國立雲林科技大學機械工程系

機器學習課堂報告

Linear Algebra

學生：機械四B B10711150 葉書廷

指導教授：吳英正 教授

日期：民國111年 3月 30日

**目的：**

先前實做Curve Fitting時我們曾使用Polyfit的方式直接解出方程式之係數，而本次將使用線性代數中的矩陣運算解出其係數，並比較兩者差異。

**方法：**

一般來說，以方程式為例，在已知(x, y)的情況下要解出a、b，需要至少兩個(x, y)才能解出唯一解，同理，方程式需要至少三組(x, y)才能解出未知數a、b、c，而解聯立方程式的方式其中一種就是利用反矩陣：

🡺 ……..….(1)

X ˙ p = y

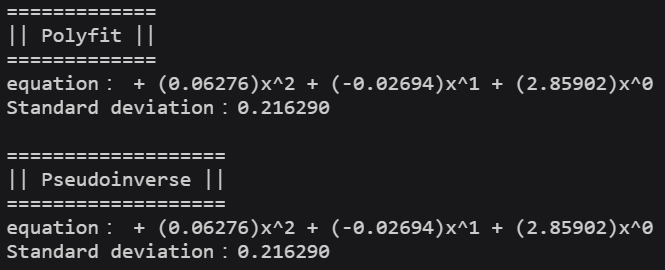
🡺 ……………………………………(2)

透過式子(2)，我們可以利用反矩陣解出各項係數，但是，反矩陣僅在方陣中存在，也就是說，若今天有40行聯立方程式，僅要求出二次方程式的三個係數，我們無法從一個40x3的矩陣中求出反矩陣以及各項係數的唯一解，此時需要利用一個方法求出近似的反矩陣，擬反矩陣(Pseudoinverse Matrix)，利用此方式也能求出方程式各項係數的最佳解，其定義為：

將其代回(2)中取代的位子，即能求出方程式各項係數之最佳解。

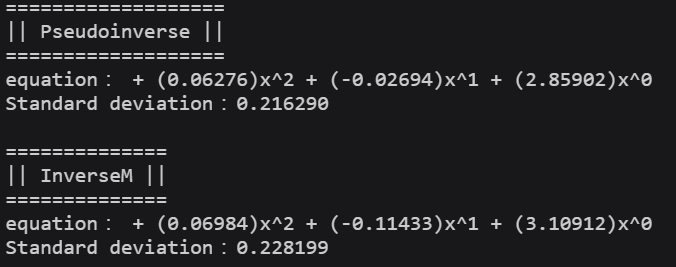
**結果：**

從(圖一)可以看出，使用Polyfit與Pseudoinverse兩種方式計算出的方程式以及標準差一模一樣，事實上在小數點後14、15位的位置開始才有一點差別，但這種微小差距基本上可以忽略。

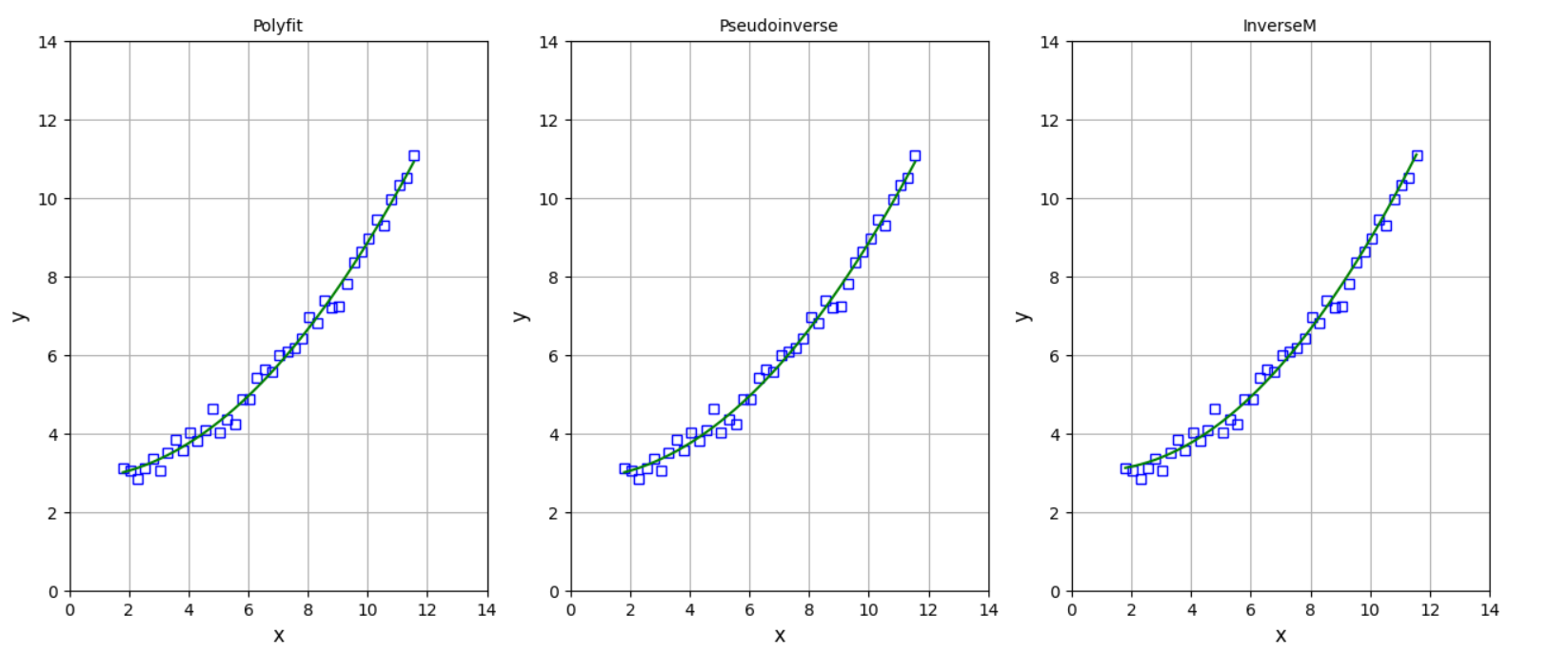


(圖一)、Polyfit與Pseudoinverse兩種輸出結果

若是假設(x, y)資料只有三筆，分別是原資料中的第1筆、第21筆、以及第40筆，利用三筆資料求三個係數項，將X塑型成方陣，就能夠使用反矩陣的方式處理，而其結果如(圖二)、(圖三)，可以看出其方程式係數、標準差以及圖形趨勢都與Pseudoinverse的結果相像，但因其資料只取最初最後以及中間的三筆(經測試這三筆具有最好的標準差)，而畫出來的曲線是與原資料(40筆)比對，所以標準差表現較差。



(圖二)、Pseudoinverse與inverseMatrix兩種輸出結果



(圖二)、三種方式圖形比較

**程式撰寫：**

import numpy as np

import matplotlib.pyplot as plt

def read\_data(datafile\_name):

# read from txt

data = np.loadtxt('python\\Deep\_Learning\\DL\_IngJeng\\' +

datafile\_name+'.csv',delimiter=',') #from vscode root directory

n = len(data[:, 0])

x = data[:, 0]

y = data[:, 1]

return n, x, y

def Polyfit(n\_poly, n, x, y):

return np.polyfit(x, y, n\_poly)

def Pseudoinverse(n\_poly, n, x, y):

a = np.ones([n\_poly + 1, n])

for i in range(n\_poly):

a[i] = x \*\* (n\_poly-i)

p = np.zeros([1, n\_poly + 1])

X = np.transpose(a)

Y = np.transpose(y)

p = np.transpose(p)

XT = np.transpose(X)

XXTinv = np.linalg.inv(np.dot(XT, X))

p = np.dot(np.dot(XXTinv, XT), Y)

return p

def InverseM(n\_poly, n, x, y):

a = np.ones([n\_poly + 1, n])

for i in range(n\_poly):

a[i] = x \*\* (n\_poly-i)

x = [a.T[0], a.T[20], a.T[39]]

y = [y[0], y[20], y[39]]

p = np.zeros([1, n\_poly + 1])

p = np.dot(np.linalg.inv(x), y)

return p

def main():

Solution = {}

Solution['Polyfit'], Solution['Pseudoinverse'], Solution['InverseM'] = {}, {}, {}

n\_poly = 2 #determine the n degree polynomial equation

datafile\_name = 'data\_of\_poly2' #input("Enter the data file name:")

n, x, y= read\_data(datafile\_name)

Solution['Polyfit']['para'] = Polyfit(n\_poly, n, x, y)

Solution['Pseudoinverse']['para'] = Pseudoinverse(n\_poly, n, x, y)

Solution['InverseM']['para'] = InverseM(n\_poly, n, x, y)

i=1

plt.figure(figsize=(18, 6))

for key in Solution.keys():

Solution[key]['y\_ex'] = 0

Solution[key]['y\_eq'] = ''

for j in range(0,n\_poly+1):

Solution[key]['y\_ex'] += Solution[key]['para'][j] \* (x\*\*(n\_poly-j))

Solution[key]['y\_eq'] += '+ (' + str(round(Solution[key]['para'][j], 5)) + ')x^' + str(n\_poly - j) + ' '

sy2 = np.sum((Solution[key]['y\_ex']-y)\*\*2)/(n-(n\_poly+1))

Solution[key]['sy'] = sy2\*\*0.5

print('','='\*(len(key)+6), '\n ||', key, '||\n','='\*(len(key)+6))

print(' equation：', Solution[key]['y\_eq'])

print(' Standard deviation：%f\n'%Solution[key]['sy'])

plt.subplot(int('13'+str(i)))

plt.xlabel('x', fontsize=12), plt.ylabel('y', fontsize=12)

plt.xlim(0, 14), plt.ylim(0, 14)

plt.title(key, fontsize=10)

plt.grid(True, which='both')

plt.plot(x, y, 'bs', markerfacecolor='none')

plt.plot(x, Solution[key]['y\_ex'], 'g')

i+=1

plt.show()

main()