Weekly Report of Research Work

WR-ABS-TEMP-2015A-No.008

汤吉(Ji TANG)
Number: WR-ABS-TEMP-2015A, E-mail: tangji08@hotmail.com
Date: 11/3/2016 - 17/1/2016

January 17, 2016

Contents

1 Work

1. I have written a python program (including 13 functions) in order to plot the chart in the time slot you want .

   Before the optimization, it took 3 minutes to plot the chart from 20070103 to 20080801.

   And after the optimization, it took only 3 seconds to plot the same chart!

2. I have studied " Econophysics" and " Basic random economics"

3. I have tried to write a program to calculate the " Hurst Exponent"

2 The Python Program

2.1 Plot the file of the form " .txt"

At first, I want to write a program to plot the data file for each year. So I need to transform the file into data list. The txt file just like this:

```
20150101,231700,1186.4,1186.5,1186.4,1186.5,4
20150101,231800,1186.4,1186.5,1186.4,1186.5,4
20150101,232000,1186.7,1186.8,1186.7,1186.8,4
20150101,232100,1186.9,1187.2,1186.8,1187.0,4
20150101,232200,1187.1,1187.1,1187.0,1187.1,4
20150101,232300,1187.0,1187.2,1187.0,1187.2,4
20150101,232400,1187.0,1187.1,1187.0,1187.1,4
20150101,232500,1187.2,1187.2,1187.0,1187.0,4
20150101,232600,1186.9,1186.9,1186.6,1186.6,4
20150101,232700,1186.5,1186.6,1186.3,1186.3,4
20150101,232800,1186.4,1186.5,1186.4,1186.5,4
20150101,232900,1186.4,1186.5,1186.3,1186.5,4
20150101,233100,1186.6,1186.6,1186.5,1186.5,4
20150101,233300,1186.6,1186.8,1186.6,1186.8,4
```

What the program read is just about

" /xf/xef20150101,231700,1186.4,1186.5,1186.4,1186.5,4"

```
#This function will read the file and transform it into data
def file2data(filename):
    Date = []
    Time = []
    Price = []
```

```
    fr = open(filename)  # Load the data
    for line in fr.readlines():
        lines = line.strip()  # Strip the beginning and the end blank
        listFromLine = lines.split(',')  # Separate each line by '\t'
        Datelist = re.findall('\d', listFromLine[0])  # Find out the useful
            number data
        Dateint = int(string.join(Datelist,''))  # Rejoin the characters
        Date.append(Dateint)  # Get the evaluate of each person
        Time.append(listFromLine[1])
        Price.append(float(listFromLine[2]))
    return Date, Time, Price
```

Then this program will split it by ”,” and read the integers only and finally return three list in the forms of ”20150101”,”231700” and ”1186.4”.

Next, I need to calculate the distance between two data, and the distance need to be uniformly defined. In my opinion, we can transform the time into the fractional part of the day and calculate the distance of two ”float days”. So I write two functions, the first one will calculate the number of days between two days, and the second one will give a sequence number for each data based on the first data (the first data is set ”0”).

```
# To creat a funtion calculating the numbers of days between two dates
def datediff(beginDate, endDate):
    format = "%Y%m%d"
    # Translate the string to time form
    bd = time.strptime(beginDate, format)
    ed = time.strptime(endDate, format)
    # Translate the time form to date form
    bd = datetime.datetime(bd[0], bd[1], bd[2], bd[3], bd[4], bd[5])
    ed = datetime.datetime(ed[0], ed[1], ed[2], ed[3], ed[4], ed[5])
    oneday = datetime.timedelta(days = 1)
    count = 0
    while bd != ed:
        ed = ed-oneday
        count += 1
    return count
```

```
#This function will set the first data time as the "0" time and format the
    following data based on this time
def time2number(Date, Time):
```

```
    NumbersOfLines = len(Date) # Get the number of the lines
    index1 = 0
    index2 = 0
    day = np.zeros(NumbersOfLines)
    hourp = np.zeros(NumbersOfLines)
    #Set the "0" moment
    Date0 = str(Date[0])
    number0 = Time[0]
    hour0 = float(number0[0: 2]) + float(number0[2: 4]) / 60
    for date in Date:
        datestring = str(date)
        day[index1] = datediff(Date0, datestring)
        index1 += 1
    for days in Time:
        hour = float(days[0: 1]) + float(days[2: 3]) / 60 +float(days[4:5]) /
            3600
        hourp = float(hour - hour0) / 24
        index2 += 1
    day = np.array(day)
    return (day + hourp)
```

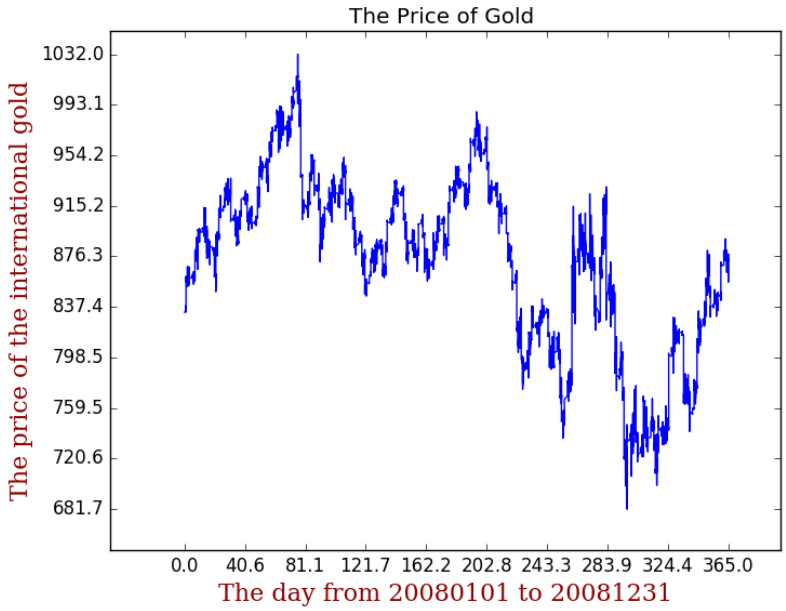
Finally, We can use the data we get to plot the file.

```
def plotforextxt(filename):
    Date, Time, Price = file2data(filename)
    Day = time2number(Date, Time)
    fp = filename[0: 6] + "_" + str(Date[0]) + "_" + str(Date[-1])

    #************************************************************
    fig = plt.figure(figsize=(8, 6), dpi=84, facecolor="white")
    axes = plt.subplot(111)
    axes.cla() # Clear all the information in the coordinate
    # Assign the font of the picture
    font = {'family' : 'serif',
            'color'  : 'darkred',
          'weight' : 'normal',
           'size'   : 16,
          }
    #************************************************************
    plt.plot(Day, Price)
    # Configurate the scale of the coordinate
    ax=plt.gca()
```

```
    ax.set_xticks(np.linspace(0, Day[-1], 10))
    #ax.set_xticklabels( ('0', '500', '1000', '1500', '2000'))
    ax.set_yticks(np.linspace(min(Price), max(Price), 10))
    #ax.set_yticklabels( ('500', '800', '1100', '1400', '1700','2000'))
    # Configurate the labels
    xlabel = "The day from "+str(Date[0])
    ylabel = "The price of the international gold"
    ax.set_ylabel(ylabel, fontdict = font)
    ax.set_xlabel(xlabel, fontdict = font)
    # Configurate the title
    titleStr = 'The Price of Gold'
    plt.title(titleStr)
    figname = fp+'.png'
    plt.savefig(figname)
    #plt.clf() # Clear the chart

    #plt.show()
    print('ALL -> Finished OK')

    plt.show()
```

For example, if we choose to plot the chart of 2008:

```
import FkNN
FkNN.plotforextxt("XAUUSD_2008.txt")
```



2.2 Plot the chart of the period you want

Then I want to plot the chart whenever you need. So I write two new functions:

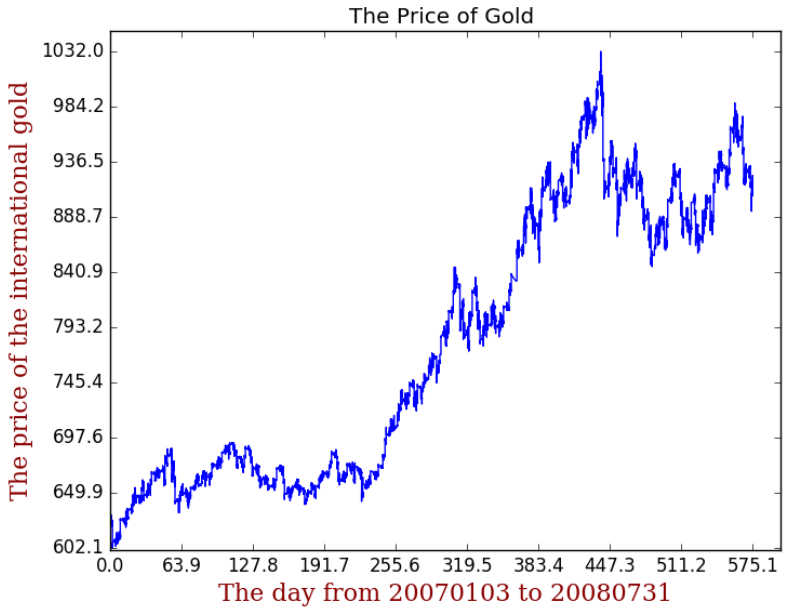
```
def input2data(forextype, startdate, enddate):
    startyear = startdate[0: 4]
    startmonth = startdate[4: 8]
    endyear = enddate[0: 4]
    endmonth = enddate[4: 8]
    startfilename = forextype + "_" + startyear + ".txt"
    endfilename = forextype + "_" + endyear + ".txt"
    returndate = []
    returntime = []
    returnprice = []
    date1, time1, price1 = file2data(startfilename)
    date2, time2, price2 = file2data(endfilename)

    startyear = int(startyear)
    endyear = int(endyear)
    starttime = int(str(startyear) + str(startmonth))
    endtime = int(str(endyear) + str(endmonth))

    # This part will select a closely date if the input date isn't the work day.
```

```
    startindex = -1
    endindex = -1
    while (startindex == -1):
        try:
            startindex = date1.index(starttime)
        except:
            starttime += 1
    while (endindex == -1):
        try:
            endindex = date2.index(endtime)
        except:
            endtime -= 1

    if (startyear == endyear):
        for index in range(startindex, endindex):
            returndate.append(date1[index])
            returntime.append(time1[index])
            returnprice.append(price1[index])
    elif (startyear < endyear):
        for index in range(startindex, len(date1)):
            returndate.append(date1[index])
            returntime.append(time1[index])
            returnprice.append(price1[index])
        startyear += 1
        while (startyear < endyear):
            startfilename = forextype + "_" + str(startyear) + ".txt"
            datewhole, timewhole, pricewhole = file2data(startfilename)
            for index in range(0, len(datewhole)):
                returndate.append(index)
                returntime.append(index)
                returnprice.append(index)
            startyear += 1
        for index in range(0, endindex):
            returndate.append(date2[index])
            returntime.append(time2[index])
            returnprice.append(price2[index])
    return returndate, returntime, returnprice
```

```
def plotforexdata(Date, Time, Price, name):
    Day = time2number(Date, Time)
    fp = name + "_" + str(Date[0]) + "_" + str(Date[-1])
```

```
4
5     #*********************************************************
6     fig = plt.figure(figsize=(8, 6), dpi=84, facecolor="white")
7     axes = plt.subplot(111)
8     axes.cla() # Clear all the information in the coordinate
9     # Assign the font of the picture
10    font = {'family' : 'serif',
11            'color'  : 'darkred',
12          'weight' : 'normal',
13           'size'   : 16,
14           }
15    #*********************************************************
16    plt.plot(Day, Price)
17    # Configurate the scale of the coordinate
18    ax=plt.gca()
19    ax.set_xticks(np.linspace(0, Day[-1], 10))
20    #ax.set_xticklabels( ('0', '500', '1000', '1500', '2000'))
21    ax.set_yticks(np.linspace(min(Price), max(Price), 10))
22    #ax.set_yticklabels( ('500', '800', '1100', '1400', '1700','2000'))
23    # Configurate the labels
24    xlabel = "The day from " + str(Date[0]) + " to " + str(Date[-1])
25    ylabel = "The price of the international gold"
26    ax.set_ylabel(ylabel, fontdict = font)
27    ax.set_xlabel(xlabel, fontdict = font)
28    # Configurate the title
29    titleStr = 'The Price of Gold'
30    plt.title(titleStr)
31    figname = fp+'.png'
32    plt.savefig(figname)
33    #plt.clf() # Clear the chart
34
35    #plt.show()
36    print('ALL -> Finished OK')
37
38    plt.show()
```

Then, if we want to plot the data from January 3, 2007 to June 31, 2008.



It does work, but it took about 3 minutes to draw this chat! It' s too slowly!

2.3 The optimizing

Because of its wasting of too much time, I have optimized the program by making the series of new data with UTC in each line of file. And I can get a series of new data with the beginning of " UTC" by the following function:

```
# code: UTF-8
import re
import string
import numpy as np
import datetime
import time
import matplotlib.pyplot as plt

#This function will read the file and translate it to data
def file2data(filename):
    Date = []
    Time = []
    Price = []
    fr = open(filename)  # Load the data
    for line in fr.readlines():
```

```
16          lines = line.strip()  # Strip the beginning and the end blank
17          listFromLine = lines.split(',')  # Separate each line by '\t'
18          Datelist = re.findall('\d', listFromLine[0])  # Find out the useful
                number data
19          Dateint = int(string.join(Datelist,''))  # Rejoin the characters
20          Date.append(Dateint)  # Get the evaluate of each person
21          Time.append(listFromLine[1])
22          Price.append(float(listFromLine[2]))
23      return Date, Time, Price
24
25
26  def str2UTC(timestr):
27      return str(int(time.mktime(time.strptime(timestr, "%Y%m%d%H%M%S"))))
28
29
30  def writeUTC(Date, Time ,Price , filename):
31      returnfile = open("UTC"+filename, "w")
32      for index in range(len(Date)):
33          returnfile.write(str2UTC((Date[index])+str(Time[index])) + "," + str(
                returnUTC[index]) + "," + str(Price[index]) + "\n")
34      returnfile.close()
```

The new data the program had set up are:

UTCXAUUSD_2001	2016/1/14 13:10	文本文档	3,391 KB
UTCXAUUSD_2002	2016/1/14 13:10	文本文档	2,659 KB
UTCXAUUSD_2003	2016/1/14 13:10	文本文档	2,517 KB
UTCXAUUSD_2004	2016/1/14 13:10	文本文档	3,373 KB
UTCXAUUSD_2005	2016/1/14 13:10	文本文档	3,201 KB
UTCXAUUSD_2006	2016/1/14 13:11	文本文档	6,266 KB
UTCXAUUSD_2007	2016/1/14 13:11	文本文档	7,464 KB
UTCXAUUSD_2008	2016/1/14 13:11	文本文档	10,779 KB
UTCXAUUSD_2009	2016/1/14 13:11	文本文档	11,227 KB
UTCXAUUSD_2010	2016/1/14 13:11	文本文档	11,447 KB
UTCXAUUSD_2011	2016/1/14 13:11	文本文档	12,028 KB
UTCXAUUSD_2012	2016/1/14 13:12	文本文档	12,023 KB

```
20040101231300, 1072969980. 0, 415. 0
20040101232300, 1072970580. 0, 415. 0
20040101233300, 1072971180. 0, 415. 0
20040101234300, 1072971780. 0, 415. 0
20040101235300, 1072972380. 0, 415. 0
20040102000100, 1072972860. 0, 415. 0
20040102001100, 1072973460. 0, 415. 0
20040102002100, 1072974060. 0, 415. 0
20040102003100, 1072974660. 0, 415. 0
20040102003700, 1072975020. 0, 415. 3
20040102004000, 1072975200. 0, 415. 4
20040102005000, 1072975800. 0, 415. 4
```

And then I wrote the new functions to plot this kind of data:

```
def UTCfile2data(filename):
    Time = []
    UTC = []
    Price = []
    fr = open(filename)  # Load the data
    for line in fr.readlines():
        lines = line.strip()  # Strip the beginning and the end blank
        listFromLine = lines.split(',')  # Separate each line by '\t'
        Time.append(listFromLine[0])
        UTC.append(float(listFromLine[1]))
        Price.append(float(listFromLine[2]))
    return Time, UTC, Price
```

The first function will transform the new UTC data into python data .

```
def plotUTCforexdata(Time, UTC, Price, name):
    fp = name + "_" + str(Time[0][0:8]) + "_" + str(Time[-1][0:8])

    #*********************************************************
    fig = plt.figure(figsize=(8, 6), dpi=84, facecolor="white")
    axes = plt.subplot(111)
    axes.cla() # Clear all the information in the coordinate
```

```
    # Assign the font of the picture
    font = { family'  :' serif' ,
           ' color'   :' darkred' ,
          ' weight'   :' normal' ,
           ' size'    : 16,
           }
    #*********************************************************
    plt.plot(UTC, Price)
    # Configurate the scale of the coordinate
    ax = plt.gca()
    interval = int(len(Price) / 5)
    ax.set_xticks(np.linspace(UTC[0], UTC[-1], 5))
    ax.set_xticklabels( (Time[0][0:8], Time[interval][0:8], Time[2*interval
        ][0:8], Time[3*interval][0:8], Time[-1][0:8]))
    ax.set_yticks(np.linspace(min(Price), max(Price), 8))
    #ax.set_yticklabels( (' 500' , ' 800' , ' 1100' , ' 1400' , ' 1700' ,' 2000' ))
    # Configurate the labels
    xlabel = "The day from " + str(Time[0][0:8]) + " to " + str(Time[-1][0:8])
    ylabel = "The price of the international gold"
    ax.set_ylabel(ylabel, fontdict=font)
    ax.set_xlabel(xlabel, fontdict=font)
    ax.grid(True)
    # Configurate the title
    titleStr =' The Price of Gold '
    plt.title(titleStr)
    figname = fp+' .png'
    try:
        os.mkdir(' ../Pictures/ )
    except:
        plt.savefig(' ../Pictures/ + figname)
    # savefig(' ../figures/contour_ex.png' ,dpi=48)
    #plt.clf() # Clear the chart

    #plt.show()
    print(' ALL -> Finished OK )

    plt.show()
```

The second function will plot the new data with UTC time, and it can finally translate the UTC time into normal time for axis x.

For example, if we want to plot the Gold Price Chart from 20070103 to 20080801 again.

It took me just 3 seconds!!! It works as 60 times quicker as the first time!

3 The Econophysices

3.1 对数收益率、随机游走、峰度

不同的交易资产有着相似的规律[Pagan(1996), Matteo, Aste, and Dacorogna(2005)]由于通常认为股价波动服从几何布朗运动，对数收益率也就自然成为了最常见的分析对象[Hull(2008)]. S(t) = lnP(t+dt) – lnP(t). 该方法的优点是，无需贴现因子，尺度变化的平均修正已被考虑在内.

随机游走(random walk)是一类最简单的随机过程，在数量金融领域有重要的理论意义与应用价值，其数学统计模型最早由英国数学家卡尔·皮尔逊(Karl Pearson, 1857-1936)在1905年提出 峰度$k = \frac{\sum_{i=1}^{N}[S(t_i)-\mu]^4}{\sigma^4}$

正态分布的峰度k=3，对于股票收益率的大量实证研究，均得到k大于3的结论

3.2 Hurst exponent

The Hurst exponent is used as a measure of long-term memory of time series. It relates to the autocorrelations of the time series, and the rate at which these decrease as the lag between pairs of values increases. Studies involving the Hurst exponent were

originally developed in hydrology for the practical matter of determining optimum dam sizing for the Nile river' s volatile rain and drought conditions that had been observed over a long period of time. The name " Hurst exponent", or " Hurst coefficient", derives from Harold Edwin Hurst (1880–1978), who was the lead researcher in these studies; the use of the standard notation H for the coefficient relates to his name also.

The Hurst exponent is referred to as the " index of dependence" or " index of long-range dependence". It quantifies the relative tendency of a time series either to regress strongly to the mean or to cluster in a direction A value H in the range 0.5–1 indicates a time series with long-term positive autocorrelation, meaning both that a high value in the series will probably be followed by another high value and that the values a long time into the future will also tend to be high. A value in the range 0 – 0.5 indicates a time series with long-term switching between high and low values in adjacent pairs, meaning that a single high value will probably be followed by a low value and that the value after that will tend to be high, with this tendency to switch between high and low values lasting a long time into the future. A value of H=0.5 can indicate a completely uncorrelated series, but in fact it is the value applicable to series for which the autocorrelations at small time lags can be positive or negative but where the absolute values of the autocorrelations decay exponentially quickly to zero. This in contrast to the typically power law decay for the 0.5 < H < 1 and 0 < H < 0.5 cases.

3.2.1 Definition

The Hurst exponent, H, is defined in terms of the asymptotic behaviour of the rescaled range as a function of the time span of a time series as follows

3.2.2 Estimating the exponent

To estimate the Hurst exponent, one must first estimate the dependence of the rescaled range on the time span n of observation. A time series of full length N is divided into a number of shorter time series of length n = N, N/2, N/4, ... The average rescaled range is then calculated for each value of n. For a (partial) time series of length n, $X = X_1, X_2, ..., X_n$, the rescaled range is calculated as follows:

1. Calculate the mean

$$m = \frac{1}{n}\sum_{i=1}^{n} X_i$$

2. Create a mean-adjusted series

$$Y_t = X_t - m \quad \text{for } t = 1, 2, \ldots, n$$

3. Calculate the cumulative deviate series Z

$$Z_t = \sum_{i=1}^{t} Y_i \quad \text{for } t = 1, 2, \ldots, n$$

4. Compute the range R

$$R(n) = \max(Z_1, Z_2, \ldots, Z_n) - \min(Z_1, Z_2, \ldots, Z_n).$$

5. Compute the standard deviation S

$$S(n) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - m)^2}$$

6. Calculate the rescaled range R(n)/S(n) and average over all the partial time series of length n.

The Hurst exponent is estimated by fitting the power law $\mathrm{E}\left[\frac{R(n)}{S(n)}\right] = Cn^H$ to the data. This can be done by plotting the logarithm of $\mathrm{E}\left[\frac{R(n)}{S(n)}\right]$ as a function of $\log n$, and fitting a straight line; the slope of the line gives H. Such a graph is called a pox plot. However, this approach is known to produce biased estimates of the power-law exponent. A more principled approach fits the power law in a maximum-likelihood fashion.

There are number of alternative techniques. Namely, DFA, Periodogram regression, aggregated variances, local Whittle' s estimator, wavelet analysis, both in the time domain and frequency domain.

3.2.3 Generalized exponent

The basic Hurst exponent can be related to the expected size of changes, as a function of the lag between observations, as measured by $E(|X_{t+1} - X_t|^2)$. For the generalized form of the coefficient, the exponent here is replaced by a more general term, denoted by q.

There are a variety of techniques that exist for estimating H, however assessing the accuracy of the estimation can be a complicated issue. Mathematically, in one technique, the Hurst exponent can be estimated such that:

$$H_q = H(q)$$

for a time series

$$g(t)(t = 1, 2, ...)$$

may be defined by the scaling properties of its structure functions $S_q(\tau)$:

$$S_q = \langle |g(t+\tau) - g(t)|^q \rangle_t \sim \tau^{qH(q)}$$

where q > 0, τ is the time lag and averaging is over the time window

$$t \gg \tau$$

usually the largest time scale of the system.

Practically, in nature, there is no limit to time, and thus H is non-deterministic as it may only be estimated based on the observed data; e.g., the most dramatic daily move upwards ever seen in a stock market index can always be exceeded during some subsequent day.

H is directly related to fractal dimension, D, where 1 < D < 2, such that D = 2 - H. The values of the Hurst exponent vary between 0 and 1, with higher values indicating a smoother trend, less volatility, and less roughness.

In the above mathematical estimation technique, the function H(q) contains information about averaged generalized volatilities (波动系数) at scale τ (only q = 1, 2 are used to define the volatility). In particular, the H_1 exponent indicates persistent $(H_1 > \frac{1}{2})$ or antipersistent (反持久性效应) $(H_1 < \frac{1}{2})$ behavior of the trend.

For the BRW (brown noise, $\frac{1}{f^2}$) one gets

$$Hq = \frac{1}{2}$$

and for pink noise ($\frac{1}{f}$)

$$Hq = 0$$

The Hurst exponent for white noise is dimension dependent, and for 1D and 2D it is

$$H_q^{1D} = -\frac{1}{2}, H_q^{2D} = -1$$

For the popular Lévy stable processes and truncated Lévy processes with parameter α it has been found that

$H_q = q/\alpha$ for $q < \alpha$ and $H_q = 1$ for $q \leq \alpha$

A method to estimate H(q) from non-stationary time series is called detrended fluctuation analysis (去趋势波动分析). When H(q) is a non-linear function of q the time series is a multifractal system.

3.2.4 The code for calculating the Hurst Exponent

```
__author__ ='Jojo'
import numpy as np
import math


def RS(Price):
    if len(Price) < 2:
        return 0
    Price = np.array(Price)
    n = len(Price)
    m = np.mean(Price)
    Y = Price - m
    Z = np.arange(n, dtype=float)
    for i in range(n):
        Z[i] = sum(Y[:i])
    R = max(Z) - min(Z)
    S = np.sqrt(np.var(Price))
    return R / S

def Hurst(Price):
    n = len(Price)
    times = int(math.log(n))
    x = []
    y = []
    price = Price
    for i in range(times):
        if(n > 4):
            x.append(n)
            y.append(RS(price[:n]))
        n = int(n / 2)
        Price_n = np.arange(n, dtype=float)
        for index in range(n):
            Price_n[index] = (price[2 * index] + price[2 * index + 1]) / 2
        price = Price_n
```

```
35      print x, y
36      k = np.polyfit(np.log(x), np.log(y), 1)
37      return k[0]
```

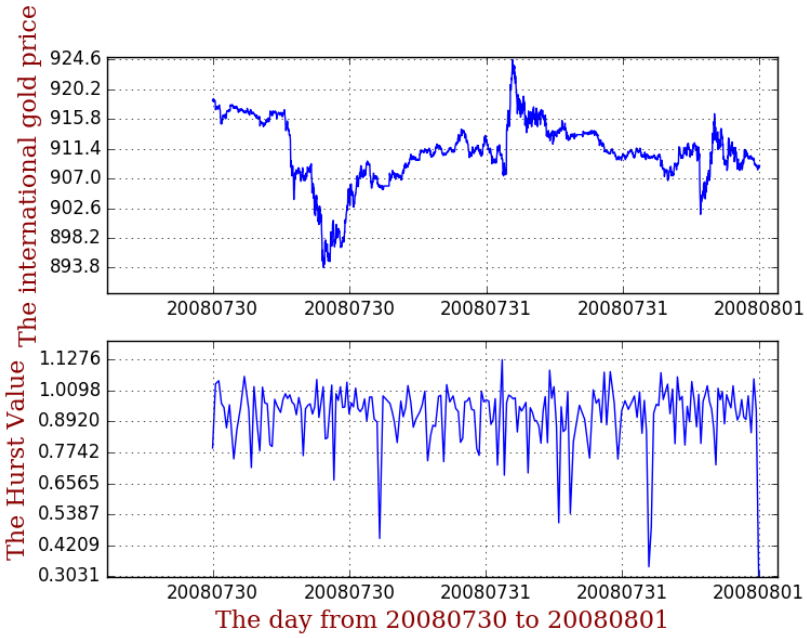
3.2.5 The code for ploting the Hurst Exponent

```
1  def data2hurst(UTC, Price, prange):
2      p = len(UTC) / prange
3      p = int(p)
4      Hurstvalue = np.arange(0, p+1, dtype=float)
5      for i in range(p):
6          Hurstvalue[i] = Hurst(Price[i*prange:(i+1)*prange])
7      Hurstvalue[p] = Hurst(Price[p*prange:])
8      x = np.arange(p+1)
9      return x, Hurstvalue
10
11
12 def plotprice_hurst(type, startime, endtime, prange):
13     Time, UTC, Price = input2data("UTC" + type[-6:], startime, endtime)
14     x, Hurstvalue = data2hurst(UTC, Price, prange)
15     HurstTime = np.arange(len(x))
16     for i in range(len(x)):
17         HurstTime[i] = UTC[i * prange]
18     fp = type[-6:] + "_" + str(Time[0][0:8]) + "_" + str(Time[-1][0:8])
19
20     #*********************************************************
21     fig = plt.figure(figsize=(8, 6), dpi=84, facecolor="white")
22     axes = plt.subplot(211)
23     axes.cla() # Clear all the information in the coordinate
24     # Assign the font of the picture
25     font = {'family' : 'serif',
26             'color'  : 'darkred',
27           'weight' : 'normal',
28             'size'   : 16,
29            }
30     #*********************************************************
31     plt.plot(UTC, Price)
32     # Configurate the scale of the coordinate
33     ax1 = plt.gca()
```

```
    interval = int(len(Price) / 5)
    ax1.set_xticks(np.linspace(UTC[0], UTC[-1], 5))
    ax1.set_xticklabels( (Time[0][0:8], Time[interval][0:8], Time[2*interval
        ][0:8], Time[3*interval][0:8], Time[-1][0:8]))
    ax1.set_yticks(np.linspace(min(Price), max(Price), 8))
    #ax.set_yticklabels( ('500','800','1100','1400','1700','2000'))
    # Configurate the labels
    xlabel = "The day from " + str(Time[0][0:8]) + " to " + str(Time[-1][0:8])
    ylabel = "The international gold price"
    ax1.set_ylabel(ylabel, fontdict=font)
    ax1.set_xlabel(xlabel, fontdict=font)
    ax1.grid(True)

    #*********************************************************
    axes = plt.subplot(212)
    plt.plot(HurstTime, Hurstvalue)
    # Configurate the scale of the coordinate
    ax2 = plt.gca()
    ax2.set_xticks(np.linspace(HurstTime[0], HurstTime[-1], 5))
    ax2.set_xticklabels( (Time[0][0:8], Time[interval][0:8], Time[2*interval
        ][0:8], Time[3*interval][0:8], Time[-1][0:8]))
    ax2.set_yticks(np.linspace(min(Hurstvalue), max(Hurstvalue), 8))
    #ax.set_yticklabels( ('500','800','1100','1400','1700','2000'))
    # Configurate the labels
    xlabel = "The day from " + str(Time[0][0:8]) + " to " + str(Time[-1][0:8])
    ylabel = "The Hurst Value"
    ax2.set_ylabel(ylabel, fontdict=font)
    ax2.set_xlabel(xlabel, fontdict=font)
    ax2.grid(True)

    # Configurate the title
    titleStr = 'The Price of Gold'
    plt.title(titleStr)
    figname = fp+'.png'
    try:
        os.mkdir('../Pictures/')
        plt.savefig('../Pictures/' + figname)
    except:
        plt.savefig('../Pictures/' + figname)
    # savefig('../figures/contour_ex.png',dpi=48)
    #plt.clf() # Clear the chart

```

```
    #plt.show()
    print( ALL -> Finished OK )

    plt.show()
```

3.2.6 The result of the code



3.3 自更新向心布朗运动

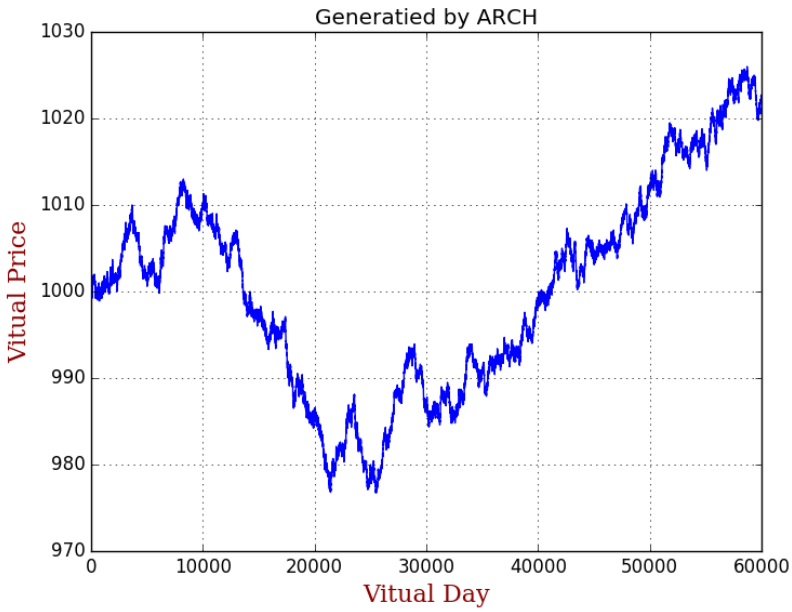
自更新向心布朗运动(Auto-Renewed Centripetal Brownian Motion)模型[周文超(2008),Yang, Zhou, and Huang(2012)] 假设一：供求原理 假设二：参与者的自我更新

以下是根据他的自更新向心布朗运动模型，编写的虚拟黄金走势程序以及图表：

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

step = 0
numInMin = 60
p0 = 1000.00  # Set the initial price
p = p0
```

```
pE = p0
percent = 0.1
T = numInMin * 1000  #Total steps
uOrD = 0
rand = 0
crit = 0
ERROR = 0.001
memory = 10
history = np.tile(p0, (1, memory))
oneDay = numInMin * 240
recordFreq = numInMin * 1000
price = []
priceTemp = step / 60
printTemp = 0
totalstep = []

while(step < T):
    step += 1
    totalstep.append(step)
    price.append(p)
    crit = p / (2 * pE)
    rand = np.random.rand()
    if(rand < crit):
        uOrD = -1
    else:
        uOrD = 1
    p = p + uOrD * percent

    pE = np.mean(history)
    history = np.delete(history, 0)
    history = np.append(history, p)

fig = plt.figure(figsize=(8, 6), dpi=84, facecolor="white")
ax = plt.gca()
xlabel = "Day"
ylabel = "Price"
plt.title("Generatied by ARCH")
plt.plot(totalstep, price)
ax.grid(True)
figname = 'Brownian Motion Price.png'
try:
    os.mkdir('../Pictures/')
```

```
    plt.savefig('../Pictures/' + figname)
except:
    plt.savefig('../Pictures/' + figname)
plt.show()
```



3.4 自相关系数与互相关系数

自回归条件异方差模型(autoregressive conditional heteroskedasticity model, ARCH)用以描绘时间序列波动性的随时演变。

自相关系数(auto-correlation function)常见于信号处理与金融时间序列分析，用于表征同一序列不同时刻取值之间的相关程度。考察一个收益率时间序列S(t)，其自协方差$Cov(\delta t)$定义为时间序列S(t)与其经过时间平移后的序列$S(t-\delta t)$之间的协方差。自协方差通过方差进行标准化之后，即成为自相关系数$\rho(\delta t)$。如下式表示

$$Cov(\delta t) = E\{[S(t)-\mu][S(t-\delta t)-\mu]\}$$

$$\rho(\delta t) = \frac{Cov(\delta t)}{\sigma}$$

其中，E(x)代表x的数学期望，μ和σ分别为时间序列S(t)的平均值和标准差，自协方差$Cov(\delta t)$与自相关函数$\rho(\delta t)$均为平移时间间隔δt的函数。

互相关系数可应用于研究两个不同序列之间的相关性，例如两只股票的收益率时间序列之间的相关性、同一只股票的价格时间序列与交易量时间序列之间的相关性。 假设两个时间序

列分别为$S_1(t)$与$S_2(t)$，则协方差$Cov(S_1, S_2)$与互相关函数$\rho(S_1, S_2)$分别定义为

$$COv(S_1, S_2) = E\{[S_1(t) - \mu_1][S_2(t) - \mu_2]\}$$

$$\rho(S_1, S_2) = \frac{Cov(S_1, S_2)}{\sigma_1 \sigma_2}$$