Homework 1 - FE 621 - Waldyr Faustini-Final

March 1, 2021

1 Homework 1 - FE 621 - Waldyr Faustini

2 Part 1

2.0.1 (Part 1.1 and 1.2) Getting Data from Yahoo Finance

I will use the yfinance API, not an official Yahoo API but i think was the best solution in this case

[2]:
```
# pip install yfinance --upgrade --no-cache-dir
```

[229]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import datetime as dt
from tqdm.notebook import tqdm

pd.options.mode.chained_assignment = None  # default='warn'

tqdm.pandas() # for progress bars
```

2.0.2 Bonus

Create a program that is capable of downloading multiple assets, combine them with the associated time column, and save the data into a csv or excel file.

Since we intend on only downloading data after market close, it is sufficient for us to download equity data only for close values. However, in the last part of this exercise we also explicitly get intraday data for the specified tickers, for completeness.

First we show how the code works for APPL

[73]:
```
def download_current_options(ticker: yf.Ticker, savepath=None):

    download_datetime = pd.to_datetime('now')
    download_date = download_datetime.date().strftime('%Y-%m-%d')

    option_list = []
    for maturity in ticker.options:
```

```
        # only download for the upcoming weeks, not including this one
        # this will bug for dates later in the year, but work now
        if (pd.to_datetime(maturity).week <= download_datetime.week):
            pass
        else:
            calls = ticker.option_chain(maturity).calls
            puts = ticker.option_chain(maturity).puts

            calls['type'] = 'call'
            puts['type'] = 'put'
            df = pd.concat([calls, puts])

            df['maturity'] = maturity
            option_list.append(df)

    options = pd.concat(option_list)
    options['download_date'] = download_date
    options['ticker_underlying'] = ticker.ticker

    if savepath:
        options.to_csv(savepath)
        print(f"{ticker.ticker} options:\tsaved to {savepath}")

    return options
```

[72]:
```
def download_equity_data(ticker: yf.Ticker, savepath=None):

    download_datetime = pd.to_datetime('now')
    download_date = download_datetime.date().strftime('%Y-%m-%d')

    hist = ticker.history(period='1d').tail(1)
    hist['ticker'] = ticker.ticker
    hist['download_date'] = download_date
    if savepath:
        hist.to_csv(savepath)
        print(f"{ticker.ticker} market data:\tsaved to {savepath}")

    return hist
```

[62]:
```
def download_ticker_full_data(ticker_str):
    ticker = yf.Ticker(ticker_str)
    download_datetime = pd.to_datetime('now')
    filename = lambda x: f"{ticker.ticker}__{x}_{download_datetime.
↪strftime('%Y-%m-%d')}.csv"
    option_file = filename('options')
    equity_file = filename('equity')
```

```
    hist, opts = download_equity_data(ticker, savepath=equity_file),␣
↪download_current_options(ticker, savepath=option_file)
    return hist, opts
```

[63]:
```
%%time
hist, opts = download_ticker_full_data('AAPL')
```

```
AAPL market data: saved to AAPL__equity_2021-02-22.csv
AAPL options: saved to AAPL__options_2021-02-22.csv
Wall time: 23.5 s
```

Visualize AAPL data and options:

[64]:
```
hist.head(3) # there is by construction just one value - the last recorded close
```

[64]:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits	ticker	download_date
2021-02-19	130.240005	130.710007	128.800003	129.869995	87668834	0	0	AAPL	2021-02-22

[65]:
```
opts.head(3)
```

[65]:

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest	impliedVolatility	inTheMoney	contractSize	currency	type	maturity	download_date	ticker_underlying
0	AAPL210305C00070000	2021-02-19 20:55:11	70.0	60.00	0.0	0.0	0.0	0.0	3.0	4.0	0.00001	True	REGULAR	USD	call	2021-03-05	2021-02-22	AAPL
1	AAPL210305C00075000	2021-02-11 15:12:01	75.0	60.25	0.0	0.0	0.0	0.0	3.0	3.0	0.00001	True	REGULAR	USD	call	2021-03-05	2021-02-22	AAPL
2	AAPL210305C00080000	2021-02-08 20:24:00	80.0	56.40	0.0	0.0	0.0	0.0	2.0	1.0	0.00001	True	REGULAR	USD	call	2021-03-05	2021-02-22	AAPL

We now proceed to formalize this into a single script for a list of tickers

[81]:
```
def download_list_ticker_data(tickers: list):
    assert tickers, "Ticker list is empty"

    hist_list, opts_list = [], []
    for ticker in tickers:
        hist, opts = download_ticker_full_data(ticker)
```

```
        hist_list.append(hist)
        opts_list.append(opts)

    hist_df = pd.concat(hist_list)
    opts_df = pd.concat(opts_list)

    # pre-process
    len_opts = len(opts_df)
    opts_df = opts_df.dropna()
    opts_df = opts_df.drop_duplicates()
    print(f"After dropping NAs and duplicates, options data went from {len_opts}␣
↪to {len(opts)} entries")

    filename = lambda x: f"{' '.join(tickers)}__{x}_{pd.to_datetime('now').
↪strftime('%Y-%m-%d')}.csv"
    hist_df.to_csv(filename('equity'))
    opts_df.to_csv(filename('options'))

    return hist_df, opts_df
```

[83]:
```
%%time
hist, opts = download_list_ticker_data(['SPY', "^VIX", "AAPL"])
```

```
SPY market data:        saved to SPY__equity_2021-02-22.csv
SPY options:    saved to SPY__options_2021-02-22.csv
^VIX market data:       saved to ^VIX__equity_2021-02-22.csv
^VIX options:   saved to ^VIX__options_2021-02-22.csv
AAPL market data:       saved to AAPL__equity_2021-02-22.csv
AAPL options:   saved to AAPL__options_2021-02-22.csv
After dropping NAs and duplicates, options data went from 9329 to 2083 entries
Wall time: 1min 18s
```

2.0.3 Bonus: intraday equity data with a single time index

To pick data only for days Feb 23 and 24, we use the Yahoo finance API and clean data afterwards

[112]:
```
bonus_data = yf.download("AMZN SPY ^VIX", start="2021-02-23",␣
↪end="2021-02-26",interval="1m", group_by='tickers')
```

```
[**********100%**********]  3 of 3 completed
```

[113]:
```
# convert to single-row column names
bonus_data.columns = [' '.join(col).strip() for col in bonus_data.columns.values]
```

[114]:
```
bonus_data.head(3)
```

[114]:
```
                           SPY Open    SPY High     SPY Low   SPY Close  \
Datetime
2021-02-22 10:00:00-05:00  388.019989  388.269989  387.980011  388.250000
2021-02-22 10:01:00-05:00  388.260010  388.359985  387.970001  388.000000
2021-02-22 10:02:00-05:00  388.010101  388.239990  387.950012  388.218994

                           SPY Adj Close  SPY Volume  ^VIX Open  ^VIX High  \
Datetime
2021-02-22 10:00:00-05:00     388.250000           0  23.459999  23.459999
2021-02-22 10:01:00-05:00     388.000000      136022  23.250000  23.400000
2021-02-22 10:02:00-05:00     388.218994      131363  23.410000  23.440001

                            ^VIX Low  ^VIX Close  ^VIX Adj Close  ^VIX Volume  \
Datetime
2021-02-22 10:00:00-05:00  23.389999   23.389999       23.389999          0.0
2021-02-22 10:01:00-05:00  23.250000   23.400000       23.400000          0.0
2021-02-22 10:02:00-05:00  23.410000   23.430000       23.430000          0.0

                             AMZN Open    AMZN High     AMZN Low   AMZN Close  \
Datetime
2021-02-22 10:00:00-05:00  3227.840088  3230.629883  3224.520020  3230.629883
2021-02-22 10:01:00-05:00  3232.319824  3232.319824  3221.669922  3221.669922
2021-02-22 10:02:00-05:00  3222.159912  3225.489990  3220.010010  3225.489990

                           AMZN Adj Close  AMZN Volume
Datetime
2021-02-22 10:00:00-05:00     3230.629883          0.0
2021-02-22 10:01:00-05:00     3221.669922       8290.0
2021-02-22 10:02:00-05:00     3225.489990      10906.0
```

[115]:
```
bonus_data['time'] = pd.to_datetime(bonus_data.index)
```

[116]:
```
bonus_data = bonus_data[bonus_data['time'].dt.day.isin([23,24])]
```

[119]:
```
bonus_data = bonus_data.dropna()
```

[120]:
```
bonus_data.head()
```

[120]:
```
                           SPY Open    SPY High     SPY Low   SPY Close  \
Datetime
2021-02-23 09:31:00-05:00  384.250000  384.279999  384.250000  384.250000
2021-02-23 09:32:00-05:00  384.250000  384.500000  384.079987  384.109894
2021-02-23 09:33:00-05:00  384.119995  384.230011  383.959991  384.010010
2021-02-23 09:35:00-05:00  383.660004  383.760010  383.480011  383.489990
2021-02-23 09:36:00-05:00  383.489990  383.739990  383.429993  383.670013

                           SPY Adj Close  SPY Volume  ^VIX Open  ^VIX High  \
```

```
Datetime
2021-02-23 09:31:00-05:00     384.250000      517867  25.120001  25.120001
2021-02-23 09:32:00-05:00     384.109894      488356  25.080000  25.100000
2021-02-23 09:33:00-05:00     384.010010      615814  25.110001  25.150000
2021-02-23 09:35:00-05:00     383.489990      487823  25.450001  25.450001
2021-02-23 09:36:00-05:00     383.670013      519078  25.450001  25.450001

                           ^VIX Low  ^VIX Close  ^VIX Adj Close  ^VIX Volume  \
Datetime
2021-02-23 09:31:00-05:00  24.959999   24.959999       24.959999          0.0
2021-02-23 09:32:00-05:00  24.940001   25.059999       25.059999          0.0
2021-02-23 09:33:00-05:00  25.030001   25.150000       25.150000          0.0
2021-02-23 09:35:00-05:00  25.320000   25.320000       25.320000          0.0
2021-02-23 09:36:00-05:00  25.450001   25.450001       25.450001          0.0

                             AMZN Open    AMZN High     AMZN Low   AMZN Close  \
Datetime
2021-02-23 09:31:00-05:00  3128.360107  3137.520020  3128.360107  3130.000000
2021-02-23 09:32:00-05:00  3135.000000  3139.949951  3132.080078  3132.080078
2021-02-23 09:33:00-05:00  3132.280029  3134.850098  3131.110107  3133.709961
2021-02-23 09:35:00-05:00  3110.989990  3110.989990  3109.469238  3109.469238
2021-02-23 09:36:00-05:00  3111.059814  3118.000000  3111.059814  3111.889893

                           AMZN Adj Close  AMZN Volume  \
Datetime
2021-02-23 09:31:00-05:00     3130.000000      17692.0
2021-02-23 09:32:00-05:00     3132.080078      18519.0
2021-02-23 09:33:00-05:00     3133.709961       6476.0
2021-02-23 09:35:00-05:00     3109.469238      86696.0
2021-02-23 09:36:00-05:00     3111.889893      20490.0

                                                 time
Datetime
2021-02-23 09:31:00-05:00 2021-02-23 09:31:00-05:00
2021-02-23 09:32:00-05:00 2021-02-23 09:32:00-05:00
2021-02-23 09:33:00-05:00 2021-02-23 09:33:00-05:00
2021-02-23 09:35:00-05:00 2021-02-23 09:35:00-05:00
2021-02-23 09:36:00-05:00 2021-02-23 09:36:00-05:00
```

2.0.4 How we got data for two days

In two consecutive days (Feb 23 and 24) we ran the script above to download data. **These happened after market close each day**, so that the spot price to consider is the market close for the underlyings.

We now join them into a single file.

Closing prices

[127]:
```
spot_close_list = ['data_downloads/SPY ^VIX AMZN__equity_2021-02-23.csv',
                   'data_downloads/SPY ^VIX AMZN__equity_2021-02-24.csv']
```

[128]:
```
spot_closes = pd.concat([pd.read_csv(file) for file in spot_close_list])
```

We will join this table with the options table to get the spot values for the underlying assets

[129]:
```
spot_closes = spot_closes[['ticker', 'download_date', 'Close', 'Volume']]
```

Options

[12]:
```
option_file_list = ['data_downloads/SPY ^VIX AMZN__options_2021-02-23.csv',
                    'data_downloads/SPY ^VIX AMZN__options_2021-02-24.csv']
```

[13]:
```
options = pd.concat([pd.read_csv(file, index_col=0) for file in␣
↪option_file_list])
```

2.0.5 Filter by times to maturity

We want to keep with $T - t < 3$ months; furthermore, the "no-same-week" filter has already been applied

[20]:
```
options['time_to_maturity'] = pd.to_datetime(options['maturity'],␣
↪format='%Y-%m-%d') - pd.to_datetime(options['download_date'],␣
↪format='%Y-%m-%d')
```

[24]:
```
options = options[(options['time_to_maturity'] <= pd.Timedelta('90 days'))]
```

[27]:
```
options['time_to_maturity'].describe()
```

[27]:
```
count                         10979
mean     32 days 00:23:05.044175243
std      23 days 03:18:33.296568793
min                5 days 00:00:00
25%               16 days 00:00:00
50%               24 days 00:00:00
75%               37 days 00:00:00
max               87 days 00:00:00
Name: time_to_maturity, dtype: object
```

Create date identifier following problem statement for Exercise Part 1.2

[34]:
```
options['day_identifier'] = options['download_date'].map({'2021-02-23':␣
↪'DATA_1', '2021-02-24': 'DATA_2'})
```

Finally, join with spot prices:

[51]:
```
options.head()
```

[51]:
```
        contractSymbol        lastTradeDate  strike  lastPrice    bid     ask  \
0  SPY210301C00210000  2021-02-05 15:19:26   210.0     176.62  176.9  177.58
1  SPY210301C00220000  2021-01-29 14:36:32   220.0     154.46  166.9  167.60
2  SPY210301C00295000  2021-01-25 14:30:00   295.0      89.82   92.2   92.30
3  SPY210301C00300000  2021-02-02 15:52:08   300.0      82.51   87.2   87.30
4  SPY210301C00305000  2021-02-19 15:02:17   305.0      87.12   82.2   82.30

   change  percentChange  volume  openInterest  impliedVolatility  inTheMoney  \
0     0.0            0.0    24.0          24.0           1.855469        True
1     0.0            0.0     1.0           0.0           1.734376        True
2     0.0            0.0     3.0           3.0           0.900392        True
3     0.0            0.0     1.0          10.0           0.851564        True
4     0.0            0.0     1.0          21.0           0.802736        True

  contractSize currency  type    maturity download_date ticker_underlying  \
0      REGULAR      USD  call  2021-03-01    2021-02-23                SPY
1      REGULAR      USD  call  2021-03-01    2021-02-23                SPY
2      REGULAR      USD  call  2021-03-01    2021-02-23                SPY
3      REGULAR      USD  call  2021-03-01    2021-02-23                SPY
4      REGULAR      USD  call  2021-03-01    2021-02-23                SPY

  time_to_maturity day_identifier
0           6 days         DATA_1
1           6 days         DATA_1
2           6 days         DATA_1
3           6 days         DATA_1
4           6 days         DATA_1
```

[53]:
```
len(options)
```

[53]:
```
10979
```

[56]:
```
options = options.merge(spot_closes,
             left_on=['ticker_underlying', 'download_date'],
             right_on=['ticker', 'download_date'],
             how='left'
            ).drop('ticker_underlying', axis=1)
```

[58]:
```
assert options.isna().sum().sum() == 0, "There were NAs after merge"
```

2.0.6 (Part 1.3)

Write a paragraph describing the symbols you are downloading data for. Explain what is the SPY and its purpose.

SPY: is a ETF tracking the S&P500. This SPY is traded in exchanges. Holding the SPY ETF you have exposure to Stocks but not holding stocks properly. SPY ETF pays dividends quarterly, SPX Index not. The

VIX: this Index measures the volatility of S&P500 options. VIX is also well known by "Fear Index" in the Financial Market, because in times of more uncertainty is more volatile and this VIX Index could be a higher. So its a good measure to see how is the market mood.

We are downloading Stocks, Index, ETF and Stock Options. Index and ETF are represented by 3 letters, the main letters representing the asset. Stocks are represented by 4 letters. Stock Options tickers are a bit more longer because we have more information there. I will take one Amazon Option as example. AMZN210305C01950000: - The first 4 letters are exactly like the Stock Ticker. - The next 6 digits are representing the Expiration Date - After the Expiration Date, we can find 'C' or 'P', representing 'Call' or 'Put' - Finally, the last digits of the Stock Options Ticker are representing the Strike. The answer is yes, you can determine the expiration date just looking at the Options Ticker. This Stock Option 'AMZN210305C01950000' is expiring March-05-2021 (just analyzed the 6 digits after 'AMZN')

2.0.7 (Part 1.4) Interest rates

Annualize time to maturity

[171]:
```
options['time_to_maturity'] = options['time_to_maturity'].dt.days/360
```

From the link provided, we use a risk-free rate of 0.07 - Federal Funds (efective):

https://www.federalreserve.gov/releases/h15/

[205]:
```
r = 0.07/100
```

3 Part 2

3.0.1 (Part 2.5)

Black-Scholes option pricing formula

$$C(S_t, K, T, t, r, \sigma) = S_t N(d_1) - K e^{-r(T-t)} N(d_2)$$

with

$$d_{1,2} = \frac{\log S_t/K + (r \pm \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}}$$

and N is the CDF of a standard normal variable, i.e.

$$N(x) := \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2} 2dy$$

[124]:
```
from scipy import stats
```

[174]:
```
def bsm_value(option_type, S, K, T, r, sigma):  # T here stands for time to␣
↪maturity
    """
    Returns Black-Scholes formula
    """
```

```
    assert option_type in ['put', 'call'], "Invalid option type"

    phi = 1 if option_type == 'call' else -1  # multiplication factor to adjust␣
↪for option type

    N = lambda x: stats.norm.cdf(x, loc=0.0, scale=1.0) # Gaussian CDF
    d1 = 1 if T == 0 else (np.log(S/K) + (r + sigma**2/2)*T)/(sigma*np.sqrt(T))
    d2 = d1 - sigma*np.sqrt(T)

    return phi*(S * N(phi*d1) - K * np.exp(-r*T) * N(phi*d2))
```

3.0.2 (Part 2.6)

[254]:
```
def root_finder_bisection(func, lower_bound, upper_bound, tolerance,␣
↪max_iter=10000):
    if np.sign(func(lower_bound)) == np.sign(func(upper_bound)):
        return np.nan
    else:
        n_iter = 1
        error = np.inf
        a, b = lower_bound, upper_bound

        while (n_iter < max_iter) & (error > tolerance):
            c = (a + b)/2
            error = (b-a)/2
            if (func(c) == 0) or (error < tolerance):
                return c
            n_iter += 1
            if np.sign(func(c)) == np.sign(func(a)):
                a = c
            else:
                b = c
    print("Max iter reached")
```

[255]:
```
### Test: calculate root of 3
root_finder_bisection(lambda x: x**2 - 3, 0, 5, 0.001)
```

[255]: 1.7315673828125

[256]:
```
options['option_price'] = options[['ask','bid']].mean(axis=1)
```

[257]:
```
# We do not need to worry about zero volume trades
(options['volume'] == 0).sum()
```

[257]: 0

[274]:
```
data_1 = options[(options['day_identifier'] == 'DATA_1')&(options['ticker'] !=␣
↪'^VIX')]
```

[275]:
```
%%time
data_1['vol_bisect'] = data_1.progress_apply(lambda row: root_finder_bisection(
    lambda sigma: bsm_value(row['type'], row['Close'], row['strike'],␣
↪row['time_to_maturity'], r, sigma) - row['option_price'],
    lower_bound=0.001,
    upper_bound=100.0,
    tolerance=1e-6
    ), axis=1
)
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=5091.0),␣
↪HTML(value='')))
```

```
Wall time: 4min 10s
```

Sanity check: calculate Black-Scholes price using these values and remove any miscalculations and NAs

[276]:
```
data_1['price_bs'] = data_1.progress_apply(lambda row: bsm_value(row['type'],␣
↪row['Close'], row['strike'], row['time_to_maturity'], r, row['vol_bisect']),␣
↪axis=1)
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=5091.0),␣
↪HTML(value='')))
```

[277]:
```
print(f"Got null results for {round(100*data_1['vol_bisect'].isna().sum()/
↪len(data_1))}% of data")
```

```
Got null results for 4% of data
```

3.0.3 (Part 2.7)

[278]:
```
def bsm_vega(S, K, T, r, sigma):
    """
    Calculates vega for the Black-Scholes formula
    """

    N_prime = lambda x: stats.norm.pdf(x, loc=0.0, scale=1.0)
    d1 = 1 if T == 0 else (np.log(S/K) + (r  + sigma**2/2)*T)/(sigma*np.sqrt(T))

    return S * N_prime(d1) * np.sqrt(T)

def bsm_imp_vol(option_type, S, K, T, r, P0, n_iter=100):
```

```
    """
    Solves for implicit volatily using Newton's method
    """
    assert option_type in ['put', 'call'], "Invalid option type"

    s = 0.7  # initial guess
    for _ in range(n_iter):
        numerator = bsm_value(option_type, S, K, T, r, s) - P0
        if np.abs(numerator) < 1e-5:
            break
        s -= numerator/bsm_vega(S, K, T, r, s)

    return s
```

[279]:
```
%%time
data_1['vol_newton'] = data_1.progress_apply(lambda row: bsm_imp_vol(
    row['type'], row['Close'], row['strike'], row['time_to_maturity'], r,␣
 ↪row['option_price']), axis=1)
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=5091.0),␣
 ↪HTML(value='')))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:22:
RuntimeWarning: divide by zero encountered in double_scalars
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:11:
RuntimeWarning: invalid value encountered in double_scalars
  # This is added back by InteractiveShellApp.init_path()
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:7:
RuntimeWarning: invalid value encountered in double_scalars
  import sys
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:11:
RuntimeWarning: overflow encountered in double_scalars
  # This is added back by InteractiveShellApp.init_path()
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:7:
RuntimeWarning: overflow encountered in double_scalars
  import sys
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:22:
RuntimeWarning: overflow encountered in double_scalars


Wall time: 35.8 s
```

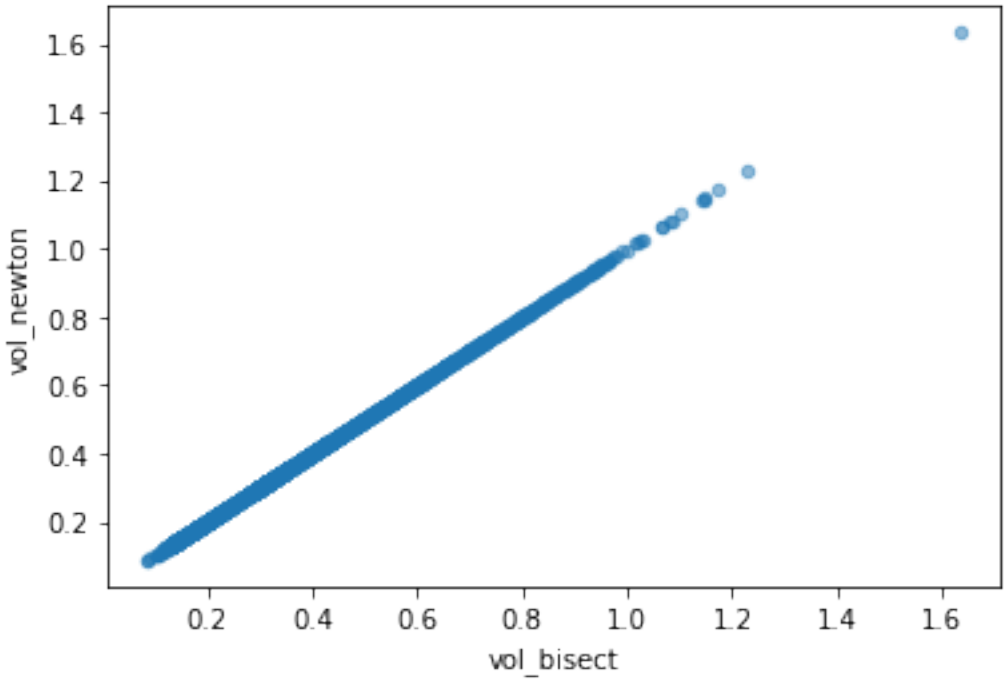
[280]:
```
print(f"Got null results for {round(100*data_1['vol_newton'].isna().sum()/
 ↪len(data_1))}% of data")
```

```
Got null results for 5% of data
```

We see that, as expected, Newton's method is faster than bisection, for very close values. We still see some numerical issues (5% of data cannot be calculated this way)

[281]:
```
data_1.plot.scatter(x='vol_bisect', y='vol_newton', alpha=0.5)
```

[281]:
```
<AxesSubplot:xlabel='vol_bisect', ylabel='vol_newton'>
```



3.0.4 (Part 2.8)

[282]:
```
data_1.groupby(['ticker', 'strike', 'maturity', 'type'])[['vol_bisect',␣
↪'vol_newton']].mean()
```

[282]:

ticker	strike	maturity	type	vol_bisect	vol_newton
AMZN	1460.0	2021-03-19	call	3.030417	NaN
			put	1.010442	NaN
		2021-04-16	call	1.309761	NaN
			put	0.711408	0.711408
	1480.0	2021-03-19	call	2.429952	NaN
...				...	...
SPY	515.0	2021-04-16	call	0.240540	0.240542
		2021-05-21	call	0.204613	0.204613
	520.0	2021-05-21	call	0.207262	0.207262
	525.0	2021-05-21	call	0.216875	0.216874
	530.0	2021-05-21	call	0.222889	0.222888

[5091 rows x 2 columns]

TO DO: compare with VIX

3.0.5 (Part 2.9)

$$C(t) - P(t) = S(t) - Ke^{-r(T-t)}$$

[287]:
```
# get put price from call price
data_1.loc[data_1['type'] == 'call','parity_price'] = data_1['option_price'] -␣
 ↪data_1['Close'] + data_1['strike']*np.exp(-r * data_1['time_to_maturity'])
# get call price from put price
data_1.loc[data_1['type'] == 'put','parity_price'] = data_1['option_price'] +␣
 ↪data_1['Close'] - data_1['strike']*np.exp(-r * data_1['time_to_maturity'])
```

[294]:
```
data_1.head()
```

[294]:

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	\
0	SPY210301C00210000	2021-02-05 15:19:26	210.0	176.62	176.9	177.58	
1	SPY210301C00220000	2021-01-29 14:36:32	220.0	154.46	166.9	167.60	
2	SPY210301C00295000	2021-01-25 14:30:00	295.0	89.82	92.2	92.30	
3	SPY210301C00300000	2021-02-02 15:52:08	300.0	82.51	87.2	87.30	
4	SPY210301C00305000	2021-02-19 15:02:17	305.0	87.12	82.2	82.30	

	change	percentChange	volume	openInterest	...	download_date	\
0	0.0	0.0	24.0	24.0	...	2021-02-23	
1	0.0	0.0	1.0	0.0	...	2021-02-23	
2	0.0	0.0	3.0	3.0	...	2021-02-23	
3	0.0	0.0	1.0	10.0	...	2021-02-23	
4	0.0	0.0	1.0	21.0	...	2021-02-23	

	time_to_maturity	day_identifier	ticker	Close	option_price	vol_bisect	\
0	0.016667	DATA_1	SPY	387.029999	177.24	1.986845	
1	0.016667	DATA_1	SPY	387.029999	167.25	1.858901	
2	0.016667	DATA_1	SPY	387.029999	92.25	0.964339	
3	0.016667	DATA_1	SPY	387.029999	87.25	0.911661	
4	0.016667	DATA_1	SPY	387.029999	82.25	0.859593	

	price_bs	vol_newton	parity_price
0	177.239999	NaN	0.207551
1	167.250000	NaN	0.217435
2	92.250001	0.964341	0.216560
3	87.250001	0.911661	0.216501
4	82.250000	0.859593	0.216443

[5 rows x 26 columns]

[313]:
```
parity_check = data_1[['strike', 'bid', 'ask', 'maturity', 'ticker',␣
 ↪'option_price', 'type', 'parity_price']]
```

```
parity_check = parity_check.rename({'type': 'original_type'}, axis=1)
parity_check['parity_type'] = parity_check['original_type'].map({'call': 'put',␣
↪'put': 'call'})
```

[318]:
```
parity_check = parity_check.drop(['bid', 'ask'], axis=1).\
    merge(parity_check.drop(['parity_price', 'parity_type', 'option_price'],␣
↪axis=1),
    left_on=['strike', 'maturity', 'ticker', 'parity_type'],
    right_on=['strike', 'maturity', 'ticker', 'original_type'],
    how='left').drop('original_type_y', axis=1).dropna()
```

[321]:
```
parity_check.sample(40).head()
```

[321]:
```
      strike    maturity ticker  option_price original_type_x  parity_price  \
1204   394.0  2021-03-17    SPY         3.370            call     10.323147
2106   301.0  2021-03-31    SPY        86.640            call      0.588932
2933   414.0  2021-04-16    SPY        29.130             put      2.201857
1489   411.0  2021-03-19    SPY         0.255            call     24.205822
930    387.0  2021-03-12    SPY         6.320            call      6.277209

     parity_type    bid    ask
1204         put  10.02  10.14
2106         put   0.52   0.54
2933        call   1.22   1.25
1489         put  25.14  25.23
930          put   5.99   6.06
```

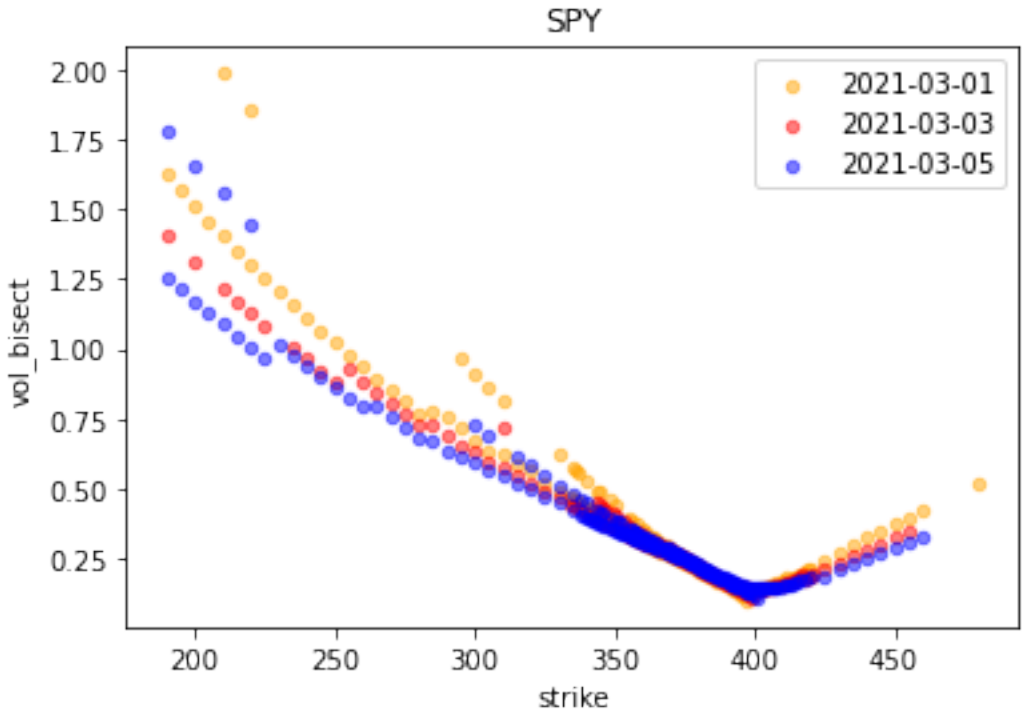
The numbers are in line with the bid/offer spread, but in some examples a little off than expected, but even there the numbers makes sense. Couple factors as a different fundamental parameters can affect the final result.

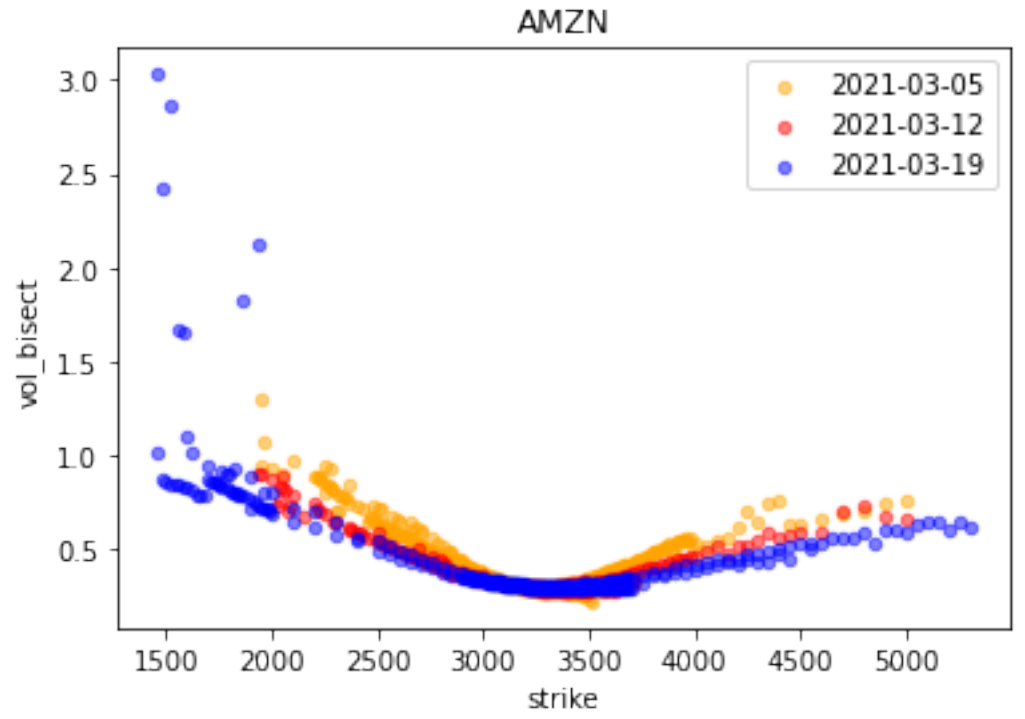
3.0.6 (Part 2.10)

[344]:
```
import matplotlib.pyplot as plt
```

[352]:
```
for ticker in data_1['ticker'].unique():

    fig, ax = plt.subplots()
    ax.set_title(ticker)
    aux = data_1[data_1['ticker'] == ticker]
    shortest_maturities = sorted(list(set(aux['time_to_maturity'])))[:3]

    for color, maturity in zip(['orange', 'red', 'blue'], shortest_maturities):
        aux2 = aux[aux['time_to_maturity'] == maturity]
        maturity_str = aux2['maturity'].iloc[0]
```

```
        aux2.plot.scatter(x='strike', y='vol_bisect', ax=ax, c=color, alpha=0.5,␣
↪label=maturity_str)
```





3.0.7 (Part 2.11)

Vega is already calculated above:

$$\text{vega} = SN'(d_1)\sqrt{T-t}$$

Delta:

$$\Delta = N(d_1) \quad \text{(call)}$$
$$\Delta = N(d_1) - 1 \quad \text{(put)}$$

Gamma:

$$\Gamma = \frac{1}{S\sigma\sqrt{T-t}}N'(d_1)$$

[363]:
```
def bsm_delta(option_type, S, K, T, r, sigma):
    N = lambda x: stats.norm.cdf(x, loc=0.0, scale=1.0)
    d1 = 1 if T == 0 else (np.log(S/K) + (r  + sigma**2/2)*T)/(sigma*np.sqrt(T))
    extra = 0 if option_type == 'call' else -1

    return N(d1) + extra

def bsm_gamma(S, K, T, r, sigma):
    N_prime = lambda x: stats.norm.pdf(x, loc=0.0, scale=1.0)
    d1 = 1 if T == 0 else (np.log(S/K) + (r  + sigma**2/2)*T)/(sigma*np.sqrt(T))

    return 1.0/(S * sigma * np.sqrt(T)) * N_prime(d1)
```

[365]:
```
greeks = data_1.dropna()
```

[366]:
```
greeks['vega'] = greeks.apply(lambda x: bsm_vega(S=x['Close'], K=x['strike'],␣
 ↪T=x['time_to_maturity'], r=r, sigma=x['vol_bisect']), axis=1)
greeks['delta'] = greeks.apply(lambda x: bsm_delta(option_type=x['type'],␣
 ↪S=x['Close'], K=x['strike'], T=x['time_to_maturity'], r=r,␣
 ↪sigma=x['vol_bisect']), axis=1)
greeks['gamma'] = greeks.apply(lambda x: bsm_gamma(S=x['Close'], K=x['strike'],␣
 ↪T=x['time_to_maturity'], r=r, sigma=x['vol_bisect']), axis=1)
```

[375]:
```
greeks[['ticker', 'type', 'Close', 'strike', 'maturity', 'option_price',␣
 ↪'vol_bisect', 'vega', 'delta', 'gamma']].sample(10)
```

[375]:

	ticker	type	Close	strike	maturity	option_price	vol_bisect	\
1054	SPY	put	387.029999	378.0	2021-03-12	3.635	0.219578	
4374	AMZN	call	3180.739990	3295.0	2021-03-19	55.025	0.306229	
4150	AMZN	put	3180.739990	2020.0	2021-03-12	0.235	0.726579	
2639	SPY	call	387.029999	317.0	2021-04-16	71.395	0.348461	

```
3256    SPY    put   387.029999   371.0  2021-05-21        11.135    0.243053
1735    SPY    put   387.029999   392.0  2021-03-19        10.080    0.182961
4562   AMZN    put  3180.739990  3180.0  2021-03-19        98.725    0.302798
1885    SPY   call   387.029999   380.0  2021-03-26        13.110    0.203941
4988   AMZN    put  3180.739990  3150.0  2021-04-01       109.500    0.307346
4857   AMZN    put  3180.739990  3250.0  2021-03-26       150.150    0.298331
```

```
            vega     delta     gamma
1054   29.320139 -0.301771  0.018877
4374  301.688196  0.342287  0.001461
4150    3.505907 -0.001565  0.000010
2639   16.999554  0.942272  0.002255
3256   69.635726 -0.339004  0.007915
1735   38.683204 -0.596962  0.021172
4562  327.338697 -0.482985  0.001603
1885   42.804450  0.632030  0.016272
4988  402.342186 -0.440952  0.001259
4857  364.873657 -0.579885  0.001404
```

Numerical approximation:

$$\frac{\partial C}{\partial S} \approx \frac{C(S+\delta) - C(S-\delta)}{2\delta}$$

$$\frac{\partial^2 C}{\partial S^2} \approx \frac{C(S+\delta) - 2C(S) + C(S-\delta)}{\delta^2}$$

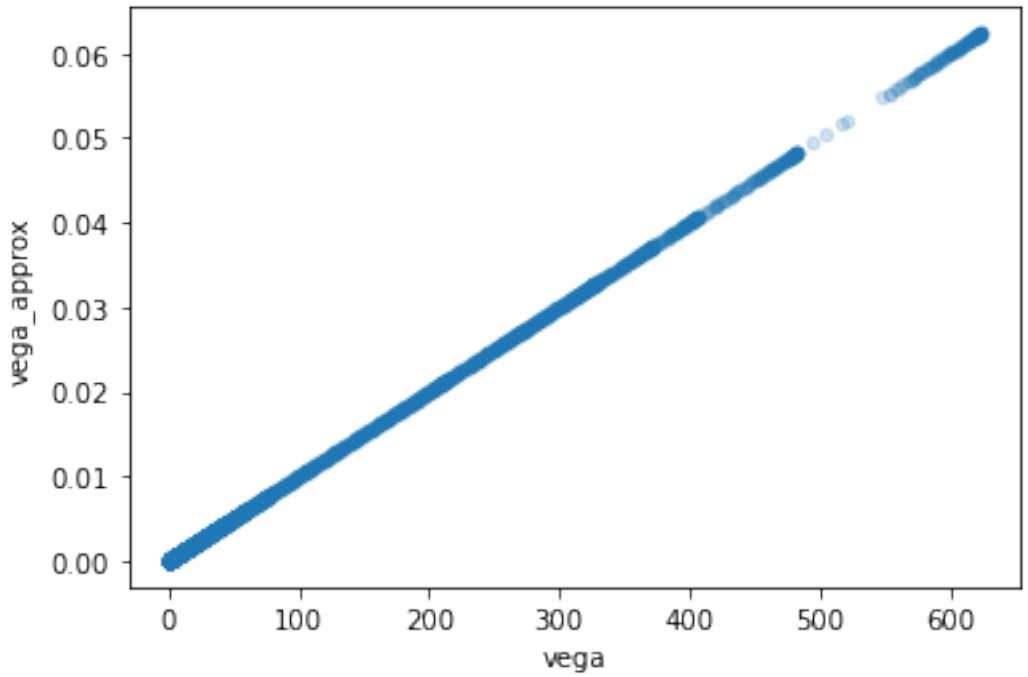
[377]:
```
# using the values below for the numerical differences
delta_vol = 0.01
delta_spot = 10
```

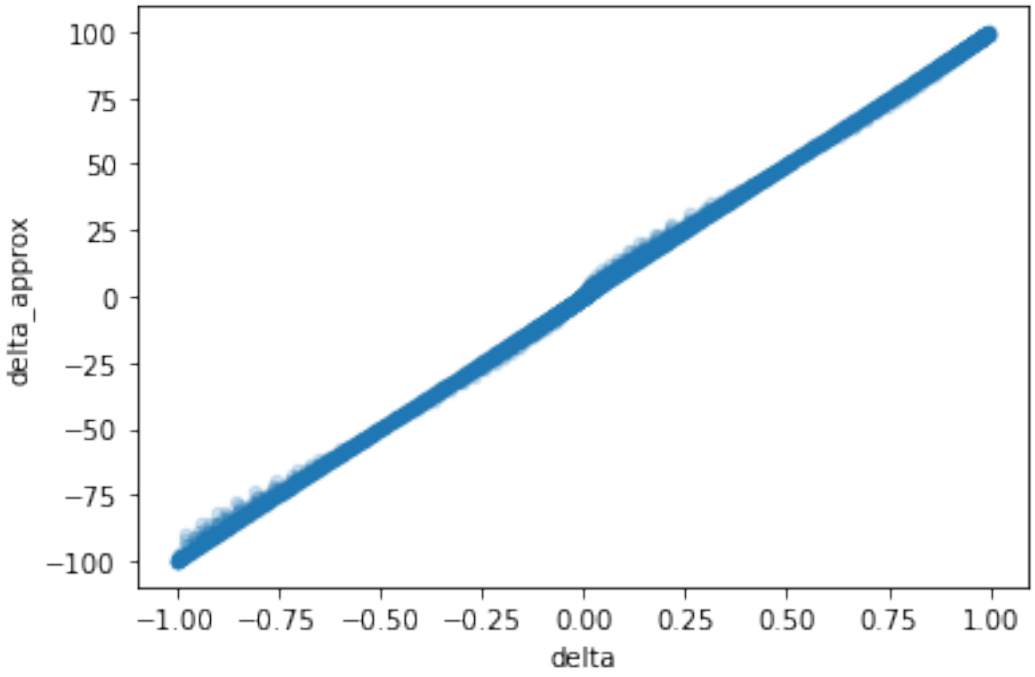
[380]:
```
greeks['vega_approx'] = (greeks.apply(lambda x: bsm_value(x['type'], x['Close'],␣
↪x['strike'], x['time_to_maturity'], r,
                                                          x['vol_bisect'] +␣
↪delta_vol), axis=1)
                         - greeks.apply(lambda x: bsm_value(x['type'],␣
↪x['Close'], x['strike'], x['time_to_maturity'], r,
                                                          x['vol_bisect'] -␣
↪delta_vol), axis=1)
                         )/2*delta_vol

greeks['delta_approx'] = (greeks.apply(lambda x: bsm_value(x['type'],␣
↪x['Close']+delta_spot,
                                                            x['strike'],␣
↪x['time_to_maturity'], r, x['vol_bisect']), axis=1)
                         - greeks.apply(lambda x: bsm_value(x['type'],␣
↪x['Close']-delta_spot,
```

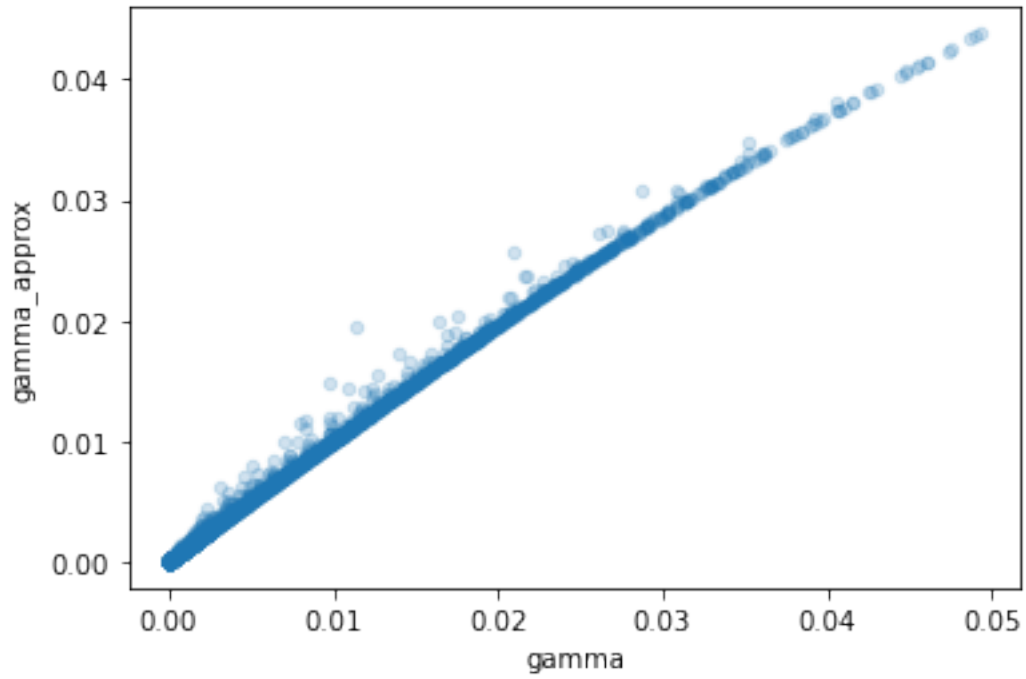
```
                                                                x['strike'],␣
↪x['time_to_maturity'], r, x['vol_bisect']), axis=1)
                        )/2*delta_spot

greeks['gamma_approx'] = (greeks.apply(lambda x: bsm_value(x['type'],␣
↪x['Close']+delta_spot,
                                                                x['strike'],␣
↪x['time_to_maturity'], r, x['vol_bisect']), axis=1)
                        - 2*greeks.apply(lambda x: bsm_value(x['type'],␣
↪x['Close'],
                                                                x['strike'],␣
↪x['time_to_maturity'], r, x['vol_bisect']), axis=1)
                        + greeks.apply(lambda x: bsm_value(x['type'],␣
↪x['Close']-delta_spot,
                                                                x['strike'],␣
↪x['time_to_maturity'], r, x['vol_bisect']), axis=1)
                        )/(delta_spot)**2
```

Plotting to check how similar they are:

[384]:
```
for greek in ['vega', 'delta', 'gamma']:
    greeks.plot.scatter(x=greek, y=f'{greek}_approx', alpha=0.2)
```




delta_approx
100
75
50
25
0
−25
−50
−75
−100
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
delta


gamma_approx
0.04
0.03
0.02
0.01
0.00
0.00
0.01
0.02
0.03
0.04
0.05
gamma

3.0.8 (Part 2.12)

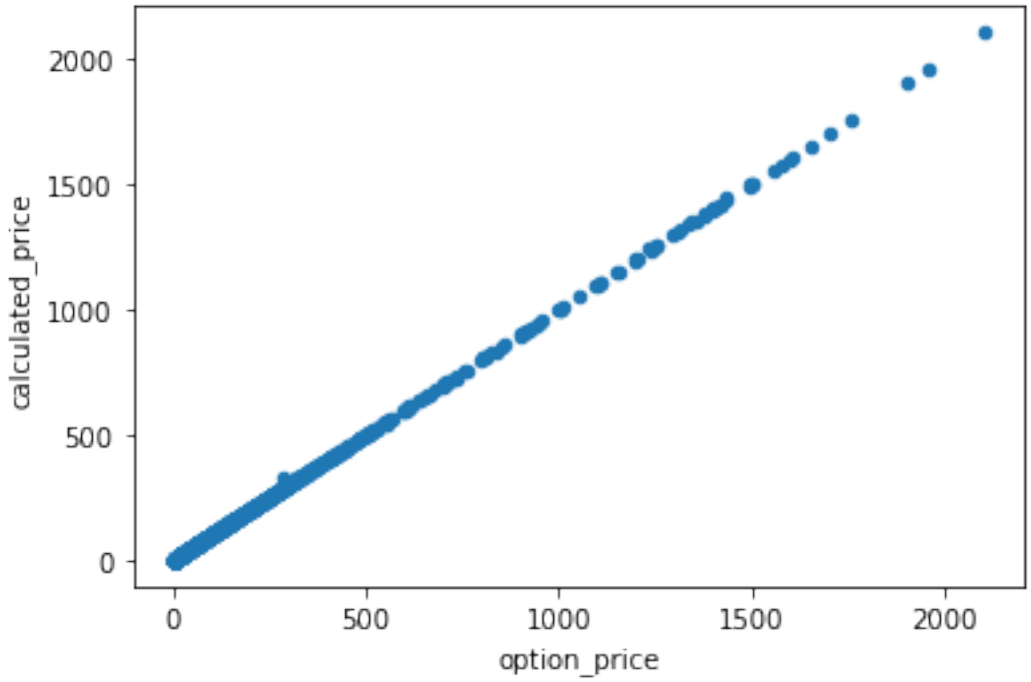
[354]:
```
data_2 = options[(options['day_identifier'] == 'DATA_2') & (options['ticker'] !=␣
↪'^VIX')]
```

[357]:
```
data_2 = data_2.merge(data_1.dropna()[['contractSymbol', 'vol_bisect']],␣
↪how='inner')
```

[359]:
```
data_2['calculated_price'] = data_2.apply(lambda row: bsm_value(row['type'],␣
↪row['Close'], row['strike'], row['time_to_maturity'], r, row['vol_bisect']),␣
↪axis=1)
```

[361]:
```
data_2[['option_price', 'calculated_price']].plot.scatter(x='option_price',␣
↪y='calculated_price')
```

[361]:
```
<AxesSubplot:xlabel='option_price', ylabel='calculated_price'>
```



4 (Part 3)

4.0.1 (Part 3.1)

1 - (a) Trapezoidal Method

[386]:
```
from math import sin, e, pi
```

[387]:
```
#function f(x) = sin(x)/x

def f(x):
    if x !=0:
        return sin(x)/x
    if x ==0:
        return 1
```

[404]:
```
def Trapezoidal (f,a,b,N):
    h = (b-a)/N
    s = 0.5*f(a) + 0.5*f(b)

    for k in (range(1,N)):
        s+= f(a+k*h)

    return h*s
```

[405]:
```
integral_trap = Trapezoidal(f, -10**4, 10**4, 100000)
integral_trap
```

[405]: 3.141782455598503

[406]:
```
erro_trap = integral_trap - pi
erro_trap
```

[406]: 0.00018980200870988284

1 - (b) Simpson's Method

[407]:
```
def Simpson(f, a, b, n):

    h = (b - a) / n
    k = 0
    z = 0

    for i in (range(1, n // 2)):
        k += 2 * f(a + 2 * i * h)

    for i in (range(1, n // 2 + 1)):
        z += 4 * f(a + (2 * i - 1) * h)

    return h * (f(a) + k + z + f(b)) / 3.0
```

[408]:
```
integral_simp = Simpson(f, -10**5, 10**5, 1000000)
integral_simp
```

[408]: 3.1416126409773426

[409]:
```
erro_simp = integral_simp - pi
erro_simp
```

[409]:
```
1.9987387549491586e-05
```

4.0.2 (Part 3.2) - Truncation Error

[443]:
```
def truncation_error(f, int_method, a, N):
    integration = int_method(f, -a, a, N)
    error = np.abs(integration - pi)
    return error
```

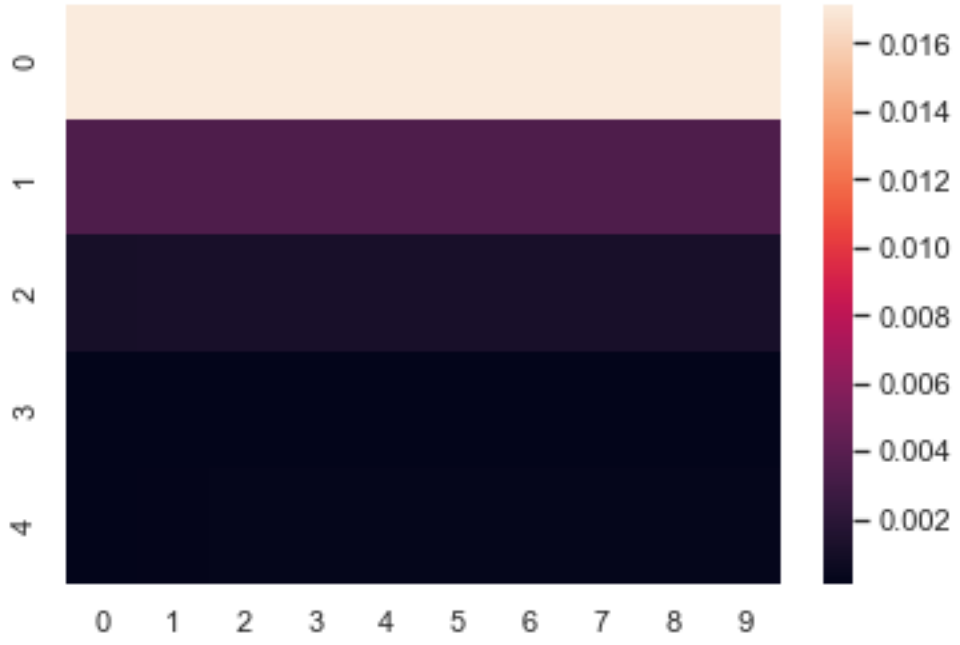
[444]:
```
from itertools import product
```

[445]:
```
a_range = [100, 500, 1000, 5000, 10000]
N_range = range(10000, 1000000, 100000)
```

[446]:
```
error = np.zeros((len(a_range), len(N_range)))
```

[447]:
```
for i, a in enumerate(a_range):
    for j, N in enumerate(N_range):
        error[i][j] = truncation_error(f, Trapezoidal, a, N)
```

[449]:
```
import seaborn as sns; sns.set_theme()
ax = sns.heatmap(error)
```



Very little sensitivity to N (horizontal axis); for a given a, convergence is pretty fast for sufficiently high N

4.0.3 (Part 3.3) - Convergence

[466]:
```
n_list = range(10,10000,10)
```

For trapezoidal:

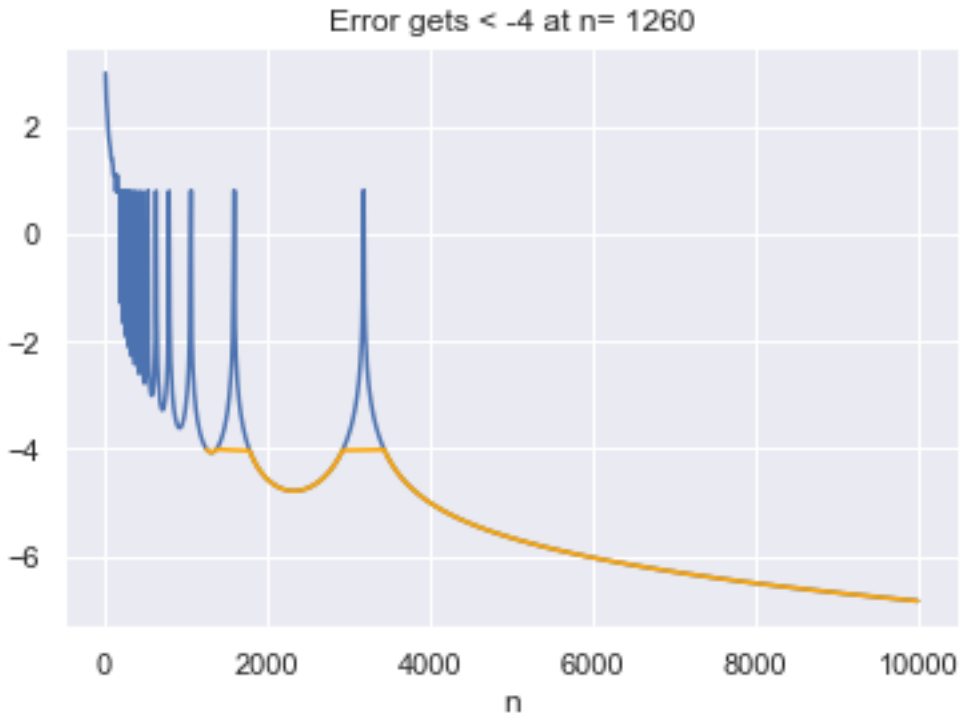
[467]:
```
trap_iters = pd.DataFrame({'n': n_list,
                           'int': [Trapezoidal(f, -10**4, 10**4, n) for n in␣
 ↪tqdm(n_list)]})
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=999.0),␣
 ↪HTML(value='')))
```

[469]:
```
trap_iters['error'] = np.abs(trap_iters['int'] - trap_iters['int'].shift(1))
trap_iters['log_error'] = np.log10(trap_iters['error'])
```

Show where error becomes smaller than 10-4, or in log scale, <-4:

[478]:
```
aux = trap_iters.set_index('n')['log_error']
aux.plot()
aux[aux<-4].plot(color='orange')
plt.title(f"Error gets < -4 at n= {aux[aux<-4].head(1).index[0]}")
plt.show()
```



For Simpson:

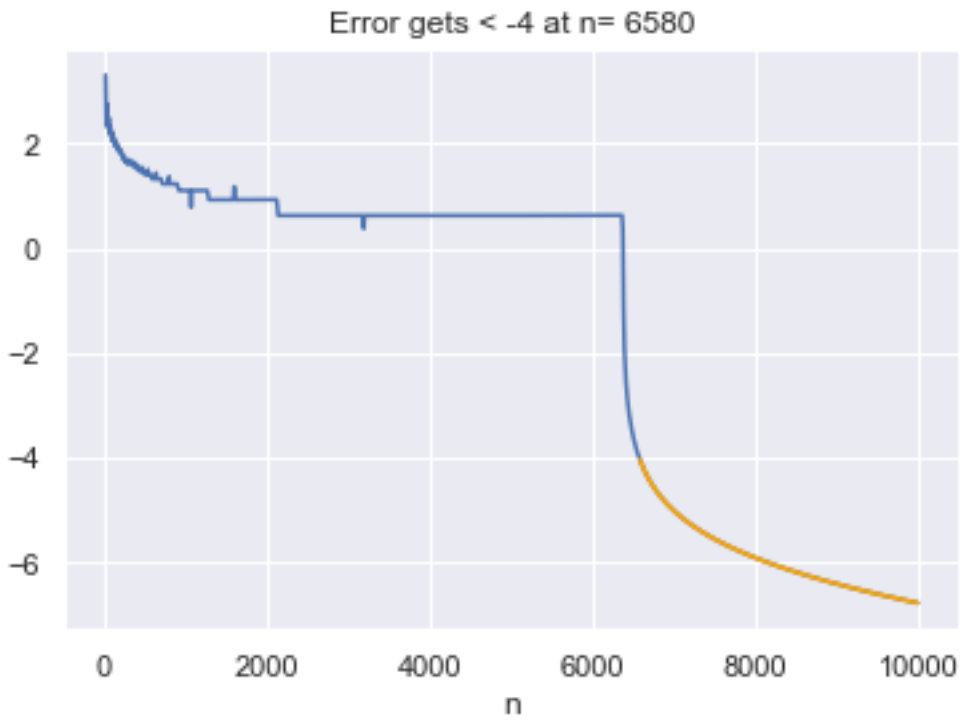
[479]:
```
simps_iters = pd.DataFrame({'n': n_list,
                            'int': [Simpson(f, -10**4, 10**4, n) for n in␣
 ↪tqdm(n_list)]})
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=999.0),␣
 ↪HTML(value='')))
```

[480]:
```
simps_iters['error'] = np.abs(simps_iters['int'] - simps_iters['int'].shift(1))
simps_iters['log_error'] = np.log10(simps_iters['error'])
```

Show where error becomes smaller than 10-4, or in log scale, <-4:

[481]:
```
aux = simps_iters.set_index('n')['log_error']
aux.plot()
aux[aux<-4].plot(color='orange')
plt.title(f"Error gets < -4 at n= {aux[aux<-4].head(1).index[0]}")
plt.show()
```



We see that the trapezoidal method oscillates but converges faster, whereas Simpson has a monotonic decrease in error but converges at higher values of n

4.0.4 (Part 3.4) - g(x) and convergence

[6]:
```
#TOL was determined by the text
TOL = 1e-4
```

[484]:
```
#function g(x) = 1 + e^(-x^2)*sin(8x^(2/3))

def g(x):
    return (1 + (e**(-x**2))*sin(8*(x**(2/3))))
```

[528]:
```
def integrate(g, method):
    i0 = 0.01
    for n in range(1,100000,10000):
        i1 = method(g, 0, 2, n)
        error = np.abs((i1-i0)/i0)
        if error < TOL:
            break
        else:
            i0 = i1
```

```
    return round(i1,4)
```

[529]:
```
integrate(g, Trapezoidal)
```

[529]: 2.0375

[530]:
```
integrate(g, Simpson)
```

[530]: 2.0374