Pierre VAMBENEPE (02-01-2021)

TPQ : Final project

Asymmetric Information Detection in options

Abstract : My goal is to use this output to detect events of asymmetrical information in stock options market.

Asymmetrical information can stem from criminal behaviours like insider trading but also from an edge given by advanced research to actors deploying extensive means like the use of mobile phone data, private polls or other types of intelligence gathering along with machine learning treatment of those data. The development of those techniques risks undermining the business model of less specialized actors, including market makers, thus jeopardizing the structure of the market.

Asymmetrical information can be detected in retrospect because they will ultimately lead to a dramatic shift of a parameter such as the spot price, the volatility or the dividend yield.

The goal here is to identify signals in the trading pattern that will alert liquidity providers that something is fishy.

Entrée [1]:

```
#First some global setting
# Importing some libraries:
import numpy as np
import pandas as pd

import requests
import pickle
import os
import warnings
import datetime

import math
import scipy.stats as si
import statsmodels.api as sm
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from keras.models import Sequential
from keras.layers import Dense

warnings.filterwarnings("ignore")
pd.set_option('display.width', 200)
pd.set_option('display.max_columns', 30)
```

Entrée [2]:

```
# Now we create folders where files are going to be written along the way
folder1 = 'D:/Users/GitHub/DBG-PDS/processed'
folder2 = 'D:/Users/GitHub/DBG-PDS/parameters'
folder3 = 'D:/Users/GitHub/DBG-PDS/XY'
folder4 = 'D:/Users/GitHub/DBG-PDS/MLoutput'

for folder in [folder1, folder2, folder3, folder4]:
    os.makedirs(folder, exist_ok=True)

#And download sample data from git to the processed folder (4 pkl files):
for file in ['UDL_DAI.pkl', 'Execs_DAI.pkl', 'UDL_SX5E.pkl', 'Execs_SX5E.pkl']:
    url = "https://raw.githubusercontent.com/pvambenepe/Detecting-asymmetric-informatio
    download = requests.get(url).content
    df = pickle.loads(download)
    print(file)
    print (df.iloc[:5,:5])
    print('')
    df.to_pickle(folder1+'/'+file)


```

```
UDL_DAI.pkl
                      PriceU  ErrorU  TradedVolume
CalcDateTime
2019-01-02 08:00:00  45.8625  0.1625        116414
2019-01-02 08:01:00  45.5200  0.1200         23712
2019-01-02 08:02:00  45.4175  0.1775         23392
2019-01-02 08:03:00  45.4825  0.0775          9455
2019-01-02 08:04:00  45.4850  0.0600         13506

Execs_DAI.pkl
                      PriceU     ErrorU TradedVolume  PriceO  ErrorO
CalcDateTime
2019-01-02 08:02:00  45.4175  39.081852        23392    0.15     0.0
2019-01-02 08:02:00  45.4175  39.081852        23392    3.11     0.0
2019-01-02 08:02:00  45.4175  39.081852        23392    5.02     0.0
2019-01-02 08:02:00  45.4175  39.081852        23392    0.12     0.0
2019-01-02 08:02:00  45.4175  39.081852        23392    0.30     0.0

UDL_SX5E.pkl
                     PriceU ErrorU  TradedVolume
CalcDateTime
2019-01-02 07:00:00    2954      1           858
2019-01-02 07:01:00    2953      1           625
2019-01-02 07:02:00  2950.5    2.5          2131
2019-01-02 07:03:00    2951      1           226
2019-01-02 07:04:00  2950.5    0.5           111

Execs_SX5E.pkl
                     PriceU   ErrorU  TradedVolume PriceO    ErrorO
CalcDateTime
2019-01-02 08:00:00    2949  20.3459         17637    0.8         0
2019-01-02 08:03:00  2942.5  5.09771          3773    4.3         0
2019-01-02 08:07:00  2939.5  5.10291          1899      3  0.680388
2019-01-02 08:08:00  2940.5  5.10117          2613   13.1         0
2019-01-02 08:12:00    2938   10.211          3640    6.5         0
```

Entrée [3]:

```
# Declares the underlyings we are going to use (DAI for Daimler is to be analysed, SX5E
ref = 'SX5E'
indexlist = ['SX5E']
stocks_list = ['SX5E', 'DAI']

```

Entrée [4]:

```
#This part is to take into account bank holidays in the computation of business days

bank_h = ['01-01-2020','10-04-2020','13-04-2020','01-05-2020','01-06-2020','24-12-2020
'01-01-2019','19-04-2019','22-04-2019','01-05-2019','24-12-2019','25-12-2019','26-12-20
'01-01-2018','02-04-2018','01-05-2018','21-05-2018','03-10-2018','24-12-2018','25-12-20
'31-12-2018','14-04-2017','17-04-2017','01-05-2017','05-06-2017','03-10-2017','31-10-20
'26-12-2017']
bank_h_ts = np.array([np.datetime64(pd.Timestamp(elt).date()) for elt in bank_h])

def time_between(a, b):
   nbd = np.busday_count(a.date(), b.date(), holidays=bank_h_ts)
   TimeA = datetime.datetime.combine(datetime.date.today(), a.time())
   TimeB = datetime.datetime.combine(datetime.date.today(), b.time())
   if TimeB > TimeA:
    addhours = (TimeB - TimeA).total_seconds() / 3600
   else:
    addhours = -((TimeA - TimeB).total_seconds() / 3600)
   return (nbd + addhours/8.5)/252


def get_last_working(dt):
    while dt in bank_h_ts:
        dt = dt - datetime.timedelta(1)
    return(dt)
```

Entrée [5]:

```
#This part is here to allow a selection of chosen expiration dates (in this case monthl
#Fist we take every friday then filter out all bar 3rd thursdays

opening_hours_str = "07:00"
closing_hours_str = "15:30"
time_fmt = "%H:%M"
opening_hours = datetime.datetime.strptime(opening_hours_str, time_fmt).time()
closing_hours = datetime.datetime.strptime(closing_hours_str, time_fmt).time()

from_date = '2019-01-01'
until_date = '2019-12-31'
last_matu = '2022-12-31'
dates = list(pd.date_range(from_date, until_date, freq='D').strftime('%Y-%m-%d'))
dates_expi = list(pd.date_range(from_date, last_matu, freq='W'))
dates_expi = [elt - datetime.timedelta(2) for elt in dates_expi]
dates_expi = [datetime.datetime.combine(elt, closing_hours) for elt in dates_expi if el
dates_expi = [get_last_working(elt) for elt in dates_expi]
dates_expi_trim = [elt for elt in dates_expi if elt.month in [3, 6, 9, 12]]

```

This class allows us to graph the results of our processing along the way

It is only used in between tasks to give a sense of what we have achieved.

Entrée [6]:

```
from Graph import Graph
```

Inputs

Deutsche Boerse shares intraday data of trades on single stock options, usable for free for non commercial purposes : https://github.com/Deutsche-Boerse/dbg-pds (https://github.com/Deutsche-Boerse/dbg-pds).

I have used a Docker container in order to retrieve them from Amazon Cloud Services. The next phase was to process them and save them into a pandas dataframe. In order to focus on the essential, I'll skip this part here and provide the pickel of the dataframes of intraday trades for options and underlying for the Eurostoxx50 (SX5E) and Daimler (DAI) I have also reduced the size of the data to 2019 in order to reduce execution time which should still be around 2 hours. The 4 following files should be in the folder "./processed": UDL_DAI.pkl Execs_DAI.pkl UDL_SX5E.pkl Execs_SX5E.pkl (UDL gives the underlying execs and exec gives the options execs)

The next stage is to build a pricer for european and american options, in order to calibrate the parameters of a vol surface each tiçme it is possible, and to calculate sensitivity of a price to some parameters (delta, vega...)

Each underlying and maturity are treated separately.

Every trade is priced only once along with options sensitivity on spot and volatility parameters. The processed is seeded with the parameters obtained with the preceding cluster. First order extrapolation along Spot, ATF, SMI and CVX sensitivities is used thereafter for the calibration of the cluster.

The pricers used are a european Black and Scholes pricer with continuous dividend yield and a binomial tree for american options also with a continuous dividend yield for american options above a certain threshold of dividenyield (we use the european closed formula pricer under this threshold for speed purposes). Even when using a binomial tree, the pricing of american options is not very precise due to the lack of data regarding the exact dividend ex-date. In order to prevent too big an impact from that, we filter out calls with a delta over X%. This inexactitude is not to damaging are we will eventually be looking for big moves in parameters.

The volatility surface model is a simple 2nd degree polynomial equation on moneyness. Sigma(K, T) = ATF(T) - SMI(T) * moneyness + CVX(T) * moneyness² with moneyness = ln(K/F) K = Strike F = Forward

Entrée [7]:

```
from PricingAndCalibration import Pricing
```

The next step is to build a class in order to fit thl curve

I decided to clusterize consecutive trades by groups containing at least 5 traded calls and 5 traded puts.

The idea here is to be able to first perform a calibration of the forward-to-spot ratio prior to the calibration of volatility parameters. This will prevent any miscalibration resulting from a sudden change of this ratio following either a dividend payment or a change in the dividend forecast.

For that, we will perform a WLS (weighted OLS) to determine which forward-to-spot ratio fits best the n trades in the cluster. The weights are used to balance the positive and negative delta (call and puts) in the cluster.

With one row per trade in the cluster so :

X = [sensi_delta_opt1 sensi_delta_opt2 sensi_delta_opt3 ...]

Y = [Traded_price_opt1 - Model_price_opt1_with_param(t-1) Traded_price_opt2 - Model_price_opt2_with_param(t-1) Traded_price_opt3 - Model_price_opt3_with_param(t-1) ...]

the result of the regression gives the shift to be applied to the forward-to-spot ratio (cf code get_new_fwd_ratio in the Fitting class)

Once this is done, we want to see how to alter volatility parameters in order to best fit the traded prices of the cluster. We will be starting by pricing the trades with the parameters of the previous cluster along with the associated sensitivities (sensi_vega, sensi_smile...). We will then be using an Elastic Net Regression rather than OLS in order to give more rigidity to parameters with less variability like smile and convexity. Each row corresponds to a trade in the cluster so :

X = [sensi_vega_opt1 * std_vega, sensi_smile_opt1 * std smile, sensi_convex_opt1 * std_convex

sensi_vega_opt2 * std_vega, sensi_smile_opt2 * std smile, sensi_convex_opt2 * std_convex

sensi_vega_opt3 * std_vega, sensi_smile_opt3 * std smile, sensi_convex_opt3 * std_convex

...]

Y = [Traded_price_opt1 - Model_price_opt1_with_param(t-1)

Traded_price_opt2 - Model_price_opt2_with_param(t-1)

Traded_price_opt3 - Model_price_opt3_with_param(t-1)

...]

We look for a vector alpha which minimizes: ||Y-X * alpha||2 + epsilon1||*alpha||1 + epsilon2*||alpha||2 (see elastic net regression) The result gives the move to apply to parameters :

ATF(t) = ATF(t-1) + alpha[0]*std_vega

SMI(t) = SMI(t-1) + alpha[1]*std_smile

CVX(t) = CVX(t-1) + alpha[2]*std_cvx

(see function "get_new_vols_params" in the Fitting class)

Entrée [8]:

```python
from PricingAndCalibration import Fitting
```

We will now use the fitting class in order to get a dataframe of parameters for both DAI and SX5E

Entrée [9]:

```
P = Pricing()

for udl in stocks_list:
    print(udl)

    res = pd.DataFrame()

    for MaturityDate in dates_expi:
#         print("Matutity Date : " + MaturityDate.strftime("%m/%d/%Y"))
        fit = Fitting(folder1, udl, MaturityDate)
        if fit.bigEnough:
            fit.clusterize()

            while (fit.cluster.shape[0] > 0) and (fit.df.loc[fit.last_index, 'timeOfTra
                fit.reref()
                fit.price_cluster(udl)
                possible = fit.get_new_fwd_ratio()
                if possible:
                    oldATF = fit.ATF
                    oldSMI = fit.SMI
                    oldCVX = fit.CVX
                    fit.get_new_vols_params()
                    if (fit.min_nb_opt_per_cluster == fit.start_min_nb_opt_per_cluster)
                        ((abs(fit.ATF-oldATF)>fit.stdParams[0]*4) or (abs(fit.SMI-oldSM
                        fit.min_nb_opt_per_cluster += 1
                    else:
                        fit.write_down()
                        fit.min_nb_opt_per_cluster = fit.start_min_nb_opt_per_cluster
                else:
                    fit.min_nb_opt_per_cluster += 1

                fit.clusterize()

            fit.compute_EWMA() #halftime in hours
            if res.shape[0] == 0:
                res = fit.df_params.copy()
            else:
                res = res.append(fit.df_params, ignore_index=True)

    # Filter out if Error is too big
    before = res.shape[0]
    compare_range = max(30, int(before/100/2))
    res.to_pickle(folder2 + '/Parameters_before_filter_' + udl + '.pkl')
    res = res.sort_values(by='StartTime', ascending=True)
    global_mean = res.Error.mean()
    res['maxE'] = res.Error.rolling(compare_range*2).mean().shift(periods=-compare_rang
    res = res.loc[res.Error < res.maxE]
    print('Pct rows out because of error threshold' + str(1 - res.shape[0] / before))

    #take out TTM column
    del res['TTM']

    res.to_pickle(folder2 + '/Parameters_' + udl + '.pkl')
    print(res[['StartTime', 'ATF', 'SMI', 'CVX',  'FwdRatio', 'Error']].iloc[:5,:])
```

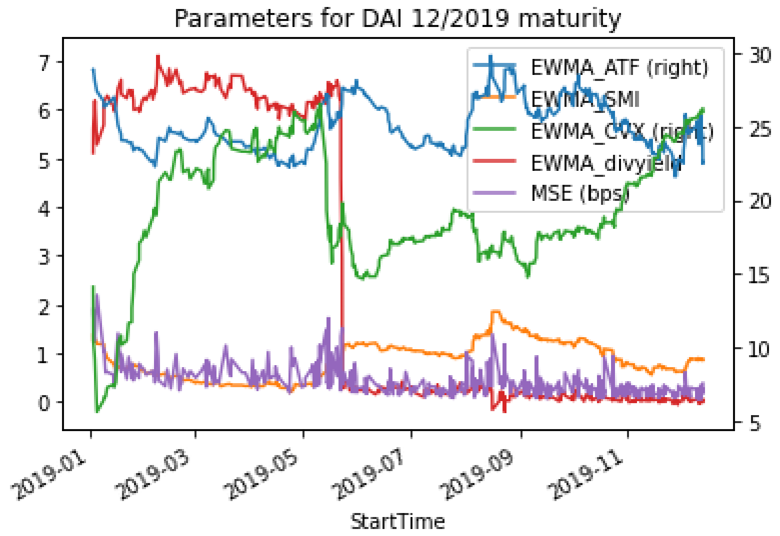
```
Matutity Date : 11/19/2021

Matutity Date : 12/17/2021
```

```
Matutity Date : 01/21/2022

Matutity Date : 02/18/2022

Matutity Date : 03/18/2022

Matutity Date : 04/15/2022

Matutity Date : 05/20/2022

Matutity Date : 06/17/2022

Matutity Date : 07/15/2022

Matutity Date : 08/19/2022
```

*The output of this code is pandas dataframes giving time series of the following calibrated parameters : ATF, SMI, CVX, divyield along with traded volumes. *

Entrée [10]:

```
# We will now visualize the output ;
# The trades have been used to calibrate a parametric vol surface.
# You can choose the period and mutirity you want to graph

g = Graph('DAI')
g.graph_params(year=2019, month =12)
```



Obtaining proper time series

A set of parameters were thus generated for each batch of 5 trades for any given maturity. This leads to a series that is unevenly sampled. Thanks to some extrapolation (1 day max) the BuildInputs class transform these data into a proper time series of parameters sampled on 1 minute intervals.

Separately, we also compute time series deriving from the actual trades (the sames that we used to calibrate the vols parameters). Trades are converted into sensitivity. For example : vega = Qty * quotity * vega of the option The vega of the option is given by a pricing using the Pricing class (same used for the calibration)

Going further still, these sensitivities are "signed" in order to (try to) signify if it was a buying or selling interest. The disputable hypothesis here is that if the parameter is going up, then it means that trades happening at this time are responsible for this trend (since trades are what is used to do the calibration) so these trades must be buying the sensitivity (vega or other).

Once merged into the df_pivot dataframe, those inputs will be used to compute the actual X and Y for our machine learning project.

The idea here is to later associate (through supervised Machine Learning) a significant pattern on one parameter to a later surge in the associated parameter : This will happen if someone had the information leading to this surge before the rest of market participants and tried to take advantage of this knowledge.

Entrée [11]:

```
from BuildInputs import BuildInputs
```

And now we use this class on SX5E and DAI

Here is how we use the BuildInputs class in order to transform fitted trade clusters into 1 minute sampled dataframed by interpolation.

Entrée [12]:

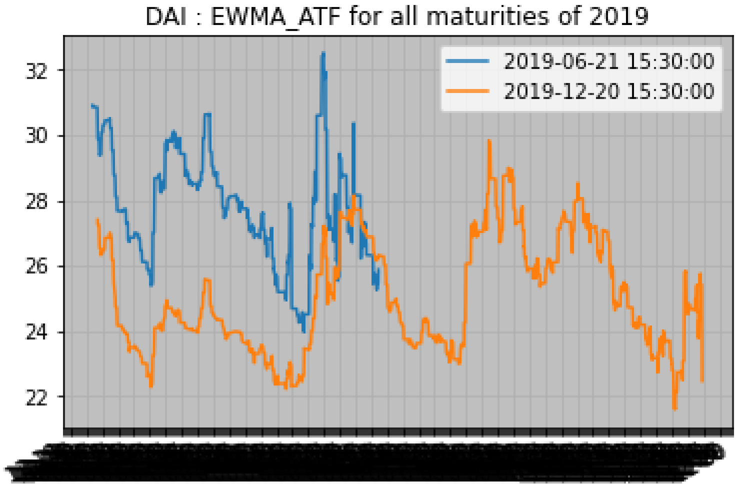
```
for udl in stocks_list:
    print(udl)
    df = pd.DataFrame()
    for pos, matu in enumerate(dates_expi):
        build = BuildInputs(udl, matu)
        if build.df_params.shape[0] > 10:
            build.even_index()
            build.get_total_sensi()
            build.merge()
            df = df.append(build.df)

    df.to_pickle(folder2 + '/Inputs_' + udl + '.pkl')
    print(df[['EWMA_ATF', 'EWMA_FwdRatio', 'TotalSignedSensiATF' , 'TotalSensiFwdRatio'
```

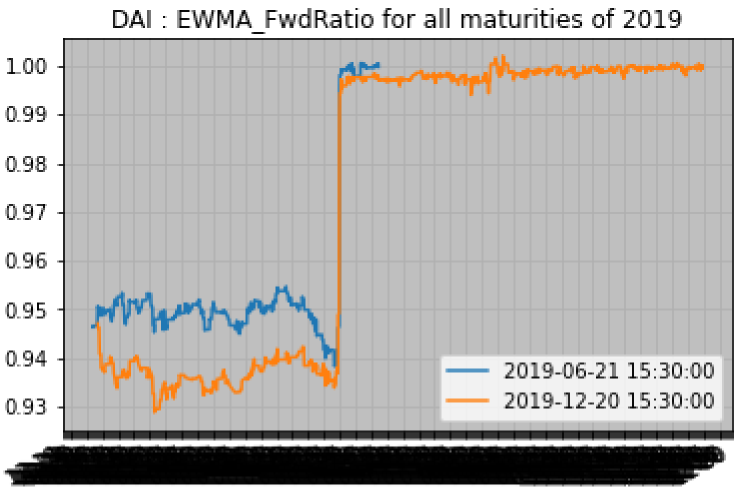
```
SX5E
                       EWMA_ATF  EWMA_FwdRatio  TotalSignedSensiATF  TotalSen
siFwdRatio
2019-01-04 09:24:00  19.244342       1.003453                  0.0         0.
000000e+00
2019-01-04 09:25:00  19.240650       1.003452                  0.0         7.
456537e-18
2019-01-04 09:26:00  19.236959       1.003451                  0.0         2.
146148e+01
2019-01-04 09:27:00  19.233267       1.003450                  0.0         0.
000000e+00
2019-01-04 09:28:00  19.229576       1.003448                  0.0         0.
000000e+00
DAI
                       EWMA_ATF  EWMA_FwdRatio  TotalSignedSensiATF  TotalSen
siFwdRatio
2019-01-02 13:01:00  35.241708       1.000080                  0.0
0.0
2019-01-02 13:02:00  35.238592       1.000069                  0.0
0.0
2019-01-02 13:03:00  35.235476       1.000058                  0.0
0.0
2019-01-02 13:04:00  35.232361       1.000047                  0.0
0.0
2019-01-02 13:05:00  35.229245       1.000037                  0.0
0.0
```

The output of this code is pandas dataframes giving time series of the following calibrated parameters : ATF, SMI, CVX, divyield along with traded volumes :

Entrée [13]:

```
# This shows the vol (resp de forward ratio) for 2 maturities once it has been turned i

g = Graph('DAI')
g.graph_inputs(year = 2019, expi_month=[6, 12], field='EWMA_ATF')
g.graph_inputs(year = 2019, expi_month=[6, 12], field='EWMA_FwdRatio')
```





The final preparation work is to build the Xs and Ys for our Machine Learning process

The BuildXY class aims to convert those time series into "stationary" ones

The idea here is that we are trying to detect a trading pattern that may be common to all instances of asymmetric information. We need to iron out everything that is specific to a certain period or stock, along with false positives, in order to get clean data.

Here are the steps taken :

1/ Look at forward parameters : This cleans out any punctual event like a dividend ex date that will inevitably prompt a jump in the spot/forward ratio but not in a "forward - spot/forward ratio" which is computed as the ratio of two spot/forward ratio on different maturities (typically a long term one divided by the nearby). Similarly, when a company publishes its earnings report, the implicit vol will collapse but the forward vol will be mostly unaffected on average, which is what we want. This is what the differentiate_matu function does.

2/ Short term trend divided by long term trend : This simply uses exponentially weighted moving average in order to detect short term variation of volume/level from a long term average. This is done by the differentiate_time function. This is also where the Y series is generated : it simply consist in observing parameters variation over the 5 days following the date at which we oberve the Xs.

3/ For each single stock underlying (here only DAI), we divide its time-series by the corresponding time series of the euro stoxx 50 index : This will iron out market regimes.

Entrée [14]:

```
from BuildXY import Data
```

And now we use this class on DAI and SX5E

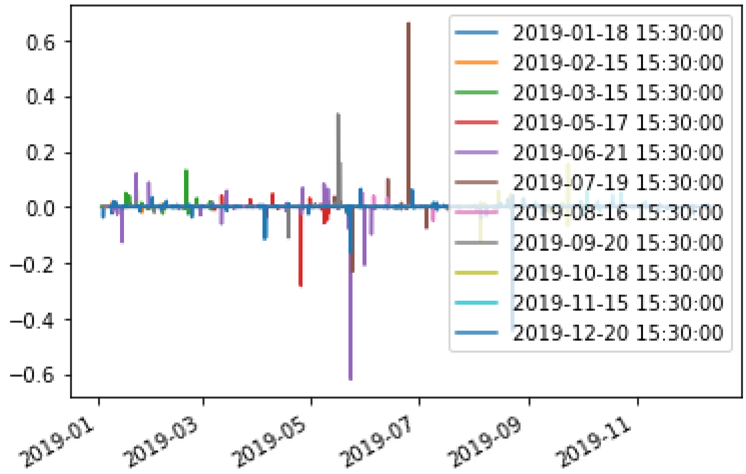
Entrée [15]:

```
# Step 1 : Look at forward parameters : This will solve dividends and earnings publicat
for udl in stocks_list:
    print(udl)
    data = Data(udl)
    data.differentiate_matu()
    data.df_pivot.to_pickle(folder3 + '/X_' + udl + '.pkl')

    df_display = data.df_pivot.xs(pd.Timestamp('2019-12-20 15:30:00'), level=1, axis=1,
    print(df_display.loc[df_display.index<pd.Timestamp('2019-11-01 15:30:00')][['EWMA_A

```

```
SX5E
EWMA_ATF
EWMA_FwdRatio
TotalSignedSensiATF
TotalSignedSensiSMI
TotalSignedSensiFwdRatio
EWMA_SMI
TotalSensiATF
TotalSensiSMI
TotalSensiFwdRatio
NumberOfTrades
                      EWMA_ATF  EWMA_FwdRatio  TotalSensiATF  TotalSensiFwdR
atio
2019-11-01 15:25:00  12.641235       0.998716       0.000000             0.00
0000
2019-11-01 15:26:00  12.637289       0.998701      14.321359            26.43
4467
2019-11-01 15:27:00  12.633343       0.998686      50.786179             0.00
0326
2019-11-01 15:28:00  12.629398       0.998671       0.000000             0.00
0000
2019-11-01 15:29:00  12.625452       0.998656       0.000000             0.00
0000
DAI
EWMA_ATF
EWMA_FwdRatio
TotalSignedSensiATF
TotalSignedSensiSMI
TotalSignedSensiFwdRatio
EWMA_SMI
TotalSensiATF
TotalSensiSMI
TotalSensiFwdRatio
NumberOfTrades
                      EWMA_ATF  EWMA_FwdRatio  TotalSensiATF  TotalSensiFwdR
atio
2019-11-01 15:25:00  24.017944       0.999886            0.0
0.0
2019-11-01 15:26:00  24.015677       0.999893            0.0
0.0
2019-11-01 15:27:00  24.013409       0.999900            0.0
0.0
2019-11-01 15:28:00  24.011141       0.999907            0.0
0.0
2019-11-01 15:29:00  24.008874       0.999914            0.0
0.0
```

Entrée [16]:

```
#Here is a representation of the vector obtained
g = Graph('DAI')
g.graph_X(year=2019, field='TotalSignedSensiATF')

#This shows the traded vega with a sign depending on whether the interest was buyer or
```



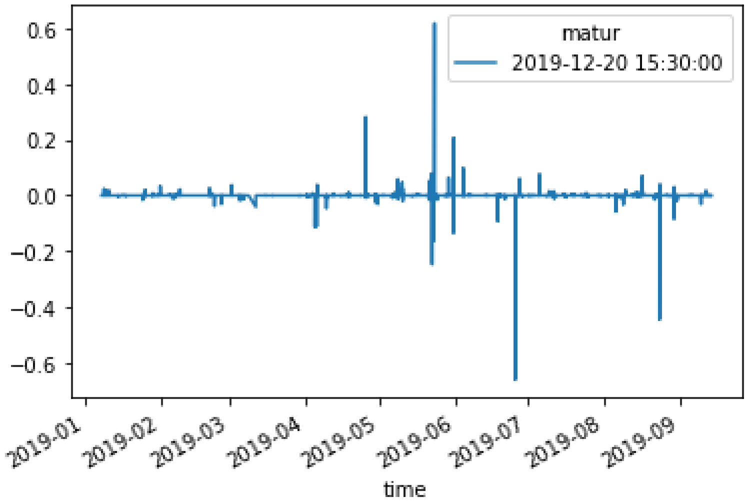
Entrée [17]:

```
# Step 2 : We smooth eh X parameters by dividening short term trend by Long term trend
# We also compute the Y vector by calculating the variation of the Y parameter over x

st = 2 #in days : short term ewm for X time differentiation
lt = 20 #in days : Long term ewm for X time differentiation
Ylag = 5 #in days How Long we Look into the future for the Y vector

for udl in stocks_list:
    print(udl)
    data = Data(udl)
    data.df_pivot = pd.read_pickle(folder3 + '/X_' + udl + '.pkl')
    data.differentiate_time(st, lt, Ylag)  #...and compute Y
    data.X.to_pickle(folder3 + '/Xtd_' + udl + '.pkl')

    print(data.X.dropna().iloc[:5,-5:])
```

```
2019-09-20T15:30:00.000000000
EWMA_ATF
EWMA_FwdRatio
TotalSignedSensiATF
TotalSignedSensiSMI
TotalSignedSensiFwdRatio
EWMA_SMI
TotalSensiATF
TotalSensiSMI
TotalSensiFwdRatio
NumberOfTrades
TradedVolume
2019-10-18T15:30:00.000000000
EWMA_ATF
EWMA_FwdRatio
TotalSignedSensiATF
TotalSignedSensiSMI
TotalSignedSensiFwdRatio

EWMA_SMI
```

Entrée [18]:

```
#Here is a representation of the vector obtained
# Since this is after time differentiation, peakss correpond to sudden rises in the veg

g = Graph('DAI')
g.graph_XY(year=2019, month=12, field='TotalSignedSensiATF', filtertype = 1, stage='xdt
```



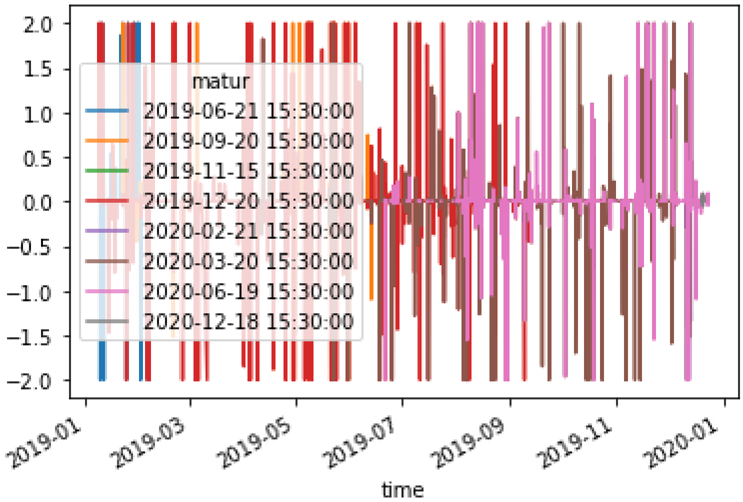
Entrée [19]:

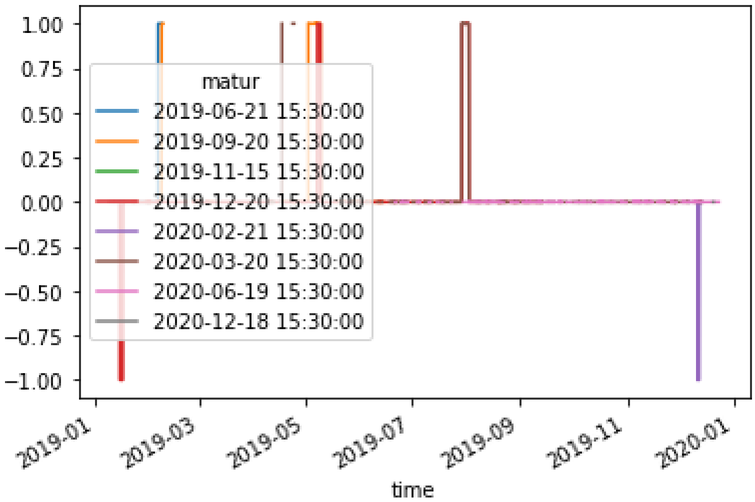
```
# Step 3 : we divide by the ref index (SX5E), apply cap-floor, and change Ys into grou
# Machine Learning (ML) algo to identify times when there has been a sudden shift in th

filter_type = 1
cap = 2

for udl in stocks_list:
    data = Data(udl)

    data.X = pd.read_pickle(folder3 + '/Xtd_' + udl + '.pkl')

    print('filter')
    data.filter(TTM=1, type=filter_type)  #TTM in years;  type in 1:fwd only, 2:nearby

    if udl not in [ref]:
        XRef = pd.read_pickle(folder3 + '/XY_' + ref + '.pkl')

        data.differentiate_refindex(XRef, exclude=['FwdRatio', 'TotalSignedSensi'])

        data.normalize(cap)
        #is it better to normalize before or after joining the underlyings? Both have p

        data.X.to_pickle(folder3 + '/XY_' + udl + '.pkl')
    else:
        data.X.to_pickle(folder3 + '/XY_' + udl + '.pkl')

    print(data.X.dropna().iloc[:5,-5:])

```

```
filter
                                          TotalSensiFwdRatio  dt-NumberOfTrad
es  NumberOfTrades  dt-TradedVolume  TradedVolume
                    Matu
2019-01-23 14:08:00 2019-06-21 15:30:00             0.000000          -0.1869
92              0.0        -0.062499         798.0
2019-01-23 14:09:00 2019-06-21 15:30:00             0.000000          -0.1883
33              0.0        -0.061677        1308.0
2019-01-23 14:10:00 2019-06-21 15:30:00           -12.289365          -0.1884
43              1.0        -0.062247         540.0
2019-01-23 14:11:00 2019-06-21 15:30:00             0.000000          -0.1885
53              1.0        -0.062930         477.0
2019-01-23 14:12:00 2019-06-21 15:30:00             0.000000          -0.1898
91              0.0        -0.063220         693.0
filter
                                          TotalSensiFwdRatio  dt-NumberOfTrad
es  NumberOfTrades  dt-TradedVolume  TradedVolume
                    Matu
2019-01-21 08:49:00 2019-06-21 15:30:00             0.039308           0.4617
93        -0.284745         0.556402     -0.040345
2019-01-21 08:50:00 2019-06-21 15:30:00             0.039308           0.4567
52        -0.284745         0.550986     -0.081955
2019-01-21 08:51:00 2019-06-21 15:30:00             0.039308           0.4704
06         0.078311         0.549874     -0.101477
2019-01-21 08:52:00 2019-06-21 15:30:00             0.039308           0.4653
51        -0.284745         0.549215     -0.100217
2019-01-21 08:53:00 2019-06-21 15:30:00            -0.009670           0.4696
41         0.441368         0.542142     -0.115579
```

Entrée [20]:

```
#Here is a representation of the vector obtained
g = Graph('DAI')

g.graph_XY(year='', field='TotalSignedSensiATF', stage='xy')
g.graph_XY(year='', field='Y-EWMA_ATF', stage='xy')

# On the 1st graph, data heve been capped/floored at 2 standard deviation hence the val
# On the 2nd graph, we can see indtances of sudden shift for different maturities
```





Entrée [21]:

```
#We finally merge all the dataframe for all single stocks (here only DAI)

df = pd.DataFrame()
for udl in [elt for elt in stocks_list if elt not in indexlist]:
    dft = pd.read_pickle(folder3 + '/XY_' + udl + '.pkl')
    dft['udl'] = udl
    df = df.append(dft)

df.to_pickle(folder3 + '/XY_all_stocks -st_' + str(st) + '-lt_' + str(lt) + '-type_' +
print(df.dropna()[['dt-EWMA_ATF', 'Y-EWMA_ATF', 'dt-TotalSignedSensiATF']].iloc[:5,:])
```

```
                                         dt-EWMA_ATF  Y-EWMA_ATF  dt-TotalSi
gnedSensiATF
                    Matu
2019-01-21 08:49:00 2019-06-21 15:30:00    -1.134502           0
-0.051059
2019-01-21 08:50:00 2019-06-21 15:30:00    -1.134842           0
-0.050843
2019-01-21 08:51:00 2019-06-21 15:30:00    -1.135180           0
-0.050440
2019-01-21 08:52:00 2019-06-21 15:30:00    -1.135516           0
-0.050224
2019-01-21 08:53:00 2019-06-21 15:30:00    -1.135850           0
-0.049805
```

The following graphical representation of the vectors obtained is not encouraging

In black are the points where Y is at +2std dev, in red those with Y at -2std.

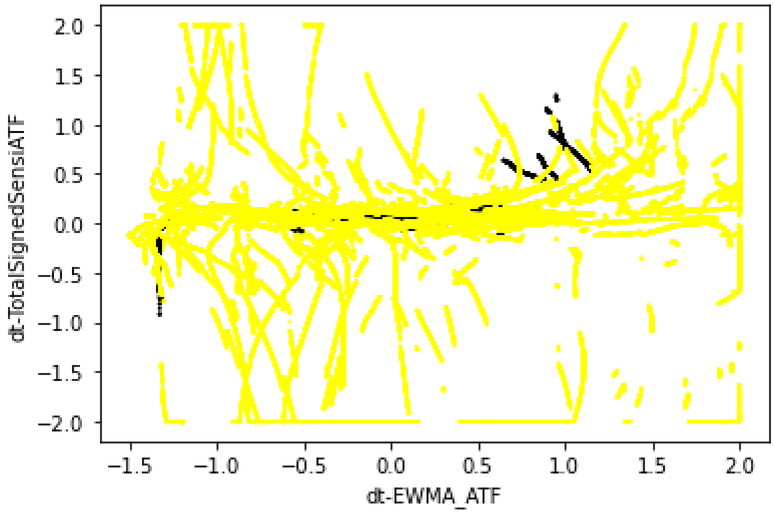
It shows that the only X vector which can be used to discriminate between cases when the parameter will move suddenly in the future (black points) and benign situations. This means that "the market will move suddenly in the near future if it has sharply moved recently".

The TotalSignedSensiATF parameter doesn't seem to discriminate much (Y axis)

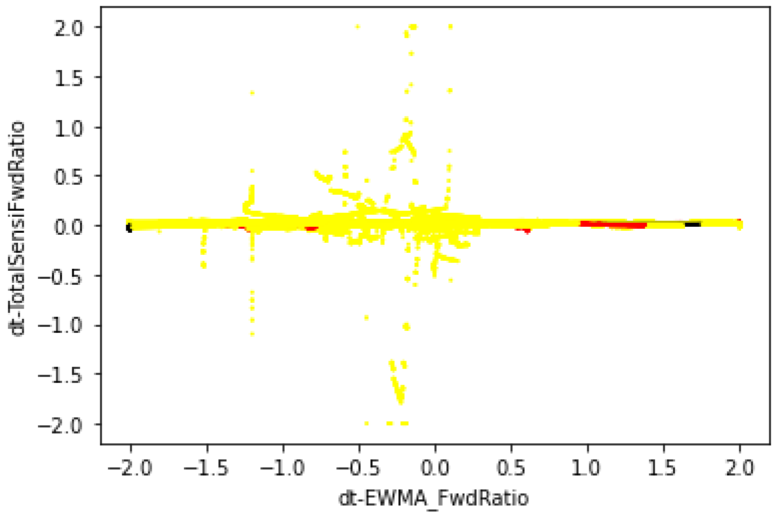
My interpretation is that our way of "signing" the trades is not satisfactory and would require the use of bid offer data which are not available for free.

Entrée [22]:

```
#Here is the result :

g = Graph('DAI')
g.graph_XY_scatter('dt-EWMA_ATF', 'dt-TotalSignedSensiATF', 'Y-EWMA_ATF')
g.graph_XY_scatter('dt-EWMA_FwdRatio', 'dt-TotalSensiFwdRatio', 'Y-EWMA_FwdRatio')

# 1st graph : This graph shows that the X chosen  ('dt-EWMA_ATF', 'dt-TotalSignedSensiA
# descriminating times just preceding a sudden shift in the vol, marked in red (down sh
# and based on the vector 'Y-EWMA_ATF'.

# 2nd graph : This graph shows that the X chosen  ('dt-EWMA_FwdRatio', 'dt-TotalSensiFw
# descriminating times just preceding a sudden shift in the fwd ratio, marked in red (d
# and based on the vector 'Y-EWMA_FwdRatio'.
```




```
average distance to center y=0 vs y=1 : dt-EWMA_ATF, dt-TotalSignedSensiATF, Y-EWMA_ATF
d1      0.811761
d2      0.049841
dall    0.975955
dtype: float64
d1      0.567779
d2      0.123641
dall    0.621337
dtype: float64
```




```
average distance to center y=0 vs y=1 : dt-EWMA_FwdRatio, dt-TotalSensiFwdRatio, Y-EWMA_FwdRatio
```

```
d1      0.725775
d2      0.013325
dall    0.732163
dtype: float64
d1      0.810980
d2      0.011882
dall    0.811382
dtype: float64
```

We will try to run Machine Learning algo in order to detect a possible asymmetric information situation

Cases of sudden move of a parameter (here we take 2 standard deviations as threshold) are rare and only a fraction of those events would have been preceded by abnormal trading patterns due to well informed agents trying to take advantage.
As a consequence, there is not enough data to feed a complex Neural Network.
If we were to use a complex NN, we would probably run into over-parameterization issues.
As a consequence, we will just run a small, 4 nodes NN.

Entrée [23]:

```
f = pd.read_pickle(folder3 + '/XY_all_stocks -st_' + str(st) + '-lt_' + str(lt) + '-tyr
Xcol = ['dt-EWMA_ATF', 'dt-TotalSensiATF', 'dt-NumberOfTrades']
df['RY'] = 1 - df['Y-EWMA_ATF']
Ycol = ['Y-EWMA_ATF', 'RY']
df = df.dropna(subset=Xcol + Ycol, how='any')
X = df[Xcol].values.astype(float)
y = df[Ycol].values.astype(float)
sep = int(X.shape[0]/2)
X_train = X[:sep]
y_train = y[:sep]
X_test = X[sep:]
y_test = y[sep:]

model = Sequential()
model.add(Dense(4, activation='relu', input_dim=3))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=1)

pred_train = model.predict(X_train)
scores = model.evaluate(X_train, y_train, verbose=0)
print('Accuracy on training data: {}% \n Error on training data: {}'.format(scores[1],

pred_test = model.predict(X_test)
scores2 = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy on test data: {}% \n Error on test data: {}'.format(scores2[1], 1 - sco



```

```
2507/2507 [==============================] - 1s 447us/step - loss: 0.2085 -
accuracy: 0.9345
Accuracy on training data: 0.9543284177780151%
 Error on training data: 0.04567158222198486
Accuracy on test data: 0.9606877565383911%
 Error on test data: 0.03931224346160889
```

```
**Conclusion**

With the X vectors ['dt-EWMA_ATF', 'dt-TotalSensiATF', 'dt-NumberOfTrades'] and the Y vector ['Y-EWMA_ATF'], representing the shift in the ATF parameter, we get a 96% accuracy result which is of course misleading as 95.75% of Y data are 0s.
In the end, the ML algorithm as designed here is unable to forecast sudden volatility shifts.
We would need to use orderbook data in addition to trades data for that.
```