python 程式設計

第 十二 講

NumPy 科學運算套件 (二)

結構陣列 (一)

- 陣列元素有個別型別與名稱，如同 C 的 struct 一樣。若不設名稱，則預設為 f0,f1, . . . 。
- 結構陣列元素的名稱與型別可使用 dtype 取得
- 取得結構陣列元素個別名稱所有資料

```
>>> a = np.zeros(3,"float, int")
>>> a
array([(0.0, 0), (0.0, 0), (0.0, 0)],
      dtype=[('f0', '<f8'), ('f1', '<i8')])

>>> a.dtype
dtype([('f0', '<f8'), ('f1', '<i8')])

>>> a['f1']
array([0, 0, 0])
```

以上的 '<f8' 代表為浮點數佔用 8 位元組，使用 little-endian 方式儲存。若是 '>f8' 則是使用 big-endian。

結構陣列 (二)

- 可先設定各個元素的 **dtype** ， 再定義陣列

```
>>> dt = np.dtype( [( 'x' , 'i') , ( 'y' , 'f8' )])
>>> a = np.ones(3,dt)
>>> a
array([(1, 1.0), (1, 1.0), (1, 1.0)],
      dtype=[('x', '<i4'), ('y', '<f8')])

>>> a['x']
array([1, 1, 1], dtype=int32)
```

- 物件名稱字典可直接取用或設定資料

```
>>> dt = np.dtype( [( 'x' , 'f8') , ( 'y' , 'f8' )])
>>> a = np.empty(11,dt)
>>> a['x'] = np.linspace(0,2.5,len(a))
>>> a['y'] = a['x']**2
>>> a
array([(0.0, 0.0), (0.25, 0.0625), (0.5, 0.25), (0.75, 0.5625),
       (1.0, 1.0),(1.25, 1.5625), (1.5, 2.25), (1.75, 3.0625),
       (2.0, 4.0),(2.25, 5.0625), (2.5, 6.25)],
      dtype=[('x', '<f8'), ('y', '<f8')])
```

讀寫檔案 (一)

- **np.loadtxt(fname,...)：由 fname 檔案讀入資料**

參數	作用	範例
fname	檔案名稱	
dtype	各筆資料的型別	dtype='S8,f8' 兩筆資料存入串列 dtype=np.dtype([('a','S8'),('b','f8')]) 以上定義字典索引 'a' 與 'b'
comments	註解字元	comments='%' 跳過註解列不讀，預設為 #
delimiter	資料間的分隔字元	delimiter=','
converter	字典由行下標映射到函式，可由字串欄位轉為需要的資料型別	
skiprows	在讀取資料前，先跳過若干列，預設為 0	skiprows=n 由第 n+1 列讀取
usecols	常串列儲存直行下標，預設為讀取全部的列數	usecols=(1,3) 僅讀第 2 行與第 4 行
unpack	若為 True，檔案內直向欄位會統一存到不同變數。若不設定，檔案資料是以列方式儲存各行資料成一個兩維陣列。	

讀寫檔案 (二)

假設為累積雨量檔案，每列的第一個欄位為時間，之後十個欄位為不同測站所量測到以兩小時為單位的累積兩量 (mm)。

```
# AM
2017-03-31:02  0  0 0 0 0 0  0 0 0 0
2017-03-31:04  0  0 0 0 0 0  0 0 0 0
2017-03-31:06  0  0 0 0 0 0  0 0 0 0
2017-03-31:08  0  0 0 0 0 0  0 0 0 0
2017-03-31:10  0  0 0 0 0 0  0 0 0 0
2017-03-31:12  0  0 0 0 0 0  0 0 0 0
# PM
2017-03-31:14  1  6 3 0 0 1  0 0 0 0
2017-03-31:16  3  2 0 0 0 0  0 0 0 0
2017-03-31:18  0  0 0 0 0 0  0 0 0 2
2017-03-31:20 15 10 8 6 5 8  6 4 3 4
2017-03-31:22  4  3 5 7 7 6 10 6 2 5
2017-03-31:24  8  6 5 6 5 6  8 7 4 4
```

讀寫檔案 (三)

- 讀入檔案資料成二維陣列：各直列存入字典，預設索引為 **f0, f1, f2, ...**

```
>>> dts = "S13" + ",f8" * 10
>>> a = np.loadtxt("rainfall2.dat",dts)
>>> a
a
array([(b'2017-03-31:02', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
       (b'2017-03-31:04', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
       (b'2017-03-31:06', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
       (b'2017-03-31:08', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
       (b'2017-03-31:10', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
       (b'2017-03-31:12', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
       (b'2017-03-31:14', 1.0, 6.0, 3.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0),
       (b'2017-03-31:16', 3.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
       (b'2017-03-31:18', 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0),
       (b'2017-03-31:20', 15.0, 10.0, 8.0, 6.0, 5.0, 8.0, 6.0, 4.0, 3.0, 4.0),
       (b'2017-03-31:22', 4.0, 3.0, 5.0, 7.0, 7.0, 6.0, 10.0, 6.0, 2.0, 5.0),
       (b'2017-03-31:24', 8.0, 6.0, 5.0, 6.0, 5.0, 6.0, 8.0, 7.0, 4.0, 4.0)],
      dtype=[('f0', 'S13'), ('f1', '<f8'), ('f2', '<f8'), ('f3', '<f8'),
             ('f4', '<f8'), ('f5', '<f8'), ('f6', '<f8'), ('f7', '<f8'),
             ('f8', '<f8'), ('f9', '<f8'), ('f10', '<f8')])
```

讀寫檔案 (四)

```
>>> a['f0']
array([b'2017-03-31:02', b'2017-03-31:04', b'2017-03-31:06',
       b'2017-03-31:08', b'2017-03-31:10', b'2017-03-31:12',
       b'2017-03-31:14', b'2017-03-31:16', b'2017-03-31:18',
       b'2017-03-31:20', b'2017-03-31:22', b'2017-03-31:24'],
      dtype='|S13')

>>> a['f1']
array([  0.,   0.,   0.,   0.,   0.,   0.,
         1.,   3.,   0.,  15.,   4.,   8.])

>>> sum(a['f1'])
31.0

>>> raining = a['f1'] > 0
>>> raining
array([False, False, False, False, False, False,  True,  True, False,  True,
       True,  True], dtype=bool)

>>> a['f1'][raining]                                    # 取出有第一個測站有下雨的雨量
array([  1.,   3.,  15.,   4.,   8.])

>>> a['f0'][raining]                                    # 取出有第一個測站有下雨的時段
array([b'2017-03-31:14', b'2017-03-31:16', b'2017-03-31:20',
       b'2017-03-31:22', b'2017-03-31:24'],
      dtype='|S13')
```

讀寫檔案 (五)

- 僅取出前兩個測站的時雨量:

```
>>> a , b = np.loadtxt("rainfall2.dat",usecols=(1,2),unpack=True)
>>> a
array([  0.,  0.,  0.,  0.,  0.,  0.,
         1.,  3.,  0., 15.,  4.,  8.])
>>> b
array([  0.,  0.,  0.,  0.,  0.,  0.,
         6.,  2.,  0., 10.,  3.,  6.])
```

讀寫檔案 (六)

- 若要加取出日期時間字串，須個別設定取出欄位的資料型別:

```
>>> dts = "S13,f8"
>>> dhr , b = np.loadtxt("rainfall2.dat",usecols=(0,2),
                         dtype=dts,unpack=True)

>>> dhr
array([b'2017-03-31:02', b'2017-03-31:04', b'2017-03-31:06',
       b'2017-03-31:08', b'2017-03-31:10', b'2017-03-31:12',
       b'2017-03-31:14', b'2017-03-31:16', b'2017-03-31:18',
       b'2017-03-31:20', b'2017-03-31:22', b'2017-03-31:24'],
      dtype='|S13')

>>> b
array([  0.,  0.,  0.,  0.,  0.,  0.,
         6.,  2.,  0., 10.,  3.,  6.])
```

以上 dhr 所取得陣列元素為 bytes 型別，字元資料不可變更。

讀寫檔案 (七)

- 設定欄位索引取出資料

```
>>> dts = np.dtype([('dtime','S13'),('rstation1','f8')])
>>> a = np.loadtxt("rainfall2.dat",usecols=(0,1),dtype=dts)

>>> a['dtime']
array([b'2017-03-31:02', b'2017-03-31:04', b'2017-03-31:06',
       b'2017-03-31:08', b'2017-03-31:10', b'2017-03-31:12',
       b'2017-03-31:14', b'2017-03-31:16', b'2017-03-31:18',
       b'2017-03-31:20', b'2017-03-31:22', b'2017-03-31:24'],
      dtype='|S13')

>>> a['rstation1']
array([  0.,  0.,  0.,  0.,  0.,  0.,
         1.,  3.,  0., 15.,  4.,  8.])
```

排序 (一)

- **foo.sort()**：對最後分量（最裡層中括號）排序
  - 一維陣列

```
>>> a = np.array([1,3,5])
>>> a.sort()
>>> a
array([1, 3, 5])
```

  - 多維陣列

```
>>> a = np.array( [[3,2,7], [4,1,3]] )
>>> a.sort()
>>> a
array([[2, 3, 7],
       [1, 3, 4]])

>>> b = np.array( [[[3,2,4],[9,3,1]],[[5,2,6],[1,4,2]] ] )
>>> b.sort()
>>> b
array([[[2, 3, 4],
        [1, 3, 9]],

       [[2, 5, 6],
        [1, 2, 4]]])
```

排序 (二)

- **foo.sort(axis=n):**
  在其他分量下標位置相同下，對第 **n** 個分量排序

```
>>> a = np.array( [[3,2,7], [4,1,3]] )
>>> a.sort(axis=0)                          # 垂直方向排序
>>> a
array([[3, 1, 3],
       [4, 2, 7]])
```

❖ **三維或以上維度的陣列，對非最後分量的排序變得不直覺。**

排序 (三)

```
>>> b = np.array( [[[3,1,4],[2,5,9]],[[9,0,8],[7,3,6]]])
>>> b
array([[[3, 1, 4],
        [2, 5, 9]],

       [[9, 0, 8],
        [7, 3, 6]]])

>>> b.sort(axis=0)
>>> b
array([[[3, 0, 4],
        [2, 3, 6]],

       [[9, 1, 8],
        [7, 5, 9]]])
```

以上的 b 元素下標依次序分別為:

```
  3 1 4       b[0,0,0]  b[0,0,1]  b[0,0,2]
  2 5 9 --->  b[0,1,0]  b[0,1,1]  b[0,1,2]
  9 0 8       b[1,0,0]  b[1,0,1]  b[1,0,2]
  7 3 6       b[1,1,0]  b[1,1,1]  b[1,1,2]
```

對第一個分量排序，即是在其餘分量下標位置不變下，調整順序。因此，3 與 9 比大小， 1 與 0 比大小，4 與 8 比大小， 2 與 7 比大小，5 與 3 比大小，9 與 6 比大小。同樣分析方式也適用在對第二個分量數字排序。

排序 (四)

- **np.sort(foo,axis=n):**
  根據第 **n** 個分量排序，回傳排序後的陣列，原始 **foo** 陣列不變。
  若不設定 **axis** 則對最後分量排序。

```
>>> a = np.array( [[3,1,4],[8,5,7]] )
>>> np.sort(a)                          # 送出排序後的結果
array([[1, 3, 4],
       [5, 7, 8]])
>>> a                                   # 原陣列不變
array([[3, 1, 4],
       [8, 5, 7]])
```

- **np.argsort(foo):** 回傳排序後的下標位置陣列，**foo** 陣列不變

```
>>> a = np.array([8,5,9,0])
>>> np.argsort(a)
array([3, 1, 0, 2])
>>> a[np.argsort(a)]                    # 透過排序後的下標位置得到排序後陣列
array([0, 5, 8, 9])
>>> a                                   # 原陣列不變
array([8, 5, 9, 0])
```

排序 (五)

- **np.searchsorted(foo,a):**
  在已排序 foo 陣列內找尋大於等於 a 且最接近 a 的下標位置
- **np.searchsorted(foo,b):** 同上, 但 b 為串列

```
>>> a = np.array([1,2,4,6])
>>> np.searchsorted(3)                    # a[2] >= 3
2

>>> np.searchsorted(a,[3,7,4,0])
array([2, 4, 2, 0])
```

❖ 若輸入的數比 foo 陣列內所有的數都大, 則回傳 len(foo)

邏輯運算與比較 (一)

- **np.all(foo): 檢查 foo 陣列的每個元素是否都為 True**
- **np.any(foo): 檢查 foo 陣列中有個元素為 True**

```
>>> np.all( np.array([2,0,1]) )
False

>>> np.any( np.array([0,0,3]) )
True
```

- **np.isreal(foo): 檢查 foo 陣列的元素是否為實數**
- **np.iscomplex(foo): 檢查 foo 陣列的元素是否為複數**

```
>>> np.isreal( np.array( [1,2+j,3] ) )
False

>>> np.iscomplex( np.array( [1,2+1j,3] ) )
True
```

❖ **複數的虛數部份如果為 1，仍須寫上 1，不得省略。**

邏輯運算與比較 (二)

- 數值上相等:

  兩數 a 與 b 為數值上相等如果 $|a-b| \leq (atol + rtol \times |b|)$

  atol 預設為 10^{-8} , rtol 為 10^{-5} 。

- np.allclose(foo,bar): foo 與 bar 陣列所有對應位置元素是否數值上相等
- np.isclose(foo,bar): foo 與 bar 陣列所有對應位置元素是否數值上相等

邏輯運算與比較 (三)

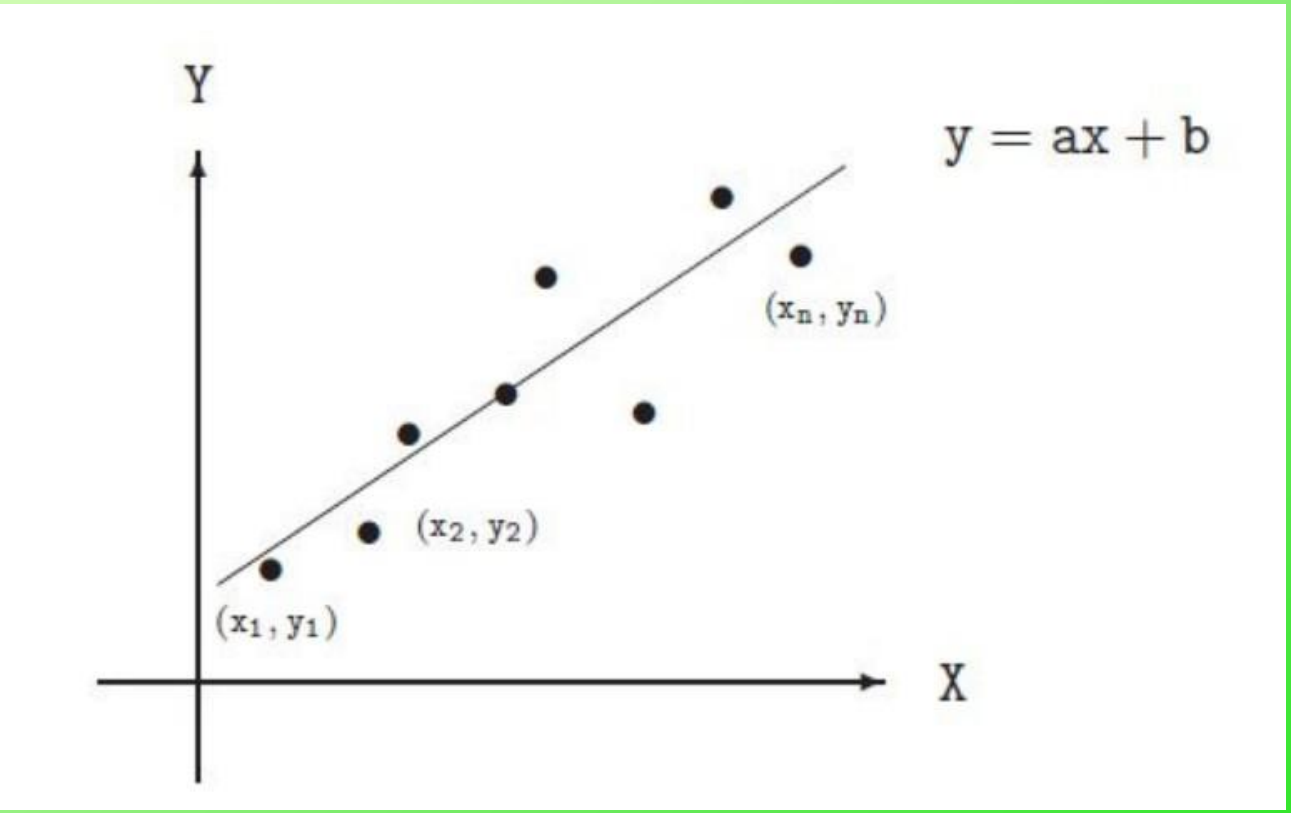
```
>>> a = np.array( [1.e10, 2e-8] )
>>> b = np.array( [1.0001e10, 1e-8] )

>>> np.isclose( a , b )
array([False, True], dtype=bool)

>>> np.allclose( a , b )
False

>>> c = np.array( [1.000001e10, 1e-8] )
>>> np.allclose(a,c)
True

>>> np.isclose(a,c,rtol=1e-8)                    # 重新設定 rtol 數值
array([False, True], dtype=bool)
```

❖ 如果兩數字為 np.nan，則預設數值上不相等，但可設定 equal_nan=True 使其相等

```
>>> a = np.array( [np.nan,10] )
>>> b = np.array( [np.nan,15] )

>>> np.isclose(a,b)
array([False, False], dtype=bool)

>>> np.isclose(a,b,equal_nan=True)
array([ True, False], dtype=bool)
```

迴歸直線：最小平方法 (一)

- 最小平方法是統計學上分析數據經常用到的方法，其目的是要在平面空間上找尋某函數使得此函數最足以代表平面上所有的資料點。在實際的應用上，函數通常為直線函數或者為指數函數，以下我們以直線來說明：



迴歸直線：最小平方法 (二)

假設平面上已知有 n 個資料點 (x_i, y_i)，欲求的直線為 $y(x) = ax + b$，則此直線的係數 a 與 b 要讓所有資料點在 x_i 的 y_i 值與在直線上的 $y(x_i)$ 值差距的平方和為最小，此直線被稱為迴歸直線 (regression line)。若以數學式子表示，相當於要找出直線的 a 與 b 兩係數使得 $\sum_{i=1}^{n}(y_i - y(x_i))^2$ 為最小，這裡的 $y(x_i)$ 為直線在 $x = x_i$ 的 y 值，也就是 $ax_i + b$。利用簡單的微分求極值觀念，分別對 a 與 b 微分即可導出以下兩個包含係數 a 與 b 的聯立方程式：

$$a\sum_{i=1}^{n} x_i + bn = \sum_{i=1}^{n} y_i$$

$$a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i$$

在以下程式我們利用函式設定 n 個傾斜並呈現帶狀分佈座標點，之後迴歸直線的運算很簡單。本程式所用到的 randint、sin、 cos 等函式都是在 numpy 套件內。

迴歸直線: 最小平方法 (三)

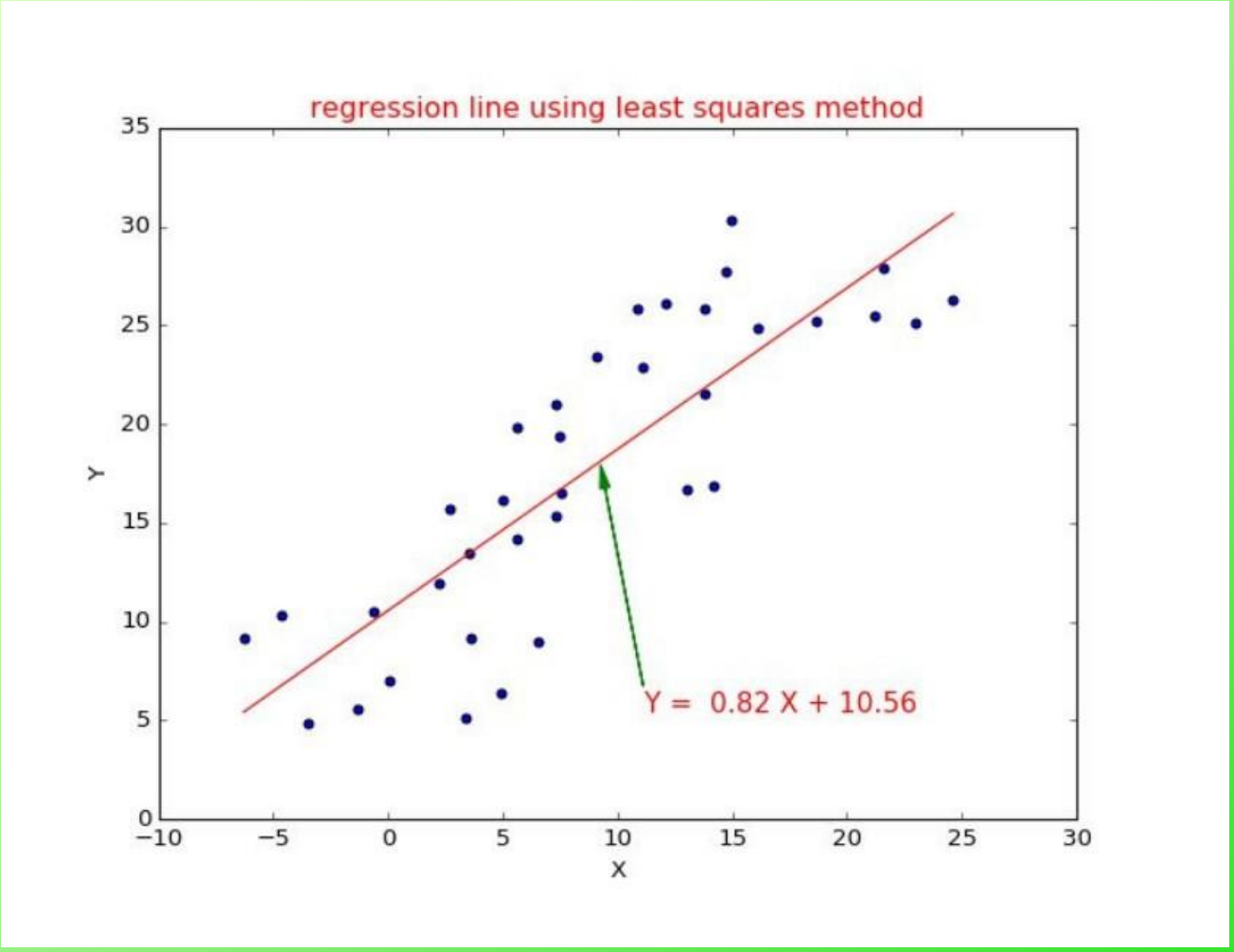
```
import pylab
import numpy as np

def main() :

    # 資料點數量
    n = np.random.randint(30,50)

    # 產生呈現傾斜帶狀的 n 個數據
    xs , ys = beltdata(n)

    pylab.figure(facecolor='white')

    c11 = sum(xs)
    c12 = n
    c13 = sum(ys)

    c21 = sum(xs**2)
    c22 = c11
    c23 = sum(xs*ys)

    det  = c11*c22 - c21*c12
    det1 = c13*c22 - c23*c12
    det2 = c11*c23 - c21*c13

    a = det1/det
    b = det2/det

    x1 , x2 = min(xs) , max(xs)
    y1 , y2 = a*x1+b , a*x2+b

    # 畫數據點
    pylab.scatter(xs,ys)

    # 畫迴歸線
    pylab.plot( (x1,x2) , (y1,y2) , color='red' )

    # 標示 X Y 與 圖案標題
    pylab.xlabel("X")
    pylab.ylabel("Y")
    pylab.title("regression line using least
                squares method",color='red')

    line = "Y = {:>5.2f} X + {:>5.2f}".format(a,b)
    x3 = (x1+x2)/2
    y3 = a*x3+b
```

迴歸直線: 最小平方法 (四)

```
    # 標示迴歸線圖示
    pylab.annotate(line, xy=(x3+0.1,y3-0.1), fontsize=14, color='red',
                   arrowprops=dict(color='green',width=0.1,headwidth=5),
                   xytext=(x3+2,y1))

    pylab.show()

# 產生 n 個帶狀分佈點, 並使之傾斜
def  beltdata(n) :

    # xs: [1,2,...,n]
    xs = np.linspace(1,n,n)

    c = np.random.randint(0,2)

    # ys: [0,1] 之間浮點數, 共有 n 個點
    ys = np.array( [ np.random.randint(0,100)/10 + c for i in range(n) ] )

    # 旋轉角度
    ang = np.pi/np.random.randint(3,10)

    # 旋轉帶狀的點
    xs , ys = np.cos(ang)*xs - np.sin(ang)*ys , np.sin(ang)*xs + np.cos(ang)*ys

    return xs , ys

main()
```

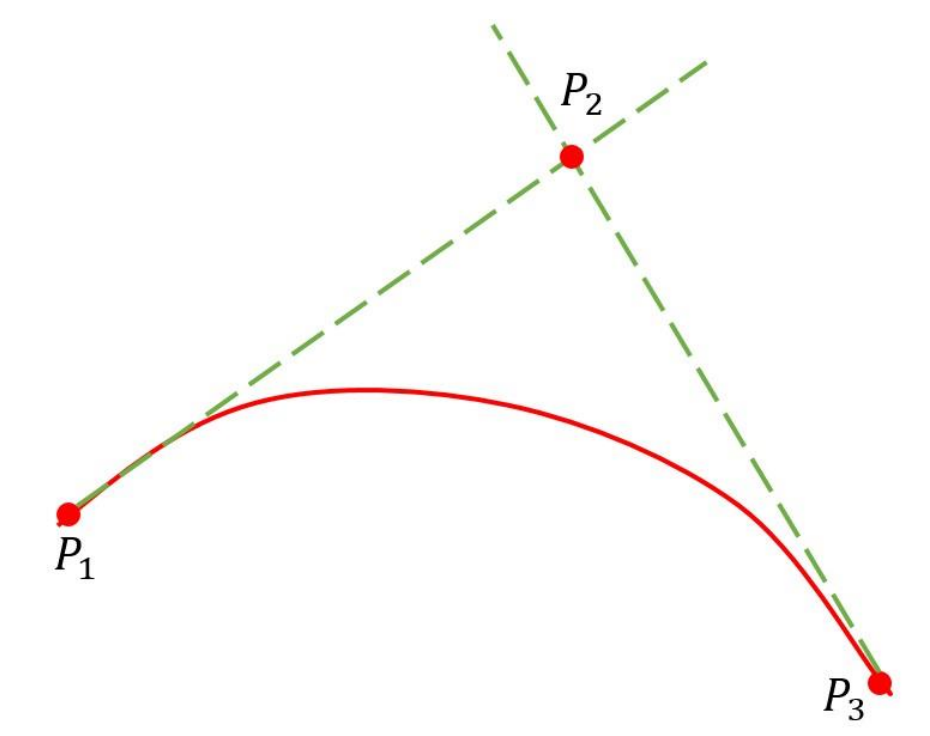
迴歸直線：最小平方法 (五)

輸出的圖形為:



Bezier 曲線 (一)

曲線可用來描繪各種幾何形狀，比如常見的動畫、字型、各式幾何圖形設計都是由曲線組成的。曲線可用數學方程式描述，其中又以 **Bezier** 曲線是最常見的曲線，下圖是由三個點所組成的二次 **Bezier** 曲線，其中，P_1 與 P_3 都是在曲線上，P_2 是由穿過 P_1 與 P_2 兩切線的交點。



Bezier 曲線 (二)

曲線的方程式為

$$P(t) = (1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3 \qquad t \in [0,1]$$

P_0、P_1、P_2、P_3 三個座標點決定了，變更不同的 t，即能算出二次 Bezier 曲線的所有的點，因此這三個點也就是整條曲線的控制點。一條複雜的幾何線條可能由許多二次 Bezier 曲線所組成，每條二次 Bezier 曲線包含三點，首尾兩點在曲線上，中間點為切線交點，通常不在曲線上。如果 P_0、P_1、P_2、P_3、P_4、P_5、P_6、P_7、P_8、P_9、...為了描述圖形的所有點，那麼 P_0、P_1、P_2 為第一段 Bezier 曲線，P_2、P_3、P_4 為第二段 Bezier 曲線，P_4、P_5、P_6為第三段，其它依此類推。

Bezier 曲線 (三)

- 以下座標點是描述米老鼠的頭部圖形的二次 **Bizier** 曲線點:

```
54.0 62.5
59.5 63.0
64.0 60.5
63.0 65.5
64.0 68.5
64.5 76.0
68.0 78.5
70.0 82.5
78.0 82.5
85.5 83.5
90.0 80.5
95.0 77.5
98.0 74.5
103.0 69.5
103.5 64.5
104.0 55.5
102.0 52.5
```

```
99.5 48.0
95.0 45.5
89.0 43.0
83.0 44.0
85.0 38.5
85.0 32.5
85.0 27.5
84.0 22.5
82.0 15.5
79.5 13.5
75.0 7.5
72.0 5.5
65.0 1.0
62.0 1.0
58.5 0.0
54.0 0.0
```

```
49.5 0.0
46.0 1.0
43.0 1.0
36.0 5.5
33.0 7.5
28.5 13.5
26.0 15.5
24.0 22.5
23.0 27.5
23.0 32.5
23.0 38.5
25.0 44.0
19.0 43.0
13.0 45.5
8.5 48.0
6.0 52.5
```

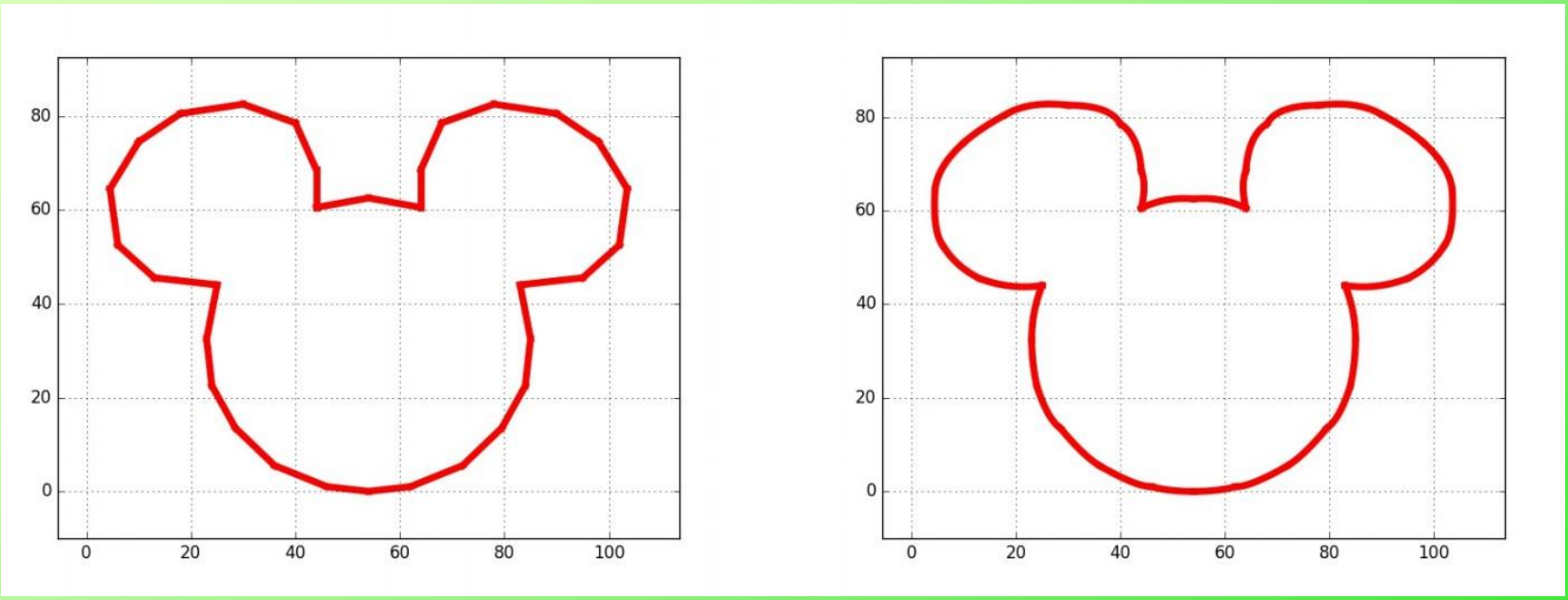
```
4.0 55.5
4.5 64.5
5.0 69.5
10.0 74.5
13.0 77.5
18.0 80.5
22.5 83.5
30.0 82.5
38.0 82.5
40.0 78.5
43.5 76.0
44.0 68.5
45.0 65.5
44.0 60.5
48.5 63.0
54.0 62.5
```

Bezier 曲線 (四)

```
import numpy as np
import pylab

# 設定每個欄位的資料名稱與型別
dtypes = np.dtype( [('x','f8'),
                    ('y','f8')] )

# 讀檔
pts = np.loadtxt("micky.dat",dtype=dtypes)
npts = len(pts)

# 取出 x 與 y 座標
ptx , pty = pts['x'] , pts['y']

# 每段曲線共有 n 個點與設定參數 ts
n = 21
ts = np.linspace(0,1,n)

# 設定圖形周邊區域為 白色
pylab.figure(facecolor='white')

xmin , xmax = 1000 , 0
ymin , ymax = 1000 , 0

# 每段包含三個點
for i in range(0,npts-1,2) :

    # 二次 Bezier curve: 使用向量式運算
    bxs = (1-ts)**2 * ptx[i] + 2*ts*(1-ts)
          * ptx[i+1] + ts**2 * ptx[i+2]
    bys = (1-ts)**2 * pty[i] + 2*ts*(1-ts)
          * pty[i+1] + ts**2 * pty[i+2]

    x1 , y1 = bxs.min() , bys.min()
    x2 , y2 = bxs.max() , bys.max()

    if x1 < xmin : xmin = x1
    if y1 < ymin : ymin = y1
    if x2 > xmax : xmax = x2
    if y2 > ymax : ymax = y2

    # 讓線寬為 5 點
    pylab.plot(bxs,bys,'r',linewidth=5)

ds = 10

# 顯示格線
pylab.grid()

# 設定 x 軸與 y 軸的顯示範圍
pylab.xlim(xmin-ds,xmax+ds)
pylab.ylim(ymin-ds,ymax+ds)

# 產生 jpg 圖檔
pylab.savefig('micky.jpg')

pylab.show()
```

Bezier 曲線 (五)

- 圖左是直接連接米老鼠頭部上的端點所構成的圖形，圖右是使用二次 **Bezier** 曲線所產生的圖形。



在實際應用上，每一段 **Bezier** 曲線的控制點通常是利用滑鼠/觸控筆直接在螢幕上決定，一旦設定了控制點，此條 **Bezier** 曲線也可立即看到，整個圖案就是由許多段曲線連接在一起，如此設計圖案就變得是尋找各段曲線控制點的位置。