How to time a Dispersion Strategy

Canari.dev (www.canari.dev), March-2021

```
'''
The definition of a _dispersion trade_ is provided here :
https://quantpedia.com/strategies/dispersion-trading/
'''
```

1
```
'''
This project showcases the possibilities offered by the API of Deutsche Boerse's A7 service. You will
For this, go to : https://www.mds.deutsche-boerse.com/mds-en/analytics/A7-analytics-platform

The following git shows how to connect to A7 API, then retrieve intraday quotes and calibrate a volat
https://github.com/canari-dev/Calibrating-implicit-volatility-surface-with-Deutsche-Boerse-A7

The first part of this git reuses large chunks of the code shown in the calibration git but the need
'''
```

16
```
'''
**First some setup**
Indicate here the folders where you want the intraday quotes to be stored (folder1)
and the calibration result (folder 2)
'''
folder1 = 'D:/Users/GitHub/Dispersion Volatility/processed'
folder2 = 'D:/Users/GitHub/Dispersion Volatility/parameters'

import os
os.makedirs(folder1, exist_ok=True)
os.makedirs(folder2, exist_ok=True)
```

17
```
# We are now importing public libraries
import numpy as np
import pandas as pd
import QuantLib as ql
import math
import datetime
import matplotlib.pyplot as plt
import requests
import warnings
from IPython.display import display, HTML

pd.set_option('display.width', 200)
pd.set_option('display.max_columns', 30)
```

18
```
# ...and specific libraries available in this git

from DateAndTime import DateAndTime
# uses QuantLib to calculate numbers of business day between dates and generate a list expiration dat

from PricingAndCalibration import Pricing
# uses Quantlib to price European and American options with continuous dividend yield and the associa

from PricingAndCalibration import FittingSpline
# uses scipy-UnivariateSpline to fit a 2nd degree spline through the implicit vol of bid and ask quot
```

19
```
# Choose here the time frame and the maturity to retrieve
from_date='20200310'
until_date='20200321'
chosen_matu = ['20201218']

#And now the underlyings with index weight
# [A7 option code, Spot whern weight in % was observed, Weight in %, Isin code, Underlying future cod
index_list = [('OESX', 3655.77, '', '', 'FESX')]

#you can reduce this stock list in ordre to save time...
udl_list = [
    ('SAP', 108.47, 6.17, 'DE0007164600', ''),
    ('ASM', 460, 5.69, 'NL0010273215', ''),
    ('LIN', 257.37, 4.69, 'IE00BZ12WP82', ''),
    ('MOH', 528.7, 4.58, 'FR0000121014', ''),
    ('SNW', 80.01, 4.13, 'FR0000120578', ''),
    ('SIE', 133.06, 3.42, 'DE0007236101', ''),
    ('TOTB', 35.07, 3.16, 'FR0000120271', ''),
    ('ALV', 195.77, 2.89, 'DE0008404005', ''),
    ('LOR', 305.6,2.89, 'FR0000120321', ''),
    ('AIR', 136.25, 2.74, 'FR0000120073', ''),
    ('IBE', 11.26, 2.57, 'ES0144580Y14', ''),
    ('SND', 124.6, 2.52, 'FR0000121972', ''),
    ('ENL5', 8.439, 2.45, 'IT0003128367', ''),
    ('BAY', 56.15, 2.21, 'DE000BAY0017', ''),
    ('ADS', 278, 2.2, 'DE000A1EWWW0', ''),
    ('BAS', 66.86, 2.02, 'DE000BASF111', ''),
    ('ADY', 1884.5, 1.86, 'NL0012969182', ''),
    ('PPX', 549, 1.81, 'FR0000121485', ''),
    ('SQU', 86, 1.71, 'FR0000125486', ''),
    ('ITK', 54.78, 1.64, 'BE0974293251', ''),
    ('DPW', 42.165, 1.63, 'DE0005552004', ''),
    ('DAI', 66.9, 1.61, 'DE0007100000', ''),
    ('PHI1', 47.08, 1.54, 'NL0000009538', ''),
    ('EAD', 93.42, 1.53, 'NL0000235190', ''),
    ('BSN', 53.34, 1.5, 'FR0000120644', ''),
    ('BNP', 53.595, 1.5, 'FR0000131104', ''),
#     ('PROSUS', 100.05, 1.44, 'NL0013654783', ''),
#     ('ESL', 128.75, 1.44, 'FR0000121667', 'no stock on Xetra'),
    ('AXA', 19.204, 1.37, 'FR0000120628', ''),
#      ('KONE', 66.03, 1.29, 'FI0009013403', ''),
    ('MUV2', 233.5, 1.58, 'DE0008430026', ''),
#     ('SEJ', 112.2, 1.26, 'FR0000073272', 'no stock on Xetra'),
    ('ANN', 55.53, 1.26, 'DE000A1ML7J1', ''),
    ('IES5', 2.059, 1.22, 'IT0000072618', ''),
    ('DB1', 136.4, 1.21, 'DE0005810055', ''),
    ('AHO', 23.35, 1.18, 'NL0011794037', ''),
    ('PER', 163.2, 1.15, 'FR0000120693', ''),
    ('IXD', 26.115, 1.12, 'ES0148396007', ''),
    ('BSD2', 2.774, 1.11, 'ES0113900J37', ''),
    ('VO3', 162.35, 1.07, 'DE0007664039', ''),
#     ('CRG', 35.51, 1.05, 'US12626K2033', 'no stock on Xetra'),
    ('INN', 7.766, 0.99, 'NL0011821202', ''),
#     ('AI3A', 56.37, 0.9, 'ES0109067019', 'no stock on Xetra'),
    ('VVU', 25.92, 0.87, 'FR0000127771', ''),
    ('BMW', 70.295, 0.83, 'DE0005190003', ''),
    ('NOA3', 3.495, 0.8, 'FI0009000681', ''),
    ('ENT5', 8.645, 0.7, 'IT0003132476', '')]
```

20
```
#indicate your A7 credentials :
owner = 'your A7 username here'

API_TOKEN = "Bearer " + "your A7 API token here"
# The API token is obtained by clicking on your name in the upper right corner of the A7 Analytics Pl

proxies = {
```

```
    "http": "",  # Enter http Proxy if needed",
    "https": ""  # Enter https Proxy if needed",
}
```

21
```
# Select an algo to retrieve quotes on A7.
# 'minsize_level_tb' allows you to look into the orderbook until finding a minimum number of lots.
# 'minsize_level_fast' goes faster by looking only at the top level
# Both algos are given in this git as a .yml file
# They must be loaded first in your A7 account.
algo = 'minsize_level_tb'

# If you have chosen the 'minsize_level' algo :
min_lots = 1
```

22
```
# Filter settings to speed up the process since we are only interested in At The Money volatility
# Levels are indicated for 1 year maturity option with an adjustment in sqrt(T)
moneyness_range_call = (-0.025, 0.15)
moneyness_range_put = (-0.15, 0.025)

# Create instances of DateAndTime both for italian and other underlyings
DT = DateAndTime(from_date, until_date, force_matu=chosen_matu)
DTi = DateAndTime(from_date, until_date, force_matu=chosen_matu, ital_rule=True)
```

23
```
<h6>The next function retrieves the intraday quotes given an option code</h6>
```

def get_quotes(opt): if opt['PutOrCall'] == 'S': market = 'XETR' else: market = 'XEUR'

```
  url = 'https://a7.deutsche-boerse.com/api/v1/algo/{}/{}/'.format(owner, algo)
  url = url + "run?marketId={}&date={}&marketSegmentId={}&securityId={}&from_h=9&from_m=1
      market, reference_date, opt['SegmentID'], opt['SecurityID'], min_lots)

  r = requests.get(url=url, headers={'Authorization': API_TOKEN}, proxies=proxies)
  res = r.json()

  if type(res) == list:
      df_opt = pd.DataFrame.from_dict(res[0]['series'][0]['content'])
      df_opt.ts = df_opt.ts.astype(np.int64)
      df_opt.ts = pd.to_datetime(df_opt.ts)
      df_opt.set_index('ts', inplace=True)

      df_opt[selected_fields_desc] = opt[selected_fields_desc]
      return (df_opt)
```

24
```
<h6>This function retrieves instruments from A7 in the res* lists, </h6>

It also gets segment codes in segment*,

and fills the  matu_list* with relevant maturities
```

def retrieve_instruments_from_A7():

```
    global res_u, res_f, res_i
    global segmentIDudl, segmentIDfut, segmentIDopt, security
    global matu_list_Stk, matu_list_Fut, matu_list_Opt

    # stock
    if (udl_p not in index_list) and (udl_p[4] != 'no stock on Xetra'):
        lst_ms = np.array([x['MarketSegment'] for x in res_gu['MarketSegments']])
        indx = np.where(lst_ms == isin)[0][0]
        segmentIDudl = res_gu['MarketSegments'][indx]['MarketSegmentID']

        url = 'https://a7.deutsche-boerse.com/api/v1/rdi/XETR/{}/{}?mode=detailed'.format(r
                                                                                          s
        r = requests.get(url=url, headers={'Authorization': API_TOKEN}, proxies=proxies)
        res_u = r.json()
        security = res_u['Securities'][0]

        matu_list_Stk = ['UDL']
    else:
        matu_list_Stk = []

    # Futures
    if (udl_p in index_list):
        udl_f = udl_p[4]

        lst_ms = np.array([x['MarketSegment'] for x in res_go['MarketSegments']])
        indx = np.where(lst_ms == udl_f)[0][0]
        segmentIDfut = res_go['MarketSegments'][indx]['MarketSegmentID']

        url = 'https://a7.deutsche-boerse.com/api/v1/rdi/XEUR/{}/{}?mode=detailed'.format(
            reference_date,
            segmentIDfut)
        r = requests.get(url=url, headers={'Authorization': API_TOKEN}, proxies=proxies)
        res_f = r.json()

        matu_list_Fut = DT_u.get_matu_list(reference_date, trim=True)[:2]
    else:
        matu_list_Fut = []

    # Options

    lst_ms = np.array([x['MarketSegment'] for x in res_go['MarketSegments']])
    indx = np.where(lst_ms == udl)[0][0]
    segmentIDopt = res_go['MarketSegments'][indx]['MarketSegmentID']

    url = 'https://a7.deutsche-boerse.com/api/v1/rdi/XEUR/{}/{}?mode=detailed'.format(refer
                                                                                      segme
    r = requests.get(url=url, headers={'Authorization': API_TOKEN}, proxies=proxies)
    res_i = r.json()

    matu_list_Opt = DT_u.get_matu_list(reference_date)
```

```
25  <h6>The next function transforms raw A7 output int dataframe of instrument to be retrieved.</h6>
    # It first tackles underlying (stock of futures) because it needs a underlying price to select "in_ra
```

```
20200303
20200310
20200317
                        ask     bid PutOrCall StrikePrice  ContractMultiplier ExerciseStyle matu  udl
ts
2020-03-03 10:10:00  177.55  177.50         S        None                   1          None  UDL  LIN
2020-03-03 12:10:00  177.80  177.70         S        None                   1          None  UDL  LIN
2020-03-03 14:10:00  177.85  177.80         S        None                   1          None  UDL  LIN
2020-03-03 16:10:00  177.75  177.65         S        None                   1          None  UDL  LIN
2020-03-03 16:25:00  176.20  176.10         S        None                   1          None  UDL  LIN

                        ask     bid PutOrCall StrikePrice  ContractMultiplier ExerciseStyle matu  udl
ts
2020-03-03 10:10:00  177.55  177.50         S        None                   1          None  UDL  LIN
2020-03-03 12:10:00  177.80  177.70         S        None                   1          None  UDL  LIN
2020-03-03 14:10:00  177.85  177.80         S        None                   1          None  UDL  LIN
2020-03-03 16:10:00  177.75  177.65         S        None                   1          None  UDL  LIN
2020-03-03 16:25:00  176.20  176.10         S        None                   1          None  UDL  LIN

                        ask     bid PutOrCall StrikePrice  ContractMultiplier ExerciseStyle matu  udl
ts
2020-03-03 10:10:00  177.55  177.50         S        None                   1          None  UDL  LIN
2020-03-03 12:10:00  177.80  177.70         S        None                   1          None  UDL  LIN
2020-03-03 14:10:00  177.85  177.80         S        None                   1          None  UDL  LIN
2020-03-03 16:10:00  177.75  177.65         S        None                   1          None  UDL  LIN
2020-03-03 16:25:00  176.20  176.10         S        None                   1          None  UDL  LIN
```

def build_options_list():

```
  global df_orderbook

  df_u = pd.DataFrame(columns=['SegmentID'] + selected_fields + selected_fields_desc)

  i = 0
  for matu in matu_list_Stk:
      df_u.loc[i] = [segmentIDudl, security['SecurityDesc'], security['SecurityID'], 'S',
      df_opt = get_quotes(df_u.loc[i])
      df_opt['matu'] = matu
      df_opt['udl'] = udl
      df_opt = df_opt.loc[(df_opt.bid > 0) & (df_opt.ask > 0)]
      df_orderbook = df_orderbook.append(df_opt)
      i += 1

  for c, matu in enumerate(matu_list_Fut):
      for x in [x for x in res_f['Securities'] if (str(x['MaturityDate']) == matu) and (>
          df_u.loc[i] = [segmentIDfut] + [x[elt] for elt in selected_fields] + [
              x['DerivativesDescriptorGroup']['SimpleInstrumentDescriptorGroup'][elt] for
          df_u.loc[i]['PutOrCall'] = 'FUT' + str(c)
          df_opt = get_quotes(df_u.loc[i])
          df_opt['matu'] = matu
          df_opt['udl'] = udl
```

```
            df_opt = df_opt.loc[(df_opt.bid > 0) & (df_opt.ask > 0)]
            df_orderbook = df_orderbook.append(df_opt)
            i += 1

    FVUmin = (df_opt.bid.min() + df_orderbook.ask.min()) / 2
    FVUmax = (df_opt.bid.max() + df_orderbook.ask.max()) / 2

    for matu in matu_list_Opt:

        i = 0
        df = pd.DataFrame(columns=['SegmentID'] + selected_fields + selected_fields_desc)

        for x in [x for x in res_i['Securities'] if
                  (str(x['MaturityDate']) == matu) and (x['SecurityType'] == 'OPT')]:
            df.loc[i] = [segmentIDopt] + [x[elt] for elt in selected_fields] + \
                        [x['DerivativesDescriptorGroup']['SimpleInstrumentDescriptorGroup']
                         in selected_fields_desc]
            i += 1

        df.sort_values(by=['StrikePrice', 'PutOrCall'], ascending=[True, True], inplace=Tru

        TTM = DT_u.time_between(pd.Timestamp(reference_date), pd.Timestamp(matu))
        df['matu'] = matu
        df['moneyness_T_min'] = df.apply(
            lambda opt: math.log(opt.StrikePrice / FVUmax) / (max(3.0 / 12.0, TTM) ** 0.5),
        # we consider that div max is 8%
        df['moneyness_T_max'] = df.apply(
            lambda opt: math.log(opt.StrikePrice / (FVUmin * 0.92)) / (max(3.0 / 12.0, TTM)

        df['in_range'] = df.apply(lambda opt: (opt.moneyness_T_max > moneyness_range_call[0
                opt.moneyness_T_min < moneyness_range_call[1]) \
            if opt.PutOrCall == '1' else \
            (opt.moneyness_T_max > moneyness_range_put[0]) and (
                    opt.moneyness_T_min < moneyness_range_put[1]),
                                  axis='columns')

    return (df.loc[df.in_range])
```

26 <h6>Let's now use these functions to retrieve the intraday quotes data and save them in the df_ordebo

```
OESX
20200304
   20201218
20200305
   20201218
20200306
   20201218
20200309
   20201218
20200311
   20201218
20200312
   20201218
20200313
   20201218
20200316
   20201218
```

20200318
   20201218
20200319
   20201218
20200320
   20201218
SAP
20200303
   20201218
20200304
   20201218
20200305
   20201218
20200306
   20201218
20200309
   20201218
20200311
   20201218
20200312
   20201218
20200313
   20201218
20200316
   20201218
20200318
   20201218
20200319
   20201218
20200320
   20201218
ASM
20200304
   20201218
20200305
   20201218
20200306
   20201218
20200309
   20201218
20200311
   20201218
20200312
   20201218
20200313
   20201218
20200316
   20201218
20200318
   20201218
20200319
   20201218
20200320
   20201218
LIN
20200304
   20201218
20200305
   20201218
20200306
   20201218
20200309
   20201218
20200311
   20201218
20200312
   20201218
20200313
   20201218

```
20200316
   20201218
20200318
   20201218
20200319
   20201218
20200320
   20201218
                                                                spline_bid
ts                  matu
2020-03-03 10:10:00 20201218  <scipy.interpolate.fitpack2.LSQUnivariateSplin...  <scipy.interpolate.fi
2020-03-03 16:25:00 20201218  <scipy.interpolate.fitpack2.LSQUnivariateSplin...  <scipy.interpolate.fi
2020-03-10 10:10:00 20201218  <scipy.interpolate.fitpack2.LSQUnivariateSplin...  <scipy.interpolate.fi
2020-03-10 12:10:00 20201218  <scipy.interpolate.fitpack2.LSQUnivariateSplin...  <scipy.interpolate.fi
2020-03-10 14:10:00 20201218  <scipy.interpolate.fitpack2.LSQUnivariateSplin...  <scipy.interpolate.fi
```

a = datetime.datetime.now() # time check

for reference_date in DT.dates_list: print(reference_date)

```
  # retrieve all instruments (stocks the options) from A7

  url = 'https://a7.deutsche-boerse.com/api/v1/rdi/XETR/{}?mode=detailed'.format(referenc
  r = requests.get(url=url, headers={'Authorization': API_TOKEN}, proxies=proxies)
  res_gu = r.json()

  url = 'https://a7.deutsche-boerse.com/api/v1/rdi/XEUR/{}?mode=detailed'.format(referenc
  r = requests.get(url=url, headers={'Authorization': API_TOKEN}, proxies=proxies)
  res_go = r.json()

  for udl_p in index_list + udl_list:
      udl = udl_p[0]
      isin = udl_p[3]

      # Determine which instance of DT class : the normal one (DT)
      # or the one giving thursday expiry dor italian stocks (DTi)
      if isin[:2] == 'IT':
          DT_u = DTi
      else:
          DT_u = DT

      try:
          df_orderbook = pd.read_pickle(folder1 + '/Quotes_' + udl + '.pkl')
      except:
          df_orderbook = pd.DataFrame()

      if df_orderbook.shape[0] > 0:
          done_already = [elt.strftime('%Y%m%d') for elt in set([elt.date() for elt in d
      else:
          done_already = []

      if reference_date not in done_already:

          try:

              retrieve_instruments_from_A7()
```

```
            # retrieves quotes

            selected_fields = ['SecurityDesc', 'SecurityID']
            selected_fields_desc = ['PutOrCall', 'StrikePrice', 'ContractMultiplier', '

            df = build_options_list()

            for index, opt in df.iterrows():
                df_opt = get_quotes(opt)
                df_opt['matu'] = opt.matu
                df_opt['udl'] = udl
                df_opt = df_opt.loc[(df_opt.bid > 0) & (df_opt.ask > 0)]
                df_orderbook = df_orderbook.append(df_opt)

        except:
            print('\n\n\n fail for : {}, {}\n\n\n'.format(reference_date, udl))

    df_orderbook.to_pickle(folder1 + '/Quotes_' + udl + '.pkl')
```

for udl_p in index_list + udl_list: print(display(HTML(df_orderbook.head().to_html()))) print('')

27
```
<h6>At this stage, you have created a df_orderbook dataframe and saved it in folder1.

We are now going to fit a vol curve on these quotes using the FittingSpline class.</h6>
```

	udl	ATFbid	ATFask	Spot	Error	RefSpot	W	isin	info	DispVolbid	Di
ts											
2020-03-03	OESX	16.503958	17.174238	3414.5	2.375209	3655.77	0.069222		FESX	10.444283	1
2020-03-10	OESX	24.646138	25.215311	3071.5	1.955628	3655.77	0.139757		FESX	5.640720	
2020-03-17	OESX	35.690101	37.699133	2404.5	2.134421	3655.77	0.155439		FESX	6.183251	1

	udl	ATFbid	ATFask	Spot	Error	RefSpot	W	isin	info	DispVolbid	Di
ts											
2020-03-10	OESX	24.646138	25.215311	3071.5	1.955628	3655.77	0.139757		FESX	5.640720	
2020-03-17	OESX	35.690101	37.699133	2404.5	2.134421	3655.77	0.155439		FESX	6.183251	1

for udl_p in index_list + udl_list: udl = udl_p[0] isin = udl_p[3] print(udl)

```
  if isin[:2] == 'IT':
      FS = FittingSpline(udl, DTi, folder1, folder2)
  else:
      FS = FittingSpline(udl, DT, folder1, folder2)

  if FS.data_found:
      FS.fit_all()
```

print(display(HTML(FS.df_params.head().to_html())))

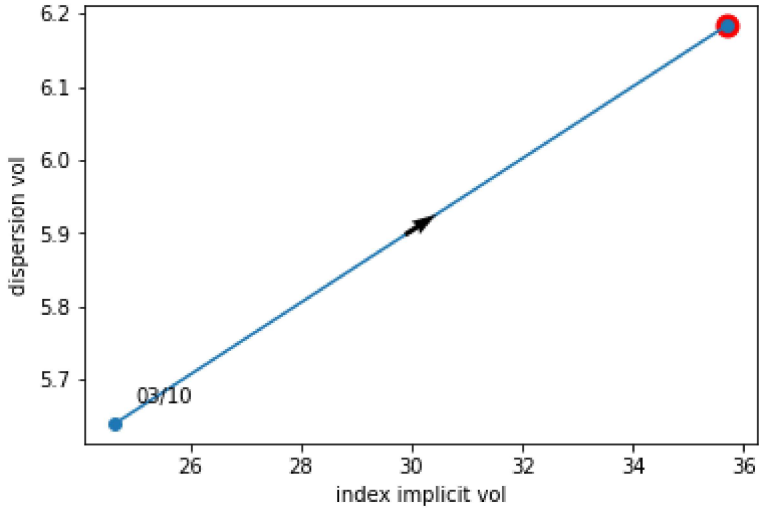
29
```
<h6>The partameters have been saved in folder2.</h6>

We are now going to compute the dispersion volatility defined as :

Dispersion Vol = ( Sum(i=1..50, Wi * ATFi^2) - (1+Leverage) * ATFindex^2 )  /  Normalisation_factor
```

where
- ATFi is the ATM implicit vol of the ith stock in the index
- ATFindex is the ATM implicit vol of the index
- Leverage is the additional notional of index varswap to sell in order to get a vega neutral dispers
- Wi is the weight of the ith component in the index
- Normalisation_factor is a factor to apply to get a vega of 1 on the index so : 2 * (1+Leverage) * A

The index weight are calculated based on a "photo" taken on a specific date. Ref spot have been saved




```
def pick_hour(df_d):
    i = df_d.Error.idxmin()
    return(df.loc[i,:])

def compute_dispvol(df_d):
    W = df_d.W.sum()
    res = df_d.loc[df_d.udl == 'OESX']
    res['W'] = W * res.RefSpot / res.Spot
    res['DispVolbid'] = (df_d.DispVolbid.sum() / W - (1+Leverage) * res.ATFbid ** 2) / (2 * (1+Levera
    res['DispVolask'] = (df_d.DispVolask.sum() / W - (1 + Leverage) * res.ATFask ** 2) / (2 * (1 + Le
    return(res)


matu = chosen_matu[0][:-2]
# udl_dic = dict([(elt[0], (elt[1], elt[2])) for elt in udl_list])
Leverage = 0.2

df = pd.DataFrame()
for udl_p in index_list + udl_list:
    udl = udl_p[0]
    try:
        df_udl = pd.read_pickle(folder2 + '/Params_' + udl + '.pkl')
    except:
        df_udl = pd.DataFrame()

    if df_udl.shape[0] > 0:
        df_udl['udl'] = udl
        df_udl['ATFbid'] = df_udl.spline_bid.apply(lambda x: x(0))
        df_udl['ATFask'] = df_udl.spline_ask.apply(lambda x: x(0))
        s = df_udl.index.levels[1].to_series()
        df_udl.index = df_udl.index.set_levels(s.map(lambda x: x[:-2]).fillna(s), level=1)
        df_udl = df_udl.xs(matu, level=1, drop_level=True)
        df_udl = df_udl[['udl', 'ATFbid', 'ATFask', 'Spot', 'Error']]
        df_udl.index = df_udl.index.map(lambda x: x.date())
        df_udl.sort_values(by=['ts', 'Error'], inplace=True)
        df_udl = df_udl.groupby(df_udl.index).first()
        df = df.append(df_udl)


dW = pd.DataFrame(index_list + udl_list, columns=['udl', 'RefSpot', 'W', 'isin', 'info'])
```

```
df = pd.merge(df, dW, left_on='udl', right_on='udl', how='left').set_index(df.index)
df['W'] = df.apply(lambda x: 0 if x.udl == 'OESX' else x.W / 100 * (x.Spot / x.RefSpot), axis='column
df['DispVolbid'] = df.apply(lambda x: x.W * x.ATFbid ** 2, axis='columns')
df['DispVolask'] = df.apply(lambda x: x.W * x.ATFask ** 2, axis='columns')
df = df.groupby(df.index, group_keys=False).apply(compute_dispvol)

print(df.head())



<h6>Finally, we will graph the DispVol vs the index vol to exhibit circular patterns as describes in



x = ((df['ATFbid'] + df['ATFask'])/2).values
y = ((df['DispVolbid'] + df['DispVolask'])/2).values
dates = [elt.strftime('%m/%d') for elt in df.index.values]

u = np.diff(x)
v = np.diff(y)
pos_x = x[:-1] + u/2
pos_y = y[:-1] + v/2
norm = np.sqrt(u**2+v**2)

fig, ax = plt.subplots()
ax.plot(x,y, marker="o")
ax.quiver(pos_x, pos_y, u/norm, v/norm, angles="xy", zorder=5, pivot="mid")

for i, txt in enumerate(dates):
    if i%3==0:
        # ax.annotate(txt, (x[i], y[i]))
        ax.annotate(txt, (x[i], y[i]), xytext=(10, 10), textcoords='offset points')

plt.scatter([x[-1]], [y[-1]], c='#ff0000', s=120)

plt.xlabel("index implicit vol")
plt.ylabel("dispersion vol")
plt.show()
```

This graph shows the evolution of the Dispersion Volatility (Y) vs the straight Implicit Volatility (X) of the index during the march 2020 market meltdown. It shows a classical example of index vol raising faster than it's components then a catch up. (See the README file of this Git for more details.)