S3Q1

November 1, 2020

[3]:
```
import numpy as np
import pandas as pd
```

[4]:
```
df_vols = pd.read_csv('EURUSD_volsurface.csv')
```

[5]:
```
df_vols.head() #shape of df_vols = (24,6)
```

[5]:

	TENOR	ATM	25D Call EUR	25D Put EUR	10D Call EUR	10D Put EUR
0	1D	8.717	9.054	8.651	9.457	8.742
1	1W	6.922	7.321	6.814	7.820	6.905
2	2W	8.477	8.889	8.406	9.415	8.570
3	3W	8.090	8.485	8.050	9.004	8.221
4	1M	7.855	8.258	7.808	8.771	7.984

[6]:
```
df_vols.tail()
```

[6]:

	TENOR	ATM	25D Call EUR	25D Put EUR	10D Call EUR	10D Put EUR
19	10Y	8.415	8.785	8.765	9.610	9.640
20	15Y	8.417	8.815	8.717	9.662	9.579
21	20Y	8.419	8.845	8.692	9.705	9.528
22	25Y	8.422	8.871	8.671	9.743	9.485
23	30Y	8.425	8.896	8.653	9.777	9.446

0.1 Q(1) part(b)

[7]:
```
def countDaysFromTenor(arr_of_tenors):
    l = len(arr_of_tenors)
    day_arr = np.array([None]*l)
    key = {'D':1,'W':7,'M':30,'Y':365}
    for i in range(l):
        tenor = str(arr_of_tenors[i])
        day = float(tenor[:-1])*key[tenor[-1]]
        day_arr[i]=day
    day_arr = day_arr.astype(np.float)
    return day_arr
```

Tests of countDaysFromTenor

[8]:
```python
countDaysFromTenor(['1W','2W','4M','2Y'])
```

[8]:
```
array([  7.,  14., 120., 730.])
```

0.2 Q1 part(c)

[9]:
```python
import scipy.stats as stats
def strikeFromDelta(delta,S,sigma,T,cp):
    d1 = (1/cp)*stats.norm.ppf(delta/cp,0,1)
    K = S/np.exp(sigma*np.sqrt(T)*d1-sigma*sigma*T/2)
    return K
```

0.3 Q1part(d)

[10]:
```python
T_vec = countDaysFromTenor(df_vols['TENOR'].values)/365
S_vec = np.array([1.166]*len(df_vols))
keys = {'25D Call EUR':[0.25,1],'25D Put EUR':[-0.25,-1],'10D Call EUR':[0.
 ↪1,1],'10D Put EUR':[-0.1,-1]}
K_array = np.concatenate((S_vec[:,np.newaxis],np.zeros((len(df_vols),4))),axis=1)
for i in range(4):
    sigma_vec = (df_vols.iloc[:,i+2].values)/100
    column = df_vols
    (delta,cp) = keys[(df_vols.columns)[i+2]]
    delta_vec = np.array([delta]*len(df_vols))
    cp_vec = np.array([cp]*len(df_vols))
    K_vec = strikeFromDelta(delta_vec,S_vec,sigma_vec,T_vec,cp_vec)
    K_array[:,i+1]=K_vec
df_K =pd.DataFrame(np.concatenate((T_vec[:,np.newaxis],K_array),axis=1),columns␣
 ↪= (['Maturity in year']+list((df_vols.columns)[1:])))
```

[11]:
```python
df_K
```

[11]:
```
    Maturity in year    ATM  25D Call EUR  25D Put EUR  10D Call EUR  \
0           0.002740  1.166      1.169746     1.162456      1.173435
1           0.019178  1.166      1.174061     1.158654      1.182365
2           0.038356  1.166      1.179951     1.153281      1.194084
3           0.057534  1.166      1.182361     1.151127      1.199003
4           0.082192  1.166      1.185101     1.148815      1.204568
5           0.164384  1.166      1.191751     1.143245      1.218848
6           0.246575  1.166      1.196371     1.139608      1.229511
7           0.328767  1.166      1.201051     1.135892      1.240136
8           0.410959  1.166      1.205339     1.132592      1.250638
9           0.493151  1.166      1.209361     1.129596      1.260181
10          0.739726  1.166      1.220441     1.121678      1.286857
11          1.000000  1.166      1.230546     1.114915      1.312329
12          1.479452  1.166      1.247330     1.102800      1.347483
```

13	2.000000	1.166	1.262956	1.092456	1.383084
14	3.000000	1.166	1.292360	1.074778	1.446023
15	4.000000	1.166	1.319927	1.060160	1.505461
16	5.000000	1.166	1.344602	1.047894	1.558731
17	6.000000	1.166	1.368244	1.037527	1.604181
18	7.000000	1.166	1.390842	1.028559	1.647907
19	10.000000	1.166	1.461619	1.005054	1.802568
20	15.000000	1.166	1.556019	0.983002	2.020129
21	20.000000	1.166	1.646439	0.967484	2.234430
22	25.000000	1.166	1.735191	0.956128	2.451119
23	30.000000	1.166	1.823824	0.947641	2.673117

	10D Put EUR
0	1.159195
1	1.151851
2	1.141348
3	1.137124
4	1.132590
5	1.121067
6	1.113280
7	1.105190
8	1.097635
9	1.090806
10	1.073071
11	1.057469
12	1.032085
13	1.009525
14	0.972440
15	0.941747
16	0.915598
17	0.895037
18	0.877204
19	0.826439
20	0.776429
21	0.739550
22	0.710533
23	0.686858

0.4 Q1part(e)

[12]:
```
S=1.166
m_array = K_array/S
df_m =pd.DataFrame(np.concatenate((T_vec[:,np.newaxis],m_array),axis=1),columns
 ↪= df_K.columns)
```

[13]:
```
df_m
```

[13]:

	Maturity in year	ATM	25D Call EUR	25D Put EUR	10D Call EUR	\
0	0.002740	1.0	1.003213	0.996961	1.006376	
1	0.019178	1.0	1.006913	0.993700	1.014035	
2	0.038356	1.0	1.011965	0.989091	1.024086	
3	0.057534	1.0	1.014032	0.987245	1.028304	
4	0.082192	1.0	1.016381	0.985262	1.033077	
5	0.164384	1.0	1.022085	0.980485	1.045324	
6	0.246575	1.0	1.026048	0.977365	1.054469	
7	0.328767	1.0	1.030061	0.974178	1.063582	
8	0.410959	1.0	1.033739	0.971348	1.072589	
9	0.493151	1.0	1.037188	0.968778	1.080772	
10	0.739726	1.0	1.046690	0.961988	1.103651	
11	1.000000	1.0	1.055357	0.956188	1.125496	
12	1.479452	1.0	1.069751	0.945798	1.155645	
13	2.000000	1.0	1.083153	0.936926	1.186178	
14	3.000000	1.0	1.108371	0.921765	1.240157	
15	4.000000	1.0	1.132013	0.909228	1.291133	
16	5.000000	1.0	1.153175	0.898708	1.336819	
17	6.000000	1.0	1.173451	0.889818	1.375798	
18	7.000000	1.0	1.192832	0.882126	1.413299	
19	10.000000	1.0	1.253532	0.861967	1.545941	
20	15.000000	1.0	1.334493	0.843055	1.732529	
21	20.000000	1.0	1.412040	0.829746	1.916321	
22	25.000000	1.0	1.488157	0.820007	2.102160	
23	30.000000	1.0	1.564171	0.812728	2.292553	

	10D Put EUR
0	0.994163
1	0.987865
2	0.978858
3	0.975235
4	0.971347
5	0.961464
6	0.954786
7	0.947847
8	0.941368
9	0.935511
10	0.920301
11	0.906920
12	0.885150
13	0.865802
14	0.833996
15	0.807673
16	0.785247
17	0.767613
18	0.752319
19	0.708782

```
20      0.665891
21      0.634262
22      0.609376
23      0.589072
```

0.5 Q1 part(f)

[14]:
```
from math import *
def g(x,y,h1=0.05,h2=0.05):
    return np.exp(-x*x/(2*h1))*np.exp(-y*y/(2*h2))/(2*pi)
```

[15]:
```
def volEstimate(m_i_array, T_i_vec, v_arr,g,m,T):
    N = len(m_i_array)*(m_i_array.shape)[1]
    m_mat = np.matmul(m.reshape(len(m),1),np.ones((1,N)))#m_mat.shape = m_size *␣
↪N
    T_mat = np.matmul(T.reshape(len(m),1),np.ones((1,N)))#T_mat.shape = m_size *␣
↪N (len(T)=len(m))


    m_i_mat = np.matmul(np.ones((len(m),1)),m_i_array.reshape((1,N))) #m_i_array.
↪shape = m_size * N

    v_array = np.matmul(np.ones((len(m),1)),v_arr.reshape((1,N))) #m_i_array.
↪shape = m_size * N

    T_i_array = np.matmul(T_i_vec.reshape(len(T_i_vec),1), np.ones((1,(m_i_array.
↪shape)[1])))
    T_i_array = T_i_array.reshape((1,N))
    T_i_array = np.matmul(np.ones((len(m),1)),T_i_array.reshape((1,N)))


    g_mat = g(m_mat-m_i_mat,T_mat-T_i_array)
    sum_g = np.sum(g_mat, axis = 1)
    sum_v_g = np.sum(v_array*g_mat,axis=1)
    vol_estimator = sum_v_g/sum_g
    return vol_estimator
```

[16]:
```
m_min = np.min(m_array)
m_max = np.max(m_array)
t_min = np.min(T_vec)
t_max = np.max(T_vec)
v_mat = (df_vols.values[:,1:]).astype(np.float)
mx = np.linspace(m_min,m_max,10)
ty = np.linspace(t_min,t_max,10)
v_estimator = volEstimate(m_array,T_vec,v_mat,g,mx,ty)
```

[17]:
```
v_estimator.reshape((len(v_estimator),))
```

[17]:
```
array([8.0102018 , 8.09454094, 8.524078  , 8.78449079, 8.85834566,
       9.2055899 , 9.39734453, 9.57825579, 9.72360674, 9.77264764])
```

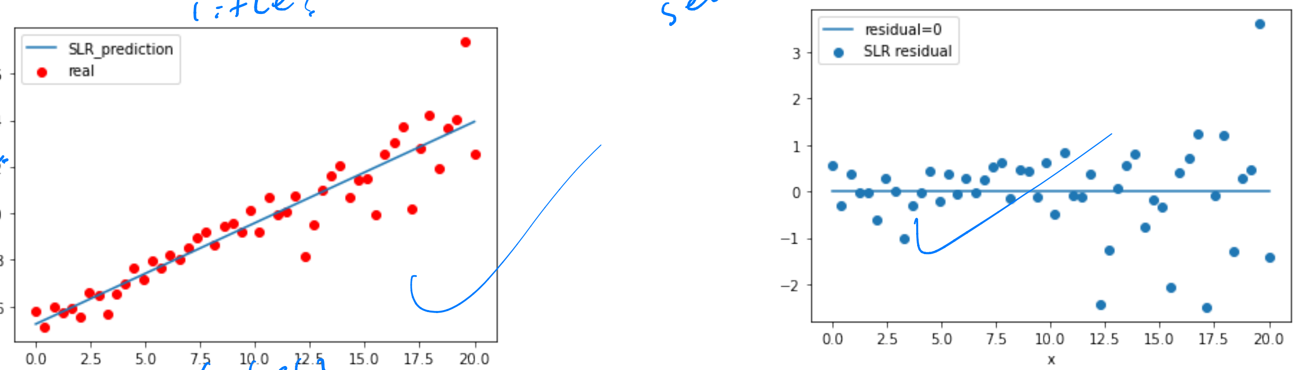
SLR?

An introductory sentence would be great.

1 part(a)

Title?

Label?



Label?

(a) fitted value from SLR against the original data (b) SLR residuals against the predictors

Figure 1: plots of fitted value and residuals from SLR

From figure 1 (b), we can observe that residuals deviate from the zero line more when x is larger than (approx.) 11, indicating that the standard deviations of residuals are not consistent all the time.

2 part(b)

We choose our weights in WLR to be the inverse of df_WLS['weights']^2.

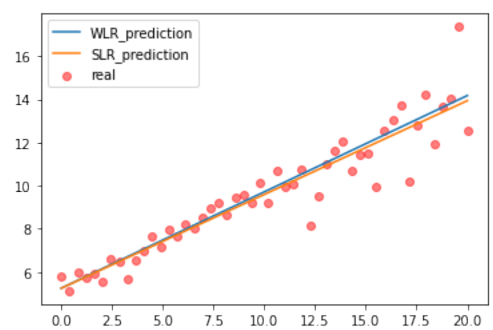


Figure 2: fitted values from SLR and WLR against the original data

	const-coeff	const-std err	x1-coeff	x1-std err
WLR	5.2469	0.143	0.4466	0.018
SLR	5.2426	0.271	0.4349	0.023

Table 1: estimated coefficients from WLS and from OLS and their respective deviations

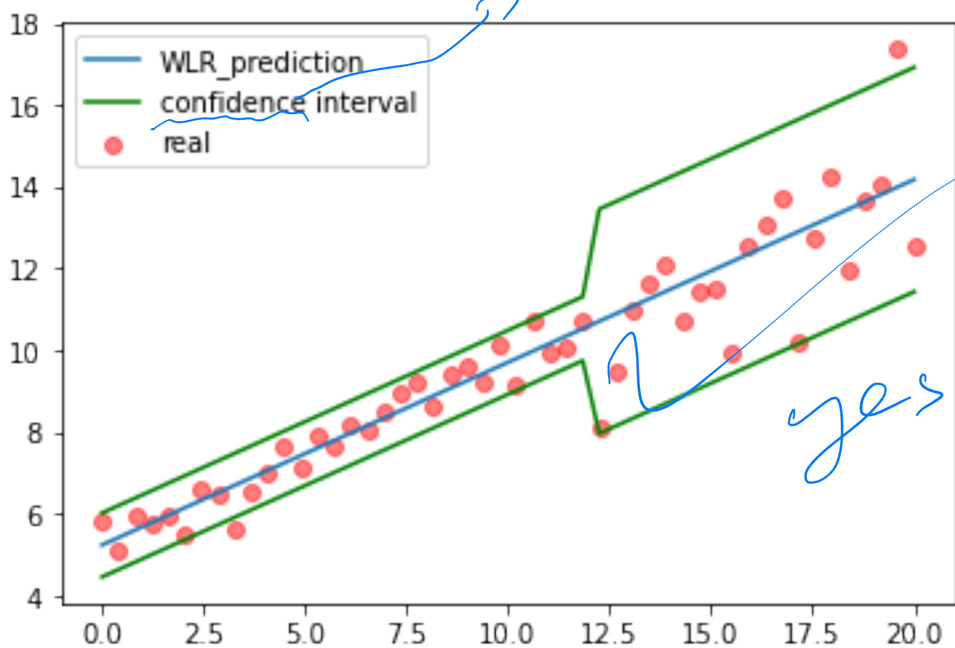
We can observe that the standard error gets smaller after introducing the weights, indicating better model.

3 part(c)

Suggested by the df_WLS['weights'] columns, the standard deviation of residuals changes after the 30th data point, so we calculate the standard deviations of residual[:30] and residual[30:] respectively, and our 95% confidence prediction interval = fitted values ± 1.96 × standard deviation of residual (different std values depending on whether the data point is in the first 30 or not).

From figure 3, we observe that except two points (one is around x = 3.5, the other is around x = 19), all points lie inside the predicted interval.

prediction interval?



yes, good

Figure 3: Plots of predicted interval, fitted values and original data