, que é similar a linguagem C++

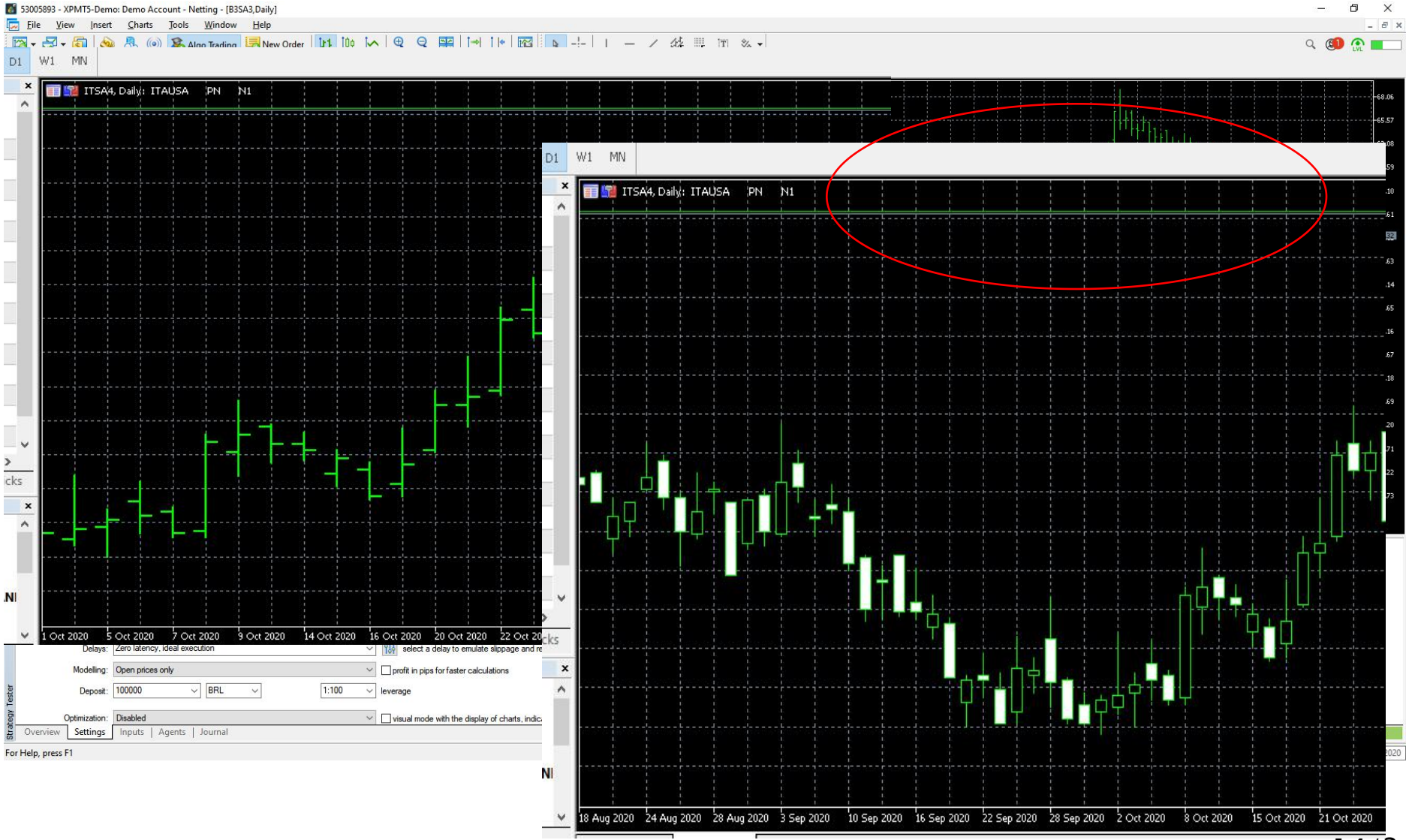
- URL: https://www.metatrader5.com/





Order Book
Ask= Sell order
Bid = buy order

Bars (open, high,low,close)



Package mt5b3

- Package 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)
- It also allows order placement!!
- Primary information source:

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

  - Notebooks, examples and tutorials

Package mt5b3

- Connecting and getting account information
- Getting market information
  - Prices
  - Books (a.k.a pools)
- Managing orders
- Building mt5b3 robots
- mt5b3 Artifcial Intelligence robots

Connecting and getting account information

- import mt5b3 as b3

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

b3.path # Metatrader program file path
b3.dataPath # Metatrader path to data folder
b3.commonDataPath # Metatrader common data path

Getting info about the account

```
acc=b3.accountInfo()   # it returns account's information
acc.login   # Account id
acc.balance # Account balance in the deposit currency
 acc.equity # Account equity in the deposit currency
acc.margin  #Account margin used in the deposit currency
acc.margin_free # Free margin of an account in the deposit currency
acc.assets # The current assets of an account
acc.name #Client name
 acc.server # Trade server name
 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
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)

last=b3.getLastPrice(bars)
open=b3.getOpenPrice(bars)
max=b3.getMaxPrice(bars)
min=b3.getMinPrice(bars)
```

Getting info about position

b3.getPosition() # return the current value of assets (not include balance or margin)

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

- Example:

```
pos=b3.getPosition('ITUB3')
pos['volume'] # position volume
pos['open'] # position open price
pos['time'] #position open time
pos['symbol'] # position symbol id
pos['price']  #current price of the asset
 b3.getPosition(group='PETR*') # returns a list of positions that are part of the group
```

Creating and sending orders

- Buying !!

```
b=b3.buyOrder(symbol_id,volume, price, sl, tp ))

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

Buying !!

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

More functions about creating orders

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

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 |


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 (Indice de Força Relativa)::

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

if b3.connect()==False:
    print('Error when trying to connect to B3')
    exit()
else:
    run('PETR4') # trade asset PETR4
```

```
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:
            order=b3.buyOrder(asset,free_shares)
        else:
            order=b3.sellOrder(asset,curr_shares)
        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," rsi=", rsi, " curr. shares=",curr_shares, "money",money, " affor shares=",free_shares)
        time.sleep(1)   # waits one second
```

Multiple asset Robot – 2

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

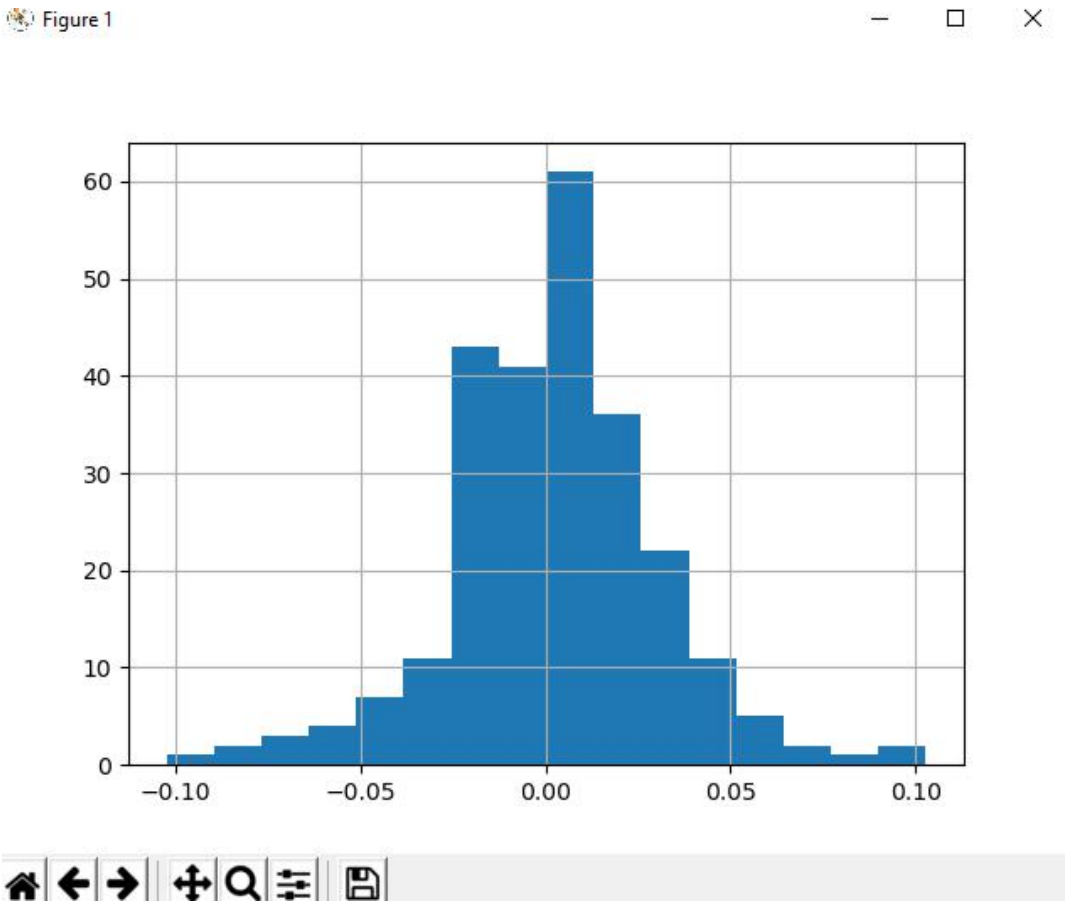
```
import mt5b3 as b3
import pandas as pd
import time

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)
```

Financial Data processing

```
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)
x=b3.getReturns(bars) # gets daily returns serie,
                      #given bars
plt.hist(x,bins=16)  # creates a histogram graph
plt.grid()
plt.show()
```

Return histogram



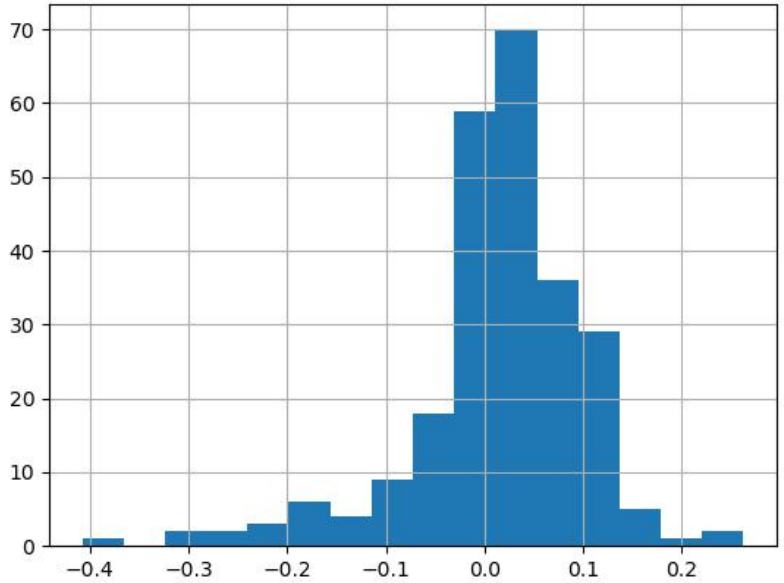
Weekly returns

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

....

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

...



Inversion of control robots

- 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 see the multiasset strategy presented before in a inverse control implementation

Trader

- Inversion of control Traders 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.

- It may also implement a constructor function

Example of Very dummy trader!

```
import numpy.random as rand
import mt5b3 as b3
class DummyTrader:
    def __init__(self):
            pass
    def setup(self,dbars):
        print('just getting started!')
    def trade(self,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)
```

Putting Trader to operate

```
# Setup the operation!
    #account data
login=”your account Number!”
password=”Guess what is this..”
    #trading data
assets=['PETR4','VALE3','ITUB4']
endTime=b3.operations.sessionEnd()
   # it will run to the end of session!
capital=100000
data_file='data_equity_file.csv'
verbose=True
timeFrame=b3.INTRADAY
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)
```

```
#Connect to B3
 b3.connect(login,password)

# creates instance of your trader
 trader=MultiAssetTrader()

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

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

```
class MyTrader(Trader):
    def getNewInfo():
        # your code goes here
    def trade():
        #your code goes here
# you create your trader instance
trader=MyTrader()
...
```

Backtesting

- O tempo em um backtesting é discretizado, de acordo com as cotações utilizadas para definí-lo e o módulo bloqueia acesso a informações a frente do tempo de simulação

- Para testar, uma estratégia basta criar uma subclasse de Trader e implementar os métodos:
  - getNewInfo
  - trade

- Um trader solicita informações atuais (através de getBars, por exemplo) em getNewInfo, envio de ordens nesta etapa não possíveis

- O tempo de simulação avança e no método 'trade' ele pode deliberar e enviar ordens se desejar

Multiasset Strategy (Trader class)

```
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
```

Backtesting

```
import mt5b3 as b3
import pandas as pd
b3.connect()
# sets 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=MyTrader()
#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 ot 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.bactest.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

Next:
Artificial Intelligence based Trading Robots

machine learning, probabilistic reasoning, search based trading robots