

AI 神經網絡深層學習與交易策略 應用實作班

Deep Learning on Trading Strategy Workshop

昀騰金融科技

技術長

董夢雲 博士

目錄

Part II 時間數列與機器學習策略介紹

七、時間數列預測基礎

八、主成分分析與時間數列預測

九、ARIMA 與 GARCH 的合併預測

十、機器學習模型(一)：羅吉斯迴歸與鑑別分析

十一、機器學習模型(二)：支持向量機

十二、機器學習模型(三)：決策樹與隨機森林

十三、機器學習模型(四)：配對交易的選對

Part II 時間數列與機器學習策略介紹

主題八、主成分分析與時間數列預測

- 一、主成分分析與降維處理
- 二、時間數列特性
- 三、時間數列的資料處理
- 四、R套件quantmod讀取資料

一、主成分分析與降維處理

◆ 主成分分析是使用直交的轉換(Orthogonal Transformation)，將一組觀察值可能具有相關的解釋變數，轉變成線性無關的解釋變數，這些轉換後的解釋變數稱之為主成分(Principal Components)。

- 主成分的數目小於或等於原先變數的數目。
- 第一個主成分解釋觀察值的變異能力最大，之後依序遞減並與前項直交無相關。

◆ 主成分分析在機器學習內被歸類成為降維(Dimension Reduction)類，特徵擷取(Feature Extraction)的一種方法，

- 降維就是希望資料的維度數減少，但整體的效能不會差異太多甚至會更好。
- 簡單說法是，降維是當資料維度數(變數)很多的時候，有沒有辦法讓維度數(變數)少一點，但資料特性不會差太多。

◆ 假設有一組 M 個變數的觀察數據，我們的目的是減少數據，使得能夠用 L 個向量來描述每個觀察值， $L < M$ 。

➤ 進一步假設，該數據被整理成一組具有 N 個向量的數據集，其中每個向量都代表 M 個變數的單一觀察數據。

➤ 原有數據

$$X^T : N \times M, X : M \times N$$

✓ N ：觀察值

✓ M ：變數數目

➤ 轉化後數據

$$Y^T : N \times L, Y : L \times N$$

✓ N ：觀察值

✓ L ：主成分數目

◆ 令 A 矩陣為 $X^T \cdot X$ 之乘積，A 為對稱矩陣，

$$A = X^T \times X = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{bmatrix}$$

➤ 使用 Schur Decomposition，分解為 Eigen-Values 矩陣 D，與 Eigen-Vectors 矩陣 U 的乘積。

$$A = U \times D \times U^T$$

$$D = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_N \end{bmatrix}, \quad U = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_N \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1N} \\ v_{21} & v_{22} & \dots & v_{2N} \\ \dots & \dots & \dots & \dots \\ v_{N1} & v_{N2} & \dots & v_{NN} \end{bmatrix}$$

➤ λ_i 表第 i 個 Eigen-Value， v_i 表第 i 個 Eigen-Vector。

◆ 只取前 L 個 Eigen-Values，只有前 L 個 Eigen-Vectors 有作用，形成新矩陣 B，

$$B = U \times O \times U^T$$

$$B = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_L \\ \dots \\ v_N \end{bmatrix} \times \begin{bmatrix} \lambda_1 & 0 & \dots & 0 & 0 & 0 \\ 0 & \lambda_2 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & 0 & 0 \\ 0 & 0 & \dots & \lambda_L & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_L \\ \dots \\ v_N \end{bmatrix}^T = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1N} \\ b_{21} & b_{22} & \dots & b_{2N} \\ \dots & \dots & \dots & \dots \\ b_{N1} & b_{N2} & \dots & b_{NN} \end{bmatrix} = Y^T \times Y$$

$$B = Y^T \times Y = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1N} \\ b_{21} & b_{22} & \dots & b_{2N} \\ \dots & \dots & \dots & \dots \\ b_{N1} & b_{N2} & \dots & b_{NN} \end{bmatrix}$$

(一)DAX指數與成分

◆ 德國 DAX 指數是由 30 個成分股編制而成，名單如下。

```
symbols = ['ADS.DE', 'ALV.DE', 'BAS.DE', 'BAYN.DE', 'BEI.DE',  
           'BMW.DE', 'CBK.DE', 'CON.DE', 'DAI.DE', 'DB1.DE',  
           'DBK.DE', 'DPW.DE', 'DTE.DE', 'EOAN.DE', 'FME.DE',  
           'FRE.DE', 'HEI.DE', 'HEN3.DE', 'IFX.DE', 'LHA.DE',  
           'LIN.DE', 'LXS.DE', 'MRK.DE', 'MUV2.DE', 'RWE.DE',  
           'SAP.DE', 'SDF.DE', 'SIE.DE', 'TKA.DE', 'VOW3.DE']
```

◆ 由 Yahoo!Finance 下載 2010/1/1~2015/12/31 歷史收盤價資料。

```
In [2]: import numpy as np
import pandas as pd
import pandas_datareader.data as web
from sklearn.decomposition import KernelPCA

symbols = ['ADS.DE', 'ALV.DE', 'BAS.DE', 'BAYN.DE', 'BEI.DE',
           'BMW.DE', 'CBK.DE', 'CON.DE', 'DAI.DE', 'DB1.DE',
           'DBK.DE', 'DPW.DE', 'DTE.DE', 'EOAN.DE', 'FME.DE',
           'FRE.DE', 'HEI.DE', 'HEN3.DE', 'IFX.DE', 'LHA.DE',
           'LIN.DE', 'LXS.DE', 'MRK.DE', 'MUV2.DE', 'RWE.DE',
           'SAP.DE', 'SDF.DE', 'SIE.DE', 'TKA.DE', 'VOW3.DE',
           '^GDAXI']
```

```
# data = pd.DataFrame()
# for sym in symbols:
#     data[sym] = web.DataReader(sym, data_source='yahoo', start='2010-01-01',
#                               end='2015-12-31')['Close']
# data = data.dropna()
# data.to_hdf('DAXCompAll.h5', key='data', mode='w')
```

◆ 由之前存好的 h5 檔，讀入資料

```
In [4]: data = pd.DataFrame()  
data = pd.read_hdf('DAXCompAll.h5', 'data');
```

◆ 顯示第一個股票

```
In [5]: print(data['ADS.DE'])
```

| Date | |
|------------|-----------|
| 2010-01-04 | 38.505001 |
| 2010-01-05 | 39.720001 |
| 2010-01-06 | 39.400002 |
| 2010-01-07 | 39.744999 |
| 2010-01-08 | 39.599998 |
| ... | |
| 2015-12-22 | 88.070000 |
| 2015-12-23 | 90.180000 |
| 2015-12-28 | 89.089996 |
| 2015-12-29 | 91.510002 |
| 2015-12-30 | 89.910004 |

Name: ADS.DE, Length: 1522, dtype: float64

◆ 指數部分

```
In [6]: print(data['^GDAXI'])
```

```
Date
2010-01-04    6048.299805
2010-01-05    6031.859863
2010-01-06    6034.330078
2010-01-07    6019.359863
2010-01-08    6037.609863
...
2015-12-22    10488.750000
2015-12-23    10727.639648
2015-12-28    10653.910156
2015-12-29    10860.139648
2015-12-30    10743.009766
Name: ^GDAXI, Length: 1522, dtype: float64
```

◆ 所有股票+指數

In [7]: data

Out[7]:

| | ADS.DE | ALV.DE | BAS.DE | BAYN.DE | BEI.DE | BMW.DE | CBK.DE | CON. |
|------------|-----------|------------|-----------|------------|-----------|-----------|-----------|----------|
| Date | | | | | | | | |
| 2010-01-04 | 38.505001 | 88.540001 | 44.849998 | 55.502998 | 46.445000 | 32.049999 | 48.426899 | 36.8678 |
| 2010-01-05 | 39.720001 | 88.809998 | 44.169998 | 54.489300 | 46.200001 | 32.310001 | 50.069698 | 39.1741 |
| 2010-01-06 | 39.400002 | 89.500000 | 44.450001 | 54.144901 | 46.165001 | 32.810001 | 51.744598 | 39.4304 |
| 2010-01-07 | 39.744999 | 88.470001 | 44.154999 | 53.436401 | 45.700001 | 33.099998 | 53.876202 | 44.4831 |
| 2010-01-08 | 39.599998 | 87.989998 | 44.020000 | 52.964001 | 44.380001 | 32.654999 | 54.573399 | 45.3050 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2015-12-22 | 88.070000 | 160.300003 | 68.989998 | 111.596001 | 82.099998 | 96.349998 | 9.427000 | 220.9498 |
| 2015-12-23 | 90.180000 | 163.000000 | 70.949997 | 114.252998 | 83.370003 | 98.550003 | 9.661000 | 225.5000 |
| 2015-12-28 | 89.089996 | 161.500000 | 70.309998 | 113.663002 | 83.040001 | 97.349998 | 9.634000 | 222.8500 |
| 2015-12-29 | 91.510002 | 165.000000 | 71.519997 | 116.025002 | 84.900002 | 98.559998 | 9.684000 | 227.5000 |
| 2015-12-30 | 89.910004 | 163.550003 | 70.720001 | 113.958000 | 84.160004 | 97.629997 | 9.572000 | 224.5500 |

1522 rows × 31 columns

< >

◆ 從 data 移除 ^GDAXI，用 ^GDAXI 型成新的 dataframe，dax

```
In [8]: data[data.columns[:6]].head()|  
dax = pd.DataFrame(data.pop( '^GDAXI '))
```

```
In [9]: dax|
```

```
Out[9]:
```

| | ^GDAXI |
|-------------|---------------|
| Date | |
| 2010-01-04 | 6048.299805 |
| 2010-01-05 | 6031.859863 |
| 2010-01-06 | 6034.330078 |
| 2010-01-07 | 6019.359863 |
| 2010-01-08 | 6037.609863 |
| ... | ... |
| 2015-12-22 | 10488.750000 |
| 2015-12-23 | 10727.639648 |
| 2015-12-28 | 10653.910156 |
| 2015-12-29 | 10860.139648 |
| 2015-12-30 | 10743.009766 |

1522 rows × 1 columns

◆ data 中只有 30 個股票

```
In [10]: data
```

```
Out[10]:
```

| | ADS.DE | ALV.DE | BAS.DE | BAYN.DE | BEI.DE | BMW.DE | CBK.DE | CON. |
|------------|-----------|------------|-----------|------------|-----------|-----------|-----------|----------|
| Date | | | | | | | | |
| 2010-01-04 | 38.505001 | 88.540001 | 44.849998 | 55.502998 | 46.445000 | 32.049999 | 48.426899 | 36.8678 |
| 2010-01-05 | 39.720001 | 88.809998 | 44.169998 | 54.489300 | 46.200001 | 32.310001 | 50.069698 | 39.1741 |
| 2010-01-06 | 39.400002 | 89.500000 | 44.450001 | 54.144901 | 46.165001 | 32.810001 | 51.744598 | 39.4304 |
| 2010-01-07 | 39.744999 | 88.470001 | 44.154999 | 53.436401 | 45.700001 | 33.099998 | 53.876202 | 44.4831 |
| 2010-01-08 | 39.599998 | 87.989998 | 44.020000 | 52.964001 | 44.380001 | 32.654999 | 54.573399 | 45.3050 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2015-12-22 | 88.070000 | 160.300003 | 68.989998 | 111.596001 | 82.099998 | 96.349998 | 9.427000 | 220.9499 |
| 2015-12-23 | 90.180000 | 163.000000 | 70.949997 | 114.252998 | 83.370003 | 98.550003 | 9.661000 | 225.5000 |
| 2015-12-28 | 89.089996 | 161.500000 | 70.309998 | 113.663002 | 83.040001 | 97.349998 | 9.634000 | 222.8500 |
| 2015-12-29 | 91.510002 | 165.000000 | 71.519997 | 116.025002 | 84.900002 | 98.559998 | 9.684000 | 227.5000 |
| 2015-12-30 | 89.910004 | 163.550003 | 70.720001 | 113.958000 | 84.160004 | 97.629997 | 9.572000 | 224.5500 |

1522 rows × 30 columns

< >

◆ Code

```
# pca1.py, pca2.py, pca3.py
import numpy as np
import pandas as pd
import pandas_datareader.data as web
from sklearn.decomposition import KernelPCA

symbols = ['ADS.DE', 'ALV.DE', 'BAS.DE', 'BAYN.DE', 'BEI.DE',
           'BMW.DE', 'CBK.DE', 'CON.DE', 'DAI.DE', 'DB1.DE',
           'DBK.DE', 'DPW.DE', 'DTE.DE', 'EOAN.DE', 'FME.DE',
           'FRE.DE', 'HEI.DE', 'HEN3.DE', 'IFX.DE', 'LHA.DE',
           'LIN.DE', 'LXS.DE', 'MRK.DE', 'MUV2.DE', 'RWE.DE',
           'SAP.DE', 'SDF.DE', 'SIE.DE', 'TKA.DE', 'VOW3.DE',
           '^GDAXI']
```



```
# start='2010-01-01', end='2015-12-31', Close Price

# data = pd.DataFrame()
# for sym in symbols:
#     data[sym] = web.DataReader(sym, data_source='yahoo', start='2010-01-01',
#                               end='2015-12-31')['Close']
# data = data.dropna()
# data.to_hdf('DAXCompAll.h5', key='data', mode='w')

data = pd.DataFrame()
data = pd.read_hdf('DAXCompAll.h5', 'data');
print(data)

data[data.columns[:6]].head()
dax = pd.DataFrame(data.pop('^GDAXI'))
```

(二)使用主成分分析

◆ 通常使用主成分分析時，是對標準化後的資料。

➤ 此資料一共有 30 個成分。顯示前 10 項。

```
dax = pd.DataFrame(data.pop('^GDAXI'))
```

```
scale_function = lambda x: (x - x.mean()) / x.std()
```

```
pca = KernelPCA().fit(data.apply(scale_function))
```

```
print(len(pca.lambdas_))
```

```
print(pca.lambdas_[:10].round())
```

```
30
```

```
[31561.  5516.  3118.  2117.  1032.   737.   290.   255.   171.   143.]
```

➤ 標準化前 10 項資料的權數，顯示如下。前六項權術，合計高達 96.60%。

```
get_we = lambda x: x / x.sum()  
print(get_we(pca.lambdas_)[:10])  
print(get_we(pca.lambdas_)[:10].sum())  
print(get_we(pca.lambdas_)[:6].sum())
```

```
[0.69167347 0.1208829  0.06833859 0.04638846 0.02262056 0.01615226  
 0.00635598 0.00558312 0.00375692 0.00312998]  
0.9848822463182842  
0.9660562465794807
```

(三)建置一個主成分指數

◆ 只使用第一項主成分來建置指數。

```
pca = KernelPCA(n_components=1).fit(data.apply(scale_function))
```

```
dax['PCA_1'] = pca.transform(data)
```

```
dax.apply(scale_function).plot(figsize=(12, 6))
```

```
plt.show()
```



◆ 使用前六項主成分，來建置指數。

```
pca = KernelPCA(n_components=6).fit(data.apply(scale_function))
```

```
pca_components = pca.transform(data)
```

```
weights = get_we(pca.lambdas_)
```

```
dax['PCA_6'] = np.dot(pca_components, weights)
```

```
dax.apply(scale_function).plot(figsize=(12, 6))
```

```
plt.show()
```



◆ DAX 報酬對 PCA 報酬迴歸分析

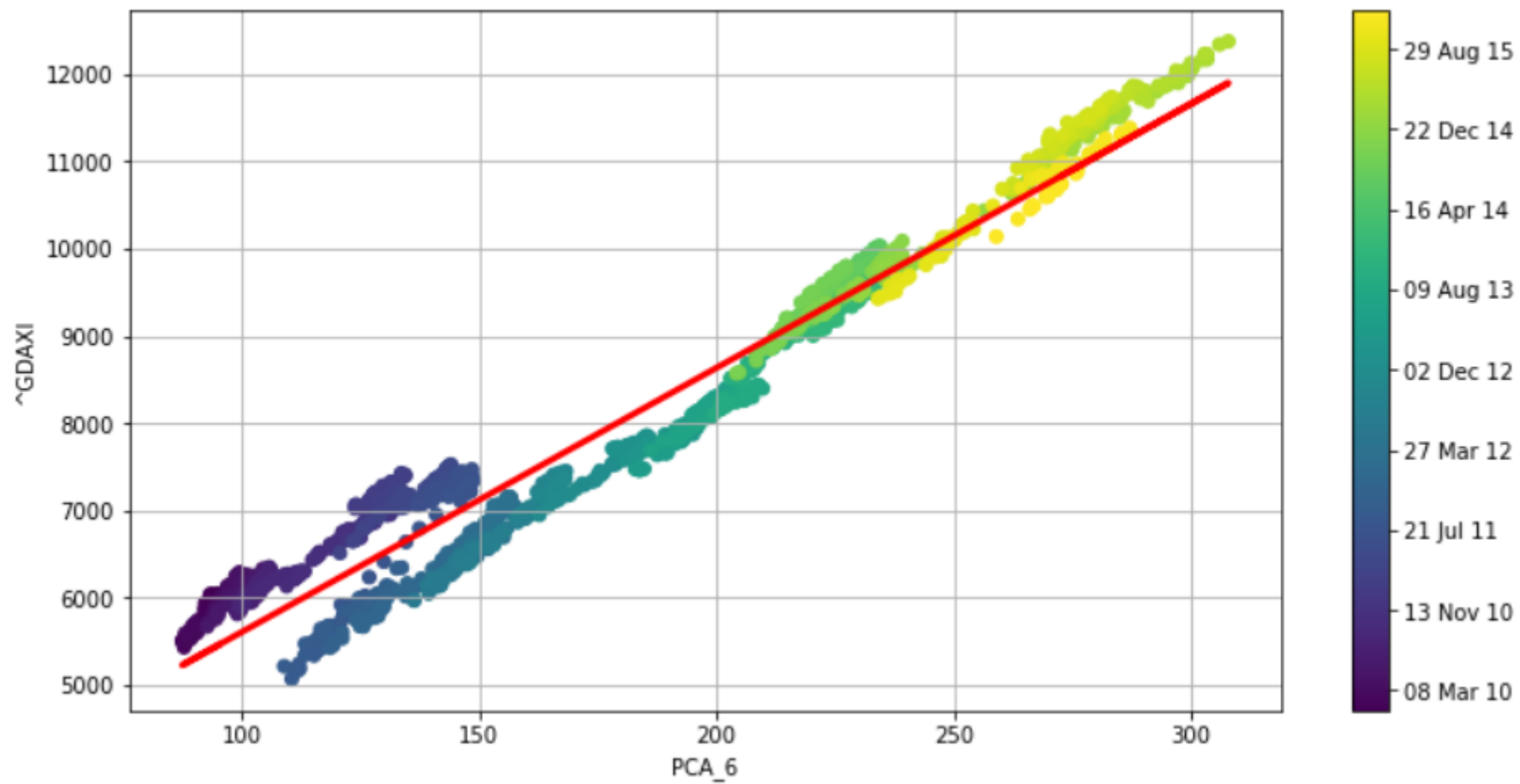
```
mpl_dates = mpl.dates.date2num(data.index.date)

plt.figure(figsize=(12, 6))

plt.scatter(dax['PCA_6'], dax['^GDAXI'], c=mpl_dates)
lin_reg = np.polyval(np.polyfit(dax['PCA_6'], dax['^GDAXI'], 1), dax['PCA_6'])

plt.plot(dax['PCA_6'], lin_reg, 'r', lw=3)
plt.grid(True)
plt.xlabel('PCA_6')
plt.ylabel('^GDAXI')

plt.colorbar(ticks=mpl.dates.DayLocator(interval=250),
             format=mpl.dates.DateFormatter('%d %b %y'))
plt.show()
```

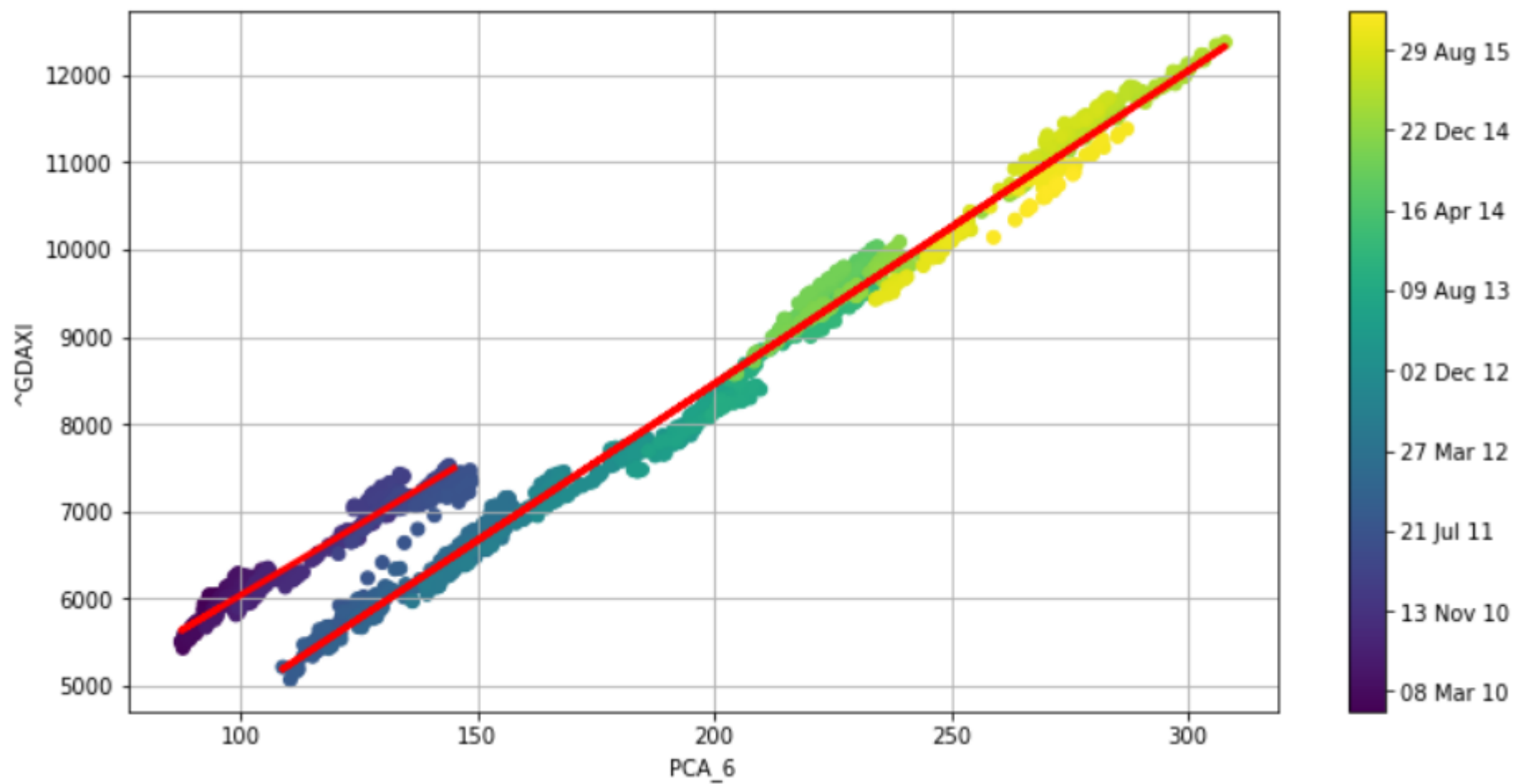



➤ 似乎有前後差異，分段在 2011 年中。

◆ 分段迴歸，2011/7/1 分段點

```
cut_date = '2011/7/1'
early_pca = dax[dax.index < cut_date]['PCA_6']
early_reg = np.polyval(np.polyfit(early_pca, dax['^GDAXI'][dax.index < cut_date], 1), early_pca)
late_pca = dax[dax.index >= cut_date]['PCA_6']
late_reg = np.polyval(np.polyfit(late_pca, dax['^GDAXI'][dax.index >= cut_date], 1), late_pca)

plt.figure(figsize=(12, 6))
plt.scatter(dax['PCA_6'], dax['^GDAXI'], c=mpl_dates)
plt.plot(early_pca, early_reg, 'r', lw=3)
plt.plot(late_pca, late_reg, 'r', lw=3)
plt.grid(True)
plt.xlabel('PCA_6')
plt.ylabel('^GDAXI')
plt.colorbar(ticks=mpl.dates.DayLocator(interval=250),
             format=mpl.dates.DateFormatter('%d %b %y'))
plt.show()
```



➤ 兩個不同的迴歸關係，同時呈現。

二、時間數列特性

- ◆ 時間數列：某一數量在一些時間跨距下(some interval)，依時間序列性下(sequentially in time)的測量。
- ◆ 時間數列分析：根據過去已發生的一序列資料，嘗試去預測未來會發生的情況。
 - 了解過去，預測未來。
- ◆ 時間數列通常具有的特性，
 - 趨勢(Trends)：方向性的變動。
 - 季節變動(Seasonal Variation)：經濟行為與氣候表現。
 - 序列相關(Serial Dependence)：金融序列最重要的特性，
 - ✓ 波動性群聚現象(Volatility Clustering)為其中之一，對量化交易影響重大。

◆ 時間數列的用途

- 預測未來數值
- 模擬序列：在了解財務時間序列的統計特性後，可以產生未來情境之模擬
 - ✓ 估計交易次數，預期交易成本，預期收益輪廓，風險輪廓
- 推論關聯性
 - ✓ 兩時間序列的關聯，

◆ 工具的選擇

- C++/Python 交易平台開發的首選工具
- R 在統計功能上有優勢
 - ✓ 非即時性分析，可以使用之

(一)R的安裝

◆ R 語言，一種自由軟體程式語言與操作環境，主要用於統計分析、繪圖、資料探勘。

➤ R 本來是由來自紐西蘭奧克蘭大學的 Ross Ihaka 和 Robert Gentleman 開發（也因此稱為 R），

✓ 現在由「R 開發核心團隊」負責開發。

➤ R 是基於 S 語言的一個 GNU 計劃專案，所以也可以當作 S 語言的一種實作，

✓ 通常用 S 語言編寫的代碼都可以不作修改的在 R 環境下執行。R 的語法是來自 Scheme。

◆ R 內建多種統計學及數字分析功能。

➤ R 的功能也可以透過安裝套件（Packages，用戶撰寫的功能）增強。

✓ 因為 S 的血緣，R 比其他統計學或數學專用的編程語言有更強的物件導向（物件導向程式設計）功能。

◆ R 的另一強項是繪圖功能，製圖具有印刷的素質，也可加入數學符號。

➤ 雖然 R 主要用於統計分析或者開發統計相關的軟體，但也有人用作矩陣計算。


✓ 其分析速度可媲美專用於矩陣計算的自由軟體 GNU Octave 和商業軟體 MATLAB。

✓ 讀者可以自行到台大網站，<http://cran.csie.ntu.edu.tw/>，下載安裝程式。

檔案 (F) 編輯 (E) 檢視 (V) 歷史 (S) 書籤 (B) 工具 (I) 說明 (H)

The Comprehensive R Archl X +

cran.csie.ntu.edu.tw 133%



The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2018-04-23, Joy in Playing) [R-3.5.0.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing

CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

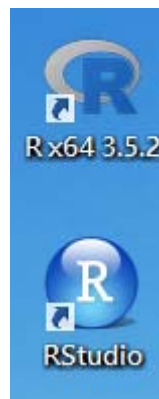
About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

◆ 安裝完成後，桌面上出現下圖左側的 R 程式圖案，執行之，出現 R 的對話執行視窗。

➤ 右側的圖案是另一個免費的程式，RStudio，它是一個整合式 R 語言開發環境。

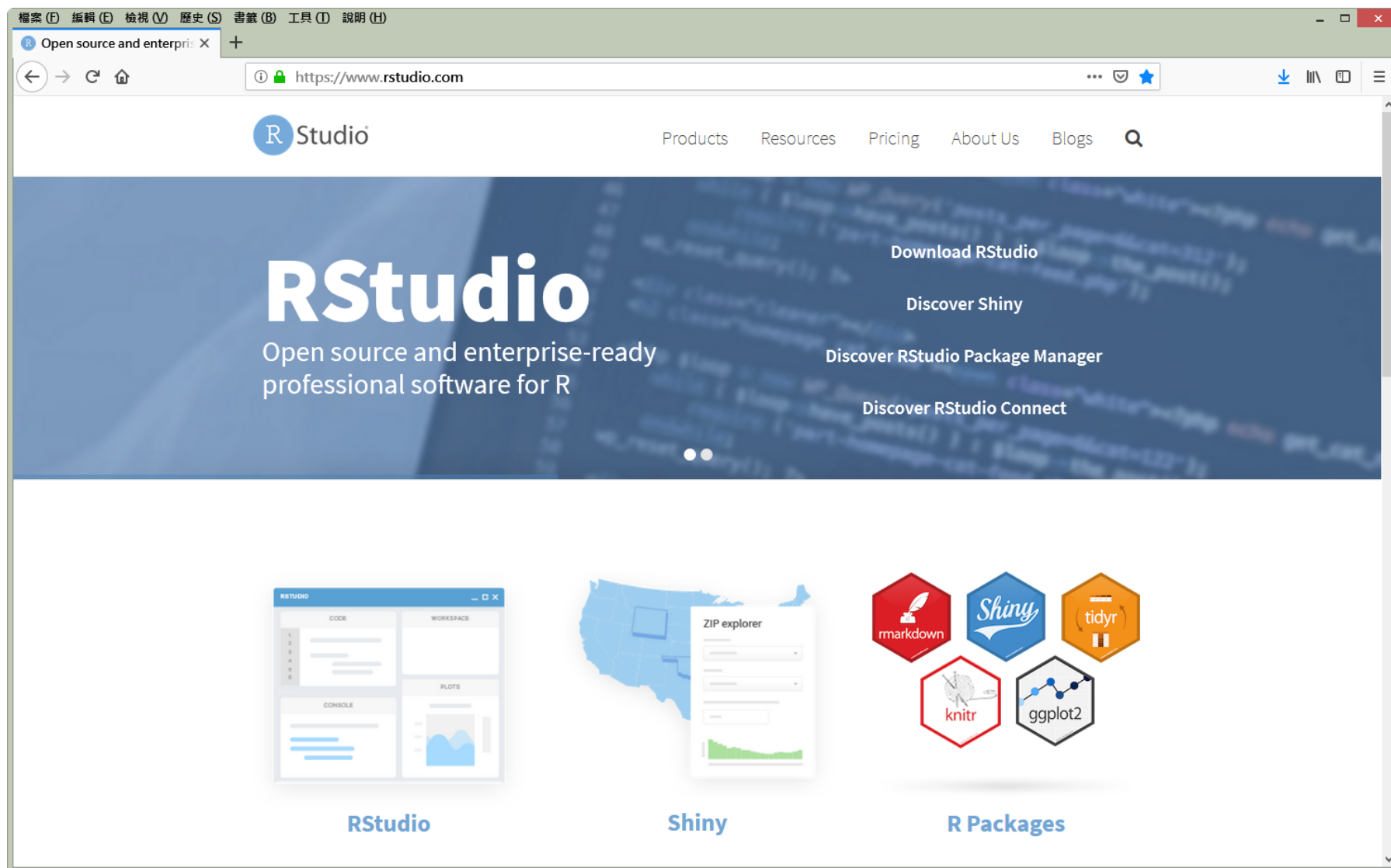


➤ R 的主控制台(R Console)畫面相當簡潔，在提示符號 '>' 後，直接打入指令，便可執行。

✓ 如果要結束，打入 `q()`，按 Enter 鍵便可結束之。



◆ R Studio 的網址為，<http://www.rstudio.com/>，讀者可下載免費版本使用。



檔案 (F)編輯 (E)檢視 (V)歷史 (S)書籤 (B)工具 (I)說明 (H)

Download RStudio - RStu...

←→↻🏠

🔒https://www.rstudio.com/products/rstudio/download/#download

⋮🔒🌟

RStudio

ProductsResourcesPricingAbout UsBlogs🔍

RStudio Desktop 1.1.463 — Release Notes

RStudio requires R 3.0.1+. If you don't already have R, download it [here](#).

Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.

Installers for Supported Platforms

| Installers | Size | Date | MD5 |
|--|---------|------------|----------------------------------|
| RStudio 1.1.463 - Windows Vista/7/8/10 | 85.8 MB | 2018-10-29 | 58b3d796d8cf96fb8580c62f46ab64d4 |
| RStudio 1.1.463 - Mac OS X 10.6+ (64-bit) | 74.5 MB | 2018-10-29 | a79032ba4d7daaa86a8da01948278d94 |
| RStudio 1.1.463 - Ubuntu 12.04-15.10/Debian 8 (32-bit) | 89.3 MB | 2018-10-29 | 8a6755fa9fae2bafce289df3358aaf63 |
| RStudio 1.1.463 - Ubuntu 12.04-15.10/Debian 8 (64-bit) | 97.4 MB | 2018-10-29 | bc50d6bd34926c1cc3ae4a209d67d649 |
| RStudio 1.1.463 - Ubuntu 16.04+/Debian 9+ (64-bit) | 65 MB | 2018-10-29 | cf659db18619cc78d1592fefaa7c753 |
| RStudio 1.1.463 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit) | 88.1 MB | 2018-10-29 | 742f0bad60dfeaa3281576e14ad6699e |
| RStudio 1.1.463 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit) | 90.6 MB | 2018-10-29 | c7303067a0ca99deea7e427b856952d1 |

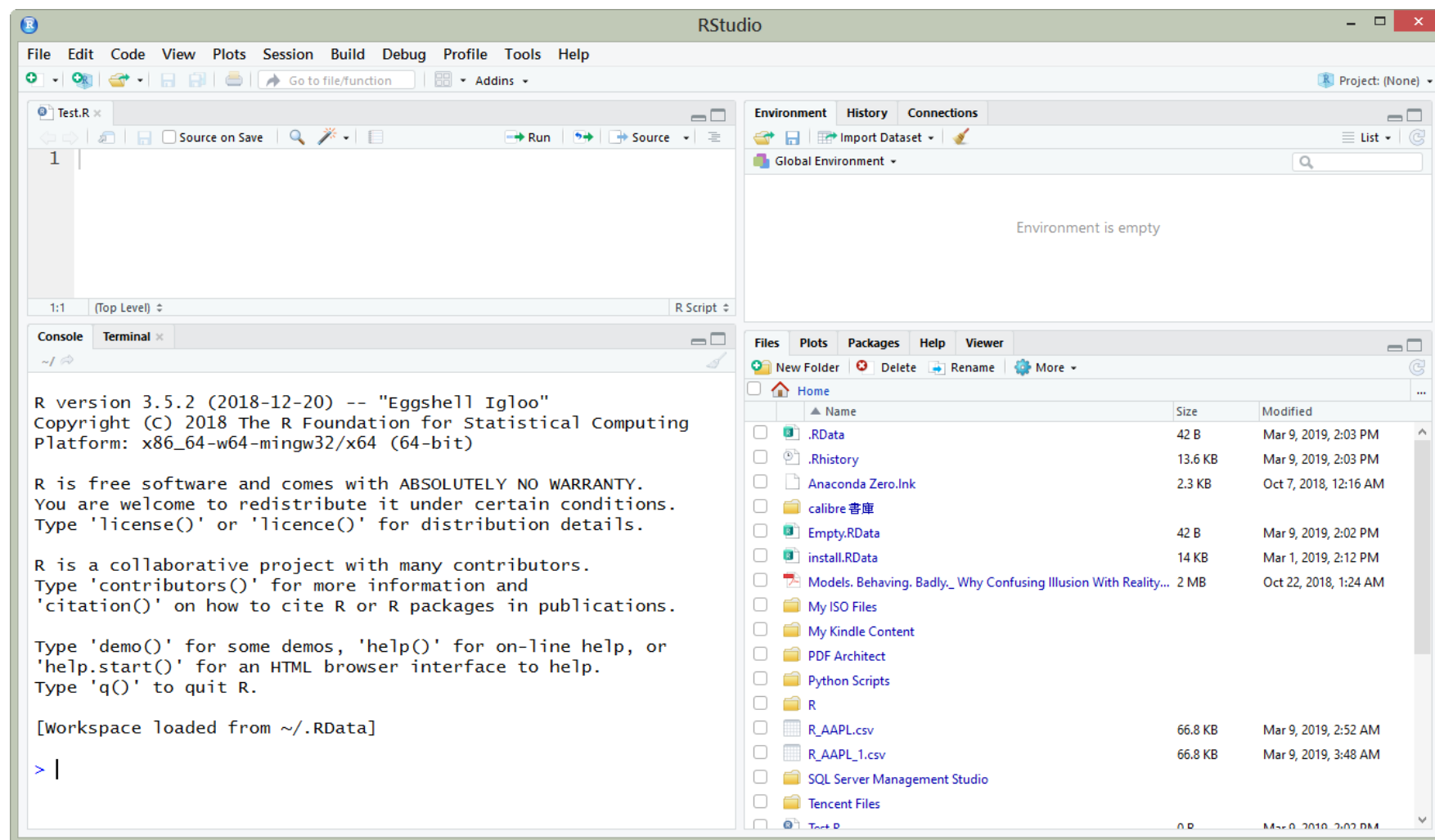
Zip/Tarballs

| Zip/tar archives | Size | Date | MD5 |
|--|----------|------------|----------------------------------|
| RStudio 1.1.463 - Windows Vista/7/8/10 | 122.9 MB | 2018-10-29 | 1eb1d7758bd4bf4bb68d4a7c3fe8d894 |
| RStudio 1.1.463 - Ubuntu 12.04-15.10/Debian 8 (32-bit) | 90 MB | 2018-10-29 | ef9242b621d36c30de9d86b808840b41 |
| RStudio 1.1.463 - Ubuntu 12.04-15.10/Debian 8 (64-bit) | 98.3 MB | 2018-10-29 | 5944b3dd118cfba46f2a6c484d768324 |
| RStudio 1.1.463 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit) | 88.8 MB | 2018-10-29 | 6bee446eeb4cc0e32967845e1f7ffe2b |
| RStudio 1.1.463 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit) | 91.4 MB | 2018-10-29 | 3f63725769b7b976fedb754eb2a19108 |

董夢雲 dongmy@ms5.hinet.net 34

◆ 安裝完成後，執行畫面如下。可以看到有較多的輔助視窗。

➤ 左側為指令執行視窗，在此輸入指令即可。



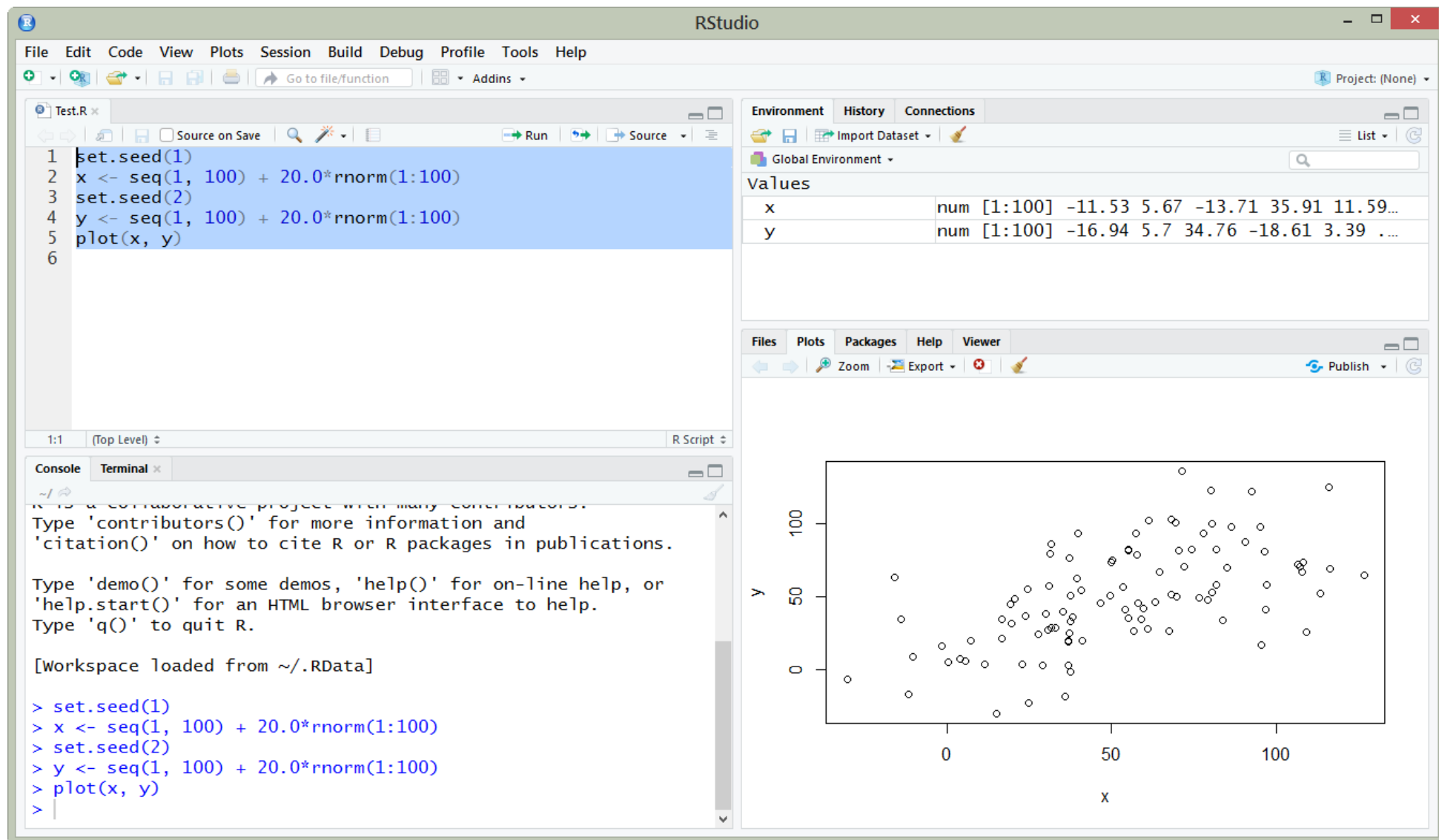
(二)相關性與序列相關

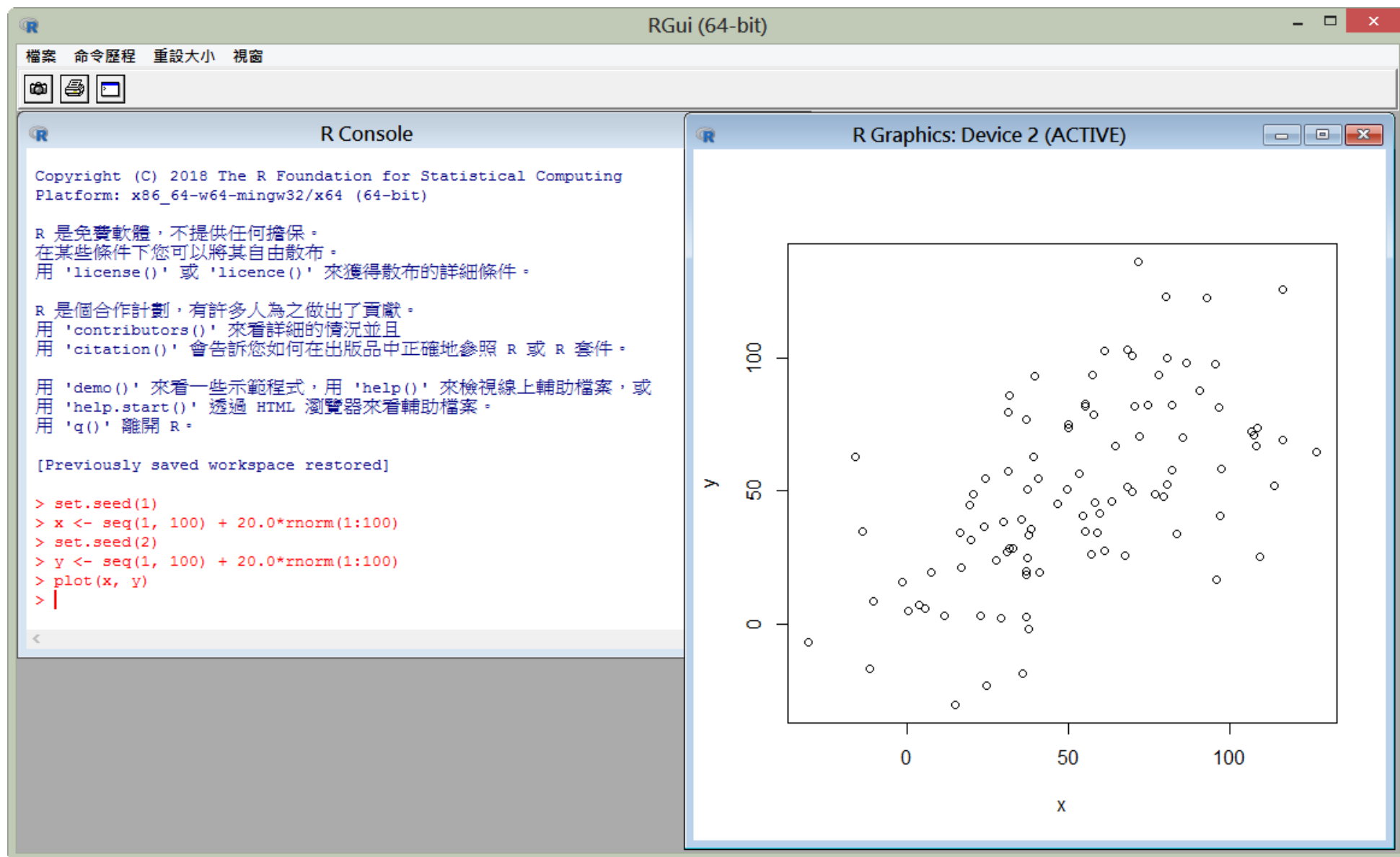
◆ 產生兩個序列，線性增加並包含常態分配雜訊。

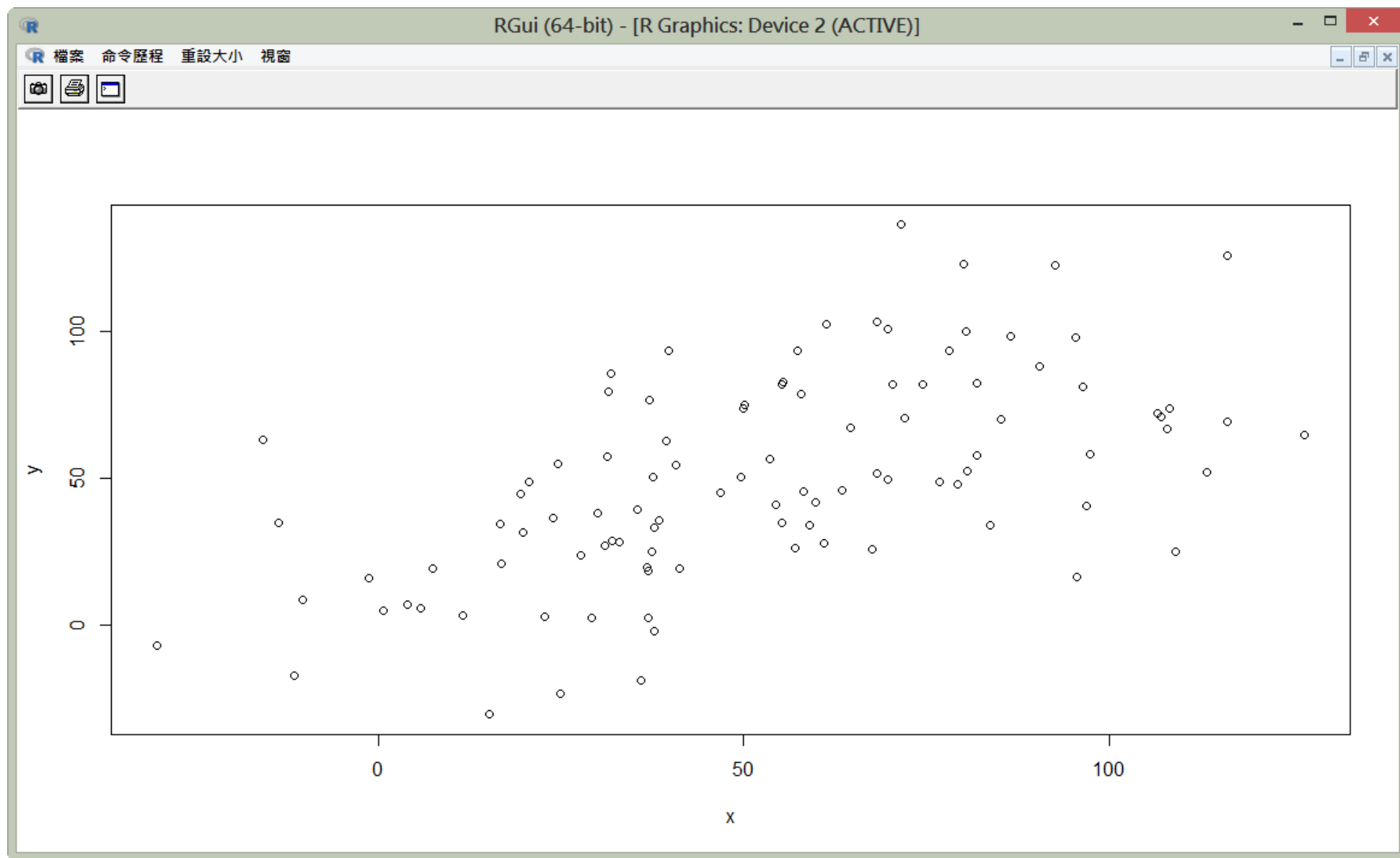
```
> Set.seed(1)
> x <- seq(1, 100) + 20.0*rnorm(1:100)

> Set.seed(2)
> y <- seq(1, 100) + 20.0*rnorm(1:100)

> plot(x, y)
```







◆ 統計量

➤ 期望值

$$\mu = E[x]$$

➤ 變異數

$$\sigma^2(x) = E[(x - \mu)^2]$$

➤ 標準差

$$\sigma(x)$$

◆ 共變異數

➤ 母體

$$\sigma(x, y) = E[(x - \mu_x)(y - \mu_y)]$$

➤ 樣本變異數

$$\text{Cov}(x, y) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

✓ 樣本平均數

\bar{x} , \bar{y}

```
> cov(x, y)
```

```
[1] 681.6859
```

◆ 相關性

➤ 母體相關性

$$\rho(x, y) = \frac{E[(x - \mu_x)(y - \mu_y)]}{\sigma_x \sigma_y} = \frac{\sigma(x, y)}{\sigma_x \sigma_y}$$

➤ 樣本相關性

$$\text{Corr}(x, y) = \frac{\text{Cov}(x, y)}{\text{sd}(x)\text{sd}(y)}$$

✓ +1：完全線性正相關

✓ 0：完全線性無關

✓ -1：完全線性負相關

> cor(x, y)

[1] 0.5796604

➤ 相當正相關

◆ 時間數列的穩態性

- 去除確定的趨勢、季節性成分之後，剩餘的殘差成分才有可能。

◆ 平均數的穩態性：

- 時間數列的平均數為常數，不為時變量。

$$\mu(t) = \mu$$

◆ 變異數的穩態性：

- 時間數列的變異數為常數，不為時變量。

$$\sigma^2(t) = \sigma^2$$

◆ 高頻財務資料常有序列相關，

- 無法真正達到平均數與變異數的穩態性。

◆ 二階穩態性：

- 序列觀察值之間的相關性，只與落差的跨距有關。

$$C_k(t) = C_k$$

◆ 時間序列自我共變異數(autocovariance)

- k 期自我共變異數

$$C_k = E[(x_t - \mu)(x_{t+k} - \mu)]$$

◆ 時間序列自我相關性(autocorrelation)

- k 期自我相關性

$$\rho_k = \frac{C_k}{\sigma^2}$$

$$\rho_0 = \frac{C_0}{\sigma^2} = \frac{E[(x_t - \mu)^2]}{\sigma^2} = \frac{\sigma^2}{\sigma^2} = 1$$

◆ 樣本自我共變異數

- k 期自我共變異數

$$c_k = \frac{1}{n} \sum_{t=1}^{n-k} (x_t - \bar{x})(x_{t+k} - \bar{x})$$

◆ 樣本自我相關性

- k 期自我相關性

$$r_k = \frac{c_k}{c_0}$$

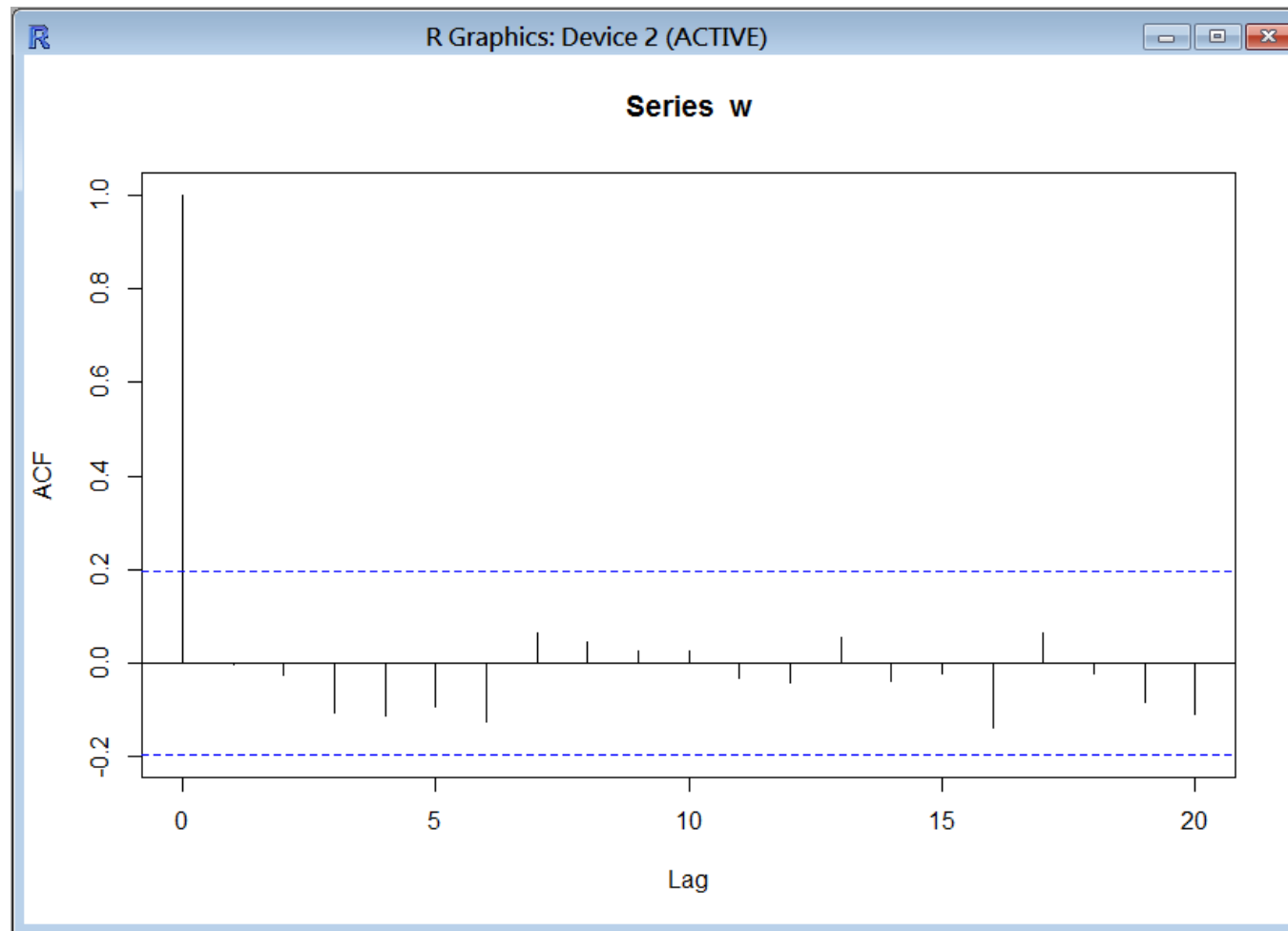
◆ Correlogram(自我相關函數圖)

- 自我相關函數的作圖
- 越過上下兩條點線，代表 5%顯著水準，拒絕數值為零的虛無假設。

◆ Random

```
w <- rnoem(100)
```

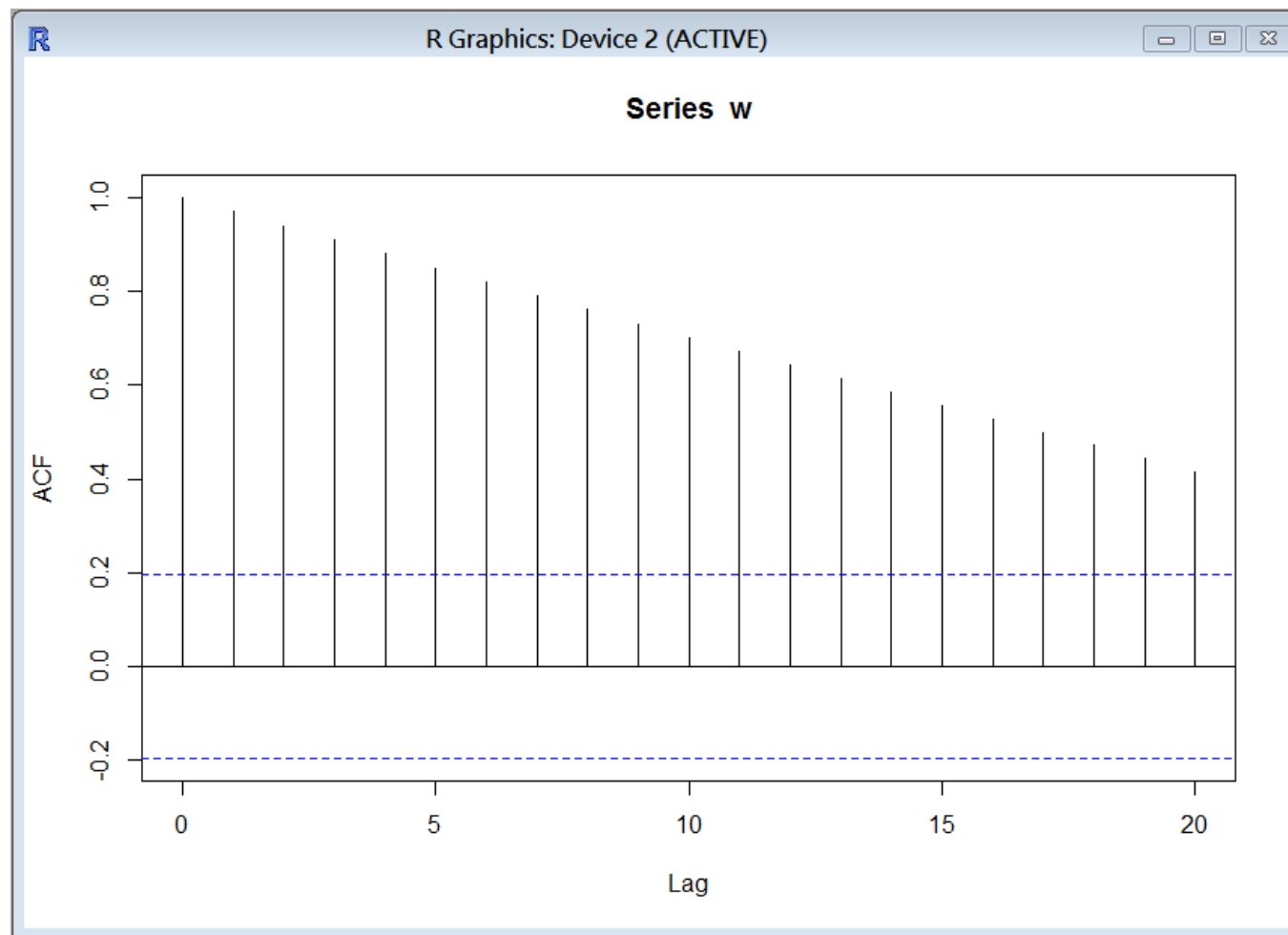
```
acf(w)
```



◆ Fixed Linear Trend

```
w <- seq(1, 100)
```

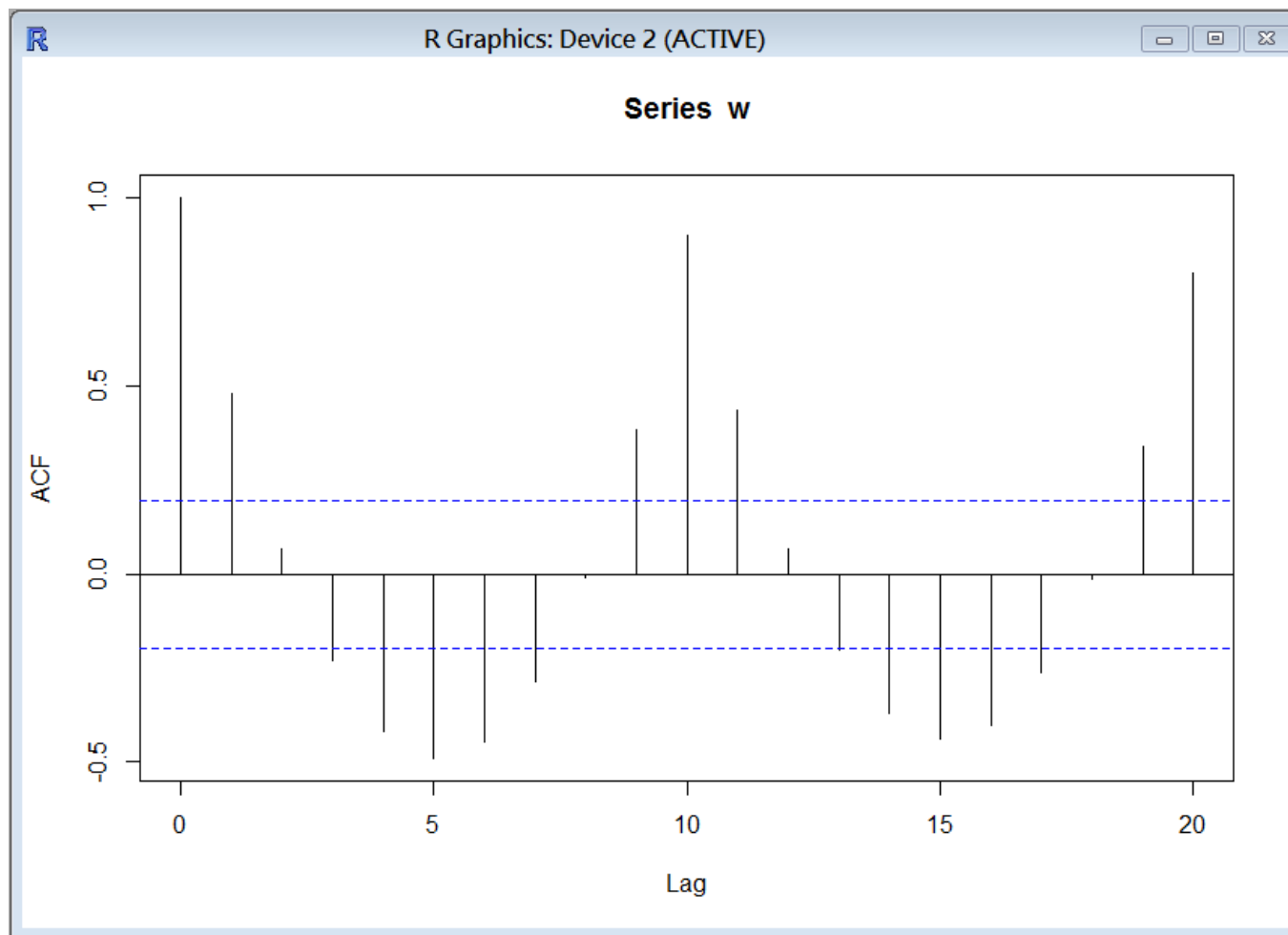
```
acf(w)
```



◆ Repeated Sequence

```
w <- rep(1:10, 10)
```

```
acf(w)
```



(三)White Noise Models & Random Walk

- ◆ 時間數列模型：一個數學模型，嘗試去解釋在時間數列中呈現的序列相關性。
- ◆ 時間數列模型化的程序：
 - 選擇最簡單又可解釋序列相關性的模型，
 - 藉由"Fitting"此模型到歷史時間數列資料，
 - 使用此模型預測時間數列的未來數值或行為模式(波動性)。
- ◆ 先了解基本時間數列的性質，
 - 白噪音(White Noise)
 - 隨機步(Random Walk)

甲、白噪音

◆ 離散白噪音(DWN, Discrete White Noise) :

- 一時間數列， $\{w_t: t=1\dots n\}$ ，每個元素 w_t ，都是獨立相同的分配，平均數為零，變異數為 σ^2 ，且不具序列相關， $Cor(w_i, w_j) = 0, \forall i \neq j$ 。
- 若這些元素都由常態分配所抽出，則此數列為高斯白噪音(Gaussian White Noise)。

◆ 此數列平均數為零，自我相關性為，

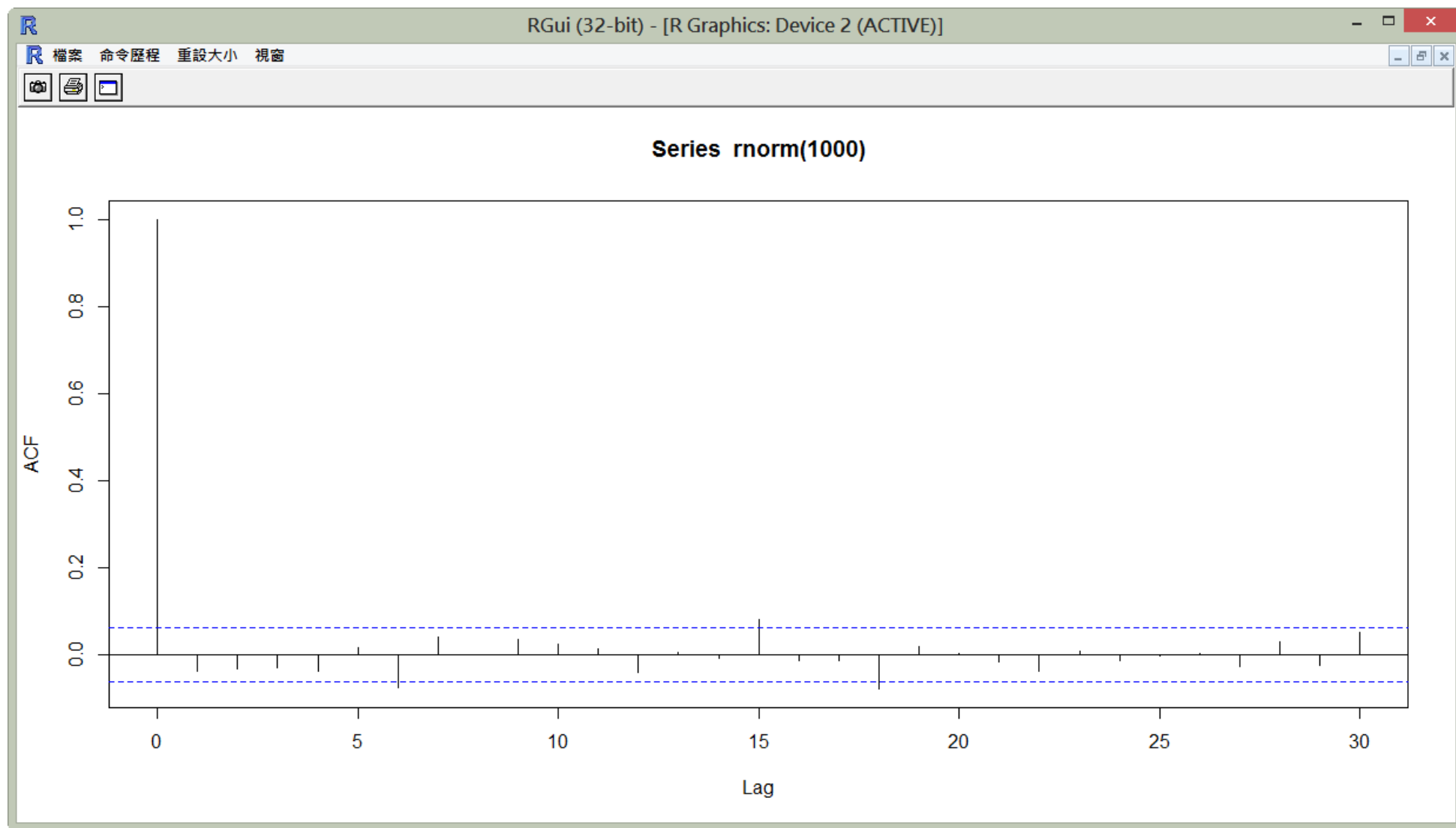
$$\mu_w = E[w_t] = 0$$

$$\rho_k = Cor(w_t, w_{t+k}) = \begin{cases} 1, & k = 0 \\ 0, & k \neq 0 \end{cases}$$

```
> set.seed(1)
> acf(rnorm(1000))
> set.seed(1)
> var(rnorm(1000, mean=0, sd=1))
```

```
[1] 1.071051
```

◆ Correlogram of Discrete White Noise



乙、隨機步

◆ 隨機步(RW, Random Walk)：一時間數列， $x_t = x_{t-1} + w_t$ ， w_t 為離散白噪音數列。

➤ 定義後移運算子(Backward Shift Operator, BSO)， B 。

$$x_t = Bx_t + w_t = x_{t-1} + w_t$$

$$x_{t-1} = Bx_{t-1} + w_{t-1} = x_{t-2} + w_{t-1}$$

...

$$x_t = (1 + B + B^2 + \dots)w_t = w_t + w_{t-1} + w_{t-2} + \dots$$

$$x_n = \sum_{i=0}^n B^i x_n = w_n + w_{n-1} + w_{n-2} + \dots + w_1 + x_0$$

$$x_0 = 0_t$$

$$x_n = \sum_{i=1}^n w_i$$

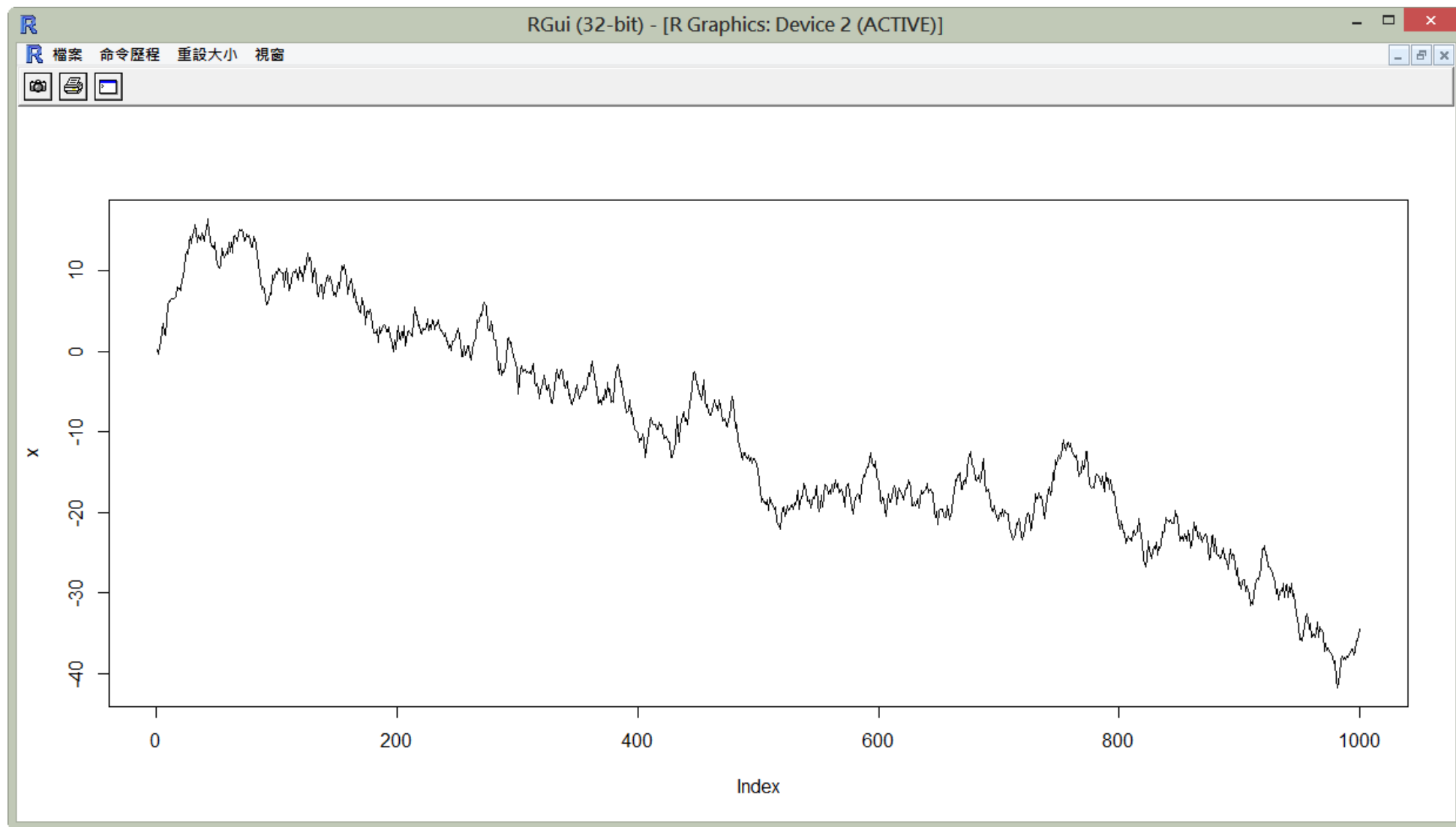
◆ 此數列為非定態的，

$$\mu_x = E[x_t] = 0 ,$$

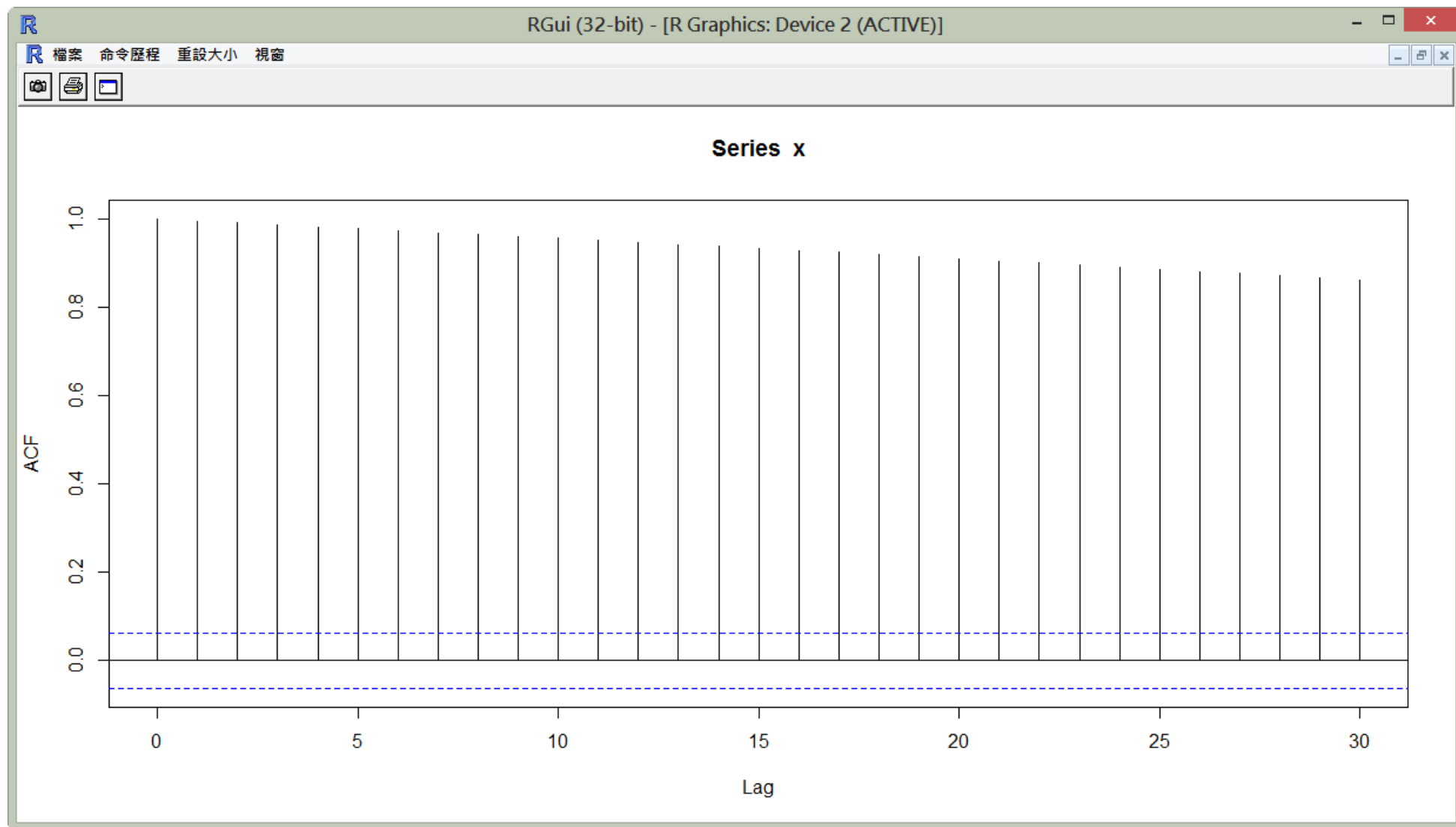
$$\gamma_k = \text{Cov}(x_t, x_{t+k}) = t\sigma^2$$

$$\rho_k = \frac{\text{Cov}(x_t, x_{t+k})}{\sqrt{\text{Var}(x_t)\text{Var}(x_{t+k})}} = \frac{t\sigma^2}{\sqrt{t\sigma^2(t+k)\sigma^2}} = \frac{1}{\sqrt{1 + \frac{k}{t}}}$$

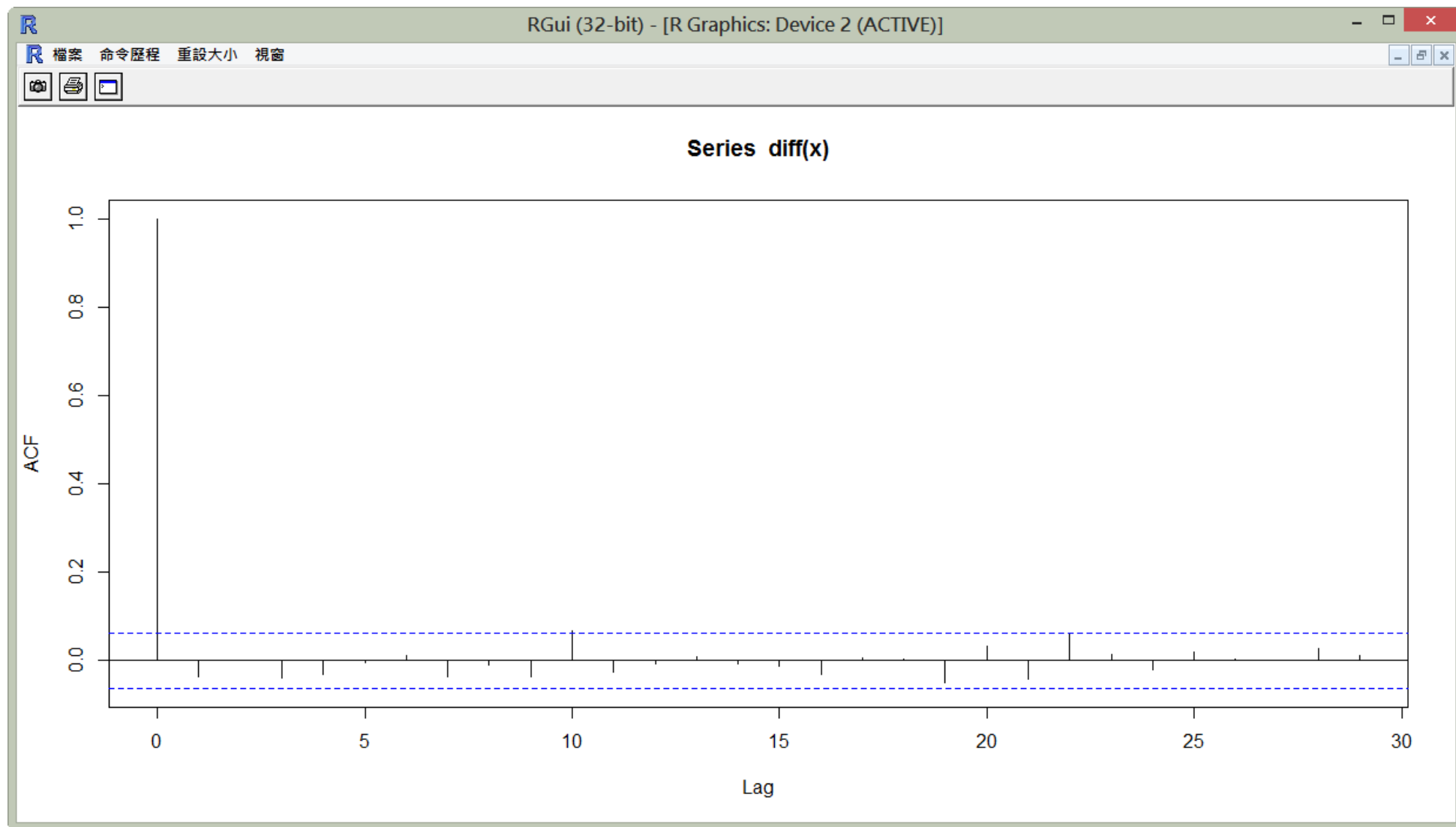
```
> set.seed(4)
> x <- w <- rnorm(1000)
> for (t in 2:1000) x[t] <- x[t-1] + w[t]
> plot(x, type='l')
```



```
> acf(x)
```




```
> acf(diff(x))
```



丙、Financial Data

◆ Microsoft & SP500

➤ 安裝

```
install.packages('quantmod')
```

➤ 執行

```
require('quantmod')
```

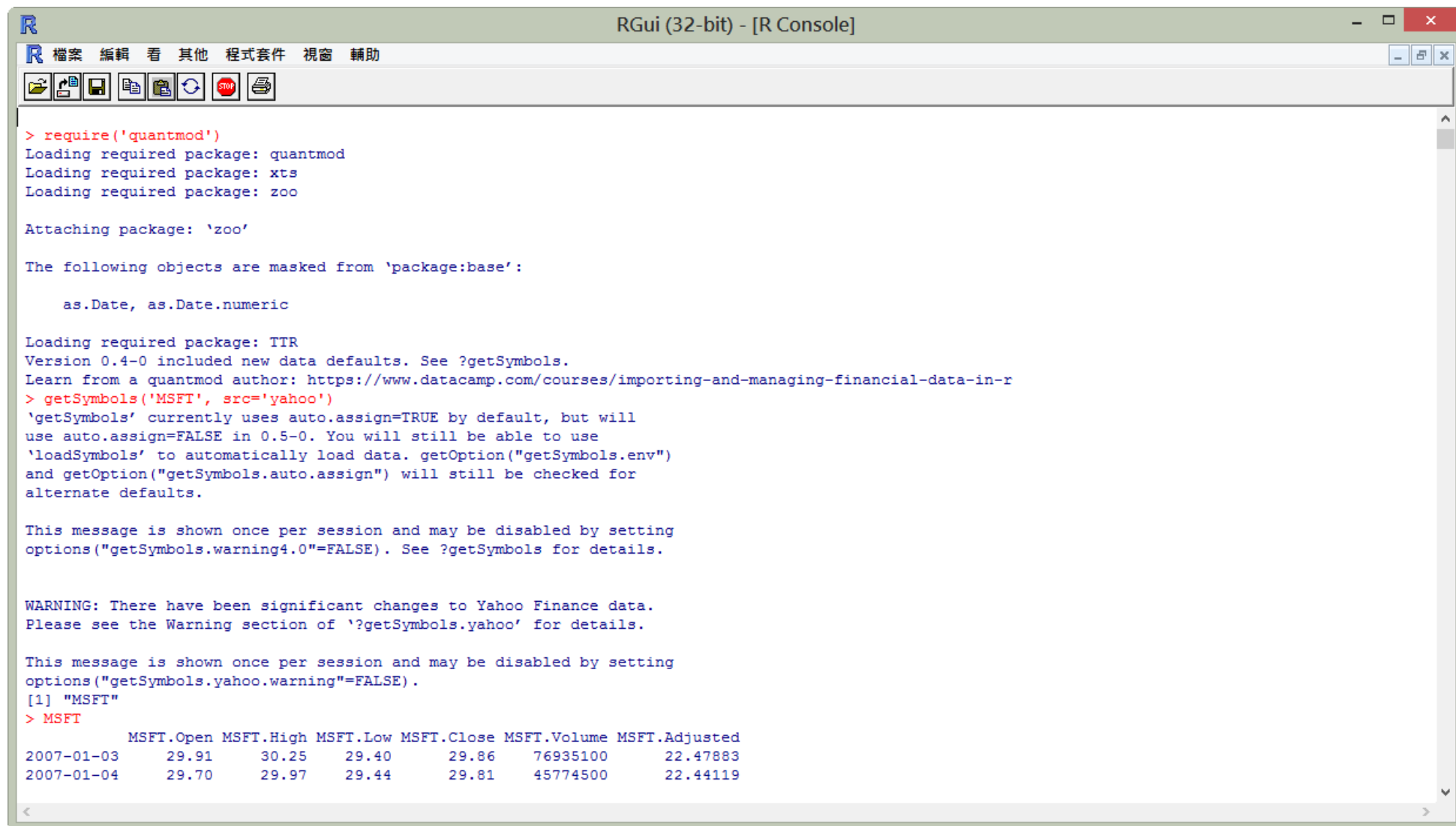
```
getSymbols('MSFT', src='yahoo')
```

```
MSFT
```

```
acf(diff(Ad(MSFT)), na.action = na.omit)
```

```
getSymbols('^GSPC', src='yahoo')
```

```
acf(diff(Ad(GSPC)), na.action = na.omit)
```



```
> require('quantmod')
Loading required package: quantmod
Loading required package: xts
Loading required package: zoo

Attaching package: 'zoo'

The following objects are masked from 'package:base':

    as.Date, as.Date.numeric

Loading required package: TTR
Version 0.4-0 included new data defaults. See ?getSymbols.
Learn from a quantmod author: https://www.datacamp.com/courses/importing-and-managing-financial-data-in-r
> getSymbols('MSFT', src='yahoo')
'getSymbols' currently uses auto.assign=TRUE by default, but will
use auto.assign=FALSE in 0.5-0. You will still be able to use
'loadSymbols' to automatically load data. getOption("getSymbols.env")
and getOption("getSymbols.auto.assign") will still be checked for
alternate defaults.

This message is shown once per session and may be disabled by setting
options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.

WARNING: There have been significant changes to Yahoo Finance data.
Please see the Warning section of '?getSymbols.yahoo' for details.

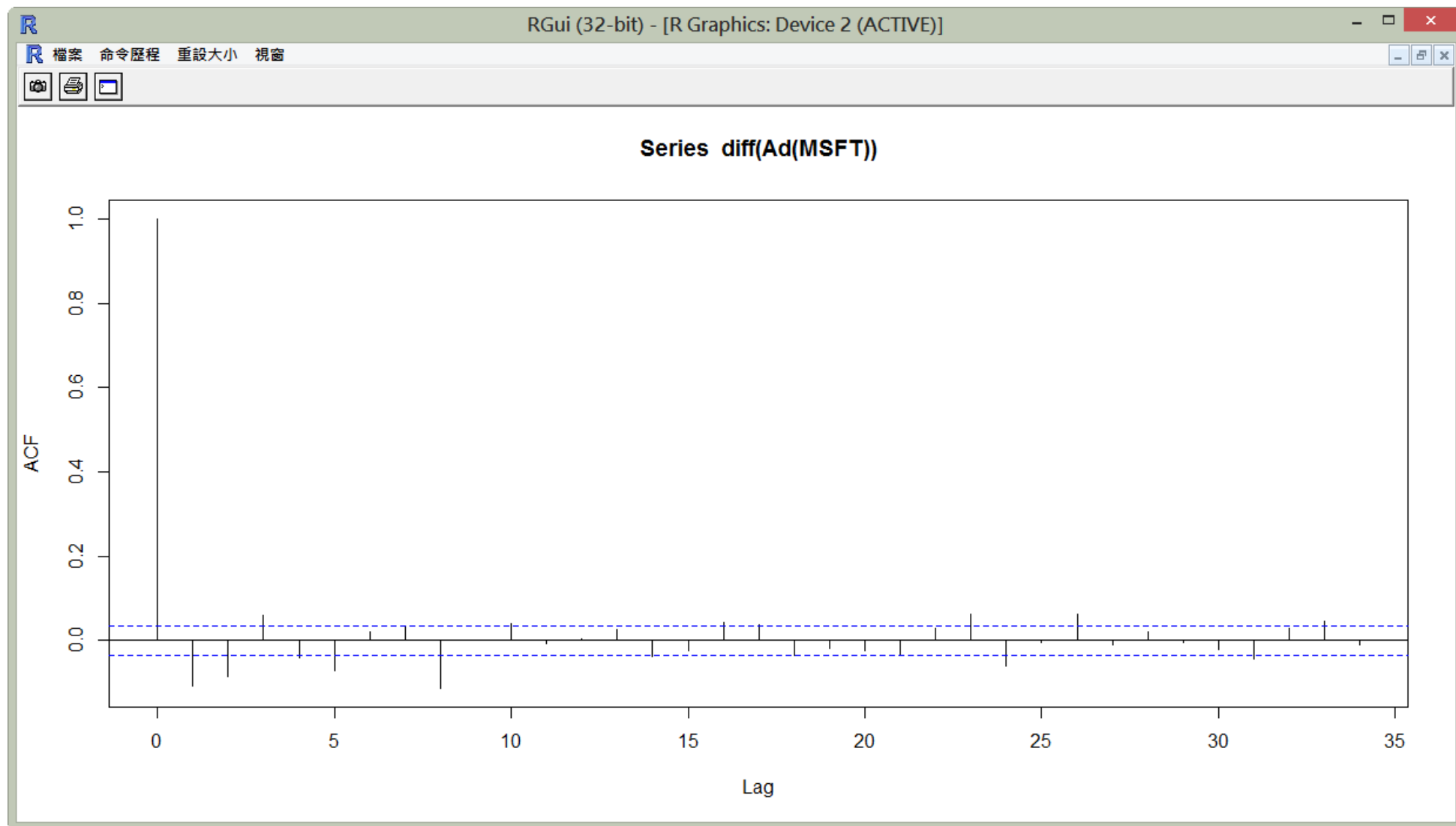
This message is shown once per session and may be disabled by setting
options("getSymbols.yahoo.warning"=FALSE).
[1] "MSFT"
> MSFT
      MSFT.Open MSFT.High MSFT.Low MSFT.Close MSFT.Volume MSFT.Adjusted
2007-01-03    29.91    30.25   29.40    29.86   76935100    22.47883
2007-01-04    29.70    29.97   29.44    29.81   45774500    22.44119
```

RGui (32-bit) - [R Console]

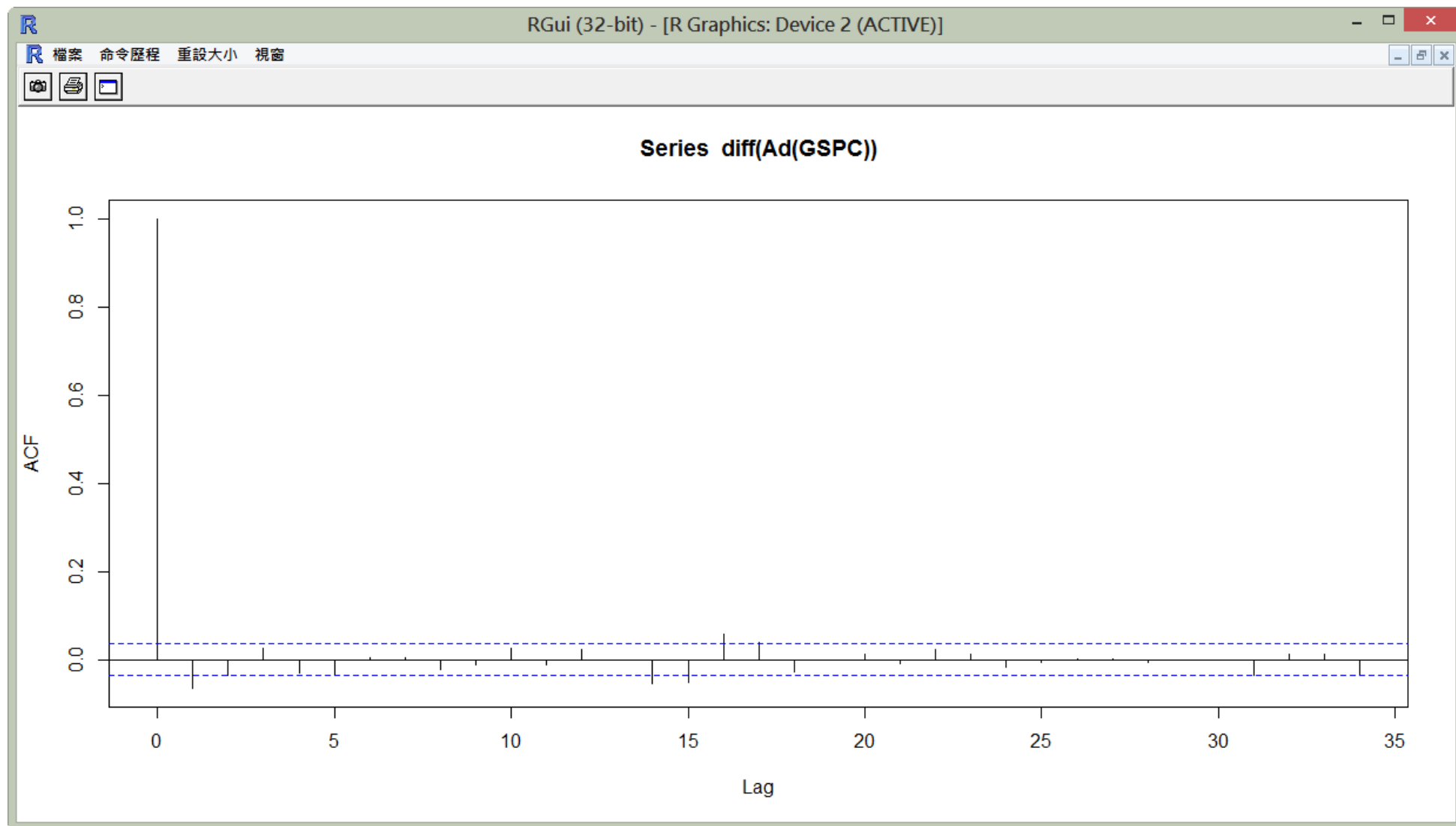
檔案 編輯 看 其他 程式套件 視窗 輔助

| | | | | | | |
|------------|--------|--------|--------|--------|----------|-----------|
| 2019-01-07 | 101.64 | 103.27 | 100.98 | 102.06 | 35656100 | 101.62598 |
| 2019-01-08 | 103.04 | 103.97 | 101.71 | 102.80 | 31514400 | 102.36284 |
| 2019-01-09 | 103.86 | 104.88 | 103.24 | 104.27 | 32280800 | 103.82658 |
| 2019-01-10 | 103.22 | 103.75 | 102.38 | 103.60 | 30067600 | 103.15943 |
| 2019-01-11 | 103.19 | 103.44 | 101.64 | 102.80 | 28314200 | 102.36284 |
| 2019-01-14 | 101.90 | 102.87 | 101.26 | 102.05 | 28437100 | 101.61603 |
| 2019-01-15 | 102.51 | 105.05 | 101.88 | 105.01 | 31587600 | 104.56344 |
| 2019-01-16 | 105.26 | 106.26 | 104.96 | 105.38 | 29853900 | 104.93186 |
| 2019-01-17 | 105.00 | 106.63 | 104.76 | 106.12 | 28393000 | 105.66872 |
| 2019-01-18 | 107.46 | 107.90 | 105.91 | 107.71 | 37427600 | 107.25195 |
| 2019-01-22 | 106.75 | 107.10 | 104.86 | 105.68 | 32371300 | 105.23059 |
| 2019-01-23 | 106.12 | 107.04 | 105.34 | 106.71 | 25874300 | 106.25621 |
| 2019-01-24 | 106.86 | 107.00 | 105.34 | 106.20 | 23164800 | 105.74837 |
| 2019-01-25 | 107.24 | 107.88 | 106.20 | 107.17 | 31225600 | 106.71425 |
| 2019-01-28 | 106.26 | 106.48 | 104.66 | 105.08 | 29476700 | 104.63314 |
| 2019-01-29 | 104.88 | 104.97 | 102.17 | 102.94 | 31490500 | 102.50224 |
| 2019-01-30 | 104.62 | 106.38 | 104.33 | 106.38 | 49471900 | 105.92761 |
| 2019-01-31 | 103.80 | 105.22 | 103.18 | 104.43 | 55636400 | 103.98591 |
| 2019-02-01 | 103.78 | 104.10 | 102.35 | 102.78 | 35535700 | 102.34292 |
| 2019-02-04 | 102.87 | 105.80 | 102.77 | 105.74 | 31315100 | 105.29034 |
| 2019-02-05 | 106.06 | 107.27 | 105.96 | 107.22 | 27325400 | 106.76405 |
| 2019-02-06 | 107.00 | 107.00 | 105.53 | 106.03 | 20609800 | 105.57910 |
| 2019-02-07 | 105.19 | 105.59 | 104.29 | 105.27 | 29760700 | 104.82233 |
| 2019-02-08 | 104.39 | 105.78 | 104.26 | 105.67 | 21461100 | 105.22063 |
| 2019-02-11 | 106.20 | 106.58 | 104.97 | 105.25 | 18914100 | 104.80242 |
| 2019-02-12 | 106.14 | 107.14 | 105.48 | 106.89 | 25056600 | 106.43545 |
| 2019-02-13 | 107.50 | 107.78 | 106.71 | 106.81 | 18394900 | 106.35578 |
| 2019-02-14 | 106.31 | 107.29 | 105.66 | 106.90 | 21784700 | 106.44540 |
| 2019-02-15 | 107.91 | 108.30 | 107.36 | 108.22 | 26606900 | 107.75979 |
| 2019-02-19 | 107.79 | 108.66 | 107.78 | 108.17 | 18038500 | 107.71000 |
| 2019-02-20 | 107.86 | 107.94 | 106.29 | 107.15 | 21607700 | 107.15000 |
| 2019-02-21 | 106.90 | 109.48 | 106.87 | 109.41 | 29063200 | 109.41000 |
| 2019-02-22 | 110.05 | 111.20 | 109.82 | 110.97 | 27748500 | 110.97000 |

```
> acf(diff(Ad(MSFT)), na.action = na.omit)
> |
```



➤ 有序列相關，負值明顯。



➤ 有序列相關，但不太明顯。

三、時間數列的資料處理

◆ 在傳統的迴歸模型中，我們會假設殘差項在時間上的共變異為零。亦即，我們假設殘差項在時間上是不相關的。

➤ 如果此一前題不成立，則他們便是“自我相關的”或是“序列相關的”。

◆ 一個變量 y_t 的落後期可以表示為 y_{t-1} ，我們以一階差分來表示此變量與其落後一期之間的變動量。

$$\Delta y_t = y_t - y_{t-1}$$

➤ 如果模型存在自我相關的現象，就會導致迴歸的估計不是 BLUE。

✓ 估計結果具備不偏和一致性，但不具備有效性和漸進有效性。

✓ 變異數的估計有偏誤，相關係數絕對值愈大，偏誤愈大，導致自變數的顯著性檢定都不正確。

- ◆ 模型存在自我相關現象的一個可能原因，可能是由於錯誤設定變量的動態過程。一般的迴歸模型，本質上都是靜態的。

$$y_t = \beta_0 + \beta_1 x_{1t} + \beta_2 x_{2t} + \beta_3 x_{3t} + u_t$$

- 這些模型僅考慮變量之間同時期的關係。

- ◆ 如果我們假定因變數受到本身前期的影響，或是其他自變數前期的影響，則我們產生的模型變成為一個動態的模型。

$$y_t = \beta_0 + \beta_1 x_{1t} + \beta_2 x_{2t} + \beta_3 x_{3t} + \gamma_0 y_{t-1} + \gamma_1 x_{1t-1} + \gamma_2 x_{2t-1} + \gamma_3 x_{3t-1} + \dots + u_t$$

- 只包含自變數前期變量的模型稱之為分布落後模型(Distributed Lag Model)，
- 包含自變數前期變量與因變數前期變量的模型稱之為自我迴歸分布落後(Autoregressive Distributed Lag Model, ADL)模型。(Chris Brooks, P.151)

◆ 金融市場中的模型需要放入落後項的可能原因為，

- 因變量的慣性作用：正相關，Trend，Overshooting。
- 過度反應：負相關，Mean Reversion。

◆ 時間數列的分析與預測是進行金融交易必不可少的一項技能，

- 本章以 Pandas 的數列分析為主，介紹一些重要的功能，
- 另外也介紹統計套件 statsmodels 的使用。

(一)使用Pandas下載與處理資料

甲、特定月份進行旅行的旅客資料

◆ 檔案名稱為 AirPassengers.csv，此時間數列為資料格式如下，

Month, Passengers

1949-01, 112

1949-02, 118

1949-03, 132

◆ 第一行為欄位名稱，資料有兩欄，分別表月份與旅客數目。

➤ 使用下面程式讀入此 CSV 檔，並顯示前 10 比資料與資料類型。

```
# Read_APData.py
```

```
from datetime import datetime
```

```
import pandas as pd
```

```
import numpy as np
```

```
data = pd.read_csv('AirPassengers.csv')
print(data.head())
```

| | Month | #Passengers |
|---|---------|-------------|
| 0 | 1949-01 | 112 |
| 1 | 1949-02 | 118 |
| 2 | 1949-03 | 132 |
| 3 | 1949-04 | 129 |
| 4 | 1949-05 | 121 |

```
print('\n Data Types:')
print(data.dtypes)
```

```
Data Types:
Month          object
#Passengers    int64
dtype: object
```

◆ 由於月份資料尚不是日期格式，不利時間數列的處理，因此將之解析轉成標準的日期格式。

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m')
data = pd.read_csv('AirPassengers.csv', parse_dates=True, index_col='Month',
                  date_parser = dateparse)
print(data.head())
```

| | #Passengers |
|------------|-------------|
| Month | |
| 1949-01-01 | 112 |
| 1949-02-01 | 118 |
| 1949-03-01 | 132 |
| 1949-04-01 | 129 |
| 1949-05-01 | 121 |

```
print(data.index)
```

```
DatetimeIndex(['1949-01-01', '1949-02-01', '1949-03-01', '1949-04-01',  
              '1949-05-01', '1949-06-01', '1949-07-01', '1949-08-01',  
              '1949-09-01', '1949-10-01',  
              ...  
              '1960-03-01', '1960-04-01', '1960-05-01', '1960-06-01',  
              '1960-07-01', '1960-08-01', '1960-09-01', '1960-10-01',  
              '1960-11-01', '1960-12-01'],  
              dtype='datetime64[ns]', name=u'Month', length=144, freq=None)
```

◆ 如此便可以日期格式來索引資料了。

- 將旅客資料取出，連索引一併產生時間數列變數 `ts`。

```
ts = data['#Passengers']  
print(ts.head(3))
```

```
          #Passengers  
Month  
1949-01-01          112  
1949-02-01          118  
1949-03-01          132
```

#1. Specific the index as a string constant:

```
print(ts['1949-01-01'])  
112
```

#2. Import the datetime library and use 'datetime' function:

```
print(ts[datetime(1949, 1 ,1)])  
112
```

```
print(ts['1949'])
```

Month

1949-01-01 112

1949-02-01 118

1949-03-01 132

1949-04-01 129

1949-05-01 121

1949-06-01 135

1949-07-01 148

1949-08-01 148

1949-09-01 136

1949-10-01 119

1949-11-01 104

1949-12-01 118

Name: #Passengers, dtype: int64

(二)使用 S&P500 資料

甲、下載 S&P500 資料

◆ 我們以 pandas 下載 S&P500 的歷史資料來進行分析。

```
#loadSP500.py
```

```
import numpy as np
```

```
import pandas as pd
```

```
import pandas_datareader.data as web
```

```
# gspc = web.DataReader('^GSPC', data_source='yahoo', start='1/1/1964', end='5/30/2016')
```

```
gspc = pd.read_csv('SP500Long.csv')
```

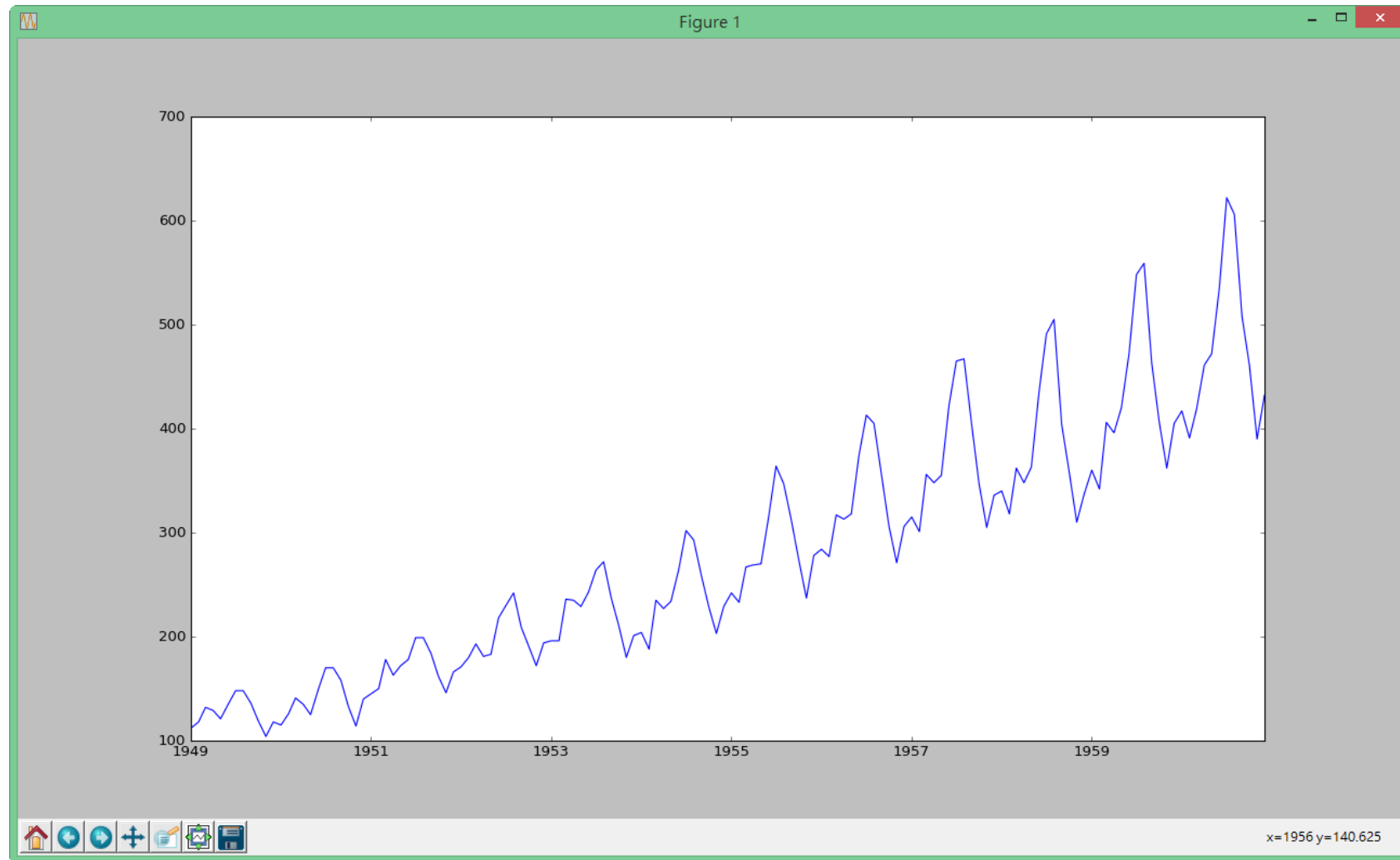
```
print(gspc.info())
```

```
print(gspc[0:10])
```

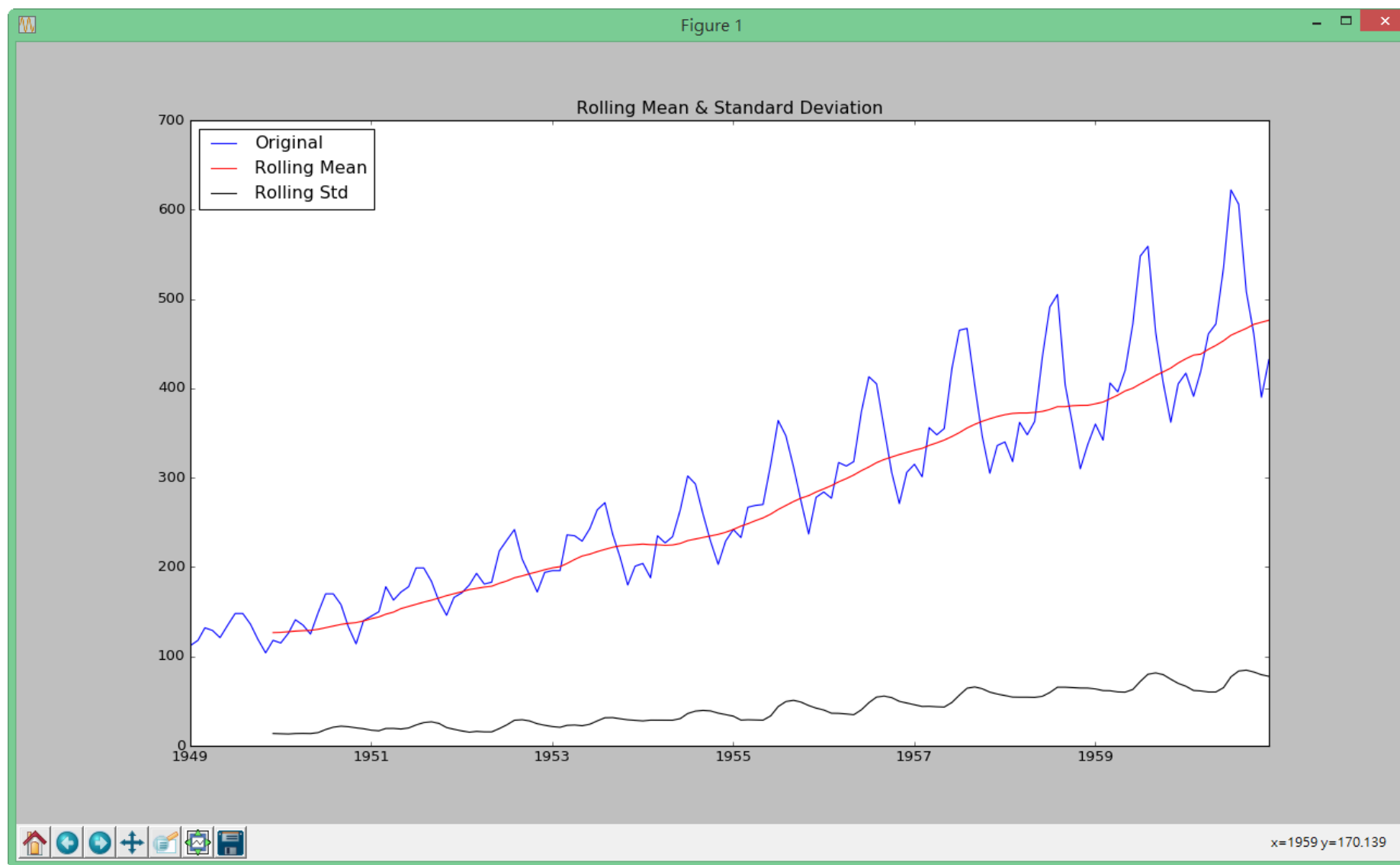
```
# gspc.to_csv('SP500Long.csv')
```


乙、穩態的檢測

- ◆ 時間數列的統計量不隨時變稱之為穩態。畫出時間數列目視可看出不為穩態的。



◆ 移動統計量與 Dickey-Fuller 檢定



Results of Dickey-Fuller Test:

| | |
|-----------------------------|------------|
| Test Statistic | 0.815369 |
| p-value | 0.991880 |
| #Lags Used | 13.000000 |
| Number of Observations Used | 130.000000 |
| Critical Value (5%) | -2.884042 |
| Critical Value (1%) | -3.481682 |
| Critical Value (10%) | -2.578770 |

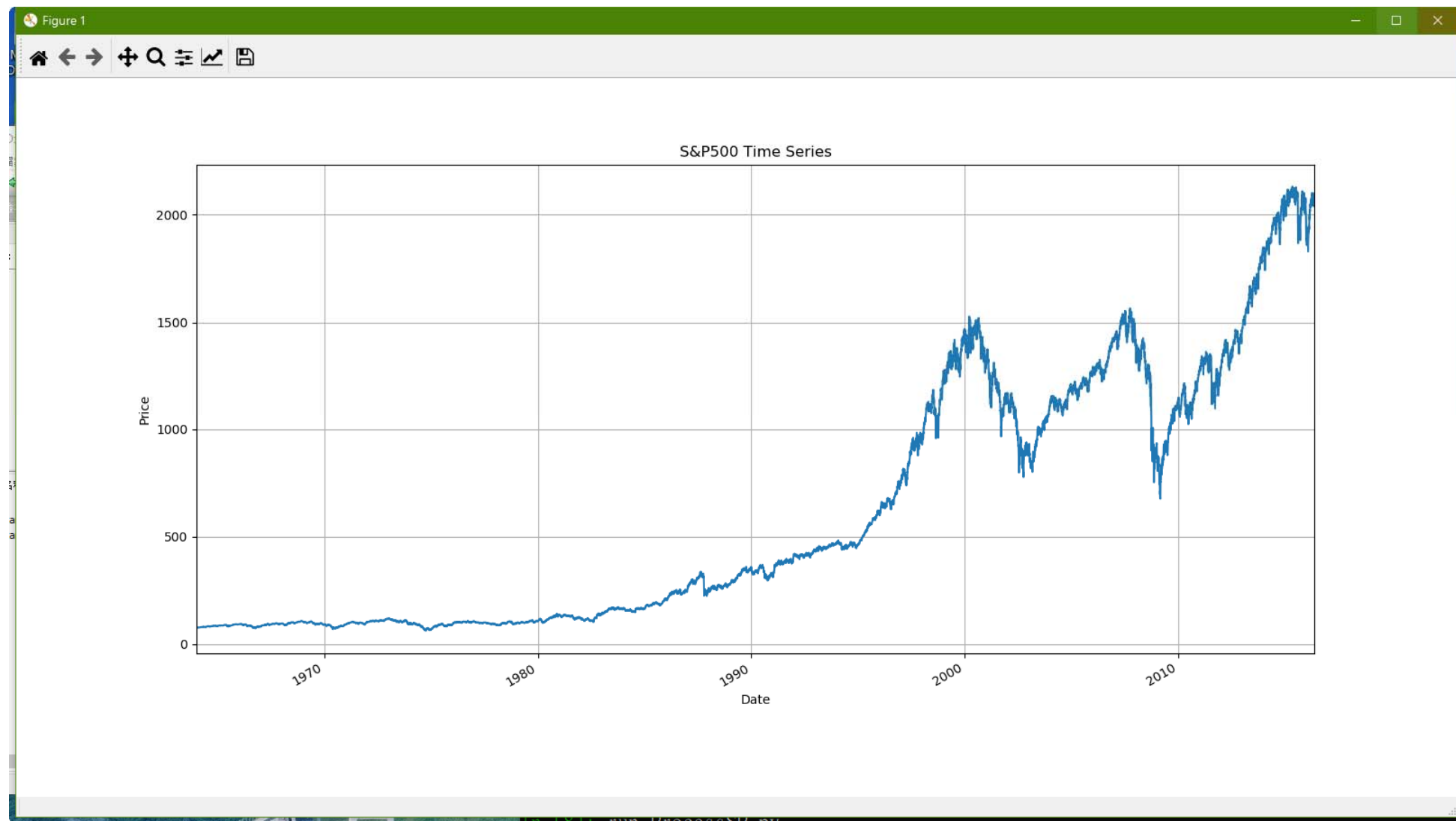
dtype: float64

◆ 計算 S&P500 波動性

ProcessSP.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

gspc = pd.read_csv('SP500Long.csv', index_col=0, parse_dates=True, infer_datetime_format=True)
gspc['Adj Close'].plot(figsize=(16, 8))
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('S&P500 Time Series')
plt.grid(True)
plt.show()
```

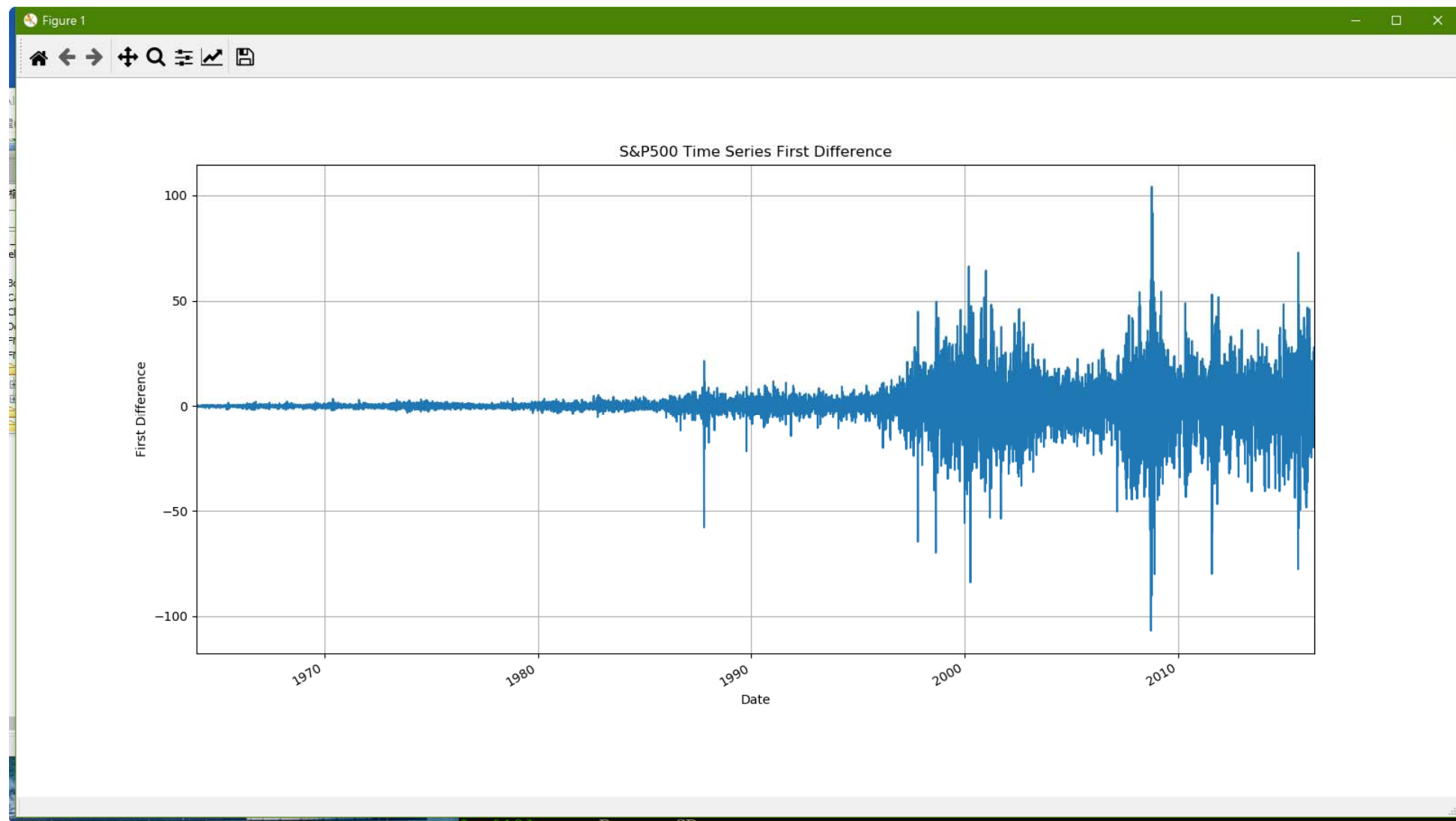


◆ 計算一次差分

```
gspc['First Difference'] = gspc['Adj Close'] - gspc['Adj Close'].shift()
gspc['First Difference'].plot(figsize=(16, 8))

plt.xlabel('Date')
plt.ylabel('First Difference')
plt.title('S&P500 Time Series First Difference')
plt.grid(True)

plt.show()
```

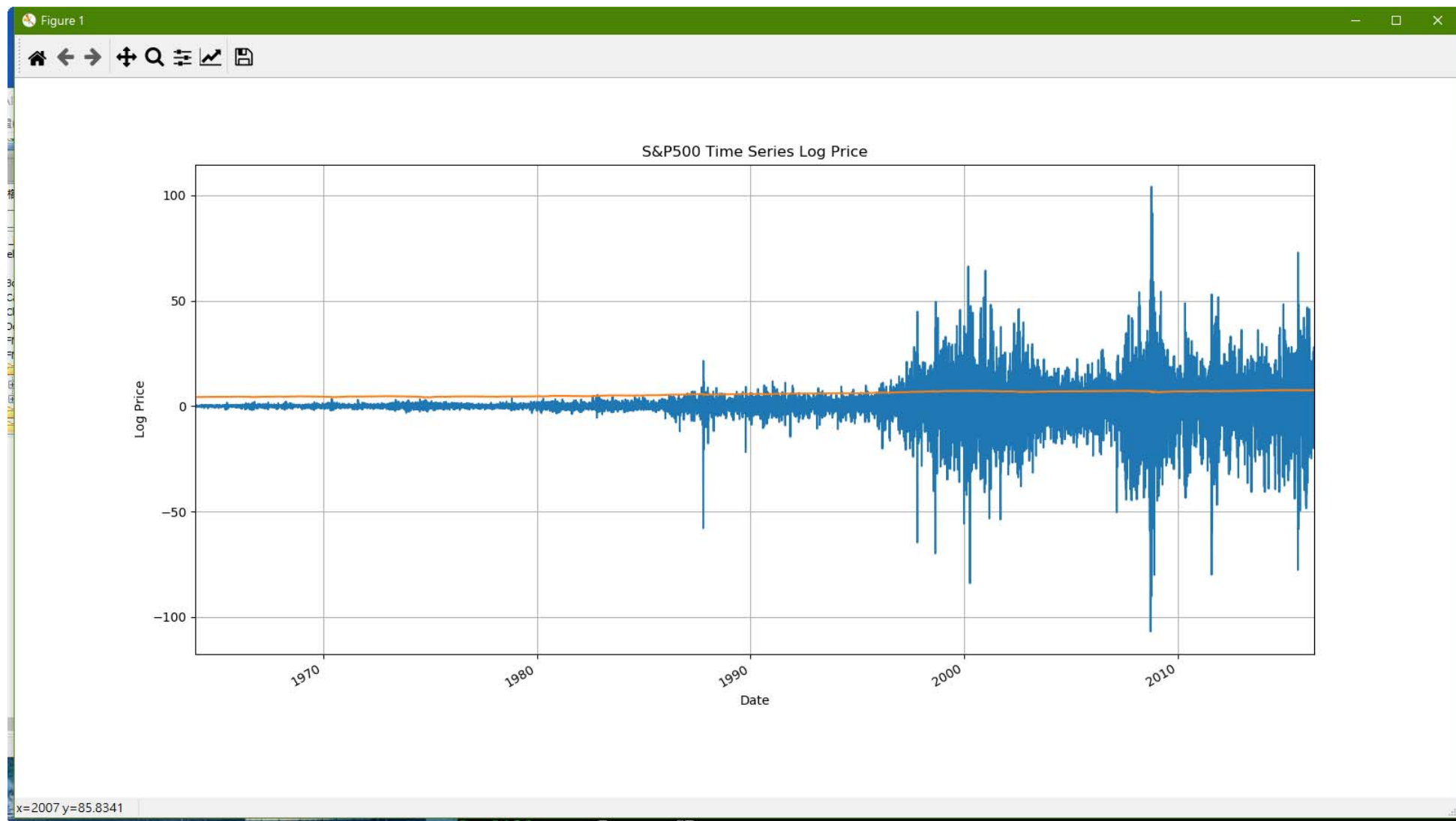


◆ Log Price

```
gspc['Natural Log'] = gspc['Adj Close'].apply(lambda x: np.log(x))
gspc['Natural Log'].plot(figsize=(16, 8))

plt.xlabel('Date')
plt.ylabel('Log Price')
plt.title('S&P500 Time Series Log Price')
plt.grid(True)

plt.show()
```

◆ 30D 收盤價變異數 v.s. 30D 對數收盤價變異數

```
gspc['Original Variance'] = gspc['Adj Close'].rolling(window=30, center=True).var()
```

```
gspc['Log Variance'] = gspc['Natural Log'].rolling(window=30, center=True).var()
```

```
fig, ax = plt.subplots(2, 1, figsize=(12, 10))
```

```
gspc['Original Variance'].plot(ax=ax[0], title='Original Variance')
```

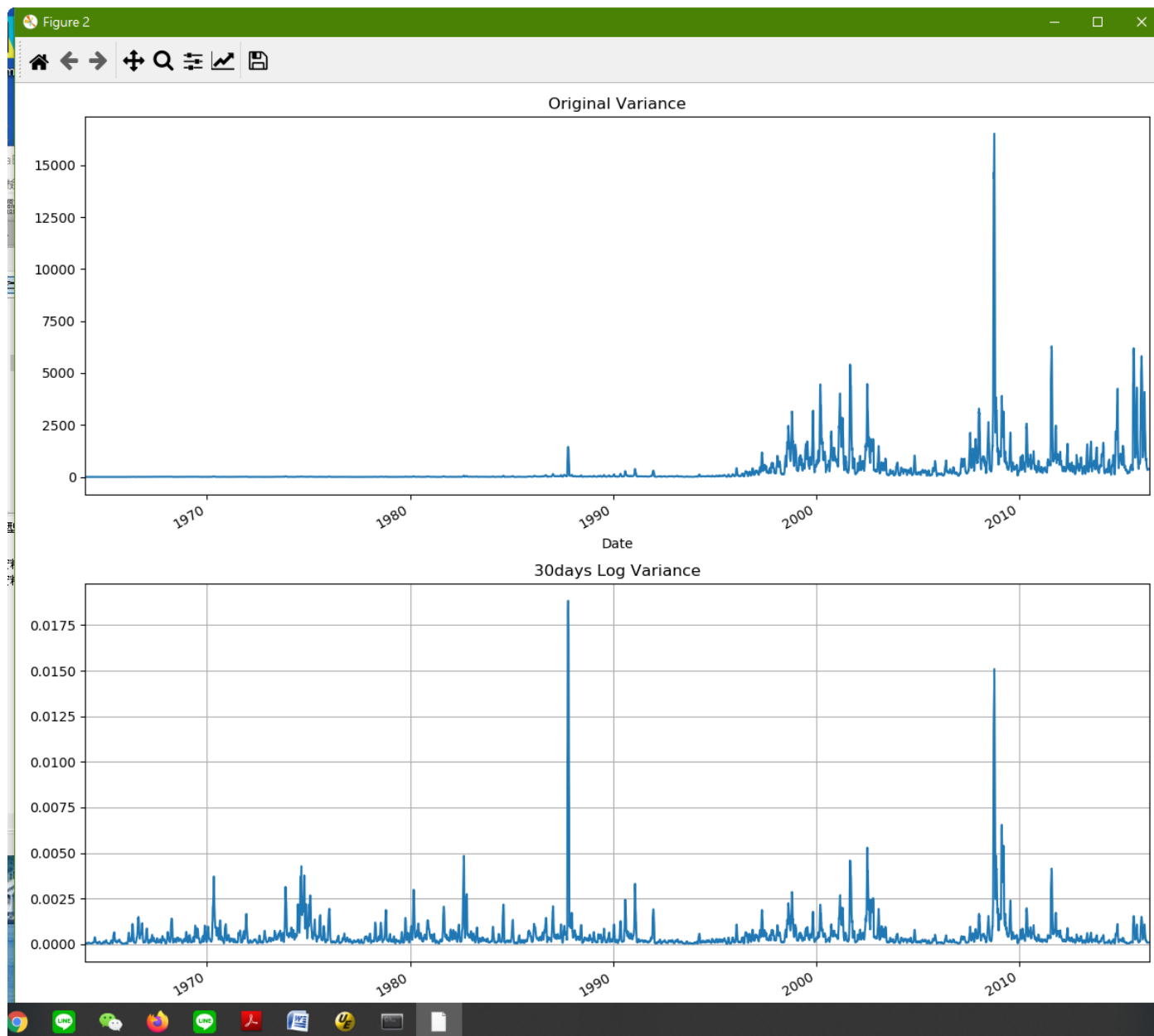
```
gspc['Log Variance'].plot(ax=ax[1], title='30days Log Variance')
```

```
fig.tight_layout()
```

```
plt.xlabel('Date')
```

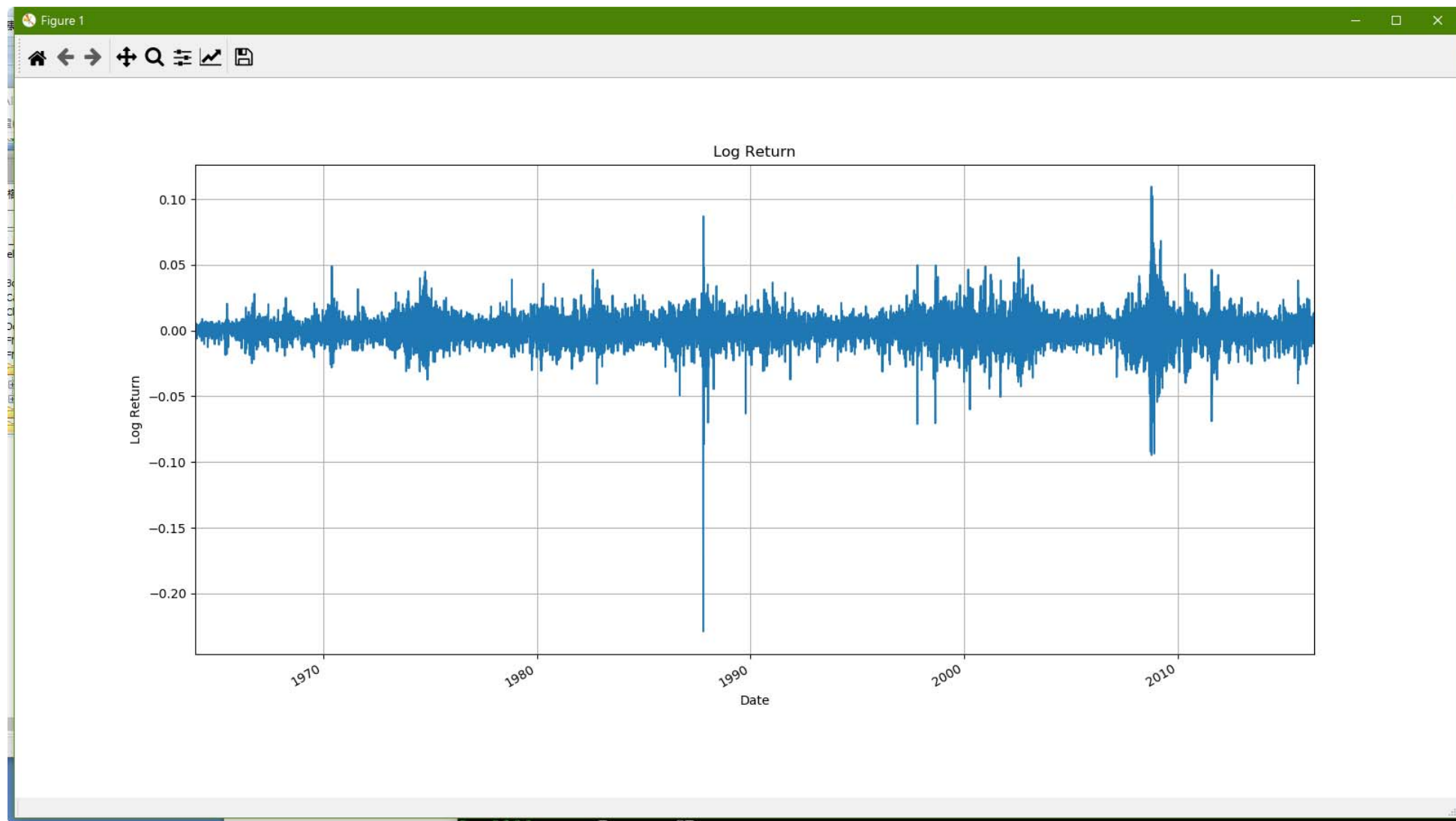
```
plt.grid(True)
```

```
plt.show()
```



◆ 對數報酬

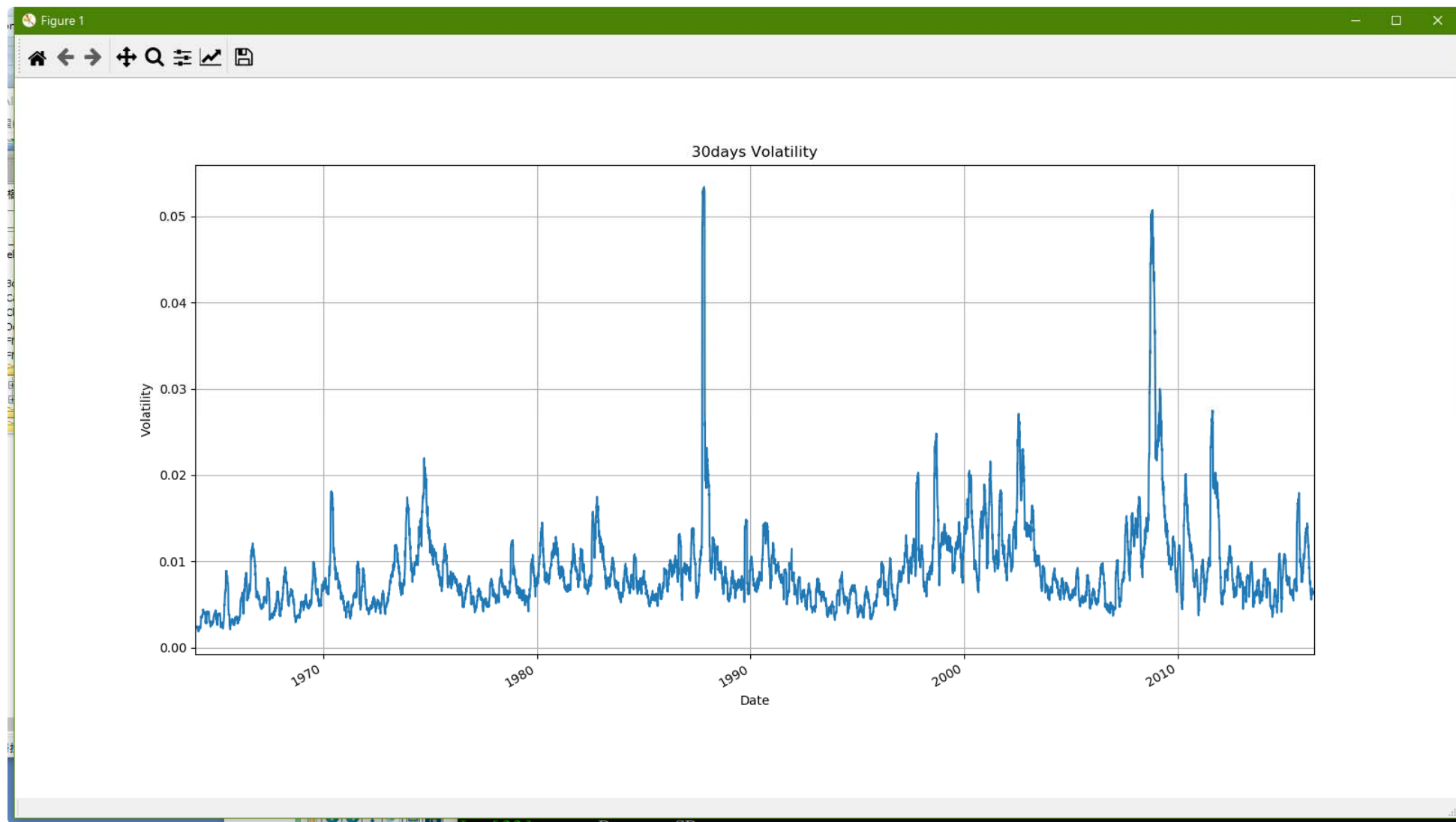
```
gspc['LogReturn'] = gspc['Natural Log'] - gspc['Natural Log'].shift()
gspc['LogReturn'].plot(figsize=(16, 8), title='Log Return')
plt.xlabel('Date')
plt.ylabel('Log Return')
plt.grid(True)
plt.show()
```



◆ 對數報酬變異數 v.s. 波動性

```
gspc['LogReturn Variance'] = gspc['LogReturn'].rolling(window=30, center=True).var()  
gspc['LogReturn Volatility'] = gspc['LogReturn'].rolling(window=30, center=True).std()  
gspc['LogReturn Volatility'].plot(figsize=(16, 8), title='30days Volatility')
```

```
plt.xlabel('Date')  
plt.ylabel('Volatility')  
plt.grid(True)  
plt.show()
```



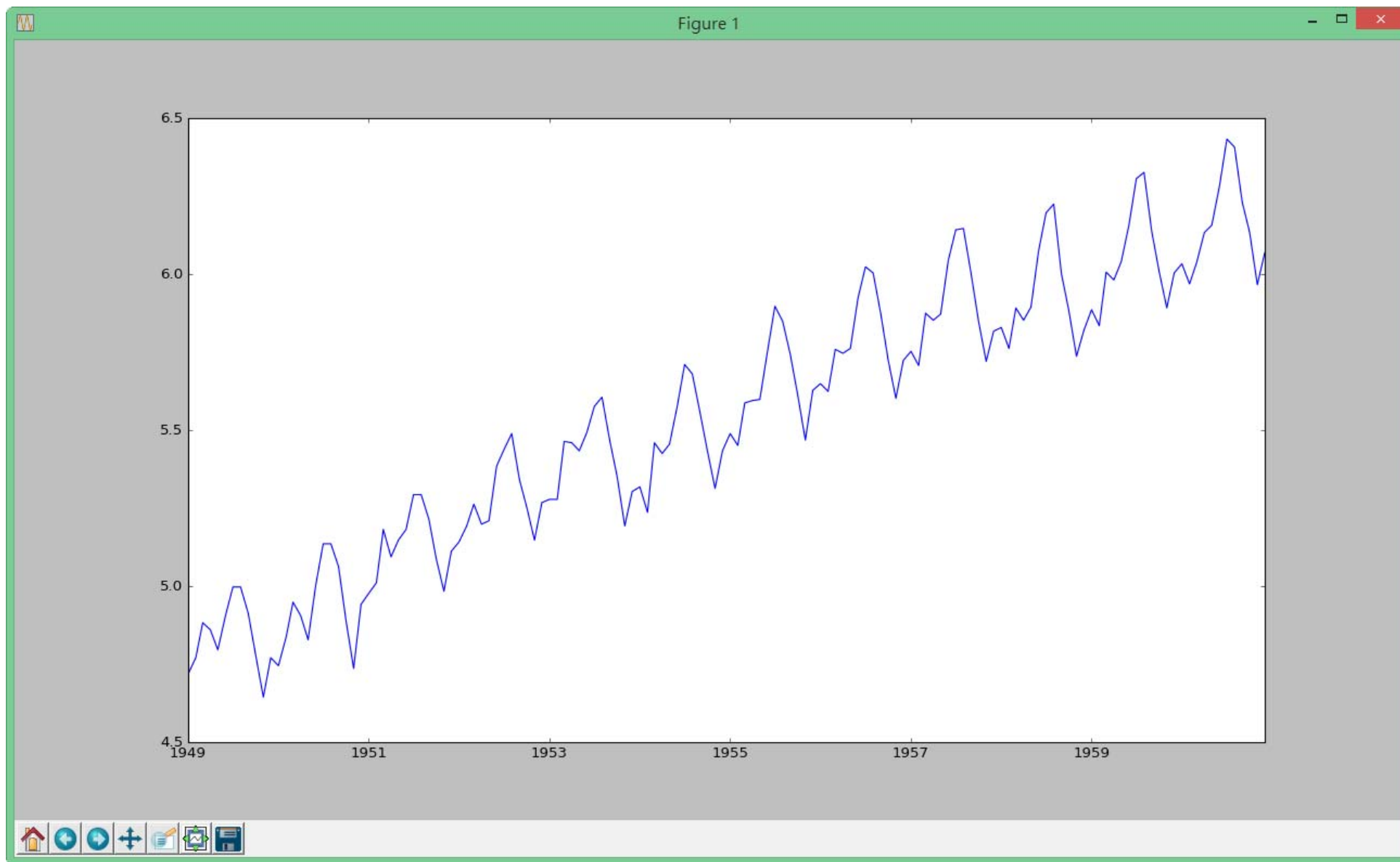
(三)產生穩態數列

- ◆ 旅客飛行資料的處理，時間數列中往往具有趨勢與季節性成分，因此無法成為穩態。
 - 使用相關技巧，去除這些成分，使之成為穩態的。

```
# Detrend_APData.py
```

```
ts_log = np.log(ts)
plt.plot(ts_log)
plt.xlabel('Date')
plt.ylabel('Number')
plt.title('Log Time Series')

plt.show()
```

甲、移動平均

```
moving_avg = ts_log.rolling(window=12, center=False).mean()
```

```
plt.plot(ts_log)
```

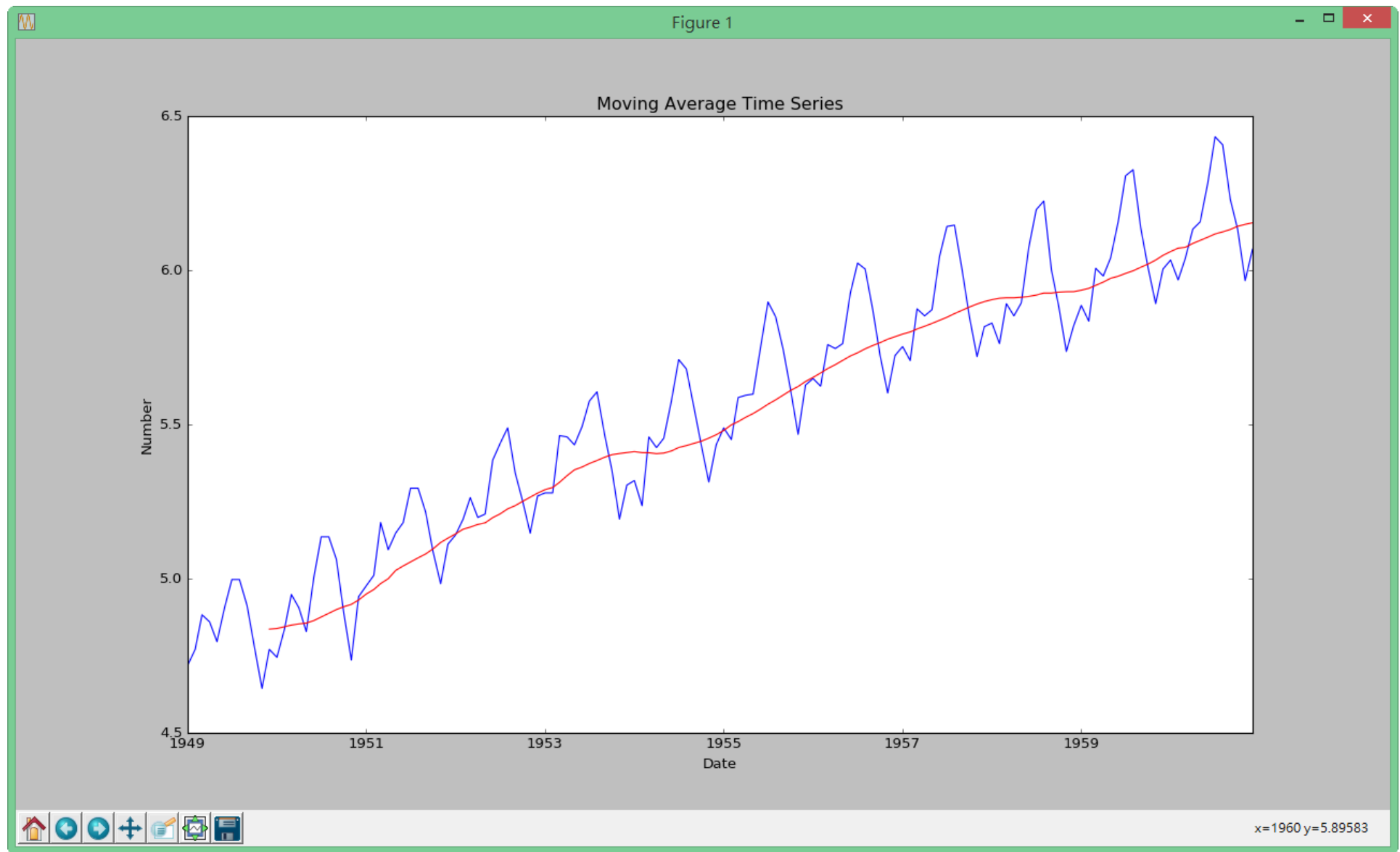
```
plt.plot(moving_avg, color='red')
```

```
plt.xlabel('Date')
```

```
plt.ylabel('Number')
```

```
plt.title('Moving Average Time Series')
```

```
plt.show()
```



```
ts_log_moving_avg_diff = ts_log - moving_avg  
ts_log_moving_avg_diff.head(12)  
ts_log_moving_avg_diff.dropna(inplace=True)
```

Month

| | |
|------------|-----------|
| 1949-01-01 | NaN |
| 1949-02-01 | NaN |
| 1949-03-01 | NaN |
| 1949-04-01 | NaN |
| 1949-05-01 | NaN |
| 1949-06-01 | NaN |
| 1949-07-01 | NaN |
| 1949-08-01 | NaN |
| 1949-09-01 | NaN |
| 1949-10-01 | NaN |
| 1949-11-01 | NaN |
| 1949-12-01 | -0.065494 |

Name: #Passengers, dtype: float64

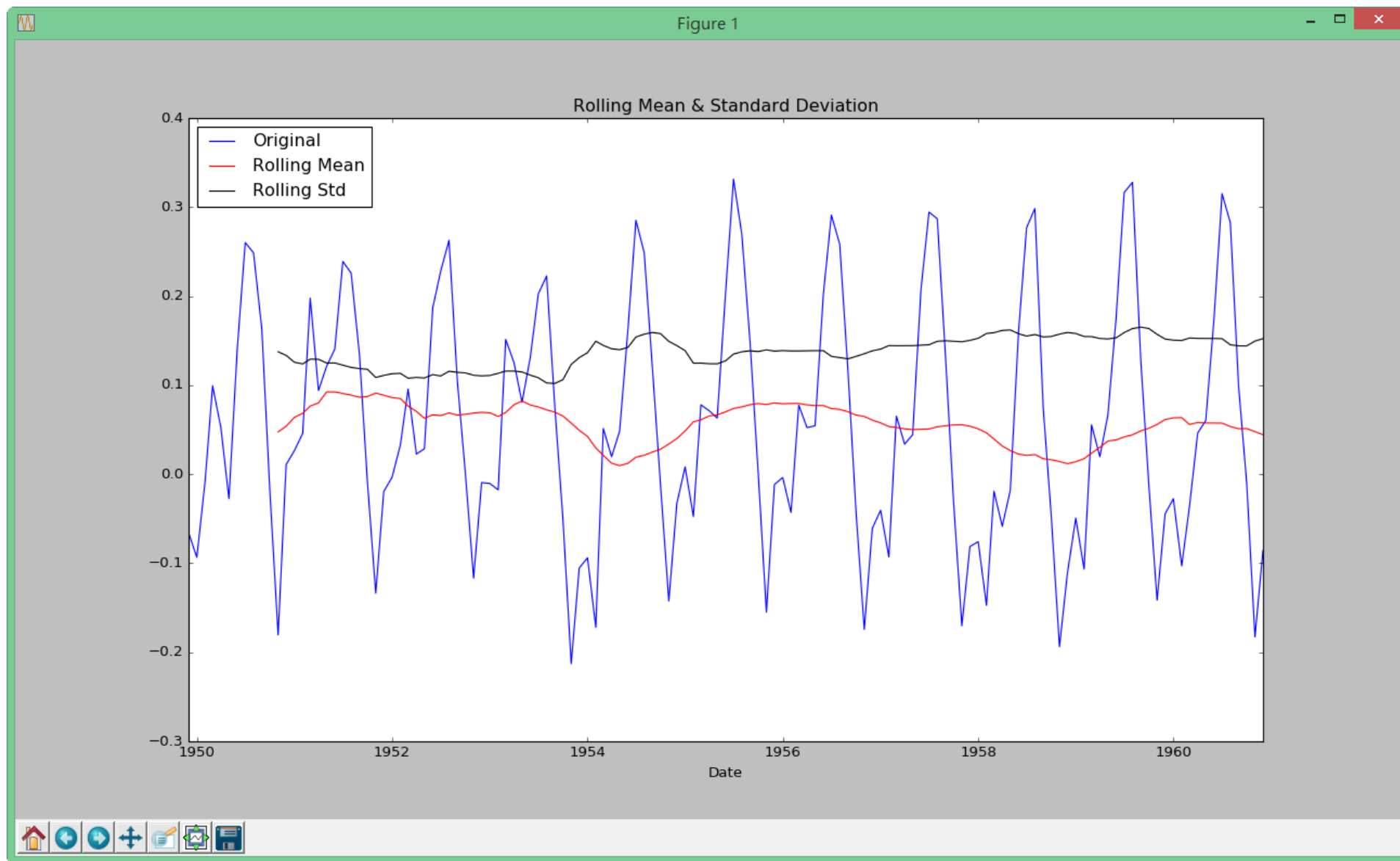
```
from statsmodels.tsa.stattools import adfuller

def test_stationarity(timeseries):
    #Determing rolling statistics
    rolmean = timeseries.rolling(window=12, center=False).mean()
    rolstd = timeseries.rolling(window=12, center=False).std()

    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.xlabel('Date')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)
```

```
#Perform Dickey-Fuller test:
print 'Results of Dickey-Fuller Test:'
dfctest = adfuller(timeseries, autolag='AIC')
dfoutput = pd.Series(dfctest[0:4], index=['Test Statistic','p-value','#Lags Used',
    'Number of Observations Used'])
for key,value in dfctest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print dfoutput
```

```
test_stationarity(ts_log_moving_avg_diff)
plt.show()
```

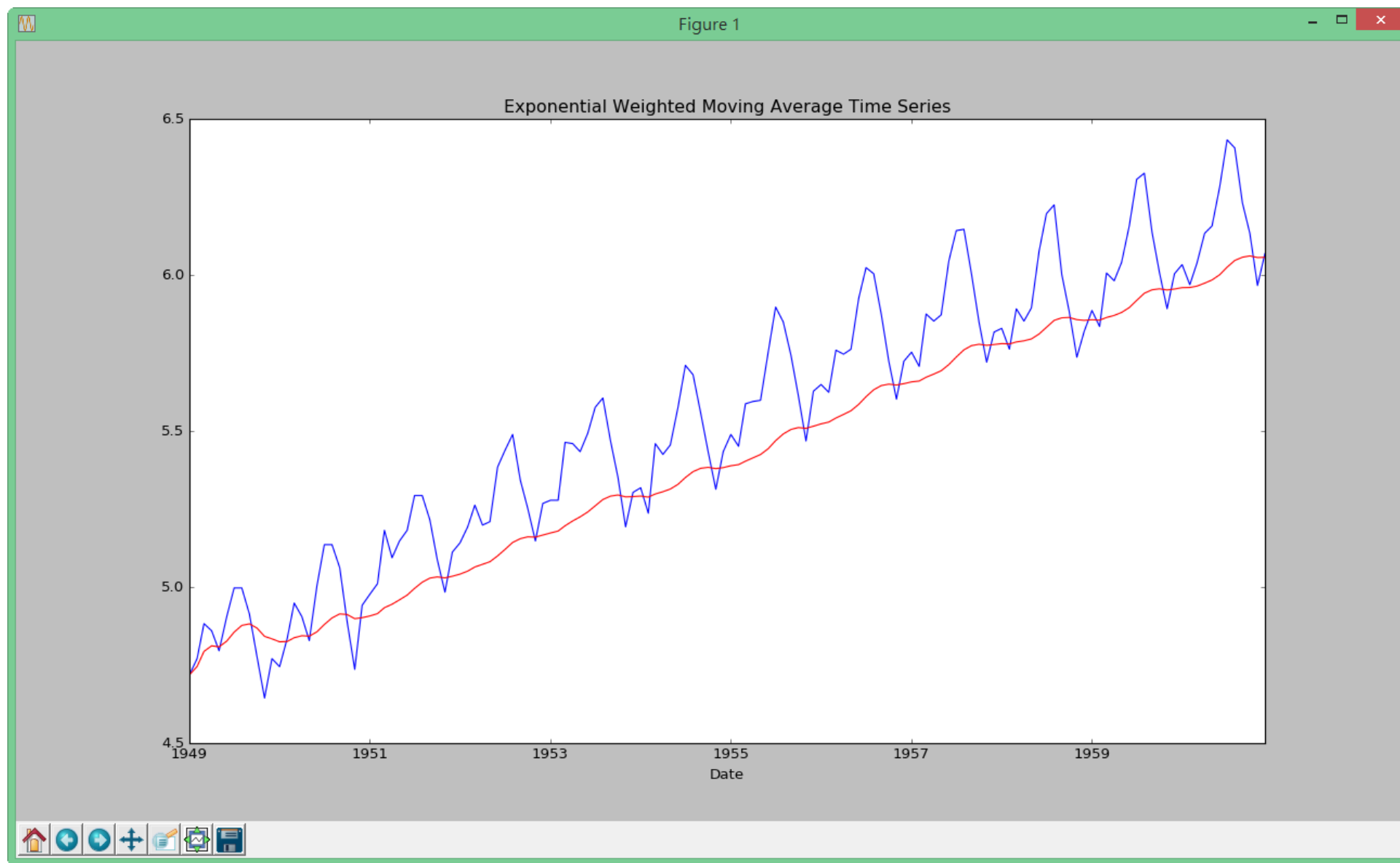


Results of Dickey-Fuller Test:

| | |
|-----------------------------|------------|
| Test Statistic | -3.162908 |
| p-value | 0.022235 |
| #Lags Used | 13.000000 |
| Number of Observations Used | 119.000000 |
| Critical Value (5%) | -2.886151 |
| Critical Value (1%) | -3.486535 |
| Critical Value (10%) | -2.579896 |

dtype: float64

```
expweighted_avg = ts_log.ewm(halflife=12, ignore_na=False, min_periods=0, adjust=True).mean()
plt.plot(ts_log)
plt.plot(expweighted_avg, color='red')
plt.xlabel('Date')
plt.title('Exponential Weighted Moving Average Time Series')
plt.show()
```

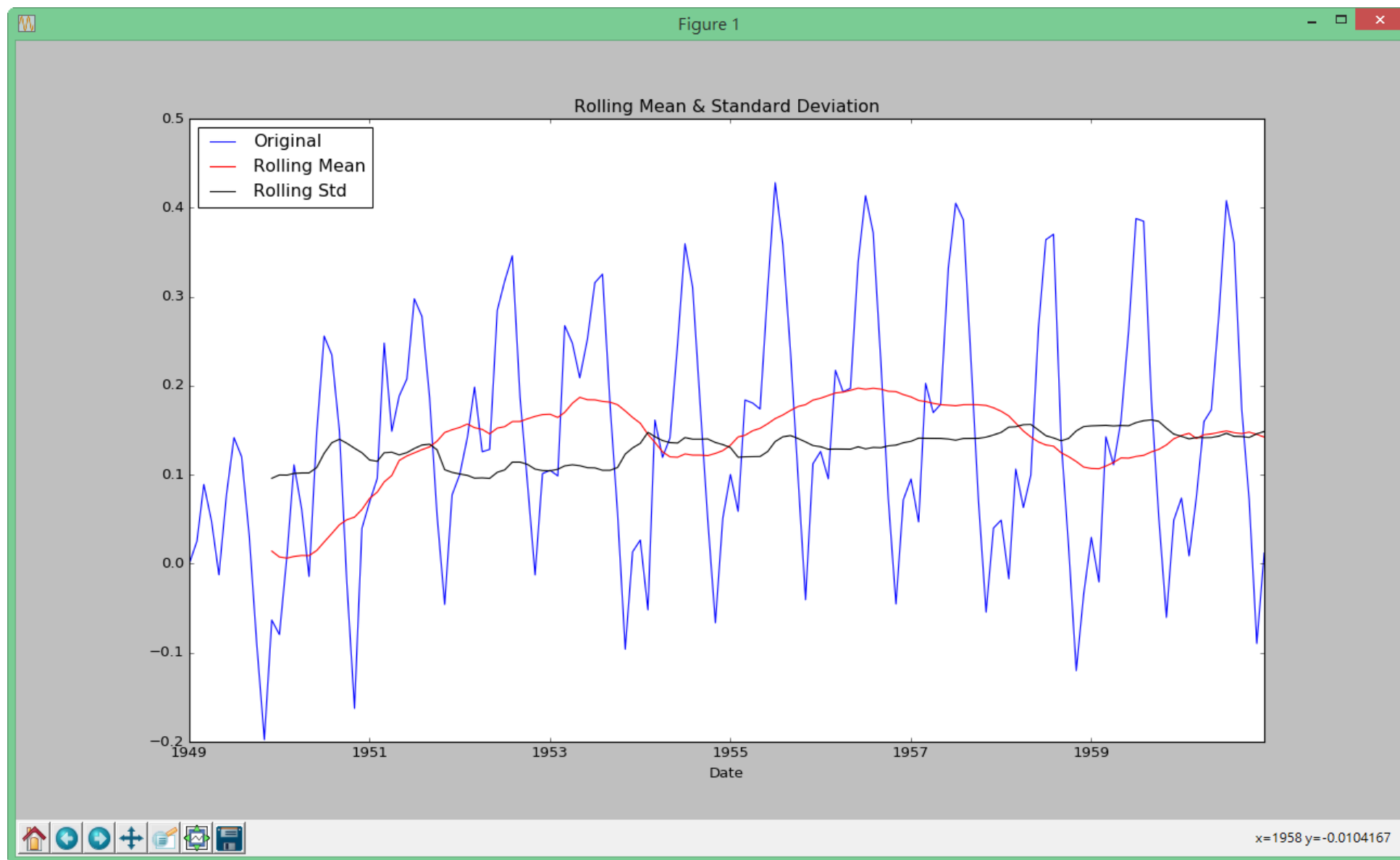



```
ts_log_ewma_diff = ts_log - expwighted_avg
test_stationarity(ts_log_ewma_diff)
plt.show()
```

Results of Dickey-Fuller Test:

| | |
|-----------------------------|------------|
| Test Statistic | -3.601262 |
| p-value | 0.005737 |
| #Lags Used | 13.000000 |
| Number of Observations Used | 130.000000 |
| Critical Value (5%) | -2.884042 |
| Critical Value (1%) | -3.481682 |
| Critical Value (10%) | -2.578770 |

dtype: float64

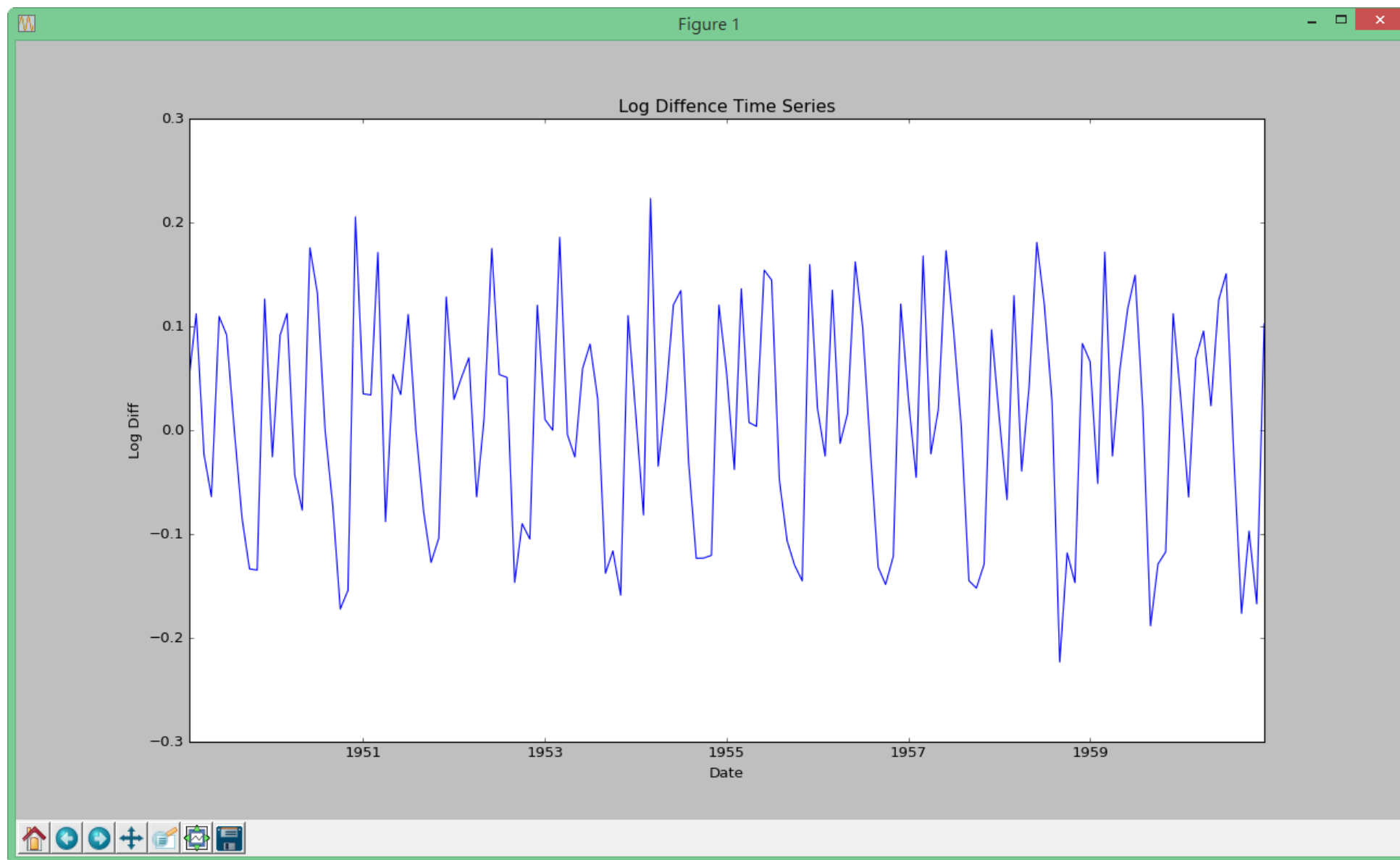


乙、消除趨勢與季節性

◆ 差分

DeComp_APData.py

```
ts_log = np.log(ts)
ts_log_diff = ts_log - ts_log.shift()
plt.xlabel('Date')
plt.ylabel('Log Diff')
plt.title('Log Diffence Time Series')
plt.plot(ts_log_diff)
plt.show()
```

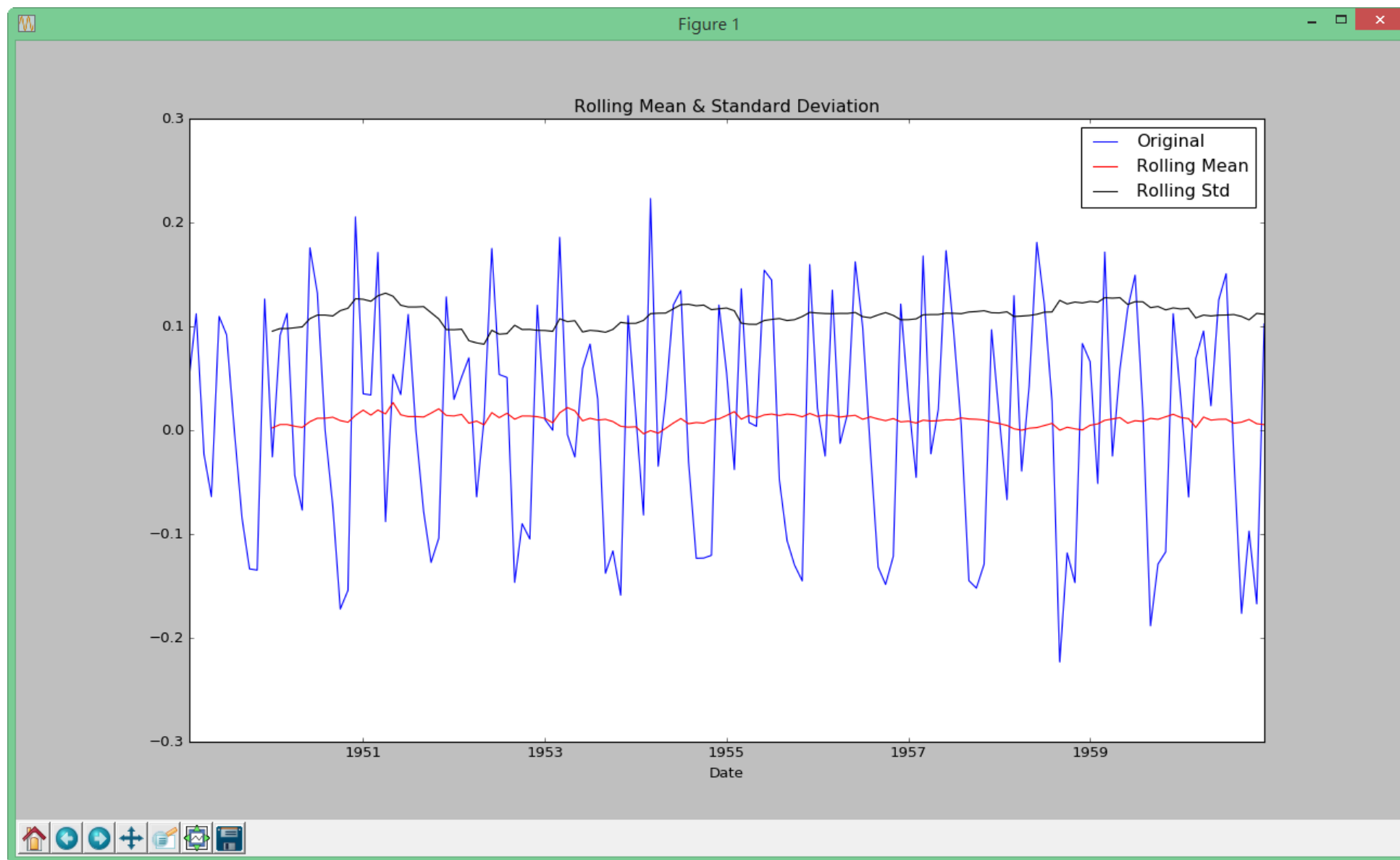


```
ts_log_diff.dropna(inplace=True)
test_stationarity(ts_log_diff)
```

Results of Dickey-Fuller Test:

| | |
|-----------------------------|------------|
| Test Statistic | -2.717131 |
| p-value | 0.071121 |
| #Lags Used | 14.000000 |
| Number of Observations Used | 128.000000 |
| Critical Value (5%) | -2.884398 |
| Critical Value (1%) | -3.482501 |
| Critical Value (10%) | -2.578960 |

dtype: float64



◆ 分解

```
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
decomposition = seasonal_decompose(ts_log)
```

```
trend = decomposition.trend
```

```
seasonal = decomposition.seasonal
```

```
residual = decomposition.resid
```

```
plt.subplot(411)
```

```
plt.plot(ts_log, label='Original')
```

```
plt.legend(loc='best')
```

```
plt.subplot(412)
```

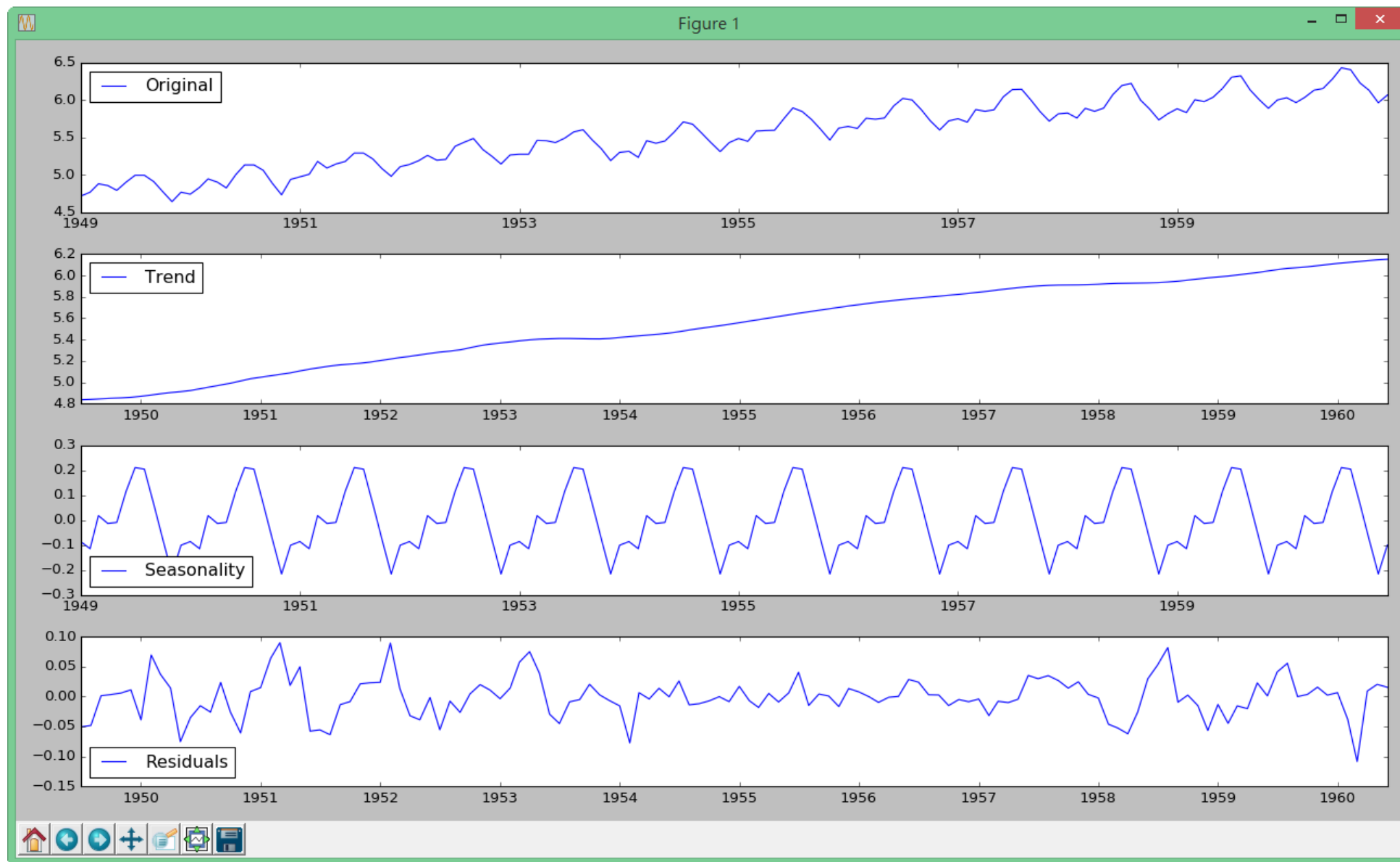
```
plt.plot(trend, label='Trend')
```

```
plt.legend(loc='best')
```



```
plt.subplot(413)
plt.plot(seasonal, label='Seasonality')
plt.legend(loc='best')
```

```
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

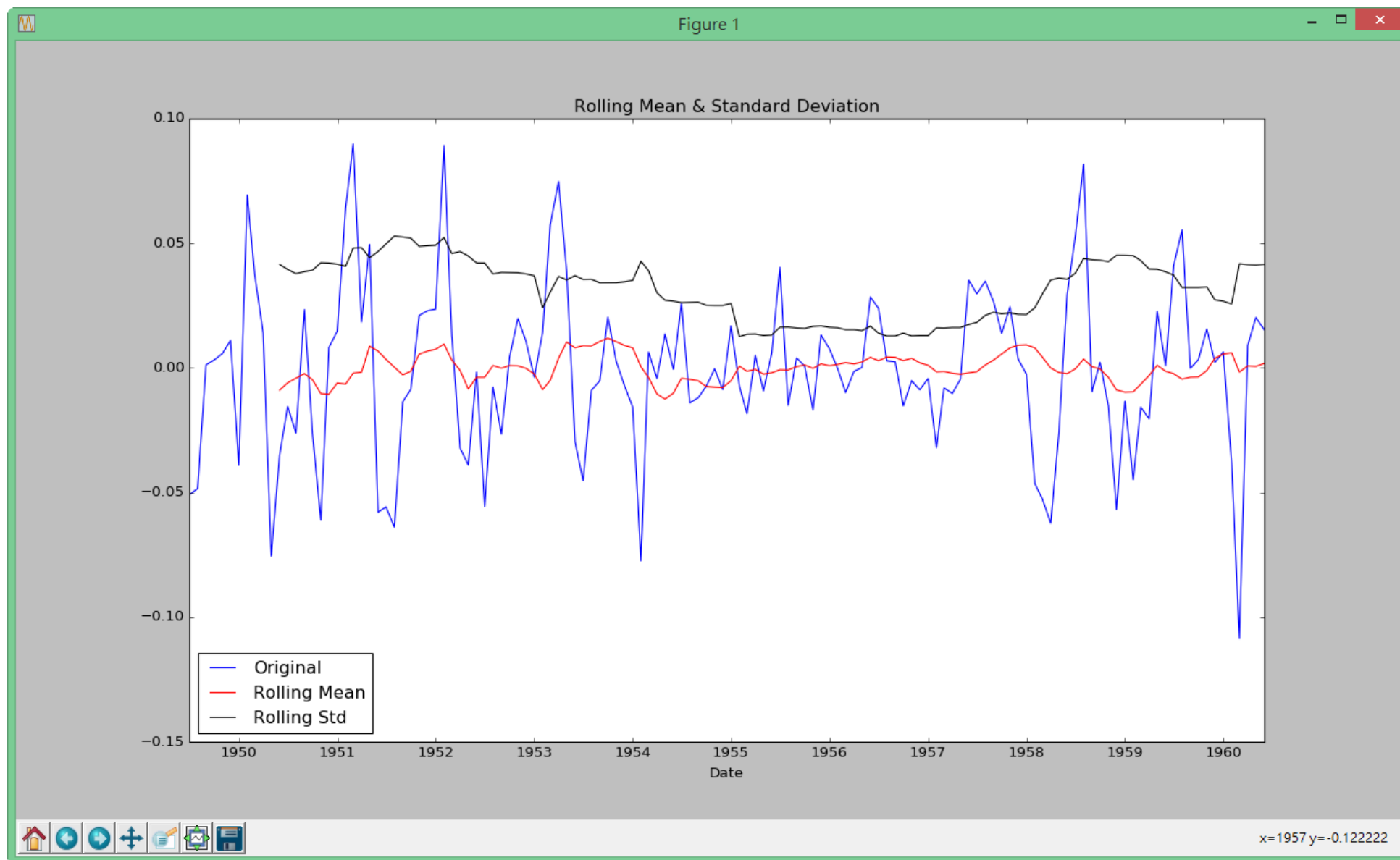


```
ts_log_decompose = residual
ts_log_decompose.dropna(inplace=True)
test_stationarity(ts_log_decompose)
```

Results of Dickey-Fuller Test:

| | |
|-----------------------------|---------------|
| Test Statistic | -6.332387e+00 |
| p-value | 2.885059e-08 |
| #Lags Used | 9.000000e+00 |
| Number of Observations Used | 1.220000e+02 |
| Critical Value (1%) | -3.485122e+00 |
| Critical Value (5%) | -2.885538e+00 |
| Critical Value (10%) | -2.579569e+00 |

dtype: float64



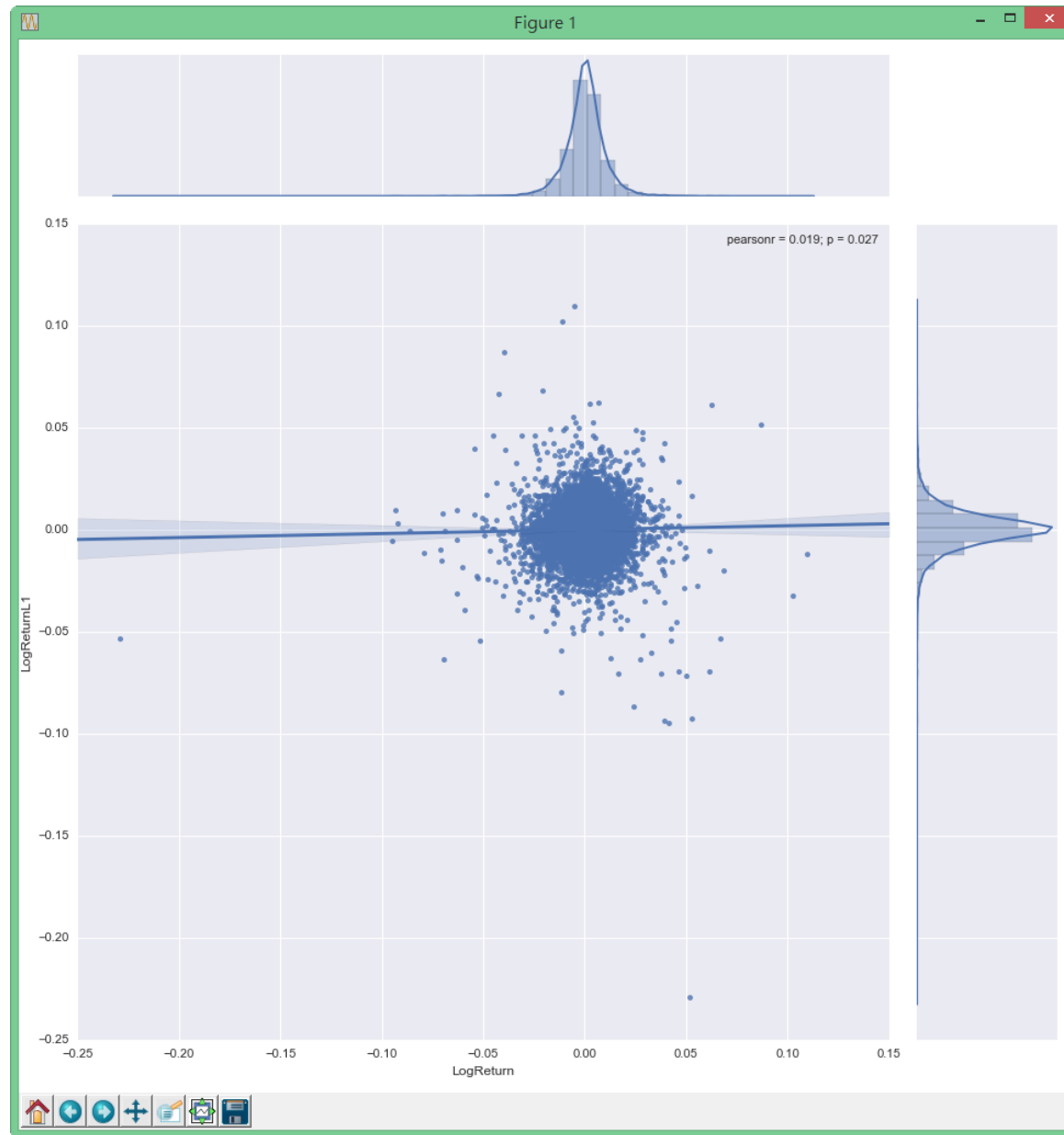
丙、S&P500檢定

◆ 針對我們產生的 S&P500 對數報酬數列，可以檢測其穩態性。

StationarySP.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

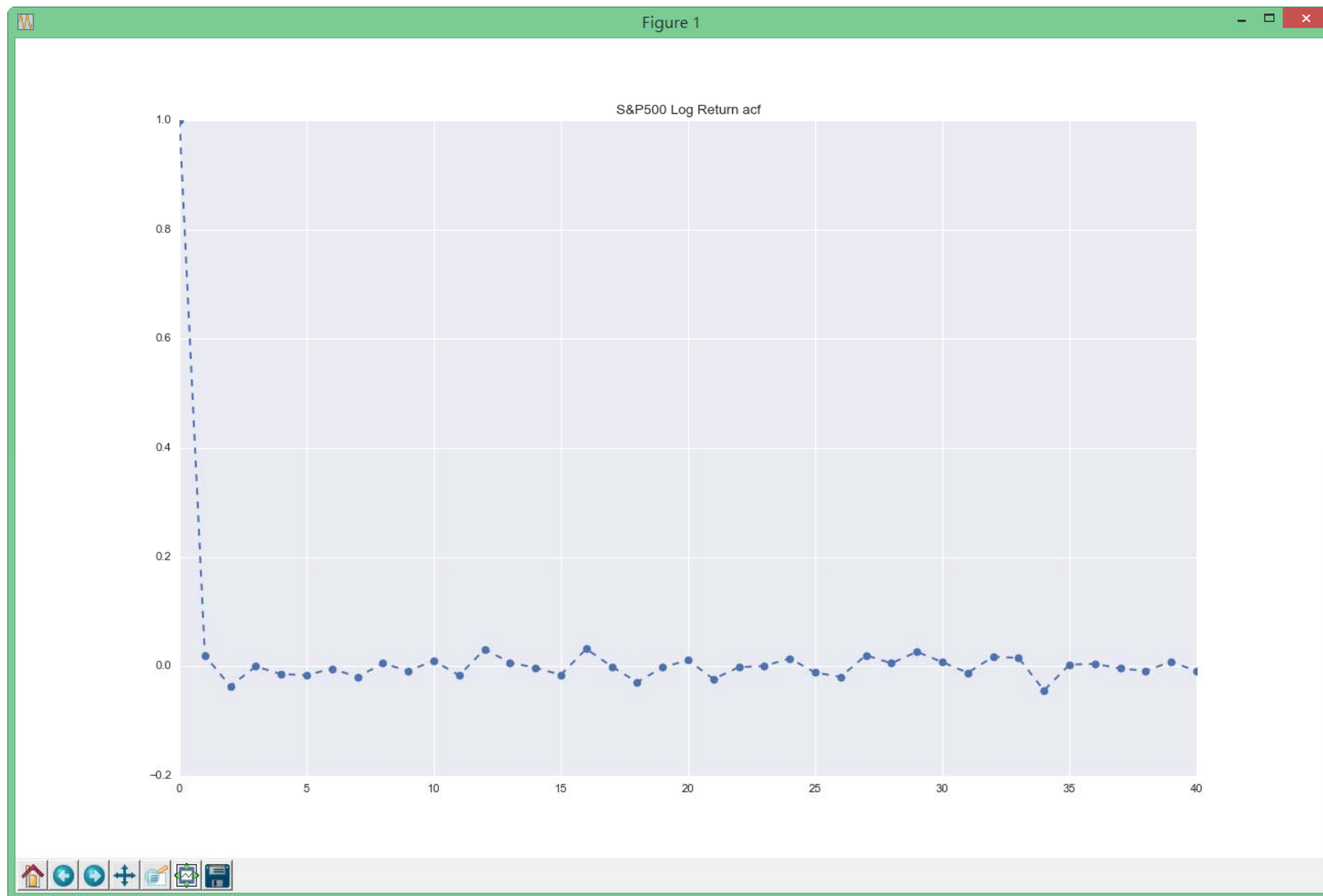
gspc = pd.read_csv('SP500Long.csv', index_col=0, parse_dates=True, infer_datetime_format=True)
gspc['Natural Log'] = gspc['Adj Close'].apply(lambda x: np.log(x))
gspc['LogReturn'] = gspc['Natural Log'] - gspc['Natural Log'].shift()
gspc['LogReturnL1'] = gspc['LogReturn'].shift()
gspc['LogReturnL2'] = gspc['LogReturn'].shift(2)
gspc['LogReturnL5'] = gspc['LogReturn'].shift(5)
gspc['LogReturnL30'] = gspc['LogReturn'].shift(30)
sb.set_style('darkgrid')
sb.jointplot('LogReturn', 'LogReturnL1', gspc, kind='reg', size=12)
plt.show()
```



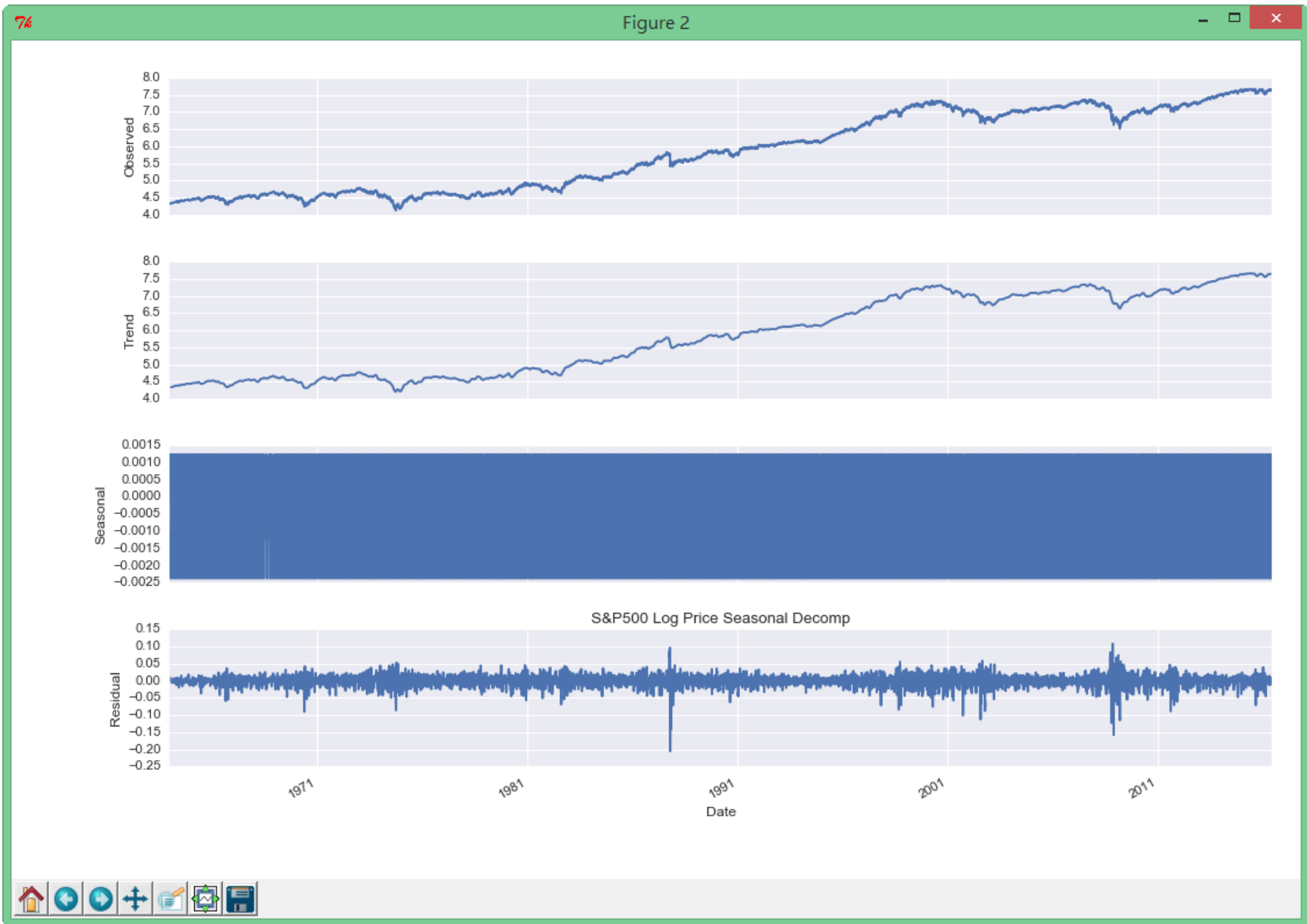
```
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf

lag_correlations = acf(gspc['LogReturn'].iloc[1:])
lag_partial_correlations = pacf(gspc['LogReturn'].iloc[1:])

fig, ax = plt.subplots(figsize=(14, 10))
ax.plot(lag_correlations, marker='o', linestyle='--')
plt.title('S&P500 Log Return acf')
plt.grid(True)
plt.show()
```




```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(gspc['Natural Log'], model='additive', freq=30)
fig = decomposition.plot()
plt.show()
```



四、R 套件 quantmod 讀取資料

<http://www.quantmod.com/examples/intro/>

(一)使用 quantmod 撈取台股資料(上市、上櫃)

◆ 用 R 語言做股票的預測是學習 R 的基礎入門。

- 台股的取的方式很簡單，用我們的股票代碼+TW 就可以了，如下：

上市：股票代碼.TW

上櫃：股票代碼.TWO

```
stock2 <- getSymbols("2317.TW", auto.assign = FALSE, from="2017-01-01" ,to="2018-01-30")
```

- ✓ 2317.TW 就是鴻海。
- ✓ auto.assign = FALSE 表示不用幫我自動 assign 參數，我要把取得的資料塞到 stock2 變數中
- ✓ from="2017-01-01"，to="2018-01-30" 代表取得日期範圍。

- 網站上搜尋到的幾乎都是類似這樣的範例，

◆ 上櫃股票在股票代碼加上.TWO 即可

- quantmod 的資料來源來自於 <https://finance.yahoo.com/>，不確定的話到旁邊搜尋一下就可以看到 8069.TWO, 2317.TW。
- 台股下午 2 點收盤，到晚上 7 點 getSymbols 尚未更新今天的資料，<https://finance.yahoo.com/>都已更新，未知 quantmod 資料何時更新。

