量化投资：以 Python 为工具

蔡立耑

第一部分

基础

第一章　略。

第二章　Python 代码的编写与执行

1. [1]

```
In [1]: %paste
for i in range(1,11):
    if i%2 == 1:
        print(i)

## -- End pasted text --
1
3
5
7
9
```

[1]在代码文件中复制代码，然后在 ipython 中输入%paste，并按 enter 键，即可执行代码并输出结果。

第三章　Python 对象类型初探

1.

```
In [1]: type((1,3,5,7))
Out[1]: tuple

In [2]: type('pi')
Out[2]: str

In [3]: type(5>6)
Out[3]: bool

In [4]: type({'High':7.45,'Low':7.30})
Out[4]: dict

In [5]: type(['Hello world',3,4,('a','b'),True])
Out[5]: list
```

2.

```
In [1]: (3+4j)*(4+3j)
Out[1]: 25j

In [2]: 3/4
Out[2]: 0.75

In [3]: True *3
Out[3]: 3

In [4]: 0.003-0.0022222
Out[4]: 0.0007777999999999999
```

3.

```
In [1]: a = [1, 2, 3]

```

```
In [2]: b=a

In [3]: b
Out[3]: [1, 2, 3]

In [4]: id(a)==id(b)
Out[4]: True

In [5]: b[0]=3

In [6]: a
Out[6]: [3, 2, 3]

In [7]: id(a)==id(b)
Out[7]: True
```

4.

(a)

```
In [1]: a=tuple(range(1,11))
```

```
In [2]: b=tuple((i+1 for i in range(20) if (i+1)%2==1))
```

(b)

```
In [3]: c=[5*i for i in range(11)]
```

```
In [4]: %paste
d=[];
for i in range(1,6):
        j=0
        while(j<3):
                d.append(i)
                j+=1
d

## -- End pasted text --
Out[4]: [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5]
```

(d)

```
In [5]: e=set({'NASDAQ','Dowjones','DAX','FTSE'})
```

```
In [1]: tuple('abc')
Out[1]: ('a','b','c')
```

```
3
4 In [2]: list('abc')
5 Out[2]: ['a','b','c']
6
7 In [3]: set('abc')
8 Out[3]: {'a','b','c'}
```

6.

```
1 In [1]: characters={'A':ord('A'),'B':ord('B'),'\n':ord('\n')}
2 In [2]: characters
3 Out[2]: {'A': 65,'\n': 10,'B': 98}
```

第四章　略。

第五章　Python 运算符与使用

1.

```
In [1]: 3+4
Out[1]: 7

In [2]: 3.4*4
Out[2]: 13.6

In [3]: 3//4
Out[3]: 0

In [4]: id(['a',3,True,'scd'][1])== id(3)
Out[4]: True

In [5]: 3==4 in [1,'345',3+4j,4 in [1,2,3]]
Out[5]: False

In [6]: (3==4*4.5%2 is 0) in [3,4,'Tom','c' in 'comic']
Out[6]: False
```

2. is 比较的是两个变量所指向的对象是否一样，== 判断的是值是否一样。

3.

```
In [1]: a=[1,2,3]
In [2]: c=2
In [3]: sum([i>c for i in a])==len(a)
Out[3]: False
```

```
In [1]: import random
#因产生随机数，代码结果可能有所不同
In [2]: a=[random.normalvariate(0,1) for i in range(20)]

In [3]: a.sort()

In [4]: a[0]
```

```
Out[4]: -1.5288923718112197

In [5]: a[-1]
Out[5]: 2.2531121988580143

In [6]: sum(a)
Out[6]: 5.7283863023842905
```

第六章　Python 常用语句

4.

```
In [1]: %paste
def is_odd(i):
    if type(i) == int:
        return(i if i%2==1 else 0)
    else:
        print('ERROR：Please input an integer.')
is_odd(7)

## -- End pasted text --
Out[1]: 7
```

```
In [2]: %paste
import random
a=[random.normalvariate(0,1) for i in range(5)]
for j in a:
    if j>=0:
        print('Big')
    else:
        print('Small')

## -- End pasted text --
Small
Small
Big
Small
Big
```

2.

```
In [3]: %paste
l = []
for i in range(5):
    row = [0 for j in range(5)]
    row[i] = 1
```

```
    l.append(row)
l

## -- End pasted text --
Out[5]:
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 0, 0, 0, 1]]
```

```
In [4]: for i in (-1,0,1,2,39):
   ...:     print(i in range(1,5))
   ...:
   ...:
False
False
True
True
False
```

5.

```
In [5]: %paste
Hilbert = []
for i in range(4):
    Hilbert.append([])
    for j in range(4):
        Hilbert[i].append(1/(i+j+1))
Hilbert

## -- End pasted text --
Out[6]:
[[1.0, 0.5, 0.3333333333333333, 0.25],
 [0.5, 0.3333333333333333, 0.25, 0.2],
 [0.3333333333333333, 0.25, 0.2, 0.16666666666666666],
 [0.25, 0.2, 0.16666666666666666, 0.14285714285714285]]
```

第七章　函数

1.

```
In [1]: def permutation(x):
   ....:         x[0],x[-1]=x[-1],x[0]
   ....:         return(x)
   ....: y=permutation(x)
   ....: y is x
   ....:
Out[1]: True
```

```
In [2]: def sum_lists(x,y):
   ...:     return([x[i]+y[i] for i in range(len(x))])
   ...:
   ...:
```

2.

```
In [3]: def sum2(*lists):
   ...:     if len(lists)==2:
   ...:         return(sum_lists(lists[0],lists[1]))
   ...:     else:
   ...:         return(sum_lists(lists[0],sum2(*lists[1:])))
   ...:
   ...:
```

```
In [4]: %paste
def fibo(n):
    if(n<3):
        a=1
    else:
        a=fibo(n-1)+fibo(n-2)
    return(a)
def seqfibo(n):
    for i in range(1,n+1):
        print(fibo(i))
seqfibo(5)
```

```
12
13 ## -- End pasted text --
14 1
15 1
16 2
17 3
18 5
```

5.

```
1 In [5]: %paste
2 def multi_abs(*numbers):
3     return(list(map(abs,numbers)))
4 multi_abs(-5,55,-6,0,-7)
5
6 ## -- End pasted text --
7 Out[7]: [5, 55, 6, 0, 7]
```

```
1 In [6]: %paste
2 def count_positive(returns):
3     return(sum(map(lambda x: x>0,returns)))
4 ret=[-0.4,0.5,-0.34,0.45,0.50]
5 count_positive(ret)
6
7 ## -- End pasted text --
8 Out[9]: 3
```

第八章　面向对象

6.

```
In [1]: class account:
   ...:     def __init__(self,name,balance):
   ...:         self.name=name
   ...:         self.balance=balance
   ...:
   ...:     def deposit(self,amount):
   ...:         self.balance += amount
   ...:
   ...:     def withdraw(self,amount):
   ...:         if amount>self.balance:
   ...:             print('余额不足，交易失败')
   ...:         else:
   ...:             self.balance -= amount
   ...:
   ...:
```

(a)

```
In [2]: sam = account('Sam',1000)
```

```
In [3]: sam.deposit(500)
In [4]: sam.withdraw(1200)
```

(b)

```
In [5]: sam.balance
Out[5]: 300
```

```
In [6]: class account:
   ...:     def __init__(self,name,balance):
   ...:         self.name=name
   ...:         self.__balance=balance
   ...:
   ...:     def deposit(self,amount):
   ...:         self.__balance += amount
```

```
    ...:
    ...:     def withdraw(self,amount):
    ...:         if amount>self.balance:
    ...:             print('余额不足，交易失败')
    ...:         else:
    ...:             self.__balance -= amount
    ...:
    ...:     def get_balance(self):
    ...:         return(self.__balance)
    ...:
    ...:     def set_balance(self,amount):
    ...:         self.__balance = amount
    ...:
    ...:
```

```
In [7]: class account:
    ...:     def __init__(self,name,balance):
    ...:         self.name=name
    ...:         self.__balance=balance
    ...:
    ...:     def deposit(self,amount):
    ...:         self.__balance += amount
    ...:
    ...:     def withdraw(self,amount):
    ...:         if amount>self.__balance:
    ...:             print('余额不足，交易失败')
    ...:         else:
    ...:             self.__balance -= amount
    ...:
    ...:     def get_balance(self):
    ...:         return(self.__balance)
    ...:
    ...:     def set_balance(self,amount):
    ...:         self.__balance = amount
    ...:
    ...:     def transfer(self,amount,target):
    ...:         if amount>self.__balance:
    ...:             print('余额不足，交易失败')
    ...:         else:
    ...:             target.deposit(amount)
    ...:             self.withdraw(amount)
```

```
    ...:
    ...:

In [8]: sam = account('Sam',1000)

In [9]: john = account('John',3000)

In [10]: john.transfer(1000,sam)
```

(2)

```
In [11]: class check(account):
    ...:     def __init__(self,name,balance,credit):
    ...:         self.name=name
    ...:         self.__balance=balance
    ...:         self.credit = credit
    ...:         self.overdraft = 0
    ...:
    ...:     def deposit(self,amount):
    ...:         if self.overdraft > 0 and self.overdraft-amount < 0:
    ...:             self.__balance = amount - self.overdraft
    ...:             self.overdraft = 0
    ...:         elif self.overdraft > 0:
    ...:             self.overdraft -= amount
    ...:         else:
    ...:             self.__balance += amount
    ...:
    ...:     def withdraw(self,amount):
    ...:         if self.__balance==0 and amount + self.overdraft > self.credit:
    ...:             print('超出信用額度，交易失敗')
    ...:             return(1)
    ...:         elif self.__balance==0:
    ...:             self.__overdraft += amount
    ...:             return(0)
    ...:         elif 0 < self.__balance < amount and amount - self.__balance <=
       self.credit:
    ...:             self.overdraft = amount - self.__balance
    ...:             self.__balance=0
    ...:             return(0)
    ...:         elif 0 < self.__balance < amount:
    ...:             print('超出信用額度，交易失敗')
```

```
...:             return(1)
...:         else:
...:             self.__balance -= amount
...:             return(0)
...:
...:     def transfer(self,amount,target):
...:         result = self.withdraw(amount)
...:         if result == 0:
...:             target.deposit(amount)
...:
...:

In [12]: sam = check('Sam',1000,1000)

In [13]: sam.withdraw(700)
Out[13]: 0

In [14]: sam.withdraw(1500)
超出信用額度，交易失敗
Out[14]: 1
```

第九章　Python 标准库与数据操作

1.

```
In [1]: import datetime as dt

In [2]: now = dt.datetime.now()

In [3]: now.strftime('%Y-%m-%d')
Out[3]: '2017-01-17'
```

```
In [4]: def create_name():
   ...:     name = input('输入用户名\n')
   ...:     if ord(name[0])<65 or ord(name[0])>122:
   ...:         print('用户名必须以字母开头')
   ...:         return(create_name())
   ...:     else:
   ...:         return(name)
   ...:
   ...:

In [5]: def verify_passwd(passwd):
   ...:     head = 65 <= ord(passwd[0]) <= 122
   ...:     contain_number = any([str(x) in passwd for x in range(9)])
   ...:     contain_symbol = any([str(x) in passwd for x in ('_','*','#')])
   ...:     return(head and (contain_number or contain_symbol))
   ...:
   ...:

In [6]: def create_passwd():
   ...:     passwd = input('输入密码\n')
   ...:     is_legal = verify_passwd(passwd)
   ...:     if is_legal:
   ...:         return(passwd)
   ...:     else:
   ...:         return(create_passwd())
   ...:
```

```
    ...:
In [7]: def create_account():
    ...:     create_name()
    ...:     create_passwd()
    ...:     print('用户创建成功')
    ...:
    ...:
In [8]: create_account()
输入用户名
python
输入密码
ffw342-rw
用户创建成功
```

2.

```
In [10]: evens = [i+1 for i in range(100) if (i+1)%2==0]
#output省略
```

4. (a)

```
In [1]: dates = [dt.datetime(2015,1,13)+dt.timedelta(i) for i in range(5)]

In [2]: closes = [7.31,7.28,7.40,7.43,7.41]

In [3]: prices = {dates[i]:closes[i] for i in range(5)}
```

```
In [4]: prices[dt.datetime(2015,1,20)] = 7.44

In [5]: dates.append(dt.datetime(2015,1,20))
```

(b)

```
In [16]: prices[dt.datetime(2015,1,21)-dt.timedelta(4)]
Out[6]: 7.41
```

```
In [7]: prices[dt.datetime(2015,1,16)] = 7.50
```

(c)

```
In [8]: %paste
import math
cash = 10000
share = {dates[0]:0}
for i in range(1,6):
    if prices[dates[i]]>prices[dates[i-1]]:
        buyshare = math.floor(0.5*cash/prices[dates[i]])
```

```
        share[dates[i]]= buyshare
        cash=cash-buyshare*prices[dates[i]]+share[dates[i-1]]*prices[dates[i]]
    else:
         share[dates[i]]= 0
share

## -- End pasted text --
Out[8]:
{datetime.datetime(2015, 1, 13, 0, 0): 0,
 datetime.datetime(2015, 1, 14, 0, 0): 0,
 datetime.datetime(2015, 1, 15, 0, 0): 675,
 datetime.datetime(2015, 1, 16, 0, 0): 333,
 datetime.datetime(2015, 1, 17, 0, 0): 0,
 datetime.datetime(2015, 1, 20, 0, 0): 508}
```

第十章　常用第三方库：Numpy 库与多维数组

1.

```
In [1]: import numpy as np

In [2]: hilbert = 1 / (np.arange(4)+np.arange(1,5)[:,np.newaxis])
```

```
In [1]: %paste
import datetime as dt
import numpy as np
import datetime as dt
import math
dates = [dt.datetime(2015,1,13)+dt.timedelta(i) for i in range(5)]
closes = [7.31,7.28,7.40,7.43,7.41]
prices = {dates[i]:closes[i] for i in range(5)}
prices[dt.datetime(2015,1,20)] = 7.44
dates.append(dt.datetime(2015,1,20))
cash = 10000
share = {dates[0]:0}
for i in range(1,6):
    if prices[dates[i]]>prices[dates[i-1]]:
        buyshare = math.floor(0.5*cash/prices[dates[i]])
        share[dates[i]]= buyshare
        cash=cash-buyshare*prices[dates[i]]+share[dates[i-1]]*prices[dates[i]]
    else:
         share[dates[i]]= 0
buyDates=[np.datetime64(date) for date in share.keys() if share[date]>0]
buyDates

## -- End pasted text --
Out[1]:
[numpy.datetime64('2015-01-15T00:00:00.000000'),
 numpy.datetime64('2015-01-20T00:00:00.000000'),
 numpy.datetime64('2015-01-16T00:00:00.000000')]

In [2]: %paste
```

```
[prices[dt.datetime.
          strptime(np.datetime_as_string(date)[:10],
                "%Y-%m-%d")] for date in buyDates]

## -- End pasted text --
Out[2]: [7.4, 7.44, 7.43]
```

2.

```
In [1]: cosin = np.cos(np.linspace(0,2*np.pi,1001))
```

```
In [1]: sample = np.array([0.5,1.43,-1.36,-0.16,0.29,-0.59,
   ....:                           1.16,-0.33,0.07,-1.36])

In [2]: np.mean(sample)
Out[2]: -0.035000000000000031

In [3]: np.var(sample)
Out[3]: 0.78990499999999986

In [4]: np.std(sample)
Out[4]: 0.88876599844953552

In [5]: np.median(sample)
Out[5]: -0.044999999999999998
```

第十一章　常用第三方库：Pandas 与数据处理

4.
```
In [1]: import numpy as np

In [2]: odds = [i+1 for i in range(20) if (i+1)%2==1]

In [3]: odds = np.array(odds)
```

2.
```
In [4]: three = tuple([i+1 for i in range(40) if (i+1)%3==0])

In [5]: three = np.array(three)
```

```
In [6]: same_number = odds[np.in1d(odds,three)]

In [7]: print(same_number[:round(len(same_number)/2)])
[ 3  9]

In [8]: for i in range(round(len(same_number)/2)):
   ...:     print(same_number[i])
   ...:
   ...:
3
9
```

3.
```
In [9]: np.random.uniform(0,10,10)
Out[9]: array([ 9.84971956,  5.63717566,  2.67350085,  3.43081329,  4.73561598,
        7.8112453 ,  0.92119308,  7.00992592,  6.31048269,  6.949389  ])
```

5.
```
In [10]: even = np.arange(2,22,2)

In [11]: even[-5:]
Out[11]: array([12, 14, 16, 18, 20])

In [12]: even[len(even)-5:]
```

```
Out[12]: array([12, 14, 16, 18, 20])
```

6.

```
In [1]: %paste
import pandas as pd
data = pd.DataFrame({'id':['a','b','c','d','e','f'],'name':
    ['Alice','Bob','Charlie','David','Esther','Fanny'],'age':
        [34,36,30,29,32,36]})

data.T.ix[2]

## -- End pasted text --
Out[1]:
0       Alice
1         Bob
2     Charlie
3       David
4      Esther
5       Fanny
Name: name, dtype: object
```

7.

```
In [2]: %paste
new = pd.DataFrame({'id':['g'],'name':['John'],'age':[19]})
data = data.append(new)
data.index = data.age
data

## -- End pasted text --
Out[2]:
     age id     name
age
34    34  a    Alice
36    36  b      Bob
30    30  c  Charlie
29    29  d    David
32    32  e   Esther
36    36  f    Fanny
19    19  g     John

In [3]: %paste
data = data.drop(30)
```

```
data

## -- End pasted text --
Out[3]:
      age  id     name
age
34     34   a    Alice
36     36   b      Bob
29     29   d    David
32     32   e   Esther
36     36   f    Fanny
19     19   g     John
```

第十二章　常用第三方库：Matplotlib 库与数据可视化

1.

```
In [1]: %paste

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

Money = pd.read_csv('Data/Part1/012/Money.csv',
                    index_col='date')

axis1 = plt.subplot()
axis1.plot(Money.index,Money.m,'-')
plt.title('Money Supply of Canada')
plt.xlabel('Year')
axis2 = axis1.twinx()
axis2.plot(Money.y,'-')

## -- End pasted text --
Out[1]: [<matplotlib.lines.Line2D at 0x7f223a2e9b70>]

In [2]: plt.show()
```

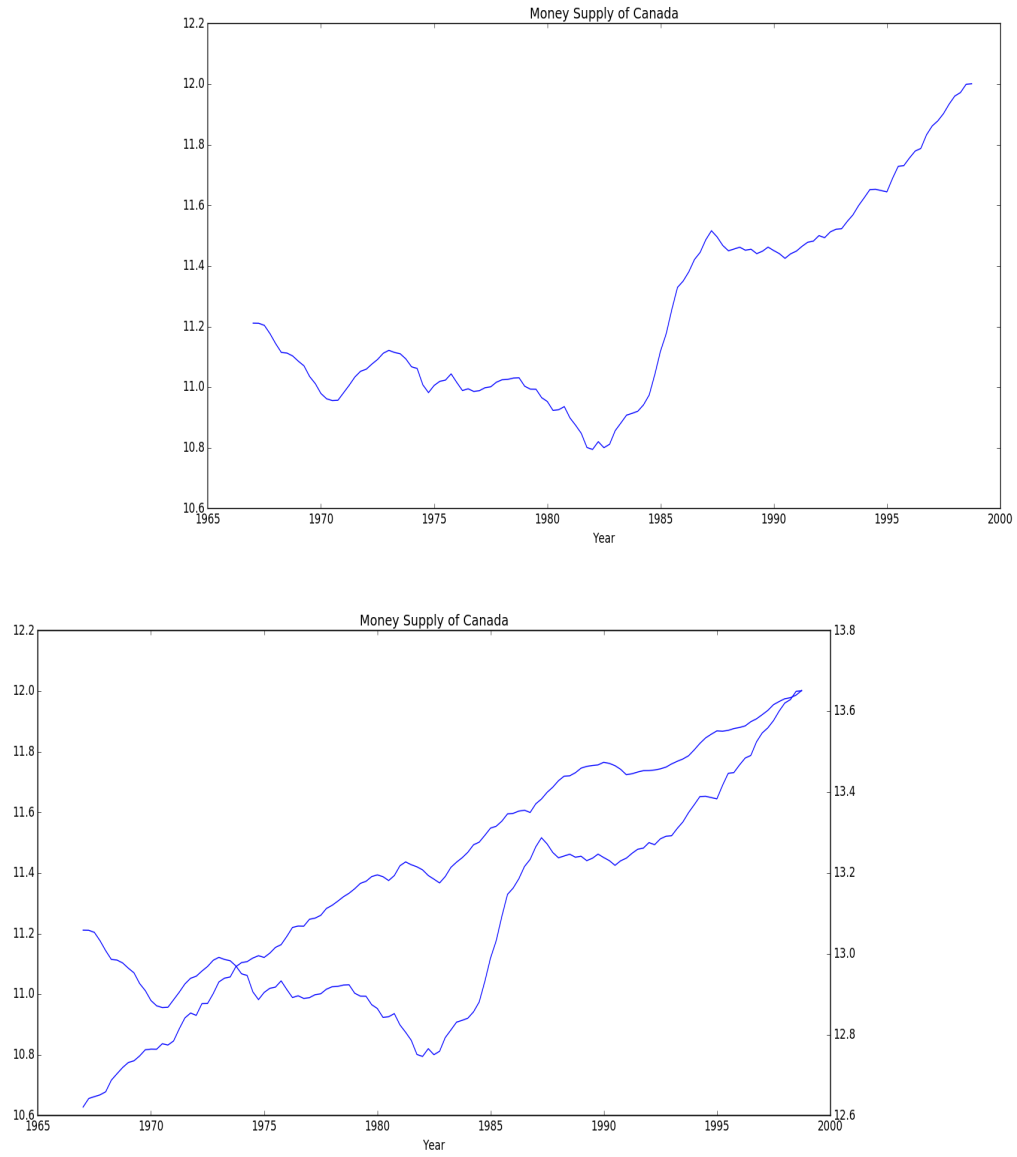


图 12.1: 题 1

2.

```
In [1]: %paste
Journals = pd.read_csv('Data/Part1/012/Journals.csv')
plt.scatter(Journals.citestot,Journals.libprice)
plt.title('Price vs Citations')
plt.xlabel('Citations')
plt.ylabel('Price')
plt.show()
```

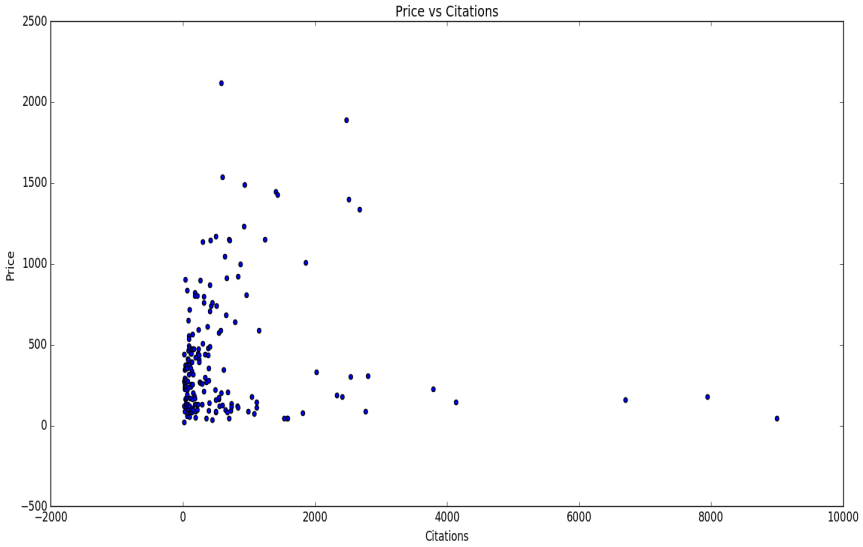


图 12.2: 题 2

3.

```
In [1]: %paste
mtcars = pd.read_csv('Data/Part1/012/mtcars.csv')
types=np.array([0.2,0.6])
number = mtcars.groupby(['gear','vs'])['vs'].agg(len)
plt.bar(types,number.ix[3],width=0.3,label='gear=3')
plt.bar(types,number.ix[4],width=0.3,
        bottom=number.ix[3],color='r',
        label='gear=4')
plt.bar(types,number.ix[5],width=0.3,
        bottom=number.ix[3]+number.ix[4],
        color='y',label='gear=5')
plt.xlim([0,1])
plt.ylim([0,25])
plt.legend()
plt.xticks(types+0.3/2,[0,1])
plt.show()
```

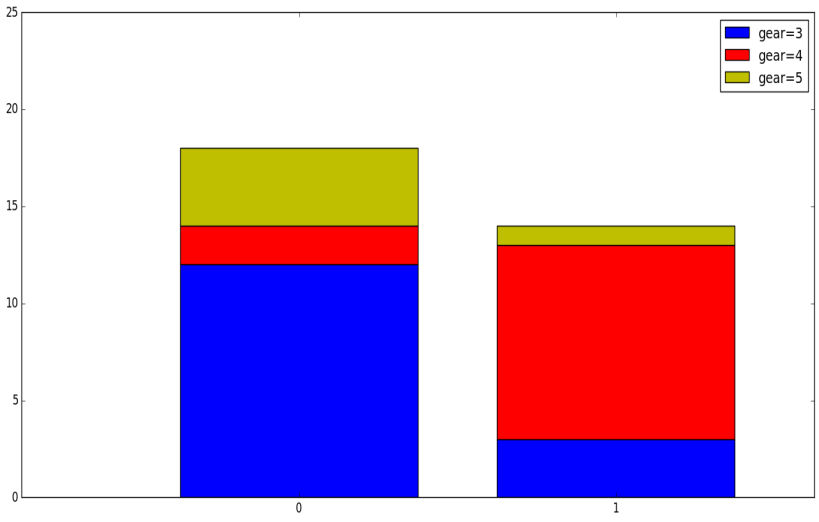


图 12.3: 题 3

4.

```
In [1]: %paste
Arthritis = pd.read_csv('Data/Part1/012/Arthritis.csv')
plt.hist(Arthritis.Age)
plt.xlabel('Age')
plt.title('Histogram of Age')
plt.show()
## -- End pasted text --
```

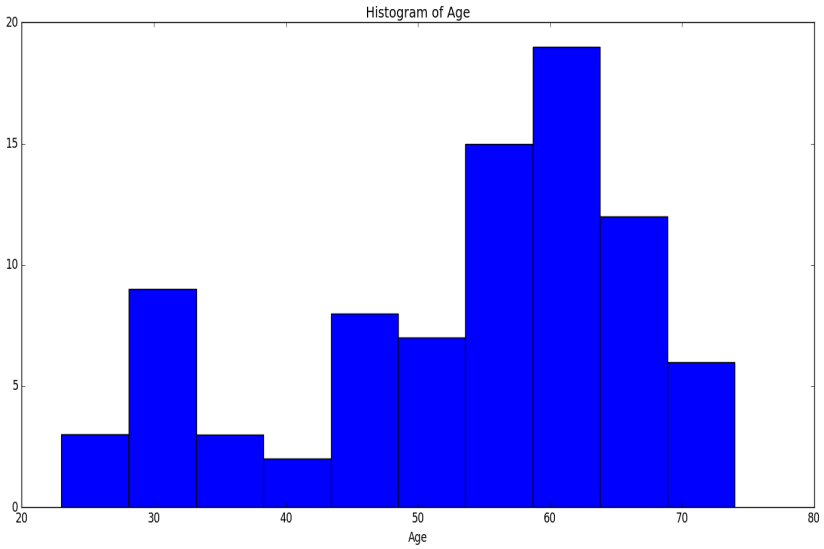


图 12.4: 题 4

5.

```
In [1]: %paste
norm1 = np.random.normal(0,1,100)
```

```
norm2 = np.random.normal(0,2,100)
norm3 = np.random.normal(0,3,100)
norm4 = np.random.normal(0,4,100)
plt.boxplot([norm1,norm2,norm3,norm4])
plt.xlabel('Standard Deviation')
plt.title('Normal Distributions with Different Standard Deviation')
plt.show()
## -- End pasted text --
```

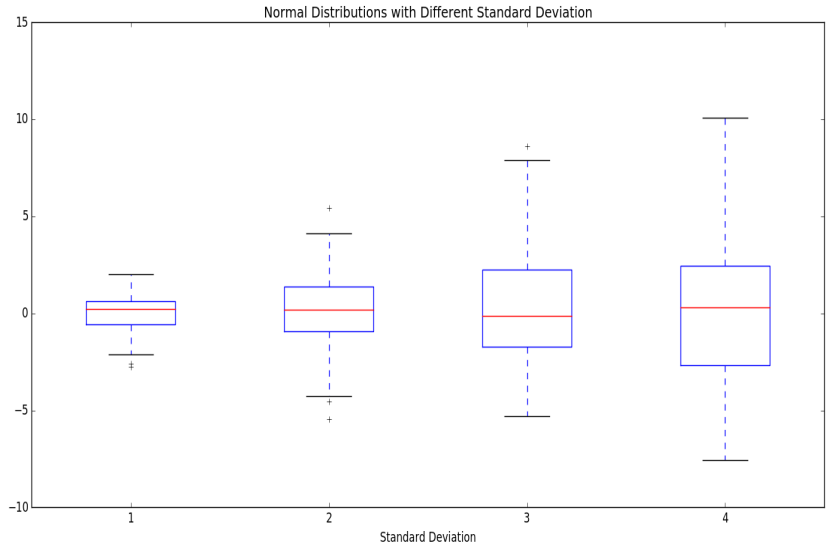


图 12.5: 题 5

6.

```
In [1]: %paste
norm1 = np.random.normal(0,1,100)
norm2 = np.random.normal(0,2,100)
norm3 = np.random.normal(0,3,100)
norm4 = np.random.normal(0,4,100)
figure,axes = plt.subplots(2,2)
axes[0,0].scatter(range(100),norm1)
axes[0,1].scatter(range(100),norm2)
axes[1,0].scatter(range(100),norm3)
axes[1,1].scatter(range(100),norm4)
plt.show()

## -- End pasted text --
```

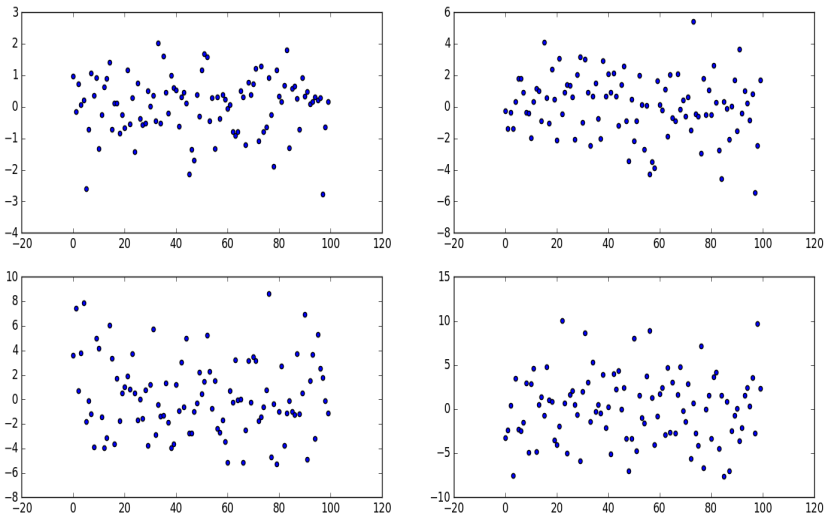


图 12.6: 题 6

7.

```
In [1]: %paste
tan_value = np.tan(np.linspace(0,2*np.pi,10001))[np.newaxis,:]
tan_value=np.concatenate((np.linspace(0,2*np.pi,10001)[np.newaxis,:],
        tan_value),0)
tan_value=np.where(tan_value>200,200,
     np.where(tan_value<-200,-200,tan_value))
plt.plot(tan_value[0],tan_value[1],'d')
plt.show()

## -- End pasted text --
```

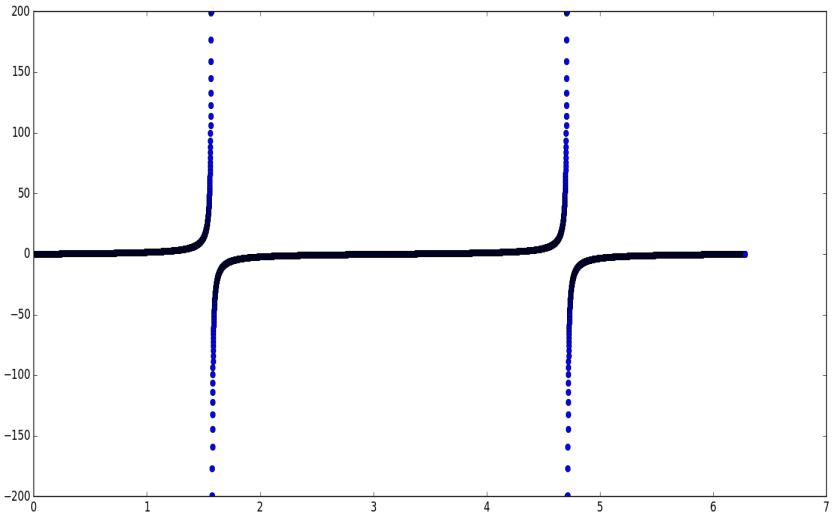


图 12.7: 题 7

第二部分

统计

第十三章　描述性统计

1. (a)

```
In [1]: import pandas as pd

In [2]: history = pd.read_csv('Data/Part2/001/history.csv',
   ...:                       index_col='Date')

In [3]: history.index = pd.to_datetime(history.index, format='%Y-%m-%d')

In [4]: history.head()
Out[4]:
            Convertible.Arbitrage  CTA.Global  Distressed.Securities  \
Date
1997-01-31                 0.0119      0.0393                 0.0178
1997-02-28                 0.0123      0.0298                 0.0122
1997-03-31                 0.0078     -0.0021                -0.0012
1997-04-30                 0.0086     -0.0170                 0.0030
1997-05-31                 0.0156     -0.0015                 0.0233
            Emerging.Markets  Equity.Market.Neutral  Event.Driven  \
Date
1997-01-31            0.0791                 0.0189        0.0213
1997-02-28            0.0525                 0.0101        0.0084
1997-03-31           -0.0120                 0.0016       -0.0023
1997-04-30            0.0119                 0.0119       -0.0005
1997-05-31            0.0315                 0.0189        0.0346
            Fixed.Income.Arbitrage  Global.Macro  Long.Short.Equity  \
Date
1997-01-31                  0.0191        0.0573             0.0281
1997-02-28                  0.0122        0.0175            -0.0006
1997-03-31                  0.0109       -0.0119            -0.0084
1997-04-30                  0.0130        0.0172             0.0084
1997-05-31                  0.0118        0.0108             0.0394
            Merger.Arbitrage  Relative.Value  Short.Selling
Date
```

```
33 1997-01-31          0.0150    0.0180    -0.0166
34 1997-02-28          0.0034    0.0118     0.0426
35 1997-03-31          0.0060    0.0010     0.0778
36 1997-04-30         -0.0001    0.0122    -0.0129
37 1997-05-31          0.0197    0.0173    -0.0737
```

```
In [5]: history['Emerging.Markets'].mean()
Out[5]: 0.0082460526315789474
```

(b)

```
In [6]: history['Emerging.Markets'].median()
Out[6]: 0.013649999999999999
```

```
In [7]: history['Emerging.Markets'].mode()
Out[7]:
0    0.016
dtype: float64
```

(c)

```
In [8]: history['Emerging.Markets'].quantile([0.1,0.9])
Out[8]:
0.1   -0.03844
0.9    0.04798
Name: Emerging.Markets, dtype: float64
```

2.

(a)

```
In [9]: history['Event.Driven'].max() - history['Event.Driven'].min()
Out[9]: 0.1328
```

```
In [10]: history['Event.Driven'].mad()
Out[10]: 0.012804657202216066
```

(b)

```
In [11]: history['Event.Driven'].var()
Out[11]: 0.00033673989369118157

In [12]: history['Event.Driven'].std()
Out[12]: 0.018350473936418688
```

```
In [13]: history[['Relative.Value','Fixed.Income.Arbitrage']].plot()
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3eef48a908>
```

可以看到，两者的收益率水平非常接近。为了验证这个结论，我们可以求出两者的平均数与标准差。

```
In [14]: history['Relative.Value'].mean()
Out[14]: 0.0067013157894736854

In [15]: history['Fixed.Income.Arbitrage'].mean()
Out[15]: 0.0042309210526315791

In [16]: history['Relative.Value'].std()
Out[16]: 0.0131946807807631

In [17]: history['Fixed.Income.Arbitrage'].std()
Out[17]: 0.014171294713188018
```

显然，两者的平均数与标准差都非常接近。因此，我们的结论是正确的。

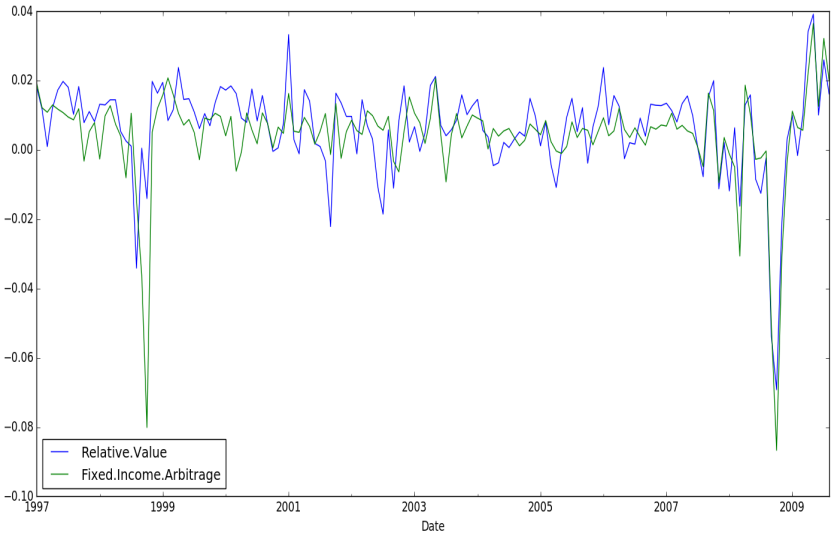


图 13.1: 题 3

3.

```
In [18]: history.describe()
Out[18]:
       Convertible.Arbitrage  CTA.Global  Distressed.Securities  \
count             152.000000  152.000000             152.000000
mean                0.006409    0.006489               0.007953
std                 0.020047    0.025131               0.018348
min                −0.123700   −0.054300              −0.083600
25%                 0.000925   −0.011500              −0.000175
50%                 0.009200    0.005250               0.009700
75%                 0.014600    0.022675               0.018225
```

```
max                 0.061100      0.069100                0.050400

       Emerging.Markets  Equity.Market.Neutral  Event.Driven  \
count        152.000000             152.000000    152.000000
mean           0.008246               0.006003      0.007622
std            0.038571               0.009006      0.018350
min           -0.192200              -0.058700     -0.088600
25%           -0.011400               0.002400     -0.000025
50%            0.013650               0.006300      0.010150
75%            0.028875               0.010125      0.018650
max            0.123000               0.025300      0.044200

       Fixed.Income.Arbitrage  Global.Macro  Long.Short.Equity  \
count              152.000000    152.000000         152.000000
mean                 0.004231      0.007672           0.007760
std                  0.014171      0.017020           0.022174
min                 -0.086700     -0.031300          -0.067500
25%                  0.002275     -0.003600          -0.003700
50%                  0.006000      0.006200           0.010150
75%                  0.009950      0.016450           0.021450
max                  0.036500      0.073800           0.074500

       Merger.Arbitrage  Relative.Value  Short.Selling
count        152.000000      152.000000     152.000000
mean           0.006785        0.006701       0.004161
std            0.011168        0.013195       0.055099
min           -0.054400       -0.069200      -0.134000
25%            0.001775        0.001850      -0.025450
50%            0.007700        0.008450      -0.000200
75%            0.013725        0.014500       0.035925
max            0.027200        0.039200       0.246300
```

第十四章　随机变量

1.

```
In [1]: import pandas as pd

In [2]: Bwages = pd.read_csv('Data/Part2/002/Bwages.csv')
```

(a)

```
In [3]: Bwages.wage.hist(normed=True)
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c7c879fd0>
```

```
In [4]: Bwages.wage.hist(normed=True,cumulative=True)
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c7c7f75f8>
```

(b)

```
In [5]: from scipy import stats

In [6]: import matplotlib.pyplot as plt

In [7]: import numpy as np

In [8]: kde = stats.gaussian_kde(Bwages.wage)

In [9]: bins=np.linspace(0, 50, num=200)

In [10]: plt.plot(bins,kde(bins).cumsum())
Out[10]: [<matplotlib.lines.Line2D at 0x7f8c7c01fe80>]
```

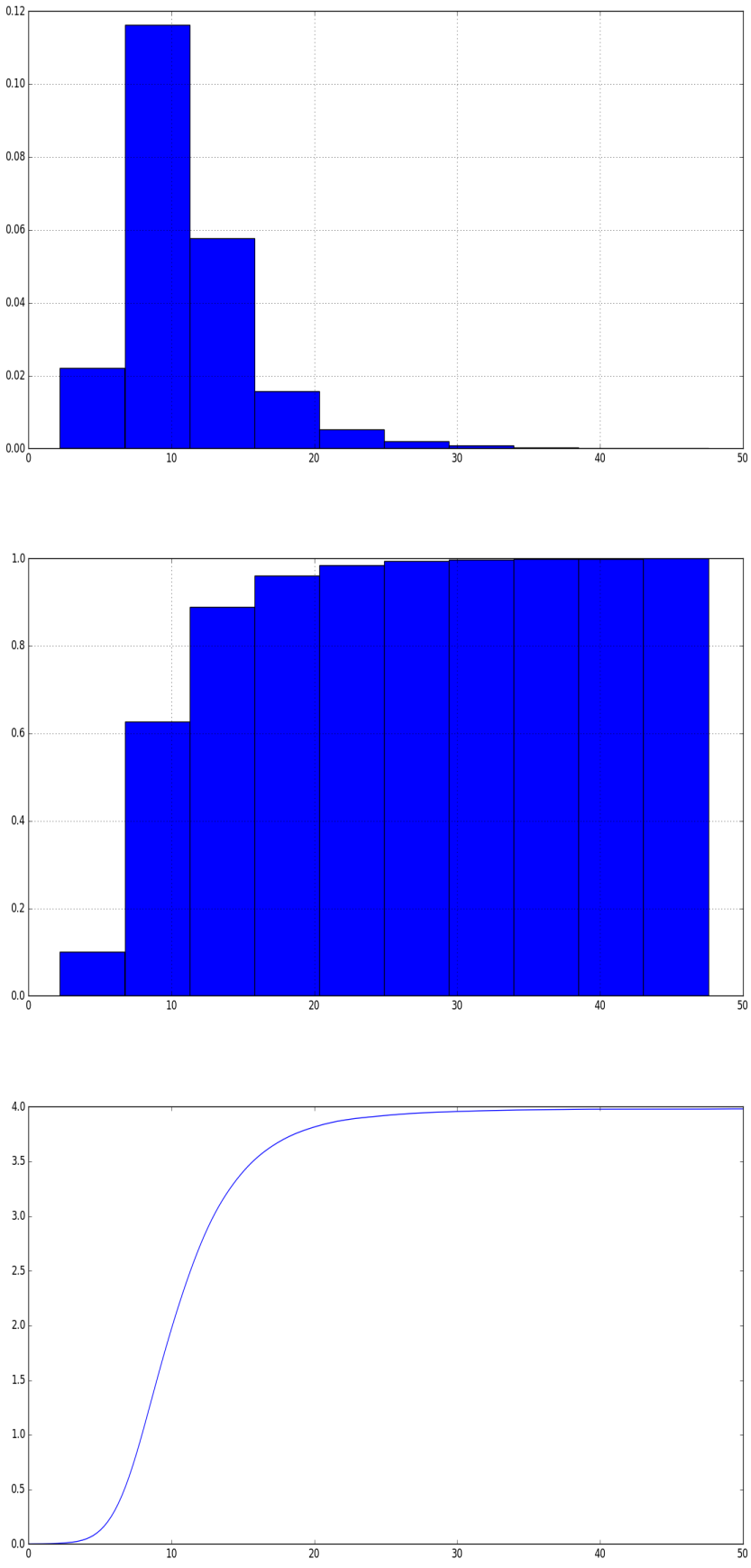

0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
40
50
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
50

图 14.1: 题 1

2.

```
In [11]: history = pd.read_csv('Data/Part2/001/history.csv',index_col='Date')
```

(a)

```
In [12]:revenue=len(history['Emerging.Markets'][history['Emerging.Markets']>0])
```

```
In [13]: loss=len(history['Emerging.Markets'][history['Emerging.Markets']<0])
```

(b)

```
In [14]: p=revenue/(revenue+loss)
```

```
In [15]: 1-stats.binom.cdf(6,12,p)
Out[15]: 0.86700957009568247
```

```
In [16]: from math import sqrt

In [17]: norm_bins=np.linspace(-5, 5, num=200)

In [18]: plt.plot(norm_bins,stats.norm.pdf(norm_bins,0,1),label='N(0,1)')
    ...:
    ...: plt.plot(norm_bins,stats.norm.pdf(norm_bins,0,sqrt(0.5)),
    ...:          label='N(0,0.5)')
    ...:
    ...: plt.plot(norm_bins,stats.norm.pdf(norm_bins,0,sqrt(2)),label='N(0,2)')
    ...:
    ...: plt.plot(norm_bins,stats.norm.pdf(norm_bins,2,1),label='N(2,1)')
    ...:
    ...: plt.legend()
Out[18]: <matplotlib.legend.Legend at 0x7f8c6c47bb38>
```

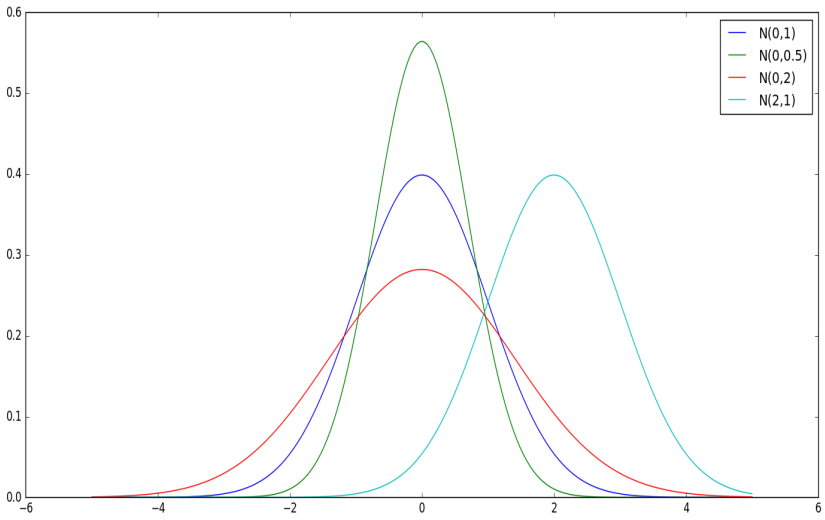


图 14.2: 题 3

(d)

(a)

$$P(x) = C_{x-1}^{r-1} p^r (1-p)^{x-r}$$

(b)

$$\begin{cases} E(x) = \frac{1}{p} \\ Var(x) = \frac{1-p}{p^2} \end{cases}$$

5.

(a)

$$F(x) = -exp\left(-\frac{x}{\lambda}\right)$$

(b)

$$\begin{cases} E(x) = \lambda \\ Var(x) = \lambda^2 \end{cases}$$

(c)

```
In [19]: import numpy as np

In [20]: sample = np.random.exponential(2,10000)

In [21]: sample.mean()
Out[21]: 2.0077693716605811

In [22]: sample.var()
```

```
Out[22]: 3.9950035039304801
```

6.

(a)

$$\begin{cases} E(x) = exp\left(\frac{\mu+\sigma^2}{2}\right) \\ Var(x) = \left(exp\left(\sigma^2\right) - 1\right) exp\left(2\mu + \sigma^2\right) \end{cases}$$

(b) 假设均值和标准差均为 1

```
In [23]: import matplotlib.pyplot as plt

In [24]: log_bins=np.linspace(0, 10, num=200)

In [25]: plt.plot(log_bins,stats.lognorm.pdf(log_bins,1,0,1))
Out[25]: [<matplotlib.lines.Line2D at 0x7f8c6c403c50>]
```

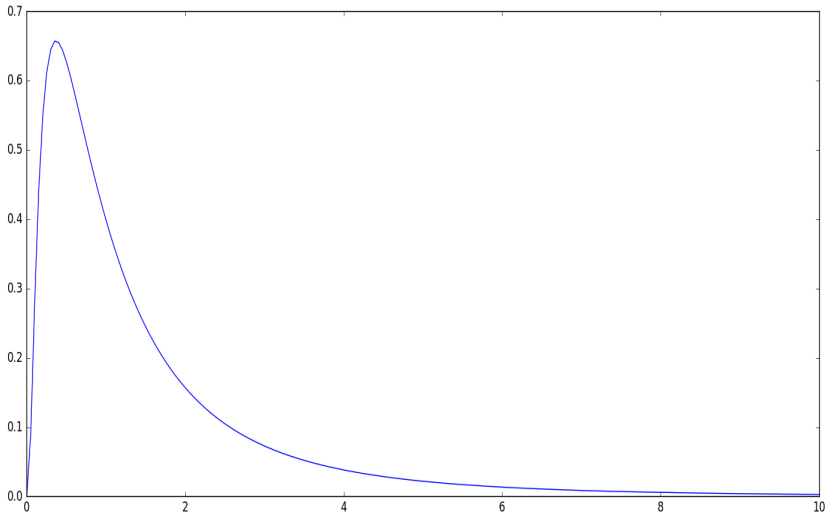


图 14.3: 题 6

第十五章　推断统计

1.

(a)

```
In [1]: import numpy as np

In [2]: import math

In [3]: from scipy import stats

In [4]: mu=np.mean((6.683,6.678,6.767,6.692,6.672,6.678))

In [5]: std=np.std((6.683,6.678,6.767,6.692,6.672,6.678))

In [6]: low = mu - stats.t.ppf(0.95,5) * std / math.sqrt(6)

In [7]: high = mu + stats.t.ppf(0.95,5) * std / math.sqrt(6)

In [8]: low
Out[8]: 6.6680404901748709

In [9]: high
Out[9]: 6.7219595098251279
```

$\therefore \mu$ 的置信水平为 0.9 的置信区间为 $(6.668, 6.722)$

(b)

```
In [9]: mu=np.mean((6.661,6.664,6.668,6.666,6.665))

In [10]: std=np.std((6.661,6.664,6.668,6.666,6.665))

In [11]: low = mu - stats.t.ppf(0.95,4) * std / math.sqrt(5)

In [12]: high = mu + stats.t.ppf(0.95,4) * std / math.sqrt(5)

In [13]: low
Out[13]: 6.6625927405704291
```

```
In [14]: high
Out[14]: 6.6670072594295702
```

$\therefore \mu$ 的置信水平为 0.9 的置信区间为 $(6.663, 6.667)$

```
In [15]: mu=np.mean((5.9,7.3,6.6,5.8,5.7,5.3,5.9,7,6.5))

In [16]: std=np.std((5.9,7.3,6.6,5.8,5.7,5.3,5.9,7,6.5))

In [17]: low = mu - stats.t.ppf(0.975,8) * std / 3

In [18]: high = mu + stats.t.ppf(0.975,8) * std / 3

In [19]: low
Out[19]: 5.7431779225460415

In [20]: high
Out[20]: 6.7012665218984031
```

$\therefore \mu$ 的置信水平为 0.95 的置信区间为 $(5.743, 6.701)$。

2. $\because \bar{x}$ 服从 $N\left(\mu, \frac{\sigma^2}{n}\right)$

$$\therefore \begin{cases} E(\bar{x}) = \mu \\ Var(\bar{x}) = \frac{\sigma^2}{n} \end{cases}$$

$$\therefore \begin{cases} E\left(\frac{\bar{x}-\mu}{\sigma/\sqrt{n}}\right) = \frac{E(\bar{x})-\mu}{\sigma/\sqrt{n}} = 0 \\ Var\left(\frac{\bar{x}-\mu}{\sigma/\sqrt{n}}\right) = \frac{Var(\bar{x}-\mu)}{\sigma^2/n} = \frac{\sigma^2/n}{\sigma^2/n} = 1 \end{cases}$$

又因为服从正态分布的随机变量和一个常数进行加减乘除运算后仍服从正态分布，所以 $\frac{\bar{x}-\mu}{\sigma/\sqrt{n}} \sim N(0,1)$

4. 一阶矩估计量为样本均值：$\bar{x} = \frac{x_1+x_2+\ldots+x_n}{n}$；

二阶矩估计量为样本方差：$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$

5.

```
In [1]: import pandas as pd

In [2]: import matplotlib.pyplot as plt

In [3]: Bwages = pd.read_csv(' Bwages.csv ')

In [4]: mu = Bwages.wage.mean()

In [5]: std=Bwages.wage.std()
```

```
In [6]: low = mu - stats.t.ppf(0.975,len(Bwages)-1) * std / math.sqrt(len(Bwages)
    )

In [7]: high = mu + stats.t.ppf(0.975,len(Bwages)-1) * std / math.sqrt(len(Bwages
    ))

In [8]: low
Out[8]: 10.823074026756808

In [9]: high
Out[9]: 11.278158191999699
```

∴wage 的置信水平为 0.95 的置信区间为 $(10.82307, 11.27816)$。

6.

```
In [10]: Bwages.wage.hist(normed=True)
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa538a0a940>

In [11]: bins=np.linspace(0,50,200)

In [12]: plt.plot(bins,stats.norm.pdf(bins,mu,std))
Out[12]: [<matplotlib.lines.Line2D at 0x7fa53808cb38>]
```

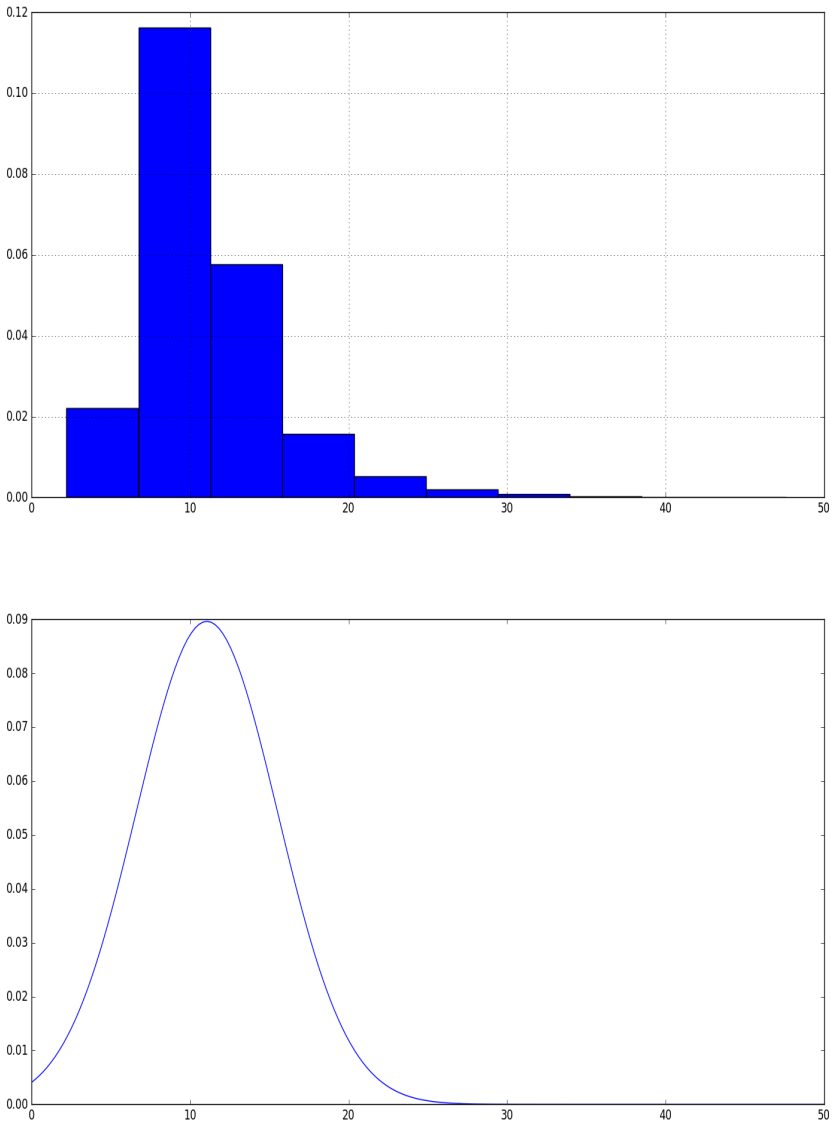


图 15.1: 题 6

可以看到，wage 的频率直方图与正态分布的曲线是较为接近的，所以我们的假设是合理的。

7.

```
In [1]: from scipy import stats

In [2]: MT=(0.225,0.262,0.217,0.24,0.23,0.229,0.235,0.217)

In [3]: Sn=(0.209,0.205,0.196,0.21,0.202,0.207,0.224,0.223)

In [4]: stats.ttest_ind(MT,Sn)
Out[4]: Ttest_indResult(statistic=3.6170843653195446,
        pvalue=0.0028018899619853565)
```

因为 p 值为 0.002802，小于 0.05，所以我们应当拒绝原假设。即，两位作家所写的小品文中包含由 3 个字母组成的单字的比例具有显著性的差异。

8.
```
In [1]: from scipy import stats

In [2]: region1=(168,180,181,172,165,160,166,165,177,174)

In [3]: region2=(169,170,176,173,166,167,166,173,171,170)

In [4]: stats.ttest_ind(region1,region2)
Out[4]: Ttest_indResult(statistic=0.28303531904272478,
  pvalue=0.78037906653623068)
```

因为 p 值为 0.7804，大于 0.05，所以我们不能拒绝原假设。因此，这两个地区的人的身高不具有显著性差异。

9.
```
In [1]: import pandas as pd

In [2]: from scipy import stats

In [3]: Bwages = pd.read_csv('Data/Part2/002/Bwages.csv')

In [4]: stats.ttest_1samp(Bwages.wage,11)
Out[4]: Ttest_1sampResult(statistic=0.4363476178926709, pvalue
   =0.66264857119758025)
```

因为 p 值为 0.6626，大于 0.05，所以我们不能拒绝原假设。

10.
```
In [1]: import pandas as pd

In [2]: from scipy import stats

In [3]: history = pd.read_csv('Data/Part2/001/history.csv',
   ...:                       index_col='Date')

In [4]: stats.ttest_ind(history['Emerging.Markets'],
   ...:         history['Global.Macro'])
Out[4]: Ttest_indResult(statistic=0.1677641921908746,
pvalue=0.86688108106493611)
```

因为 p 值为 0.8669，大于 0.05，所以我们不能拒绝原假设。检验新兴市场风格对冲基金和全球宏观风格对冲基金收益率不具有显著性差异。

11.
```
In [5]: stats.ttest_rel(history['Emerging.Markets'],
   ....:         history['Global.Macro'])
Out[5]: Ttest_relResult(statistic=0.23846933281992622,
```

```
4 pvalue=0.81184041604248691)
```

因为 p 值为 0.8118，大于 0.05，所以我们不能拒绝原假设。

第十六章　方差分析

1.

$$\begin{aligned} FSS &= \sum_{i=1}^{5} 10 \times (\bar{x_i} - \bar{x})^2 \\ &= 10 \times (11.4 - 10.7)^2 + 10 \times (11.7 - 10.7)^2 \\ &+ 10 \times (9.9 - 10.7)^2 + 10 \times (9.8 - 10.7)^2 + 10 \times (7.9 - 10.7)^2 \\ &= 107.8 \end{aligned}$$

$$ESS = \sum_{j=1}^{5} \sum_{i=1}^{10} (x_{ij} - \bar{x_j})^2 = 445.9$$

$$F = \frac{FSS/4}{ESS/45} = 2.72$$

我们可以查到 $F_{4,45,0.95}$ 的值为 2.56。因为 F 统计量的值大于 2.56，所以我们应当拒绝原假设。即，5 个地区的单日开户数量具有显著性差异。

2.

$$\begin{aligned} FSS = &\sum_{i=1}^{4} 5 \times (\bar{x_i} - \bar{x})^2 \\ &= 5 \times (1.138\% - 0.5785\%)^2 + 5 \times (0.002\% - 0.5785\%)^2 \\ &+ 5 \times (0.22\% - 0.5785\%)^2 + 5 \times (0.954\% - 0.5785\%)^2 \\ &= 0.046\% \end{aligned}$$

$$ESS = \sum_{j=1}^{4} \sum_{i=1}^{5} (x_{ij} - \bar{x_j})^2 = 0.42\%$$

$$F = \frac{FSS/3}{ESS/16} = 0.58$$

我们可以查到 $F_{3,16,0.95}$ 的值为 3.24。因为 F 统计量的值小于 3.24，所以我们不能拒绝原假设。即，调息方案对上证综指不具有显著性影响。

3.

$$\begin{aligned} FSS &= \sum_{i=1}^{3} 5 \times (\bar{x}_i - \bar{x})^2 \\ &= 5 \times (43.6 - 39.87)^2 + 5 \times (30.2 - 39.87)^2 + 5 \times (45.8 - 39.87)^2 \\ &= 712.93 \end{aligned}$$

$$ESS = \sum_{j=1}^{3} \sum_{i=1}^{5} (x_{ij} - \bar{x}_j)^2 = 278.8$$

$$F = \frac{FSS/2}{ESS/12} = 15.34$$

我们可以查到 $F_{2,12,0.95}$ 的值为 3.89。因为 F 统计量的值大于 3.89，所以我们应当拒绝原假设。即，电池的平均寿命具有显著性的差异。

4.

```
In [1]: import pandas as pd

In [2]: managers = pd.read_csv('Data/Part2/004/managers.csv',index_col='Date')

In [3]: MANA = managers.iloc[:,(0,2,3)]
```

(a)

```
In [4]: ((MANA.HAM1-MANA.HAM1.mean())**2).sum()
Out[4]: 0.08604549181818183

In [5]: ((MANA.HAM3-MANA.HAM3.mean())**2).sum()
Out[5]: 0.1746451887878788

In [6]: ((MANA.HAM4-MANA.HAM4.mean())**2).sum()
Out[6]: 0.37073304333333335
```

```
In [7]: mu = MANA.mean().mean()

In [8]: ((MANA.HAM1.mean()-mu)**2 + (MANA.HAM3.mean()-mu)**2 +(MANA.HAM4.mean
    ()-mu)**2) * 132
Out[8]: 0.00016766787878787911
```

(b)

```
In [9]: 0.0001676679+0.08604549+0.1746452+0.370733
Out[9]: 0.6315913579
```

(d) ESS、FSS、TSS 的自由度分别为 393、2、395。三者的关系为：$df_{ESS} + df_{FSS} = df_{TSS}$。

(e) $F = \frac{0.0002/2}{(0.086+0.175+0.371)/393} = 0.062$

我们可以查到 $F_{2,12,0.95}$ 的值为 3.07。因为 F 统计量的值小于 3.07，所以我们不能拒绝原假设。即，这三者的平均收益率不具有显著性差异。

```
In [10]: from statsmodels.formula.api import ols

In [11]: import statsmodels.stats.anova as anova

In [12]: returns = pd.DataFrame(pd.concat([MANA.HAM1,MANA.HAM3,MANA.HAM4]))

In [13]: returns['Class'] = ['HAM1' for i in range(132)]+['HAM3' for i in range
    (132)]+['HAM4' for i in range(132)]

In [14]: returns.columns = ['Return','Class']

In [15]: model = ols('Return~C(Class)',data=returns).fit()

In [16]: print(anova.anova_lm(model))
                df     sum_sq   mean_sq         F    PR(>F)
C(Class)       2.0   0.000168  0.000084  0.052178  0.949166
Residual     393.0   0.631424  0.001607       NaN       NaN
```

可以看到，anova.anova_lm( ) 得到的 F 取值与我们自己计算得到的结果很相近。

第十七章　回归

5.

(a)

```
In [1]: import matplotlib.pyplot as plt

In [2]: x = list(range(1952,2016,4))

In [3]: y = (29.3,28.8,28.5,28.4,29.4,27.6,27.7,27.7,
   ...:          27.8,27.4,27.8,27.1,27.3,27.1,27.0,27.5)

In [4]: plt.plot(x,y)
Out[4]: [<matplotlib.lines.Line2D at 0x7f4eec296518>]
```

(b) $\bar{x} = \frac{\sum_{i=1}^{16} x_i}{16} = 1982$

$\bar{y} = \frac{\sum_{i=1}^{16} y_i}{16} = 27.9$

$\hat{b} = \frac{\sum_{i=1}^{16}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{16}(x_i-\bar{x})^2} = -0.034$

$\hat{a} = \bar{y} - \hat{b}\bar{x} = 94.647$

y 关于 x 的线性回归方程为 $y = 94.647 - 0.034x$

```
In [5]: import statsmodels.api as sm

In [6]: model=sm.OLS(y ,sm.add_constant(x)).fit()

In [7]: print(model.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.711
Model:	OLS	Adj. R-squared:	0.690
Method:	Least Squares	F-statistic:	34.41
Date:	Fri, 20 Jan 2017	Prob (F-statistic):	4.11e-05
Time:	18:20:40	Log-Likelihood:	-7.8861
No. Observations:	16	AIC:	19.77
Df Residuals:	14	BIC:	21.32
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]	
const	94.6468	11.380	8.317	0.000	70.239	119.054
x1	-0.0337	0.006	-5.866	0.000	-0.046	-0.021

Omnibus:	5.698	Durbin-Watson:	2.210
Prob(Omnibus):	0.058	Jarque-Bera (JB):	3.021
Skew:	0.985	Prob(JB):	0.221
Kurtosis:	3.807	Cond. No.	2.13e+05

(c) $t = \frac{\hat{b}\sqrt{\sum_{i=1}^{16}(x_i-\bar{x})^2}}{\hat{\sigma}} = -5.866$

因为 $t_{14,0.975} = 2.145 < |t|$，所以我们可以拒绝原假设。

(d) 将 $x = 2016$ 代入可得：$y = 26.103$。所以，2016 年里约奥运会男子 10000 米冠军的成绩的预测值为 26.103。

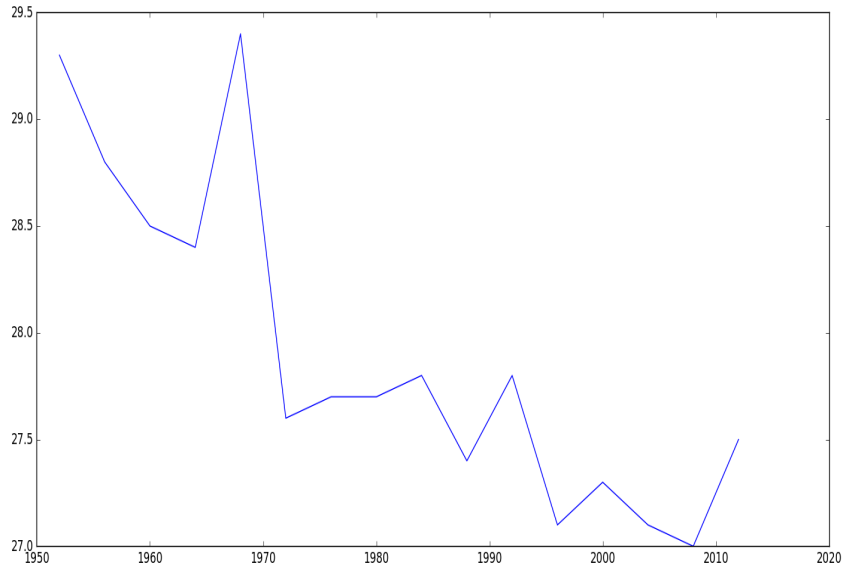


图 17.1: 题 1

2.

```
In [1]: import pandas as pd

In [2]: EU = pd.read_csv('Data/Part2/005/EuStockMarkets.csv')

In [3]: plt.plot(EU.DAX,EU.FTSE,'.')
   ...:
   ...: plt.xlabel('DAX')
```

```
    ...:
    ...: plt.ylabel('FTSE')
Out[3]: <matplotlib.text.Text at 0x7f4eec208f98>
```

很明显，两者存在着线性正相关关系。

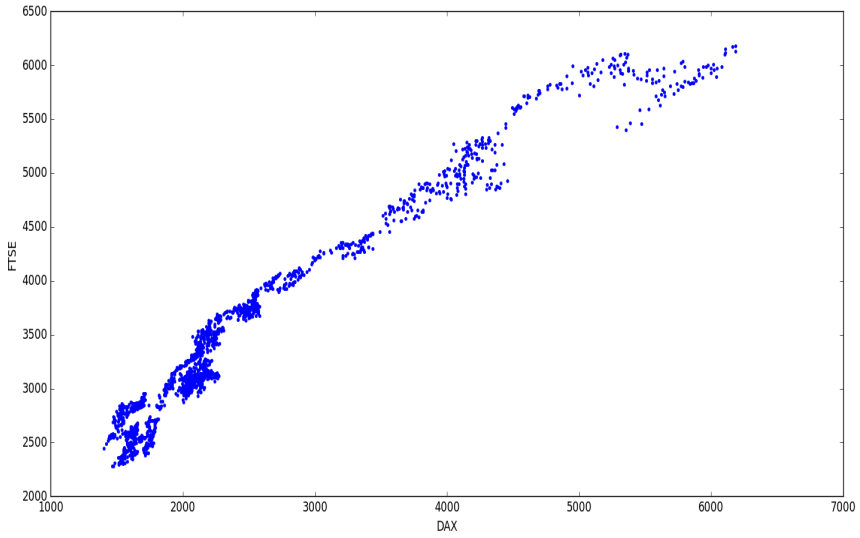


图 17.2: 题 2

3.

(a)

```
In [1]: import statsmodels.api as sm
In [2]: model = sm.OLS(EU.DAX,sm.add_constant(EU.FTSE)).fit()
```

```
In [3]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    DAX   R-squared:                       0.951
Model:                            OLS   Adj. R-squared:                  0.951
Method:                 Least Squares   F-statistic:                 3.604e+04
Date:                Tue, 17 Jan 2017   Prob (F-statistic):               0.00
Time:                        19:49:50   Log-Likelihood:                -12834.
No. Observations:                1860   AIC:                         2.567e+04
Df Residuals:                    1858   BIC:                         2.568e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const      -1331.2374     21.092    -63.115      0.000     -1372.604 -1289.871
FTSE           1.0831      0.006    189.838      0.000         1.072     1.094
==============================================================================
Omnibus:                      329.509   Durbin-Watson:                   0.012
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              612.319
Skew:                           1.084   Prob(JB):                    1.09e-133
Kurtosis:                       4.789   Cond. No.                     1.40e+04
==============================================================================
```

(b)

```
In [4]: plt.plot(EU.FTSE,EU.DAX,'.',EU.FTSE,model.fittedvalues,'-')
   ...:
   ...: plt.xlabel('FTSE')
   ...:
   ...: plt.ylabel('DAX')
Out[4]: <matplotlib.text.Text at 0x7f4ed2e6e4e0>
```

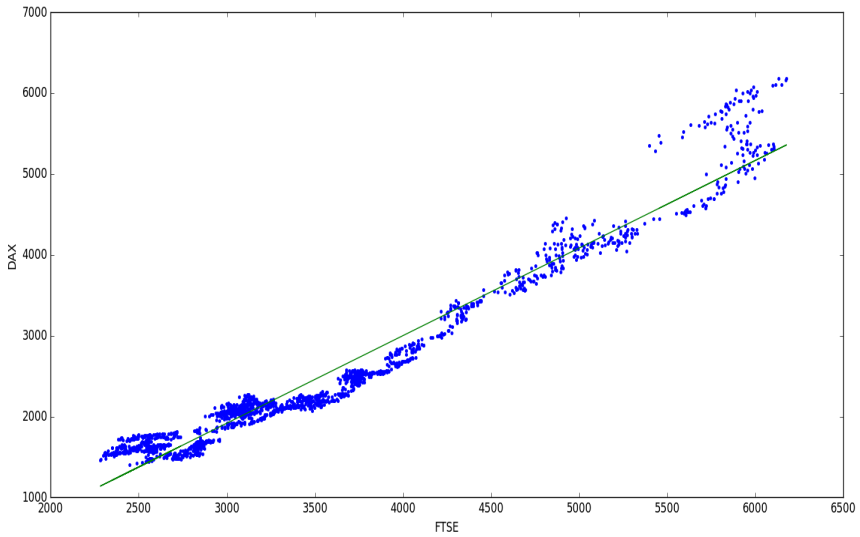
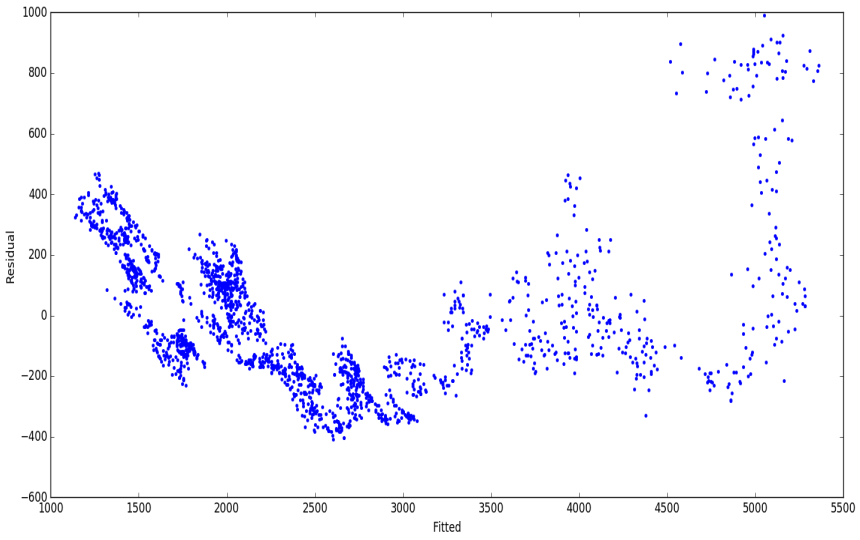


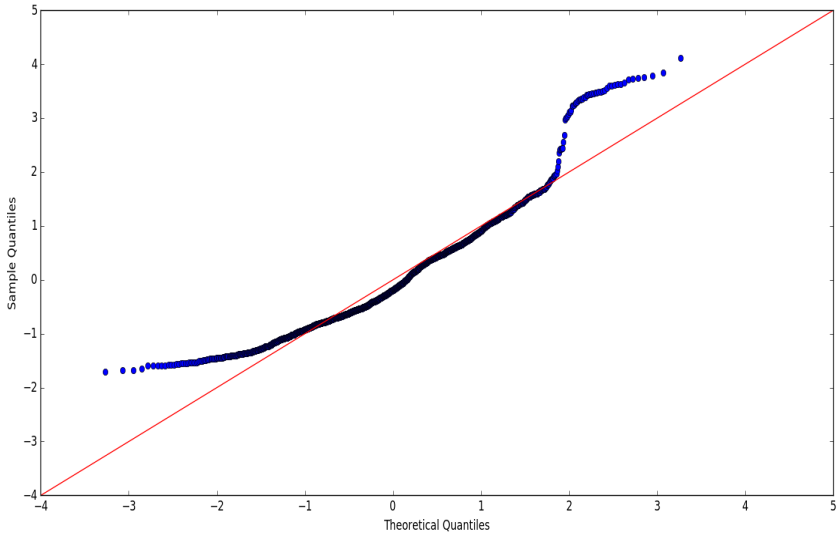
图 17.3: 题 3

4.

```
In [1]: plt.plot(model.fittedvalues,model.resid,'.')
   ...:
   ...: plt.xlabel('Fitted')
   ...:
   ...: plt.ylabel('Residual')
Out[1]: <matplotlib.text.Text at 0x7f4ed2e58748>

In [2]: import scipy.stats as stats

In [3]: sm.qqplot(model.resid_pearson,stats.norm,line='45')

In [4]: plt.plot(model.fittedvalues,model.resid_pearson**0.5,'.')
   ...:
   ...: plt.xlabel('Fitted')
   ...:
   ...: plt.ylabel('Square Root of Standardized Residual')
```

在第一张图和第三张图中，我们都可以看到很明显的趋势。这说明线性及同方差的条件并没有被满足。而第二张图也存在严重的偏离，表明因变量并不服从正态分布。因此，我们的模型并不能很好地模拟两者之间的关系。实际上，DAX 指数与 FTSE 指数属于时间序列，我们应该使用时间序列模型而不是线性回归模型。


Residual
Fitted


Sample Quantiles
Theoretical Quantiles

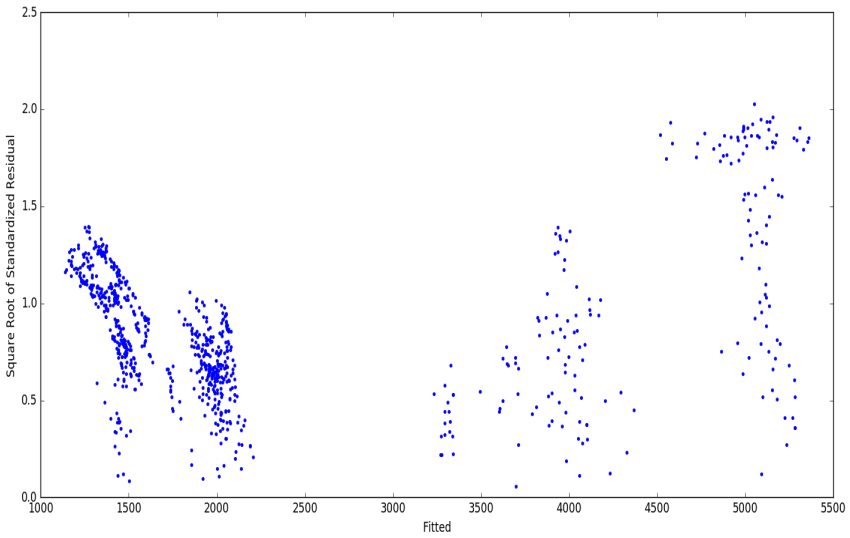

Square Root of Standardized Residual
Fitted

图 17.4: 题 4

5.

(a)

```
In [1]: import matplotlib.pyplot as plt
   ...: import statsmodels.api as sm
   ...: import numpy as np
   ...:

In [2]: x = [20,25,30,35,40,50,60,65,70,75,80,90]

In [3]: y = [1.81,1.7,1.65,1.55,1.48,1.4,1.3,1.26,1.24,1.21,1.2,1.18]

In [4]: plt.plot(x,y,'.')
Out[4]: [<matplotlib.lines.Line2D at 0x7f4ed2cddb70>]
```

```
In [5]: independent = np.array([x,[i**2 for i in x]]).T

In [6]: model = sm.OLS(y,sm.add_constant(independent)).fit()

In [7]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.997
Model:                            OLS   Adj. R-squared:                  0.997
Method:                 Least Squares   F-statistic:                     1767.
Date:                Tue, 17 Jan 2017   Prob (F-statistic):           2.10e-12
Time:                        20:01:19   Log-Likelihood:                 37.586
No. Observations:                  12   AIC:                            -69.17
Df Residuals:                       9   BIC:                            -67.72
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          2.1983      0.023     97.485      0.000         2.147     2.249
x1            -0.0225      0.001    -23.898      0.000        -0.025    -0.020
x2             0.0001   8.66e-06     14.446      0.000         0.000     0.000
==============================================================================
Omnibus:                        0.874   Durbin-Watson:                   2.783
Prob(Omnibus):                  0.646   Jarque-Bera (JB):                0.642
Skew:                          -0.096   Prob(JB):                        0.726
Kurtosis:                       1.884   Cond. No.                     2.65e+04
==============================================================================
```

(b) 根据回归结果中的 F Statistic，我们可以拒绝原假设。

(d)

```
In [8]: model.predict(np.array([1,95,95**2]).T)
Out[8]: array([ 1.18735552])
```

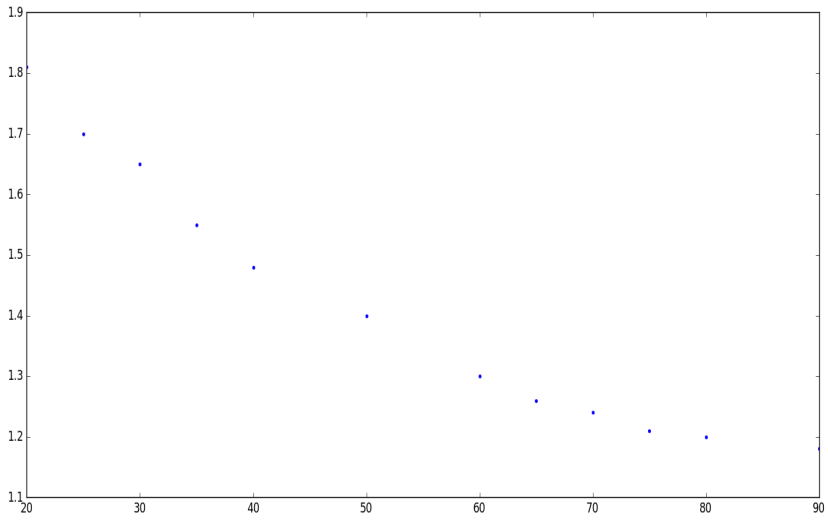


图 17.5: 题 4

6.

```
In [1]: import pandas as pd

In [2]: import statsmodels.formula.api as smf

In [3]: import numpy as np

In [4]: cps = pd.read_csv('Data/Part2/005/CPS1988.csv')
```

(a)

```
In [5]: cps.head()
Out[5]:
     wage  education  experience ethnicity smsa     region parttime
0  354.94          7          45      cauc  yes  northeast       no
1  123.46         12           1      cauc  yes  northeast      yes
2  370.37          9           9      cauc  yes  northeast       no
3  754.94         11          46      cauc  yes  northeast       no
4  593.54         12          36      cauc  yes  northeast       no
```

```
In [6]: model = smf.ols('np.log(wage)~experience+education+ethnicity',
   ...:                 data=cps).fit()
```

(b)

```
In [7]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           np.log(wage)   R-squared:                       0.221
Model:                            OLS   Adj. R-squared:                  0.221
Method:                 Least Squares   F-statistic:                     2665.
Date:                Tue, 17 Jan 2017   Prob (F-statistic):               0.00
Time:                        20:04:22   Log-Likelihood:                -27020.
No. Observations:               28155   AIC:                         5.405e+04
Df Residuals:                   28151   BIC:                         5.408e+04
Df Model:                           3
Covariance Type:            nonrobust
==========================================================================================
                             coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------
Intercept                  4.2889      0.023    183.656      0.000          4.243      4.335
ethnicity[T.cauc]          0.2431      0.014     17.395      0.000          0.216      0.271
experience                 0.0196      0.000     65.198      0.000          0.019      0.020
education                  0.0996      0.001     73.258      0.000          0.097      0.102
==============================================================================
Omnibus:                     2846.769   Durbin-Watson:                   1.814
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4634.595
Skew:                          -0.733   Prob(JB):                         0.00
Kurtosis:                       4.343   Cond. No.                         163.
==============================================================================
```

(d)

```
In [8]: model.fittedvalues.head()
Out[8]:
0    6.110772
1    5.746557
2    5.604576
3    6.528665
4    6.432310
dtype: float64
```

```
In [9]: model2 = smf.ols(' np.log(wage)~experience+np.power(experience,2)+
   ...:                   education+ethnicity ',
   ...:                   data=cps).fit()
```

7. (a)

```
In [10]: print(model2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           np.log(wage)   R-squared:                       0.335
Model:                            OLS   Adj. R-squared:                  0.335
Method:                 Least Squares   F-statistic:                     3541.
Date:                Tue, 17 Jan 2017   Prob (F-statistic):               0.00
Time:                        20:06:23   Log-Likelihood:                -24801.
No. Observations:               28155   AIC:                         4.961e+04
Df Residuals:                   28150   BIC:                         4.965e+04
Df Model:                           4
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------------------------
Intercept                        4.0780      0.022    187.086      0.000          4.035      4.121
ethnicity[T.cauc]                0.2434      0.013     18.839      0.000          0.218      0.269
experience                       0.0775      0.001     88.033      0.000          0.076      0.079
np.power(experience, 2)         -0.0013    1.9e-05    -69.312      0.000         -0.001     -0.001
education                        0.0857      0.001     67.343      0.000          0.083      0.088
==============================================================================
Omnibus:                     2034.828   Durbin-Watson:                   1.795
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3963.309
Skew:                          -0.508   Prob(JB):                         0.00
Kurtosis:                       4.532   Cond. No.                     5.23e+03
==============================================================================
```

(b) 首先，从单个变量上看，$experience^2$ 的系数是显著的。这说明我们应该将其加入模型。而且，加入 $experience^2$ 后，模型的 R^2 得到了提升。因此，相较于第一个模型，第二个模型要更好。

8. 从结果来看，F 统计量为 3541，p 值远小于 0.05。因此，我们应该拒绝原假设。

9. 从结果来看，β_3 的 p 值远小于 0.05。因此，我们应当拒绝原假设。也就是说，β_3 显著不为 0。

第三部分

金融

第十八章　资产收益率和风险

1.

(a) $R_4 = \frac{P_4 - P_3}{P_3} = -4.4\%$

(b) $R_5(2) = \frac{P_5 - P_3}{P_3} = -7.2\%$

(c) $R_{12}(8) = \frac{P_{12} - P_4}{P_4} = -1.2\%$

2. $R_{12}(8) = \prod_{i=4}^{12}(1 + R_i) - 1$

3. $R_t = \frac{Pt - D - P_{t-1}}{P_{t-1}} = -8.3\%$

4. $(1 + R)^{365} = 1 + 0.03416$

   解得：$R = 0.0092\%$

5. $R = 1.5\% \times \frac{12}{3} = 6\%$

6. $R_t = lim_{n\to\infty}\left(1 + \frac{r_t}{n}\right)^n - 1 = lim_{n\to\infty}\left(1 + \frac{r_t}{n}\right)^{\frac{n}{r_t}r_t} - 1 = e^{r_t} - 1$

7. $ln(1 + R) = 0.06$

   $R = 6.18\%$

8. $R = 0.5\% \times 12 = 6\%$

9.

```
In [1]: import pandas_datareader.data as web

In [2]: import datetime as dt

In [3]: import numpy as np
```

(a)

```
In [4]: start = dt.datetime(2014,1,1)

In [5]: end = dt.datetime(2014,12,31)

In [6]: baidu = web.DataReader('BIDU','yahoo',start,end)
```

```
In [7]: returns = (baidu.Close - baidu.Close.shift(1))/baidu.Close.shift(1)
```

(c)

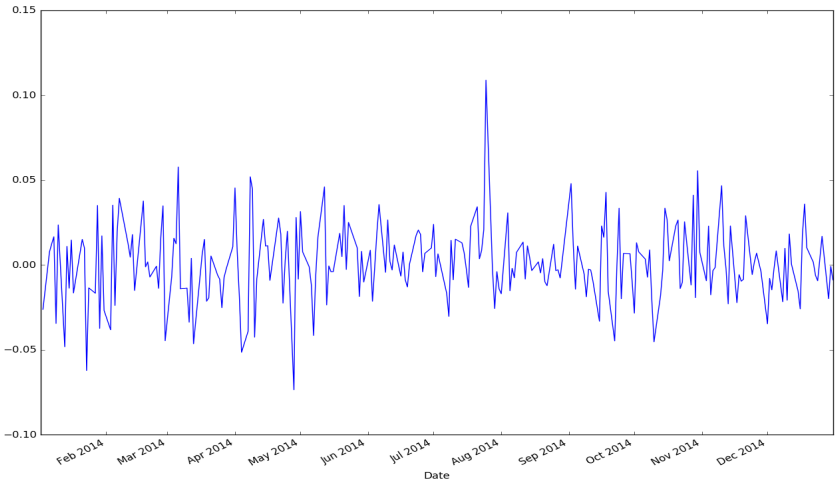
```
In [8]: comp_returns = np.log(baidu.Close/baidu.Close.shift(1))
```

```
In [9]: comp_returns.sum()
Out[9]: 0.23631272416563825
```

(d)

```
In [10]: returns.plot()
```

```
In [11]: comp_returns.plot()
```



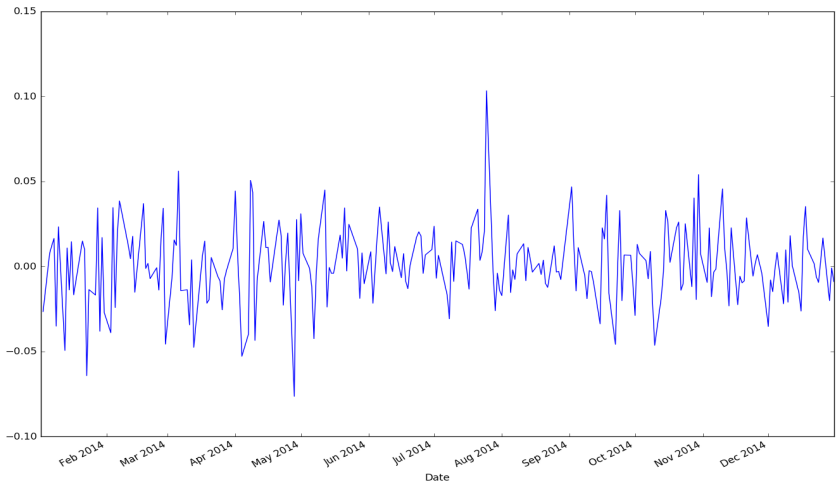


图 18.1: 题 1

(f)

10. 略

第十九章　投资组合理论及其拓展

1. 组合 A 和组合 E 的收益率都是 0.05，但是组合 E 的风险更高，因此不是有效的。同理，组合 B、C 也是无效的。组合 A 的收益和风险皆比组合 D 低，因此两者无法比较，都有可能是有效的。

2. 股票 A 和股票 D。因为两者的相关系数最小。

3.

   (a) $E(R) = 0.3 \times 8\% + 0.7 \times 6\% = 6.6\%$

   $\sigma = \sqrt{(0.3 \times 0.02)^2 + (0.7 \times 0.03)^2 + 2 \times 0.3 \times 0.7 \times 0.5 \times 0.02 \times 0.03} = 0.025$

   (b) $E(R) = 0.6 \times 8\% + 0.4 \times 6\% = 7.2\%$

   $\sigma = \sqrt{(0.6 \times 0.02)^2 + (0.4 \times 0.03)^2 + 2 \times 0.6 \times 0.4 \times 0.5 \times 0.02 \times 0.03} = 0.021$

4. 因为收益率为 5%，所以我们只有两种可能的组合，即 A 和 B、A 和 C。当组合为 A 和 B 时，A 和 B 的比重分别为 0.6 和 0.4，组合的标准差为 0.016。当组合为 A 和 C 时，两者的比重分别为 0.2 和 0.8，组合的标准差为 0.023。所以。我们应该选择组合 A 和 B。

5. 直接利用文中的函数，我们可以绘得最小方差前缘曲线。

```
In [1]: %paste
import numpy as np
import math
import matplotlib.pyplot as plt
import ffn
import pandas as pd
import pandas_datareader.data as web
import datetime as dt
from scipy import linalg

class MeanVariance:
    def __init__(self,returns):
        self.returns=returns
    def minVar(self,goalRet):
        covs=np.array(self.returns.cov())
        means=np.array(self.returns.mean())
        L1=np.append(np.append(covs.swapaxes(0,1),[means],0),
                     [np.ones(len(means))],0).swapaxes(0,1)
```

```
        L2=list(np.ones(len(means)))
        L2.extend([0,0])
        L3=list(means)
        L3.extend([0,0])
        L4=np.array([L2,L3])
        L=np.append(L1,L4,0)
        results=linalg.solve(L,np.append(np.zeros(len(means)),[1,goalRet],0))
        return(np.array([list(self.returns.columns),results[:-2]]))
    def frontierCurve(self):
        goals=[x/500000 for x in range(-100,4000)]
        variances=list(map(lambda x: self.calVar(self.minVar(x)[1,:].astype(np.
            float)),goals))
        plt.plot(variances,goals)
    def meanRet(self,fracs):
        meanRisky=ffn.to_returns(self.returns).mean()
        assert len(meanRisky)==len(fracs),'Length of fractions must be equal to
            number of assets'
        return(np.sum(np.multiply(meanRisky,np.array(fracs))))
    def calVar(self,fracs):
        return(np.dot(np.dot(fracs,self.returns.cov()),fracs))
```

获取数据，绘制最小方差前缘曲线。

(a)

```
In [2]: %paste
sclq = web.DataReader('600039.SS','yahoo',
            dt.datetime(2009,1,2),dt.datetime(2012,12,31)).Close
sclqRet=((sclq-sclq.shift(1))/sclq.shift(1))
sclqRet.name='sclq'

## -- End pasted text --

In [3]: %paste
hsly = web.DataReader('600054.SS','yahoo',
            dt.datetime(2009,1,2),dt.datetime(2012,12,31)).Close
hslyRet=((hsly-hsly.shift(1))/hsly.shift(1))
hslyRet.name='hsly'

## -- End pasted text --

In [4]: %paste
wldc = web.DataReader('600173.SS','yahoo',
```

```
                dt.datetime(2009,1,2),dt.datetime(2012,12,31)).Close
wldcRet=((wldc-wldc.shift(1))/wldc.shift(1))
wldcRet.name= 'wldc'

## -- End pasted text --

In [5]: %paste
ghny = web.DataReader('600256.SS','yahoo',
                dt.datetime(2009,1,2),dt.datetime(2012,12,31)).Close
ghnyRet=((ghny-ghny.shift(1))/ghny.shift(1))
ghnyRet.name= 'ghny'

## -- End pasted text --

In [6]: %paste
zhjt = web.DataReader('600252.SS','yahoo',
               dt.datetime(2009,1,2),dt.datetime(2012,12,31)).Close
zhjtRet=((zhjt-zhjt.shift(1))/zhjt.shift(1))
zhjtRet.name= 'zhjt'

## -- End pasted text --

In [7]: %paste
fiveStocks=pd.concat([sclqRet,hslyRet,wldcRet,ghnyRet,zhjtRet],1)
fiveStocks=fiveStocks.dropna()

## -- End pasted text --

In [8]: minVar=MeanVariance(fiveStocks)

In [9]: minVar.frontierCurve()
```

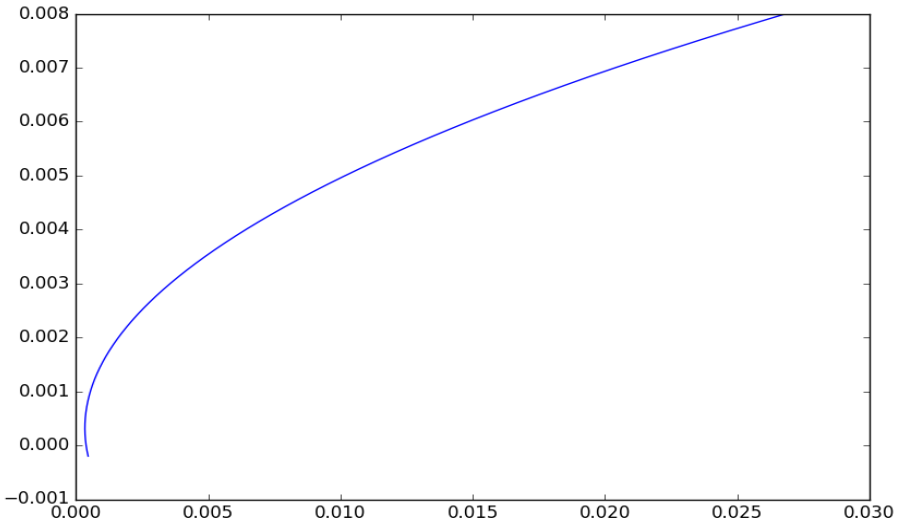


图 19.1: 题 5

```
In [10]: %paste
goal_return=0.05
portfolio_weight=minVar.minVar(goal_return)
portfolio_weight

## -- End pasted text --
Out[10]:
array([['sclq', 'hsly', 'wldc', 'ghny', 'zhjt'],
       ['9.040199555721635', '-44.98072150897727', '9.347995192679459',
        '24.975718390028632', '2.6168083705475422']],
      dtype='<U32')
```

6. 选择矩阵：

$$\begin{array}{cccc} A & B & C & D \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \\ 1 & 0 & -1 & 0 \end{array}$$

看法向量：$(0.06, 0.02, 0.03)$

信心向量：$(0.85, 0.90, 0.75)$

8. 计算步骤与文中一样。

```
In [1]: %paste
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
```

```
import pandas_datareader.data as web
import datetime as dt
import ffn
from scipy import linalg

## -- End pasted text --

In [2]: %paste
def blacklitterman(returns,tau, P, Q):
  mu=returns.mean()
  sigma=returns.cov()
  pi1=mu
  ts = tau * sigma
  Omega = np.dot(np.dot(P,ts),P.T) * np.eye(Q.shape[0])
  middle = linalg.inv(np.dot(np.dot(P,ts),P.T) + Omega)
  er = np.expand_dims(pi1,axis=0).T + np.dot(np.dot(np.dot(ts,P.T),middle),
                        (Q - np.expand_dims(np.dot(P,pi1.T),axis=1)))
  posteriorSigma = sigma + ts - np.dot(ts.dot(P.T).dot(middle).dot(P),ts)
  return [er, posteriorSigma]

def blminVar(blres,goalRet):
    covs=np.array(blres[1])
    means=np.array(blres[0])
    L1=np.append(np.append((covs.swapaxes(0,1)),[means.flatten()],0),
                              [np.ones(len(means))],0).swapaxes(0,1)
    L2=list(np.ones(len(means)))
    L2.extend([0,0])
    L3=list(means)
    L3.extend([0,0])
    L4=np.array([L2,L3])
    L=np.append(L1,L4,0)
    results=linalg.solve(L,np.append(np.zeros(len(means)),[1,goalRet],0))
    return(pd.DataFrame(results[:-2],
                          index=blres[1].columns,columns=[ p_weight' ]))
## -- End pasted text --

In [3]: %paste
pick1 = np.array([1,0,0,0])
pick2 = np.array([0,-1,0,1])
pick3 = np.array([1,0,-1,0])
```

```
P = np.array([pick1,pick2,pick3])
Q=np.array([[0.06],[0.02],[0.03]])

## -- End pasted text --

In [4]: %paste
lcxx = web.DataReader('000977.SZ','yahoo',
                dt.datetime(2012,1,2),dt.datetime(2014,12,31)).Close
lcxxRet=((lcxx-lcxx.shift(1))/lcxx.shift(1))
lcxxRet.name='lcxx'

pfyh = web.DataReader('600000.SS','yahoo',
                dt.datetime(2012,1,2),dt.datetime(2014,12,31)).Close
pfyhRet=((pfyh-pfyh.shift(1))/pfyh.shift(1))
pfyhRet.name='pfyh'

htzq = web.DataReader('601688.SS','yahoo',
                dt.datetime(2012,1,2),dt.datetime(2014,12,31)).Close
htzqRet=((htzq-htzq.shift(1))/htzq.shift(1))
htzqRet.name='htzq'

zqb = web.DataReader('300052.SZ','yahoo',
                dt.datetime(2012,1,2),dt.datetime(2014,12,31)).Close
zqbRet=((zqb-zqb.shift(1))/zqb.shift(1))
zqbRet.name='zqb'

## -- End pasted text --

In [5]: %paste
Stocks=pd.concat([lcxxRet,pfyhRet,htzqRet,zqbRet],1)
Stocks=Stocks.dropna()

## -- End pasted text --

In [6]: %paste
res=blacklitterman(Stocks,0.1, P, Q)
blminVar(res,0.75/252)

## -- End pasted text --
Out[6]:
```

```
        p_weight
lcxx   -0.015149
pfyh    0.810286
htzq    0.082629
zqb     0.122235
```

9. 假设投资组合为 $w=\sum_{i=1}^{5} w_i x_i$，那么我们需要求解的方差最小化问题则为：

$$\min Var\left(w\right)$$

subject to

$$\begin{cases} \sum_{i=1}^{5} w_i = 1 \\ \sum_{i=1}^{5} w_i \bar{R}_i = 0.05 \end{cases}$$

接着，我们可以建立如下的拉格朗日函数：

$$L(w_1, w_2, w_3, w_4, w_5, \lambda_1, \lambda_2) = Var\left(w\right) - \lambda_1 \left(\sum_{i=1}^{5} w_i - 1 \right) - \lambda_2 \left(\sum_{i=1}^{5} w_i \bar{R}_i - 0.05 \right)$$

因为 $Var\left(w\right) = \sum_{i=1}^{5} w_i^2 Var\left(R_i\right) + \sum_{i=1}^{5} \sum_{j=1, j\neq i}^{j=5} w_i w_j Cov\left(R_i, R_j\right)$，所以我们只需求解下列方程组即可得到方差最小化的组合权重。[1]

$$\begin{cases} 2Var\left(R_i\right) w_i + 2\sum_{j=1, j\neq i}^{j=5} w_j Cov\left(R_i, R_j\right) - \lambda_1 - \lambda_2 \bar{R}_i = 0 \quad i = 1, 2, 3, 4, 5 \\ \sum_{i=1}^{5} w_i = 1 \\ \sum_{i=1}^{5} w_i \bar{R}_i = 0.05 \end{cases}$$

[1] 严格地说，该方程组的解仅是可能的极值点。但在该例中，方程组的解不仅是极小值，而且是最小值。

第二十章　资本资产定价模型

1.

(a) $R = R_f + (R_m - R_f) = 10\%$

(b) $R = R_f + 1.5\,(R_m - R_f) = 13\%$

(c) $R = R_f + 0.7\,(R_m - R_f) = 8.2\%$

2. A、B、C 都被错误地定价了。

3.

(a) 无影响

(b) 该事件提高了公司的经营风险，从而提高公司的 β 值。

(c) 该事件降低了公司的财务杠杆，从而降低了公司的 β 值。

4.

(a) 该公司的系统风险与市场组合风险变动的幅度和方向一致

(b) 该公司的系统风险与市场组合的风险无关

5. 风险厌恶的投资者应该选择 β 值为 0.7 的股票，因为 β 值越大表明公司的系统风险相对于市场组合风险就越大。

6. $\beta = \frac{Cov(R_A, R_m)}{Var(R_m)} = \frac{0.96}{0.66} = 1.45$

7. A 公司的 α 值为：$9\% - (4\% + 1.2 \times 6\%) = -2.2\%$

B 公司的 α 值为：$12\% - (4\% + 1.3 \times 6\%) = 0.2\%$

所以，投资者应该选取 B 公司的股票。

8.

```
In [1]: import pandas as pd
   ...:
   ...: import pandas_datareader.data as web
   ...:
   ...: import datetime as dt
   ...:
   ...: import statsmodels.formula.api as smf

In [2]: rf=1.036**(1/360)-1
```

```
In [3]: nyyh = web.DataReader('601288.SS','yahoo',
   ...:                       dt.datetime(2014,1,1),dt.datetime(2014,12,31))

In [4]: returns = (nyyh.Close-nyyh.Close.shift(1))/nyyh.Close.shift(1)

In [5]: indexcd=pd.read_csv('Data/Part3/003/TRD_Index.csv',index_col='Trddt')

In [6]: mktcd=indexcd[indexcd.Indexcd==902]

In [7]: mktret=pd.Series(mktcd.Retindex.values,index=pd.to_datetime(mktcd.index))

In [8]: mktret.name='mktret'

In [9]: mktret=mktret['2014-01-02':'2014']

In [10]: dat = pd.concat([mktret,returns],1)

In [11]: dat = dat - rf

In [12]: model = smf.ols('Close~mktret',data=dat).fit()

In [13]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Close   R-squared:                       0.118
Model:                            OLS   Adj. R-squared:                  0.114
Method:                 Least Squares   F-statistic:                     32.41
Date:                Tue, 17 Jan 2017   Prob (F-statistic):           3.59e-08
Time:                        20:47:27   Log-Likelihood:                 661.60
No. Observations:                 245   AIC:                            -1319.
Df Residuals:                     243   BIC:                            -1312.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept      0.0009      0.001      0.849      0.397        -0.001      0.003
mktret         0.5339      0.094      5.693      0.000         0.349      0.719
==============================================================================
Omnibus:                      130.839   Durbin-Watson:                   1.954
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2133.880
Skew:                           1.687   Prob(JB):                         0.00
Kurtosis:                      17.059   Cond. No.                         89.9
==============================================================================
```

所以，模型为

$$R_i - R_f = 0.0009 + 0.5339 \times (R_m - R_f) + \varepsilon$$

其中，$R_f = 1.036^{1/360} - 1$。

1.

(a)

```
In [14]: from statsmodels.api import add_constant

In [15]: rf=1.036**(1/360)-1

In [16]: lsw = web.DataReader('300104.SZ','yahoo',
    ...:                      dt.datetime(2014,1,1),dt.datetime(2014,12,31))

In [17]: returns = (lsw.Close-lsw.Close.shift(1))/lsw.Close.shift(1)

In [18]: indexcd=pd.read_csv('Data/Part3/003/TRD_Index.csv',index_col='Trddt'
    )

In [19]: mktcd=indexcd[indexcd.Indexcd==902]

In [20]: mktret=pd.Series(mktcd.Retindex.values,index=pd.to_datetime(mktcd.
    index))

In [21]: mktret.name='mktret'

In [22]: mktret=mktret['2014-01-02':'2014']

In [23]: dat = pd.concat([mktret,returns],1)

In [24]: dat = dat - rf

In [25]: model = smf.ols('Close~mktret',data=dat).fit()

In [26]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Close   R-squared:                       0.079
Model:                            OLS   Adj. R-squared:                  0.075
Method:                 Least Squares   F-statistic:                     20.77
Date:                Tue, 17 Jan 2017   Prob (F-statistic):           8.21e-06
Time:                        20:49:55   Log-Likelihood:                504.68
No. Observations:                 245   AIC:                            -1005.
Df Residuals:                     243   BIC:                            -998.4
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept     -0.0017      0.002     -0.870      0.385        -0.006      0.002
mktret         0.8108      0.178      4.557      0.000         0.460      1.161
==============================================================================
Omnibus:                       27.033   Durbin-Watson:                   1.956
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               57.347
Skew:                           0.545   Prob(JB):                     3.53e-13
Kurtosis:                       5.105   Cond. No.                         89.9
==============================================================================
```

(b)

```
In [27]: ret_2015=pd.Series(mktcd.Retindex.values,index=pd.to_datetime(mktcd.
    index))['2015-01']

In [28]: ret_2015.name='mktret'

In [29]: ret_2015 = (ret_2015 - rf)

In [30]: prediction = model.predict(add_constant(ret_2015),transform=False)
```

读者可自行查询乐视的股价。总的来说，预测值与实际值的差距还是比较大的。这说明市场回报率是无法完全解释个股的回报率，我们还需要其他因素。下一章的 Fama-French 三因子模型即是一个对资本资产定价模型的拓展。

2. 由于股票众多，在此以银行产业的股票作为范例。从 Stock.accdb 文件中选择银行产业的股票分别建立 CAPM 模型，并筛选出 2013 年 alpha 值最大的前 3 只股票。

```
In [1]: %paste
import pypyodbc
import pandas as pd
import datetime as dt
import numpy as np
import statsmodels.formula.api as smf
from statsmodels.api import add_constant

## -- End pasted text --
```

(a) 获取银行产业股票代码。

```
In [2]: %paste
```

```
for line in open('Data/Part3/003/industryCodes.txt'):
        listData=line.split(',')
        fileName=listData[0]
        listData[-1]=listData[-1][:6]
        if fileName=='银行':
                break

## -- End pasted text --
```

(b) 以中证流通收益率数据作为市场收益率数据。

```
In [3]: %paste
indexcd=pd.read_table('Data/Part3/003/TRD_Index.txt',sep='\t',index_col='
    Trddt')
mktcd=indexcd[indexcd.Indexcd==902]
mktret=pd.Series(mktcd.Retindex.values,index=pd.to_datetime(mktcd.index))
mktret.name='mktret'

## -- End pasted text --
```

(c) 编写函数，并对求出股票的 alpha 值。

```
In [4]: %paste
def GetStockAlpha(symbol,mktret):
        if symbol[0]=='0':
                i=0
                while (symbol[i]=='0'):
                        i+=1
                symbol=symbol[i:]
 #先提前安装Microsoft Access Database Engine
 #可以去Microsoft下载中心下载Microsoft Access Database Engine
      conn = pypyodbc.connect('Driver=Microsoft Access Driver (*.mdb, *.
          accdb);DBQ=%s;' %accessName)
        cursor = conn.cursor()
        cursor.execute('select Stkcd, Trddt, Clsprc from stock where Stkcd=' +
            symbol+'')
        data=cursor.fetchall()
        prices=[]
        dates=[]
        for entry in data:
                prices.append(entry[-1])
                time=str(entry[1])
                date=pd.to_datetime(time[1:])
```

```
                dates.append(date)
        price=pd.Series(np.array(prices),index=dates)['2013']
        returns = (price-price.shift(1))/price.shift(1)
        returns.name='stock'

        rf=1.036**(1/360)-1
        dat = pd.concat([mktret['2013-01-06':'2013']],returns['2013-01-06':
            '2013']],1)
        dat = dat - rf
        model = smf.ols('stock~mktret',data=dat).fit()
        return(model.params[0])

## -- End pasted text --

In [4]: %paste
alphas={}
for symbol in listData[1:]:
        try:
                alphas[symbol]=GetStockAlpha(symbol,mktret)
        except Exception as e:
                print (Exception,":",e)

## -- End pasted text --
```

(d) 筛选 alpha 值前 3 大的股票代码及其 alpha 值。

```
In [5]: %paste
selectAlpha=[('-00',-100000),('-00',-100000),('-00',-100000)]
for symbol in alphas.keys():
        if alphas[symbol]>selectAlpha[0][1]:
                selectAlpha[0]=(symbol,alphas[symbol])
                selectAlpha.sort(key=lambda d:d[1])
selectAlpha

## -- End pasted text --
Out[5]:
[('601998', -0.00056736014634597726),
 ('600000', -0.00032213839451176914),
 ('600016', -0.00020288468807797023)]
```

第二十一章　三因子模型

1. 一个可能的猜想即是同行业的其他公司的股价收益率会影响该股票的收益率。下面我们来验证一下。

```
In [1]: import pandas as pd
   ...:
   ...: import pandas_datareader.data as web
   ...:
   ...: import datetime as dt
   ...:
   ...: import statsmodels.formula.api as smf
   ...:
   ...: from statsmodels.api import add_constant

In [2]: wanke = web.DataReader('000002.SZ','yahoo',
   ...:                        dt.datetime(2015,1,1),dt.datetime(2015,12,31))

In [3]: gldc = web.DataReader('600185.SS','yahoo',
   ...:                       dt.datetime(2015,1,1),dt.datetime(2015,12,31))

In [4]: price = pd.concat([wanke.Close,gldc.Close],1)

In [5]: price.columns=['wanke','geli']

In [6]: ret = (price-price.shift(1)) / price.shift(1)

In [7]: model = smf.ols('wanke~geli',data=ret).fit()

In [8]: print(model.summary())
```

OLS Regression Results

Dep. Variable:	wanke	R-squared:	0.176
Model:	OLS	Adj. R-squared:	0.173
Method:	Least Squares	F-statistic:	55.25
Date:	Tue, 17 Jan 2017	Prob (F-statistic):	1.56e-12
Time:	20:58:05	Log-Likelihood:	568.10
No. Observations:	260	AIC:	-1132.
Df Residuals:	258	BIC:	-1125.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf.	Int.]
Intercept	0.0024	0.002	1.398	0.163	-0.001	0.006
geli	0.2808	0.038	7.433	0.000	0.206	0.355

Omnibus:	43.977	Durbin-Watson:	1.947
Prob(Omnibus):	0.000	Jarque-Bera (JB):	103.520
Skew:	0.799	Prob(JB):	3.32e-23
Kurtosis:	5.646	Cond. No.	22.3

模型的结果显示变量 $geli$ 的系数是显著的，说明格力地产的股价确实和万科的股价有一定的关系。我们的假设是正确的。

1. CAPM 模型和三因子模型的异同点：

   (a) 相同点：都用于通过建立线性模型来探究影响股票收益率的因素。

   (b) 不同点：

      i. CAPM 模型只考虑了一个因子，市场因子; 三因子模型考虑了市场风险溢酬因子、市值因子和账面市值比因子这三种因子。

      ii. CAPM 模型只考虑了市场风险, 三因子模型除了市场风险以外, 还考虑了不同公司本身的运营情况和在市场中的价值等因素, 更加多方面考虑了股票收益率的影响因素。

2. $SMB_t = 1/3\,(SL_t + SM_t + SH_t) - 1/3\,(BL_t + BM_t + BH_t) = -0.03$

   $HML_t = 1/2\,(SL_t + BL_t) - 1/2\,(SH_t + BH_t) = 0.1$

3.

   (a) 0.01

   (b) $R_i = R_f + 0.01 + 1.2 \times (2\% - 0.5\%) + 0.5 \times 2.4\% + 0.1 \times 1.8\% = 4.68\%$

```
In [9]: zyhy = pd.read_table('Data/Part3/004/problem21.txt',
   ....:     sep='\t',usecols=['zyhy','Date'],index_col='Date')

In [10]: zyhy.index=pd.to_datetime(zyhy.index)

In [11]: ret = (zyhy-zyhy.shift(1))/zyhy.shift(1)

In [12]: ThreeFactors = pd.read_table('Data/Part3/004/ThreeFactors.txt',
   ....:     sep='\t',index_col='TradingDate')
```

```
In [13]: ThreeFactors.index=pd.to_datetime(ThreeFactors.index)

In [14]: ThrFac=ThreeFactors['2014']

In [15]: ThrFac=ThrFac.iloc[:,[2,4,6]]

In [16]: dat=pd.concat([ret,ThrFac],1)

In [17]: model = smf.ols('zyhy~RiskPremium2+SMB2+HML2',data=dat).fit()

In [18]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   zyhy   R-squared:                       0.223
Model:                            OLS   Adj. R-squared:                  0.213
Method:                 Least Squares   F-statistic:                     23.01
Date:                Mon, 23 Jan 2017   Prob (F-statistic):           3.94e-13
Time:                        15:56:16   Log-Likelihood:                598.75
No. Observations:                 245   AIC:                            -1190.
Df Residuals:                     241   BIC:                            -1175.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept      0.0021      0.001      1.507      0.133        -0.001     0.005
RiskPremium2   0.9948      0.126      7.893      0.000         0.746     1.243
SMB2          -0.7332      0.223     -3.288      0.001        -1.172    -0.294
HML2           0.7493      0.211      3.559      0.000         0.335     1.164
==============================================================================
Omnibus:                      105.816   Durbin-Watson:                   1.766
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              440.407
Skew:                           1.763   Prob(JB):                     2.33e-96
Kurtosis:                       8.542   Cond. No.                         220.
==============================================================================
```

5.

```
In [19]: zhongxin = pd.read_table('problem21.txt',sep='\t',usecols=['zhongxin','
    Date'],index_col='Date')

In [20]: zhongxin.index=pd.to_datetime(zhongxin.index)

In [21]: ret = (zhongxin-zhongxin.shift(1))/zhongxin.shift(1)

In [22]: ThreeFactors = pd.read_table('ThreeFactors.txt',sep='\t',index_col='
    TradingDate')

In [23]: ThreeFactors.index=pd.to_datetime(ThreeFactors.index)

In [24]: ThrFac=ThreeFactors['2014']
```

```
In [25]: ThrFac=ThrFac.iloc[:,[2,4,6]]

In [26]: dat=pd.concat([ret,ThrFac],1)
```

(a)

```
In [27]: model = smf.ols('zhongxin~RiskPremium2',data=dat).fit()

In [28]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               zhongxin   R-squared:                       0.207
Model:                            OLS   Adj. R-squared:                  0.204
Method:                 Least Squares   F-statistic:                     63.51
Date:                Mon, 23 Jan 2017   Prob (F-statistic):           6.19e-14
Time:                        15:59:44   Log-Likelihood:                 557.71
No. Observations:                 245   AIC:                            -1111.
Df Residuals:                     243   BIC:                            -1104.
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [95.0% Conf. Int.]
--------------------------------------------------------------------------------
Intercept        0.0015      0.002      0.950      0.343      -0.002       0.005
RiskPremium2     1.1710      0.147      7.970      0.000       0.882       1.460
==============================================================================
Omnibus:                       89.839   Durbin-Watson:                   1.716
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              370.027
Skew:                           1.460   Prob(JB):                     4.46e-81
Kurtosis:                       8.265   Cond. No.                         92.2
==============================================================================
```

(b)

```
In [29]: model2 = smf.ols('zhongxin~RiskPremium2+SMB2+HML2',data=dat).fit()

In [30]: print(model2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               zhongxin   R-squared:                       0.377
Model:                            OLS   Adj. R-squared:                  0.369
Method:                 Least Squares   F-statistic:                     48.59
Date:                Tue, 17 Jan 2017   Prob (F-statistic):           1.35e-24
Time:                        21:05:23   Log-Likelihood:                 587.21
No. Observations:                 245   AIC:                            -1166.
Df Residuals:                     241   BIC:                            -1152.
Df Model:                           3
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [95.0% Conf. Int.]
--------------------------------------------------------------------------------
Intercept        0.0011      0.001      0.786      0.433      -0.002       0.004
RiskPremium2     1.1763      0.132      8.903      0.000       0.916       1.437
SMB2            -0.0076      0.234     -0.032      0.974      -0.468       0.453
HML2             0.8663      0.221      3.926      0.000       0.432       1.301
==============================================================================
Omnibus:                       84.500   Durbin-Watson:                   1.780
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              388.225
Skew:                           1.318   Prob(JB):                     4.99e-85
Kurtosis:                       8.575   Cond. No.                         220.
==============================================================================
```

(c)

```
In [31]: ThrFac=ThreeFactors['2015-01']

```

```
In [32]: preCAPM = model.predict(add_constant(ThrFac.RiskPremium2),transform=
    False)

In [33]: preFactors = model2.predict(add_constant(ThrFac[['RiskPremium2','
    SMB2','HML2']]),
   ...:                              transform = False))
```

```
In [1]: %paste
import pandas as pd

codes = pd.read_csv('Data/Part3/004/codes.csv', header=None,dtype=str)

ThreeFactors = pd.read_table('Data/Part3/004/ThreeFactors.txt', sep='\t',
    index_col='TradingDate')

ThreeFactors.index=pd.to_datetime(ThreeFactors.index)

ThrFac=ThreeFactors['2014']

ThrFac=ThrFac.iloc[:,[2,4,6]]

def create_func(model):
    def cal_alpha(code,model_name=model):
        price = web.DataReader(code,'yahoo',
                               dt.datetime(2014,1,1),dt.datetime(2014,12,31)).Close

        ret = (price-price.shift(1))/price.shift(1)

        ret.name = 'ret'

        dat=pd.concat([ret,ThrFac],1)

        if model_name=='CAPM':
            model = smf.ols('ret~RiskPremium2',data=dat).fit()
        elif model_name=='factors':
            model = smf.ols('ret~RiskPremium2+SMB2+HML2',data=dat).fit()

        return(model.params[0])
    return(cal_alpha)

```

```
## -- End pasted text --
```

6. (a)

```
In [14]: alpha_CAPM=list(map(create_func('CAPM'),codes[0].values))

In [15]: alpha_CAPM2=pd.Series(alpha_CAPM).sort(ascending=False,inplace=False
    )

In [16]: alpha_CAPM2[:3]
Out[16]:
12    0.001531
8     0.000900
9     0.000768
dtype: float64
```

```
In [17]: alpha_factors=list(map(create_func('factors'),codes[0]))

In [18]: alpha_factors2 = pd.Series(alpha_factors).sort(ascending=False,
    inplace=False)

In [19]: alpha_factors2[:3]
Out[19]:
12    0.001128
8     0.000532
13    0.000476
dtype: float64
```

(b) 两者选股的结果是较为相似的，仅有一个不一样。

第四部分

时间序列

第二十二章　简介

1. 略
2.

```
In [1]: import pandas as pd
   ...: import matplotlib.pyplot as plt
   ...: Yen=pd.read_csv('Data/Part4/001/Yen.csv',index_col='date')

In [2]: Yen.index=pd.to_datetime(Yen.index,format='%Y%m%d')
```

3. 对于 xts 对象，plot() 一次只能绘制一个变量。因此，我们选择变量 s 为例。可以看到，图 22.1中存在着明显的下降的趋势。

```
In [3]: Yen.s.plot()
```

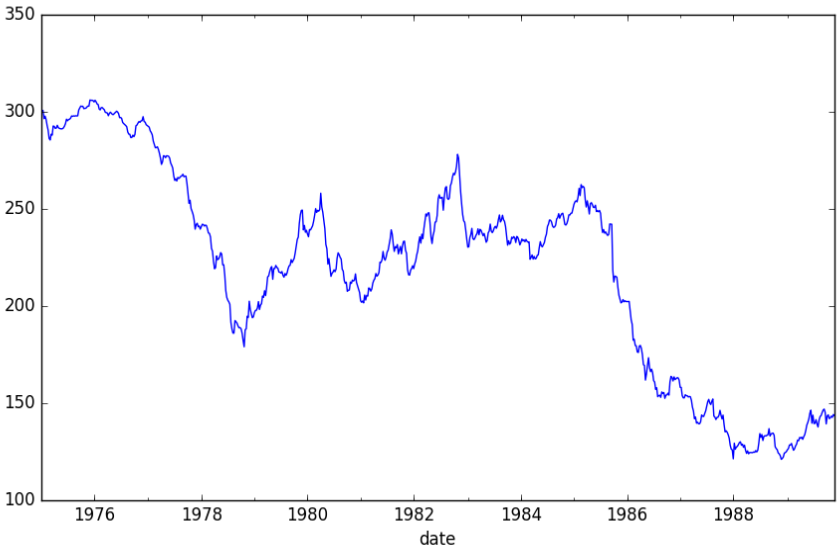


图 22.1: 题 3

4.

```
In [4]: pfyh=pd.read_csv('Data/Part4/001/pfyh.csv',index_col='Date')

In [5]: pfyh.index=pd.to_datetime(pfyh.index,format='%Y-%m-%d')

In [6]: returns=(pfyh.Close-pfyh.Close.shift(1))/pfyh.Close.shift(1)
```

```
6
7 In [7]: returns=returns[1:]
8
9 In [8]: returns.plot()
10 Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8c47c6b358>
```

浦发银行的收益率围绕着 0 上下波动。

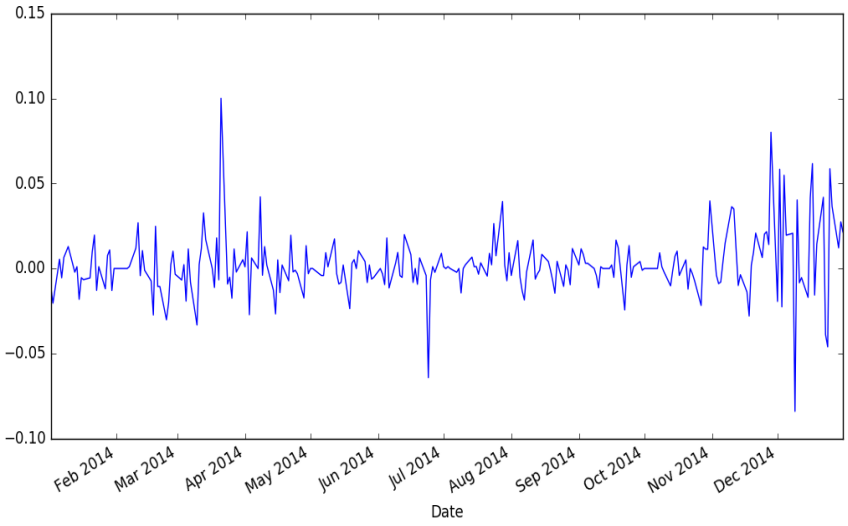


图 22.2: 题 4

第二十三章　基本性质

1.

(a) 是。当 $k=1$ 时，$r_k = corr\left(X_t, X_{t-k}\right) = corr\left(e_t - e_{t-1}, e_{t-1} - e_{t-2}\right) = -\frac{1}{2}$；

当 $k>1$ 时，$r_k = corr\left(X_t, X_{t-k}\right) = corr\left(e_t - e_{t-1}, e_{t-k} - e_{t-k-1}\right) = 0$

(b) 是，其自相关系数皆为 0。

$r_k = corr\left(X_t, X_{t-k}\right) = corr\left(3t, 3\left(t-k\right)\right) = 0$

(c) 是，其自相关系数皆为 0。

$r_k = corr\left(X_t, X_{t-k}\right) = corr\left(\left(-1\right)^t e_t, \left(-1\right)^{t-k} e_{t-k}\right) = \left(-1\right)^{2t-k} corr\left(e_t, e_{t-k}\right) = 0$

2. 因为 $X_t = Y$，所以在任意时点，X_t 的值都与 Y 的值一样。因此，对于任意正整数 n，任取 $t_1, t_2, \cdots, t_n \in T$，对任意整数 τ 有：

$$F_X\left(x_{t_1}, x_{t_2}, \cdots, x_{t_n}\right) = F_Y\left(x_{t_1}\right) = F_X\left(x_{t_1+\tau}, x_{t_2+\tau}, \cdots, x_{t_n+\tau}\right)$$

所以 X_t 是严平稳的。又因为 Y 存在均值和方差，所以 X_t 又一定是弱平稳的。

3.

$$r_k = \begin{cases} \frac{Cov(X_t, X_{t-1})}{\sqrt{Var(X_t)Var(X_{t-1})}} = \frac{Cov(e_t - 0.4e_{t-1} + 0.3e_{t-2}, e_{t-1} - 0.4e_{t-2} + 0.3e_{t-3})}{1.25\sigma^2} = -0.416, & k=1 \\ \frac{Cov(X_t, X_{t-2})}{\sqrt{Var(X_t)Var(X_{t-2})}} = \frac{Cov(e_t - 0.4e_{t-1} + 0.3e_{t-2}, e_{t-2} - 0.4e_{t-3} + 0.3e_{t-4})}{1.25\sigma^2} = 0.24, & k=2 \\ \frac{Cov(X_t, X_{t-k})}{\sqrt{Var(X_t)Var(X_{t-k})}} = \frac{Cov(e_t - 0.4e_{t-1} + 0.3e_{t-2}, e_{t-k} - 0.4e_{t-k-1} + 0.3e_{t-k-2})}{1.25\sigma^2} = 0, & k>2 \end{cases}$$

显然，X_t 的自相关系数与 t 无关，所以 X_t 是弱平稳的。

4. $\{X_t\}$ 的滞后算子多项式方程为

$$1 - x = 0$$

该方程唯一的解为 2，大于 1。因此，$\{X_t\}$ 是一个平稳序列。其自相关系数为：

$$r_k = 0.5^k$$

5.

```
In [1]: import pandas as pd

In [2]: import matplotlib.pyplot as plt
```

```
In [3]: CRSPday=pd.read_csv('Data/Part4/002/CRSPday.csv')

In [4]: ibm=CRSPday.ibm
```

(a)

```
In [5]: ibm.plot()
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7f09015cb240>
```

```
In [6]: from statsmodels.graphics.tsaplots import *
In [7]: plot_acf(ibm,lags=20)
Out[7]: <matplotlib.figure.Figure at 0x7fe634701cf8>
```

从 ACF 图中，我们可以看到，所有的自相关系数都不显著，所以 IBM 的日回报率很有可能是白噪声。

(b)

```
In [8]: from statsmodels.tsa import stattools
In [9]: LjungBox=stattools.q_stat(stattools.acf(ibm)[1:13],len(ibm))
In [10]: LjungBox[1][-1]
Out[10]: 0.47767342041175087
```

Ljung-Box 检验的 p 值为 0.4777，大于 0.05，同样表明 IBM 的日回报率是白噪声。

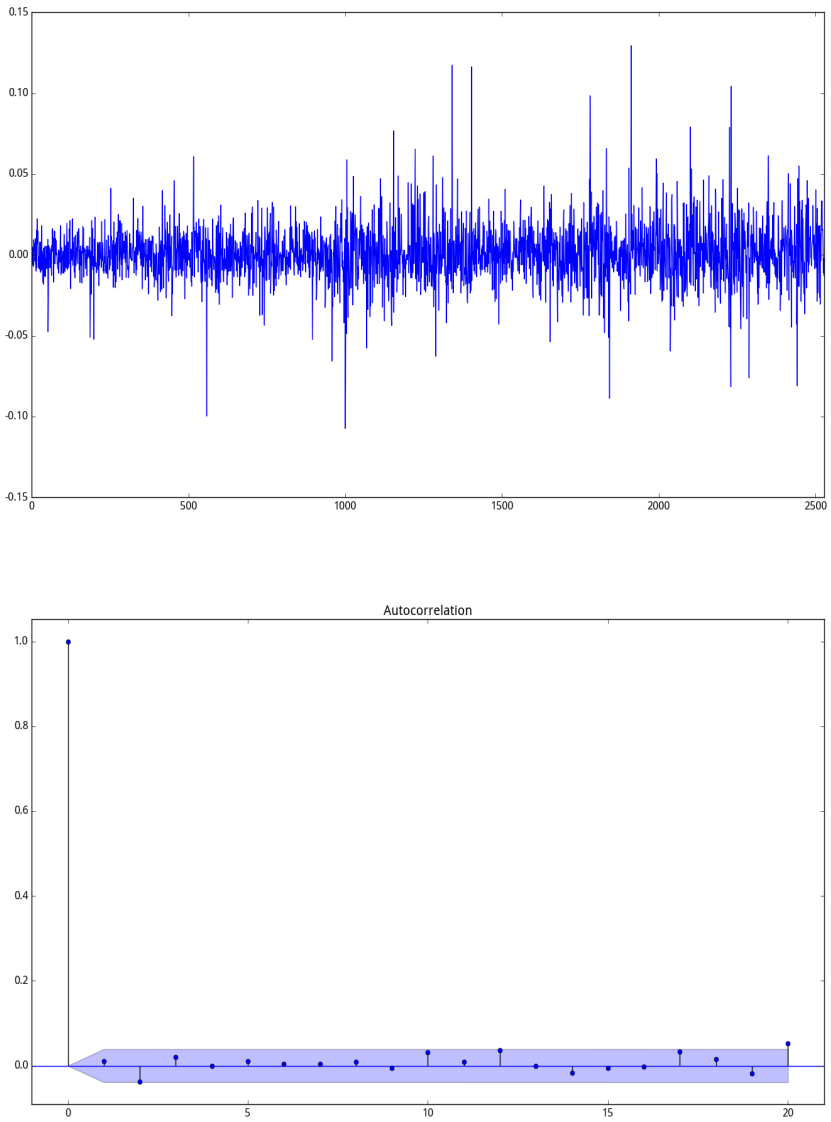


图 23.1: 题 5

6.

```
In [11]: ge=CRSPday.iloc[:,3]
```

(a)

```
In [12]: ge.plot()
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f08e54a9da0>
```

```
In [13]: plot_acf(ge,lags=20)
```

(b)

```
In [14]: LjungBox=stattools.q_stat(stattools.acf(ge)[1:2],len(ge))
In [15]: LjungBox[1][-1]
Out[15]: 0.63415629590274136
```

```
In [16]: LjungBox=stattools.q_stat(stattools.acf(ge)[1:9],len(ge))
```

```
In [17]: LjungBox[1][-1]
Out[17]: 0.038343684289074531
```

(d) 当我们使用 2 阶的自相关系数的时候，Ljung-Box 检验的 p 值为 0.6342。而当我们使用 9 阶的自相关系数的时候，Ljung-Box 检验的 p 值为 0.0383。两者的结论完全相反，这表明选取的滞后项过少会导致检验的 p 值偏高。因此，在实际使用 Ljung-Box 检验的时候，我们一般会将 $6-15$ 阶的自相关系数。

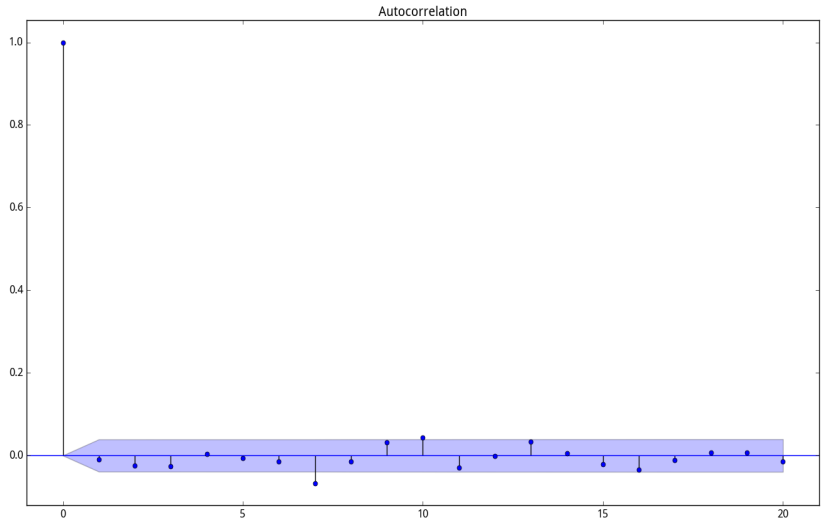


图 23.2: 题 6

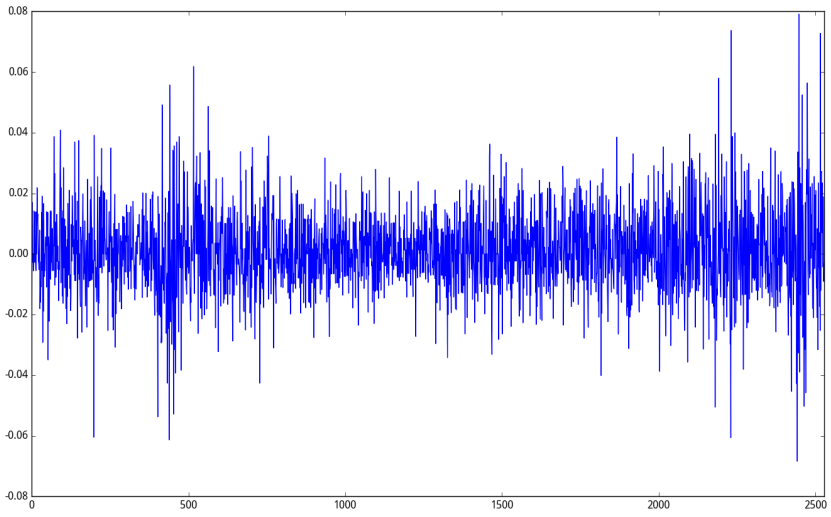


图 23.3: 题 6

7.

```
In [18]: SP500=pd.read_csv( Data/Part4/002/SP500.csv )

In [19]: r500=SP500.r500
```

(a)

```
In [20]: r500.plot()
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1593321128>
```

标普 500 指数的日回报率大致围绕着 0 上下波动，虽然中间有几个点波动过大，但总体来说还是较为平稳的。

(b)

```
In [21]: plot_acf(r500,lags=20)
Out[21]: <matplotlib.figure.Figure at 0x7fe62ee44780>

In [22]: plot_pacf(r500,lags=20)
Out[22]: <matplotlib.figure.Figure at 0x7fe62ee54cc0>
```

```
In [23]: from arch.unitroot import ADF

In [24]: adf=ADF(r500,lags=3)

In [25]: print(adf.summary().as_text())
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                -28.096
P-value                         0.000
Lags                                3
-------------------------------------

Trend: Constant
Critical Values: -3.43 (1%), -2.86 (5%), -2.57 (10%)
Null Hypothesis: The process contains a unit root.
Alternative Hypothesis: The process is weakly stationary.
```

ADF 检验的统计量的值为 -28.096，远小于 5% 的临界值 -2.86。因此，标普 500 指数的日回报率是平稳序列。我们在 (a) 中的判断是正确的。

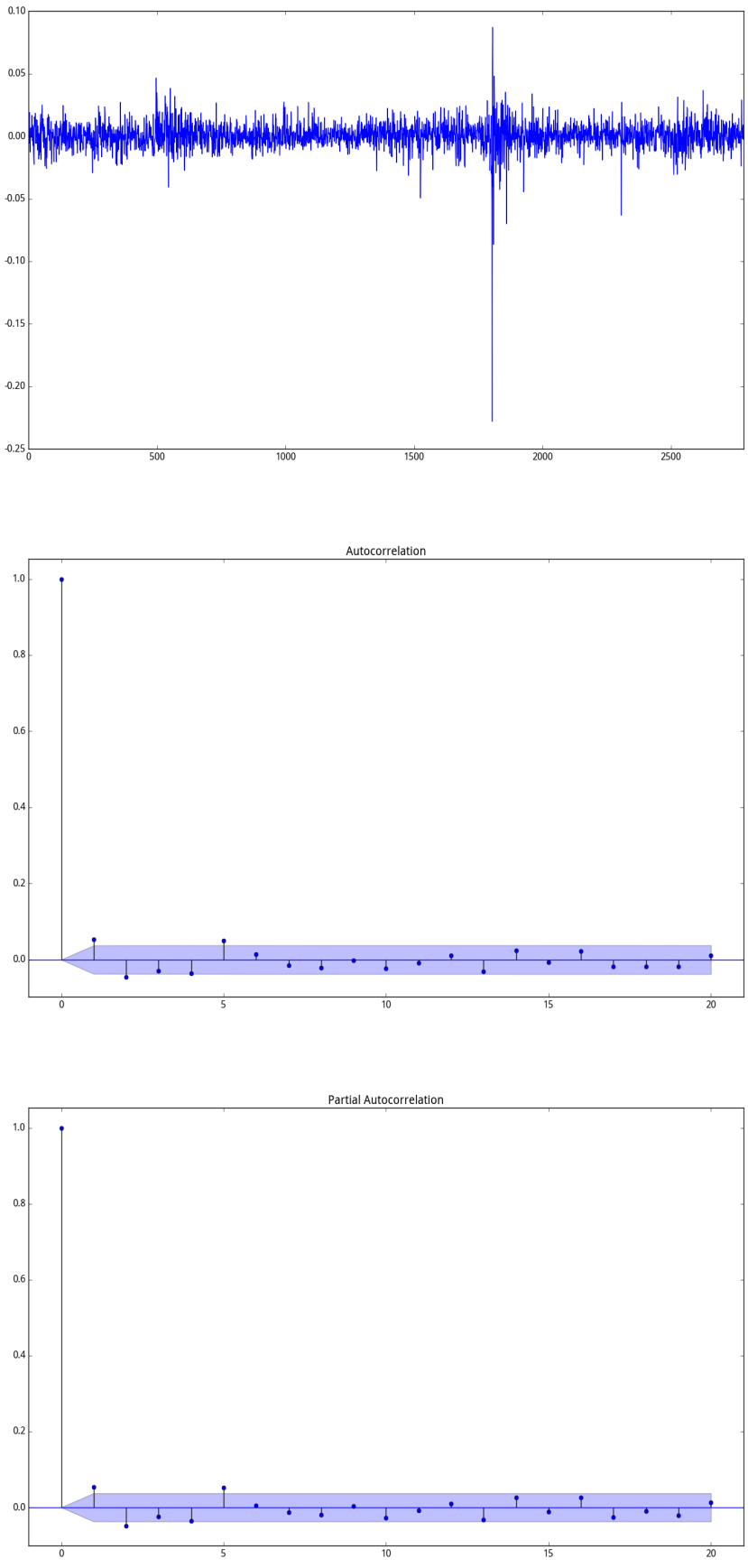

0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
0
500
1000
1500
2000
2500
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20

图 23.4: 题 7

第二十四章　预测

(d)

(a) MA 模型，$q = 2$

(b) MA 模型，$q = 4$

(c) ARMA 模型，$p = 2$，$q = 3$

(d) AR 模型，$p = 2$

2. 假设 X_t 一个 MA(q) 的时间序列，那么 $X_t = \sum_{i=0}^{q} \theta_i e_{t-i}$。当 $k > q$ 时：

$$\rho_k = \frac{Cov\left(X_t, X_{t-k}\right)}{Var\left(X_t\right)} = \frac{Cov\left(\sum_{i=0}^{q} \theta_i e_{t-i}, \sum_{i=0}^{q} \theta_i e_{t-k-i}\right)}{Var\left(X_t\right)}$$

因为当 $i \neq j$ 时，$Cov\left(e_i, e_j\right) = 0$。所以，$\rho_k = 0$。

3. 第 5 期的预测值为 7.45875。

4.

```
In [1]: import statsmodels.tsa.arima_process as sm

In [2]: from statsmodels.graphics.tsaplots import *

In [3]: arma=sm.ArmaProcess([-1,-0.6],[1])

In [4]: sample=arma.generate_sample(200)

In [5]: plot_acf(sample,lags=20)
Out[5]: <matplotlib.figure.Figure at 0x7ffb9b7bfc88>

In [6]: plot_pacf(sample,lags=20)
Out[6]: <matplotlib.figure.Figure at 0x7ffb78ac8240>
```

ACF 图呈现出指数递减的趋势，而 PACF 图则在二阶滞后处出现截断现象。因此，根据绘制出来的 ACF 图与 PACF 图，我们应该选择 AR(1) 模型。该模型符合真实情况。

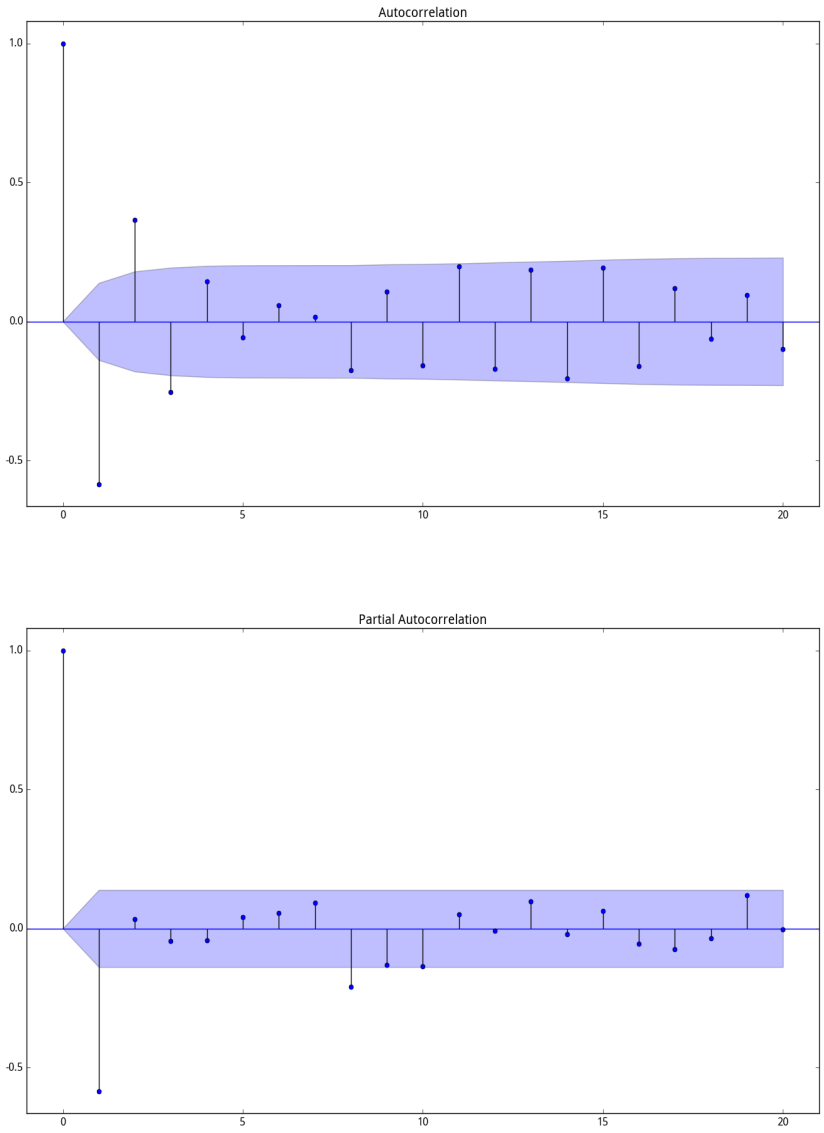


图 24.1: 题 4

5.

```
In [7]: arma=sm.ArmaProcess([-1],[1,0.4])

In [8]: sample=arma.generate_sample(200)

In [9]: plot_acf(sample,lags=20)
Out[9]: <matplotlib.figure.Figure at 0x7ffb78ae1470>

In [10]: plot_pacf(sample,lags=20)
Out[10]: <matplotlib.figure.Figure at 0x7ffb749feac8>
```

在 ACF 图中，一阶自相关系数与二阶自相关系数都是显著的。而在 PACF 图中，一阶偏自相关系数是显著的，之后的偏自相关系数都不显著。因此，根据 ACF 图和 PACF 图，我们可能会选择 MA(2) 或是 AR(1)。这时，选择的模型就不再符合真实情况了。这是因为模拟的期数过少，样本统计量存在较大的偏差。在实务中，我们也要

注意这种偏差，多尝试几个模型，而不是一味地依照 ACF 与 PACF。比如说，在本题中，我们就可以再多尝试 ARMA(1,1) 与 MA(1) 这两个模型。

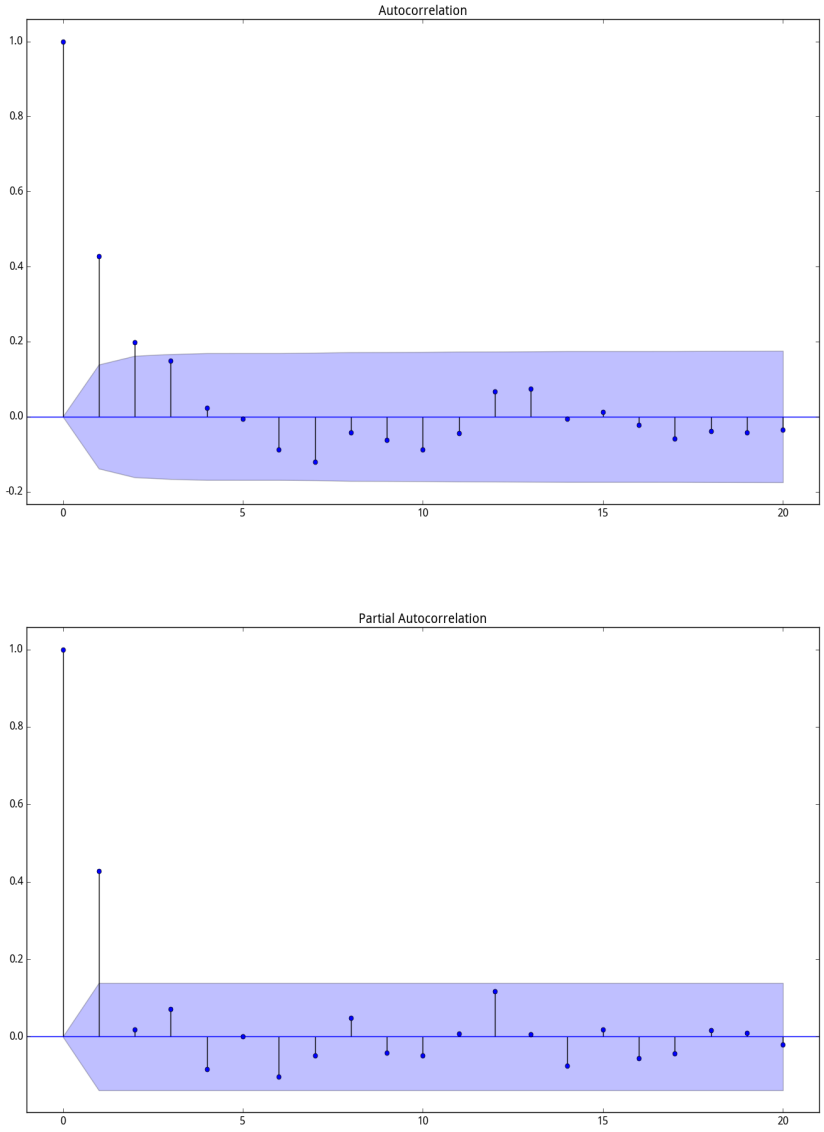


图 24.2: 题 5

6.

```
In [1]: import statsmodels.tsa.arima_process as sm

In [2]: from statsmodels.graphics.tsaplots import *

In [3]: import numpy as np

In [4]: import pandas as pd

In [5]: numbers=np.random.normal(size=100)

```

```
In [6]: numbers=pd.Series(numbers)
```

(a)

```
In [7]: numbers.plot()
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f24c58a2358>

In [8]: plot_acf(numbers,lags=20)
Out[8]: <matplotlib.figure.Figure at 0x7f24c528b828>
```

很明显，$numbers$ 围绕着 0 上下随机波动，而且其 ACF 都不显著。因此，我们可以认为 $numbers$ 是白噪声。

(b)

```
In [9]: from statsmodels.tsa import stattools

In [10]: stattools.arma_order_select_ic(numbers.values,max_ma=4)
Out[10]:
{'bic':             0           1           2           3           4
0  301.560380  305.113771  309.241432  313.831146  318.394066
1  305.268515  309.372041  313.826739  318.431879  322.992102
2  309.240906  313.835172  318.902116  323.034685  324.016683
3  313.840677  318.439734         NaN  327.099384         NaN
4  318.346121  323.058940  324.742326  331.636021  334.205935,
'bic_min_order': (0, 0)}
```

arma_order_select_ic( ) 函数得到的结果为 ARIMA(0,0,0)，也就是说，我们只能使用均值来预测 $numbers$ 的值。可见，对于白噪声，我们无法进行有意义的建模。

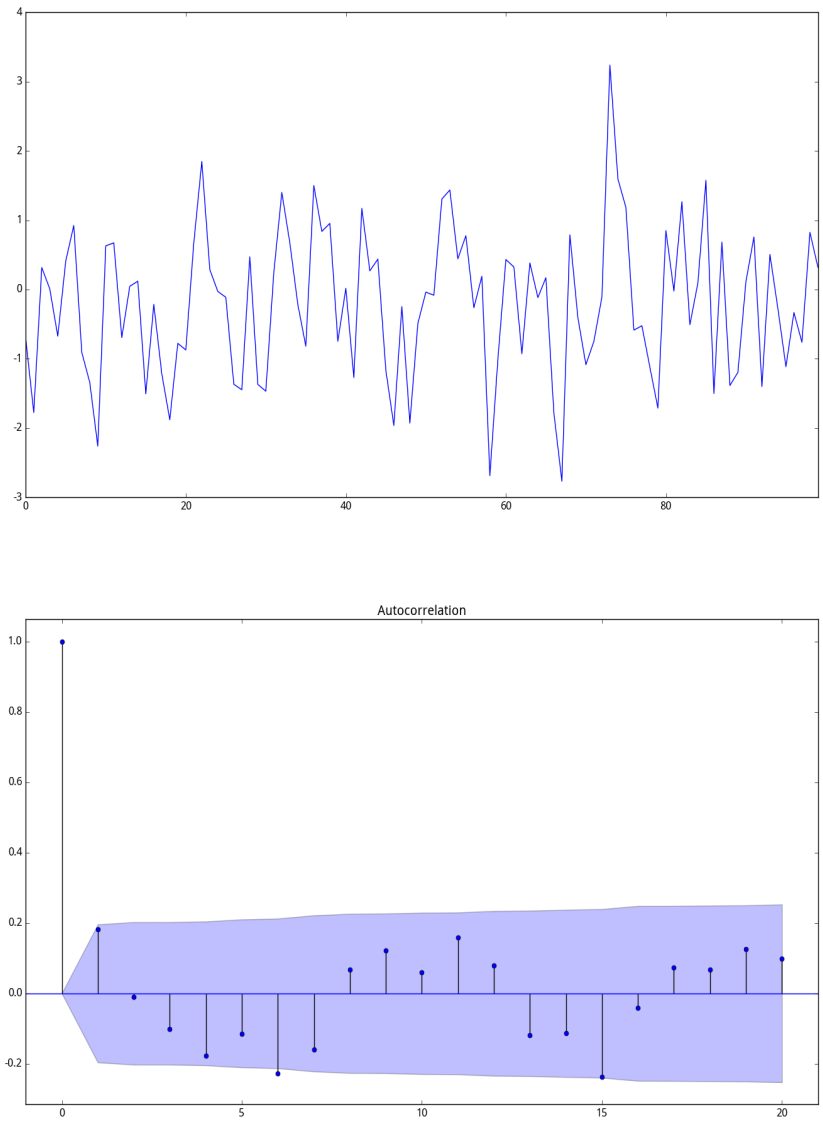


图 24.3: 题 6

7.

```
In [1]: zgsy=pd.read_csv('Data/Part4/003/zgsy.csv')

In [2]: clprice=zgsy.iloc[:,4]
```

(a)

```
In [3]: clprice.plot()
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x7f24e3de8160>

In [4]: plot_acf(clprice,lags=20)
Out[4]: <matplotlib.figure.Figure at 0x7f24e3cb3f98>
```

$clprice$ 的时间序列图呈现出明显的下降趋势，而且其 ACF 也是缓慢递减。因此，$clprice$ 不平稳。

(b)

```
In [5]:from arch.unitroot import ADF
```

```
    ...: adf=ADF(clprice,lags=6)

In [6]: print(adf.summary().as_text())
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                 -1.185
P-value                         0.680
Lags                                6
-------------------------------------

Trend: Constant
Critical Values: -3.46 (1%), -2.87 (5%), -2.57 (10%)
Null Hypothesis: The process contains a unit root.
Alternative Hypothesis: The process is weakly stationary.
```

ADF 检验的统计量为 -1.185，大于 5% 的临界值。因此，我们不能拒绝原假设，$clprice$ 不平稳。

(c)

```
In [7]: logReturn=pd.Series((np.log(clprice))).diff().dropna()

In [8]: logReturn.plot()
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f24e3cd1cf8>
```

可以看到，$logReturn$ 围绕着 0 上下波动，并没有呈现出明显的趋势。

(d)

```
In [9]: adf=ADF(logReturn,lags=6)

In [10]: print(adf.summary().as_text())
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                 -6.402
P-value                         0.000
Lags                                6
-------------------------------------

Trend: Constant
Critical Values: -3.46 (1%), -2.87 (5%), -2.57 (10%)
Null Hypothesis: The process contains a unit root.
Alternative Hypothesis: The process is weakly stationary.
```

ADF 检验的结果表明我们可以拒绝原假设，因此 $logReturn$ 是平稳的。

(e)

```
In [11]: plot_acf(logReturn,lags=20)
Out[11]: <matplotlib.figure.Figure at 0x7f24e3c19748>
```

```
In [12]: plot_pacf(logReturn,lags=20)
Out[12]: <matplotlib.figure.Figure at 0x7f24e3b7c828>
```

在 ACF 图中，自相关系数从二阶开始变得不显著。在 PACF 图中，偏自相关系数同样从二阶开始变得不显著。因此，我们可以有两个选择——MA(1) 和 AR(1)。

(f)

```
In [13]: from statsmodels.tsa import arima_model

In [14]: model1=arima_model.ARIMA(logReturn.values,order=(0,0,1)).fit()
```

```
RUNNING THE L-BFGS-B CODE
           * * *
Machine precision = 2.220D-16
 N =            2     M =           12
 This problem is unconstrained.

At X0         0 variables are exactly at the bounds

At iterate    0    f= -3.28295D+00    |proj g|=  4.68026D-04

At iterate    5    f= -3.28295D+00    |proj g|=  9.19265D-06
           * * *
Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value
           * * *
   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      6     16      1     0     0   3.864D-06  -3.283D+00
  F =  -3.2829527560393754

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH

 Cauchy                time 0.000E+00 seconds.
 Subspace minimization time 0.000E+00 seconds.
 Line search           time 0.000E+00 seconds.

 Total User time 0.000E+00 seconds.
```

```
In [15]: model1.summary()
Out[5]:
<class 'statsmodels.iolib.summary.Summary'>
```

```
"""
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  259
Model:                     ARMA(0, 1)   Log Likelihood                 850.285
Method:                       css-mle   S.D. of innovations              0.009
Date:                Sun, 22 Jan 2017   AIC                          -1694.570
Time:                        00:44:06   BIC                          -1683.899
Sample:                             0   HQIC                         -1690.279

==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const         -0.0006      0.001     -0.953      0.342        -0.002      0.001
ma.L1.y        0.1411      0.056      2.505      0.013         0.031      0.252
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
MA.1           -7.0862           +0.0000j            7.0862            0.5000
-----------------------------------------------------------------------------
"""
```

```
In [16]: model2=arima_model.ARIMA(logReturn.values,order=(1,0,0)).fit()
```

```
RUNNING THE L-BFGS-B CODE
           * * *
Machine precision = 2.220D-16
 N =            2     M =           12
 This problem is unconstrained.
At X0         0 variables are exactly at the bounds

At iterate    0    f= -3.28458D+00    |proj g|=  2.81846D-02

At iterate    5    f= -3.28458D+00    |proj g|=  2.97518D-03

At iterate   10    f= -3.28458D+00    |proj g|=  2.88435D-02

At iterate   15    f= -3.28458D+00    |proj g|=  2.36255D-05
           * * *
Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value
           * * *
   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2     17     20      1     0     0   0.000D+00  -3.285D+00
  F =  -3.2845776032338465
CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL

 Cauchy                time 0.000E+00 seconds.
 Subspace minimization time 0.000E+00 seconds.
 Line search           time 0.000E+00 seconds.

 Total User time 0.000E+00 seconds.
```

```
In [17]: model2.summary()
Out[17]:
<class 'statsmodels.iolib.summary.Summary'>
```

```
"""
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  259
Model:                     ARMA(1, 0)   Log Likelihood                 850.706
Method:                       css-mle   S.D. of innovations              0.009
Date:                Sun, 22 Jan 2017   AIC                          -1695.411
Time:                        00:44:08   BIC                          -1684.741
Sample:                             0   HQIC                         -1691.121

==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const         -0.0006      0.001     -0.913      0.362        -0.002      0.001
ar.L1.y        0.1615      0.061      2.638      0.009         0.042      0.281
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            6.1939           +0.0000j            6.1939            0.0000
-----------------------------------------------------------------------------
"""
```

我们分别构建了 MA(1) 模型和 AR(1) 模型。在这两个模型中，系数都是显著的。因此，我们无法通过系数的显著性来选择模型。这时，我们可以通过 AIC 来选择模型。很明显，$model2$ 的 AIC 较小。因此，我们应该选择 $model2$，也就是 AR(1) 模型。

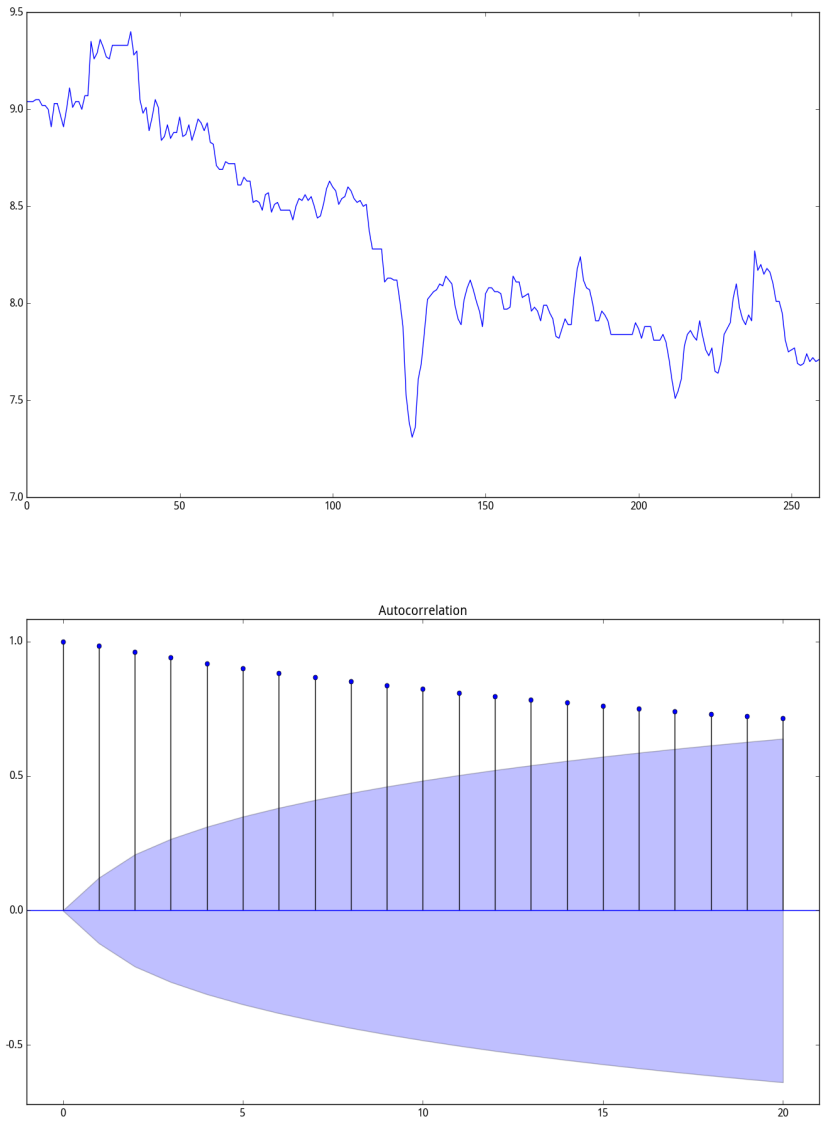

9.5
9.0
8.5
8.0
7.5
7.0
0
50
100
150
200
250
Autocorrelation
1.0
0.5
0.0
-0.5
0
5
10
15
20

图 24.4: 题 7

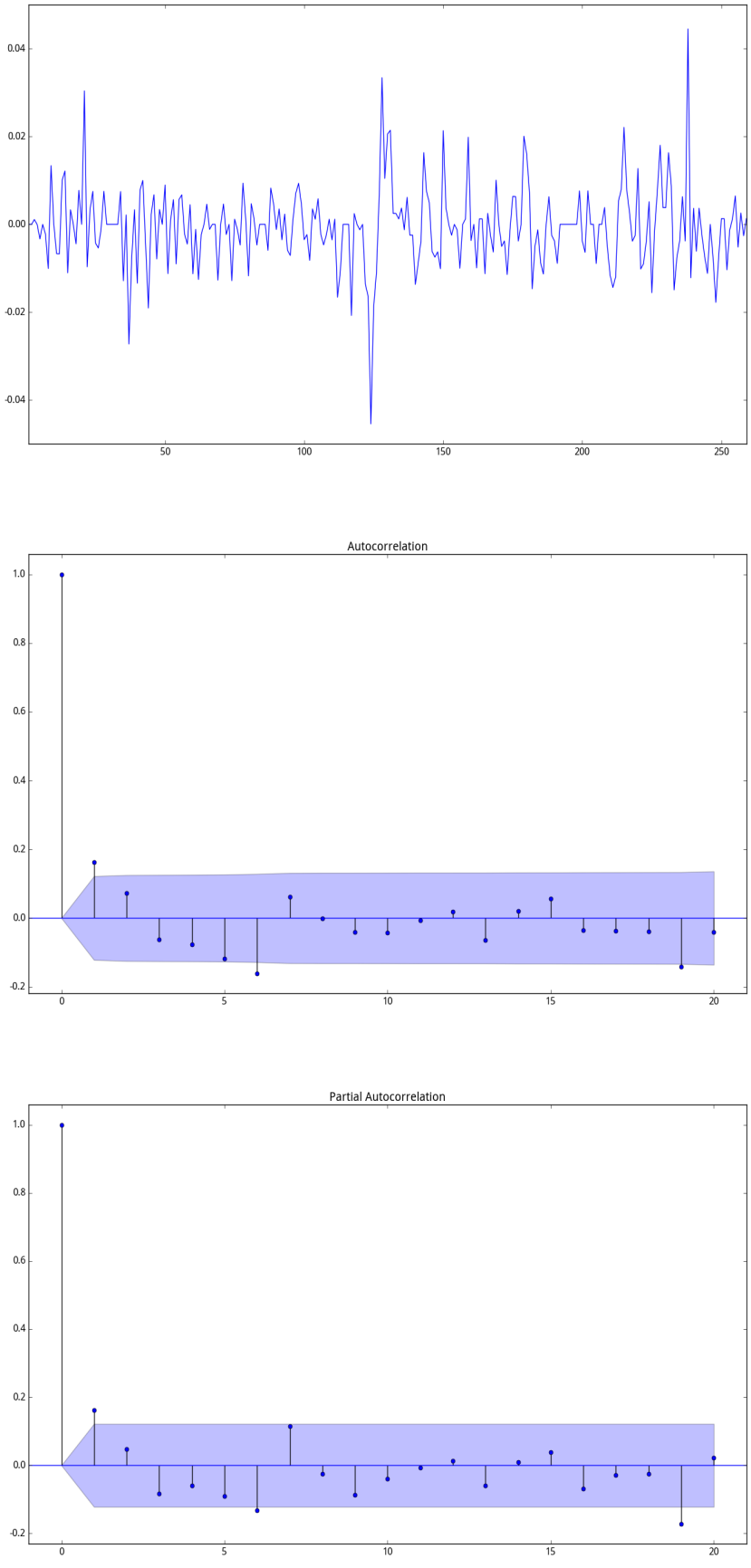

0.04
0.02
0.00
-0.02
-0.04
50
100
150
200
250
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0
5
10
15
20
Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0
5
10
15
20

图 24.5: 题 7

1.

```
In [1]: baiyun=zgsy=pd.read_csv('baiyun.csv',index_col='Date')

In [2]: baiyun.index=pd.to_datetime(baiyun.index)

In [3]: clprice=baiyun.Close
```

(a)

```
In [4]: logReturn=pd.Series((np.log(clprice))).diff().dropna()

In [5]: logReturn.plot()
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7efdb08b41d0>
```

```
In [6]: adf=ADF(logReturn,lags=6)

In [7]: print(adf.summary().as_text())
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                 -6.189
P-value                         0.000
Lags                                6
-------------------------------------

Trend: Constant
Critical Values: -3.46 (1%), -2.87 (5%), -2.57 (10%)
Null Hypothesis: The process contains a unit root.
Alternative Hypothesis: The process is weakly stationary.
```

ADF 检验的统计量小于 5% 的临界值，因此我们能够拒绝原假设，收益率序列是平稳的。

(b)

```
In [8]: plot_acf(logReturn,lags=20)
Out[8]: <matplotlib.figure.Figure at 0x7f24e3ae7c50>

In [9]: plot_pacf(logReturn,lags=20)
Out[9]: <matplotlib.figure.Figure at 0x7f24e39f5ef0>

In [10]: model1=arima_model.ARIMA(logReturn.values,order=(0,0,2)).fit()

In [11]: model2=arima_model.ARIMA(logReturn.values,order=(2,0,0)).fit()

In [12]: model1.aic
Out[12]: -1563.422363156626

In [13]: model2.aic
```

```
15 Out[13]: −1561.4132520773128
```

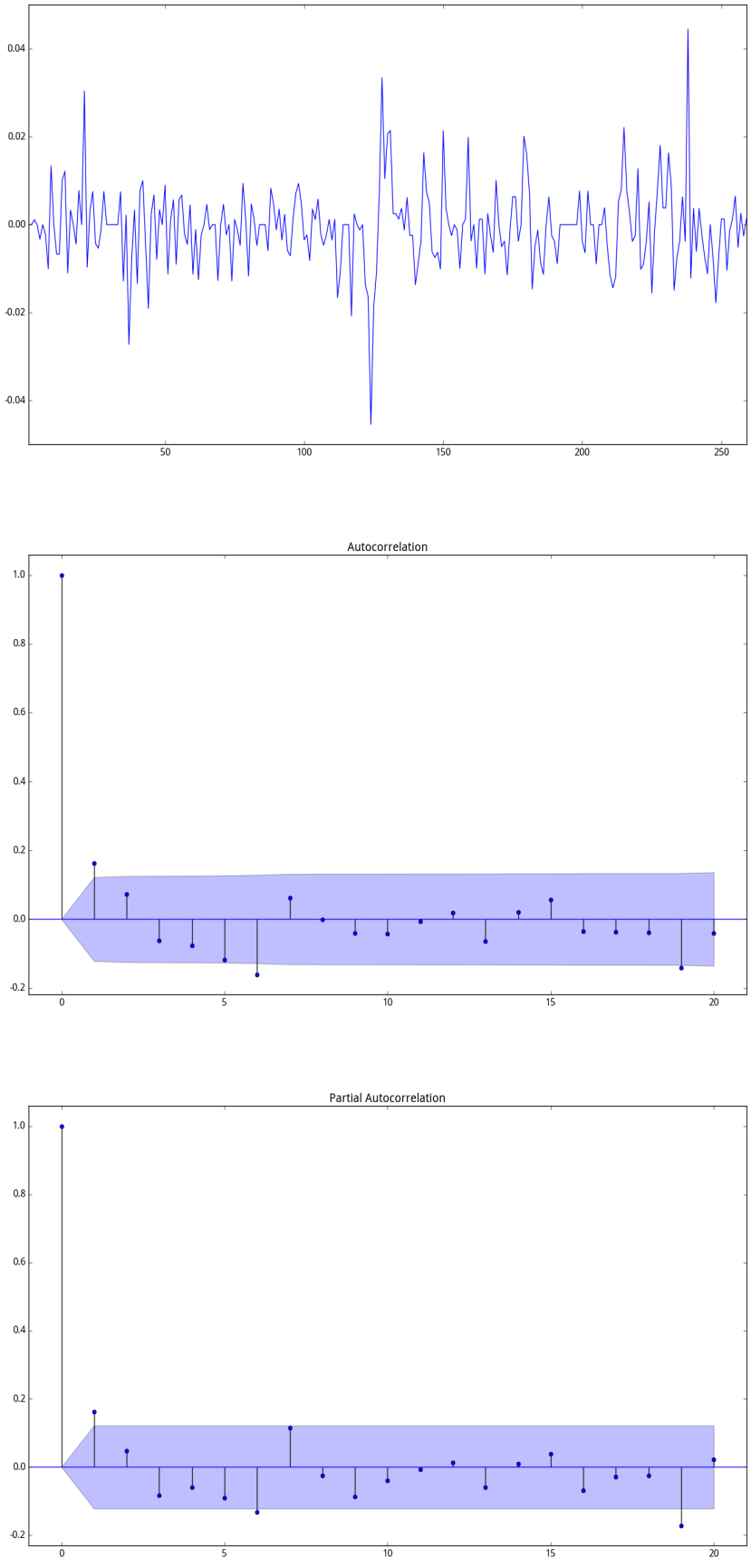

0.04
0.02
0.00
-0.02
-0.04
50
100
150
200
250
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0
5
10
15
20
Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0
5
10
15
20

图 24.6: 题 8

在 ACF 图中，二阶的自相关系数是显著的。而在 PACF 图中，二阶的偏自相关系数也是显著的。因此，我们可以选择 MA(2) 与 AR(2) 模型。同其中，$model1$ 的 AIC 较小。因此，我们应该选取 MA(2) 模型。

(d)

```
In [14]: import math

In [15]: stdresid=model2.resid/math.sqrt(model2.sigma2)

In [16]: stdresid.plot()
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7efdb077edd8>

In [17]: plot_acf(stdresid,lags=20)
Out[17]: <matplotlib.figure.Figure at 0x7f24e3b64dd8>

In [18]: LjungBox=stattools.q_stat(stattools.acf(stdresid)[1:13],len(stdresid
    ))

In [19]: LjungBox[1][-1]
Out[19]: 0.21712592584238385
```

第一张图是模型的标准残差图，大部分的残差都在 ± 2 之内，这表明我们的模型对数据拟合得很好。第二张图是残差的 ACF 图，所有的自相关系数都是不显著的。因此，我们可以认为残差是一个白噪声。之后我们进行了 Ljung-Box 检验，p 值为 0.2171，表明残差序列为白噪声。因此，我们可以认为我们的模型是一个不错的模型。

(e)

```
In [20]: pd.Series(model2.forecast(10)[0]).plot()
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7efdb068a550>
```

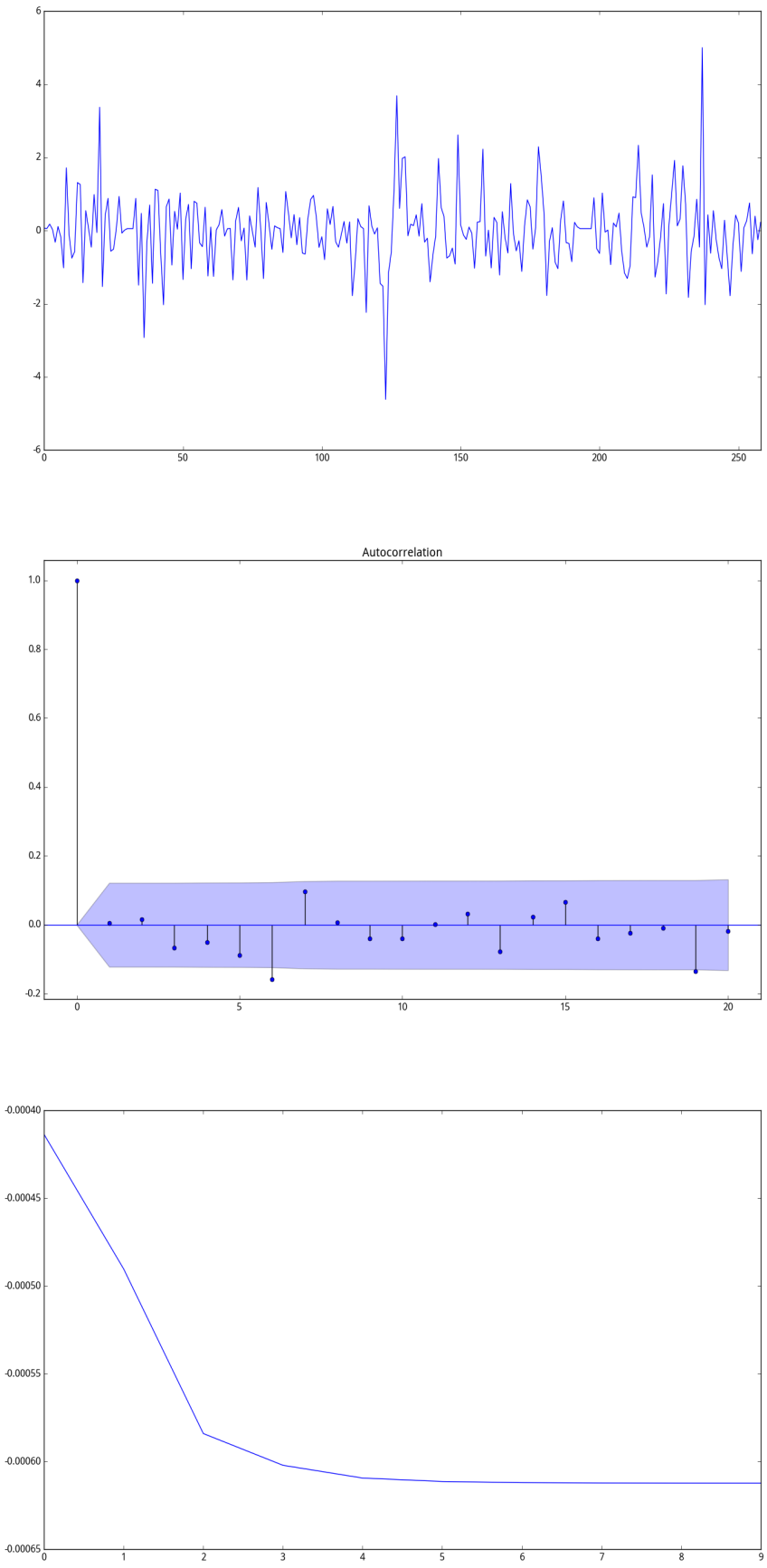


图 24.7: 题 8

第二十五章　GARCH

1. $\because Cov(\varepsilon_t, \varepsilon_{t-k}) = E((\varepsilon_t - E(\varepsilon_t))(\varepsilon_{t-k} - E(\varepsilon_{t-k}))) = E(\varepsilon_t\varepsilon_{t-k}) - E(\varepsilon_t)E(\varepsilon_{t-k}) = E(\sigma_t u_t \sigma_{t-k} u_{t-k}) = E(\sigma_t)E(\sigma_{t-k})E(u_t)E(u_{t-k}) = 0$

   $\therefore Cor(\varepsilon_t, \varepsilon_{t-k}) = 0$

2. $\because \varepsilon_t = \sigma_t u_t$

   $\therefore \varepsilon_t^2 = \sigma_t^2 u_t^2 = \sigma_t^2 + \sigma_t^2(u_t^2 - 1) = \sigma_t^2 + \eta_t$

   $\sigma_t^2 = \sum_{i=1}^{p} \alpha_i \varepsilon_t^2$ 就可以转换为：$\varepsilon_t^2 = \sum_{i=1}^{p} \alpha_i \varepsilon_t^2 + \eta_t$

3.
```
In [1]: import pandas as pd

In [2]: CRSPday=pd.read_csv('CRSPday.csv')

In [3]: ibm=CRSPday.ibm
```

(a)
```
In [4]: ibm.plot()
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f4a1bd198>
```

可以看到，较大的波动后总是跟随着一些较大的波动，而较小的波动后则会跟随着较小的波动。因此，图中出现了较为明显的波动聚集现象。

(b)
```
In [5]: from statsmodels.graphics.tsaplots import *

In [6]: plot_acf(ibm**2,lags=20)
```

很明显，前几阶的自相关系数都是显著的。因此，我们可以认为 ibm 具有 ARCH 效应。

(c)
```
In [7]: from statsmodels.tsa import stattools

In [8]: LjungBox=stattools.q_stat(stattools.acf(ibm**2)[1:13],len(ibm))

In [9]: LjungBox[1][-1]
Out[9]: 8.5307836575770328e-12
```

Ljung-Box 检验的 p 值为 $8.53e-12$，小于 0.05。因此，该序列不是一个白噪声，我们可以认为 ibm 具有 ARCH 效应。

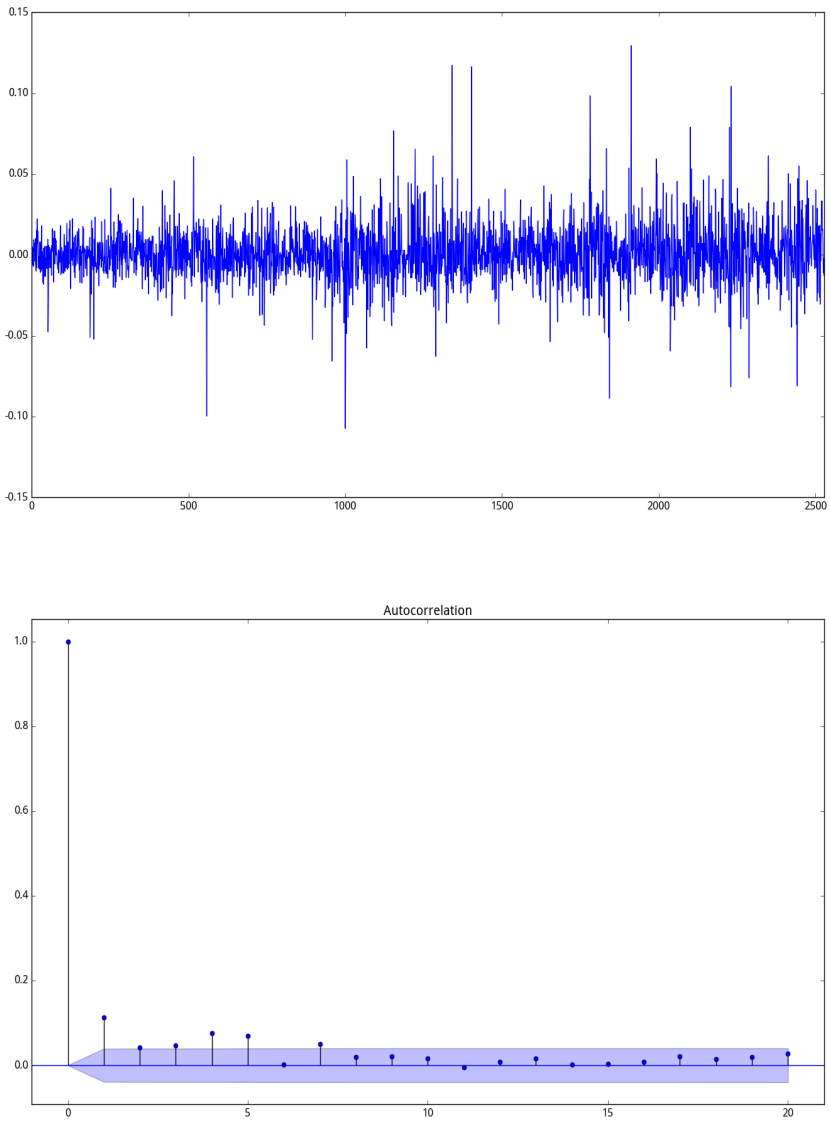


图 25.1: 题 3

4.

```
In [10]: import pandas_datareader.data as web
    ...: import datetime as dt

In [11]: google = web.DataReader('GOOGL','yahoo',
    ...:                         dt.datetime(2004,1,1),dt.datetime(2015,12,31))

In [12]: google = google.asfreq('M','ffill','end')

In [13]: googleRet = (google.Close-google.Close.shift(1))/google.Close.shift(1)

In [14]: googleRet = googleRet.dropna()
```

(a)

```
In [15]: googleRet.plot()
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f487240b8>
```

从时间序列图上看，$googleRet$ 并没有明显的趋势，是平稳的时间序列。

(b)

```
In [16]: plot_acf(googleRet,lags=20)

In [17]: plot_pacf(googleRet,lags=20)
```

根据 ACF 图和 PACF 图，$googleRet$ 不仅是一个平稳序列，还是一个白噪声。

(c)

```
In [18]: LjungBox=stattools.q_stat(stattools.acf(googleRet)[1:13],len(
    googleRet))

In [19]: LjungBox[1][-1]
Out[19]: 0.78429206851907785
```

Ljung-Box 检验的 p 值为 0.7843，大于 0.05。因此，该序列是一个白噪声。

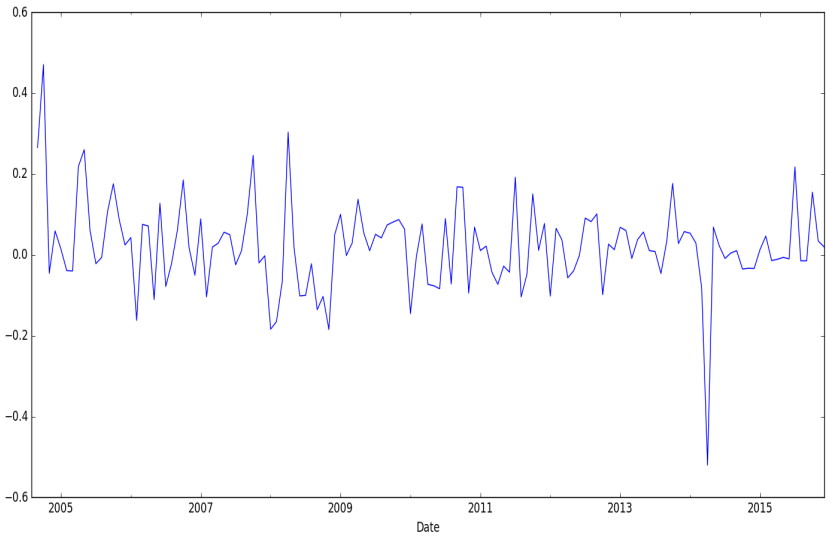
(d)

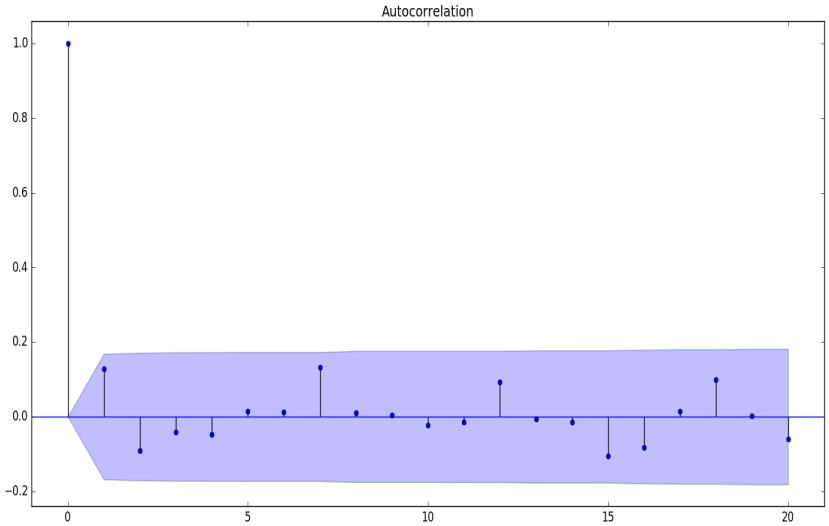
```
In [20]: (googleRet**2).plot()
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1f2d3e36d8>
```

通过绘制序列平方的时间序列图，我们可以更加容易地发现序列的波动聚集现象。在该图中，波动聚集现象就很明显。

(e)

```
In [21]: plot_acf(googleRet**2,lags=20)

In [22]: plot_pacf(googleRet**2,lags=20)
```


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
2005
2007
2009
2011
2013
2015
Date


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20

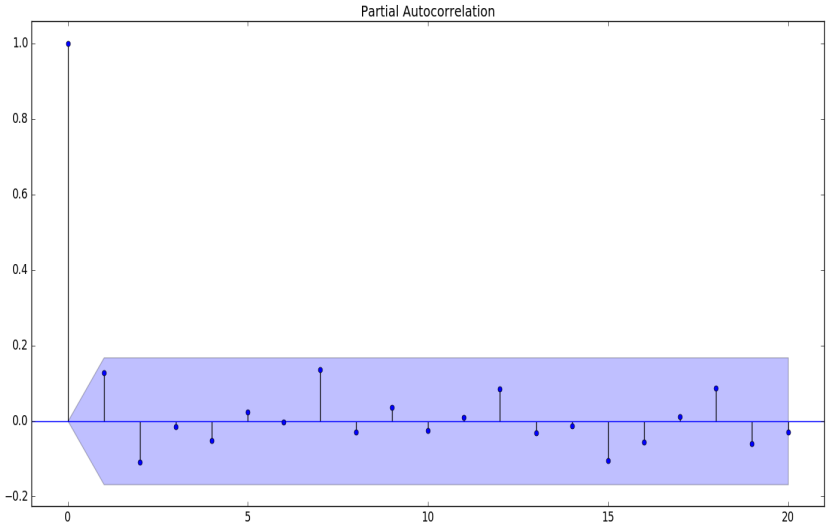

Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20

图 25.2: 题 4

(a)

```
In [23]: LjungBox=stattools.q_stat(stattools.acf(googleRet**2)[1:13],len(
    googleRet))

In [24]: LjungBox[1][-1]
Out[24]: 0.93944694696639508
```

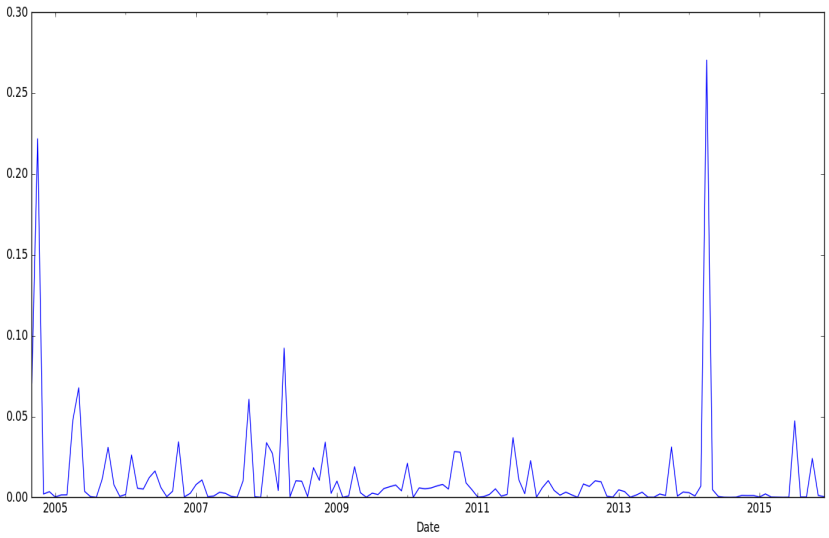
Ljung-Box 检验的 p 值为 0.9394，大于 0.05。因此，该序列是一个白噪声，说明不存在 ARCH 效应。

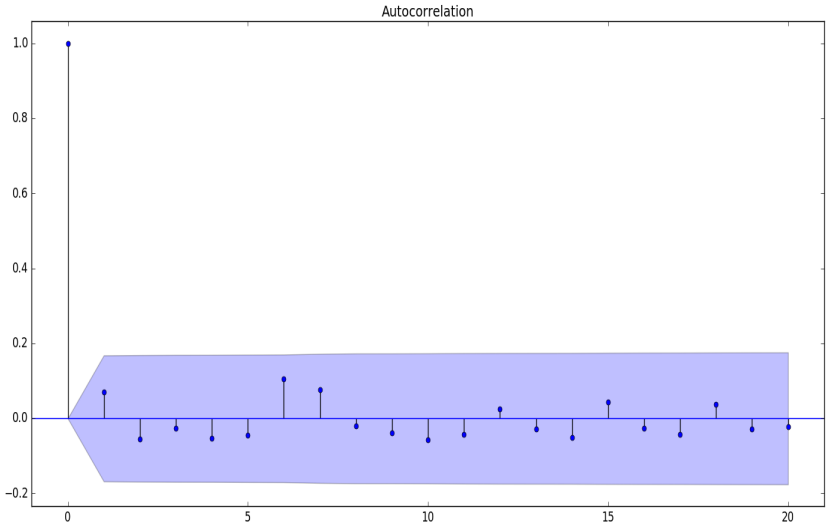
(b) 我们试着拟合一个 GARCH(1,1) 模型来看看我们之前的结论是否正确。

```
In [25]: from arch import arch_model

In [26]: am = arch_model(googleRet)

In [27]: model = am.fit(update_freq=0)
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 1489.58011154
            Iterations: 10
            Function evaluations: 68
            Gradient evaluations: 10

In [28]: print(model.summary())
```

```
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                  Close   R-squared:                      -0.002
Mean Model:             Constant Mean   Adj. R-squared:                 -0.002
Vol Model:                      GARCH   Log-Likelihood:                116.703
Distribution:                  Normal   AIC:                           -225.407
Method:            Maximum Likelihood   BIC:                           -213.756
                                        No. Observations:                  136
Date:                   日, 1月 22 2017   Df Residuals:                      132
Time:                        11:44:44   Df Model:                            4
                                 Mean Model
==============================================================================
                 coef    std err          t      P>|t|        95.0% Conf. Int.
------------------------------------------------------------------------------
mu             0.0261  7.296e-03      3.580  3.430e-04   [1.182e-02,4.042e-02]
                              Volatility Model
==============================================================================
                 coef    std err          t      P>|t|        95.0% Conf. Int.
------------------------------------------------------------------------------
omega      5.6277e-03  1.698e-03      3.313  9.214e-04   [2.299e-03,8.957e-03]
alpha[1]       0.8281      0.476      1.741  8.169e-02       [ -0.104,  1.760]
beta[1]        0.0000  6.237e-02      0.000      1.000       [ -0.122,  0.122]
==============================================================================
Covariance estimator: robust
```

可以看到，系数在 5% 的水平都不显著，说明 Google 的月度收益率序列确实不存在 ARCH 效应。


0.30
0.25
0.20
0.15
0.10
0.05
0.00
2005
2007
2009
2011
2013
2015
Date


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20

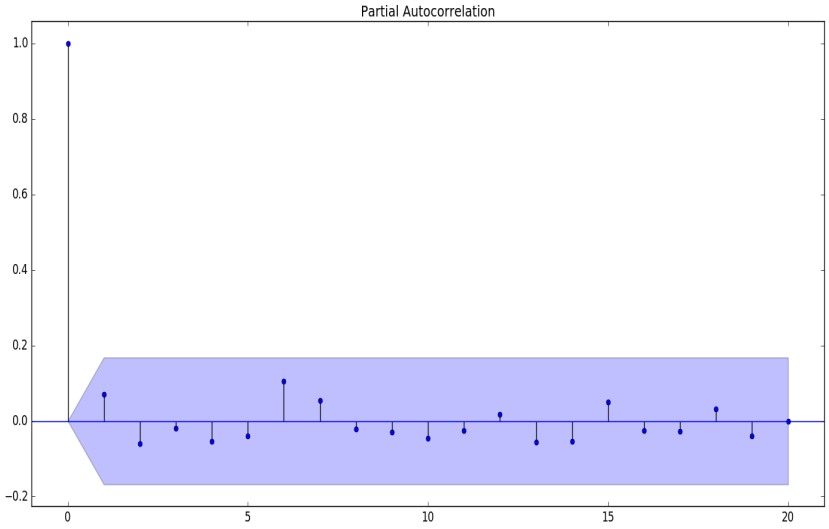

Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20

图 25.3: 题 4

5.

```
In [29]: cny = web.DataReader('CNY=X','yahoo',
    ...:                      dt.datetime(2015,1,1),dt.datetime(2015,12,31))

In [30]: ret = (cny.Close-cny.Close.shift(1))/cny.Close.shift(1)

In [31]: ret = ret.dropna()
```

(a)

```
In [25]: cny.Close.plot()
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7a18599ef0>
```

图中存在很明显的上升趋势，因此人民币的汇率不是一个平稳序列。

(b)

```
In [27]: ret.plot()
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7a18475630>
```

与人民币的汇率不同，其变动百分比序列并没有明显的趋势，是一个平稳序列。

(c)

```
In [28]: plot_acf(ret,lags=20)

In [28]: plot_pacf(ret,lags=20)
```

变动百分比序列的 ACF 图和 PACF 图中均有几个系数是显著的，因此，该序列可能并不是一个白噪声。

(d)

```
In [29]: LjungBox=stattools.q_stat(stattools.acf(ret)[1:13],len(ret))

In [30]: LjungBox[1][-1]
Out[30]: 0.0036343939189770673
```

Ljung-Box 的 p 值为 0.00363，因此我们可以拒绝原假设。该结果证明了我们上面的猜想，变动百分比序列并不是一个白噪声。

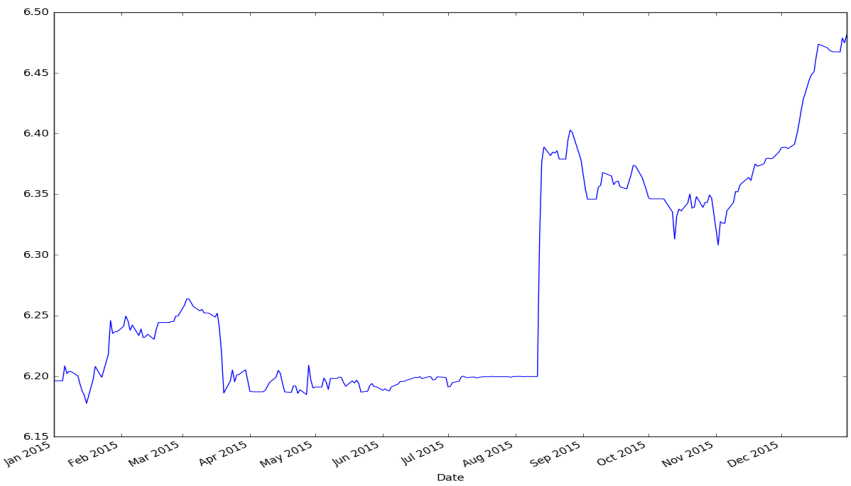
(e)

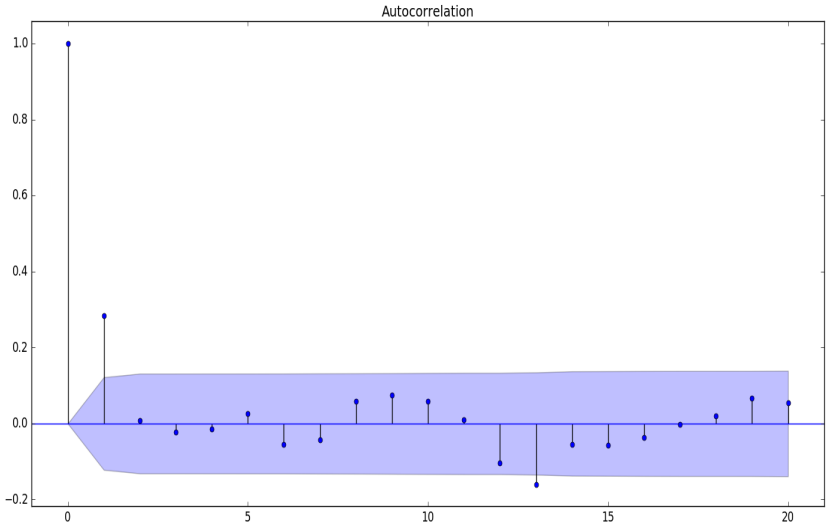
```
In [31]: (ret**2).plot()
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7a1849a550>
```

很明显，较大的波动和较小的波动分别聚集在一起。因此，变动百分比序列呈现出波动聚集现象。

(f)

```
In [28]: plot_acf(ret**2,lags=20)

In [28]: plot_pacf(ret**2,lags=20)
```


6.50
6.45
6.40
6.35
6.30
6.25
6.20
6.15
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Nov 2015
Dec 2015
Date


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20

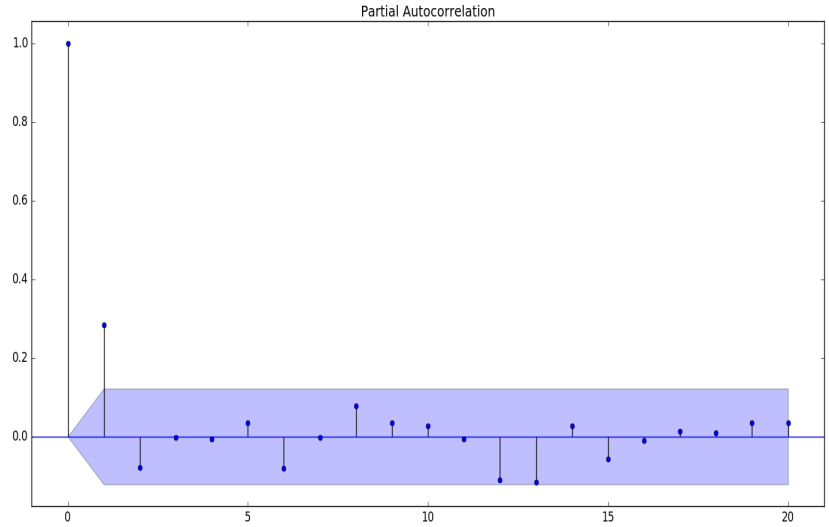

Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20

图 25.4: 题 5

(g)

```
In [29]: LjungBox=stattools.q_stat(stattools.acf(ret**2)[1:13],len(ret))

In [30]: LjungBox[1][-1]
Out[30]: 0.14677921787366507
```

Ljung-Box 的 p 值为 0.1467，因此我们不能拒绝原假设。这与我们之前观察 ACF 图和 PACF 图得出的结论不一致，因此，我们可以通过拟合 GARCH 模型来看看到底有没有 ARCH 效应。

(h)

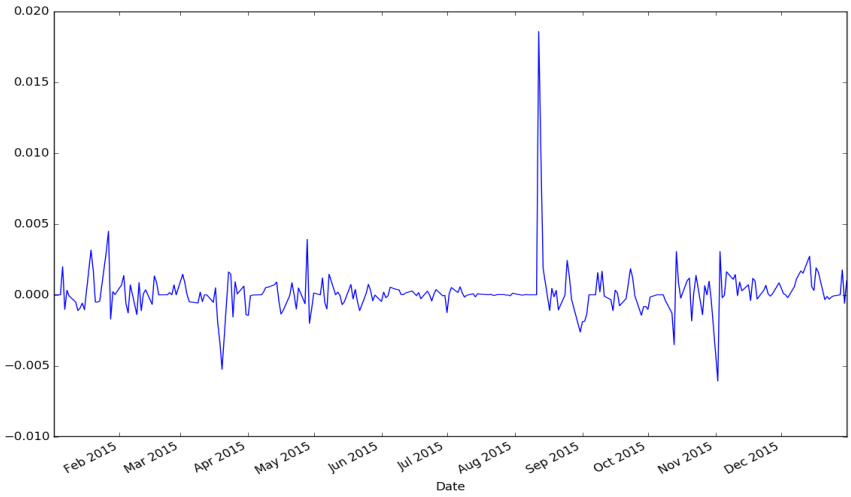
```
In [39]: from arch.univariate import ARX,GARCH

In [40]: model=ARX(ret,lags=1)

In [41]: model.volatility=GARCH()

In [41]: res = model.fit()

In [44]: print(res.summary())
```

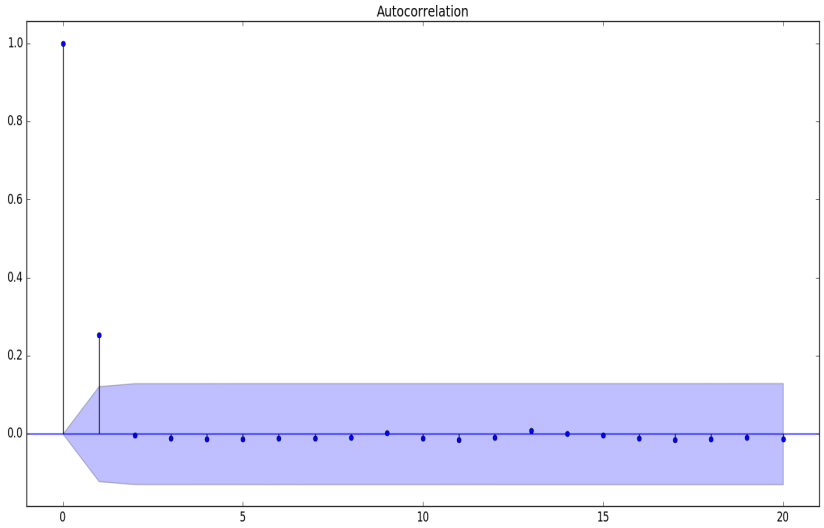
```
                        AR - GARCH Model Results
==============================================================================
Dep. Variable:                   Close   R-squared:                       0.081
Mean Model:                         AR   Adj. R-squared:                  0.077
Vol Model:                      GARCH   Log-Likelihood:                1288.18
Distribution:                  Normal   AIC:                           -2566.36
Method:            Maximum Likelihood   BIC:                           -2548.59
                                        No. Observations:                  258
Date:                   日, 1月 22 2017   Df Residuals:                      253
Time:                          19:45:35   Df Model:                          5
                                 Mean Model
==============================================================================
                 coef    std err          t      P>|t|        95.0% Conf. Int.
------------------------------------------------------------------------------
Const      1.2963e-04  1.221e-05     10.619  2.444e-26   [1.057e-04,1.536e-04]
Close[1]       0.2839      0.104      2.720  6.520e-03     [7.936e-02,  0.488]
                            Volatility Model
==============================================================================
                 coef    std err          t      P>|t|        95.0% Conf. Int.
------------------------------------------------------------------------------
omega      1.3716e-06  1.105e-12  1.241e+06      0.000   [1.372e-06,1.372e-06]
alpha[1]       0.0500      0.218      0.229      0.819       [ -0.378,  0.478]
beta[1]        0.4500      0.461      0.976      0.329       [ -0.454,  1.354]
==============================================================================
Covariance estimator: robust
```

拟合的模型为

$$\begin{cases} cny_t = 1.296e-03 + 0.284 cny_{t-1} + \varepsilon_t \\ \varepsilon_t = \sigma_t u_t \\ \sigma_t^2 = 1.372e-06 + 0.050\sigma_{t-1}^2 + 0.450\varepsilon_t^2 \end{cases}$$

根据拟合的模型，volatility model 的系数都不显著，说明并不存在 ARCH 效应。


0.020
0.015
0.010
0.005
0.000
−0.005
−0.010
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Nov 2015
Dec 2015
Date


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20

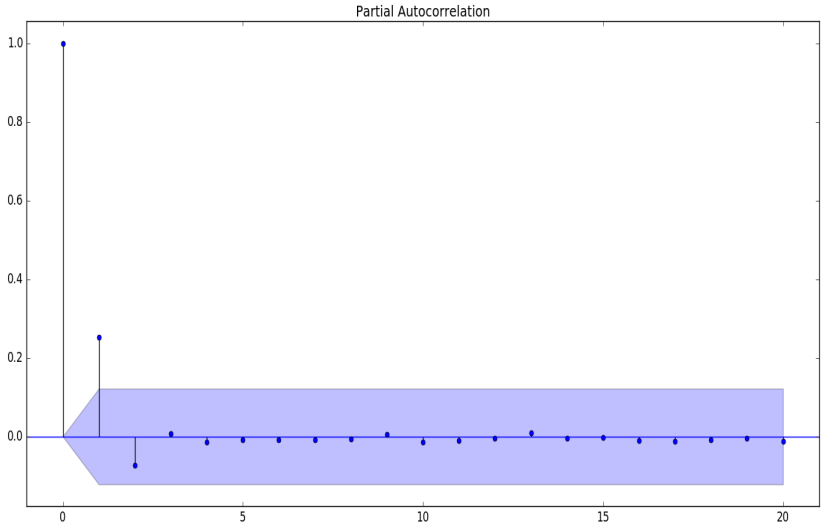

Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20

图 25.5: 题 5

第二十六章　配对交易

1.

```
In [1]: import pandas as pd
   ...:
   ...: import pandas_datareader.data as web
   ...:
   ...: import datetime as dt
   ...:
   ...: import statsmodels.formula.api as smf
   ...:
   ...: import numpy as np
   ...:
   ...: from arch.unitroot import ADF

In [2]: zgyh = web.DataReader('601988.SS','yahoo',
   ...:                       dt.datetime(2009,1,2),dt.datetime(2012,12,31))

In [3]: zxyh = web.DataReader('601998.SS','yahoo',
   ...:                       dt.datetime(2009,1,2),dt.datetime(2012,12,31))
```

(a)

```
In [4]: train_zgyh = zgyh.Close['2009':'2011']

In [5]: train_zxyh = zxyh.Close['2009':'2011']

In [6]: train_zxyh.corr(train_zgyh)
Out[6]: 0.81995687956478036
```

两者的相关系数高达 0.82，表明我们可以对这两支股票进行配对。

(b)

```
In [7]: trainset=pd.concat([train_zxyh,train_zgyh],1)

In [8]: trainset.columns=['zxyh','zgyh']

In [9]: trainset=np.log(trainset)

```

```
In [10]: model = smf.ols('zgyh~zxyh',data=trainset).fit()

In [11]: resid = model.resid

In [12]: adf=ADF(resid,trend='nc')

In [13]: print(adf.summary().as_text())
   Augmented Dickey-Fuller Results
=====================================
Test Statistic                 -2.428
P-value                         0.015
Lags                                1
-------------------------------------

Trend: No Trend
Critical Values: -2.57 (1%), -1.94 (5%), -1.62 (10%)
Null Hypothesis: The process contains a unit root.
Alternative Hypothesis: The process is weakly stationary.
```

单位根检验的统计量为 -2.428，小于 5% 的临界值。因此，价差序列是平稳的。

(c)

```
In [14]: spread = trainset.zgyh - trainset.zxyh

In [15]: resid.describe()
Out[15]:
count    7.280000e+02
mean     2.976862e-16
std      6.673705e-02
min     -1.094785e-01
25%     -5.151544e-02
50%     -2.006203e-02
75%      4.786318e-02
max      2.227136e-01
Name: None, dtype: float64
```

2. 略

3.

```
In [16]: sh50 = pd.read_csv('Data/Part5/001/sh50.csv')

In [17]: code = sh50.Code[:20]

#定义计算SSD函数
In [18]: def pair(a):
```

```
    ...:     if len(a) == 2:
    ...:         stock_a = web.DataReader('%d.SS' % a[0],'yahoo',
    ...:                     dt.datetime(2015,1,1),dt.datetime(2015,6,30))
    ...:         stock_b = web.DataReader('%d.SS' % a[1],'yahoo',
    ...:                     dt.datetime(2015,1,1),dt.datetime(2015,6,30))
    ...:         x = np.log(stock_a.Close)
    ...:         y = np.log(stock_b.Close)
    ...:         ret_x = x.diff()[1:]
    ...:         ret_y = y.diff()[1:]
    ...:         standard_x = (1+ret_x).cumprod()
    ...:         standard_y = (1+ret_y).cumprod()
    ...:         SSD = ((standard_x-standard_y)**2).sum()
    ...:         return(pd.DataFrame({'SSD':[SSD],'stock1':[a[0]],'stock2':[a
       [1]]}))
    ...:     else:
    ...:         dat = pd.DataFrame({'SSD':[0],'stock1':[0],'stock2':[0]})
    ...:         for i in range(1,len(a)):
    ...:             dat = dat.append(pair([a[0],a[i]]))
    ...:         return(dat.append(pair(a[1:])))
    ...:
    ...:

In [19]: pair_SSD = pair(code.values)
#选取SSD值小于1的股票进行配对
In [20]: pair_SSD = pair_SSD[pair_SSD.SSD!=0]

In [21]: pair_SSD = pair_SSD[pair_SSD.SSD<=1]
#输出结果省略
```

第五部分

技术指标

第二十七章　K 线

1.

```
In [1]: %paste
import pandas as pd
import numpy as np
import candle as cd
import matplotlib.pyplot as plt

shzheng = pd.read_csv('Data/Part5/001/problem27-1.csv',
                      index_col='date')
shzheng.index.name='Date'
shzheng.index = pd.to_datetime(shzheng.index,format='%Y-%m-%d')
shzheng13=shzheng['2013-03-01':'2013-05-31']
cd.candleVolume(shzheng13,
                      candletitle='Candle Plot of Shanghai Composite Index',
                      bartitle='volume')
```

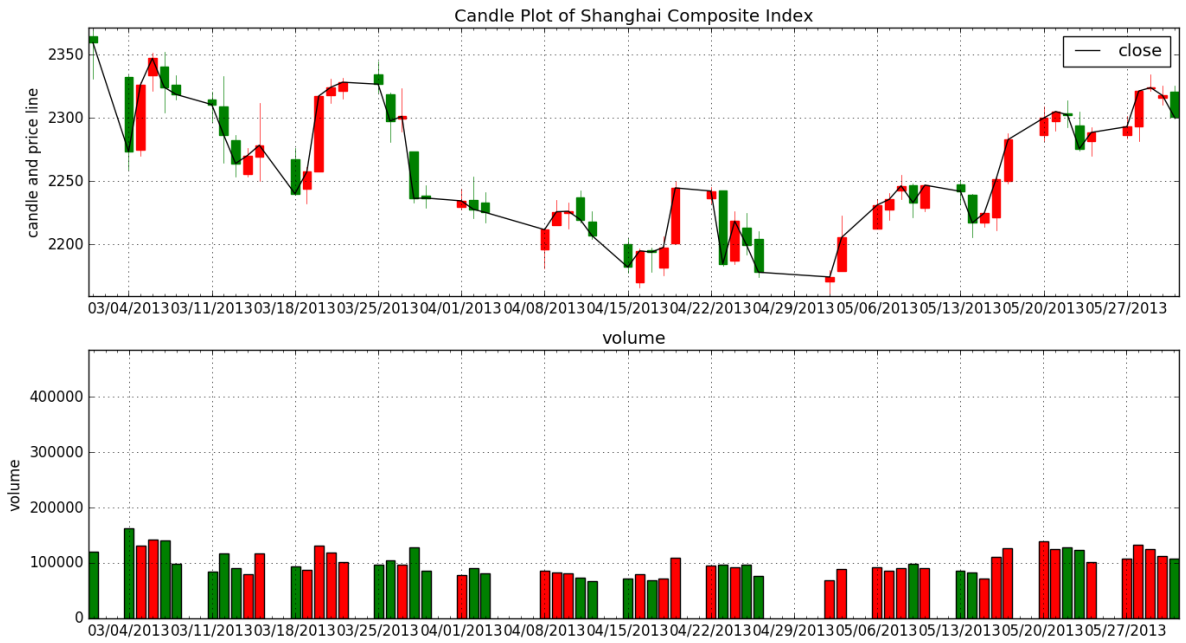


图 27.1: 题 1

2.

```
In [1]: shzheng = pd.read_csv('Data/Part5/001/problem27-2.csv',
   ....:                      index_col='date')

In [2]: shzheng.index.name='Date'

In [3]: shzheng.index = pd.to_datetime(shzheng.index,format='%Y-%m-%d')

In [4]: shzheng131=shzheng['2013-01-01':'2013-06-30']

In [5]: CL_OP = shzheng131.Close - shzheng131.Open

In [6]: CL_OP.describe()
Out[6]:
count      78.000000
mean       -1.854376
std        25.289310
min      -105.627076
25%       -15.605286
50%         0.348023
75%        11.783997
max        59.637939
dtype: float64

In [7]: Doji = pd.Series(np.where(np.abs(CL_OP.values)<8,1,0),
   ....:                 index=CL_OP.index)

In [8]: Doji[Doji==1].index
Out[8]:
DatetimeIndex(['2013-03-01', '2013-03-08', '2013-03-11', '2013-03-21',
               '2013-03-22', '2013-03-27', '2013-03-29', '2013-04-01',
               '2013-04-02', '2013-04-03', '2013-04-10', '2013-04-17',
               '2013-04-22', '2013-05-02', '2013-05-07', '2013-05-08',
               '2013-05-13', '2013-05-15', '2013-05-22', '2013-05-24',
               '2013-05-27', '2013-05-29', '2013-05-30', '2013-06-03',
               '2013-06-05', '2013-06-18', '2013-06-19', '2013-06-26',
               '2013-06-27'],
              dtype='datetime64[ns]', name='Date', freq=None)

In [9]: cd.candlePlot(shzheng131,
   ....:                 title='Candle Plot of Shanghai Composite Index')
```

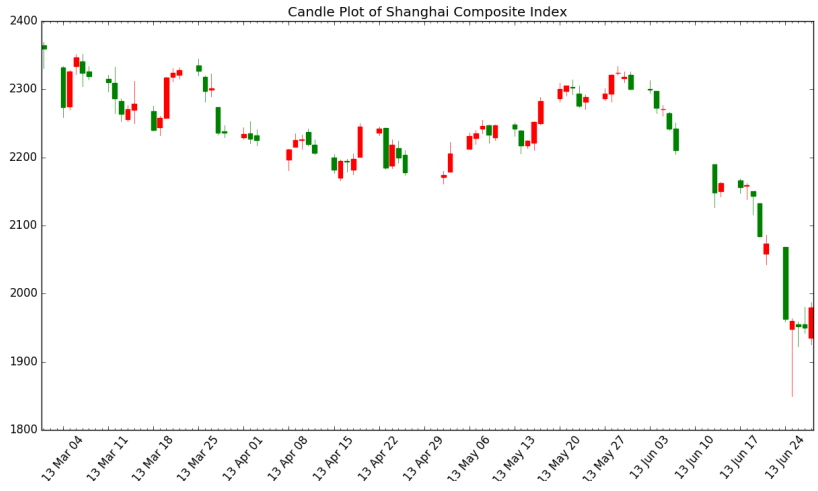


图 27.2: 题 2

3.

(a)

```
In [1]: shzheng = pd.read_csv('Data/Part5/001/problem27-3.csv',
   ...:                       index_col='date')

In [2]: shzheng.index.name='Date'

In [3]: shzheng.index = pd.to_datetime(shzheng.index,format='%Y-%m-%d')
```

```
#刻画蜡烛实体
In [4]: CL_OP = shzheng.Close - shzheng.Open

In [5]: CL_OP.describe()
Out[5]:
count     243.000000
mean        2.244368
std        22.884043
min       -73.685059
25%       -11.442016
50%         1.513183
75%        14.296997
max        91.843994
dtype: float64

In [6]: dat = pd.concat([CL_OP,CL_OP.shift(1),CL_OP.shift(2)],1)

In [22]: candle = np.all([np.abs(dat.iloc[:,1])<=3,dat.iloc[:,2]>11,
   ....:                  dat.iloc[:,0]<-6,
```

```
....:                     np.abs(dat.iloc[:,0])<np.abs(dat.iloc[:,2])/2],
....:                     0)

#刻画十字星
In [8]: dataCOP = pd.concat([shzheng.Close,shzheng.Open.shift(1),

...: shzheng.Close.shift(1),shzheng.Open.shift(2)],1)

In [9]: Doji = np.all([dataCOP.iloc[:,1]>dataCOP.iloc[:,3],
...: dataCOP.iloc[:,1]>dataCOP.iloc[:,0],
...: dataCOP.iloc[:,2]>dataCOP.iloc[:,3],
...: dataCOP.iloc[:,2]>dataCOP.iloc[:,0]],0)

#刻画上涨趋势
In [10]: ret = (shzheng.Close-shzheng.Close.shift(1))/shzheng.Close.shift(1)

In [11]: trend = np.all([ret.shift(2)>0,ret.shift(1)>0],0)
```

(b)

```
In [12]: signal = np.all([candle,Doji,trend],0)

In [13]: sum(signal)
Out[13]: 0
```

最终，没有发现“黄昏之星”。

4.

(a)

```
In [28]: shzheng = pd.read_csv('Data/Part5/001/problem27-4.csv',
...:                           index_col='date')

In [29]: shzheng.index.name='Date'

In [30]: shzheng.index = pd.to_datetime(shzheng.index,format='%Y-%m-%d')
```

```
#刻画蜡烛实体
In [31]: CL_OP = shzheng.Close - shzheng.Open

In [32]: candle = np.all([CL_OP<0,CL_OP.shift(1)>0],0)

#刻画价格差距
In [33]: diffPrice = np.all([shzheng.Open>shzheng.Close.shift(1),shzheng.
    Close<shzheng.Open.shift(1)],0)
```

```
#刻画上涨趋势
In [34]: ret = (shzheng.Close-shzheng.Close.shift(1))/shzheng.Close.shift(1)

In [35]: trend = np.all([ret.shift(2)>0,ret.shift(1)>0],0)
```

(b)

```
In [36]: signal = np.all([candle,diffPrice,trend],0)

In [37]: sum(signal)
Out[37]: 0
```

最终，我们并没有发现“看跌吞没”。

5.

(a)

```
#接上题
#刻画蜡烛实体
In [38]: CL_OP.describe()
Out[38]:
count    245.000000
mean       0.886421
std       28.573461
min      -85.802002
25%      -15.752930
50%        0.811035
75%       17.271973
max       84.511963
dtype: float64

#刻画蜡烛实体
In [39]: candle = np.all([CL_OP<0,CL_OP.shift(1)>17],0)

#刻画价格差距
In [40]: diffPrice = np.all([shzheng.Close<shzheng.Open.shift(1)],0)

#刻画上涨趋势
In [41]: trend = np.all([ret.shift(1)>0,ret.shift(2)>0],0)
```

```
In [42]: signal = pd.Series(np.all([candle,diffPrice,trend],0),index=CL_OP.
    index)

```

```
In [43]: signal[signal==True]
Out[43]:
Date
2011-01-20      True
2011-05-04      True
2011-09-22      True
dtype: bool

In [44]: cd.candlePlot(shzheng['2011-01-01':'2011-02-15'],'2011-01-20')

In [45]: cd.candlePlot(shzheng['2011-04-15':'2011-05-15'],'2011-05-04')

In [46]: cd.candlePlot(shzheng['2011-09-15':'2011-10-15'],'2011-09-22')
```

从 K 线图的走势我们可以看出，1 月 20 号的“倾盆大雨”是虚假信号，而另外两个都是真实的信号。

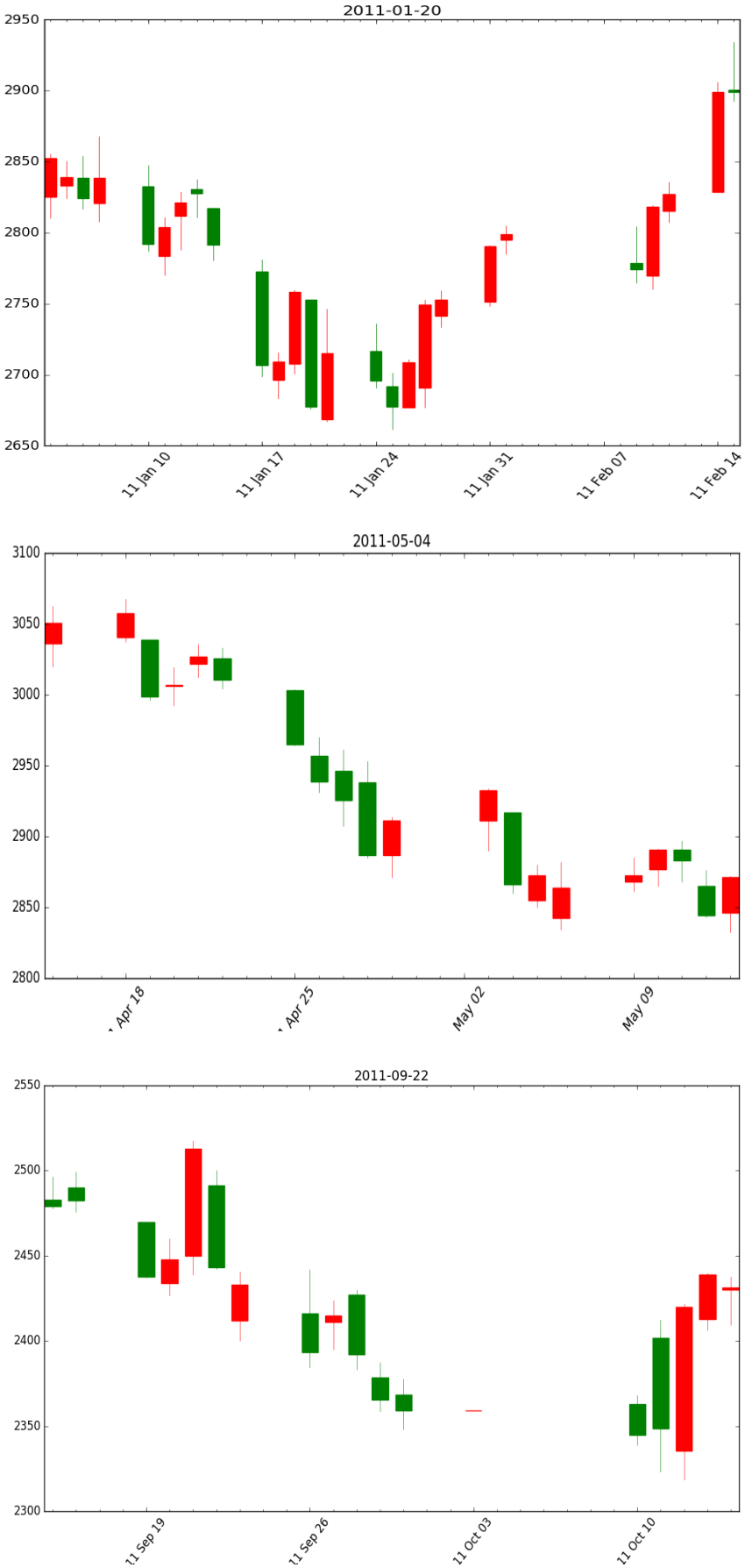

2011-01-20
2950
2900
2850
2800
2750
2700
2650
11 Jan 10
11 Jan 17
11 Jan 24
11 Jan 31
11 Feb 07
11 Feb 14
2011-05-04
3100
3050
3000
2950
2900
2850
2800
Apr 18
Apr 25
May 02
May 09
2011-09-22
2550
2500
2450
2400
2350
2300
11 Sep 19
11 Sep 26
11 Oct 03
11 Oct 10

图 27.3: 题 5

第二十八章　动量

(b)

```
In [1]: import pandas as pd
   ...:
   ...: import numpy as np
   ...:
   ...:
In [2]: pufa = pd.read_csv('Data/Part5/002/problem28-1.csv',index_col='date')

In [3]: pufa.index.name='Date'

In [4]: pufa.index = pd.to_datetime(pufa.index,format='%Y-%m-%d')
```

(a)

```
#6日动量
In [5]: mom6_1 = pufa.Close - pufa.Close.shift(6)

In [6]: mom6_2 = (pufa.Close - pufa.Close.shift(6))/pufa.Close.shift(6)

#30日动量
In [7]: mom30_1 = pufa.Close - pufa.Close.shift(30)

In [8]: mom30_2 = (pufa.Close - pufa.Close.shift(30))/pufa.Close.shift(30)

#90日动量
In [9]: mom90_1 = pufa.Close - pufa.Close.shift(90)

In [10]: mom90_2 = (pufa.Close - pufa.Close.shift(90))/pufa.Close.shift(90)
```

```
#定义计算交易信号函数
In [11]: def getSignal(x):
    ...:     signal = np.where(x>0,1,np.where(x<0,-1,0))
    ...:     return(signal)
    ...:
    ...:
```

```
#计算交易信号
In [12]: signal6 = getSignal(mom6_2)

In [13]: signal30 = getSignal(mom30_2)

In [14]: signal90 = getSignal(mom90_2)

#计算收益率
In [15]: ret = (pufa.Close-pufa.Close.shift(1))/pufa.Close.shift(1)

In [16]: ret6 = ret[6:]

In [17]: ret30 = ret[30:]

In [18]: ret90 = ret[90:]

In [19]: signal6 = pd.Series(signal6[6:],index=ret6.index)

In [20]: Mom6Ret = ret6[1:] * signal6.shift(1)[1:]

In [21]: signal30 = pd.Series(signal6[30:],index=ret30.index)

In [22]: Mom30Ret = ret30[1:] * signal30.shift(1)[1:]

In [23]: signal90 = pd.Series(signal6[90:],index=ret90.index)

In [24]: Mom90Ret = ret90[1:] * signal90.shift(1)[1:]
#计算获胜率
In [25]: winrate6 = sum(Mom6Ret>0)/sum(Mom6Ret!=0)

In [26]: winrate6
Out[26]: 0.44642857142857145

In [27]: winrate30 = sum(Mom30Ret>0)/sum(Mom30Ret!=0)

In [28]: winrate30
Out[28]: 0.44390243902439025

In [29]: winrate90 = sum(Mom90Ret>0)/sum(Mom90Ret!=0)
```

```
47
48 In [30]: winrate90
49 Out[30]: 0.44966442953020136
```

在这三个策略中，三种动量的获胜率几乎相同，使用 90 日动量的策略稍微略胜一筹。

```
1 #定义计算交易信号函数
2 In [31]: def getSignal(x):
3     ...:     first = x.iloc[:,0]
4     ...:     second = x.iloc[:,1]
5     ...:     signal = [1 if first[i]>0 and second[i]>0 else -1 if first[i]<0 and
          second[i]<0 else 0 for i in range(len(first))]
6     ...:     return(signal)
7     ...:
8     ...:
9
10 #计算交易信号
11 In [32]: momen = pd.concat([mom6_2,mom90_2],1)
12
13 In [33]: momen = momen.dropna()
14
15 In [34]: signal = getSignal(momen)
16
17 In [35]: signal = pd.Series(signal,index= mom90_2.dropna().index)
18 #计算获胜率
19 In [36]: MomRet = ret90[2:] * signal.shift(2)[2:]
20
21 In [37]: sum(MomRet>0)/sum(MomRet!=0)
22 Out[37]: 0.45000000000000001
```

(b)

(a)

```
1 In [1]: import pandas as pd
2
3 In [2]: import numpy as np
4
5 In [3]: prices = pd.read_csv('Data/Part5/002/problem28-3.csv',
6     ...:                      index_col='date')
7
8 In [4]: prices.index.name='Date'
9
```

```
In [5]: prices.index = pd.to_datetime(prices.index,format='%Y-%m-%d')

In [6]: prices = prices.asfreq('M',how='end',method='ffill')

In [7]: ret_pf = (prices.pfyh - prices.pfyh.shift(3))/prices.pfyh.shift(3)

In [8]: ret_zg = (prices.zgyh - prices.zgyh.shift(3))/prices.zgyh.shift(3)

In [9]: ret_ms = (prices.msyh - prices.msyh.shift(3))/prices.msyh.shift(3)

In [10]: ret_gs = (prices.gsyh - prices.gsyh.shift(3))/prices.gsyh.shift(3)

In [11]: ret_js = (prices.jsyh - prices.jsyh.shift(3))/prices.jsyh.shift(3)
```

```
In [12]: %paste
cash = np.ones(len(ret_pf))*1000000
stock = np.zeros(len(ret_pf))

for i in range(4,len(cash)):
    momen = pd.Series([ret_gs[i-1],ret_js[i-1],
      ret_ms[i-1],ret_pf[i-1],ret_zg[i-1]])
    if i!=4:
        momen_lag = pd.Series([ret_gs[i-2],ret_js[i-2],
      ret_ms[i-2],ret_pf[i-2],ret_zg[i-2]])
        cash[i] = cash[i-1] + 10000 * prices.iloc[:,momen_lag.idxmax()][i]
    stock[i] = 10000 * prices.iloc[:,momen.idxmax()][i]
    cash[i] -= 10000 * prices.iloc[:,momen.idxmax()][i]

(cash[35]+stock[35]-1000000)/1000000*len(cash)
## -- End pasted text --
Out[16]: 0.072000000000000008
```

第二十九章　RSI

```
In [1]: %paste
import pandas as pd
import numpy as np
def rsi(price,period=6):
    import pandas as pd
    clprcChange=price-price.shift(1)
    clprcChange=clprcChange.dropna()
    indexprc=clprcChange.index
    upPrc=pd.Series(0,index=indexprc)
    upPrc[clprcChange>0]=clprcChange[clprcChange>0]
    downPrc=pd.Series(0,index=indexprc)
    downPrc[clprcChange<0]=-clprcChange[clprcChange<0]
    rsidata=pd.concat([price,clprcChange,upPrc,downPrc],\
    axis=1)
    rsidata.columns=['price','PrcChange','upPrc','downPrc']
    rsidata=rsidata.dropna();
    SMUP=[]
    SMDOWN=[]
    for i in range(period,len(upPrc)+1):
        SMUP.append(np.mean(upPrc.values[(i-period):i],\
        dtype=np.float32))
        SMDOWN.append(np.mean(downPrc.values[(i-period):i],\
        dtype=np.float32))
        rsi=[100*SMUP[i]/(SMUP[i]+SMDOWN[i]) \
        for i in range(0,len(SMUP))]
    indexRsi=indexprc[(period-1):]
    rsi=pd.Series(rsi,index=indexRsi)
    return(rsi)

## -- End pasted text --
```

(b)

```
In [1]: import pandas as pd
```

```
    ...:
    ...: import numpy as np

In [2]: jtyh = pd.read_csv('Data/Part5/003/problem29-1.csv',index_col='date')

In [3]: jtyh.index.name='Date'

In [4]: jtyh.index = pd.to_datetime(jtyh.index,format='%Y-%m-%d')

In [5]: RSI5 = rsi(jtyh.Close,5)

In [6]: RSI6 = rsi(jtyh.Close,6)

In [7]: RSI7 = rsi(jtyh.Close,7)

In [8]: RSI8 = rsi(jtyh.Close,8)

In [9]: RSI9 = rsi(jtyh.Close,9)

In [10]: RSI10 = rsi(jtyh.Close,10)

In [11]: RSI12 = rsi(jtyh.Close,12)

In [12]: RSI_all =pd.concat([RSI5,RSI6,RSI7,RSI8,RSI9,RSI10,RSI12],1)
```

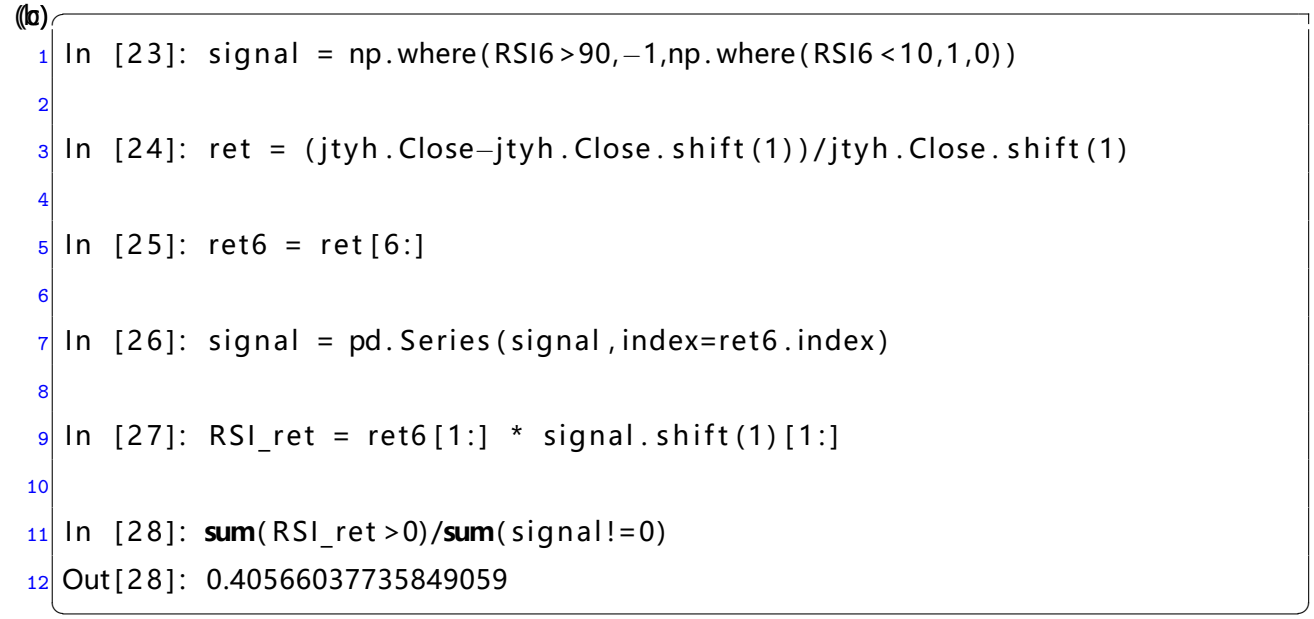
```
In [13]: jtyh = pd.read_csv('Data/Part5/003/problem29-2.csv',index_col='date')

In [14]: jtyh.index.name='Date'

In [15]: jtyh.index = pd.to_datetime(jtyh.index,format='%Y-%m-%d')
```

2. (a)

```
In [16]: import candle

In [17]: candle.candlePlot(jtyh,'Candle Plot of Bank of Communications')
```

```
In [18]: RSI6 = rsi(jtyh.Close,6)

In [19]: RSI30 = rsi(jtyh.Close,30)

In [20]: RSI = pd.concat([RSI6,RSI30],1)
```

```
In [21]: RSI.columns=['RSI6','RSI30']

In [22]: RSI.plot()
```



```
In [23]: signal = np.where(RSI6>90,-1,np.where(RSI6<10,1,0))

In [24]: ret = (jtyh.Close-jtyh.Close.shift(1))/jtyh.Close.shift(1)

In [25]: ret6 = ret[6:]

In [26]: signal = pd.Series(signal,index=ret6.index)

In [27]: RSI_ret = ret6[1:] * signal.shift(1)[1:]

In [28]: sum(RSI_ret>0)/sum(signal!=0)
Out[28]: 0.40566037735849059
```

```
In [29]: RSI_all = pd.concat([RSI6,RSI30,RSI6.shift(1),RSI30.shift(1)],1)

In [30]: RSI_all = RSI_all.dropna()

In [31]: RSI_all.columns = ['RSI6','RSI30','L_RSI6','L_RSI30']

In [32]: signal = [1 if RSI_all.RSI6[i] > RSI_all.RSI30[i] and RSI_all.L_RSI6
    [i] < RSI_all.L_RSI30[i] else \
    ...:           -1 if RSI_all.RSI6[i] < RSI_all.RSI30[i] and RSI_all.
       L_RSI6[i] > RSI_all.L_RSI30[i] else 0 \
    ...:           for i in range(len(RSI_all))]

In [33]: ret30 = ret[30:]

In [34]: signal = pd.Series(signal,index=ret30.index)

In [35]: RSI_ret = ret30[1:] * signal.shift(1)[1:]

In [36]: sum(RSI_ret>0)/sum(signal!=0)
Out[36]: 0.33695652173913043
```

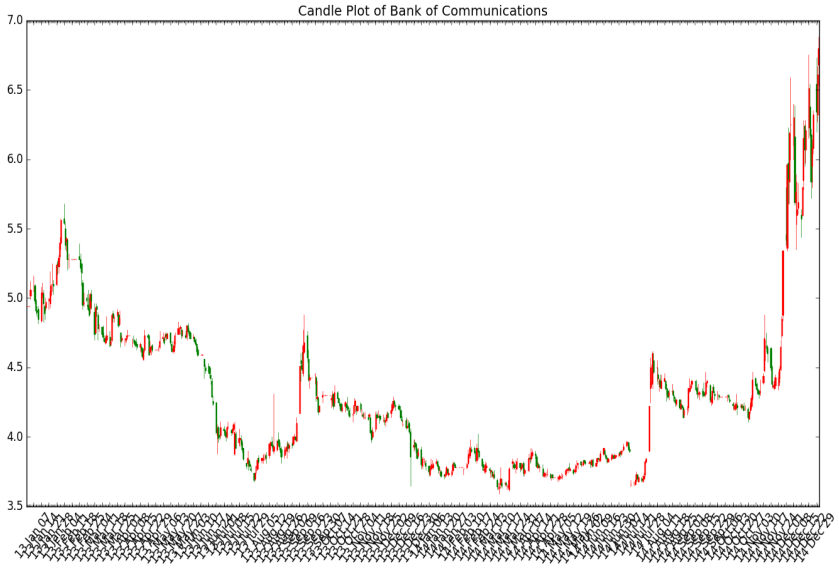


图 29.1: 题 2

(3)

```
In [37]: RSI_all = pd.concat([RSI6,RSI30],1)[24:]

In [38]: signal = np.zeros(len(RSI_all))

In [39]: RSI_all.columns = ['RSI6','RSI30']
#计算交易信号
In [40]: for i in range(3,len(RSI_all)):
    ...:     if RSI_all.RSI6[i-3] < RSI_all.RSI30[i-3] and \
    ...:     RSI_all.RSI6[i-2] > RSI_all.RSI30[i-2] and \
    ...:     RSI_all.RSI6[i-1] > RSI_all.RSI30[i-1] and \
    ...:     (RSI_all.RSI6[i-1] - RSI_all.RSI30[i-1]) < (RSI_all.RSI6[i-2] -
       RSI_all.RSI30[i-2]) and \
    ...:     (RSI_all.RSI6[i] - RSI_all.RSI30[i]) > (RSI_all.RSI6[i-1] - RSI_all.
       RSI30[i-1]):
    ...:         signal[i] = 1
    ...:
    ...:

In [41]: signal = pd.Series(signal,index = RSI_all.index)

In [42]: signal[signal==1]
Out[42]:
Date
2013-08-01    1.0
2013-10-29    1.0
```

```
2014-01-09    1.0
2014-04-10    1.0
2014-09-04    1.0
2014-11-13    1.0
dtype: float64
```

鉴于买点较少，我们可以直接查到买点次日的收益率大于 0。

```
In [33]: buyDate=signal[signal==1]

In [34]: ret.shift(-1)[buyDate.index]
Out[34]:
Date
2013-08-01   -0.002611
2013-10-29    0.007092
2014-01-09    0.002639
2014-04-10    0.002577
2014-09-04    0.006865
2014-11-13   -0.008621
Name: Close, dtype: float64
```

第三十章　均线

1.

```
In [1]: import pandas as pd
   ...: import numpy as np
   ...: import movingAverage as ma
   ...:

In [2]: dow = pd.read_csv('Data/Part5/004/problem30-1.csv', index_col='date')

In [3]: dow.index.name='Date'

In [4]: dow.index = pd.to_datetime(dow.index,format='%Y-%m-%d')

In [5]: sma = ma.smaCal(dow.Close,30)

In [6]: wma = ma.wmaCal(dow.Close,[(i+1)/465 for i in range(30)])

In [7]: ema = ma.ewmaCal(dow.Close,30,0.8)

In [8]: maValues = pd.concat([sma,wma,ema],1).replace(0,np.nan).dropna()

In [9]: maValues.columns = ['sma','wma','ema']

In [10]: maValues.plot()
```

可以看到，三者是非常接近的。读者可以自行更改 WMA 和 EMA 的权重，观察发生的变化。

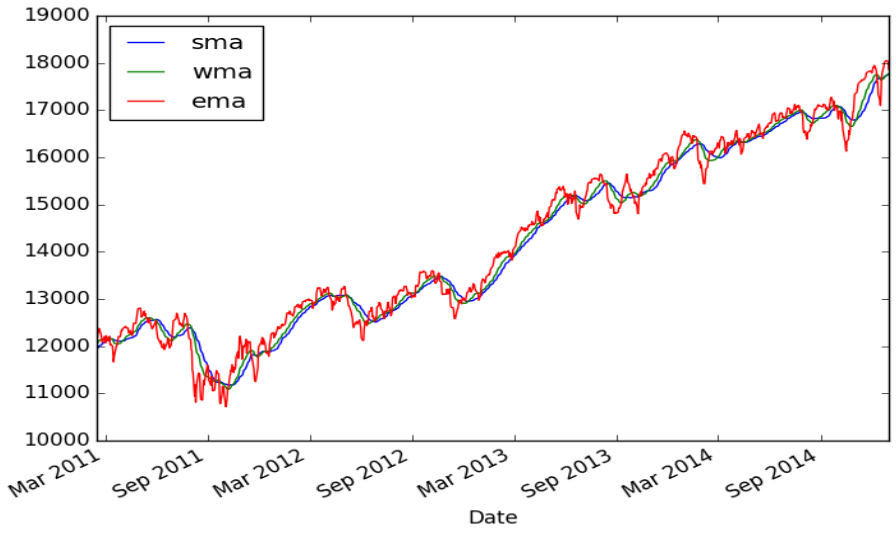


图 30.1: 题 1

2.

```
In [11]: zgyh = pd.read_csv('Data/Part5/004/problem30-2.csv',index_col='date')

In [12]: zgyh.index.name='Date'

In [13]: zgyh.index = pd.to_datetime(zgyh.index,format='%Y-%m-%d')
```

(a)

```
In [14]: import matplotlib.pyplot as plt

In [15]: mom14 = (zgyh.Close-zgyh.Close.shift(14))/zgyh.Close.shift(14)

In [16]: ma14 = ma.smaCal(zgyh.Close,14)

In [17]: data = pd.concat([zgyh.Close,mom14,ma14],1).dropna()

In [18]: data.columns = ['Close','Mom','SMA']

In [19]: axis1 = plt.subplot()
    ...:
    ...: plot1, = axis1.plot(data.index,data.Close,'-r',label='Close')
    ...: plot2, = axis1.plot(data.index,data.SMA,'-g',label='SMA')
    ...: axis2 = axis1.twinx()
    ...:
    ...: plot3, = axis2.plot(data.index,data.Mom,'-b',label='Mom')
    ...:
    ...: plt.legend(handles=[plot1,plot2,plot3])
```

```
Out[19]: <matplotlib.legend.Legend at 0x7f25cc421240>
```

```
In [20]: ret = (zgyh.Close-zgyh.Close.shift(1))/zgyh.Close.shift(1)

In [21]: signal = np.all([data.Close > data.SMA, data.Mom>0],0)

In [22]: signal = pd.Series(signal,index = ret[14:].index)

In [23]: ret = ret[15:] * signal.shift(1)[1:]

In [24]: ret.sum()
Out[24]: -0.2389164532938523
```

当价格处于均线上方时，均线可以构成对价格的有力支撑，而动量大于 0 则表明股价正在上升。因此，我们可以设计这样的一个交易策略：当价格在均线上方且动量大于 0 时，买入。当然，这只是一个简单的想法，指标和参数的选择都较为随意，所以最后的收益还是负的。

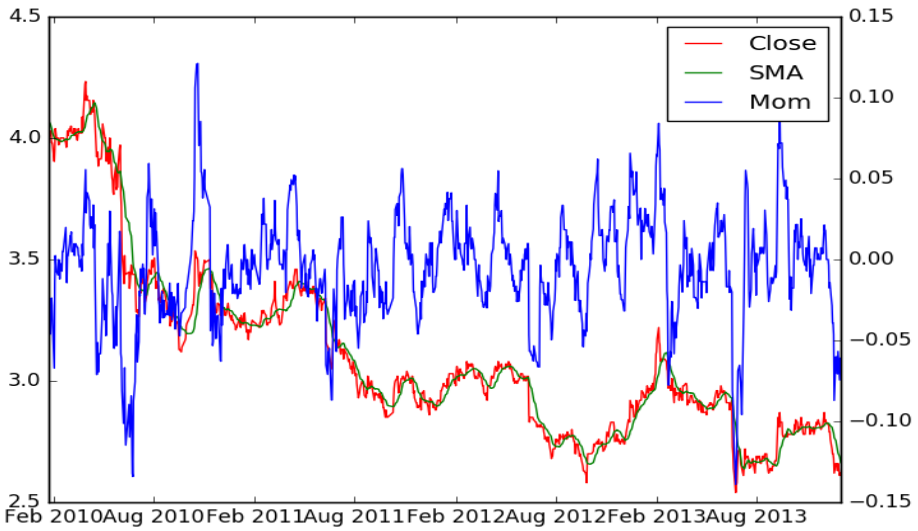


图 30.2: 题 2

(b)

```
In [1]: aapl = pd.read_csv('problem30-3.csv',index_col='date')

In [2]: aapl.index.name='Date'

In [3]: aapl.index = pd.to_datetime(aapl.index,format='%Y-%m-%d')
```

(a)

```
In [4]: import candle

```

```
In [5]: candle.candleLinePlots(aapl['2014-01':'2014-02'],splitFigures=False,
candleTitle='Candle Plot of AAPL',Data=aapl.Close['2014-01':'2014-02'],)
```

```
#因为苹果发放了股利，为了价格的一致性，我们选取了Adjusted Price，即复权过的价格
In [6]: dif = ma.ewmaCal(aapl1.Adjusted,12,2/13)-ma.ewmaCal(aapl1.Adjusted
,26,2/27)

In [7]: dif = dif[25:]

In [8]: dea = ma.ewmaCal(dif,9,2/10)

In [9]: dea = dea[8:]

In [10]: dif = dif[8:]

In [11]: macd = dif -dea

In [12]: plt.subplot(211)
    ...: plt.plot(dif[12:],label="DIF",color='k')
    ...: plt.plot(dea[12:], label="DEA",color='b',linestyle='dashed')
    ...: plt.title("DIF and DEA")
    ...: plt.legend()
    ...: plt.subplot(212)
    ...: plt.bar(left=macd[12:].index,height=macd[12:],
    ...:         label='MACD',color='r')
    ...: plt.legend()
Out[12]: <matplotlib.legend.Legend at 0x7f25cc6d9ac8>
```

(a)

```
In [13]: price = aapl1.Adjusted

In [14]: dat = pd.concat([price,dea,dif],1).dropna()]

In [15]: dat.columns = ['price','dea','dif']

In [16]: asset = 100000 * np.ones(len(dat))

In [17]: cash = 100000 * np.ones(len(dat))

In [18]: share = np.zeros(len(dat))

```

```
In [19]: for i in range(1,len(dat)):
    ...:     cash[i]=cash[i-1]+share[i-1]*dat.price[i]
    ...:     asset[i]=cash[i]
    ...:     if dat.dif[i] > dat.dea[i] >0 and dat.dif[i-1] < dat.dea[i-1]:
    ...:         share[i]=100
    ...:         cash[i]=cash[i]-100*dat.price[i]
    ...:     elif dat.dif[i] < dat.dea[i] < 0 and dat.dif[i-1] > dat.dea[i
        -1]:
    ...:         share[i]=-100
    ...:         cash[i]=cash[i]+100*dat.price[i]
    ...:
    ...:

In [20]: asset[-5:]
Out[20]: array([ 100022.3615,  100183.0869,  100183.0869,  100183.0869,
    100183.0869])
```

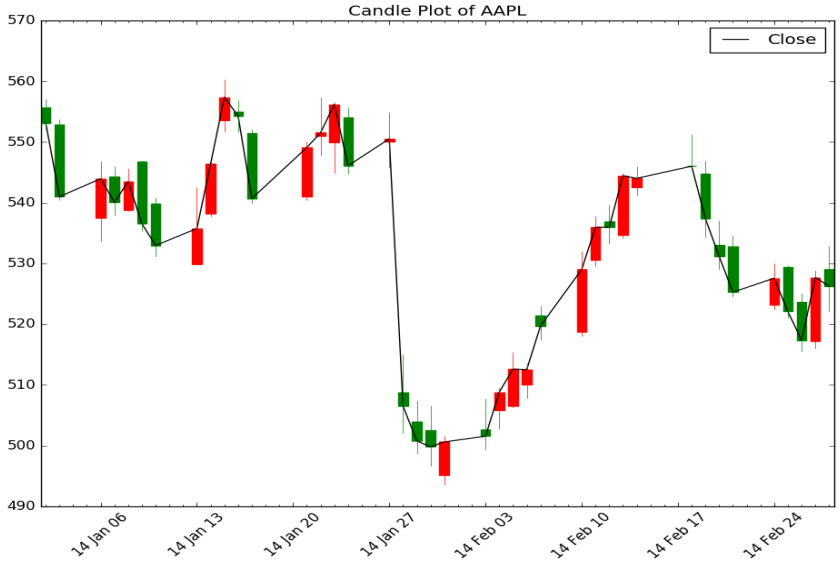


图 30.3: 题 3

4.

(a)

```
In [21]: SMA6 = ma.smaCal(aapl.Close,6)

In [22]: BIAS6 = (aapl.Close-SMA6)/SMA6*100
```

```
In [23]: EMA9 = ma.ewmaCal(aapl.Close,9)

```

```
In [24]: SMA38 = ma.smaCal(aapl.Close,38)

In [25]: BIAS = (EMA9 - SMA38)/SMA38
```

第三十一章　通道突破

(b) (a)

```
In [1]: import pandas as pd
   ...:
   ...: import numpy as np
   ...:
   ...: import matplotlib.pyplot as plt
   ...:
   ...: import candle
   ...: def bbands(tsPrice,period=20,times=2):
   ...:     upBBand=pd.Series(0.0,index=tsPrice.index)
   ...:     midBBand=pd.Series(0.0,index=tsPrice.index)
   ...:     downBBand=pd.Series(0.0,index=tsPrice.index)
   ...:     sigma=pd.Series(0.0,index=tsPrice.index)
   ...:     for i in range(period-1,len(tsPrice)):
   ...:         midBBand[i]=np.nanmean(tsPrice[i-(period-1):(i+1)])
   ...:         sigma[i]=np.nanstd(tsPrice[i-(period-1):(i+1)])
   ...:         upBBand[i]=midBBand[i]+times*sigma[i]
   ...:         downBBand[i]=midBBand[i]-times*sigma[i]
   ...:     BBands=pd.DataFrame({'upBBand':upBBand[(period-1):],\
   ...:                          'midBBand':midBBand[(period-1):],\
   ...:                          'downBBand':downBBand[(period-1):],\
   ...:                          'sigma':sigma[(period-1):]})
   ...:     return(BBands)
   ...:
   ...: Bands)
   ...:

In [2]: GSPC = pd.read_csv('Data/Part5/005/problem31-1.csv',index_col='date'
   ...: )

In [3]: GSPC.index.name='Date'

In [4]: GSPC.index = pd.to_datetime(GSPC.index,format='%Y-%m-%d')
```

```
In [5]: GSPC1 = GSPC[:'2014-02']

In [6]: candle.candleLinePlots(GSPC1,candleTitle='Candle Plot of SPY500',Data
   =GSPC1.Close)
```

```
In [7]: candle.candleLinePlots(GSPC1,candleTitle='Candle Plot of SPY500',
   ...: Data=bbands(GSPC1.Close)['downBBand','upBBand']])
```

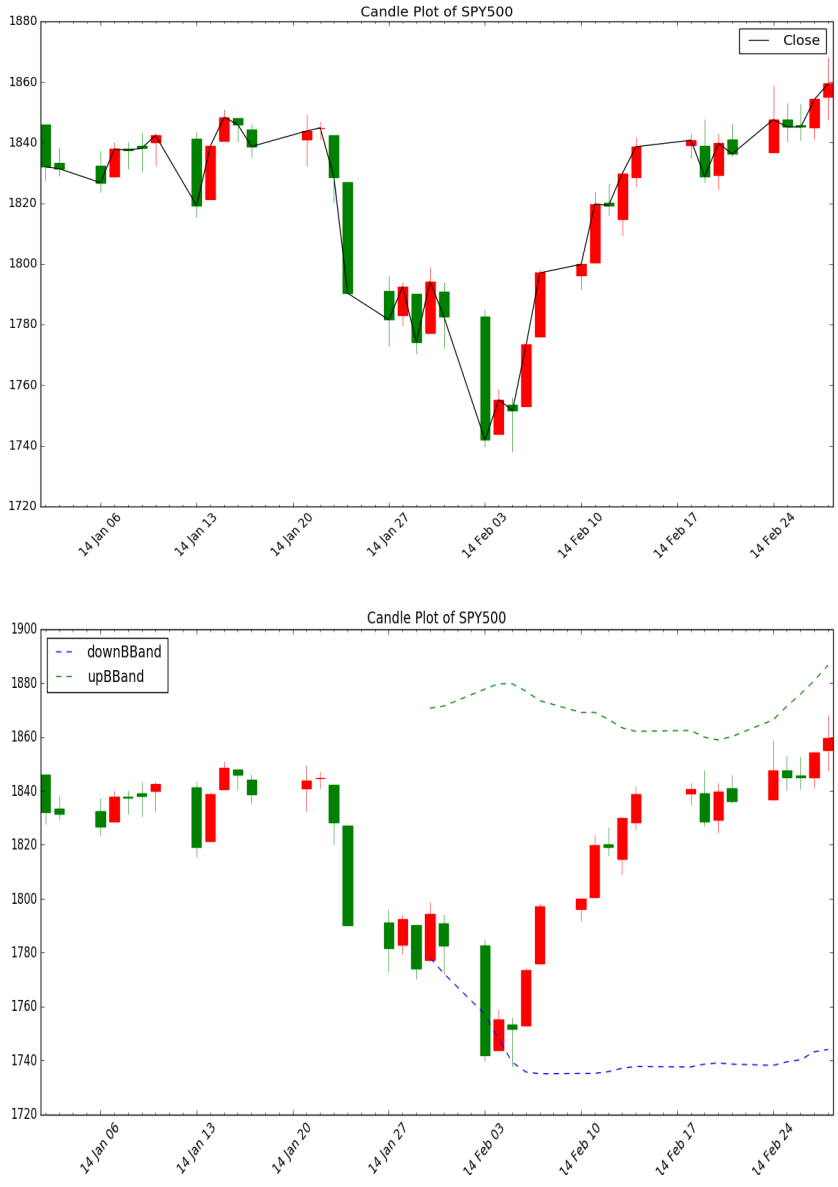


图 31.1: 题 1

收盘价与布林带中轨线具有大致相同的趋势，中轨线会稍微更加平滑一些。而上下轨线之间的距离则受到最近几天收盘价波动幅度的影响。最近几天收盘价的波动幅度越大，上下轨线之间的距离也就越大。

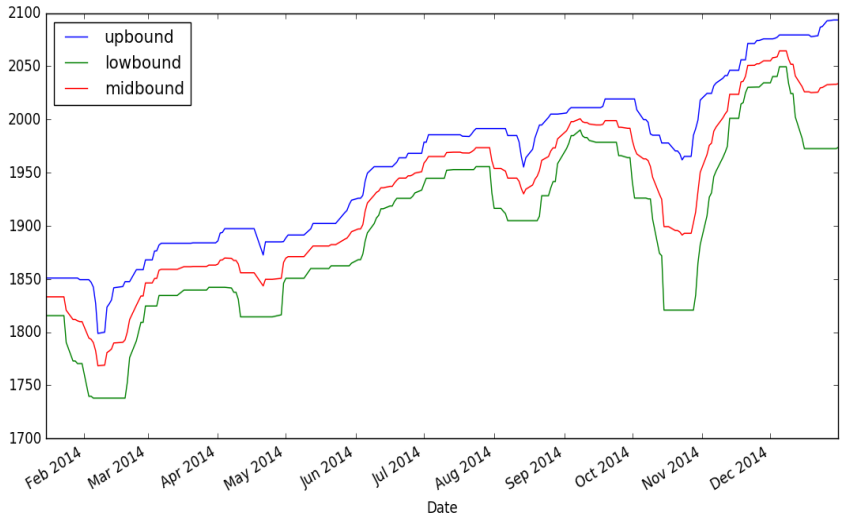
(b)

```
In [8]: upbound = pd.rolling_max(GSPC.High,10)
```

```
In [9]: lowbound = pd.rolling_min(GSPC.Low,10)

In [10]: midbound = (upbound+lowbound)/2

In [11]: bounds = pd.concat([upbound,lowbound,midbound],1)

In [12]: bounds.columns=[ upbound ,' lowbound ,' midbound ]

In [13]: bounds.dropna().plot()
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8465df1550>
```

```
In [14]: std = pd.rolling_std(GSPC.Close,10)

In [15]: midbound = pd.rolling_mean(GSPC.Close,10)

In [16]: upbound = midbound + 1.5 * std

In [17]: lowbound = midbound - 1.5 * std

In [18]: bounds = pd.concat([upbound,lowbound,midbound],1)

In [19]: bounds.columns=[ upbound ,' lowbound ,' midbound ]

In [20]: bounds.dropna().plot()
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8465d9dc50>
```

(d) 两者的原理基本相同，唯一的不同在于所使用的价格数据。布林带使用的是调整过后的价格，而 (d) 问使用的仅仅是收盘价。



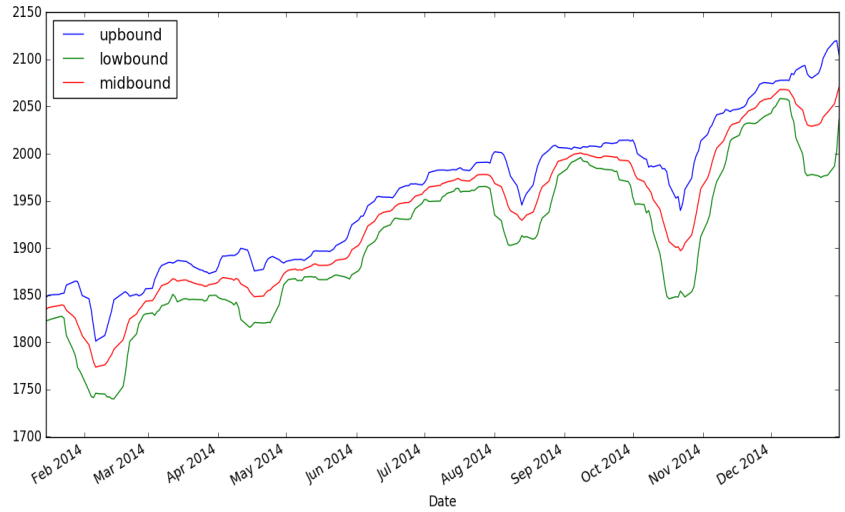


图 31.2: 题 1

2.

(a)

```
In [21]: bounds = bbands(GSPC.Close)

In [22]: bvalue = (GSPC.Close - bounds.downBBand)/(bounds.upBBand-bounds.
    downBBand)

In [23]: bvalue.plot()
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7f84644af080>
```

```
In [24]: bvalue = bvalue.dropna()

In [25]: dat = pd.concat([GSPC.Close,bvalue],1).dropna()
```

```
In [26]: dat.columns = ['bvalue','price']

In [27]: asset = 100000 * np.ones(len(dat))

In [28]: cash = 100000 * np.ones(len(dat))

In [29]: share = np.zeros(len(dat))

In [30]: for i in range(1,len(dat)):
    ...:     cash[i]=cash[i-1]+share[i-1]*dat.price[i]
    ...:     asset[i]=cash[i]
    ...:     if dat.bvalue[i-1]<0:
    ...:         share[i]=100
    ...:         cash[i]=cash[i]-100*dat.price[i]
    ...:     elif dat.bvalue[i-1]>0:
    ...:         share[i]=-100
    ...:         cash[i]=cash[i]+100*dat.price[i]
    ...:
    ...:
```

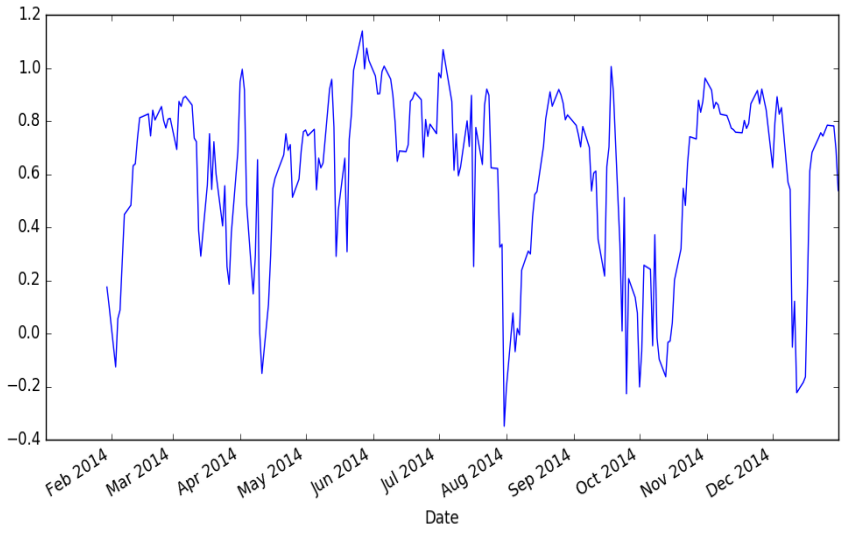


图 31.3: 题 2

(b)

(a)

```
In [31]: BW = (bounds.upBBand - bounds.downBBand)/bounds.midBBand
```

```
In [32]: BW.describe()
Out[32]:
count    233.000000
mean       0.046497
std        0.023566
min        0.012923
25%        0.028057
50%        0.042070
75%        0.061352
max        0.119867
dtype: float64
```

(b) 从布林带的走势上看，标普 500 指数整体上是呈现出上涨的趋势。也就是说，长期上，标普 500 指数仍会上涨。从 BW 的分布上看，在大部分的时间中，BW 值较小，市场较为平稳。但是，BW 的最大值表明仍然存在着一些极端值。结合布林带的图像，我们可以发现这些极端值主要由 10 月份之后的振荡造成。综合来说，长期上，标普 500 指数仍处于上涨趋势中。但是，短期上，标普 500 指数正处于由距离振荡逐渐恢复平稳的过程中，存在着较高的风险。

```
In [33]: midbound = (pd.rolling_mean(GSPC.Close,3) + pd.rolling_mean(GSPC.Close
    ,6)\
    ...:              + pd.rolling_mean(GSPC.Close,12) + pd.rolling_mean(GSPC.Close
        ,24))/4

In [34]: std = pd.rolling_std(midbound,11)

In [35]: upbound = midbound + 6 * std

In [36]: lowbound = midbound - 6 * std

In [37]: bounds = pd.concat([upbound,lowbound,midbound],1)

In [38]: bounds.columns=[ upbound ,' lowbound ,' midbound ]

In [39]: bounds.dropna().plot()
Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8464405860>
```

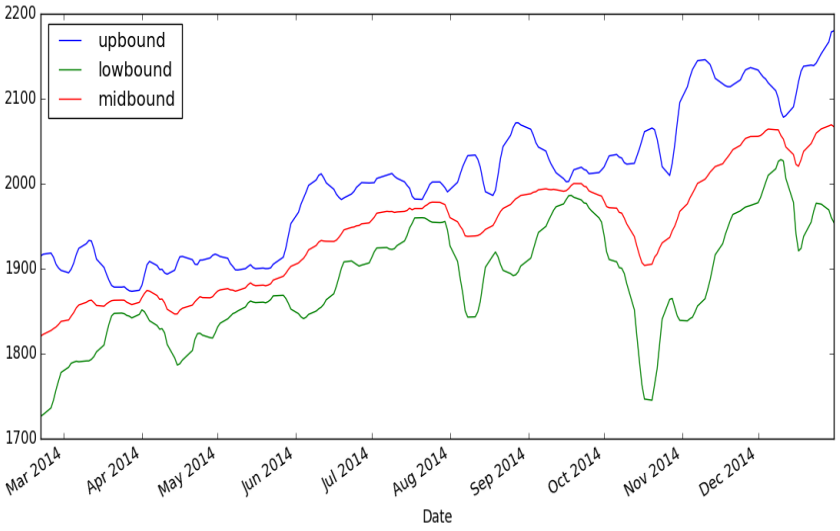

upbound
lowbound
midbound
2200
2100
2000
1900
1800
1700
Mar 2014
Apr 2014
May 2014
Jun 2014
Jul 2014
Aug 2014
Sep 2014
Oct 2014
Nov 2014
Dec 2014
Date

图 31.4: 题 4

第三十二章　KDJ

1. 略

2.

```
In [1]: import pandas as pd
   ...:
   ...: import numpy as np
   ...:
   ...: import movingAverage as ma
   ...:
   ...: import matplotlib.pyplot as plt

In [2]: GSPC = pd.read_csv('Data/Part5/006/problem32-2.csv',index_col='date')

In [3]: GSPC.index.name='Date'

In [4]: GSPC.index = pd.to_datetime(GSPC.index,format='%Y-%m-%d')

In [5]: minValue = GSPC.Low.rolling(9).min()

In [6]: maxValue = GSPC.High.rolling(9).max()

In [7]: RSV = (GSPC.Close - minValue)/(maxValue - minValue) * 100

In [8]: RSV = RSV.dropna()

In [9]: K = ma.ewmaCal(RSV,2,1/3)[1:]

In [10]: D = ma.ewmaCal(K,2,1/3)[1:]

In [11]: J = 3*K - 2*D

In [12]: J = J.dropna()

In [13]: KDJ = pd.concat([K,D,J],1).dropna()
```

```
In [14]: KDJ.columns = ['K','D','J']

In [15]: signal1 = [1 if KDJ.K[i] < 20 else -1 if KDJ.K[i] > 80 else 0 for i in
    range(len(KDJ))]

In [16]: signal2 = [1 if KDJ.D[i] < 20 else -1 if KDJ.D[i] > 80 else 0 for i in
    range(len(KDJ))]

In [17]: signal3 = [1 if KDJ.J[i] < 0 else -1 if KDJ.J[i] > 100 else 0 for i in
    range(len(KDJ))]

In [18]: signal = np.array(signal1) + np.array(signal2) + np.array(signal3)

In [19]: signal = pd.Series(signal, index = KDJ.index)

In [20]: ret = (GSPC.Close - GSPC.Close.shift(1))/GSPC.Close.shift(1)

In [21]: KDJRet = ret * signal.shift(1)

In [22]: KDJRet = KDJRet.dropna()

In [23]: sum(KDJRet>0)/sum(KDJRet!=0)
Out[23]: 0.45977011494252873
```

3.

(a)

```
In [24]: minValue = GSPC.Low.rolling(14).min()

In [25]: maxValue = GSPC.High.rolling(14).max()

In [26]: WR = (maxValue - GSPC.Close) / (maxValue - minValue) * 100

In [27]: WR = WR.dropna()
```

```
In [28]: signal = [1 if WR[i]>80 else -1 if WR[i]<20 else 0 for i in range(
    len(WR))]

In [29]: signal = pd.Series(signal,index = WR.index)

In [30]: WRRet = ret * signal.shift(1)
```

```
6
7 In [31]: WRRet = WRRet.dropna()
8
9 In [32]: sum(WRRet>0)/sum(signal!=0)
10 Out[32]: 0.48148148148148145
```

第三十三章　量价关系

(b)

```
In [1]: import pandas as pd
   ...:
   ...: import numpy as np
   ...:
   ...: import movingAverage as ma
   ...:
   ...: import matplotlib.pyplot as plt
   ...:
   ...: import candle

In [2]: cjzq = pd.read_csv('Data/Part5/007/problem33-1.csv',index_col='date')

In [3]: cjzq.index.name='Date'

In [4]: cjzq.index = pd.to_datetime(cjzq.index,format='%Y-%m-%d')

In [5]: candle.candleVolume(cjzq,candletitle='Candle Plot of Changjiang
   Securities',bartitle='Volume of Changjiang Securities')
```

比如，1 月 27 号到 1 月 30 号出现价涨量增的现象，而之后，股价也连续多日上涨。

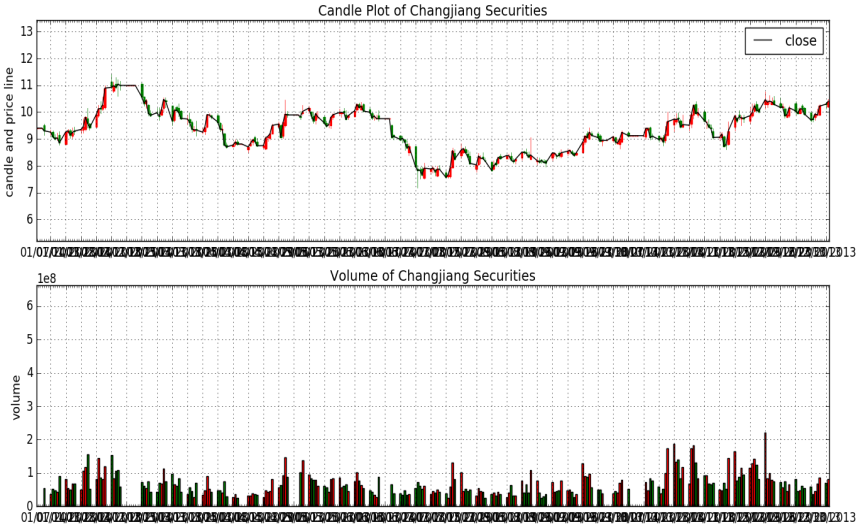


图 33.1: 题 1

2.

```
In [6]: close = cjzq.Close

In [7]: close.describe()
Out[7]:
count     260.000000
mean        9.335538
std         0.784656
min         7.570000
25%         8.802500
50%         9.385000
75%         9.900000
max        11.040000
Name: Close, dtype: float64

In [8]: BreakClose = np.ceil(close)

In [9]: BreakClose.name = 'BreakClose'

In [10]: volume = cjzq.Volume

In [11]: PrcChange = close.diff()

In [12]: UpVol = volume.replace(volume[PrcChange>0],0)

In [13]: UpVol[0] = 0
```

```
In [14]: DownVol = volume.replace(volume[PrcChange<=0],0)

In [15]: DownVol[0] = 0

In [16]: def VOblock(vol):
    ...:     return([np.sum(vol[BreakClose==x]) for x in range(6,12)])
    ...:
    ...:

In [17]: cumUpVol = VOblock(UpVol)

In [18]: cumDownVol = VOblock(DownVol)

In [19]: ALLVol = np.array([cumUpVol,cumDownVol]).transpose()

In [20]: fig,ax=plt.subplots()
    ...:
    ...: ax1=ax.twiny()
    ...:
    ...: ax.plot(close)
    ...:
    ...: ax.set_title('Cumulative volume in different price intervals')
    ...:
    ...: ax.set_ylim(6,12)
    ...:
    ...: ax.set_xlabel('time')
    ...:
   ...: plt.setp(ax.get_xticklabels(), rotation=20,horizontalalignment='center')
    ...:
    ...: ax1.barh(bottom=range(6,12),width=ALLVol[:,0],
    ...:          height=1,color='g',alpha=0.2)
    ...:
   ...: ax1.barh(bottom=range(6,12),width=ALLVol[:,1],height=1,left=ALLVol[:,0],
    ...:          color='r',alpha=0.2)
Out[20]: <Container object of 6 artists>
```

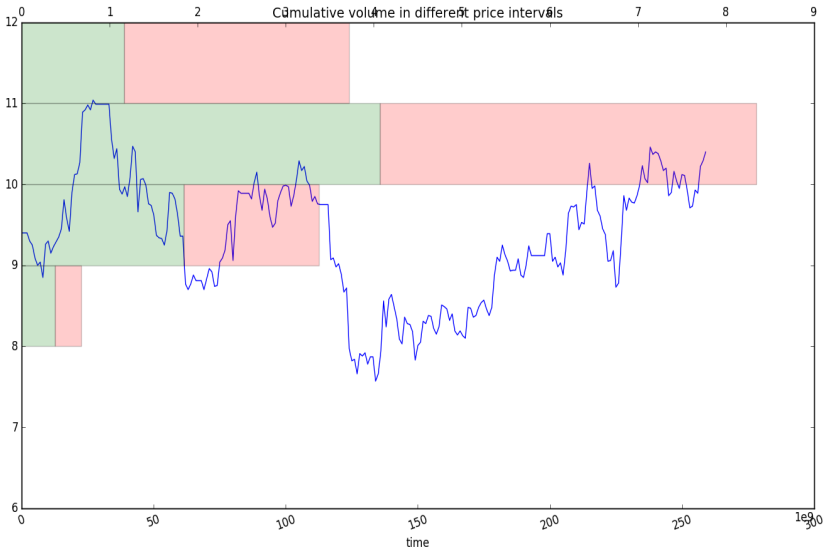


图 33.2: 题 2

3.

```
In [21]: VoMA = np.zeros(len(cjzq))

In [22]: for i in range(5,len(cjzq)):
    ...:     Volume = volume[i-5:i]
    ...:     Close = close[i-5:i]
    ...:     VoMA[i] = sum(Close * Volume / sum(Volume))
    ...:
    ...:

In [23]: VoMA = pd.Series(VoMA,index = cjzq.index).replace(0,np.nan).dropna()

In [24]: VoMA.plot()
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f16bc5ec208>
```

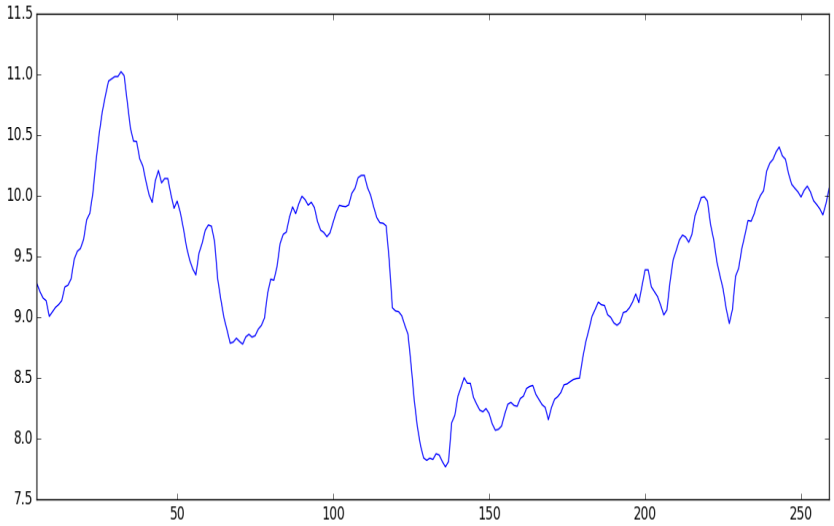


图 33.3: 题 3

4.

```
In [25]: midbound = VoMA

In [26]: upbound = midbound + 1.5 * pd.rolling_std(midbound,6)

In [27]: lowbound = midbound - 1.5 * pd.rolling_std(midbound,6)

In [28]: dat = pd.concat([close,upbound,lowbound],1).dropna()

In [29]: dat.columns = ['price','upbound','lowbound']

In [30]: asset = 2000 * np.ones(len(dat))

In [31]: cash = 2000 * np.ones(len(dat))

In [32]: share = np.zeros(len(dat))

In [33]: for i in range(1,len(dat)):
    ...:     cash[i]=cash[i-1]+share[i-1]*dat.price[i]
    ...:     asset[i]=cash[i]
    ...:     if dat.price[i-1]< dat.upbound[i-1]:
    ...:         share[i]=100
    ...:         cash[i]=cash[i]-100*dat.price[i]
    ...:     elif dat.price[i-1]>dat.lowbound[i-1]:
    ...:         share[i]=-100
    ...:         cash[i]=cash[i]+100*dat.price[i]
```

```
    ...:
    ...:
```

第三十四章　OBV

1.

```
In [1]: import pandas as pd

In [2]: import numpy as np

In [3]: she = pd.read_csv('Data/Part5/008/problem34-1.csv',
   ...:                   index_col='date')

In [4]: she.index.name='Date'

In [5]: she.index = pd.to_datetime(she.index,format='%Y-%m-%d')

In [6]: MendOBV = (she.Volume * ((she.Close-she.Low) - (she.High-she.Close))\
   ...:           /(she.High - she.Low)).cumsum().dropna()

In [7]: signal = pd.Series(np.where(MendOBV.diff()>0 , 1, -1),index = MendOBV.
   index)[1:]

In [8]: ret = (she.Close - she.Close.shift(1))/she.Close.shift(1)

In [9]: OBVret = ret[2:] * signal.shift(1)[1:]
#计算获胜率
In [10]: sum(OBVret>0)/len(signal)
Out[10]: 0.48589341692789967

#模拟资产账户
In [11]: %paste
dat = pd.concat([signal,she.Close],1).dropna()
dat.columns = ['signal','price']

asset = 10000 * np.ones(len(dat))
cash = 10000 * np.ones(len(dat))
share = np.zeros(len(dat))
```

```
for i in range(1,len(dat)):
    cash[i]=cash[i-1]+share[i-1]*dat.price[i]
    asset[i]=cash[i]
    if dat.signal[i-1] == 1:
        share[i]=100
        cash[i]=cash[i]-100*dat.price[i]
    elif dat.signal[i-1] == -1:
        share[i]=-100
        cash[i]=cash[i]+100*dat.price[i]

## -- End pasted text --
```

2.

```
In [12]: %paste
difClose = she.Close.diff()

difClose[0] = 0

volume = she.Volume

volume = volume.replace(0, np.nan).dropna()

difClose = difClose[volume.index]

OBV=(((difClose>=0)*2-1) * she.Volume).cumsum()

dat = pd.concat([OBV,she.Close],1).dropna()

dat.columns = ['OBV','price']

asset = 10000 * np.ones(len(dat))

cash = 10000 * np.ones(len(dat))

share = np.zeros(len(dat))

for i in range(2,len(dat)):
    cash[i]=cash[i-1]+share[i-1]*dat.price[i]
    asset[i]=cash[i]
    if dat.OBV[i-2] < 0 and dat.OBV[i-1] > 0:
        share[i]=100
```

```
        cash[i]=cash[i]-100*dat.price[i]
    elif dat.OBV[i-2] > 0 and dat.OBV[i-1] < 0:
        share[i]=-100
        cash[i]=cash[i]+100*dat.price[i]

## -- End pasted text --
```

3.

```
In [13]: %paste
wanke = pd.read_csv('Data/Part5/008/problem34-3.csv',
                    index_col='date')

wanke.index.name='Date'

wanke.index = pd.to_datetime(wanke.index,format='%Y-%m-%d')

difClose = wanke.Close.diff()

difClose[0] = 0

volume = wanke.Volume

volume = volume.replace(0, np.nan).dropna()

difClose = difClose[volume.index]

OBV=(((difClose>=0)*2-1) * wanke.Volume).cumsum()

smOBV = pd.rolling_mean(OBV,12)

## -- End pasted text --
```