MAFS 5270 Assignment 3 Solution Report

ZHANG, Jian

Hong Kong University of Science and Technology

Author Note

This is a report for the solutions of assignment 3.

MAFS 5270 Assignment 3 Solution Report

This report aims to explain the thoughts when I was trying to solve the question 2 to 4, and in this report, not only the solutions but also the ideas, methodologies, data analysis, reasoning and the results will be covered.

Solution to Question 1

The solution to Question 1 does not involve data. So, I will just give the answer here.

Answer to Question 1

$$E(IMB) = E\left[\sum_{i=1}^{N} S_i V_i\right] = E\left[E\left(\sum_{i=1}^{N=n} S_i V_i \,\middle|\, N = n\right)\right] = E[nE(S_i)E(V_i)] = E(N = n)E(S_i)E(V_i)$$

$$= \lambda\beta(2p - 1)$$

$$Var(IMB) = Var\left(\sum_{i=1}^{N} S_i V_i\right) = Var\left(E\left(\sum_{i=1}^{N=n} S_i V_i \,\middle|\, N = n\right)\right) + E\left(Var\left(\sum_{i=1}^{N=n} S_i V_i \,\middle|\, N = n\right)\right)$$

$$= Var\big(nE(S_i)E(V_i)\big) + E\big(Var(S_1V_1 + S_2V_2 + \cdots + S_nV_n)\big)$$

$$= Var(N = n)E(S_i)E(V_i)$$

$$+ E\left(Var(S_1V_1) + Var(S_2V_2) + \cdots + Var(S_nV_n) + \sum_{i \neq j} cov\big(S_iV_i, S_jV_j\big)\right)$$

$$= \lambda\beta(2p - 1) + \lambda\beta^2(1 - (2p - 1)^2) + \beta^2 \sum_{i \neq j} cov\big(S_i, S_j\big)$$

$$= \lambda\beta(2p - 1) + \lambda\beta^2(1 - (2p - 1)^2) + \beta^2 \sum_{i \neq j} \rho^{|i-j|}\,(1 - (2p - 1)^2)$$

■

Solution to Question 2

Question: *Using the trade data of SH600519 (see attached data files), calibrate parameters λ, β, p and ρ. For the moment, you can set τ to be 5 minutes, or 300 seconds. Besides, it would be a good idea to normalize V_i by its first moment or standard deviation when estimating β; otherwise the estimation of β could be close to inf. The first moment V^1 would be a preferable choice here because, theoretically, the second moment of V_i does not always exist. We will also use this V^1 in Part Two.*

In this part, all the details about the solution will be covered.

Estimation of parameters: MLE and Method of Moments

There are mainly two methods to estimate the parameters of a certain distribution: MLE (Maximum Likelihood Estimation) and Method of Moments. We are going to try both methods to estimate parameters and make a comparison between them.

The basis of MLE is to get the likelihood function $\mathcal{L}(\theta; x)$ and then maximize the function. In practice, it is often convenient to work with the natural logarithm of the likelihood function, called the log-likelihood. If the data are independent and identically distributed, then we have:

$$\widehat{\ell}(\theta; x) = \frac{1}{n} \sum_{i=1}^{n} \ln f(x_i \mid \theta)$$

An the for Method of Moments, we use the moments to estimate the parameters. For example, we can use the sample mean to estimate the mean of the total set.

Calibrate λ

Suppose that during time interval $[t, t + \tau)$, the number of trades $N(\tau)$ follows a Poisson process, with the arrival rate of λ:

$$P[N(\tau) = k] = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!}$$

Highlight: And I think this t in $[t, t+\tau)$ should not be time spots appeared in the data, it should be **ANY TIME SPOT** in the whole research dataset, which will make a bias(because in this way, there will be definitely one trade at time t). So, my thought is that I need to pick any spot of time during the period people can trade for at least 5 more minutes and then count the number of trades happened in the next five minutes. To realize this, random sampling time spot is needed.

Maximum Likelihood Estimation

Because $N(\tau)$ is not a function of time t, the likelihood function can be written like this:

$$L_N = \prod_{i=1}^{n} f(x_i|\lambda\tau)$$

$$then, Log\ L_N = \sum_{i=1}^{n} \log\big(f(x_i|\lambda\tau)\big) \quad where\ \tau = 300\ sec$$

In my Python program, I tried two ways to realize it.

Method : Write the function then optimize it.

We first define the function $Log\ L_N$ as logL_poi(lda):

```
def logL_poi(lda):
    L = -np.log(np.sum(np.array([k*np.log(lda)+np.log(np.e)*(-lda)-
np.sum([np.log(x) for x in range(1,k+1)]) for k in k_list])))
    return L
```

It is negative because when we optimize, we use the minimize.

Then we randomly sampled 100,1000,5000,10000 300 seconds' long-time sample, and count the number of trades happened during every 300 sec sample. This operation corresponds with **Line 33** to **Line 57** plus **Line 74** to **Line 83** in my code assignment 3 code.py.

After importing the optimization function `from scipy.optimize import minimize,` we Maximize by `res = minimize(logL_poi,np.array([100,200,300,400,800]),method='nelder-mead').`

Unfortunately, it takes centuries for the PC to calculate the result because of the large amount of data and calculation.

Method of Moments

Then I tried the method of moments. It takes a lot less time and easier calculation.

```
k_list = []
ft = pd.Series(td)
fts = ft.sample(10000)
for i in fts.index:
    date = np.int64(fts[i][:8])
    t_5 = int(time5after.loc[fts[i][-6:]]+'000')
    time = np.int64((fts[i][-6:]+'000'))
    sp = total.loc[(total['date']==date)]
    sp = sp.loc[sp['time']>=time]
    sp = sp.loc[sp['time']<=t_5]
    k = len(sp)
    k_list.append(k)
#Calibrate \lambda using method of moments
print(np.mean(k_list))
```

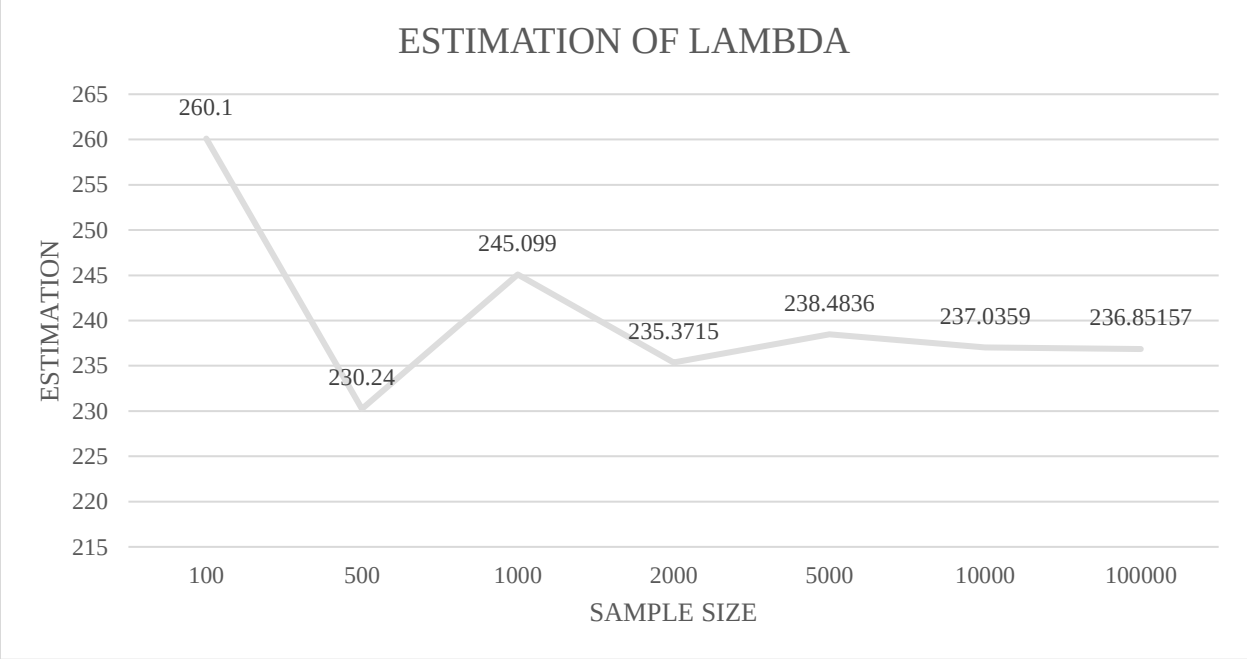
The result of 10000 sample size: 236.6722

The result of other sample sizes:

Sample Size	Method of moments-beta
100	260.1
500	230.24

Sample Size	Method of moments-beta
1000	245.099
2000	235.3715
5000	238.4836
10000	237.0359
100000	236.85157

We can see the result stabilized at 235-237.



Calibrate β

The size Vi (i = 1,2, …) follows an exponential distribution, with the pdf being $f(v, \beta) = \frac{1}{\beta} e^{-\frac{v}{\beta}}$

We also assume that Vi and Vj are uncorrelated, and it is required that we must use all the sample period data. We still have two ways of calibration to try. We use the sample size 100000. But I will give more choice of sample size later.

Maximum Likelihood Estimation of β

Still first we tried to define the function by myself and optimize the function.

```
v_list = list(total.sample(2000)['ntrade'])

def logL_expo(lda):

    L = -np.log(np.prod(np.array([expo(lda,x) for x in v_list])))

    return L

res = minimize(logL_expo,0.5)
```

But it comes out that the function converged to infinity.

```
In [15]: res
Out[15]:
      fun: inf
 hess_inv: array([[1]])
      jac: array([nan])
  message: 'Optimization terminated successfully.'
     nfev: 3
      nit: 0
     njev: 1
   status: 0
  success: True
        x: array([0.5])
```

Then I tried the numerical method to get the result. I found that there is a fit function that solves the exponential function fitting and get the numerical result by MLE.

```
dist2 = st.expon#Define the distribution function

loc, beta = dist2.fit(v_list,floc = 0)#MLE fit
```

Then we get the beta= 352.59121.

Method of Moments of β

We calculate the mean of sample (100000 samples) as the estimation of beta.

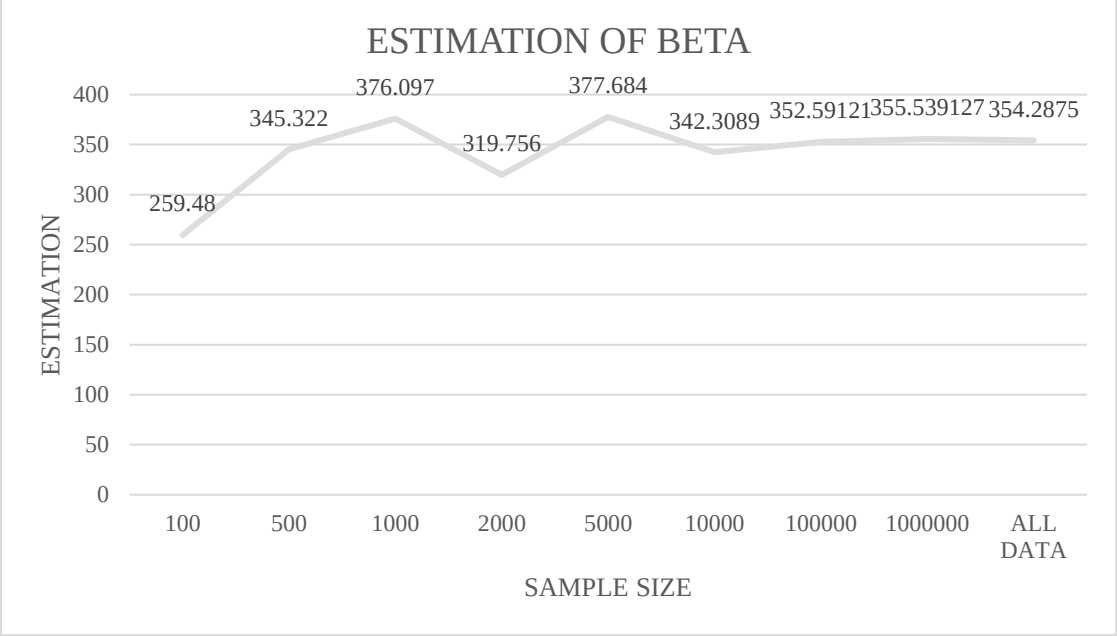
```
np.mean(v_list)
```

And we get the same result beta= 352.59121. It corresponds with we got in MLE method.

We made a summary about the results of different sizes of samples:

Sample Size	MLE of beta	Method of Moments for beta
100	259.48	259.48
500	345.322	345.322
1000	376.097	376.097
2000	319.756	319.756
5000	377.684	377.684
10000	342.3089	342.3089
100000	352.59121	352.59121
1000000	355.539127	355.539127
ALL DATA	354.2875	354.2875

As we can see, after the size of sample increase to more than 100000, the estimation is more and more stable round 354. So, we estimate the beta is around 354-355.



Calibrate ρ and p for S_i

The definition of ρ is: $Corr(S_i, S_j) = \rho^{|i-j|}$, in other words, it is the autocorrelation of S. Then we remove the empty spots, replace 'B' with 1 and 'S' with -1, use pandas.Series.autocorrelation to get the correlations with the lag = 1, which means that |i-j| =1.

$$\mathrm{Corr}(S_i, S_j) = \rho \quad \text{in this situation}$$

```
#Calculate autocorrelation rho and p

bs_series = total['BS'].loc[total['BS']!=' ']

bs_series = bs_series.replace('B',1)

bs_series = bs_series.replace('S',-1)

rho = bs_series.autocorr()
```

The result is $\rho = 0.5613843488150271$ and for p, we just use the sample mean to do it.

```
p = (np.mean(bs_series)+1)/2
```

Then we can get the estimation of result: P = 0.49197099128803423 with all the dataset.

Solution to Question 3

Question: ...for both SH600519 and SH601398, what is the accuracy of this simple version of the Lee and Ready algorithm, using the exchange-provided information in the column "BS" as a benchmark. Please use as large sample size as possible from the attached files.

So, what we need to do is to calculate the sign of each trade by "Lee and Ready Algorithm". Then whole process will be as followed.

Solution

SH600519 has the same method of solution as SH601398

In this part, we are going to use the **ALL THE DATA** available as samples. But because some data do not have price, bid or ask, before the calculation, we need to remove those points.

```
total_quote = total_quote.loc[total_quote['price']!=0]
```

First then, we must get the mid quote for each of the trades. Because in this part, it is required to use the simple version of the mid quote which is the simple average of bid1 and ask1. The data required is all available directly in the quote dataset. The code works like this:

```
total_quote = total_quote.loc[total_quote['price']!=0]

total_quote['mid_quote'] = (total_quote['BidPrice1']+total_quote['AskPrice1'])/2

q = total_quote.loc[total_quote['mid_quote']>total_quote['price']]
```

```
q = q.loc[q['BS']=='S']
uo = total_quote.loc[total_quote['mid_quote']<total_quote['price']]
uo = uo.loc[uo['BS']=='B']
te = total_quote.loc[total_quote['mid_quote']==total_quote['price']]
te = te.loc[te['BS']==' ']
qt = pd.concat([q,uo])
acc_519 = len(qt)/len(total_quote)
```

We handle the data in dataframe which is a good fit for calculating and screening the columns. **`Q, uo ,`**and **`te`** count the numbers of buying, selling and unknown.

The result of SH600519 is **52.39455% accuracy**.

The same way, the accuracy of SH601398 is **59.27466%.**

Solution to Question 4

Question: ... For both SH600519 and SH601398, calibrate the four parameters: $\beta+$, $\beta-$, $\gamma+$ and $\gamma-$ in formulas (10) for two cases: (1) τ = 5 minutes, or 300 seconds; (2) τ = 1 minutes, or 60 seconds. Check if there are significant differences between the two cases and comment on them, if any.

We plan to sample the data and calibrate the parameters by taking the log and use least squares to get the estimation of beta and gamma. Due to the long time of calculation we just use 10000 as our sample size. 10000 sample size is good enough to get the relatively precise result estimation according to the results of lambda and beta.

Processing the data

After importing the data to the storage, the work is to process the data. First, just like question 2, we sample a fixed number of samples by 5 min long sample or 1 min long sample. We take 5 min as an example. The code correspond to this sampling by time method is from **`line 33 to line 66 and line 198 to line 220.`** The code will be available in my submissions.

And for each sample, which is an either 5- or 1-minute long time period with all the trades data happened in this certain period. The structure is like this:

date	**time**	**price**	**volume**	**…**	**AskPrice1**	**…**	**BidPrice1**
…	…	…	…	…	…	…	…

Then by the formulas:

$$Ret = \frac{MidQuote(t+\tau) - MidQuote(t)}{MidQuote(t)}.$$

$$MidQuote(t) = \frac{bid1 * askSize1 + ask1 * bidSize1}{askSize1 + bidSize1}.$$

$$IMB = \sum_{i=1}^{N} S_i \frac{V_i}{\bar{V}}.$$

We calculate the results by each samples' data.

```
for i in fts2.index:
    date = np.int64(fts2[i][:8])
    t_5 = int(time5after.loc[fts2[i][-6:]])
    time = np.int64((fts2[i][-6:]))
    sp = total_quote1.loc[(total_quote1['date']==date)]
    sp = sp.loc[sp['time']>=time]
    sp = sp.loc[sp['time']<=t_5]
    Ret = (float(sp[-1:]['mid_quote'])-
float(sp[:1]['mid_quote']))/float(sp[:1]['mid_quote'])
    IMB = sum(sp['BS']*sp['volume'])/V_bar1
    #Separate the outcomes into two groups by the sign of IMB
    if IMB > 0:
        IMB_list1.append(IMB)
        Ret_list1.append(Ret)
```

```
    if IMB < 0:
        IMB_list2.append(IMB)
        Ret_list2.append(Ret)
```

Then we use these sample points to fit the parameters. The original ones are not linear, so it may be not very easy to fit the parameters directly. But due to the exponential form of the formula, we can take the log and this problem is transformed into easier linear regression problem.

$$\log Ret = \log \beta + \gamma \log \sigma_{Ret} I_{IMB} |IMB|$$

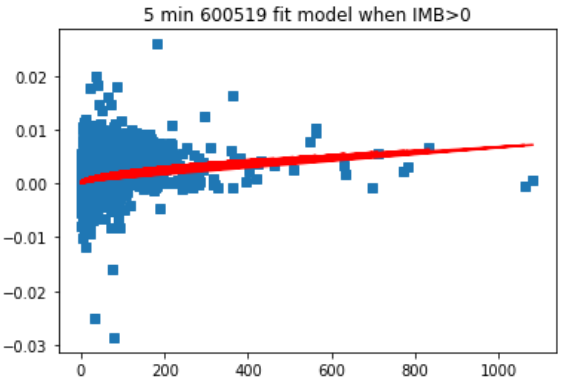
But when we start to take the log, we found that it is not applicable because the sign of Ret is not always positive, hence, there will be nan if we take the log. We must find another way out.

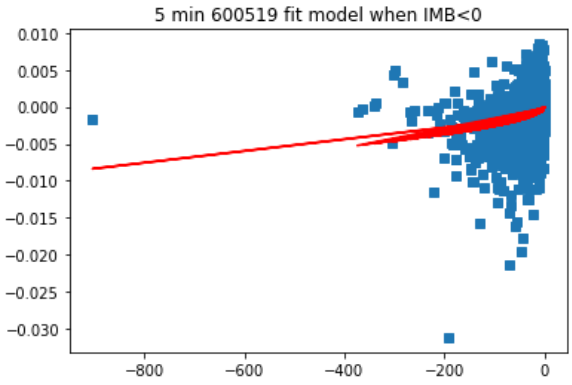
Then I found scipy.optimize.curve_fit, with can fit the model by using the function you defined. So, we just define the function like the formula in the question.

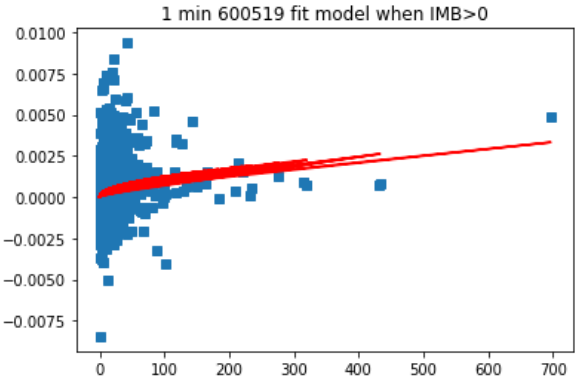
```
def func(x,beta,gam):
    f = np.std(x)*beta*np.abs(x)**gam
    return f
```

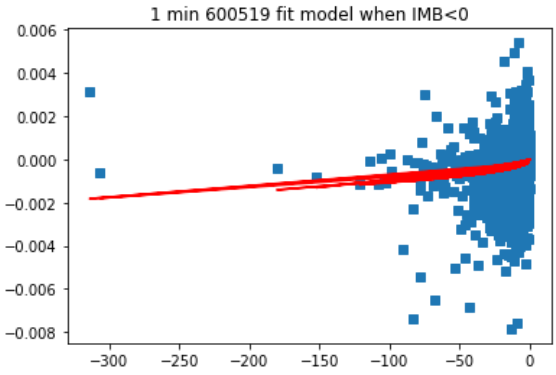
Then with the help of scipy.optimize.curve_fit, we can get the beta and gamma we need. We also plot a comparison figure to show the goodness of fit for each estimation.

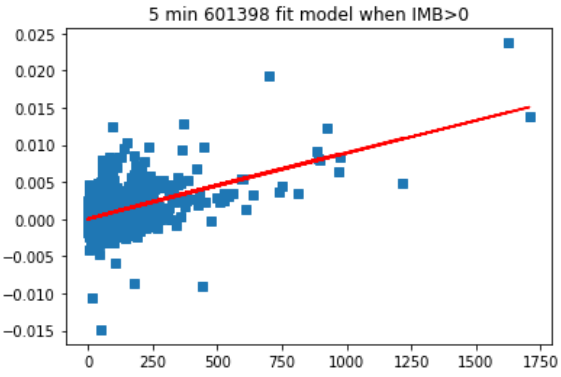
```
popt, pcov = curve_fit(func, x, y)
a = popt[0]
b = popt[1]
yvals = func(x,a,b)
plot1 = plt.plot(x, y, 's',label='original values')
plot2 = plt.plot(x, yvals, 'r',label='polyfit values')
plt.title('5 min 600519 fit model when IMB>0')
plt.show()
```

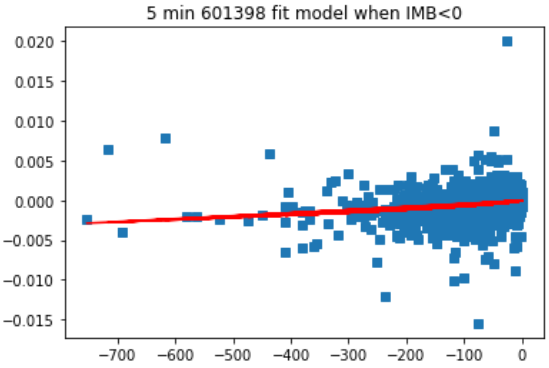

5 min 600519 fit model when IMB>0

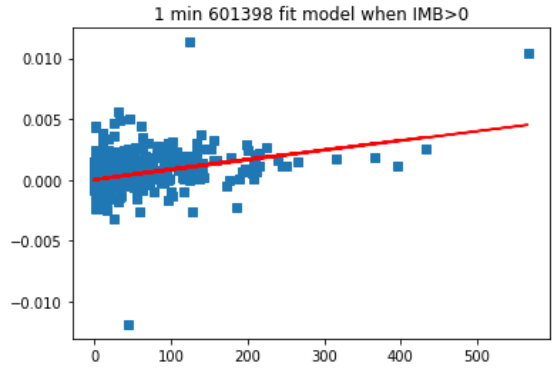

5 min 600519 fit model when IMB<0

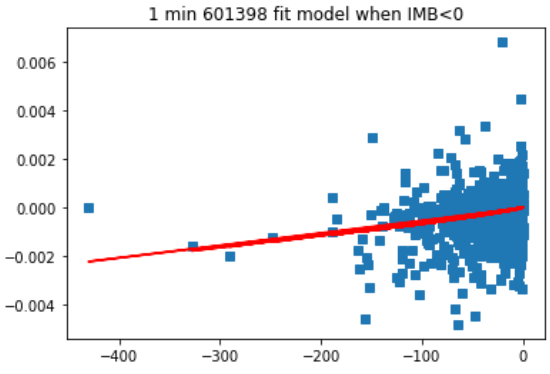

1 min 600519 fit model when IMB>0


1 min 600519 fit model when IMB<0


5 min 601398 fit model when IMB>0


5 min 601398 fit model when IMB<0


1 min 601398 fit model when IMB>0


1 min 601398 fit model when IMB<0

Then we the help of this method we get the estimations of the parameters.

INFO	BETA	GAMMA
5 MIN 600519 IMB>0	4.0014115971239895e-06	0.5528881013779816
5 MIN 600519 IMB<0	-5.6070054336630874e-06	0.5478924575995585
1 MIN 600519 IMB>0	5.857199804370141e-06	0.45608040310934633
1 MIN 600519 IMB<0	-1.040113160254373e-05	0.5015851735595837
5 MIN 601398 IMB>0	1.1918162913035402e-07	0.9708553348012654
5 MIN 601398 IMB<0	-3.0538325234611586e-07	0.8589007062784484
1 MIN 601398 IMB>0	5.934449731560963e-07	0.8047083075190999
1 MIN 601398 IMB<0	-6.320054044351046e-07	0.884577777882431

For the parameters, we can see there are truly significant differences between the two cases and comment on them:

SH600519 have MUCH LARGER absolute values of β than SH601398, which means that **SH600519 is have much larger intraday variation**. The period change of Ret is larger.

SH600519 have smaller γ than SH601398, and all the γs are positive. **It means the same IMB will have a larger impact on SH600519 if |IBM|<1, and the same IMB will have a larger impact on SH601398 if |IBM|>1**

Some Comments

In this report, both the results and the procedures of solutions are covered. The report only referred 60% of my code to make an illustration. To understand my solutions better, please check the program “assignment 3 code.py” which is also attached in my submission. In the sampling process, only one number of sizes is in the code for the reason of simplification, when I was getting the result for sample size 100, 500 , … , 10000, I just change the number directly in the code.