經濟預測期末報告_第一組

各國對台匯率分析

指導老師：陳宏銘、林雪瑜

學生：S07240003 鄭竣隆 S07240018 溫宏岳
S07240013 簡睿德 S07240047 陳昱廷
S07240058 王詠慈 G10240005 林咨吟
S08240012 朱銘浩

中華民國 111 年 6 月 25 日

目錄

1. 研究動機

本文主旨是各國與台灣之匯率預測，我們會先介紹經濟預測之目的，接著介紹本文使用之預測方法，有最小平方法 OLS(Ordinary least squares)，自我迴歸條件異質變異數模型 ARCH(Autoregressive conditional heteroskedasticity)及廣義自回歸條件異質變異數模型 GARCH(Generalized Auto Regressive Conditional Heteroskedasticity)，最後簡介使用程式之內容，再比較三種預測方法的輸出結果，挑選出最準確的預測方式。

2. 研究目的

經濟預測是預測的一個分支，是指以準確的調查統計資料和經濟信息為依據，從經濟現象的歷史、現狀和規律性出發，運用科學的方法，經過對濟活動的各個方面情況的調查，獲得了大量的數據、資料和信息，透由整理、分析和研究，來揭示經濟現象的發展規律及各類經濟現象之間的相互聯繫，指出經濟現象未來發展趨勢和可能達到的水平。

開放經濟體，匯率影響甚鉅，匯率是國際貿易中最重要的調節杠桿，因為一個國家生產的商品都是按本國貨幣來計算成本的，要拿到國際市場上競爭，其商品成本一定會與匯率相關。

本篇文章目的為匯率預測，由於匯率對於國家適合進口或出口的影響佔其中一部分，當台幣升值時適合進口，台幣貶值時適合出口，所以我們挑選台灣於 110 年的前三大出口國與前四大進口國中的中日美，三國對於台灣之匯率。

3. 研究方法

3.1. OLS(Ordinary least squares)

本節將簡單介紹最小平方法 OLS(Ordinary least squares)，OLS 是一個最佳線性無偏估計，主要用於線性回歸的參數估計，希望找到一條預測的回歸線，使觀測值與預測值之間的誤差平方最小，也就是說 OLS 迴歸的目標就是最小化殘差，因為殘差容易分析而且計算。OLS 會假設誤差項的變異數是不變的(同質變異數)。

最簡單的線性式為$y_i = \beta_0 + \beta_1 x_i$，殘差為$\widehat{u_i} = y_i - \widehat{y_i}$，但這會使所有的$y_i$有一樣的權重。所以改用殘差的平方和才會給離迴歸線越遠的y_i比較大的權重，這樣才能反映離差的大小，殘差平方的總和(RSS)越小越好，

$RSS = \sum_{i=1}^{n} \widehat{u_i}^2 = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$，$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$，$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$。

在程式裡我們使用三種方式估計，固定視窗法，遞增視窗法以及滾動視窗法，其中固定視窗法是估計一次之未來的資料都用相同的估計參數運算。遞增視窗法是從第一筆開始估計 每次都增加一筆，估計完全部的資料。滾動視窗法是從第一筆開始估計，每次都固定估計資料的長度，估計完全部的資料。接下來我們將介紹自我迴歸條件異質變異數模型 ARCH(Autoregressive conditional heteroskedasticity)及其延伸廣義自回歸條件異質變異數模型 GARCH(Generalized AutoRegressive Conditional Heteroskedasticity)。

3.2. ARCH(Autoregressive conditional heteroskedasticity)

ARCH 是由 Engle 於 1982 年在《計量經濟學》中提出，被廣泛應用於金融理論中的規律描述與金融市場的預測和決策，ARCH 解決了傳統計量經濟學的其中一個假設，即便異數是固定的，傳統計量經濟學的另一個假設則是平均值是固定的，ARCH 模型的主要概念就是，如果波動容易有群聚現象，那就令波動的條件變異數與前期波動的平方有正相關，ARCH(q)模型性質為尾厚與波動群聚性。

ARCH(q)模型，考慮報酬率(r_t)模型為$r_t = \mu_t + \varepsilon_t$，$\mu_t$為$r_t$的條件期望值，$\varepsilon_t$為 t 期的干擾項，Engle 教授將$\varepsilon_t$分成兩項相乘如下

$\varepsilon_t = \sigma_t \epsilon_t$， $\epsilon_t \sim iid\ N(0,1)$(期望值為 0，變異數為 1 的常態分佈)，$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2$(其中$\alpha_0 > 0, \alpha_i \geq 0, i > 0$)，有兩種方法可用來估計 ARCH 模型，一種為 OLS 另一種為最大概似估計(MLE)，其中以 MLE 為最佳。由於 ARCH(P)模型落後期數(P)太多時，會產生太多參數需要估計，但因為樣本數有限，所以會造成估計精密度降低的麻煩，在線性 AR 模型落後期數太長我們可使用 ARMA 模型，此概念可用於 ARCH 模型，也就是由 Bollerslev 於 1986 年所提出的 GARCH，以下我們將介紹 GARCH 模型。

3.3. GARCH(Generalized Auto regressive Conditional Heteroskedasticity)

GARCH(p,q)模型，$\varepsilon_t = \sigma_t \epsilon_t$，$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_i \sigma_{t-i}^2$，為符合弱穩

定性質，需限制 $0 \leq \alpha_1, \beta_1 \leq 1, \alpha_1 + \beta_1 < 1$，GARCH 模型也可導出報酬率分配具厚尾及波動群聚性的性質，估計仍是 MLE，而預測數夠大時 GARCH(1,1) 預測值會收斂到 Var(ε_t)

4. 研究結果與分析

這個章節分為三大部分，依序為資料前處裡、模型行分析、分析結果比較，模型行分析部份會先以台灣對美國的匯率資料來訓練 OLS、ARCH、GRACH 模型，再用固定視窗法、遞增視窗法、滾動視窗法去進行預測。固定視窗法: 估計一次之未來的資料都用相同的估計參數運算、遞增視窗法:從第一筆開始估計 每次都增加一筆，估計完全部的資料、滾動視窗法: 從第一筆開始估計 每次都固定估計資料的長度，估計完全部的資料，至於分析結果的部份會針對台灣對美國、大陸、日本做成表格來說明。

4.1. 資料前處理

此次報告的台美匯率數據是由 FRED[10]網站上下載 2021-01-04~2022-06-10(共 375 天)的匯率資料，而台灣對大陸的匯率及台灣對日本的匯率則是從台灣銀行 [11]下載 2021-01-04~2022-06-17(共 360 天)的匯率資料，台灣銀行有提供不同類型的匯率，例如:現金匯率、即期匯率，這裡我們挑選的是即期匯率，然而資料下載下來有幾天的匯率值是#N/A，因此我們用全部匯率的平均來填補這個#N/A(如圖一)

```
#取平均值
df.fillna(value = {'DEXTAUS': df['DEXTAUS'].mean()}, inplace = True )
df.head()
```

圖一

後續模型分析的部分，OLS 是利用前面兩百多筆資料去訓練模型，再利用最後一百筆資料來測試訓練的情況，而 ARCH、GARCH 則是用全部的資料去訓練並測試。

4.2. 模型行分析

利用 OLS、ARCH、GRACH 三種模型來訓練台灣對美國的匯率、台灣對大陸的匯率、台灣對日本的匯率。

4.2.1. OLS

利用 statsmodels.api 提供的 OLS 指令來訓練前面整理好的資料(如圖二)

```
#估計模型參數
fixwinmodel  =  sm.OLS(np.array(fixwinY),  np.array(fixwinX)).fit()
```

圖二

訓練完之後分別帶入固定視窗法、遞增視窗(如圖三)、滾動視窗(如圖四)三種預測方法

```
for  i  in  range(xsizem-recwindowsN):
        if  i==0:
                #print(i)
                #設定X的資料開始與結束的範圍
                recwinX=X[0:recwindowsN+i]
                print(len(recwinX))
                #設定Y的資料開始與結束的範圍，要落後一期，才能預測。
                recwinY=Y[1:recwindowsN+1+i]['DEXTAUS']
                #估計OLS
                recwinmodel  =  sm.OLS(np.array(recwinY),  np.array(recwinX)).fit()
                #計算預測與估計的值
                predrecwinY=np.sum(recwinX*recwinmodel.params,1)
                #將資料轉為np  array  方便計算
                predRecarray=np.array(predrecwinY)
        else:
                #print(i)
                #設定X的資料開始與結束的範圍
                recwinX=X[0:recwindowsN+i]
                #print(len(recwinX))
                #設定Y的資料開始與結束的範圍，要落後一期，才能預測。
                recwinY=Y[1:recwindowsN+1+i]['DEXTAUS']
                #估計OLS
                recwinmodel  =  sm.OLS(np.array(recwinY),  np.array(recwinX)).fit()
                predrecwinY=np.sum(recwinX*recwinmodel.params,1)
                #每執行一次，就將資料加入到predRecarray的資料中
                predRecarray=np.append(predRecarray,predrecwinY[-1:])
```

圖三

```
rollingN=10
for  i  in  range(xsizem-rollingN):
       if  i==0:
              #print(i)
              rollingwinX=X[0+i:rollingN+i]
              #print(len(rollingwinX))
              rollingwinY=Y[1+i:rollingN+1+i]['DEXTAUS']
              #估計OLS
              rollingwinmodel  =  sm.OLS(np.array(rollingwinY),  np.array(rollingwinX)).fit()
              #計算預測與估計的值
              predrollingwinY=np.sum(rollingwinX*rollingwinmodel.params,1)
              #將資料轉為np  array  方便計算
              predRollingarray=np.array(predrollingwinY)
       else:
              #print(i)
              rollingwinX=X[0+i:rollingN+i]
              #print(len(rollingwinX))
              rollingwinY=Y[1+i:rollingN+1+i]['DEXTAUS']
              #估計OLS
              rollingwinmodel  =  sm.OLS(np.array(rollingwinY),  np.array(rollingwinX)).fit()

              predrollingwinY=np.sum(rollingwinX*rollingwinmodel.params,1)
              #每執行一次，就將資料加入到predRollingarray的資料中
              predRollingarray=np.append(predRollingarray,predrollingwinY[-1:])
```

圖四

為了衡量模型預測值，因此算三種不同方法的 RMSE(如圖五)來檢視模型訓練結果以及用曲線圖的方式(如圖六)看不同方法預測的結果。就 RMSE 和曲線圖來說都是滾動視窗法的效果最好，可見滾動視窗法對 OLS 來說是一個可以預測比較準確的方法。

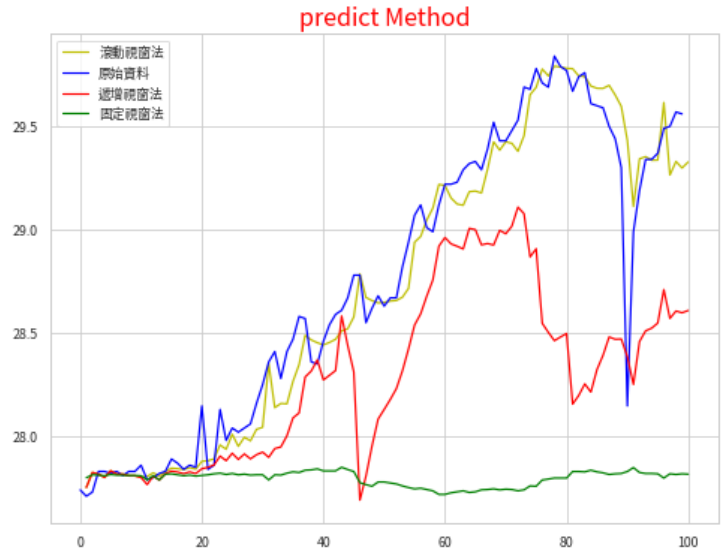
```
realY=Y[1:]['DEXTAUS']
#三種預測方法  RMSE的比較
#第一種方法  固定視窗法  RMSE
rmsefix=np.sqrt(sum((predFixarray[0:-1]-realY)**2))
#第二種方法  遞增視窗法  RMSE
rmserec=np.sqrt(sum((predRecarray[0:-1]-realY)**2))
#第三種方法  滾動視窗法  RMSE
rmserolling=np.sqrt(sum((predRollingarray[0:-1]-realY)**2))
print("第一種固定視窗法預測方法的RMSE",rmsefix)
print("第二種遞增視窗法預測方法的RMSE",rmserec)
print("第三種滾動視窗法預測方法的RMSE",rmserolling)
```

圖五

固定視窗法的 RMSE 11.824367116727437

遞增視窗法的 RMSE 6.285766198558837

滾動視窗法的 RMSE 1.744052545544236



圖六

4.2.2. ARCH

用 pmdarima 套件找出最佳的參數組合，透過報表(圖七)，得知 pmdarima 所提供的最佳參數組合為(1, 0, 1)，此外，也可以從參數估計報表中得知每項 P-value 皆小於 0.5，所以各參數的有著顯著的水準。而從 ARIMA 模型診斷圖上(圖八)發現殘差診斷可以目測它是無明顯的趨勢(圖八左上)，但是觀察殘差直方圖(圖八右上)卻發現具有窄峰厚尾的現象，代表不符合迴歸常態假設；並且從 QQPLOT(圖八左下)來看，模型殘差的常態性不足，所以 ARIMA 模型的殘差項應該還有潛在的解釋變量，因此需要用 ARCH 模型(圖九)找出誤差項中的變異數解釋變量(圖十一)。

隨後藉由 Ljung-Box 檢定觀察 ARCH 模型的殘差項是否為隨機變動，若檢定結果呈現白噪音(隨機)，則可以知道 ARCH 模型的配適程度良好，便直接做預測(如圖十一)。

由圖十二檢定結果讓我們能夠以 P-value 判斷，此誤差項是白噪音(P-value > 0.05)，代表殘差項目中沒有其他解釋變量可以提取，已經是隨機過程，所以下一步就開

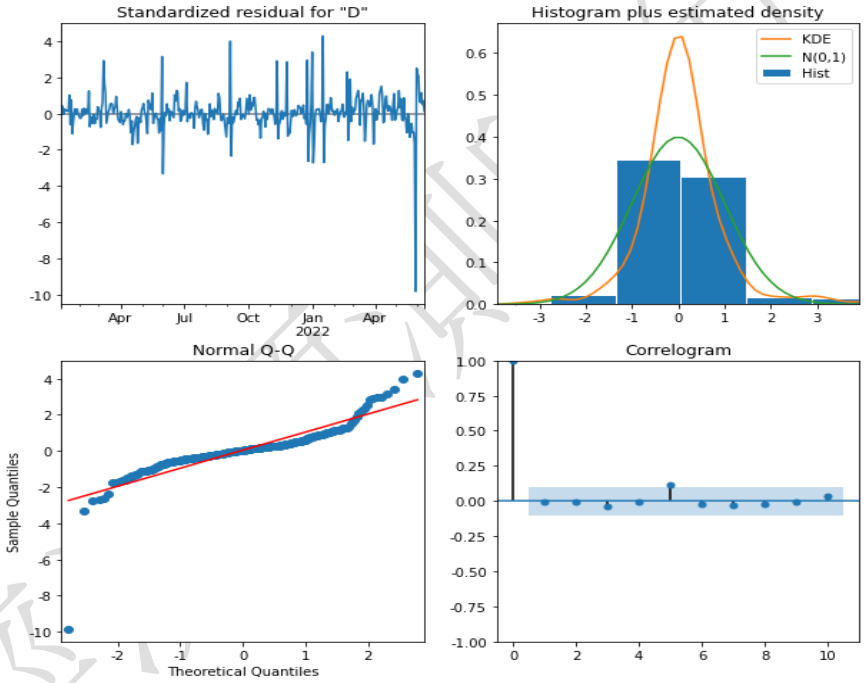
始預測。

SARIMAX Results

Dep. Variable:	y	No. Observations:	375
Model:	SARIMAX(1, 0, 1)	Log Likelihood	239.036
Date:	Mon, 20 Jun 2022	AIC	-470.072
Time:	17:40:19	BIC	-454.364
Sample:	0	HQIC	-463.836
	- 375		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.1417	0.189	0.749	0.454	-0.229	0.512
ar.L1	0.9950	0.007	149.482	0.000	0.982	1.008
ma.L1	-0.4169	0.025	-16.647	0.000	-0.466	-0.368
sigma2	0.0162	0.000	36.229	0.000	0.015	0.017

圖七



圖八

```
_arch_model1 = arch_model(_model1_result.resid, mean='Zero', p=4)
```

圖九

```
                     Zero Mean - GARCH Model Results
==============================================================================
Dep. Variable:                    None   R-squared:                       0.000
Mean Model:                 Zero Mean   Adj. R-squared:                  0.003
Vol Model:                      GARCH   Log-Likelihood:                266.804
Distribution:                  Normal   AIC:                           -521.607
Method:            Maximum Likelihood   BIC:                           -498.046
                                        No. Observations:                  375
Date:                Mon, Jun 20 2022   Df Residuals:                      375
Time:                        17:52:16   Df Model:                            0
                            Volatility Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.0108  3.725e-03      2.898  3.755e-03  [3.494e-03,1.810e-02]
alpha[1]       0.5471  2.721e-02     20.108  6.233e-90      [  0.494,  0.600]
alpha[2]       0.0000  4.709e-03      0.000      1.000 [-9.230e-03,9.230e-03]
alpha[3]       0.0000  4.033e-06      0.000      1.000 [-7.905e-06,7.905e-06]
alpha[4]       0.0000  1.121e-05      0.000      1.000 [-2.197e-05,2.197e-05]
beta[1]        0.0000  8.478e-03      0.000      1.000 [-1.662e-02,1.662e-02]
```

圖十

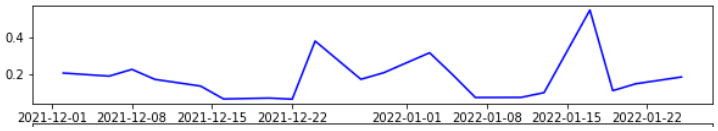
```
from statsmodels.stats.diagnostic import acorr_ljungbox

garch_std_resid = pd.Series(_arch_result.resid / _arch_result.conditional_volatility)
white_noise = acorr_ljungbox(garch_std_resid, lags = [10], return_df=True)
white_noise
```

圖十一

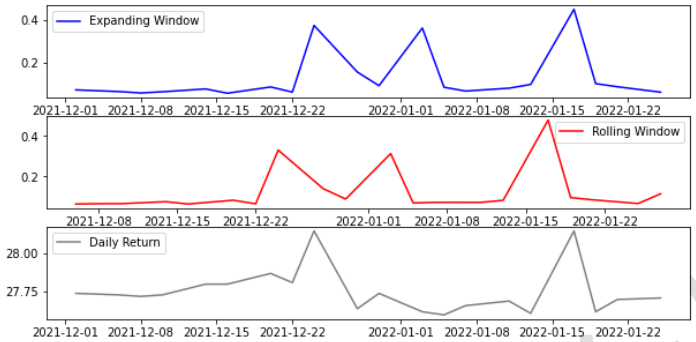
lb_stat	lb_pvalue
0.610837	0.999983

圖十二

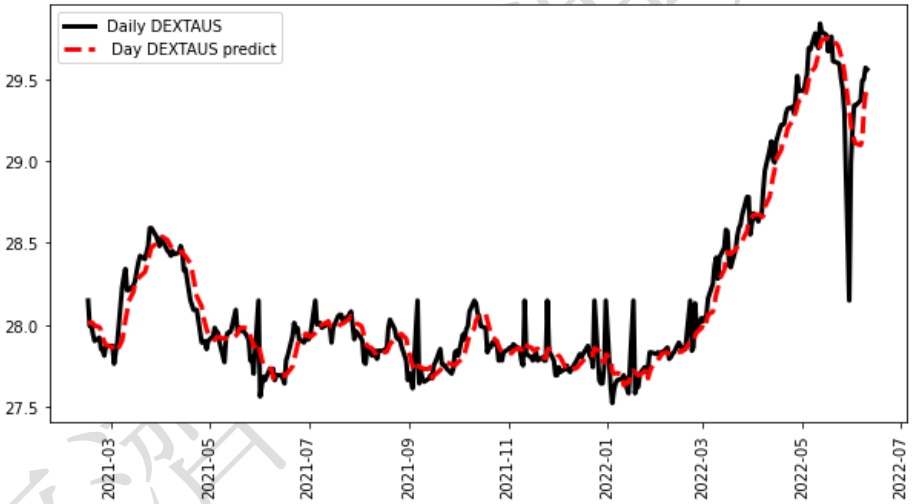
接著是預測的殘差結果，依序分為三個方法，遞增視窗法（如圖十三）、固定視窗法(如圖十四的第一張)、滾動視窗(如圖十四的第二張)，這裡可以發現其實不管哪個方法，都大致有把原始資料的的折線圖(如圖十四的第三張)走向給描繪出來，然而，從圖十六更可以清楚的發現預測的結果都有把原先資料起伏的趨勢描繪出來，表示著模型訓練的結果效果不錯，只是有些變化程度的大小沒有表示得很準確。



圖十三



圖十四



圖十五

4.2.3. GARCH

與 ARCH 作法差不多，一開始一樣用 pmdarima 套件找出最佳的參數組合，透過報表(圖十六)，得知 pmdarima 所提供的最佳參數組合為(1, 0, 1)，此外，也可以從參數估計報表中得知每項 P-value 皆小於 0.5，所以各參數的有著顯著的水

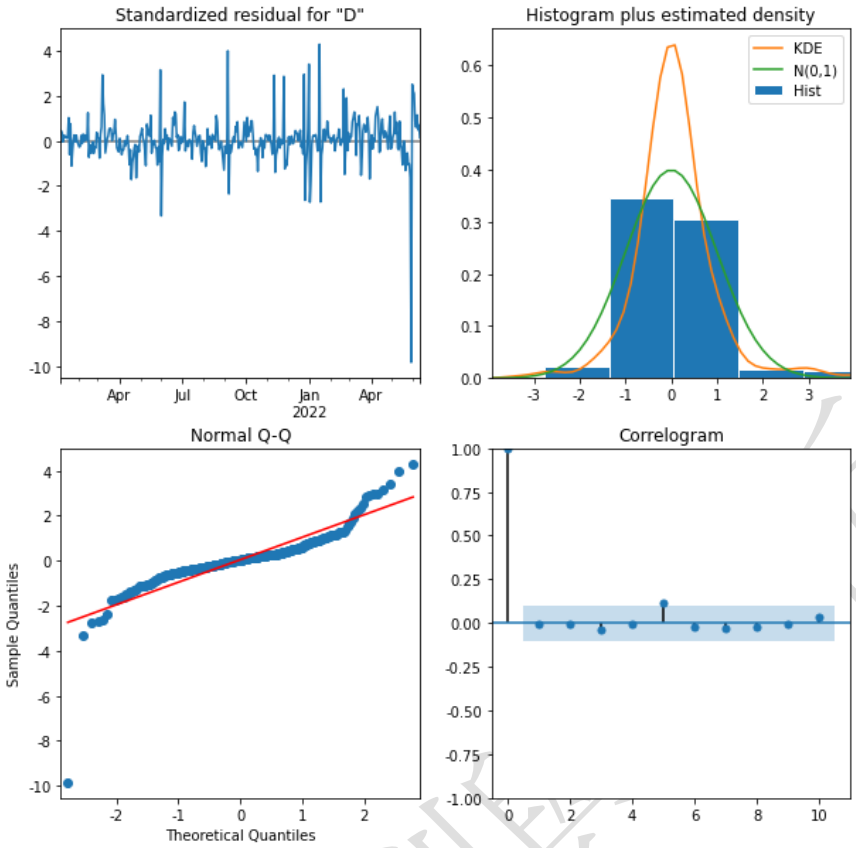
準。而從 ARIMA 模型診斷圖上(圖十七)發現殘差診斷可以目測它是無明顯的趨勢(圖十七左上),但是觀察殘差直方圖(圖十七右上)卻發現具有窄峰厚尾的現象,代表不符合迴歸常態假設;並且從 QQPLOT(圖十七左下)來看,模型殘差的常態性不足,所以 ARIMA 模型的殘差項應該還有潛在的解釋變量,因此需要用 GARCH 模型找出誤差項中的變異數解釋變量(圖十八)。

隨後藉由 Ljung-Box 檢定觀察 GARCH 模型的殘差項是否為隨機變動,若檢定結果呈現白噪音(隨機),則可以知道 GARCH 模型的配適程度良好,便直接做預測(如圖十九)。

由圖二十檢定結果讓我們能夠以 P-value 判斷,此誤差項是白噪音(P-value > 0.05),代表殘差項目中沒有其他解釋變量可以提取,已經是隨機過程,所以下一步就開始預測。

```
Best model:  ARIMA(1,0,1)(0,0,0)[0] intercept
Total fit time: 6.069 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  375
Model:               SARIMAX(1, 0, 1)   Log Likelihood                239.036
Date:                Tue, 21 Jun 2022   AIC                           -470.072
Time:                        03:58:57   BIC                           -454.364
Sample:                             0   HQIC                          -463.836
                                  - 375
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.1417      0.189      0.749      0.454      -0.229       0.512
ar.L1          0.9950      0.007    149.482      0.000       0.982       1.008
ma.L1         -0.4169      0.025    -16.647      0.000      -0.466      -0.368
sigma2         0.0162      0.000     36.229      0.000       0.015       0.017
===================================================================================
Ljung-Box (L1) (Q):                   0.09   Jarque-Bera (JB):              9497.45
Prob(Q):                              0.76   Prob(JB):                         0.00
Heteroskedasticity (H):               3.55   Skew:                            -1.77
Prob(H) (two-sided):                  0.00   Kurtosis:                        27.40
```

圖十六



圖十七

Zero Mean - GARCH Model Results

Dep. Variable:	None	R-squared:	0.000
Mean Model:	Zero Mean	Adj. R-squared:	0.003
Vol Model:	GARCH	Log-Likelihood:	262.847
Distribution:	Normal	AIC:	-509.695
Method:	Maximum Likelihood	BIC:	-478.279
		No. Observations:	375
Date:	Tue, Jun 21 2022	Df Residuals:	375
Time:	04:00:01	Df Model:	0

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	8.7549e-03	2.025e-03	4.323	1.542e-05	[4.785e-03,1.272e-02]
alpha[1]	0.3864	9.912e-03	38.982	0.000	[0.367, 0.406]
alpha[2]	4.5402e-07	5.257e-03	8.636e-05	1.000	[-1.030e-02,1.030e-02]
alpha[3]	2.2726e-06	6.482e-06	0.351	0.726	[-1.043e-05,1.498e-05]
alpha[4]	4.0972e-06	1.083e-05	0.378	0.705	[-1.713e-05,2.533e-05]
beta[1]	1.0446e-07	1.160e-02	9.008e-06	1.000	[-2.273e-02,2.273e-02]
beta[2]	1.2507e-06	4.911e-04	2.547e-03	0.998	[-9.613e-04,9.638e-04]
beta[3]	1.8378e-06	8.034e-05	2.287e-02	0.982	[-1.556e-04,1.593e-04]

圖十八

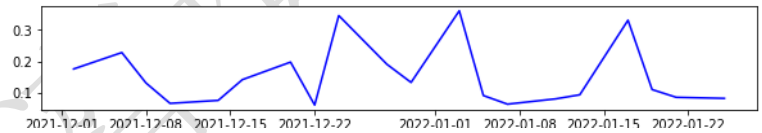
```
from statsmodels.stats.diagnostic import acorr_ljungbox

garch_std_resid = pd.Series(_garch_result.resid / _garch_result.conditional_volatility)
white_noise = acorr_ljungbox(garch_std_resid, lags = [10], return_df=True)
white_noise
```

圖十九

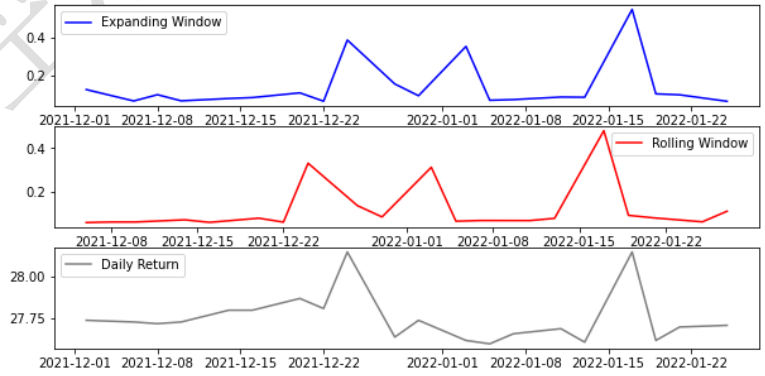
	lb_stat	lb_pvalue
10	9.121857	0.520579

圖二十

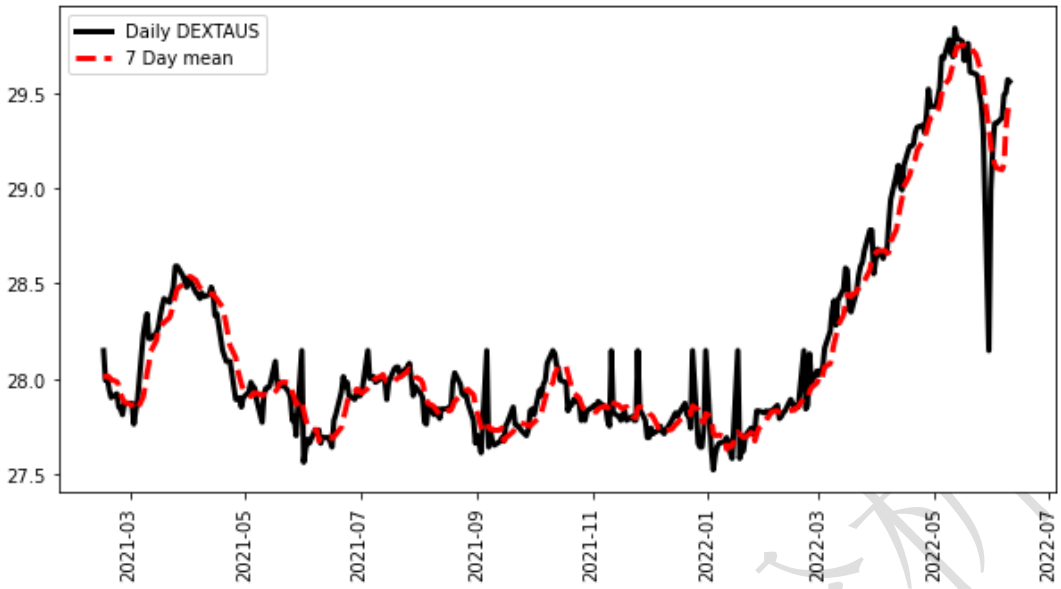
這裡是預測的殘差結果，依序分為三個方法，遞增視窗法（如圖二十一）、固定視窗法(如圖二十二的第一張)、滾動視窗(如圖二十二的第二張)，這裡可以發現其實不管哪個方法，都大致有把原始資料的的折線圖(如圖二十二的第三張)走向給描繪出來，然而，從圖二十三更可以清楚的發現預測的結果都有把原先資料起伏的趨勢描繪出來，表示著模型訓練的結果效果不錯，只是有些變化程度的大小沒有表示得很準確。



圖二十一



圖二十二



圖二十三

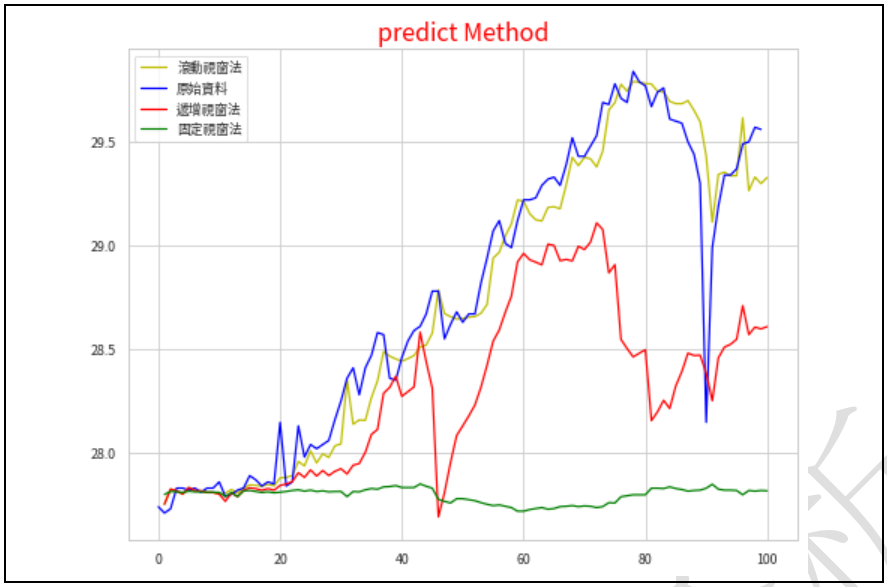
4.3. 輸出結果

這個部分在整理台灣對三個國家的匯率利用不同模型的比較，以預測結果來看，台灣對日本匯率的資料都比另外兩個國家相較的準確不少，像表一、表三、表五當中可以明顯的發現台灣對日本匯率的 RMSE 是最低的，其次是台灣對大陸匯率再來才是台灣對美國匯率，而表二、表四、表六也可以發現每日預測趨勢，用台灣對日本匯率是最接近，其次是台灣對大陸匯率再來才是台灣對美國匯率

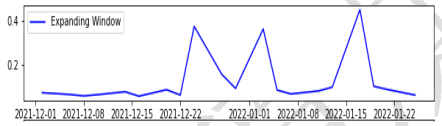
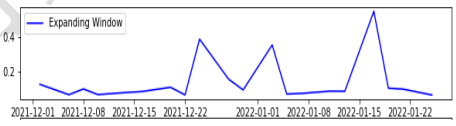
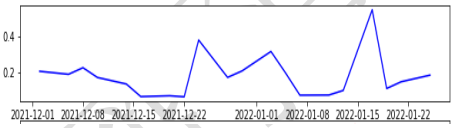
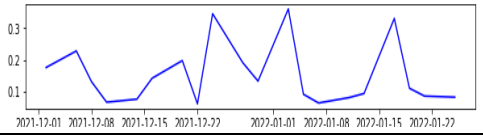
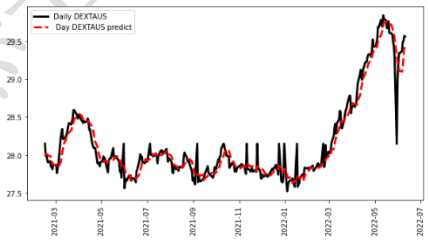
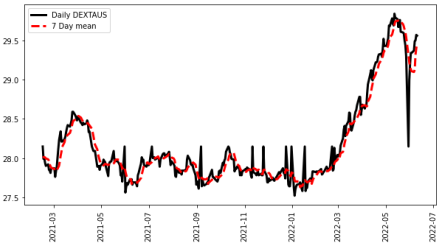
4.3.1. 台灣對美國匯率

表一

OLS	
固定視窗法 RMSE	11.8243
遞增視窗法 RMSE	6.2857
滾動視窗法 RMSE	1.7440

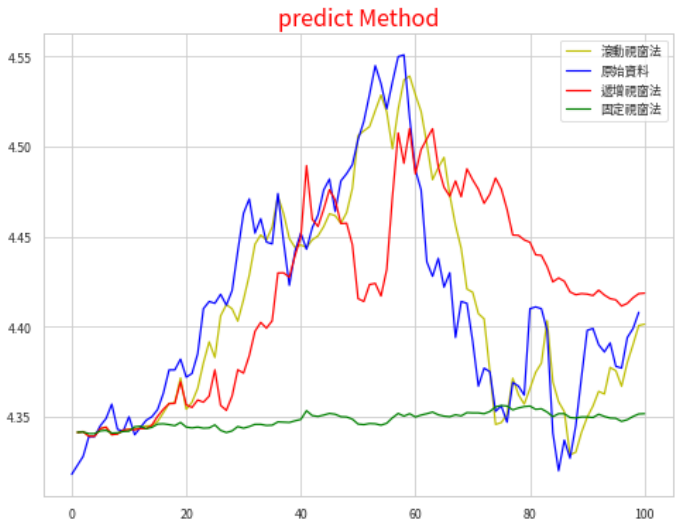


表二

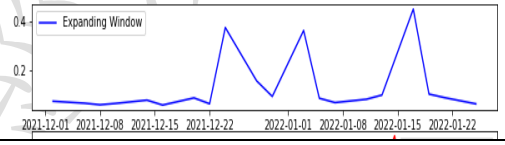
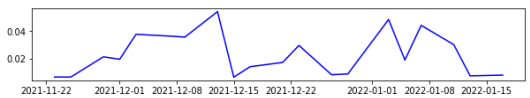
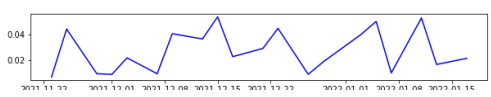
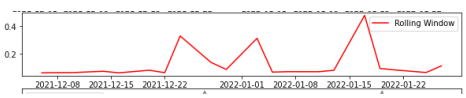
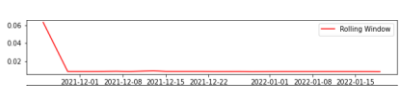
	ARCH	GRACH
固定視窗法		
遞增視窗法		
滾動視窗法		
每日預測趨勢		

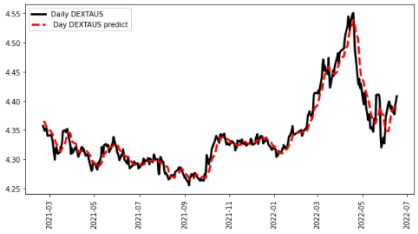
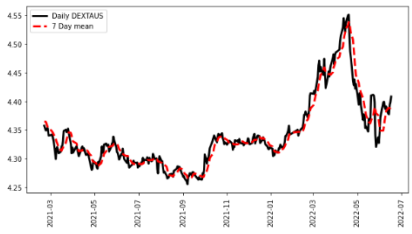
4.3.2. 台灣對大陸匯率

表三

OLS	
固定視窗法 RMSE	0.8673
遞增視窗法 RMSE	0.5720
滾動視窗法 RMSE	0.2466
	

表四

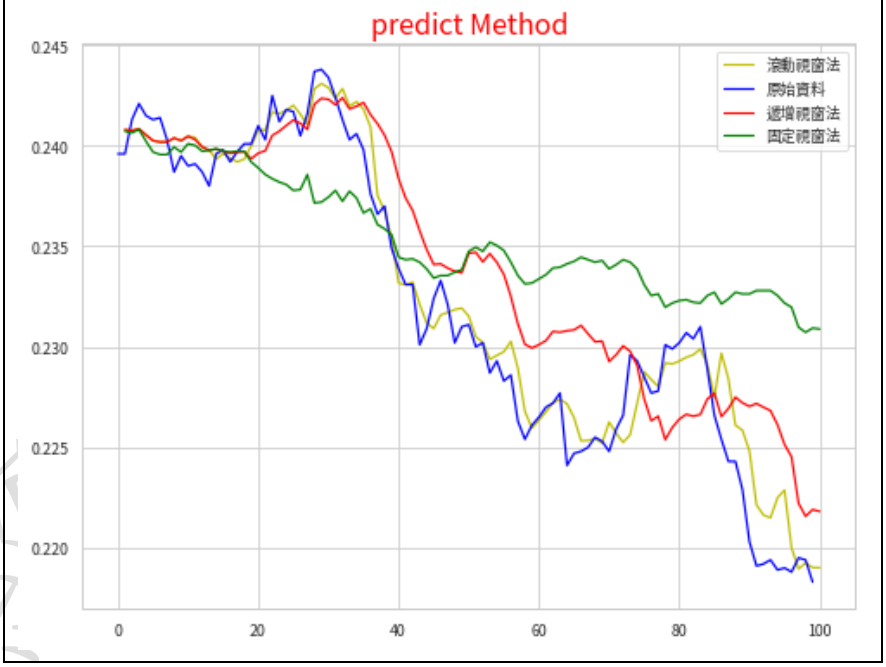
	ARCH	GRACH
固定視窗法		
遞增視窗法		
滾動視窗法		

每日預測趨勢		

4.3.3. 台灣對日本匯率

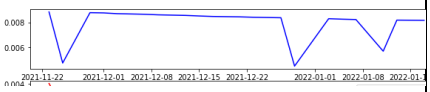
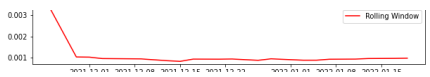
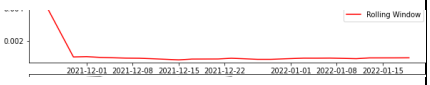
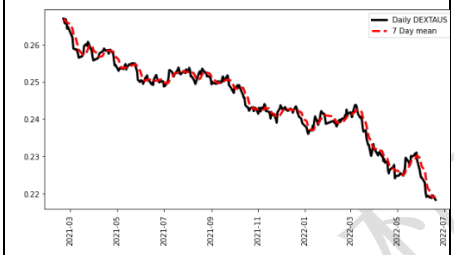
表五

OLS	
固定視窗法 RMSE	0.0611
遞增視窗法 RMSE	0.0356
滾動視窗法 RMSE	0.0154



表六

	ARCH	GRACH
固定視窗法		

遞增視窗法		
滾動視窗法		
每日預測趨勢		

5. 結論

這個部分在整理台灣對三個國家的匯率利用不同模型的比較，以預測結果來看，台灣對日本匯率的資料都比另外兩個國家相較的準確不少，像表一、表三、表五，三國之 OLS 當中可以明顯的發現台灣對日本匯率的 RMSE 是最低的，其次是台灣對大陸匯率再來才是台灣對美國匯率，且皆是滾動視窗法較為優秀，ARCH 及 GARCH 三個方法中也是滾動視窗法較好，後面殘差皆靠近 0.001，而表二、表四、表六也可以發現每日預測趨勢，用台灣對日本匯率是最接近，其次是台灣對大陸匯率再來才是台灣對美國匯率。

6. 參考資料

[1] Regression Analysis Simple Linear Regression 蔡佳泓 國立政治大學東亞所 May 7, 2014

[2] https://zh.wikipedia.org/zh-tw/%E6%99%82%E9%96%93%E5%BA%8F%E5%88%97

[3] ARCH/GARCH 模型設定、估計、檢定與實例分析 林金龍 東華大學財務金融系

[4] https://fred.stlouisfed.org/series/DEXTAUS

[5] https://nccur.lib.nccu.edu.tw/bitstream/140.119/35777/6/803506.pdf

[6] https://haosquare.com/normal-distribution-qqplot/

[7] https://www.796t.com/content/1545130624.html

[8] https://reurl.cc/ErN9yv

[9] https://wiki.mbalib.com/zh-tw/%E6%B1%87%E7%8E%87

[10]https://fred.stlouisfed.org/series/DEXTAUS

[11]https://rate.bot.com.tw/xrt?Lang=zh-TW

7. 分工項目

書面：林咨吟 鄭竣隆 陳昱廷 王詠慈 朱銘浩 簡睿德
程式: 林咨吟 簡睿德 溫宏岳
整理數據: 陳昱廷
PPT: 鄭竣隆 王詠慈

8. 附錄(程式碼)

8.1. OLS

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from collections import Counter
from pandas import DataFrame

sns.set_style("whitegrid")
sns.set_context("paper")
df = pd.read_csv("/content/drive/MyDrive/經濟預測/data/人民幣.csv")
```

檢查資料是否有缺失值

```
df.isnull().sum(axis=0)
```

```
df.index = pd.to_datetime(df['observation_date'])
df=df.drop(['observation_date'],axis =1)
df.head()
```

```
#畫匯率
plotInd=0

plt.figure()
ax = df.DEXTAUS[plotInd:].plot(rot=78)
ax.set_xticklabels(df[plotInd:].index)
plt.title('新台幣兌美元即期匯率')
plt.rcParams['font.sans-serif'] = ['Taipei Sans TC Beta']
plt.legend()
```

```
#取平均值
df.fillna(value = {'DEXTAUS': df['DEXTAUS'].mean()}, inplace = True )
df.head()
```

```
# split into train/test
n_test = 100
X, Y = df[:n_test], df[-n_test:]#後面100筆當作test data
print("train筆數:",len(X))
print(X.head())
print("-----------------")
print("test筆數:",len(Y))
print(Y.head())
```

```
X = sm.add_constant(X['DEXTAUS'])
display(X)
```

```
windowsN=13
```

```
#設定要估計的參數資料長度
fixwinX=X[0:windowsN]
display(fixwinX)
```

```
#設定要估計的參數資料長度
fixwinY=Y[1:windowsN+1]['DEXTAUS']
display(fixwinY)
print(type(fixwinY))
```

```
#估計模型參數
fixwinmodel = sm.OLS(np.array(fixwinY), np.array(fixwinX)).fit()
```

```
#estimate fixwinmodel Y 透過參數自己運算 fixwinmodel
estifixwinY=np.sum(fixwinX*fixwinmodel.params,1)
```

```
#設定X資料的數量
fixwinpredX=X[windowsN:]
display(fixwinpredX)
```

```
#predict fixwinmodel Y 透過參數自己運算 fixwinmodel
predFixwinY=np.array(np.sum(fixwinX*fixwinmodel.params,1))
```

```
#將資料轉換為np array 方便計算用
predFixarray=np.array(predFixwinY)
```

```
#設定預測模型的資料
predFixY=np.array(np.sum(fixwinpredX*fixwinmodel.params,1))
```

```
#設定資料
predFixarray=np.append(predFixarray,predFixY)
```

```
[ ] #設定估計遞增視窗的辦法
    recwindowsN=10
```

```
[ ] for i in range(xsizem-recwindowsN):
        if i==0:
            #print(i)
            #設定X的資料開始與結束的範圍
            recwinX=X[0:recwindowsN+i]
            print(len(recwinX))
            #設定Y的資料開始與結束的範圍，要落後一期，才能預測。
            recwinY=Y[1:recwindowsN+1+i]['DEXTAUS']
            #估計OLS
            recwinmodel = sm.OLS(np.array(recwinY), np.array(recwinX)).fit()
            #計算預測與估計的值
            predrecwinY=np.sum(recwinX*recwinmodel.params,1)
            #將資料轉為np array 方便計算
            predRecarray=np.array(predrecwinY)
        else:
            #print(i)
            #設定X的資料開始與結束的範圍
            recwinX=X[0:recwindowsN+i]
            #print(len(recwinX))
            #設定Y的資料開始與結束的範圍，要落後一期，才能預測。
            recwinY=Y[1:recwindowsN+1+i]['DEXTAUS']
            #估計OLS
            recwinmodel = sm.OLS(np.array(recwinY), np.array(recwinX)).fit()
            predrecwinY=np.sum(recwinX*recwinmodel.params,1)
            #每執行一次，就將資料加入到predRecarray的資料中
```

```
            recwinY=Y[1:recwindowsN+1+i]['DEXTAUS']
            #估計OLS
            recwinmodel = sm.OLS(np.array(recwinY), np.array(recwinX)).fit()
            predrecwinY=np.sum(recwinX*recwinmodel.params,1)
            #每執行一次，就將資料加入到predRecarray的資料中
            predRecarray=np.append(predRecarray,predrecwinY[-1:])
```

```
10
```

```
[ ] #將最後一筆X的資料用來預測未來一期的資料
    predRecarray=np.append(predRecarray,np.sum(X[-1:]*recwinmodel.params,1))
    #顯示結果
    print(predRecarray)
```

```
[ ] rollingN=10
    for i in range(20):
        if i==0:
            #print(i)
            rollingwinX=X[0+i:rollingN+i]
            #print(len(rollingwinX))
            rollingwinY=Y[1+i:rollingN+1+i]['DEXTAUS']
            #估計OLS
            rollingwinmodel = sm.OLS(np.array(rollingwinY), np.array(rollingwinX)).fit()
            #計算預測與估計的值
            predrollingwinY=np.sum(rollingwinX*rollingwinmodel.params,1)
            #將資料轉為np array 方便計算
            predRollingarray=np.array(predrollingwinY)
        else:
            #print(i)
            rollingwinX=X[0+i:rollingN+i]
            #print(len(rollingwinX))
            rollingwinY=Y[1+i:rollingN+1+i]['DEXTAUS']
            #估計OLS
            rollingwinmodel = sm.OLS(np.array(rollingwinY), np.array(rollingwinX)).fit()

            predrollingwinY=np.sum(rollingwinX*rollingwinmodel.params,1)
            #每執行一次，就將資料加入到predRollingarray的資料中
            predRollingarray=np.append(predRollingarray,predrollingwinY[-1:])
```

```
[ ] #將最後一筆X的資料用來預測未來一期的資料
    predRollingarray=np.append(predRollingarray,np.sum(X[-1:]*rollingwinmodel.params,1))
```

```
#第一種固定視窗法
x = np.linspace(0,xsizem-1,xsizem)
xshift=np.linspace(1,xsizem,xsizem)
#圖形的大小8*6可以自行調整
fig, ax = plt.subplots(figsize=(8,6))
#畫估計的Y值estimateY
ax.plot(xshift, predFixarray, '2k', label="predict value")
#畫Y值
ax.plot(x, Y['DEXTAUS'], 'y-.', label="True Y")
#顯示預測結果
ax.legend(loc='best')
ax.set_title('Fixed Method',fontsize=16,color='r')
```

```
realY=Y[1:]['DEXTAUS']
#三種預測方法 RMSE的比較
#第一種方法 固定視窗法 RMSE
rmsefix=np.sqrt(sum((predFixarray[0:-1]-realY)**2))
#第二種方法 遞增視窗法 RMSE
rmserec=np.sqrt(sum((predRecarray[0:-1]-realY)**2))
#第三種方法 滾動視窗法 RMSE
rmserolling=np.sqrt(sum((predRollingarray[0:-1]-realY)**2))
print("第一種固定視窗法預測方法的RMSE",rmsefix)
print("第二種遞增視窗法預測方法的RMSE",rmserec)
print("第三種滾動視窗法預測方法的RMSE",rmserolling)
```

```
#第三種滾動視窗法
#圖形的大小8*6可以自行調整
fig, ax = plt.subplots(figsize=(8,6))
#畫估計的Y值estimateY
ax.plot(xshift, predRollingarray, 'y-', label="滾動視窗法")
#畫Y值
ax.plot(x, Y['DEXTAUS'], 'b-', label="原始資料")
ax.plot(xshift, predRecarray, 'r-', label="遞增視窗法")
ax.plot(xshift, predFixarray, 'g-', label="固定視窗法")
#顯示預測結果
ax.legend(loc='best')
ax.set_title('predict Method',fontsize=16,color='r')
```

8.2. ARCH

```
!pip install pmdarima
!pip install arch
!pip install --upgrade pandas-datareader

!pip install --upgrade pandas
```

```
import sys
import random
#import requests
import pandas as pd
#import datetime as dt
import numpy as np
import pmdarima as pm
import statsmodels.api as sm
from pandas_datareader import data as pdr
from datetime import date
from numpy import cumsum, log, polyfit, sqrt, std, subtract
from numpy.linalg import LinAlgError
from pandas_datareader import data as web
from arch import arch_model
#import statsmodels.tsa.api as tsa
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import acf, q_stat, adfuller
from scipy.stats import probplot, moment
import matplotlib.pyplot as plt
#設定圖形的package
import matplotlib as mpl
```

```
[ ] #設定圖形中文顯示字形
    mpl.rcParams["font.sans-serif"] = ["Microsoft YaHei"]
    mpl.rcParams['axes.unicode_minus'] = False
```

```
[ ]
    #估計遠期變動量
    def taufunction(df,startindex,k):
            dfmean=df[startindex-2*(k-1):].rolling(k).mean()
            dfmeanklambda=dfmean.dropna()
            dfmeank=dfmean[(k-1):]
            dfmeankln=np.log(dfmeank)
            dfmeanklndiff=dfmeankln.diff(k-1)
            tauN=-k/dfmeanklndiff
            tauN.rename('tau_'+str(k), inplace=True)
            tauN=tauN.dropna()
            return tauN
```

```
[ ] def automodel(x):

            model = pm.auto_arima(x,
            d=0, # non-seasonal difference order
            start_p=1, # initial guess for p
            start_q=1, # initial guess for q
            max_p=4, # max value of p to test
            #max_q=4, # max value of q to test

            seasonal=False, # is the time series seasonal

            information_criterion='aic', # used to select best model
            trace=True, # print results whilst training
            error_action='trace', # ignore orders that don't work
            stepwise=True, # apply intelligent order search
            )
            return model
```

```
[ ] import pandas as pd
    import numpy as np
    from matplotlib import pyplot
    import matplotlib.pyplot as plt
    import arch
    from arch import arch_model
    from statsmodels.graphics.tsaplots import plot_acf
    from matplotlib.font_manager import FontProperties
    from arch.__future__ import reindexing
    import statsmodels.graphics.tsaplots as sgt

    #讀取資料
    stockdata = pd.read_csv('/content/drive/MyDrive/經濟預測/data/台日即期匯率.csv')
    #刪除空缺值
    #data = data[data['DEXTAUS'].notna()]
    #取平均值
    stockdata.fillna(value = {'DEXTAUS': stockdata['DEXTAUS'].mean()}, inplace = True )
    df=stockdata
    df.index = pd.to_datetime(stockdata['observation_date'])
    df=df.drop(['observation_date'],axis =1)
```

```
    #畫匯率
    plotInd=0

    plt.figure()
    ax = df.DEXTAUS[plotInd:].plot(rot=78)
    ax.set_xticklabels(df[plotInd:].index)
    plt.title('Daily DEXTAUS')
    plt.savefig('pct_result.png',bbox_inches='tight',transparent=True, pad_inches=0.5,dpi = 600)
    plt.show()
```

```
    model=automodel(df['DEXTAUS'])
    print(model.summary())
```

```
# Fit best model
_arma_model = sm.tsa.SARIMAX(endog=df['DEXTAUS'],order=(2,0,2))
_model_result = _arma_model.fit()

# Plot model residuals
_model_result.plot_diagnostics(figsize=(10, 10))
plt.savefig('plot_diagnostics.png',bbox_inches='tight',transparent=True, pad_inches=0.5,dpi = 600)
plt.show()
```

```
#畫arch model
# Fit GARCH model with ARMA model residuals
_model_result.resid=_model_result.resid.fillna(value=0)
_arch_model = arch_model(_model_result.resid, mean='Zero', p=4)


_arch_result = _arch_model.fit(disp = 'Normal')
print(_arch_result.summary())
# Plot GARCH model fitted results
_arch_result.plot()
plt.savefig('_arch_result.png',bbox_inches='tight',transparent=True, pad_inches=0.5,dpi = 600)
plt.show()
```

```
from statsmodels.stats.diagnostic import acorr_ljungbox

garch_std_resid = pd.Series(_arch_result.resid / _arch_result.conditional_volatility)
white_noise = acorr_ljungbox(garch_std_resid, lags = [10], return_df=True)
white_noise
```

```
index = df.index
start_loc = 1

#end_loc = 7#dftimeseries.index.shape[0]-windowsk+1
end_loc = np.where(index >= "2021-06-20")[0].min()
#windowsk=df.index.shape[0]-end_loc+1
forecastswin = {}
for i in range(20):
        sys.stdout.write('-')
        sys.stdout.flush()
        res = _arch_model.fit(first_obs=start_loc + i, last_obs=start_loc +i + end_loc, disp='off')
        temp = res.forecast(horizon=1).variance
        fcast = temp.iloc[start_loc +i + end_loc - 1]
        forecastswin[fcast.name] = fcast
print(' Done!')
variance_fixedwinwin = pd.DataFrame(forecastswin).T
print(variance_fixedwinwin)
```

```
#Implement expanding window forecast¶
forecasts = {}
for i in range(20):
        sys.stdout.write('-')
        sys.stdout.flush()
        res = _arch_model.fit(first_obs = start_loc, last_obs = i + end_loc, disp = 'off')
        temp = res.forecast(horizon=1).variance
        fcast = temp.iloc[i + end_loc - 1]
        forecasts[fcast.name] = fcast
print(' Done!')
variance_expandwin = pd.DataFrame(forecasts).T

#預測變異數
# Calculate volatility from variance forecast with an expanding window
vol_expandwin = np.sqrt(variance_expandwin)
#預測固定波動變異數
# Calculate volatility from variance forecast with a fixed rolling window
vol_fixedwin = np.sqrt(variance_fixedwinwin)
```

```
#=====================================================
# Plot results
figure =plt.figure(figsize=(10,5))
# Plot volatility forecast with a fixed rolling window
axes_1 = figure.add_subplot(3,1,1)
# Plot volatility forecast with an expanding window
axes_1.plot(vol_expandwin, color = 'blue', label='Expanding Window')#預測變異數
plt.legend()
axes_2 = figure.add_subplot(3,1,2)
axes_2.plot(vol_fixedwin, color = 'red', label='Rolling Window')#預測固定波動變異數
plt.legend()
axes_3 = figure.add_subplot(3,1,3)
axes_3.plot(df['DEXTAUS'].loc[variance_expandwin.index], color = 'grey', label='Daily Return')
plt.legend()
plt.savefig('volatility_forecast.png',bbox_inches='tight',transparent=True, pad_inches=0.5,dpi = 600)
figure.show()
```

```
[ ] k=7
startindex=30
tau2=taufunction(df.DEXTAUS,startindex,2)
tau3=taufunction(df.DEXTAUS,startindex,3)
tau5=taufunction(df.DEXTAUS,startindex,5)
tau7=taufunction(df.DEXTAUS,startindex,7)
tau10=taufunction(df.DEXTAUS,startindex,10)
tau14=taufunction(df.DEXTAUS,startindex,14)

dfmeank=df.DEXTAUS[startindex-2*(k-1):].rolling(k).mean()
lambdamean7=dfmeank.dropna()
dfmeank=lambdamean7[(k-1):]
```

```
[ ] figure =plt.figure(figsize=(10,5))
xtick_init=df[startindex:].index
plt.plot(xtick_init,df.DEXTAUS[startindex:], color = 'black', label='Daily DEXTAUS', linewidth = '3')
plt.plot(xtick_init,dfmeank, color = 'red', label=' Day DEXTAUS predict', linestyle='--', linewidth = '3')

plt.xticks(rotation=90)
plt.legend()
plt.savefig('predictResult.png',bbox_inches='tight',transparent=True, pad_inches=0.5,dpi = 600)
figure.show()
```

```
import numpy as np
import pandas as pd

index = df.index
start_loc = 0
end_loc = np.where(index >= "2021-06-20")[0].min()#開始日期
forecasts = {}
for i in range(20):#開始日期往後預測20天
        sys.stdout.write(".")
        sys.stdout.flush()
        res = _arch_model.fit(last_obs=i + end_loc, disp="off")
        temp = res.forecast(horizon=3, reindex=False).variance
        fcast = temp.iloc[0]
        forecasts[fcast.name] = fcast
print()
print(pd.DataFrame(forecasts).T)
```

```
[ ] forecasts = {}
for i in range(20):
        sys.stdout.write('-')
        sys.stdout.flush()
        res = _arch_model.fit(first_obs = start_loc, last_obs = i + end_loc, disp = 'off')
        temp = res.forecast(horizon=1).variance
        fcast = temp.iloc[i + end_loc - 1]
        forecasts[fcast.name] = fcast
print(' Done!')
variance_expandwin = pd.DataFrame(forecasts).T

#預測變異數
# Calculate volatility from variance forecast with an expanding window
vol_expandwin = np.sqrt(variance_expandwin)
#預測固定波動變異數
# Calculate volatility from variance forecast with a fixed rolling window
vol_fixedwin = np.sqrt(variance_fixedwinwin)
```

```
[ ] # Plot results
    figure =plt.figure(figsize=(10,5))
    # Plot volatility forecast with a fixed rolling window
    axes_1 = figure.add_subplot(3,1,1)
    # Plot volatility forecast with an expanding window
    axes_1.plot(vol_expandwin, color = 'blue', label='Expanding Window')#預測變異數
    axes_2 = figure.add_subplot(3,1,2)
    axes_2.plot(vol_fixedwin, color = 'red', label='Rolling Window')#預測固定波動變異數
    plt.legend()
    axes_3 = figure.add_subplot(3,1,3)
    axes_3.plot(df['DEXTAUS'].loc[variance_expandwin.index], color = 'grey', label='Daily Return')
    plt.legend()
    plt.savefig('volatility_forecast.png',bbox_inches='tight',transparent=True, pad_inches=0.5,dpi = 600)
    figure.show()
```

8.3. GRACH

與 ARCH 大多數雷同,這裡只貼上布一樣的地方

```
[ ] def automodel(x):

            model = pm.auto_arima(x,
            d=0, # non-seasonal difference order
            start_p=1, # initial guess for p
            start_q=1, # initial guess for q
            max_p=4, # max value of p to test
            max_q=4, # max value of q to test

            seasonal=False, # is the time series seasonal

            information_criterion='aic', # used to select best model
            trace=True, # print results whilst training
            error_action='trace', # ignore orders that don't work
            stepwise=True, # apply intelligent order search
            )
            return model
```

```
#畫garch model
# Fit GARCH model with ARMA model residuals
_model_result.resid=_model_result.resid.fillna(value=0)
_garch_model = arch_model(_model_result.resid, mean='Zero', p=1, q=1)


_garch_result = _garch_model.fit(disp = 'Normal')
print(_garch_result.summary())
# Plot GARCH model fitted results
_garch_result.plot()
#plt.savefig('_garch_result.png',bbox_inches='tight',transparent=True, pad_inches=0.5,dpi = 600)
plt.show()
```