Imperial College
London

Market Microstructure
PROBLEM SET 1 - EXERCISE 3

Tuesday 28 January 2020
Johannes Muhle-Karbe, j.muhle-karbe@imperial.ac.uk
Claudio Bellani, c.bellani17@imperial.ac.uk

[1]:
```python
import os
import sys
import pickle
from pathlib import Path
path_problemset = os.path.abspath("./")
path_lobster = os.path.abspath('../lobster/')
os.chdir(path_lobster)
path_data = path_lobster+'/data'
sys.path.append(path_lobster+'/src')
```

[2]:
```python
from produce_data import produce
import numpy as np
import pandas as pd
```

Specify symbol, date, initial, time and final time:

[3]:
```python
symbol='MSFT'
date='2012-06-21'
initial_time=float(9*60*60)
final_time=float(16*60*60)
```

Load or produce data from source:

[4]:
```python
produce_data=False
load_data=True
if produce_data:
    data=produce(
        symbol,date,initial_time,final_time
    )
else:
    if load_data:
        time_window = str('{}-{}'.format(int(initial_time), int(final_time)))
        with open(path_problemset+'/{}_{}_{}_data'.format(symbol, date, time_window), 'rb') as source:
            data=pickle.load(source)
```

Exercise 1.3.1

[5]:
```
mf = data.messagefile
idx = (mf['event_type'].isin([4]))
idx = np.logical_and(idx,np.logical_and((mf['time']<14.5*60*60),(mf['time']>10*60*60)))
trades = mf[idx].copy()
trades = trades.iloc[1:,:].reset_index(drop=True)
lob_trades = data.LOB[idx].copy()
lob_trades = lob_trades.iloc[:-1,:].reset_index(drop=True)
```

Let's take a look at the two pandas dataframe just created.

[6]:
```
trades
```

[6]:
```
      direction  event_type  level  original_idx   price  size          time  \
0             1           4      0         62222  308800  5345  36000.419575
1             1           4      1         62304  308700  2000  36000.446011
2             1           4      0         62312  308700   807  36000.446849
3             1           4      1         62353  308600   700  36000.465200
4             1           4      0         62362  308600  1744  36000.465487
...         ...         ...    ...           ...     ...   ...           ...
4241          1           4      1        482011  303300   100  52189.187548
4242          1           4      1        482014  303300   100  52189.387056
4243          1           4      1        482025  303300  2739  52189.387887
4244          1           4      0        482026  303300   100  52189.387896
4245         -1           4      1        482030  303400   100  52189.387940

[4246 rows x 9 columns]
```

[7]:
```
lob_trades
```

[7]:
```
      ask_price_1  ask_volume_1  bid_price_1  bid_volume_1  ask_price_2  \
0          308900          1855       308800          5345       309000
1          308900          1855       308700          3125       309000
2          308800         10850       308700           807       308900
3          308800         10850       308600          2650       308900
4          308800         14850       308600          1744       308900
...           ...           ...          ...           ...          ...
4241       303400          8358       303300          3639       303500
4242       303400         10965       303300          3139       303500
4243       303400         10965       303300          3039       303500
4244       303400         10965       303300           100       303500
4245       303400         10965       303200         13355       303500

      ask_volume_2  bid_price_2  bid_volume_2  ask_price_3  ask_volume_3  ...  \
0             5380       308700          3125       309100          5560  ...
1             5380       308600          2650       309100          5560  ...
2            15120       308600          2650       309000          7380  ...
3            15066       308500         13750       309000          7380  ...
4            13866       308500         10158       309000          7993  ...
...            ...          ...           ...          ...           ...  ...
4241         19547       303200         15219       303600         19746  ...
4242         19447       303200         13355       303600         21046  ...
4243         19447       303200         13355       303600         21046  ...
4244         19447       303200         13355       303600         21046  ...
4245         19447       303100         19690       303600         21046  ...

      ask_price_9  ask_volume_9  bid_price_9  bid_volume_9  ask_price_10  \
0          309700          4670       308000          4602        309800
1          309700          4670       307900           100        309800
2          309600          3013       307900          1500        309700
3          309600          3013       307800          6416        309700
```

```
4         309600         3013      307800         6416       309700
...          ...          ...         ...          ...          ...
4241      304200         7340      302500        16614       304300
4242      304200         7340      302500        16614       304300
4243      304200         7340      302500        16614       304300
4244      304200         7340      302500        16614       304300
4245      304200         7340      302400        14300       304300

      ask_volume_10  bid_price_10  bid_volume_10  original_idx          time
0              1600        307900            100         62201  36000.418617
1              1600        307800           6616         62222  36000.419575
2              4670        307800           6416         62304  36000.446011
3              4670        307700            460         62312  36000.446849
4              4470        307700            460         62353  36000.465200
...             ...           ...            ...           ...           ...
4241           7700        302400          14300        481978  52188.120529
4242           7700        302400          14300        482011  52189.187548
4243           7700        302400          14300        482014  52189.387056
4244           7700        302400          14300        482025  52189.387887
4245           7700        302300          13600        482026  52189.387896

[4246 rows x 42 columns]
```

[8]:

```
def empirical_spread(lob_trades):
    return np.mean((lob_trades['ask_price_1']-lob_trades['bid_price_1']))
```

[9]:

```
class Roll:
    def __init__(self,trades):
        self.trades = trades;
        self.directions = np.array((-1)*trades['direction'].values, dtype=np.int) #Notice the change of sign
        self.prices = np.array(trades['price'].values, dtype=np.int)
    def covariance_of_price_increments(self,):
        delta_p = np.diff(self.prices)
        dp_for = delta_p[1:]
        m_for = np.mean(dp_for)
        dp_back = delta_p[:-1]
        m_back = np.mean(dp_back)
        N = len(dp_for)
        assert N == len(dp_back)
        cov_deltap = np.dot(dp_for - m_for, dp_back - m_back)/N
        assert cov_deltap <= 0.0
        self.cov_deltap = cov_deltap
    def spread_assuming_balanced_orderflow(self,):
        return 2*np.sqrt(-self.cov_deltap)
    def spread(self,):
        theta = np.sum(self.directions==+1)/len(self.directions)
        assert theta>0.0
        assert theta<1.0
        return np.sqrt(-self.cov_deltap/(theta*(1.0-theta)))
```

[10]:

```
roll = Roll(trades)
roll.covariance_of_price_increments()
S_balance_of = roll.spread_assuming_balanced_orderflow()
S_unbalance_of = roll.spread()
avg_spread = empirical_spread(lob_trades)
```

[11]:

```
print(S_balance_of)
print(S_unbalance_of)
print(avg_spread)
```

```
42.94604071467853
43.03056413872584
129.51012717852097
```

Exercises 1.3.3

[12]:

```
def select_timewindow(mf,t0,t1):
    idx = np.logical_and((mf['time']<t1),(mf['time']>t0))
    return mf[idx].copy().reset_index(drop=True)
```

[13]:

```
def categorise_in_time_slots(timestamps, delta_t):
    assert delta_t > 0
    assert np.all(np.diff(timestamps)>=0.0)
    return np.array((timestamps-timestamps[0])/delta_t, dtype=np.int)
```

[14]:

```
class Amihud:
    def __init__(self,trades):
        self.trades = trades
        self.trades['monetary_vol'] = trades['price']*trades['size']
    def illiquidity_ratio(self,delta_t = 60):
        self.trades['time_slot'] = categorise_in_time_slots(self.trades['time'].
 ↪values, delta_t)
        self.volumes = self.trades.groupby(by='time_slot')['monetary_vol'].sum()
        times = self.trades.groupby(by='time_slot')['time'].min()
        self.times = times
        idx = self.trades['original_idx'].isin(
            list(self.trades.groupby(by='time_slot')['original_idx'].min().values)
        )
        self.prices = self.trades.loc[idx]['price']
        self.returns = np.diff(self.prices)/self.prices[:-1]
        assert len(self.returns) == len(self.volumes) -1
        return  np.mean(np.abs(self.returns.values)/self.volumes.values[:-1])
```

[15]:

```
mf = data.messagefile
idx = (mf['event_type'].isin([4]))
mf = mf[idx].copy()
```

[16]:

```
# Time window 9am-10am
t0=9*60*60
t1=10*60*60
trades = select_timewindow(mf,t0,t1)
amihud_09001000 = Amihud(trades)
ratio_09001000 = amihud_09001000.illiquidity_ratio(delta_t = 60)
print(ratio_09001000)
```

```
1.0296871776495485e-13
```

[17]:

```
# Time window 11.30am-1.30am
t0=11.5*60*60
t1=13.5*60*60
trades = select_timewindow(mf,t0,t1)
amihud_11301330 = Amihud(trades)
ratio_11301330 = amihud_11301330.illiquidity_ratio(delta_t = 60)
print(ratio_11301330)
```

```
1.065959216549388e-13
```

[18]:

```
# Time window 3pm-4pm
t0=15*60*60
t1=16*60*60
trades = select_timewindow(mf,t0,t1)
amihud_15001600 = Amihud(trades)
ratio_15001600 = amihud_15001600.illiquidity_ratio(delta_t = 60)
print(ratio_15001600)
```

```
9.870802066339614e-14
```