```python
# modules.opts.py
# https://github.com/QuantCS109/TrumpTweets/blob/master/modules/opts.py
# This is a .py file used in the notebooks

import numpy as np
import pandas as pd
from scipy.stats import norm
from scipy.optimize import newton
from datetime import timedelta, date
from dateutil.relativedelta import relativedelta
from numpy.polynomial.polynomial import polyval
import pickle
import re


def black(fwd , k, vol , r, t, opt_type='call'):
    """
    This function calculates option prices through Black-76 model
    Source: https://en.wikipedia.org/wiki/Black_model
    :param fwd: forward price of underlying
    :param k: strike
    :param vol: volatility
    :param r: interest rate
    :param t: time to expiration
    :param opt_type: option type (call or put)
    :return: option price
    """
    d1 = (np.log(fwd / k) + (vol**2/2)*t) / (vol * np.sqrt(t))
    d2 = d1 - vol * np.sqrt(t)
    if opt_type == 'call':
        opt_px = np.exp(-r * t) * (fwd * norm.cdf(d1) - k * norm.cdf(d2))
    else:
        opt_px = np.exp(-r * t) * (k * norm.cdf(-d2) - fwd * norm.cdf(-d1))
    return opt_px


def delta(fwd , k, vol , r, t, opt_type='call'):
    """
    This function calculates the delta of an option assuming Black-76 model
    Source: https://www.glynholton.com/notes/black_1976/
    :param fwd: forward price of underlying
    :param k: strike
    :param vol: volatility
    :param r: interest rate
    :param t: time to expiration
    :param opt_type: option type (call or put)
    :return: delta
    """
    d1 = (np.log(fwd / k) + (vol**2/2)*t) / (vol * np.sqrt(t))
    if opt_type == 'call':
        delta = np.exp(-r * t) * norm.cdf(d1)
    else:
        delta = np.exp(-r * t) * (norm.cdf(d1) - 1)
    return delta


def gamma(fwd , k, vol , r, t):
    """
    This function calculates the gamma of an option assuming Black-76 model
    Source: https://www.glynholton.com/notes/black_1976/
    :param fwd: forward price of underlying
    :param k: strike
    :param vol: volatility
    :param r: interest rate
    :param t: time to expiration
    :return: gamma
    """
```

```
68      d1 = (np.log(fwd / k) + (vol ** 2 / 2) * t) / (vol * np.sqrt(t))
69      return np.exp(-r * t) * norm.pdf(d1)
70
71
72  def imp_vol(opt_px, fwd, k, r, t, opt_type='call'):
73      """
74      This function calculates the implied volatility of an option assuming Black-76 model
75      :param opt_px: option price
76      :param fwd: forward price of underlying
77      :param k: strike
78      :param r: interest rate
79      :param t: time to expiration
80      :param opt_type: option type (call or put)
81      :return: implied volatility
82      """
83      if opt_type == 'call':
84          vol = newton( lambda vol : black(fwd , k, vol , r, t ) - opt_px , 0.2)
85      else:
86          vol = newton( lambda vol : black(fwd , k, vol , r, t ,'put') - opt_px , 0.2)
87      return vol
88
89
90  def delta_moneyness(x, vol, r, t, opt_type='call'):
91      x = np.exp(x)
92      if opt_type == 'call':
93          return delta(1, x, vol, r, t, opt_type=opt_type)
94      else:
95          return delta(1, x, vol, r, t, opt_type=opt_type)
96
97
98  def find_delta(vol_poly, r, t, delta=0.25, opt_type='call'):
99      x = newton(lambda x : abs(delta_moneyness(x, polyval(x,vol_poly), r, t, opt_type=
        opt_type)) - delta, 0)
100     return float(x), float(polyval(x,vol_poly))
101
102
103 class CMECalendar:
104     """
105     IMM_dates: Options expirations for options that expire on IMM dates, 3rd Friday of
        each month
106     minus a day if holiday
107     EOM_dates: Options expirations for options that expire on the last business day of
        every month
108     """
109
110     IMM_dates = pd.to_datetime(['18-DEC-15','18-MAR-16','17-JUN-16','16-SEP-16',
        '16-DEC-16',
111                 '17-MAR-17','16-JUN-17','15-SEP-17','15-DEC-17','16-MAR-18','15-JUN-18',
112                '21-SEP-18','21-DEC-18','15-MAR-19','21-JUN-19','20-SEP-19','20-DEC-19',
                   '20-MAR-20'])
113     start = pd.to_datetime('31-DEC-15')
114     end = date(start.year + 5, 3, 31)
115     EOM_dates = pd.date_range(start, end, freq='BM')
116
117
118 class RatesCurve:
119     """
120     This class obtains 1 month libor's close.
121     Work in progress: Add 3 month libor and strip a curve for better results
122     In Black model, if you have the futures price, interest rates only play a factor
123     through discounting, so having a less-than-perfect estimate of the relevant interest
124     rate isn't as important as in, say, Black-Scholes.
125     """
126     def __init__(self):
127         self.rates = pd.read_csv('../data/input_data/libor_1m.csv')
128         self.rates.DATE = pd.to_datetime(self.rates.DATE)
129         self.rates = self.rates.set_index('DATE')
```

```
130             self.rates['LIBOR'] = self.rates['LIBOR'].astype(float)
131
132             self.rates_1m = self.rates
133
134             self.rates_3m = pd.read_csv('../data/input_data/libor_3m.csv')
135             self.rates_3m.DATE = pd.to_datetime(self.rates_3m.DATE)
136             self.rates_3m = self.rates_3m.set_index('DATE')
137             self.rates_3m['LIBOR'] = self.rates_3m['LIBOR'].astype(float)
138
139             self.rates_6m = pd.read_csv('../data/input_data/libor_6m.csv')
140             self.rates_6m.DATE = pd.to_datetime(self.rates_6m.DATE)
141             self.rates_6m = self.rates_6m.set_index('DATE')
142             self.rates_6m['LIBOR'] = self.rates_6m['LIBOR'].astype(float)
143
144         def get(self, today, fut_date='' ):
145             """
146             Get 1 month libor for a specific date
147             :param today: date in any format that can be converted to datetime
148             :return: 1 month libor (float)
149             """
150             current_date = pd.to_datetime(today)
151             if fut_date== '':
152                 return float(self.rates.loc[current_date])
153             else:
154                 fut_date = pd.to_datetime(fut_date)
155                 if fut_date <= current_date + relativedelta(months=1):
156                     return float(self.rates.loc[current_date])
157                 elif fut_date <= current_date + relativedelta(months=3):
158                     t0 = 0
159                     t1 = 91
160                     t = (pd.to_datetime(fut_date) - pd.to_datetime(current_date)).days
161                     r = ((t - t0) / t1) * float(self.rates_3m.loc[current_date]) + \
162                         ((t1 - t) / t1) * float(self.rates.loc[current_date])
163                     return r
164                 elif fut_date <= current_date + relativedelta(months=6):
165                     t0 = 0
166                     t1 = 182
167                     t = (pd.to_datetime(fut_date) - pd.to_datetime(current_date)).days
168                     r = ((t - t0) / t1) * float(self.rates_6m.loc[current_date]) + \
169                         ((t1 - t) / t1) * float(self.rates_3m.loc[current_date])
170                     return r
171                 else:
172                     return float(self.rates_6m.loc[current_date])
173
174
175 class FuturesCurve:
176     """
177     FuturesCurve reads a database of 1 week, 1 month, and 2 month futures prices and
        interpolates to obtain
178     futures prices for any date inside 2 months.
179     """
180
181     def __init__(self, path='../data/input_data/fut.pkl'):
182         self.instrument_list = ['ES', 'NQ', 'CD', 'EC', 'JY', 'MP', 'TY', 'US', 'C', 'S'
            , 'W', 'CL', 'GC']
183         self.df = self.load(path)
184         self.col_dict = {inst: [key for key in self.df.columns if re.match(r"{}_+".
            format(inst), key)]
185                          for inst in self.instrument_list}
186
187     def get(self, inst, today, fut_date):
188         """
189         This function returns the futures price for a specific asset, date, and expiry,
            as long as expiry
190         is within two months of date.
191         :param inst: Instrument: ['ES', 'NQ', 'CD', 'EC', 'JY', 'MP', 'TY', 'US', 'C',
            'S', 'W', 'CL', 'GC']
```

```
192             :param today: date in any format that can be converted to datetime
193             :param fut_date: Expiry date. if '1W', '1M', or '2M', obtain price for 1 week,
                1 month or 2 months,
194             respectively. Else, expiry date in any format that can be converted to datetime.
195             :return: futures price
196             """
197             # Get price directly from database
198             if fut_date == '1W':
199                 return self.df[self.col_dict[inst][0]][today]
200             if fut_date == '1M':
201                 return self.df[self.col_dict[inst][1]][today]
202             if fut_date == '2M':
203                 return self.df[self.col_dict[inst][2]][today]
204
205             # If under 1 week, assume price is 1 week price. We won't use prices under 1
                week in any
206             # calculation with this function
207             if pd.to_datetime(fut_date) <= pd.to_datetime(today) + timedelta(weeks=1):
208                 return self.df[self.col_dict[inst][0]][today]
209
210             # If within 1 month, interpolate between 1 week and 1 month
211             elif pd.to_datetime(fut_date) <= pd.to_datetime(today) + relativedelta(months=1):
212                 t0 = 0
213                 t1 =((pd.to_datetime(today) + relativedelta(months=1)) - pd.to_datetime(
                    today)).days
214                 t = (pd.to_datetime(fut_date) - pd.to_datetime(today)).days
215                 f = ((t-t0)/t1) * self.df[self.col_dict[inst][1]][today] + \
216                     ((t1-t)/t1) * self.df[self.col_dict[inst][0]][today]
217                 return f
218
219             # If under two months, interpolate between 1 month and 2 month
220             elif pd.to_datetime(fut_date) <= pd.to_datetime(today) + relativedelta(months=2)
                +timedelta(days=2):
221                 t0 = 0
222                 t1 = ((pd.to_datetime(today) + relativedelta(months=2)) +timedelta(days=2) -
223                       (pd.to_datetime(today) + relativedelta(months=1))).days
224                 t = (pd.to_datetime(fut_date)  - (pd.to_datetime(today) + relativedelta(
                    months=1))).days
225                 f = ((t-t0)/t1) * self.df[self.col_dict[inst][2]][today] + \
226                     ((t1-t)/t1) * self.df[self.col_dict[inst][1]][today]
227                 return f
228             else:
229                 raise IndexError('Date must be within two months of today')
230
231         def load(self, path):
232             pickle_in = open(path, "rb")
233             fut_pd = pickle.load(pickle_in)
234             pickle_in.close()
235             return fut_pd.fillna(fut_pd.mean())
236
237
238     class VolCurve:
239         """
240         This class has vol_poly, a dictionary with instrument codes as keys and values a
            pandas
241         dataframe with indices as dates, and columns as the oefficients of a 5 degree
            polynomial
242         which represents the volatility surface.
243         """
244         def __init__(self):
245             self.vol_poly_1M = self.load('../data/input_data/vol_poly_1M.pkl')
246             self.vol_poly_2M = self.load('../data/input_data/vol_poly_2M.pkl')
247             self.rate_curve = RatesCurve()
248             self.futures_curve = FuturesCurve()
249
250         def load(self, path):
251             pickle_in = open(path, "rb")
```

```
252         vol_poly = pickle.load(pickle_in)
253         pickle_in.close()
254         return vol_poly
255 
256 
257 class VolCurveAgg:
258 
259     def __init__(self, instrument, today, vol_dict):
260 
261         self.instrument = instrument
262         self.today = pd.to_datetime(today)
263         self.vol_dict = vol_dict
264         self.rate_curve = RatesCurve()
265         self.fut_prices = {}
266         self.vol_curve = self.calc_ivols()
267         self.up_gamma = 0
268         self.down_gamma = 0
269         self.up_gamma_5 = 0
270         self.down_gamma_5 = 0
271         self.agg_gamma()
272         self.features = self.calc_features()
273 
274     def calc_features(self):
275         features = pd.DataFrame(columns=[self.instrument + '_up_gamma',
276                                          self.instrument + '_up_gamma_5',
277                                          self.instrument + '_down_gamma',
278                                          self.instrument + '_down_gamma_5'],
279                                 index=[self.today])
280         features.loc[self.today] = [self.up_gamma, self.up_gamma_5, self.down_gamma,self
        .down_gamma_5]
281         return features
282 
283     def agg_gamma(self):
284         for key in self.vol_curve.keys():
285             r = self.rate_curve.get(self.today, key)
286             t = (key - self.today).days / 365
287             for strike in self.vol_curve[key].index:
288                 if strike >= 1.025 * self.fut_prices[key]:
289                     self.up_gamma += self.vol_curve[key].oi.loc[strike] * gamma(self.
                    fut_prices[key],
290                                                 strike,
291                                                 self.vol_curve[key].imp_vol.loc[strike],
292                                                 r,
293                                                 t)
294                     self.up_gamma_5 += self.vol_curve[key].oi.loc[strike] * gamma(1.05 *
                     self.fut_prices[key],
295                                                 strike,
296                                                 self.vol_curve[key].imp_vol.loc[strike],
297                                                 r,
298                                                 t)
299                 if strike <= 0.975 * self.fut_prices[key]:
300                     self.down_gamma_5 += self.vol_curve[key].oi.loc[strike] * gamma(0.95
                     * self.fut_prices[key],
301                                                 strike,
302                                                 self.vol_curve[key].imp_vol.loc[strike],
303                                                 r,
304                                                 t)
305                     self.down_gamma += self.vol_curve[key].oi.loc[strike] * gamma(self.
                    fut_prices[key],
306                                                 strike,
307                                                 self.vol_curve[key].imp_vol.loc[strike],
308                                                 r,
309                                                 t)
310 
311     def calc_ivols(self):
312         vc = {}
313         for key in self.vol_dict[self.today]['Call'].keys():
```

```
314
315             # See if no options data for a particular expiration date, if missing for
                either calls or puts,
316             # ignore date
317             try:
318                 if (str(self.vol_dict[self.today]['Call'][key]) == '') | (str(self.
                    vol_dict[self.today]['Put'][key]) == ''):
319                     pass
320                 else:
321                     imp_vol_dict = {}
322
323                     # Calculate options moneyness. Only consider 80%-120% moneyness
                    (fut/strike)
324                     fut = self.vol_dict[self.today]['Call'][key].future.iloc[0]
325                     expiration = self.vol_dict[self.today]['Call'][key].expiration.iloc[
                    0]
326                     # saving futures price ina  dictionary for gamma calculation
327                     self.fut_prices[expiration] = fut
328                     # Above 1 moneyness consider calls, under consider puts
329                     ind_call = (fut * 1 < self.vol_dict[self.today]['Call'][key].strike
330                                 ) &  (self.vol_dict[self.today]['Call'][key].strike <
                                    fut * 1.2)
331                     ind_put = (fut * 0.8 < self.vol_dict[self.today]['Put'][key].strike
332                                 ) &  (self.vol_dict[self.today]['Put'][key].strike < fut
                                     * 1)
333
334                     calls = self.vol_dict[self.today]['Call'][key].loc[ind_call]
335                     puts = self.vol_dict[self.today]['Put'][key].loc[ind_put]
336
337                     # cycle through all puts and calculate implied volatility
338                     for j in range(puts.shape[0]):
339                         t = expiration - self.today
340                         t = t.days
341                         imp_vol_dict[puts.strike.iloc[j]] = [imp_vol(puts.settle.iloc[j],
342                                                                      fut,
343                                                                      puts.strike.iloc[j],
344                                                                      self.rate_curve.get(self
                                                                     .today) * 0.01,
345                                                                      t / 365,
346                                                                      opt_type='put'), puts.oi
                                                                     .iloc[j]]
347
348                     # cycle through all calls and calculate implied volatility
349                     for j in range(calls.shape[0]):
350                         t = expiration - self.today
351                         t = t.days
352                         imp_vol_dict[calls.strike.iloc[j]] = [imp_vol(calls.settle.iloc[
                            j],
353                                                                      fut,
354                                                                      calls.strike.iloc[j],
355                                                                      self.rate_curve.get(self
                                                                     .today) * 0.01,
356                                                                      t / 365,
357                                                                      opt_type='call'), calls.
                                                                     oi.iloc[j]]
358                     vc[expiration] = pd.DataFrame(imp_vol_dict.values(),
359                                                   index= imp_vol_dict.keys(),
360                                                   columns = ['imp_vol','oi'])
361             except:
362                 pass
363         return vc
364
365
366 class CreateVolCurveSample:
367
368     def __init__(self, instrument, today, IMM1_call_prices, IMM1_put_prices,
369                                 IMM2_call_prices, IMM2_put_prices,
```

```
                                ):

        self.today = pd.to_datetime(today)
        self.instrument = instrument
        self.rate_curve = RatesCurve()
        self.calendar = CMECalendar()
        self.opt_prices = {}
        self.fut_prices = {}
        exp_date1 = self.today
        for i, d in enumerate(self.calendar.IMM_dates):
            if exp_date1 < d:
                exp_date1 = d
                break
        exp_date2 = self.calendar.IMM_dates[i + 1]
        self.expiration = {'IMM1_call': exp_date1, 'IMM1_put': exp_date1,
                           'IMM2_call': exp_date2, 'IMM2_put': exp_date2}
        self.parse(IMM1_call_prices, 'IMM1_call')
        self.parse(IMM2_call_prices, 'IMM2_call')
        self.parse(IMM1_put_prices, 'IMM1_put')
        self.parse(IMM2_put_prices, 'IMM2_put')
        self.vol_curve = self.calc_ivols()

    def parse(self, file, exp):
        df = pd.read_csv(file)
        self.opt_prices[exp] = df[['strike', 'settle']]
        self.fut_prices[exp] = df.future[0]

    def calc_ivols(self):
        vc = {}
        for i, exp in enumerate(['IMM1', 'IMM2']):
            vol_dict = {}
            # Calculate option moneyness
            ind_call = (self.fut_prices[exp + '_call'] * 1 < self.opt_prices[exp +
            '_call'].strike
                        ) &  (self.opt_prices[exp + '_call'].strike < self.fut_prices[
                        exp + '_call'] * 1.2)
            ind_put = (self.fut_prices[exp + '_put'] * 0.8 < self.opt_prices[exp +
            '_put'].strike
                       ) & (self.opt_prices[exp + '_put'].strike < self.fut_prices[exp +
                        '_put'] * 1)

            calls = self.opt_prices[exp + '_call'][ind_call]
            puts = self.opt_prices[exp + '_put'][ind_put]

            for j in range(puts.shape[0]):
                xp = exp + '_put'
                t = self.expiration[xp] - self.today
                t = t.days
                vol_dict[puts.strike.iloc[j]] = imp_vol(puts.settle.iloc[j],
                                                        self.fut_prices[xp],
                                                        puts.strike.iloc[j],
                                                        self.rate_curve.get(self.today)
                                                        * 0.01,
                                                        t / 365,
                                                        opt_type='put')
            for j in range(calls.shape[0]):
                xp = exp + '_call'
                t = self.expiration[xp] - self.today
                t = t.days
                vol_dict[calls.strike.iloc[j]] = imp_vol(calls.settle.iloc[j],
                                                        self.fut_prices[xp],
                                                        calls.strike.iloc[j],
                                                        self.rate_curve.get(self.today)
                                                        * 0.01,
                                                        t / 365,
                                                        opt_type='call')
            vc[exp] = pd.DataFrame(vol_dict.values(), index= vol_dict.keys(), columns =
```

```
                ['imp_vol'])
431         return vc
```