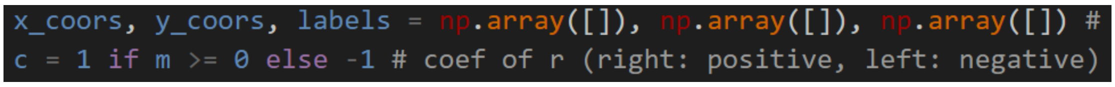
Q1.

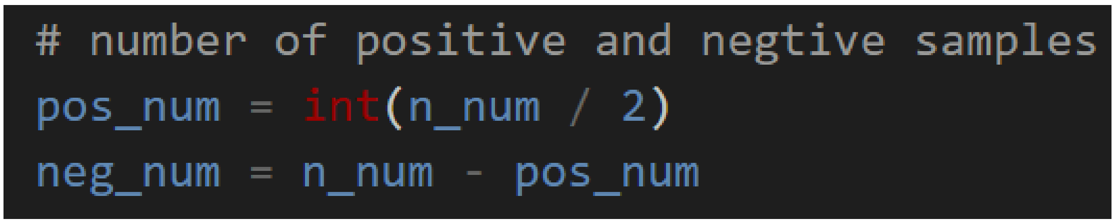
Execution description:

變數宣告、讓直線的右邊為正樣本，左邊為負樣本(不論直線斜率 m 為何)



```
x_coors, y_coors, labels = np.array([]), np.array([]), np.array([]) #
c = 1 if m >= 0 else -1 # coef of r (right: positive, left: negative)
```

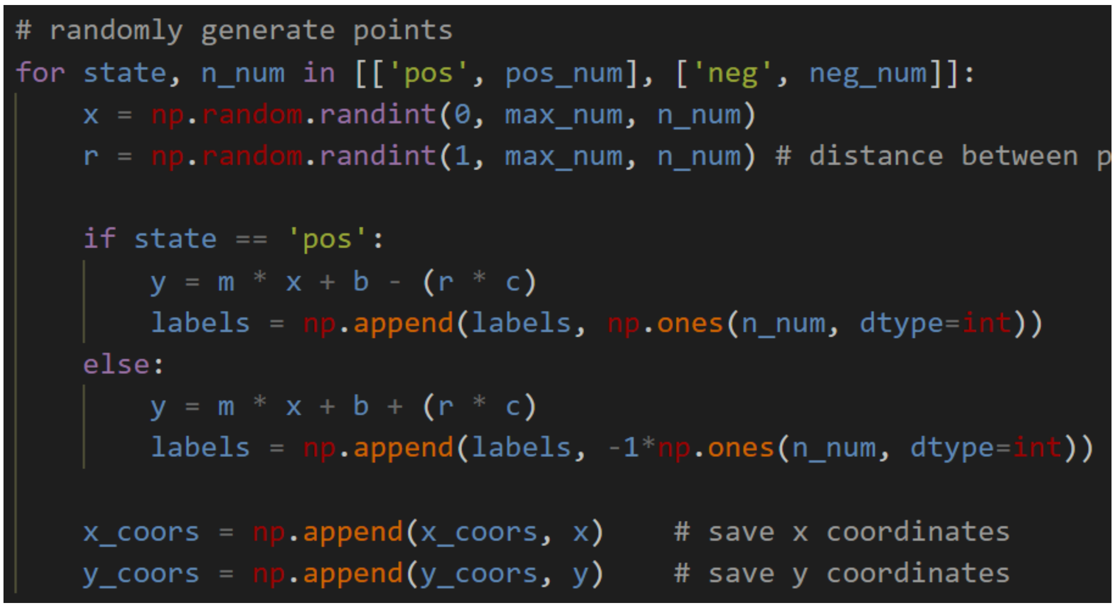
正樣本與負樣本的數量各一半:



```
# number of positive and negtive samples
pos_num = int(n_num / 2)
neg_num = n_num - pos_num
```

製造隨機樣本，因為樣本不能在線上，所以上下平移隨機距離 r，同時正樣本

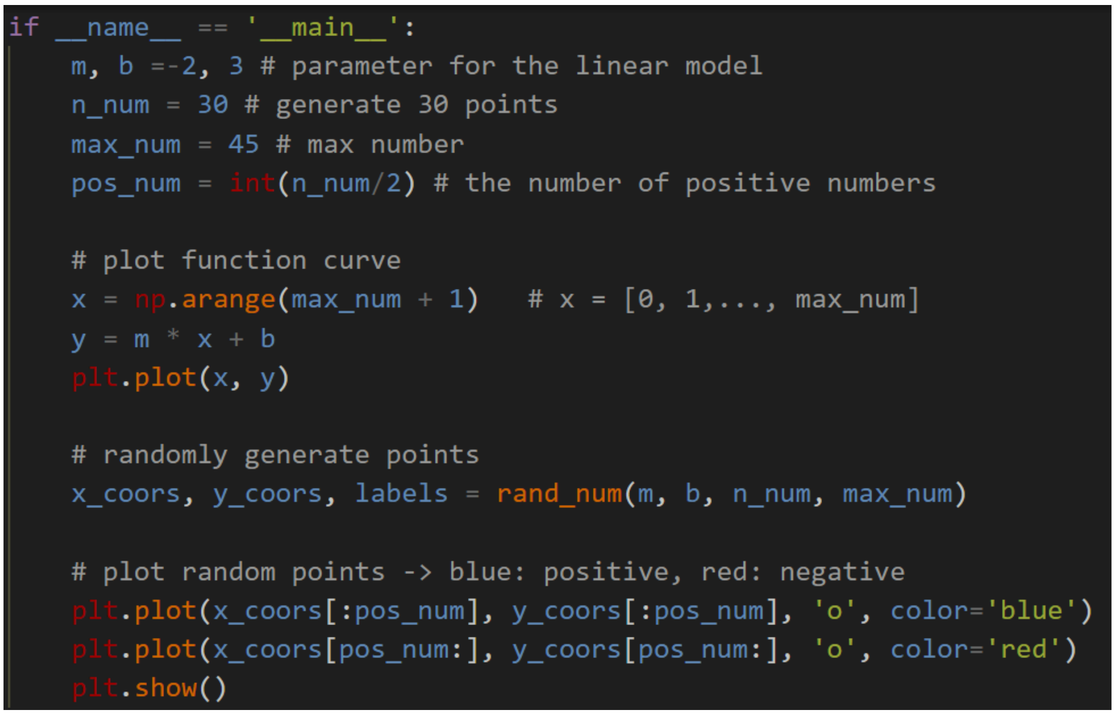
label 標 1、負樣本 label 標 -1



```
# randomly generate points
for state, n_num in [['pos', pos_num], ['neg', neg_num]]:
    x = np.random.randint(0, max_num, n_num)
    r = np.random.randint(1, max_num, n_num) # distance between p

    if state == 'pos':
        y = m * x + b - (r * c)
        labels = np.append(labels, np.ones(n_num, dtype=int))
    else:
        y = m * x + b + (r * c)
        labels = np.append(labels, -1*np.ones(n_num, dtype=int))

    x_coors = np.append(x_coors, x)    # save x coordinates
    y_coors = np.append(y_coors, y)    # save y coordinates
```

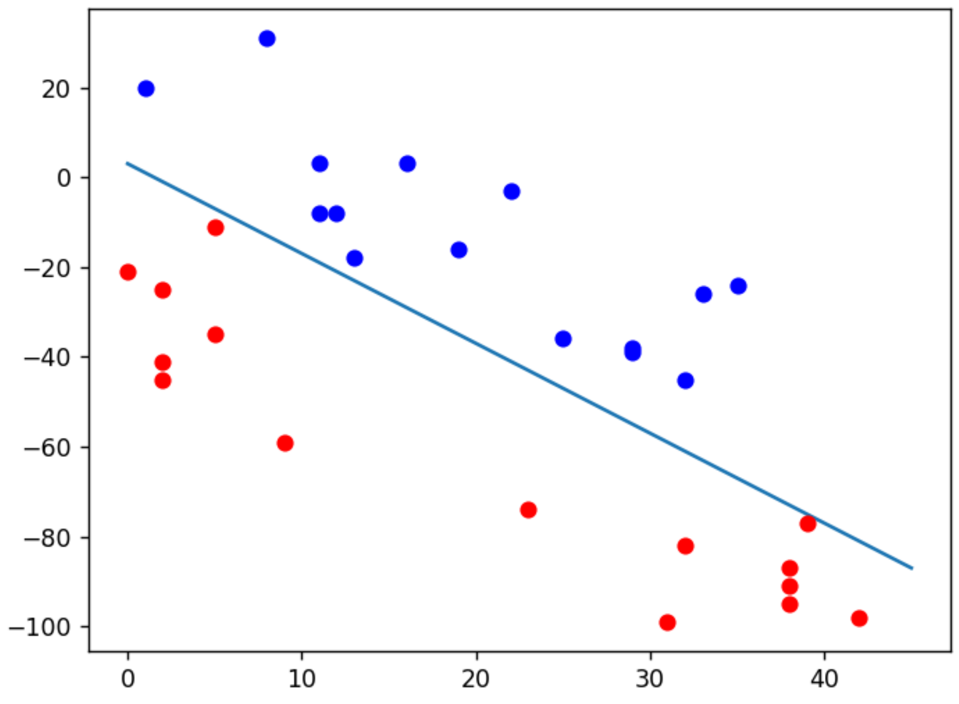
Main function 設定參數(這裡使用 m=-2，b=3,，樣本數 30，最大數為 45)

呼叫函式製作樣本點後畫圖(樣本點與直線)



```
if __name__ == '__main__':
    m, b =-2, 3 # parameter for the linear model
    n_num = 30 # generate 30 points
    max_num = 45 # max number
    pos_num = int(n_num/2) # the number of positive numbers

    # plot function curve
    x = np.arange(max_num + 1)   # x = [0, 1,..., max_num]
    y = m * x + b
    plt.plot(x, y)

    # randomly generate points
    x_coors, y_coors, labels = rand_num(m, b, n_num, max_num)

    # plot random points -> blue: positive, red: negative
    plt.plot(x_coors[:pos_num], y_coors[:pos_num], 'o', color='blue')
    plt.plot(x_coors[pos_num:], y_coors[pos_num:], 'o', color='red')
    plt.show()
```

Experimental results:



Conclusion:

樣本點完美的分佈在直線兩側

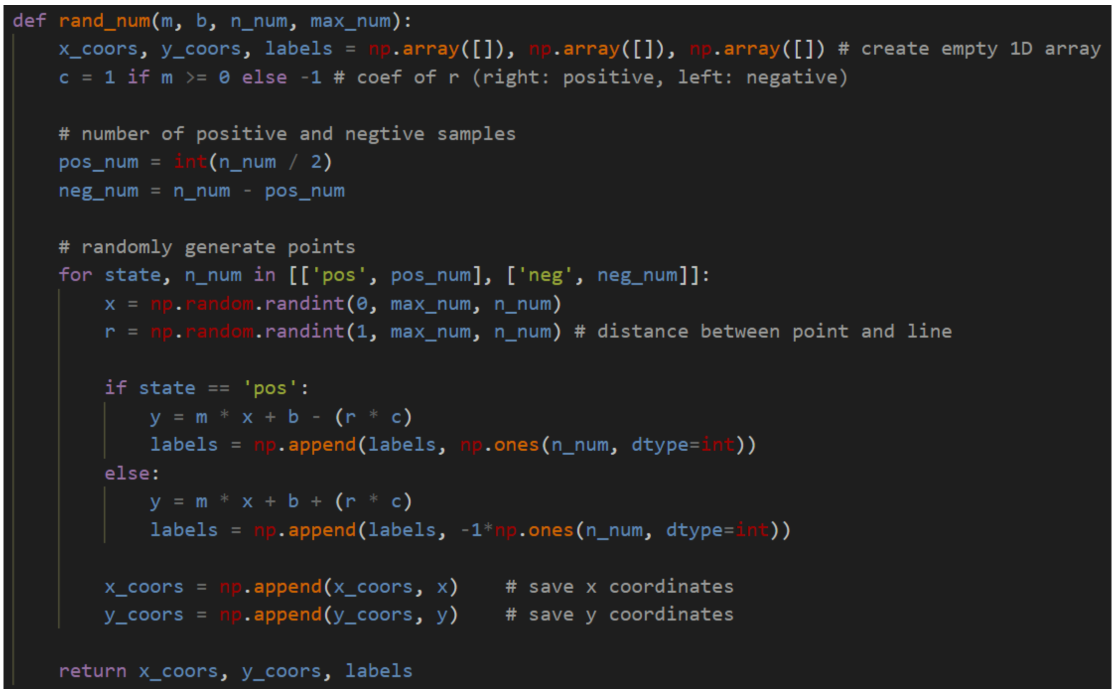
Discussion:

想不太出來比 sample code 更好的方法來製作題目要求的樣本，所以就稍微更改參數而已

Q2.

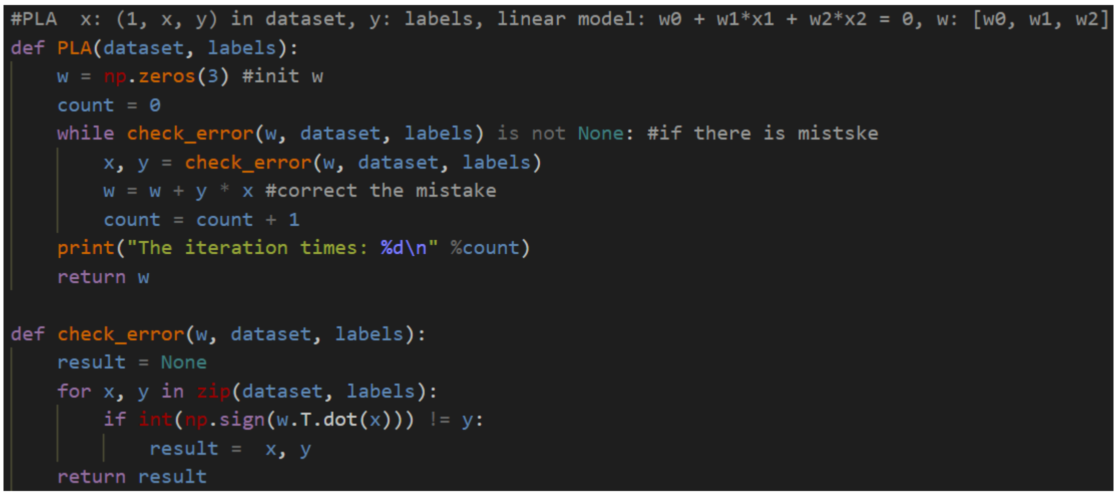
Execution description:

和 Q1.相同的函式



```python
def rand_num(m, b, n_num, max_num):
    x_coors, y_coors, labels = np.array([]), np.array([]), np.array([]) # create empty 1D array
    c = 1 if m >= 0 else -1 # coef of r (right: positive, left: negative)

    # number of positive and negtive samples
    pos_num = int(n_num / 2)
    neg_num = n_num - pos_num

    # randomly generate points
    for state, n_num in [['pos', pos_num], ['neg', neg_num]]:
        x = np.random.randint(0, max_num, n_num)
        r = np.random.randint(1, max_num, n_num) # distance between point and line

        if state == 'pos':
            y = m * x + b - (r * c)
            labels = np.append(labels, np.ones(n_num, dtype=int))
        else:
            y = m * x + b + (r * c)
            labels = np.append(labels, -1*np.ones(n_num, dtype=int))

        x_coors = np.append(x_coors, x)    # save x coordinates
        y_coors = np.append(y_coors, y)    # save y coordinates

    return x_coors, y_coors, labels
```

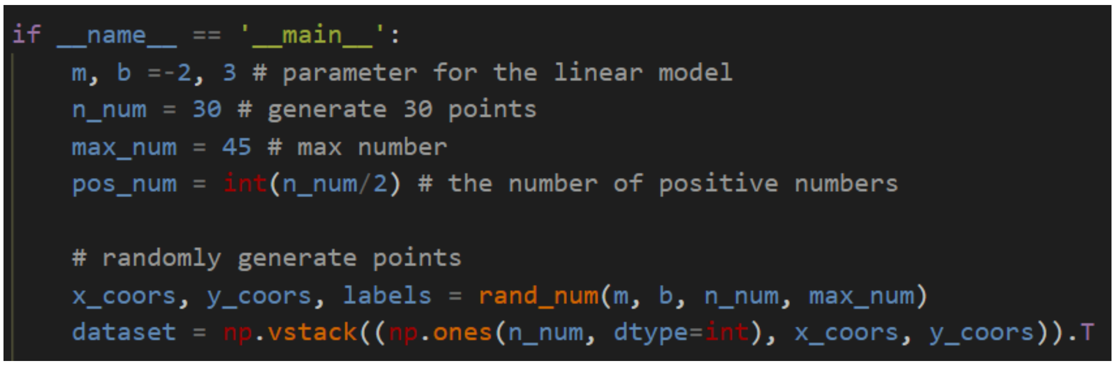
PLA 實作，w0 為[0,0,0]，一直執行到沒有錯誤為止



```python
#PLA  x: (1, x, y) in dataset, y: labels, linear model: w0 + w1*x1 + w2*x2 = 0, w: [w0, w1, w2]
def PLA(dataset, labels):
    w = np.zeros(3) #init w
    count = 0
    while check_error(w, dataset, labels) is not None: #if there is mistske
        x, y = check_error(w, dataset, labels)
        w = w + y * x #correct the mistake
        count = count + 1
    print("The iteration times: %d\n" %count)
    return w

def check_error(w, dataset, labels):
    result = None
    for x, y in zip(dataset, labels):
        if int(np.sign(w.T.dot(x))) != y:
            result =  x, y
    return result
```

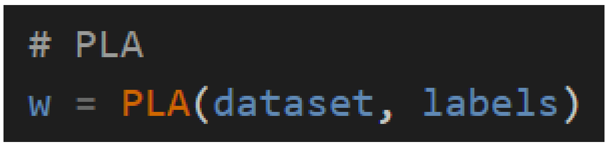
Main function:

產生隨機資料點



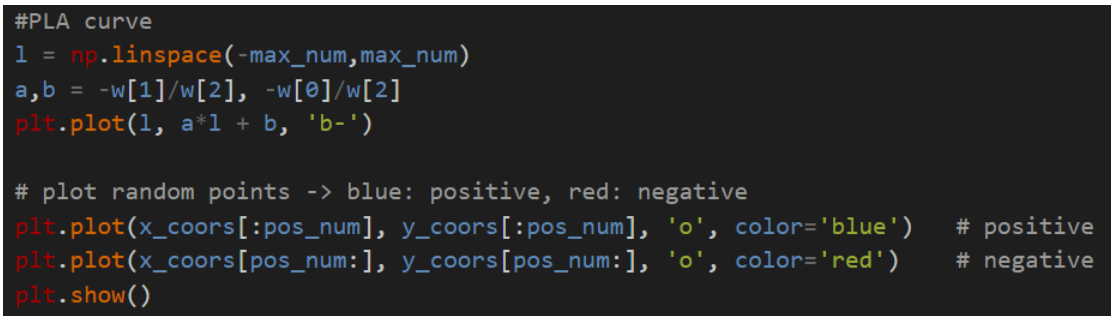
```python
if __name__ == '__main__':
    m, b =-2, 3 # parameter for the linear model
    n_num = 30 # generate 30 points
    max_num = 45 # max number
    pos_num = int(n_num/2) # the number of positive numbers

    # randomly generate points
    x_coors, y_coors, labels = rand_num(m, b, n_num, max_num)
    dataset = np.vstack((np.ones(n_num, dtype=int), x_coors, y_coors)).T
```

執行 PLA，取得 w



```python
# PLA
w = PLA(dataset, labels)
```

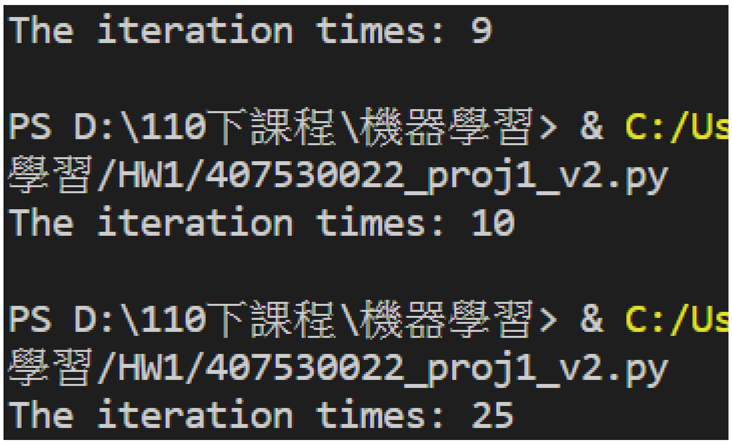
畫圖



```python
#PLA curve
l = np.linspace(-max_num,max_num)
a,b = -w[1]/w[2], -w[0]/w[2]
plt.plot(l, a*l + b, 'b-')

# plot random points -> blue: positive, red: negative
plt.plot(x_coors[:pos_num], y_coors[:pos_num], 'o', color='blue')   # positive
plt.plot(x_coors[pos_num:], y_coors[pos_num:], 'o', color='red')    # negative
plt.show()
```

Experimental results:

會執行到 Wt 沒有錯誤為止



```
The iteration times: 9

PS D:\110下課程\機器學習> & C:/Us
學習/HW1/407530022_proj1_v2.py
The iteration times: 10

PS D:\110下課程\機器學習> & C:/Us
學習/HW1/407530022_proj1_v2.py
The iteration times: 25
```

平均迭代次數為 (9 + 10 +25)/3 = 14.67

Conclusion:

PLA 在執行線性可分割的資料點時很快

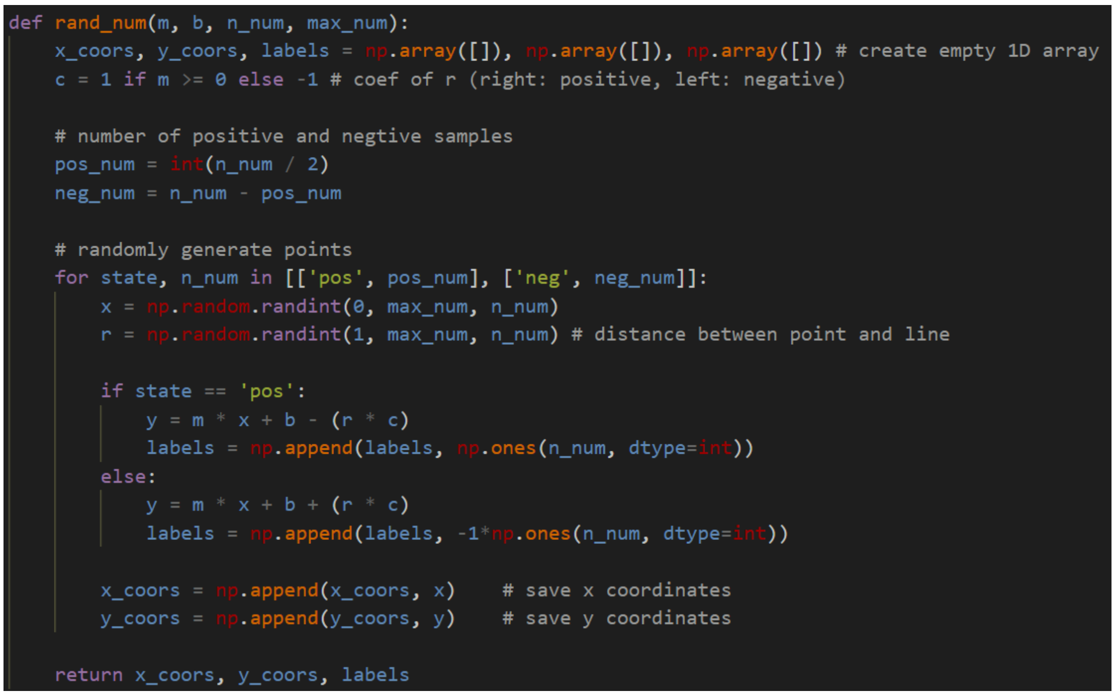
Discussion:

看懂 PLA 演算法的步驟花了點時間，不太懂 sign 函式的功用

Q3.

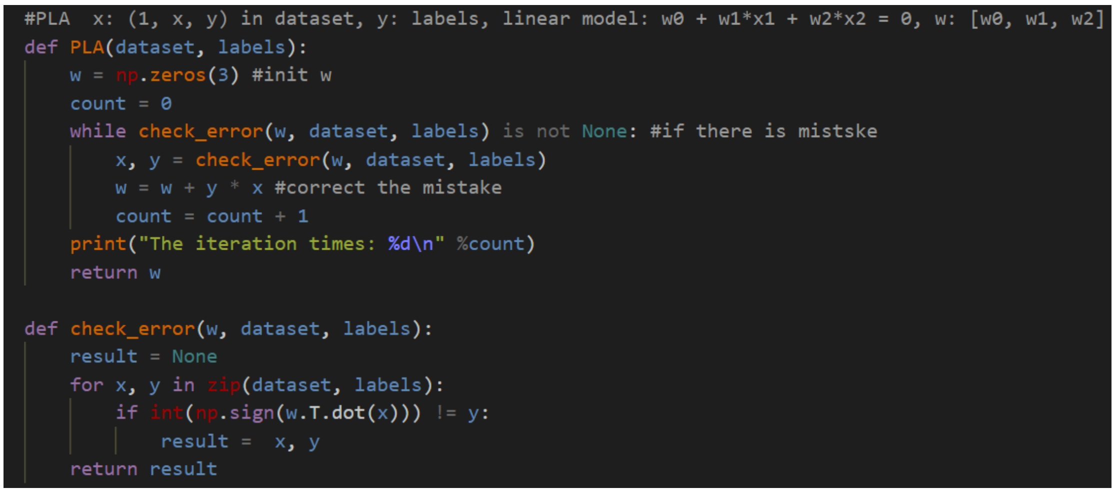
Execution:

同 Q1 的函式



```python
def rand_num(m, b, n_num, max_num):
    x_coors, y_coors, labels = np.array([]), np.array([]), np.array([]) # create empty 1D array
    c = 1 if m >= 0 else -1 # coef of r (right: positive, left: negative)

    # number of positive and negtive samples
    pos_num = int(n_num / 2)
    neg_num = n_num - pos_num

    # randomly generate points
    for state, n_num in [['pos', pos_num], ['neg', neg_num]]:
        x = np.random.randint(0, max_num, n_num)
        r = np.random.randint(1, max_num, n_num) # distance between point and line

        if state == 'pos':
            y = m * x + b - (r * c)
            labels = np.append(labels, np.ones(n_num, dtype=int))
        else:
            y = m * x + b + (r * c)
            labels = np.append(labels, -1*np.ones(n_num, dtype=int))

        x_coors = np.append(x_coors, x)    # save x coordinates
        y_coors = np.append(y_coors, y)    # save y coordinates

    return x_coors, y_coors, labels
```

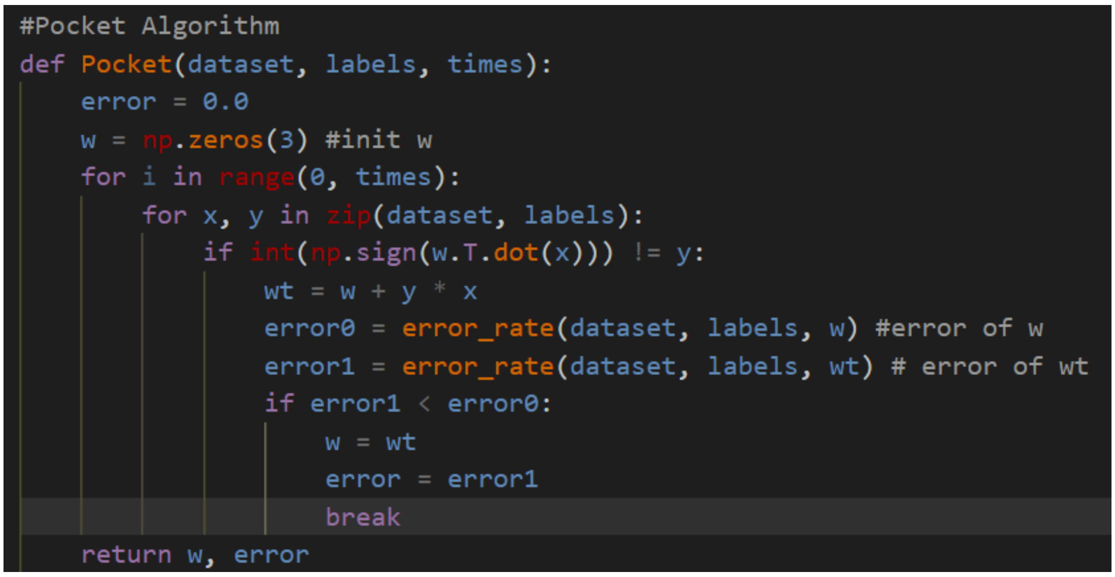
同 Q2 的 PLA 實作



```python
#PLA  x: (1, x, y) in dataset, y: labels, linear model: w0 + w1*x1 + w2*x2 = 0, w: [w0, w1, w2]
def PLA(dataset, labels):
    w = np.zeros(3) #init w
    count = 0
    while check_error(w, dataset, labels) is not None: #if there is mistske
        x, y = check_error(w, dataset, labels)
        w = w + y * x #correct the mistake
        count = count + 1
    print("The iteration times: %d\n" %count)
    return w

def check_error(w, dataset, labels):
    result = None
    for x, y in zip(dataset, labels):
        if int(np.sign(w.T.dot(x))) != y:
            result =  x, y
    return result
```

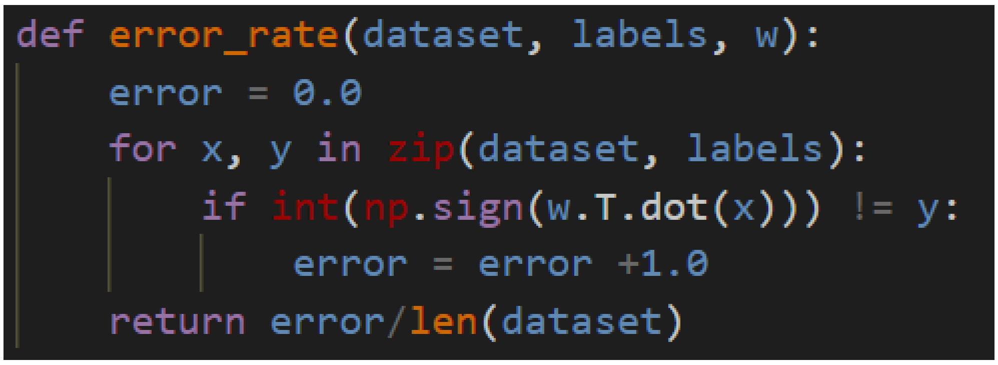
Pocket 演算法實作，計算出 Wt 後，如果錯誤率較原本低就更新，沒有就不更

新



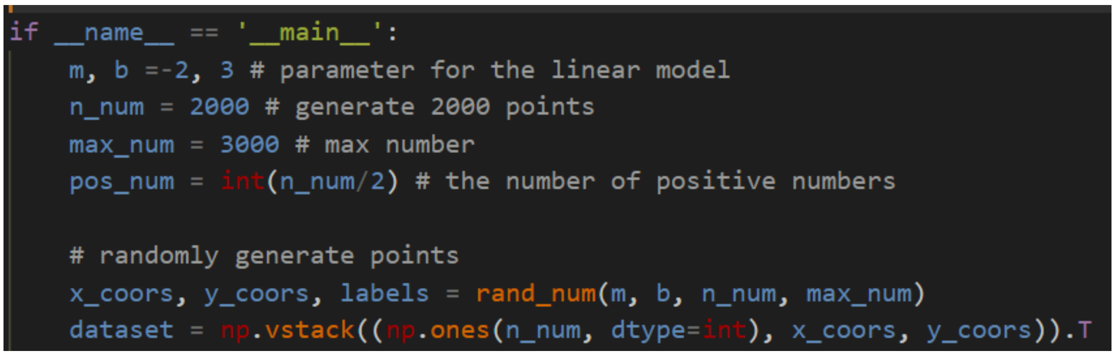
```python
#Pocket Algorithm
def Pocket(dataset, labels, times):
    error = 0.0
    w = np.zeros(3) #init w
    for i in range(0, times):
        for x, y in zip(dataset, labels):
            if int(np.sign(w.T.dot(x))) != y:
                wt = w + y * x
                error0 = error_rate(dataset, labels, w) #error of w
                error1 = error_rate(dataset, labels, wt) # error of wt
                if error1 < error0:
                    w = wt
                    error = error1
                    break
    return w, error
```

計算每次 Wt 的錯誤率



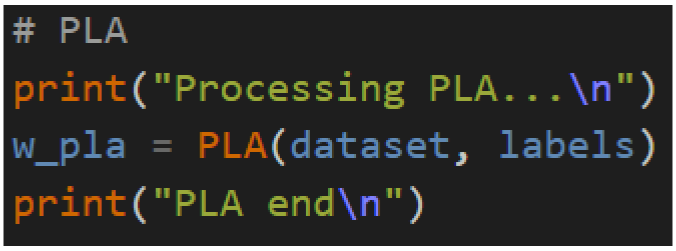
```python
def error_rate(dataset, labels, w):
    error = 0.0
    for x, y in zip(dataset, labels):
        if int(np.sign(w.T.dot(x))) != y:
            error = error +1.0
    return error/len(dataset)
```

設定參數，製作隨機樣本點



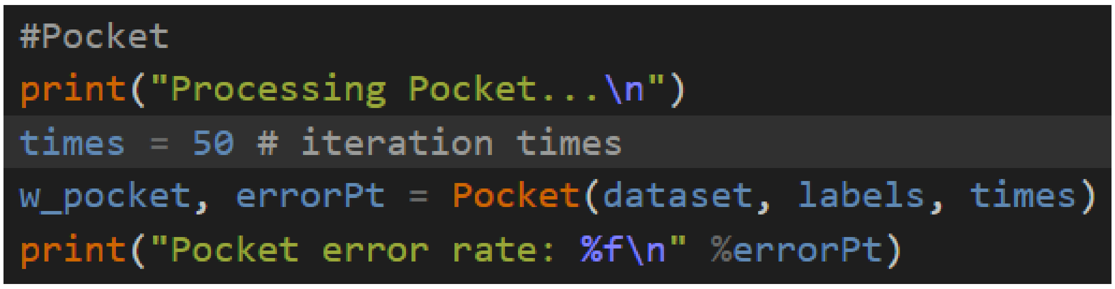
```python
if __name__ == '__main__':
    m, b =-2, 3 # parameter for the linear model
    n_num = 2000 # generate 2000 points
    max_num = 3000 # max number
    pos_num = int(n_num/2) # the number of positive numbers

    # randomly generate points
    x_coors, y_coors, labels = rand_num(m, b, n_num, max_num)
    dataset = np.vstack((np.ones(n_num, dtype=int), x_coors, y_coors)).T
```

執行 PLA



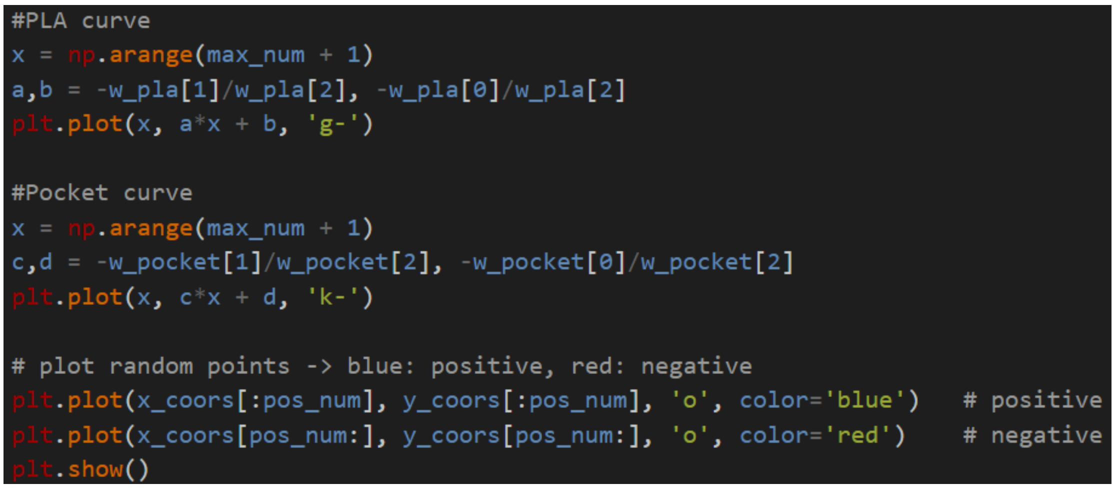
```python
# PLA
print("Processing PLA...\n")
w_pla = PLA(dataset, labels)
print("PLA end\n")
```

執行 Pocket，迭代次數設定為 50 次



```python
#Pocket
print("Processing Pocket...\n")
times = 50 # iteration times
w_pocket, errorPt = Pocket(dataset, labels, times)
print("Pocket error rate: %f\n" %errorPt)
```

畫圖



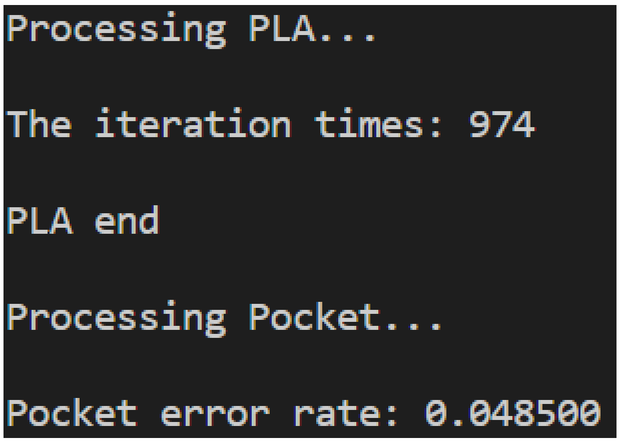
```python
#PLA curve
x = np.arange(max_num + 1)
a,b = -w_pla[1]/w_pla[2], -w_pla[0]/w_pla[2]
plt.plot(x, a*x + b, 'g-')

#Pocket curve
x = np.arange(max_num + 1)
c,d = -w_pocket[1]/w_pocket[2], -w_pocket[0]/w_pocket[2]
plt.plot(x, c*x + d, 'k-')

# plot random points -> blue: positive, red: negative
plt.plot(x_coors[:pos_num], y_coors[:pos_num], 'o', color='blue')   # positive
plt.plot(x_coors[pos_num:], y_coors[pos_num:], 'o', color='red')    # negative
plt.show()
```

Experimental results:

Pocket 明顯比 PLA 慢

Conclusion:

同樣資料，PLA 迭代次數為 974 次，但執行時間卻比 Pocket 快上許多



```
Processing PLA...

The iteration times: 974

PLA end

Processing Pocket...

Pocket error rate: 0.048500
```

Discussion:

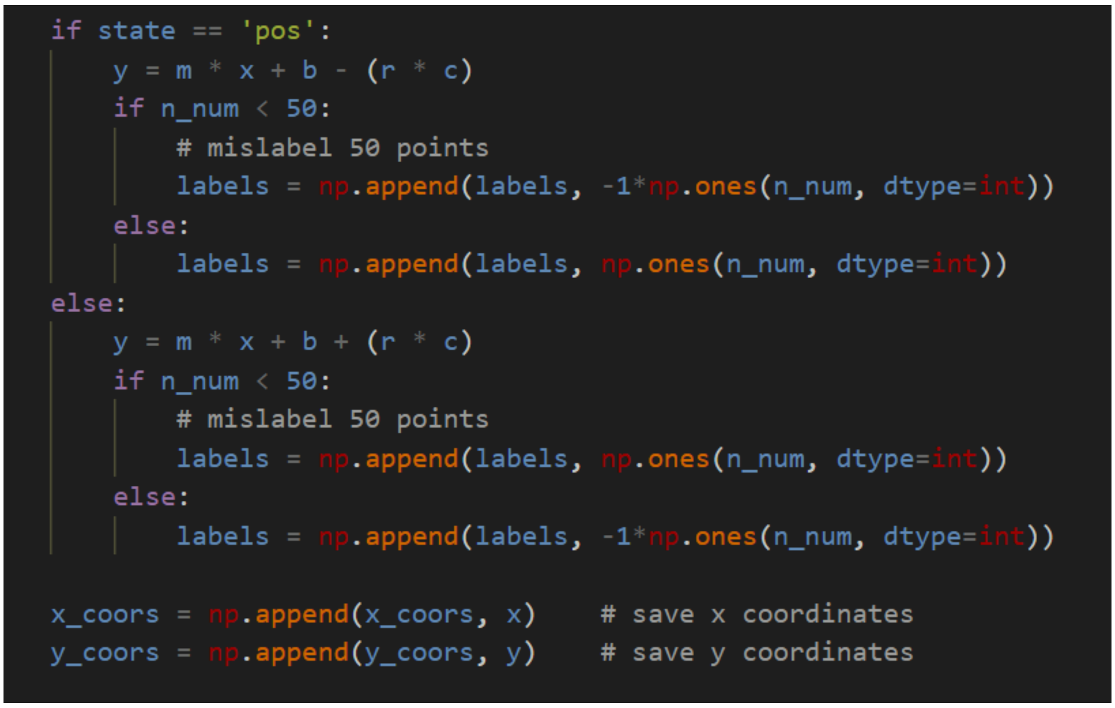
一開始 Pocket 的迭代次數設太大跑很久，改小後就跑比較快了

Q4.

Execution description:

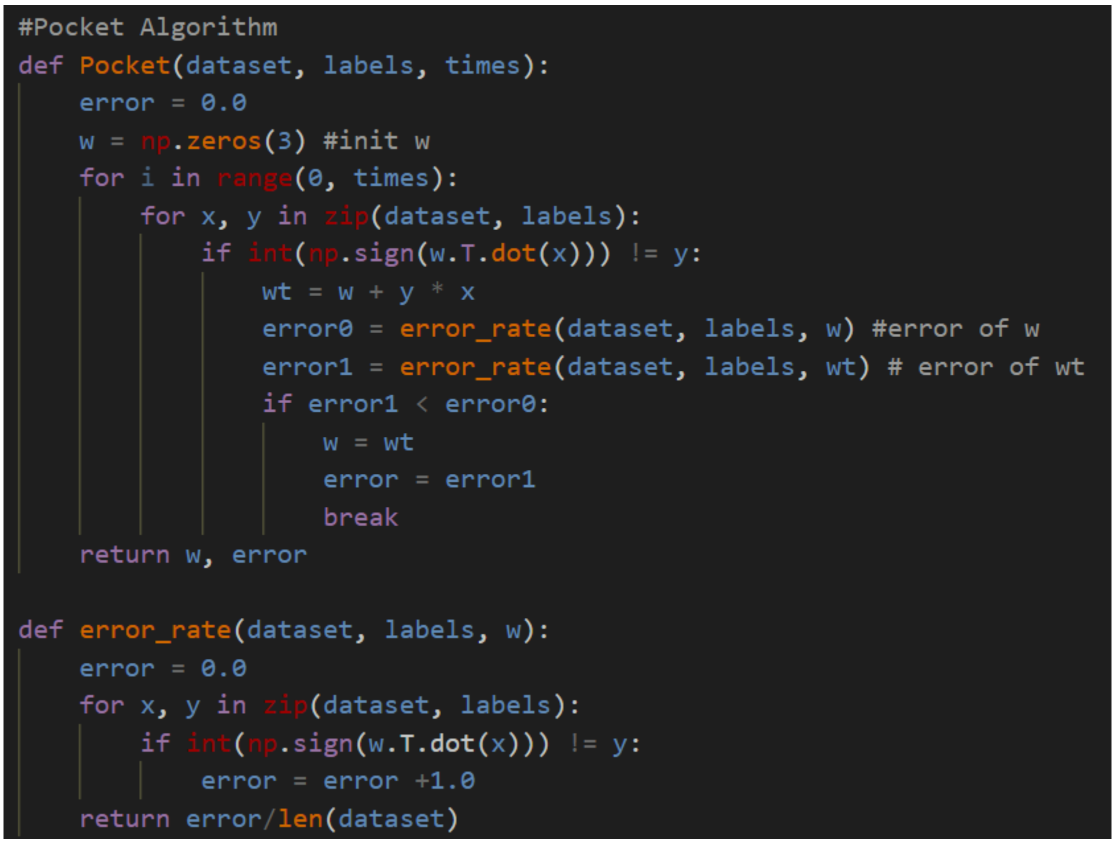
新增一個 Rand_num_mislabel 函式: 將 Rand_num 函式裡標標籤的地方做修

改，正負樣本前 50 筆都故意標錯



```python
if state == 'pos':
    y = m * x + b - (r * c)
    if n_num < 50:
        # mislabel 50 points
        labels = np.append(labels, -1*np.ones(n_num, dtype=int))
    else:
        labels = np.append(labels, np.ones(n_num, dtype=int))
else:
    y = m * x + b + (r * c)
    if n_num < 50:
        # mislabel 50 points
        labels = np.append(labels, np.ones(n_num, dtype=int))
    else:
        labels = np.append(labels, -1*np.ones(n_num, dtype=int))

x_coors = np.append(x_coors, x)    # save x coordinates
y_coors = np.append(y_coors, y)    # save y coordinates
```

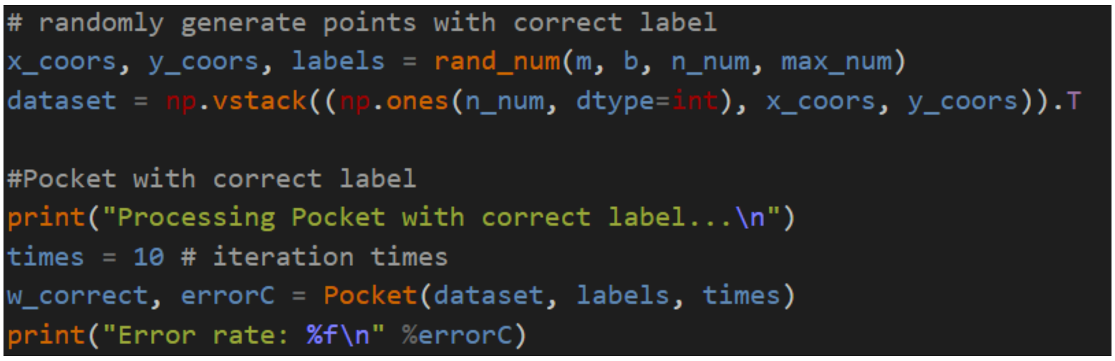
Pocket 演算法



```python
#Pocket Algorithm
def Pocket(dataset, labels, times):
    error = 0.0
    w = np.zeros(3) #init w
    for i in range(0, times):
        for x, y in zip(dataset, labels):
            if int(np.sign(w.T.dot(x))) != y:
                wt = w + y * x
                error0 = error_rate(dataset, labels, w) #error of w
                error1 = error_rate(dataset, labels, wt) # error of wt
                if error1 < error0:
                    w = wt
                    error = error1
                    break
    return w, error

def error_rate(dataset, labels, w):
    error = 0.0
    for x, y in zip(dataset, labels):
        if int(np.sign(w.T.dot(x))) != y:
            error = error +1.0
    return error/len(dataset)
```

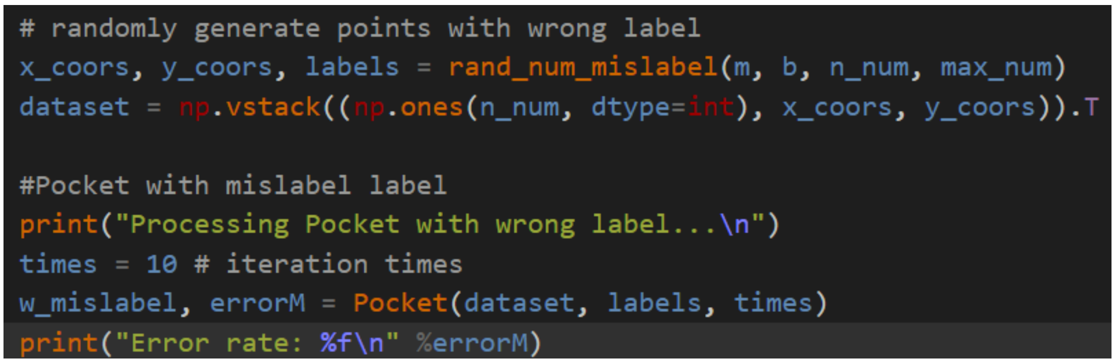
Main Function:

自定義參數，製作隨機樣本(正確)，執行 Pocket 演算法(迭代次數 10 次)



```python
# randomly generate points with correct label
x_coors, y_coors, labels = rand_num(m, b, n_num, max_num)
dataset = np.vstack((np.ones(n_num, dtype=int), x_coors, y_coors)).T

#Pocket with correct label
print("Processing Pocket with correct label...\n")
times = 10 # iteration times
w_correct, errorC = Pocket(dataset, labels, times)
print("Error rate: %f\n" %errorC)
```

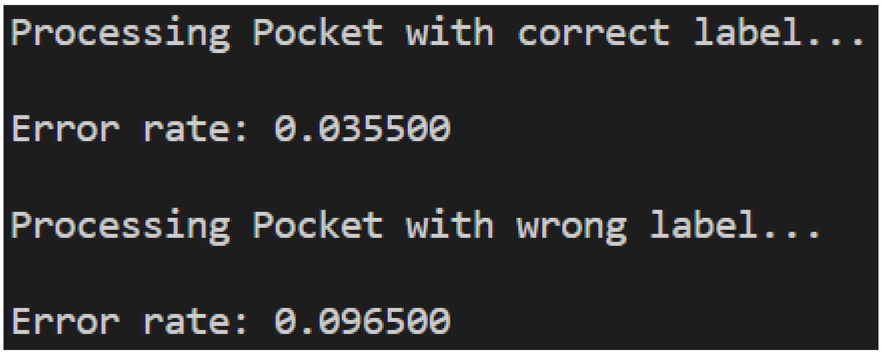
製作題目要求的樣本，執行 Pocket 演算法(迭代次數 10 次)



```
# randomly generate points with wrong label
x_coors, y_coors, labels = rand_num_mislabel(m, b, n_num, max_num)
dataset = np.vstack((np.ones(n_num, dtype=int), x_coors, y_coors)).T

#Pocket with mislabel label
print("Processing Pocket with wrong label...\n")
times = 10 # iteration times
w_mislabel, errorM = Pocket(dataset, labels, times)
print("Error rate: %f\n" %errorM)
```

Experimental results:

樣本點沒有出錯的正確率比較高



```
Processing Pocket with correct label...

Error rate: 0.035500

Processing Pocket with wrong label...

Error rate: 0.096500
```

Conclusion:

從結果可以得知正確的標籤錯誤率較低，也就是說正確率較高

Discussion:

雖然說標錯的資料錯誤率較高，但她是線性不可分割的資料，所以用 PLA 無法

實作