options

July 6, 2023

[ ]:
```
# Tick History PCAP, SQL request API :
import maystreet_data

# Refinitiv Data Platform API :
import refinitiv.data as rd

import pandas as pd
import numpy as np
import datetime as dt
from scipy.stats import norm
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import cm
```

[ ]:
```
### Date used for Historical Data Simulation
date = "2023-06-28"
date2 = "2023-06-29"

### Option used for the Simulation
option_name = 'BMW SI 20231215 PS AM C 100.00 0'
```

[ ]:
```
### Using PCAP Normalized Data to download one option price with timestamp
def query_apu():
    return f"""
    SELECT
        dt, product, f,  firstexchangetimestamp, bidprice_1, askprice_1
    FROM
        "prod_lake"."p_mst_data_lake"."mt_aggregated_price_update"
    WHERE
        f='emdi_eurex_t7'
        AND dt= '{date}'
        AND product = '{option_name}'
    ORDER BY
        firstexchangetimestamp
"""
records_iter = maystreet_data.query(maystreet_data.DataSource.DATA_LAKE,␣
 ↪query_apu())
```

```
df_option = pd.DataFrame(records_iter)
df_option['firstexchangetimestamp'] = pd.to_datetime(df_option.
 ↪firstexchangetimestamp)
df_option = df_option.rename(columns = {
                    'product' : 'opt',
                    'bidprice_1' : 'opt_bid',
                    'askprice_1' : 'opt_ask',
                    'firstexchangetimestamp' : 'opt_timestamp'
                    })
df_option = df_option.drop(['f', 'dt'], axis = 1)
df_option['opt_avg'] = (df_option['opt_bid'] + df_option['opt_ask']) / 2
df_option.head()
```

[ ]:

	opt	opt_timestamp	opt_bid	opt_ask	opt_avg
0	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:02:28.782369494	12.95	13.35	13.150
1	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:03:02.354608566	13.05	13.35	13.200
2	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:06:11.073035513	13.40	13.70	13.550
3	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:06:57.446147325	13.30	13.65	13.475
4	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:06:58.928088158	13.30	13.65	13.475

[ ]:
```
rd.open_session(config_name='../pwd/session.json')
```

[ ]:
```
<refinitiv.data.session.Definition object at 0x7fa93a65d3d0 {name='rdp'}>
```

[ ]:
```
df_BMW =  rd.get_history(
    universe=['BMWG.DE'],
    fields=["TRDPRC_1"],
    interval="tick",
    start=date,
    end=date2)
df_BMW = df_BMW.reset_index()
df_BMW = df_BMW.rename(columns = {
                    'TRDPRC_1' : 'underlying_trade_price',
                    'Timestamp' : 'underlying_timestamp'
                    })
df_BMW.head()
```

[ ]:

BMWG.DE	underlying_timestamp	underlying_trade_price
0	2023-06-28 06:50:00.413	108.3
1	2023-06-28 06:50:00.555	108.3

```
2       2023-06-28 06:50:01.250                  108.3
3       2023-06-28 06:50:01.308                 108.06
4       2023-06-28 06:50:05.042                 108.06
```

[ ]:
```
### Merge Option price df, and underlying price df
df = pd.merge_asof(df_option, df_BMW, left_on= "opt_timestamp",␣
 ↪right_on="underlying_timestamp")
df.head()
```

[ ]:
```
                                opt                opt_timestamp  opt_bid  \
0  BMW SI 20231215 PS AM C 100.00 0 2023-06-28 07:02:28.782369494    12.95
1  BMW SI 20231215 PS AM C 100.00 0 2023-06-28 07:03:02.354608566    13.05
2  BMW SI 20231215 PS AM C 100.00 0 2023-06-28 07:06:11.073035513    13.40
3  BMW SI 20231215 PS AM C 100.00 0 2023-06-28 07:06:57.446147325    13.30
4  BMW SI 20231215 PS AM C 100.00 0 2023-06-28 07:06:58.928088158    13.30

   opt_ask  opt_avg    underlying_timestamp  underlying_trade_price
0    13.35   13.150 2023-06-28 07:02:28.569                   108.2
1    13.35   13.200 2023-06-28 07:03:02.191                  108.32
2    13.70   13.550 2023-06-28 07:06:08.140                  108.74
3    13.65   13.475 2023-06-28 07:06:51.130                  108.64
4    13.65   13.475 2023-06-28 07:06:51.130                  108.64
```

0.0.1 Call OPtion Price - Black Schole Model

$$C = N(d_1)S_t - N(d_2)Ke^{-rt}$$

$$\text{where } d_1 = \frac{\ln \frac{S_t}{K} + (r + \frac{\sigma^2}{2})t}{\sigma\sqrt{t}}$$

$$\text{and } d_2 = d_1 - \sigma\sqrt{t}$$

C = call option price N = CDF of the normal distribution St= spot price of an asset K = strike price r = risk-free interest rate t = time to maturity = volatility of the asset

[ ]:
```
### Fixing some parameters
r = 0.045
N = norm.cdf
N_prime = norm.pdf
sigma = np.sqrt(252) * 0.015304757885578736
K = 100
exp_date = dt.datetime(2023, 12, 15, 0, 0) #'2023-12-15'
```

```
today_date = dt.datetime(2023, 6, 28, 0, 0)
T = ((exp_date - today_date).days) / 365
```

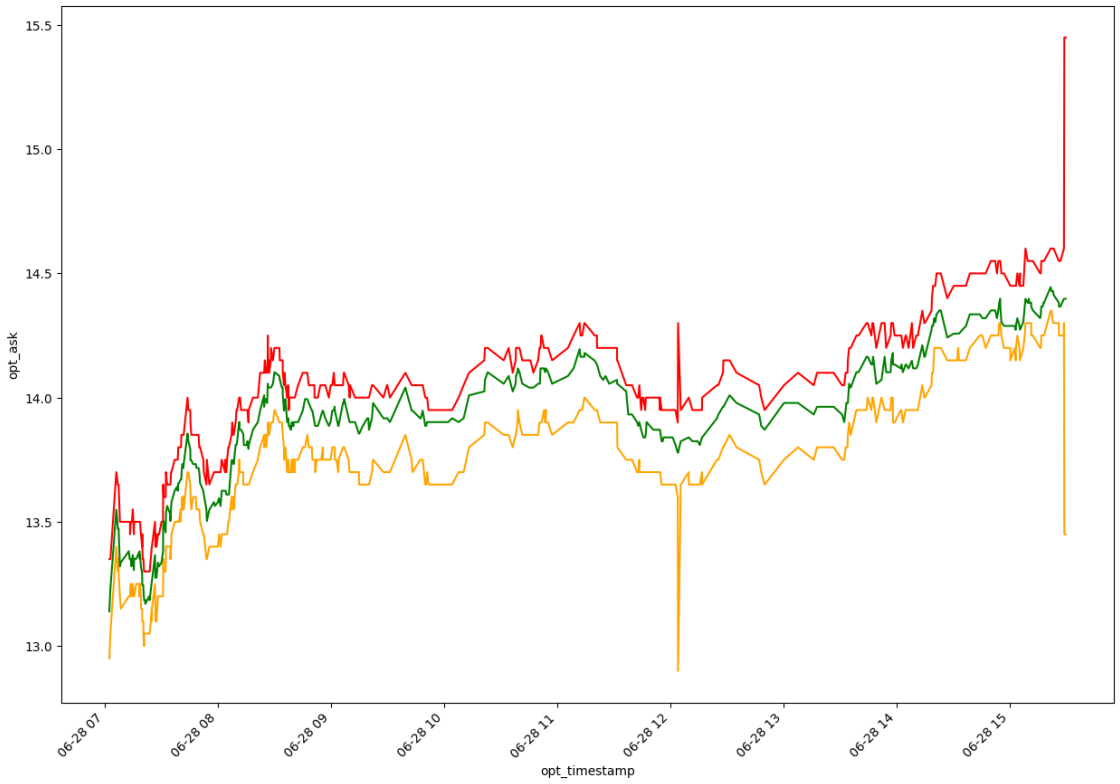
[ ]:
```
def black_scholes_call(S, K, T, r, sigma):
    '''
    :param S: Asset price
    :param K: Strike price
    :param T: Time to maturity
    :param r: risk-free rate (treasury bills)
    :param sigma: volatility
    :return: call price
    '''
    d1 = (np.log(S / K) + (r + sigma ** 2 / 2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    call = S * N(d1) -  N(d2)* K * np.exp(-r * T)
    return call

df['opt_estimated_price'] = df.underlying_trade_price.apply(black_scholes_call,␣
 ↪args = (K, T, r, sigma))
df.head()
```

[ ]:

	opt	opt_timestamp	opt_bid	opt_ask	opt_avg	underlying_timestamp	underlying_trade_price	opt_estimated_price
0	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:02:28.782369494	12.95	13.35	13.150	2023-06-28 07:02:28.569	108.2	13.139522
1	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:03:02.354608566	13.05	13.35	13.200	2023-06-28 07:03:02.191	108.32	13.230034
2	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:06:11.073035513	13.40	13.70	13.550	2023-06-28 07:06:08.140	108.74	13.548798
3	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:06:57.446147325	13.30	13.65	13.475	2023-06-28 07:06:51.130	108.64	13.472625
4	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:06:58.928088158	13.30	13.65	13.475	2023-06-28 07:06:51.130	108.64	13.472625

[ ]:
```
fix, ax = plt.subplots(figsize=(15,10))
sns.lineplot(df, x = 'opt_timestamp', y = 'opt_ask', color = 'red')
sns.lineplot(df, x = 'opt_timestamp', y = 'opt_bid', color = 'orange')
sns.lineplot(df, x = 'opt_timestamp', y = 'opt_estimated_price', color =␣
 ↪'green')
```

```
plt.xticks(rotation=45, ha='right')
plt.show()
```



0.1 Implied Volatility Calculation using Newton-Raphson Method

[ ]:
```
def vega(S, K, T, r, sigma):
    '''
    :param S: Asset price
    :param K: Strike price
    :param T: Time to Maturity
    :param r: risk-free rate (treasury bills)
    :param sigma: volatility
    :return: partial derivative w.r.t volatility
    '''
    d1 = (np.log(S / K) + (r + sigma ** 2 / 2) * T) / sigma * np.sqrt(T)
    vega = S * N_prime(d1) * np.sqrt(T)
    return vega


def implied_volatility_call(C, S, K, T, r, sigma, tol=0.0001,␣
 ↪max_iterations=100):
    '''
```

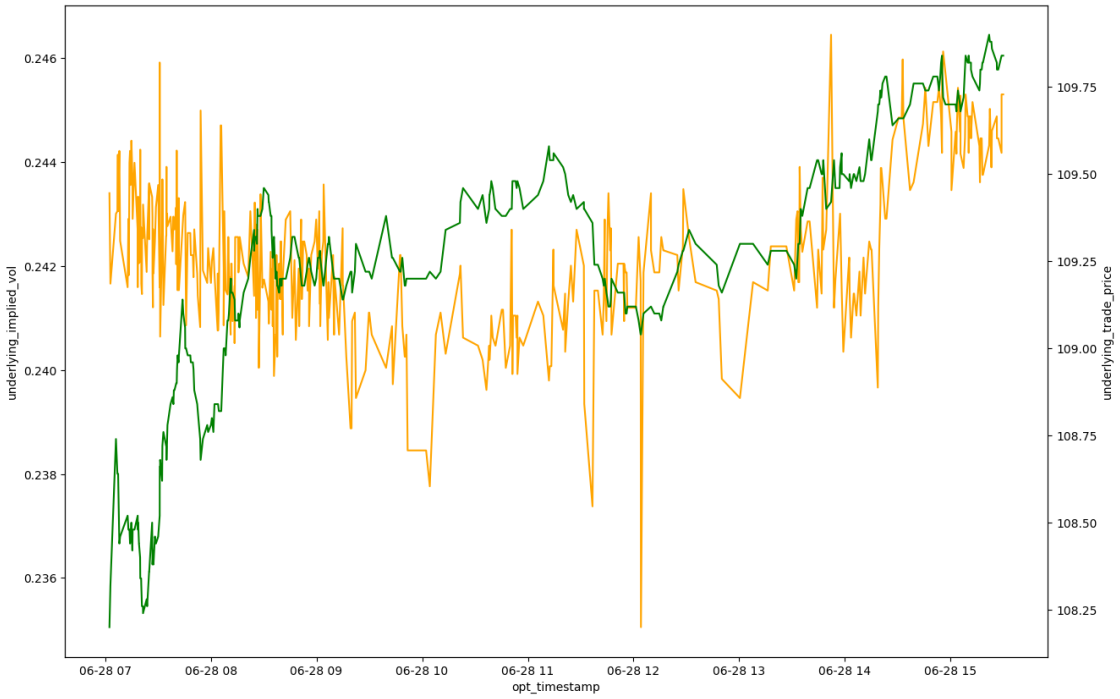
```
    :param C: Observed call price
    :param S: Asset price
    :param K: Strike Price
    :param T: Time to Maturity
    :param r: riskfree rate
    :param tol: error tolerance in result
    :param max_iterations: max iterations to update vol
    :return: implied volatility in percent
    '''
    for i in range(max_iterations):
        diff = black_scholes_call(S, K, T, r, sigma) - C
        if abs(diff) < tol:
            #print(f'On {i}th iteration, difference is equal to {diff}')
            break
        sigma = sigma - diff / vega(S, K, T, r, sigma)
    return sigma

df['underlying_implied_vol'] = df.apply(lambda x: implied_volatility_call(x.
 ↪opt_avg, x.underlying_trade_price,K, T, r, sigma), axis=1)
df.head()
```

[ ]:

	opt	opt_timestamp	opt_bid	opt_ask	opt_avg	underlying_timestamp	underlying_trade_price	opt_estimated_price	underlying_implied_vol
0	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:02:28.782369494	12.95	13.35	13.150	2023-06-28 07:02:28.569	108.2	13.139522	0.243403
1	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:03:02.354608566	13.05	13.35	13.200	2023-06-28 07:03:02.191	108.32	13.230034	0.241662
2	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:06:11.073035513	13.40	13.70	13.550	2023-06-28 07:06:08.140	108.74	13.548798	0.243006
3	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:06:57.446147325	13.30	13.65	13.475	2023-06-28 07:06:51.130	108.64	13.472625	0.243055
4	BMW SI 20231215 PS AM C 100.00 0	2023-06-28 07:06:58.928088158	13.30	13.65	13.475	2023-06-28 07:06:51.130	108.64	13.472625	0.243055

[ ]:
```
# df['lr']= np.log(df.underlying_trade_price.pct_change()+1)
fix, ax1 = plt.subplots(figsize=(15,10))
sns.lineplot(df, x = 'opt_timestamp', y = 'underlying_implied_vol', color =␣
 ↪'orange', ax = ax1)
ax2 = ax1.twinx()
```

```
sns.lineplot(df, x = 'opt_timestamp', y = 'underlying_trade_price', color =␣
 ↪'green', ax = ax2)
plt.show()
```



0.2 Volatility Surface

[ ]:
```
from mpl_toolkits import mplot3d
from datetime import datetime
from itertools import chain
from tqdm import tqdm
```

[ ]:
```
### Using PCAP Normalized Data to download one option price with timestamp
def query_apu():
    return f"""
    SELECT
        dt, product, f,  firstexchangetimestamp, bidprice_1, askprice_1
    FROM
        "prod_lake"."p_mst_data_lake"."mt_aggregated_price_update"
    WHERE
        f='emdi_eurex_t7'
        AND dt= '{date}'
        AND product LIKE '%BMW%'
    ORDER BY
        firstexchangetimestamp
```

```
"""
records_iter = maystreet_data.query(maystreet_data.DataSource.DATA_LAKE,␣
 ↪query_apu())
df_options = pd.DataFrame(records_iter)
df_options['firstexchangetimestamp'] = pd.to_datetime(df_options.
 ↪firstexchangetimestamp)
df_options = df_options.rename(columns = {
                    'product' : 'opt',
                    'bidprice_1' : 'opt_bid',
                    'askprice_1' : 'opt_ask',
                    'firstexchangetimestamp' : 'opt_timestamp'
                    })
df_options = df_options.drop(['f', 'dt'], axis = 1)
df_options['opt_avg'] = (df_options['opt_bid'] + df_options['opt_ask']) / 2

df_options.loc[df_options['opt_avg'].isna(), 'opt_avg'] = df_options['opt_ask']
df_options.loc[df_options['opt_avg'].isna(), 'opt_avg'] = df_options['opt_bid']
df_options[['p', 'si', 'opt_exp', 'ps', 'am', 'opt_side', 'opt_strike', 'O']]=␣
 ↪df_options.opt.str.split(' ', expand = True)
df_options['opt_exp'] = pd.to_datetime(df_options['opt_exp'])
df_options = df_options.drop(['p', 'si', 'ps', 'am', 'O'], axis = 1)
df_options['opt_strike'] = df_options['opt_strike'].astype('float')
df_options.head()
```

[ ]:

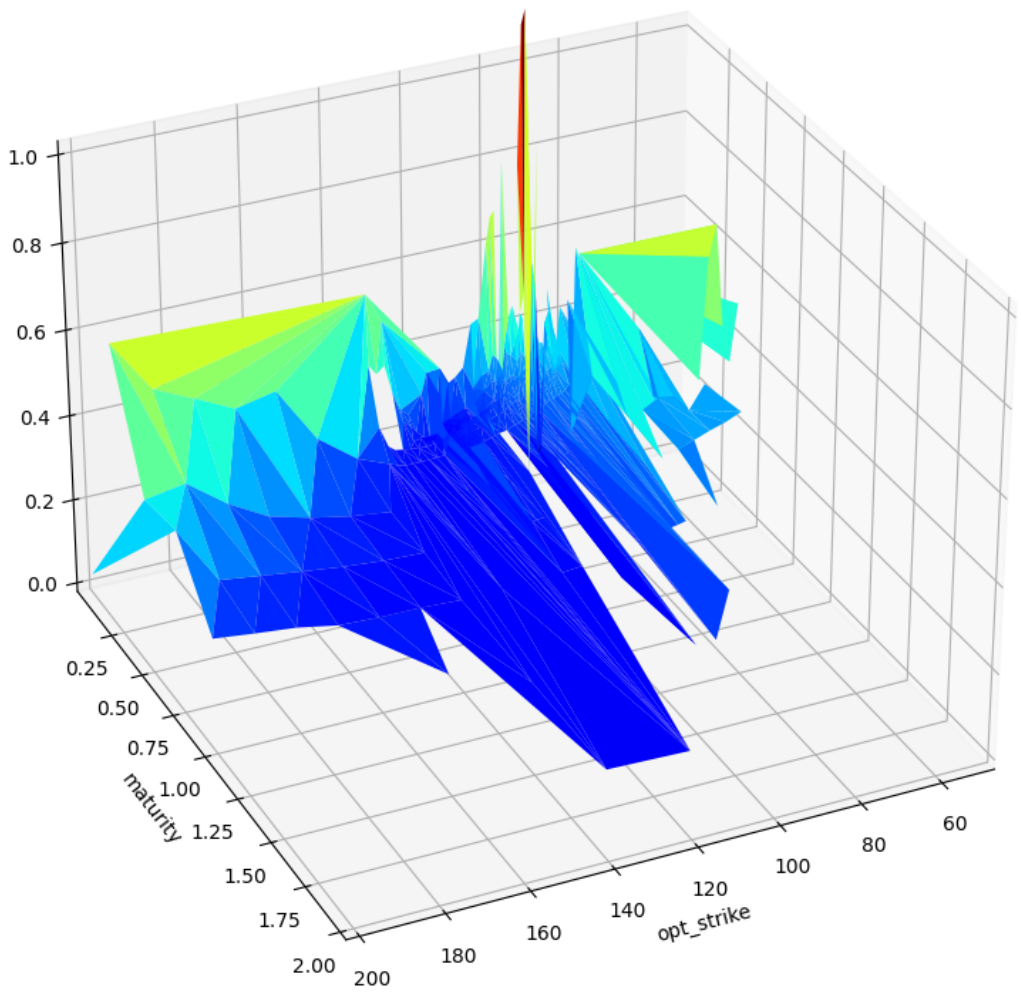
	opt	opt_timestamp	opt_bid	\
0	BMW SI 20230915 PS AM C 116.00 0	2023-06-28 07:00:30.093037663	NaN	
1	BMWE SI 20230818 PS EU P 104.00 0	2023-06-28 07:00:30.146045768	1.50	
2	BMW SI 20240621 PS AM C 160.00 0	2023-06-28 07:01:34.272008018	0.25	
3	BMW SI 20231215 PS AM P 72.00 0	2023-06-28 07:01:34.272008018	NaN	
4	BMW SI 20231215 PS AM P 98.00 0	2023-06-28 07:01:34.272619097	NaN	

	opt_ask	opt_avg	opt_exp	opt_side	opt_strike
0	3.00	3.00	2023-09-15	C	116.0
1	NaN	1.50	2023-08-18	P	104.0
2	NaN	0.25	2024-06-21	C	160.0
3	0.65	0.65	2023-12-15	P	72.0
4	2.93	2.93	2023-12-15	P	98.0

[ ]:
```
### Keep CAll options only, calculate Maturity
df_options = df_options[df_options['opt_side'] == 'C']
df_options['maturity'] = ((df_options['opt_exp'] -  today_date)/ np.
 ↪timedelta64(1, 'D')) / 365

### Merge Option price df, and underlying price df
df_vol= pd.merge_asof(df_options, df_BMW, left_on= "opt_timestamp",␣
 ↪right_on="underlying_timestamp")
```

[ ]:
```
### Calculating underlying_implied_vol
tqdm.pandas()
# def implied_volatility_call(C, S, K, T, r, sigma, tol=0.0001,␣
 ↪max_iterations=100):
#     '''
#     :param C: Observed call price
#     :param S: Asset price
#     :param K: Strike Price
#     :param T: Time to Maturity
#     :param r: riskfree rate
#     :param tol: error tolerance in result
#     :param max_iterations: max iterations to update vol
#     :return: implied volatility in percent
#     '''
df_vol['underlying_implied_vol'] = df_vol.progress_apply(lambda x:␣
 ↪implied_volatility_call(x.opt_avg, x.underlying_trade_price,  x.opt_strike,␣
 ↪x.maturity, r, sigma), axis=1)
df_vol.head()
df_vol.to_csv('df_vol.csv', index = False)
# df_vol = pd.read_csv('df_vol.csv')
```

[ ]:
```
# initiate figure
fig = plt.figure(figsize=(10,10))
# set projection to 3d
axs = plt.axes(projection="3d")
# use plot_trisurf from mplot3d to plot surface and cm for color scheme
axs.plot_trisurf(df_vol.opt_strike, df_vol.maturity, df_vol.
 ↪underlying_implied_vol, cmap=cm.jet)
# change angle
axs.view_init(30, 65)
axs.set_xlim(50, 200)
axs.set_ylim(0.0054, 2)
axs.set_zlim(0, 1.01)
# add labels
plt.xlabel("opt_strike")
plt.ylabel("maturity")
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
maturity
200
180
160
140
120
100
80
60
opt_strike