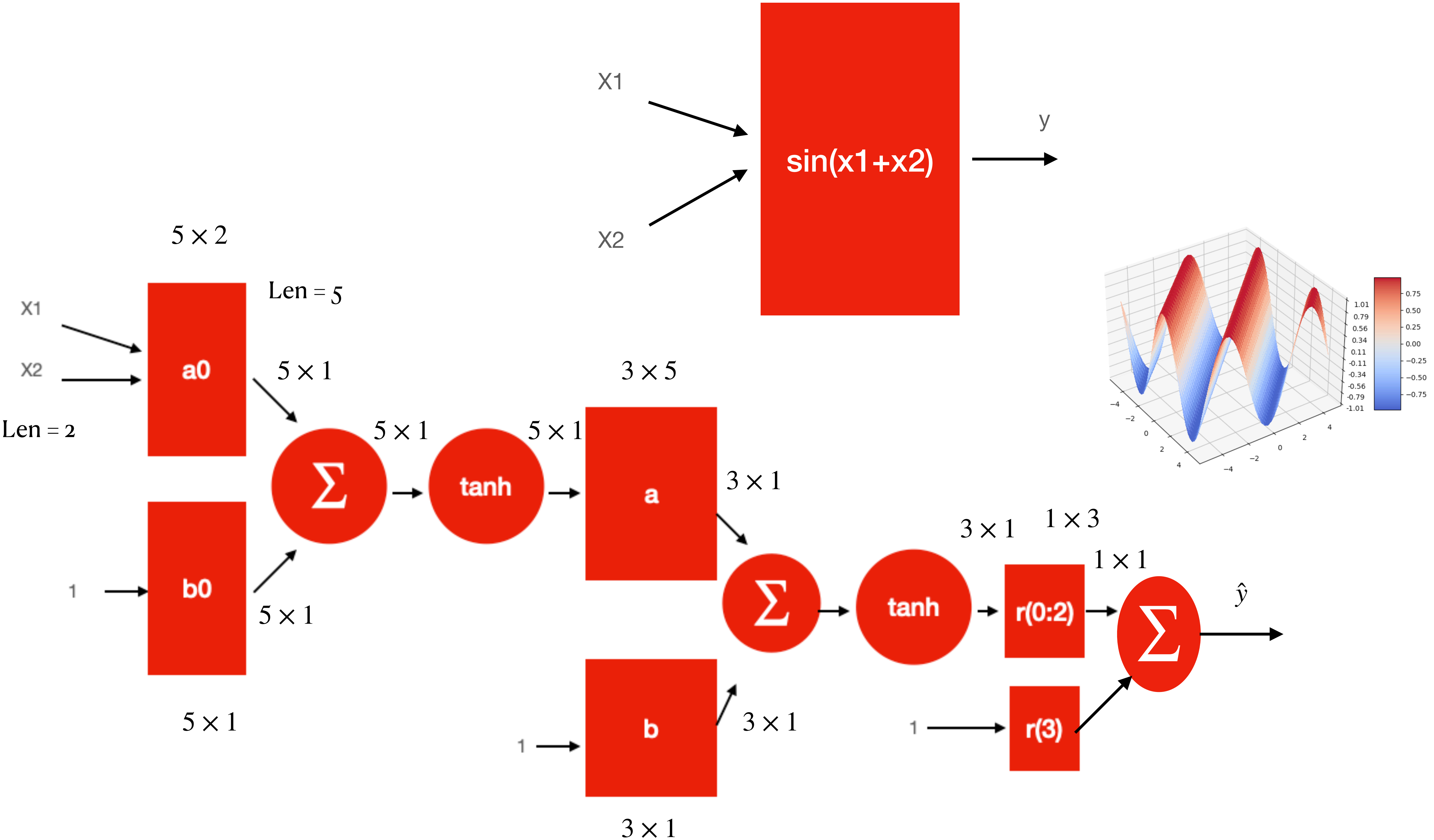
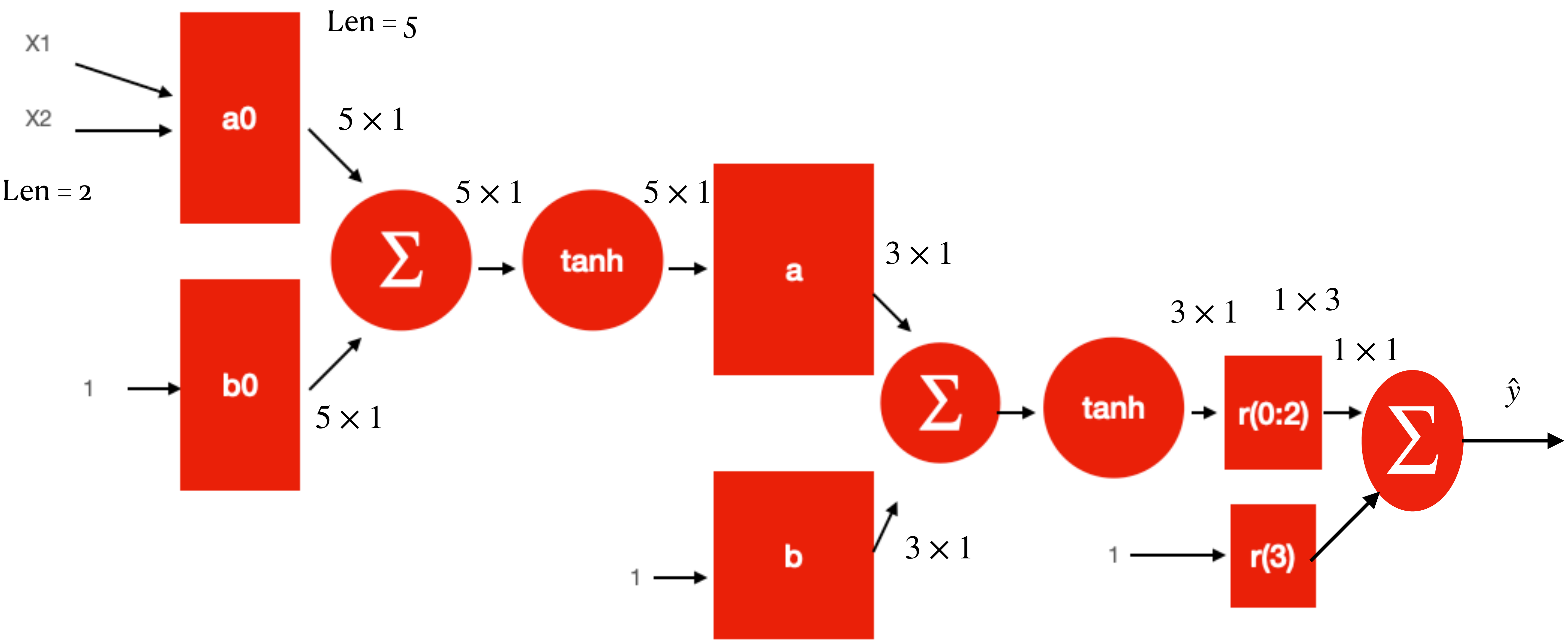
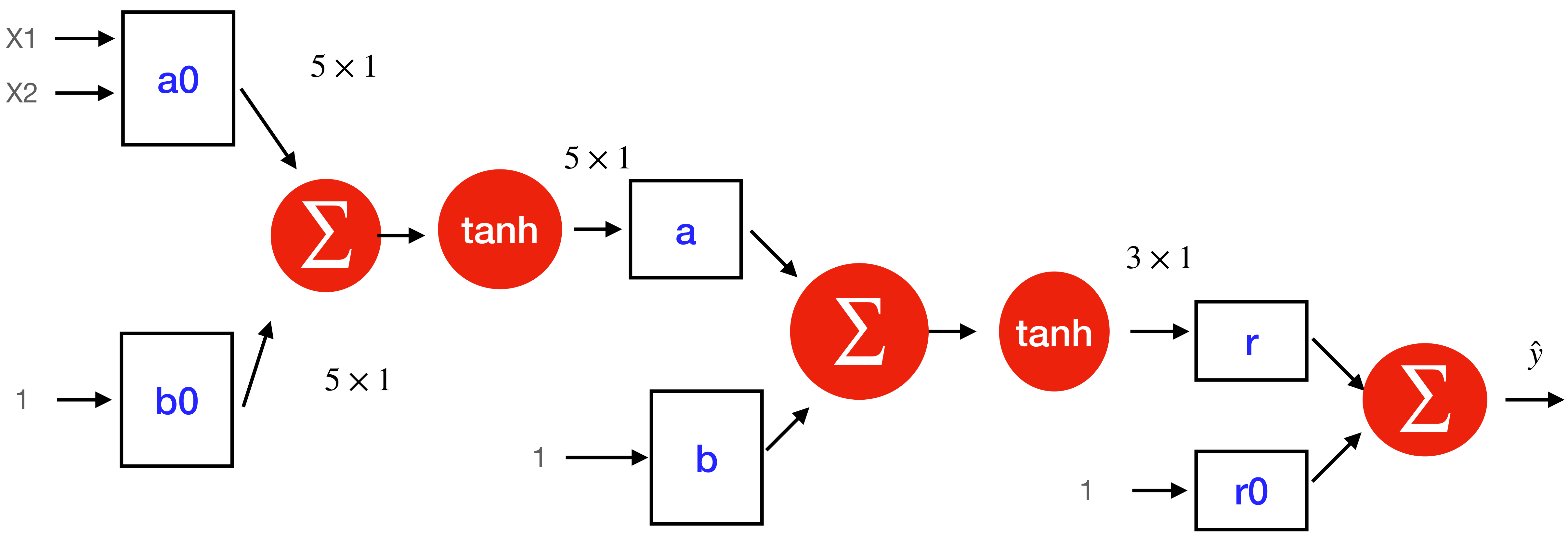
反矩陣的資料分析應用

給定不共線的多個點求一直線


X1
X2
sin(x1+x2)
y
5 × 2
Len = 5
X1
X2
a0
Len = 2
1
b0
5 × 1
5 × 1
5 × 1
5 × 1
5 × 1
Σ
tanh
3 × 5
a
3 × 1
1
b
3 × 1
3 × 1
Σ
tanh
3 × 1
1 × 3
r(0:2)
1 × 1
1
r(3)
Σ
ŷ

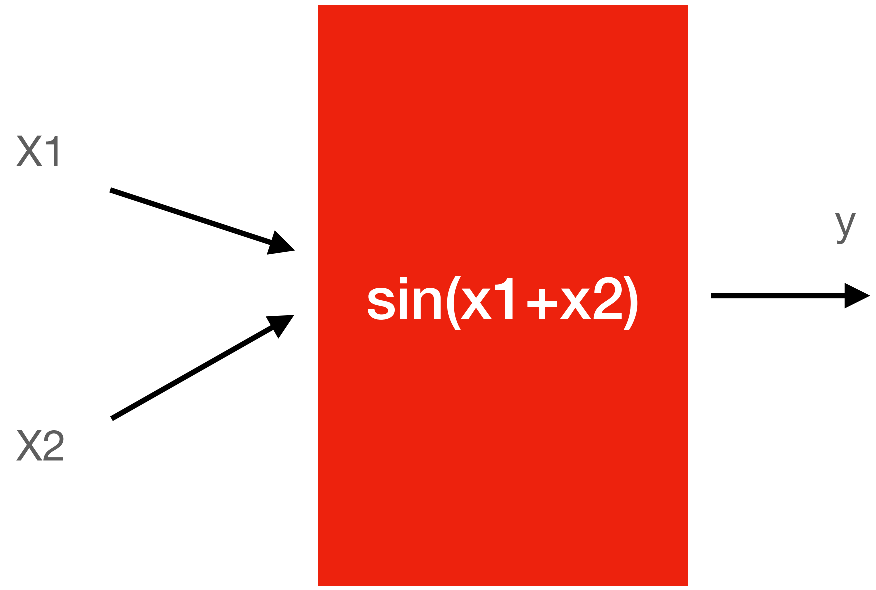


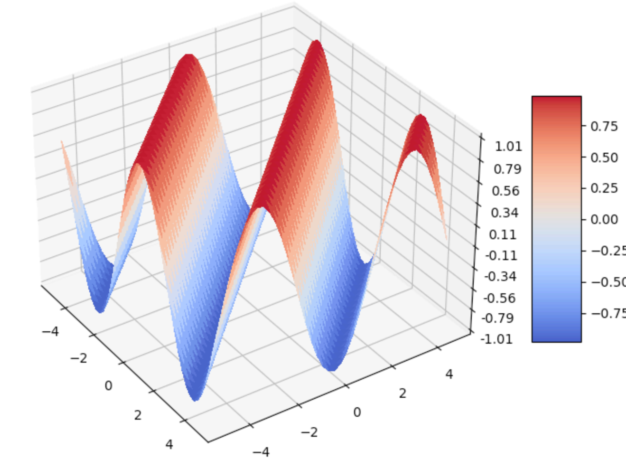
X1
X2
Len = 2
a0
Len = 5
5 × 1
b0
1
5 × 1
Σ
5 × 1
tanh
5 × 1
a
3 × 1
b
1
3 × 1
Σ
tanh
3 × 1
1 × 3
r(0:2)
1 × 1
r(3)
1
Σ
ŷ


X1
X2
a0
5×1
1
b0
5×1
Σ
tanh
5×1
a
1
b
Σ
tanh
3×1
r
1
r0
Σ
$\hat{y}$

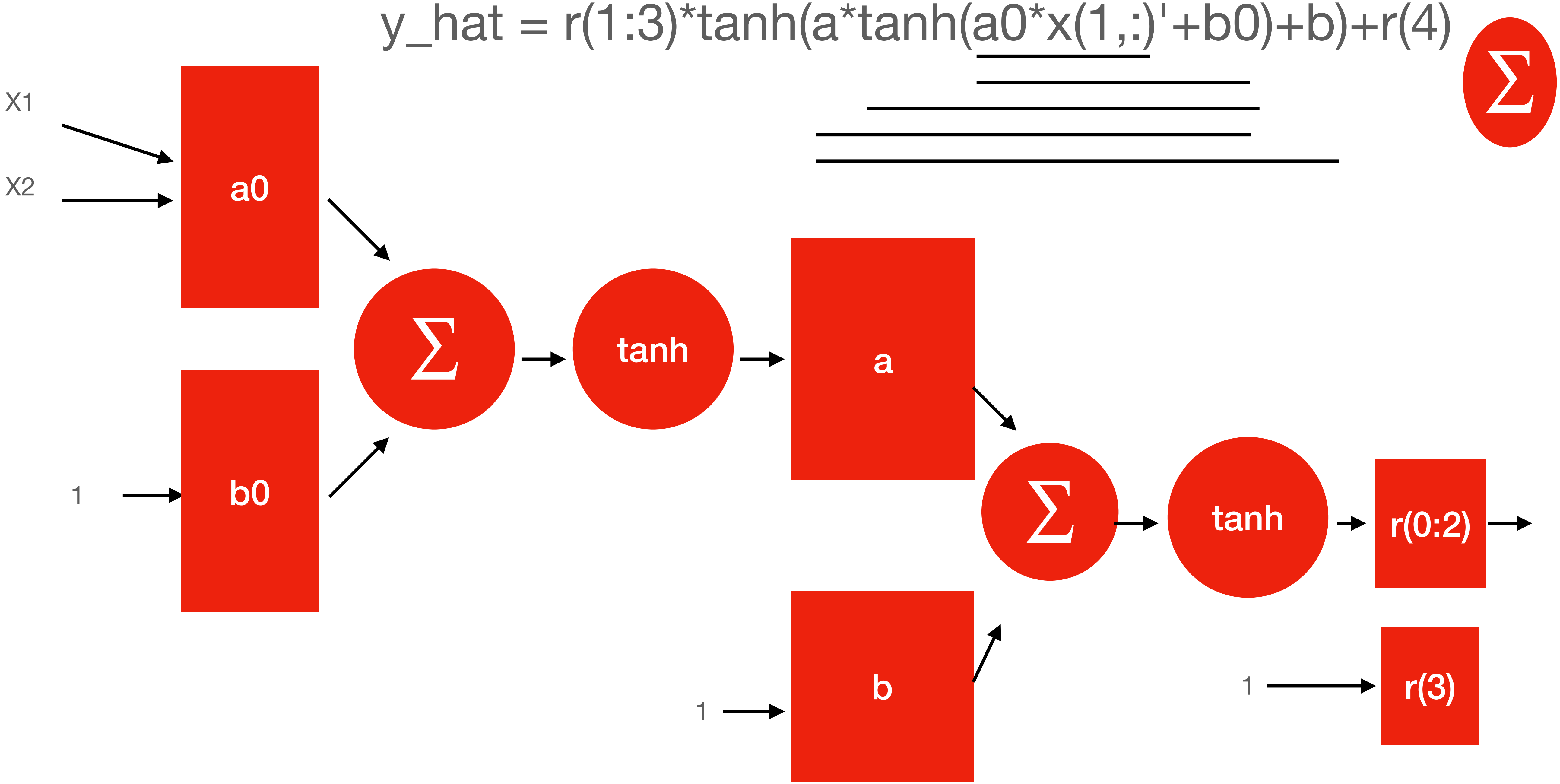
Mean square error 10^-6

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

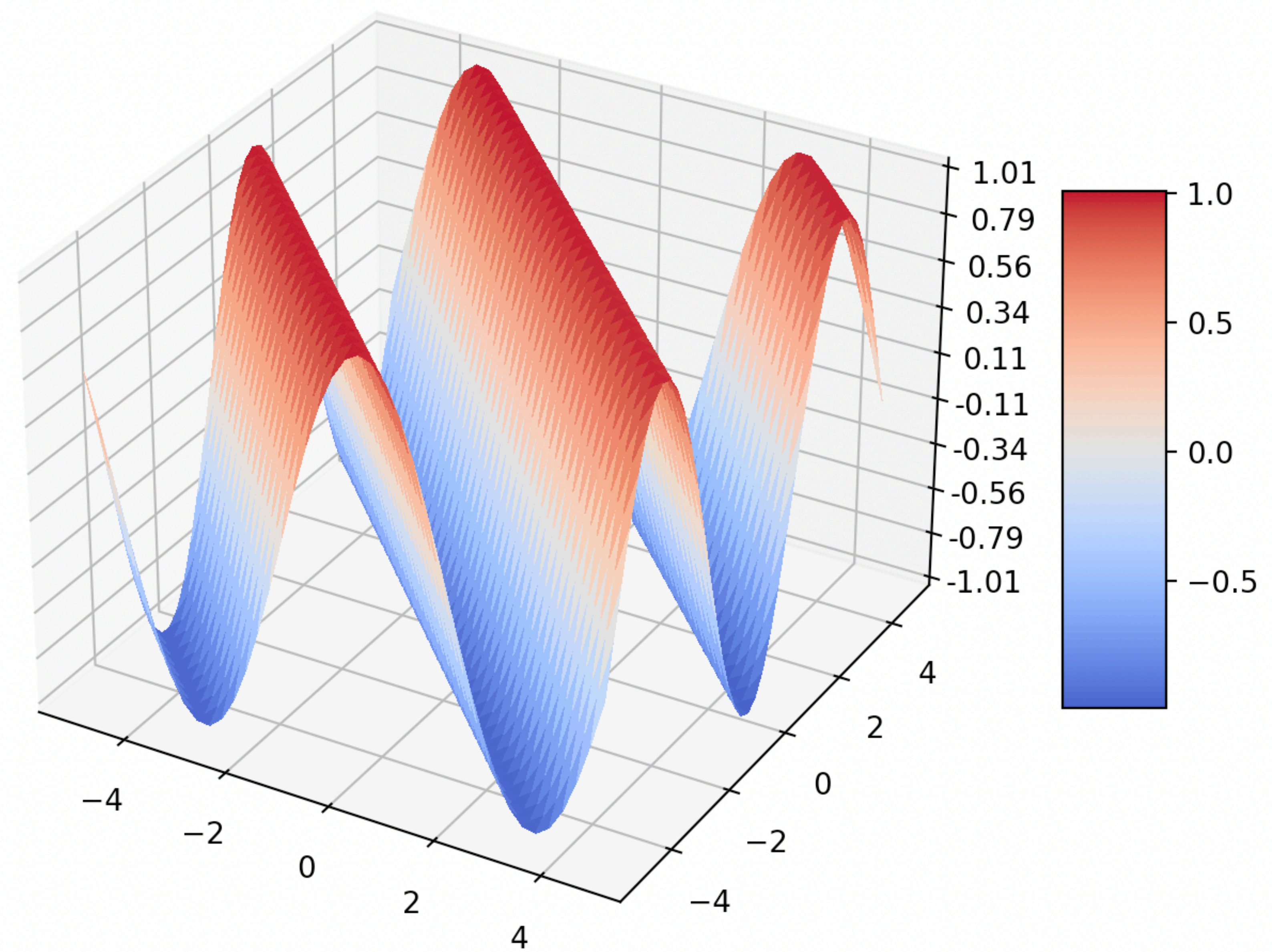




y_hat = r(1:3)*tanh(a*tanh(a0*x(1,:)'+b0)+b)+r(4)



```python
def mySin(x):
    a0 = np.array([[-0.3918, -0.3918], [-0.0975, -0.0975],
                   [-0.3304, -0.3303], [-0.2186, -0.2185],
                   [-0.3472, -0.3472]])
    b0 = np.array(
    [[-3.5903], [0.1043], [1.9609], [1.1944], [-1.0927]])
    a = np.array([[-0.6988, -0.1312, 0.5554, -2.1678, -0.1380],
                  [-0.2736, -4.3798, 0.7643, 5.0048, -2.2072],
                  [0.4992, -3.2731, -0.8272, 2.2502, 0.5677]])
    b = np.array([[0.6347], [-0.3891], [-0.1289]])
    r = np.array([-16.7872, 7.2905, 36.4077, 7.4048])
    z = a0 @ x[0, :]
    h0 = np.reshape(z, np.shape(b0)) + b0
    v = np.tanh(h0)
    h1 = a @ v + b
    v2 = np.tanh(h1)
    y_hat = r[0:3] @ v2 + r[3]
    return y_hat
```



−4
−2
0
2
4
−4
−2
0
2
4
1.01
0.79
0.56
0.34
0.11
-0.11
-0.34
-0.56
-0.79
-1.01
1.0
0.5
0.0
−0.5

```python
# Make data.
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid( *xi: X, Y)
Z = np.sin(X+Y)

# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
```

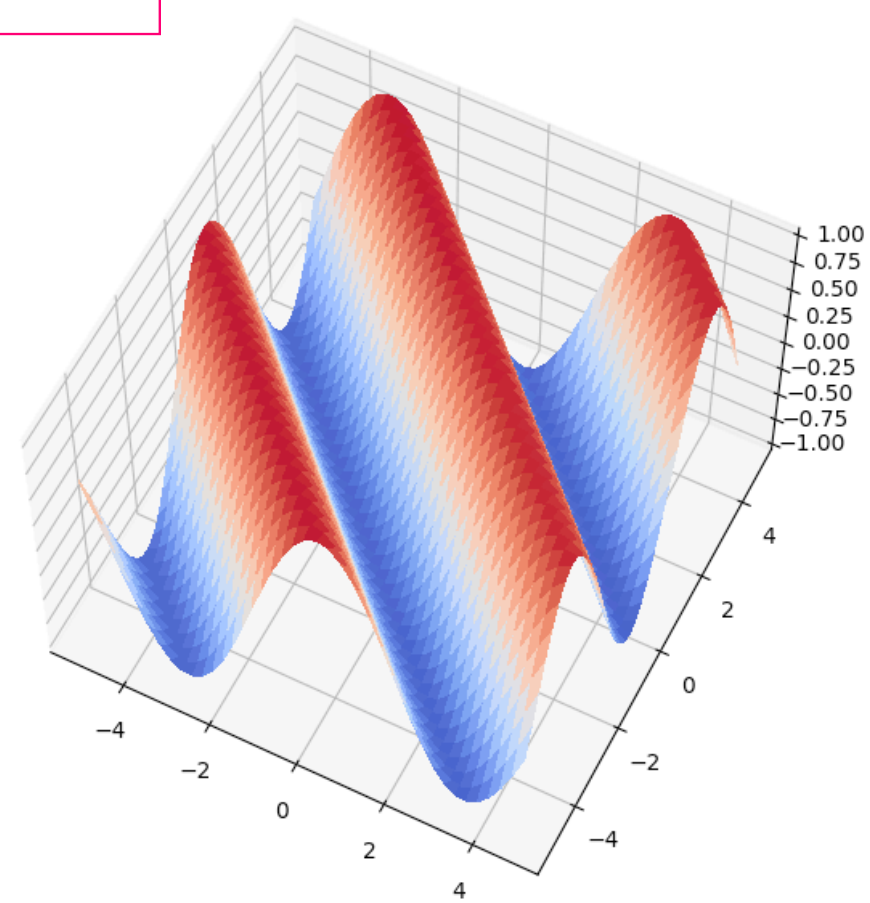
Shape of X+Y
(40,40)

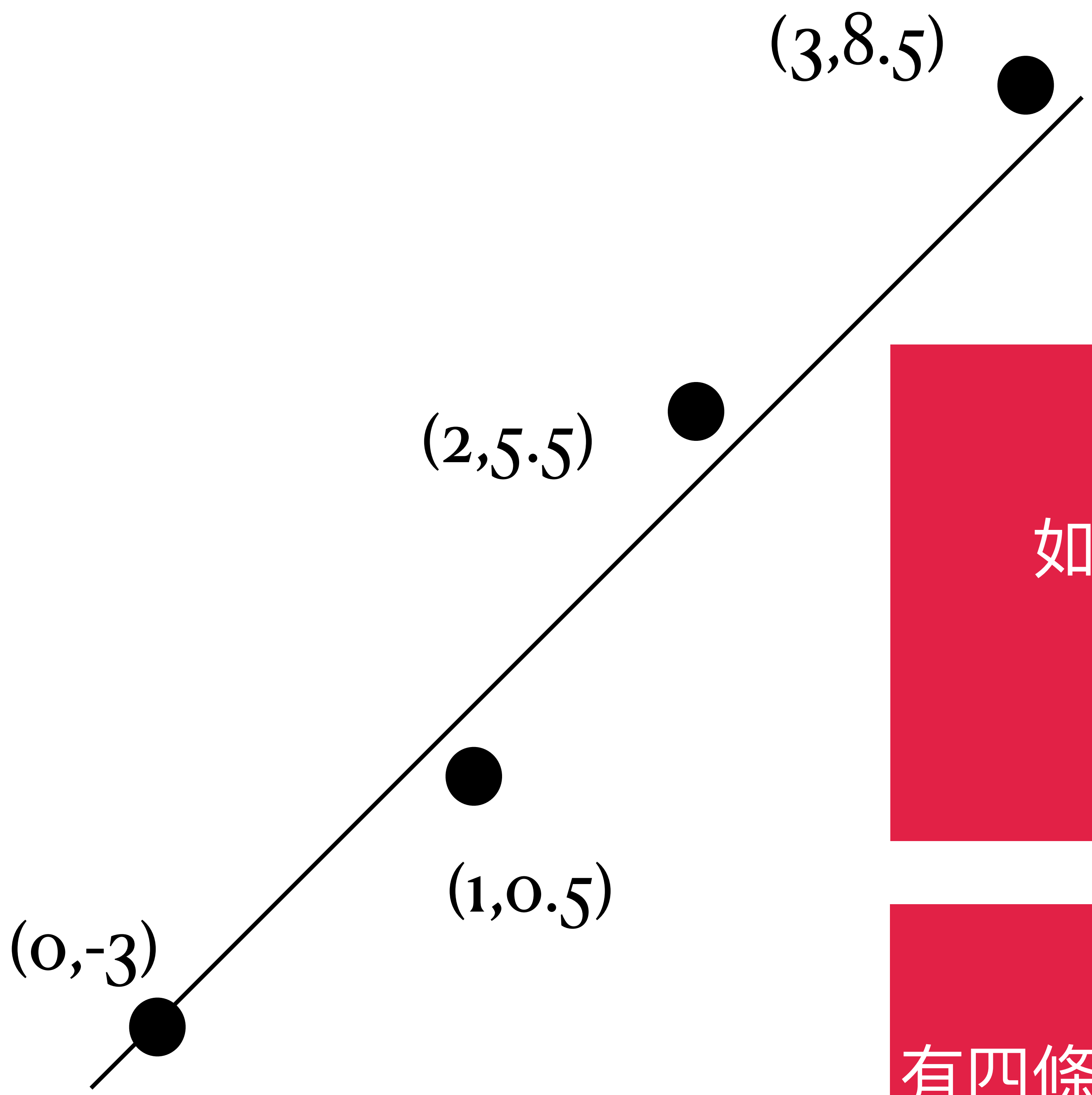
Figure4

```python
x = np.random.rand(1, 2) * 4 * np.pi - 2 * np.pi
y = np.sin(x[0, 0] + x[0, 1])
y_hat = mySin(x)
print((y - y_hat[0]) ** 2)
```

Shape of x
(1, 2)

```python
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid( *xi: X, Y)
shapeX = np.shape(X)
Z = np.zeros((shapeX[0],shapeX[1]))
v = np.random.rand(1,2)
for i in range(shapeX[0]):
    for j in range(shapeX[1]):
        v[0,0] = X[i,j]
        v[0,1]= Y[i,j]
        Z[i,j] = mysin(v)
```





四點不共線
如何求出最好的一條線，
描述這四點呢？

線性系統是長方形的
有四條方程式，但是只有兩個未知數

步驟一：匯入套裝

```
import numpy as np
from numpy.linalg import inv
```

步驟二：將資料以矩陣A向量b表示

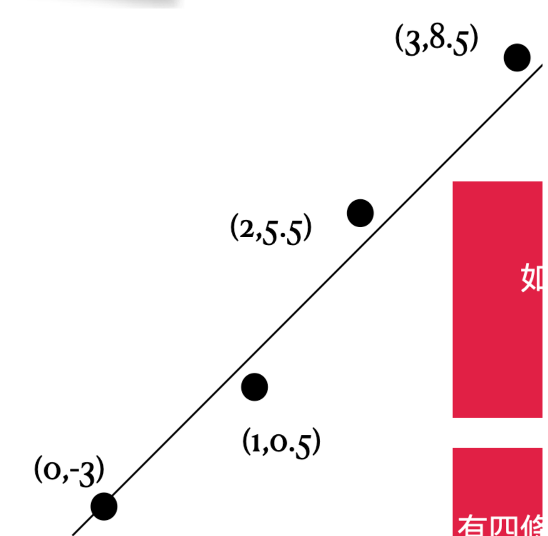
```
A = np.matrix([[2,1],[1,1],[3,1],[0,1]])
b = np.matrix([[5.5],[0.5],[8.5],[-3]])
```

步驟三：求轉置矩陣

```
AT = np.matrix.        (A)
```

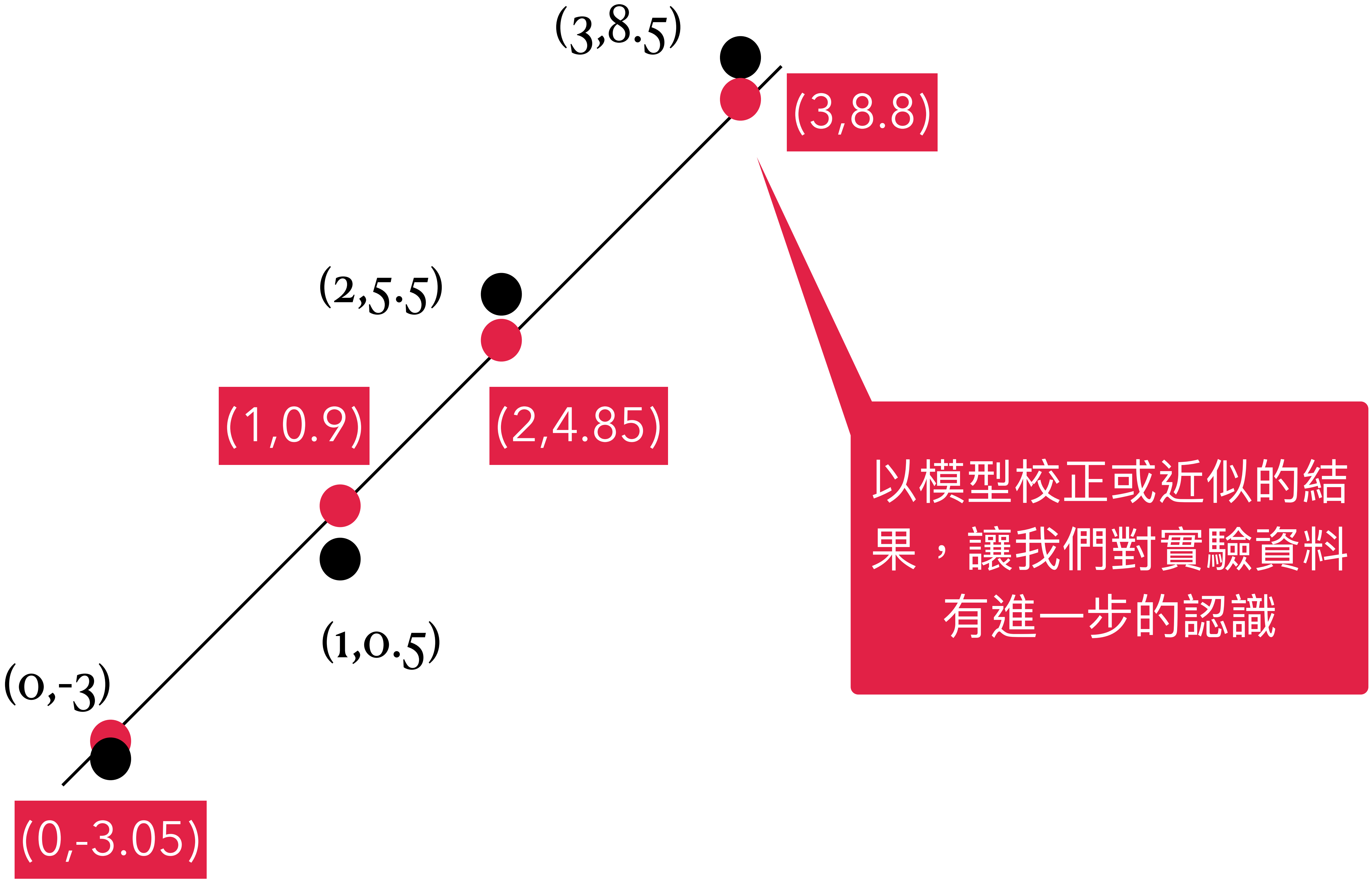
步驟四：求最佳參數

```
B =         A
invB =     (B)
ans =
```



如

有四


(3,8.5)
(3,8.8)
(2,5.5)
(1,0.9)
(2,4.85)
(1,0.5)
(0,-3)
(0,-3.05)
以模型校正或近似的結果，讓我們對實驗資料有進一步的認識

```python
import numpy as np
from numpy.linalg import inv
```

Matrix

```
A = np.matrix([range(1,5)])
```

```
matrix([[1, 2, 3, 4]])
```

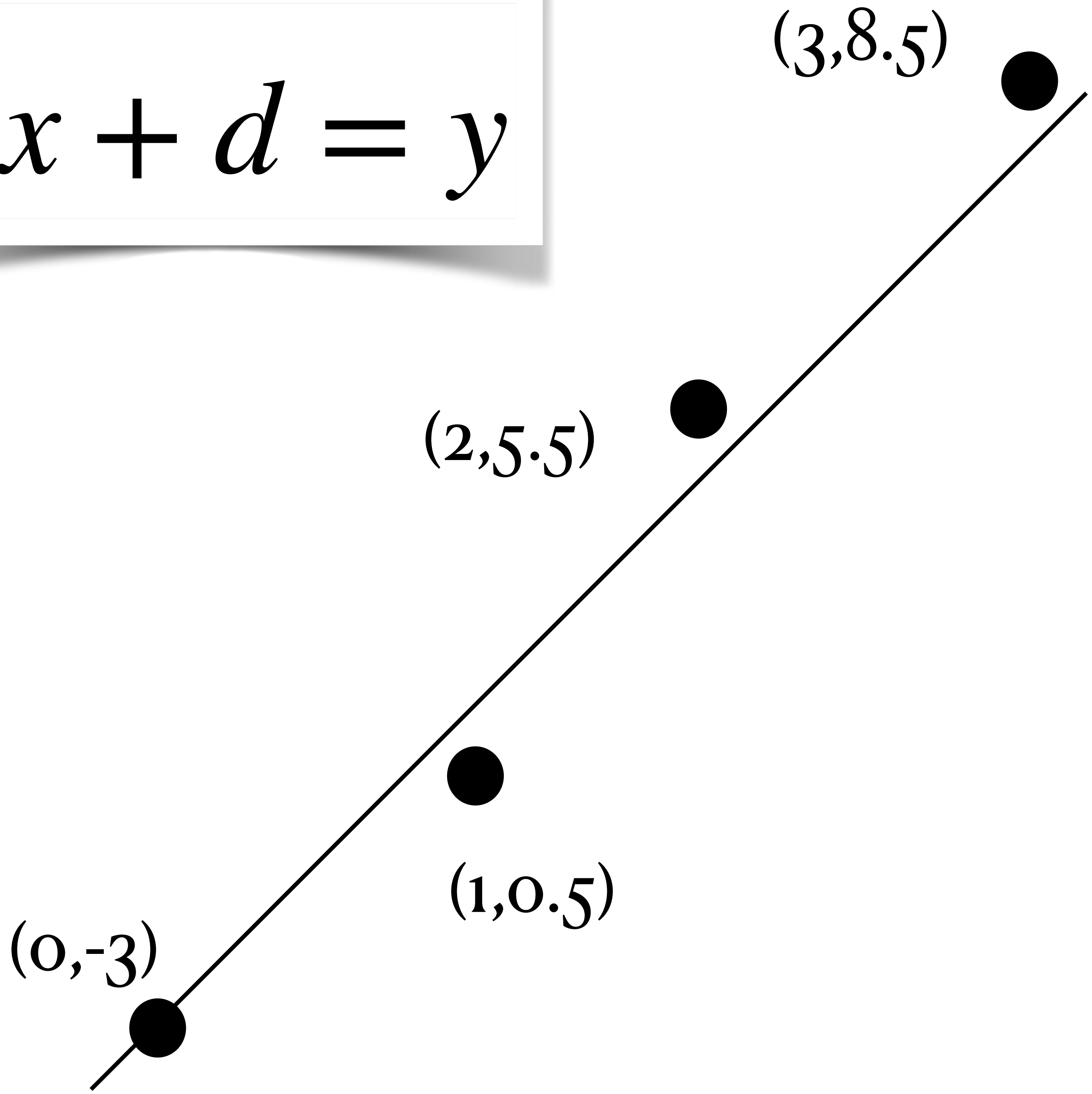
轉置矩陣

Matrix 轉置

```
A = np.matrix([range(1,5)])
np.matrix.transpose(A)
```

```
>>> np.matrix.transpose(A)
matrix([[1],
        [2],
        [3],
        [4]])
```

$$ax + d = y$$



$$\begin{bmatrix} x & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = y$$

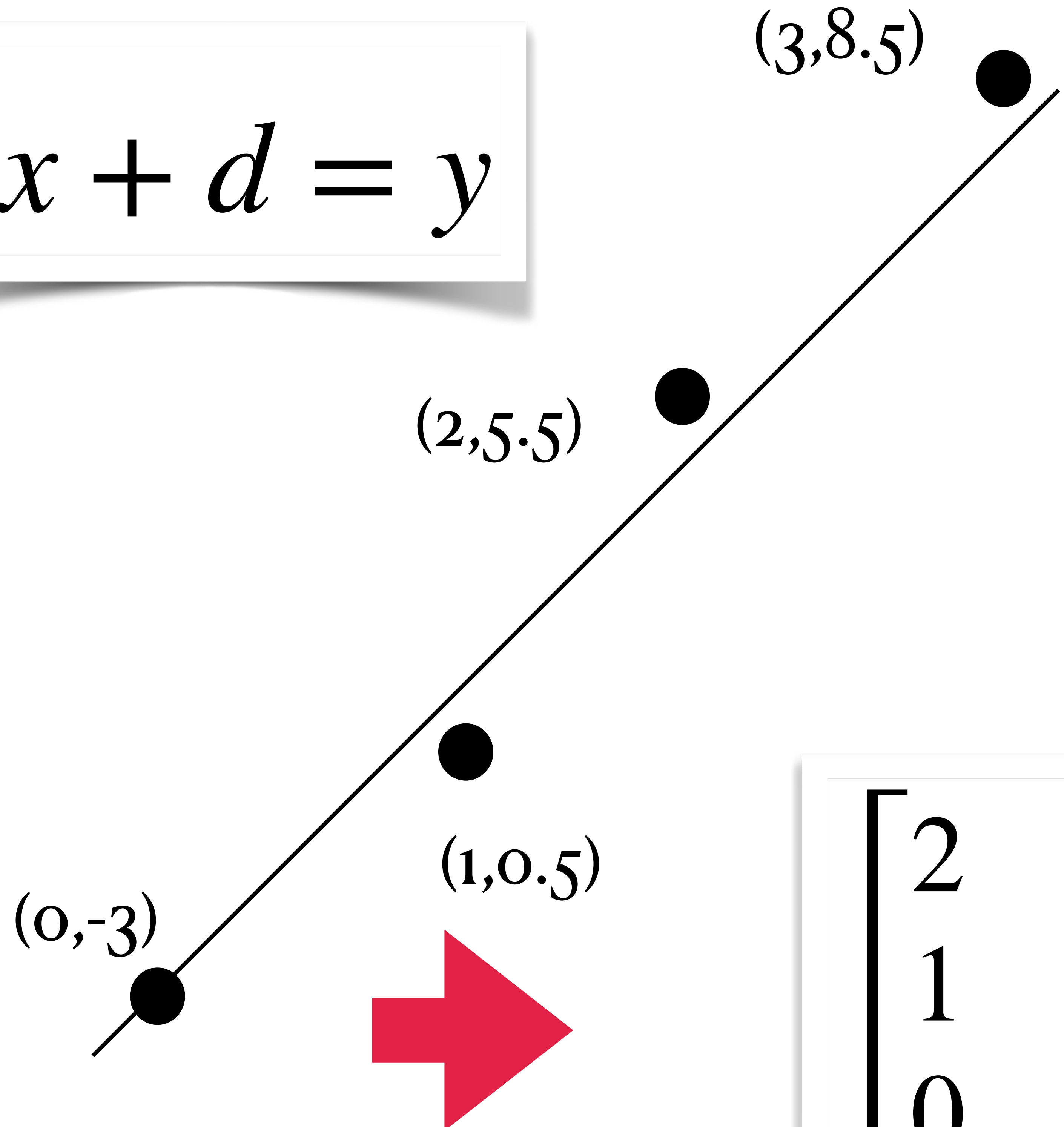
$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = 5.5$$

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = 0.5$$

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = -3$$

$$\begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = 8.5$$

$$ax + d = y$$



$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = 5.5$$

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = 0.5$$

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = -3$$

$$\begin{bmatrix} 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = 8.5$$

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 0 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}$$

長方形的系統

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 0 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}$$

矩陣A

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 0 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}$$

向量b

```
A = np.matrix([[2,1],[1,1],[3,1],[0,1]])
b = np.matrix([[5.5],[0.5],[8.5],[-3]])
```

```
>>> A
matrix([[2, 1],
        [1, 1],
        [3, 1],
        [0, 1]])
```

```
>>> b
matrix([[ 5.5],
        [ 0.5],
        [ 8.5],
        [-3. ]])
```

```
A = np.matrix([[2,1],[1,1],[3,1],[0,1]])
b = np.matrix([[5.5],[0.5],[8.5],[-3]])
```

可以求A的反矩陣嗎？

反矩陣一定是正方形的矩陣才有反矩陣

使用轉置矩陣
讓等號左邊變成
正方形矩陣

$$\overset{\substack{2\times 4\\ A^t}}{\begin{bmatrix} 2 & 1 & 0 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}} \overset{\substack{4\times 2\\ A}}{\begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 0 & 1 \\ 3 & 1 \end{bmatrix}} \begin{bmatrix} a \\ d \end{bmatrix} = \overset{\substack{2\times 4\\ A^t}}{\begin{bmatrix} 2 & 1 & 0 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}} \overset{\substack{4\times 1\\ b}}{\begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}}$$

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 0 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}$$

```python
A = np.matrix([[2,1],[1,1],[3,1],[0,1]])
b = np.matrix([[5.5],[0.5],[8.5],[-3]])
AT = np.transpose(A)
AT @ A
```

```
>>> print(AT@A)
[[14  6]
 [ 6  4]]
```

```
>>> AT @ b
matrix([[37. ],
        [11.5]])
```

$$\begin{bmatrix} 2 & 1 & 0 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 0 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 2 & 1 & 0 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}$$

```
>>> print(AT@A)
[[14  6]
 [ 6  4]]
```

```
>>> AT @ b
matrix([[37. ],
        [11.5]])
```

$$\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 37 \\ 11.5 \end{bmatrix}$$

$$\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 37 \\ 11.5 \end{bmatrix}$$

可以使用反矩陣嗎？

等號左邊已經是正方形矩陣
那就可能有反矩陣

觀察：向量元素不同

討論：d不是只代表截距

那代表時麼呢？

$$\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 37 \\ 11.5 \end{bmatrix}$$

d代表截距+雜訊
因為點不在直線上

$$\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 37 \\ 11.5 \end{bmatrix}$$

$$ax + d = y$$

近似的概念：
容忍雜訊
直接求解

$$\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 37 \\ 11.5 \end{bmatrix}$$

最小（平方）近似誤差
提供近似概念

可以求最小（平方）近似誤差的直線

近似的概念：
忽略雜訊，直接求解

$$\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 37 \\ 11.5 \end{bmatrix}$$

$$\begin{bmatrix} a \\ d \end{bmatrix} = \boxed{inv\left(\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix}\right)\begin{bmatrix} 37 \\ 11.5 \end{bmatrix}}$$

```
>>> A
matrix([[14,  6],
        [ 6,  4]])
```

```
B = np.matrix([[14,6],[6,4]])
invB = inv(B)
ans = invB @ np.matrix([[37],[11.5]])
```

```
>>> invA
matrix([[ 0.2, -0.3],
        [-0.3,  0.7]])
```

$$inv(\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix})\begin{bmatrix} 37 \\ 11.5 \end{bmatrix}$$

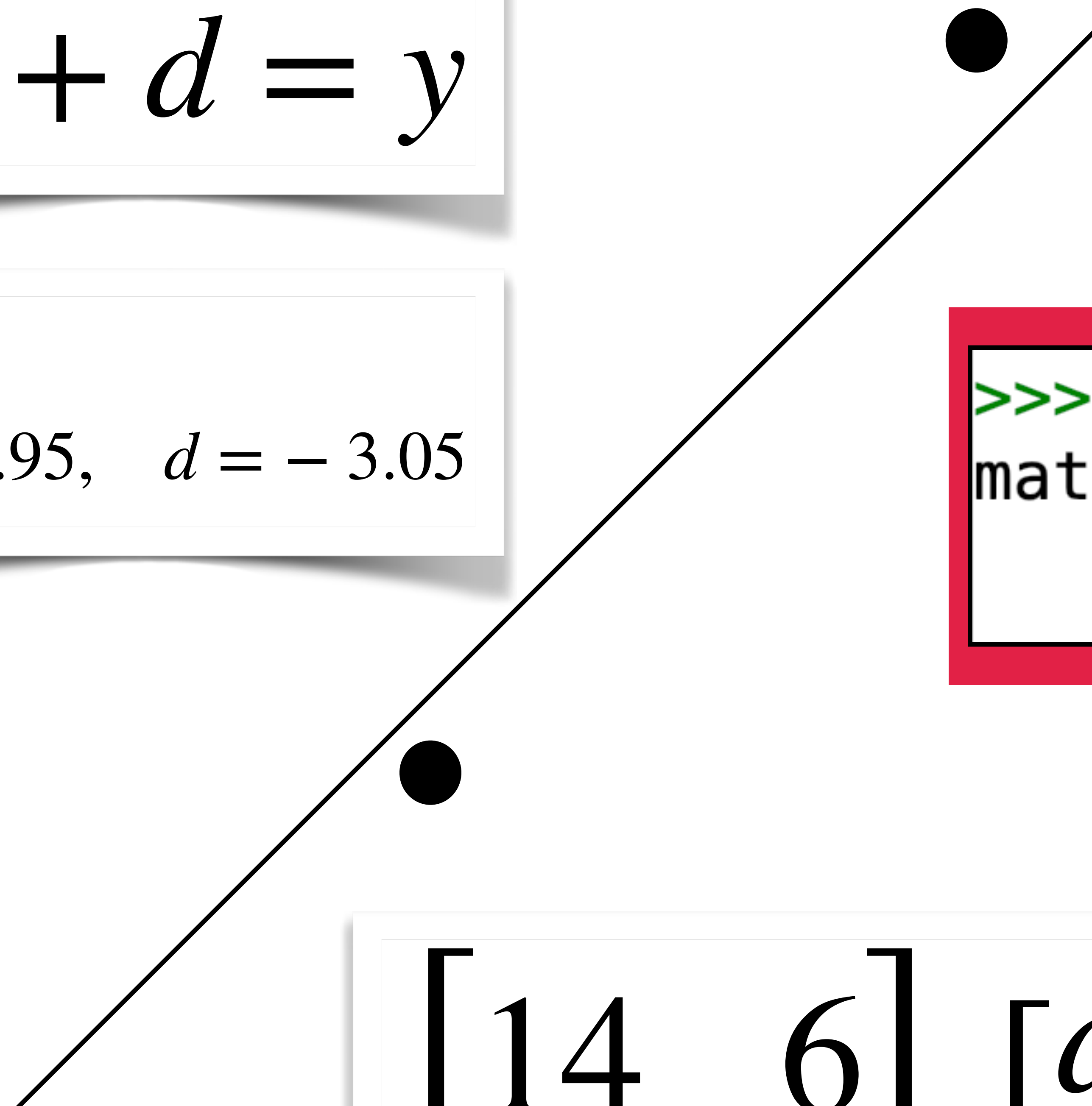
```
>>> ans
matrix([[ 3.95],
        [-3.05]])
```

```
A = np.matrix([[2,1],[1,1],[3,1],[0,1]])
b = np.matrix([[5.5],[0.5],[8.5],[-3]])
AT = np.transpose(A)
```

```
B = AT @ A
invB = inv(B)
ans = invB @ AT @ b
```

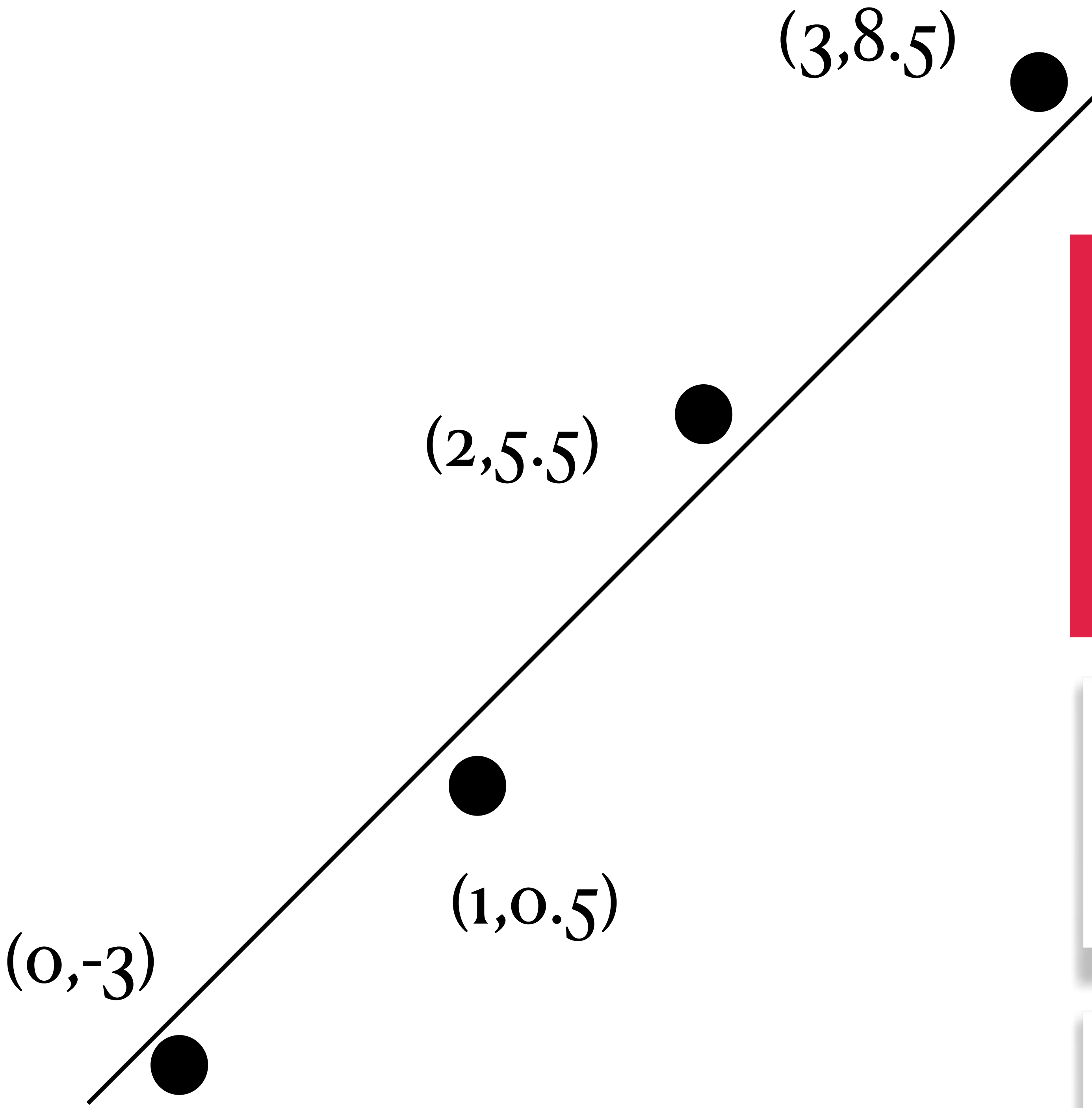
$$ax + d = y$$

$$a = 3.95, \quad d = -3.05$$



```
>>> ans
matrix([[ 3.95],
        [-3.05]])
```

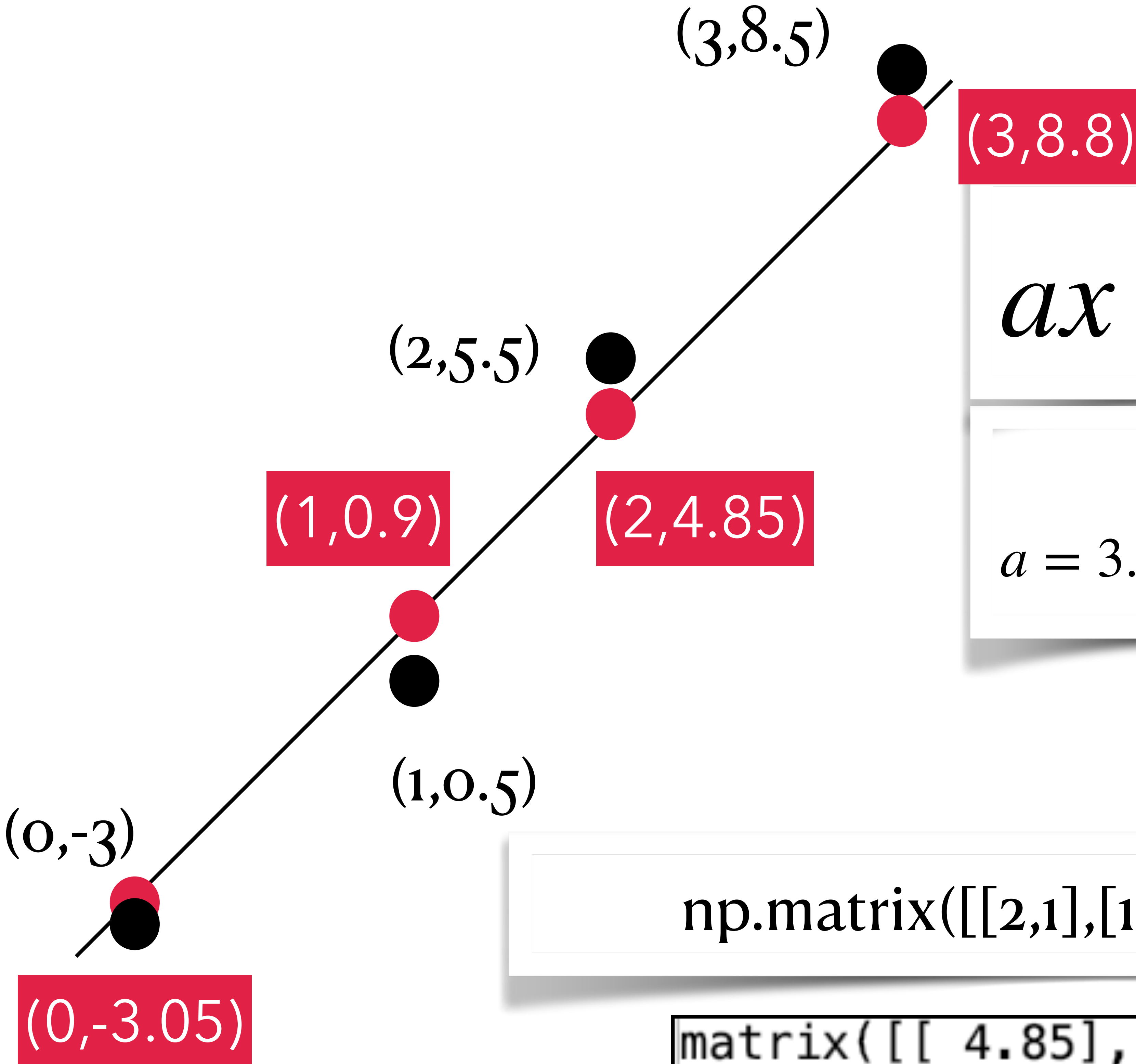
$$\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix} \begin{bmatrix} a \\ d \end{bmatrix} = \begin{bmatrix} 37 \\ 11.5 \end{bmatrix}$$



四點不共線
如何求出最好的一條線，
描述這四點呢？

$$ax + d = y$$

$$a = 3.95, \quad d = -3.05$$

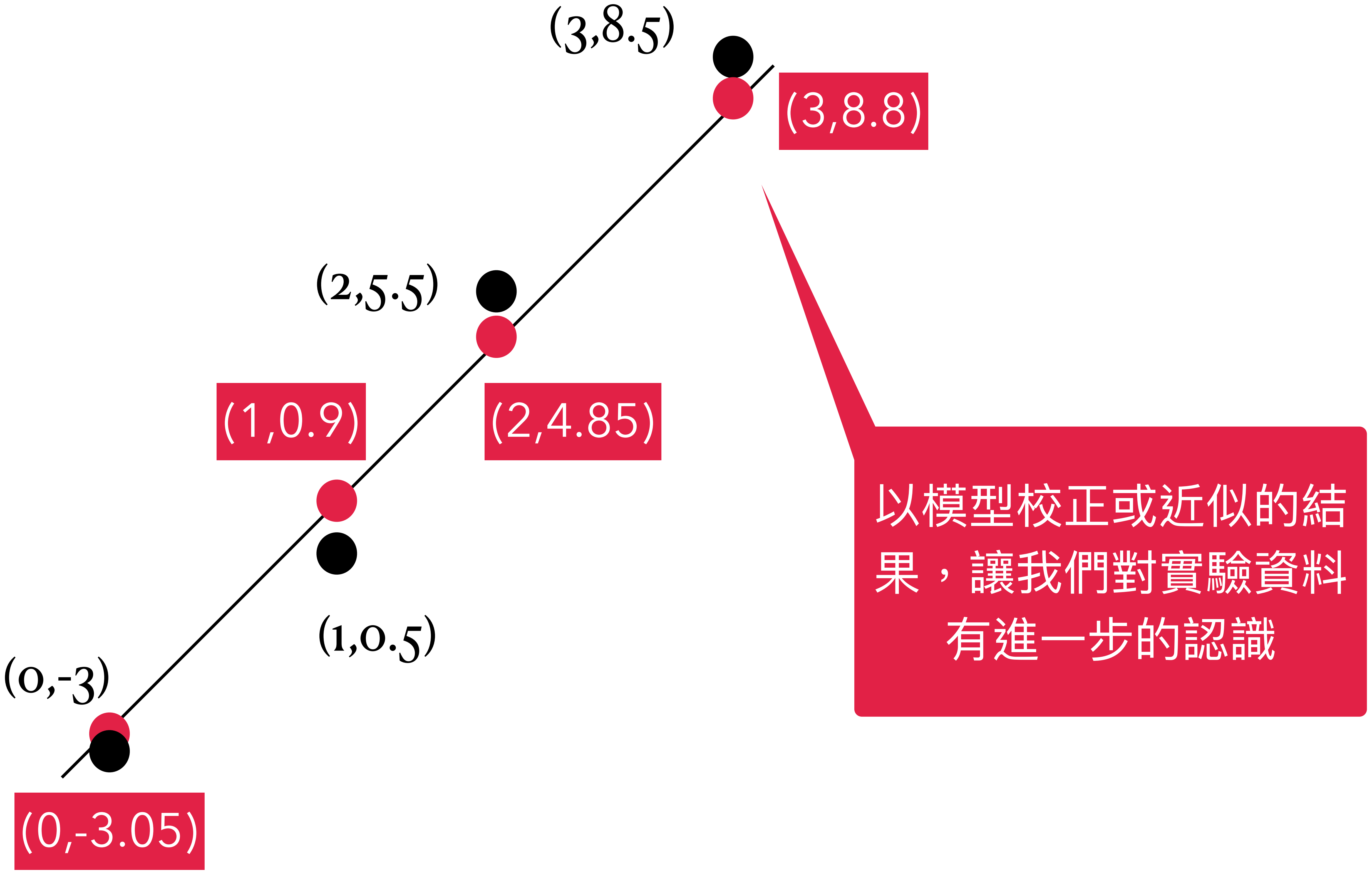


$$ax + d = y$$

$$a = 3.95, \quad d = -3.05$$

np.matrix([[2,1],[1,1],[0,1],[3,1]]) @ ans

```
matrix([[ 4.85],
        [ 0.9 ],
        [-3.05],
        [ 8.8 ]])
```


(3,8.5)
(3,8.8)
(2,5.5)
(1,0.9)
(2,4.85)
以模型校正或近似的結果，讓我們對實驗資料有進一步的認識
(1,0.5)
(0,-3)
(0,-3.05)

方法

$$\overset{A^t}{\begin{bmatrix} 2 & 1 & 0 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}} \overset{A}{\begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 0 & 1 \\ 3 & 1 \end{bmatrix}} \begin{bmatrix} a \\ d \end{bmatrix} = \overset{A^t}{\begin{bmatrix} 2 & 1 & 0 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}} \overset{b}{\begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}}$$

$$\overset{B = A^t A}{\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix}} \begin{bmatrix} a \\ d \end{bmatrix} = \overset{A^t}{\begin{bmatrix} 2 & 1 & 0 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}} \overset{b}{\begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}}$$

$$\begin{bmatrix} a \\ d \end{bmatrix} = \overbrace{inv\left(\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix}\right)}^{B^{-1}} \overbrace{\begin{bmatrix} 2 & 1 & 0 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}}^{A^t} \overbrace{\begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}}^{b}$$

$$\begin{bmatrix} a \\ d \end{bmatrix} = \underbrace{inv\left(\begin{bmatrix} 14 & 6 \\ 6 & 4 \end{bmatrix}\right)}_{B^{-1}} \underbrace{\begin{bmatrix} 2 & 1 & 0 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{A^t} \underbrace{\begin{bmatrix} 5.5 \\ 0.5 \\ -3 \\ 8.5 \end{bmatrix}}_{b}$$

```
B = AT @ A
invB = inv(B)
ans = invB @ AT @ b
```

步驟一：匯入套裝

```
import numpy as np
from numpy.linalg import inv
```

步驟二：將資料以矩陣A向量b表示

```
A = np.matrix([[2,1],[1,1],[3,1],[0,1]])
b = np.matrix([[5.5],[0.5],[8.5],[-3]])
```

步驟三：求轉置矩陣

```
AT = np.transpose(A)
```

步驟四：求最佳參數

```
B = AT @ A
invB = inv(B)
ans = invB @ AT @ b
```

製作模擬的實驗資料

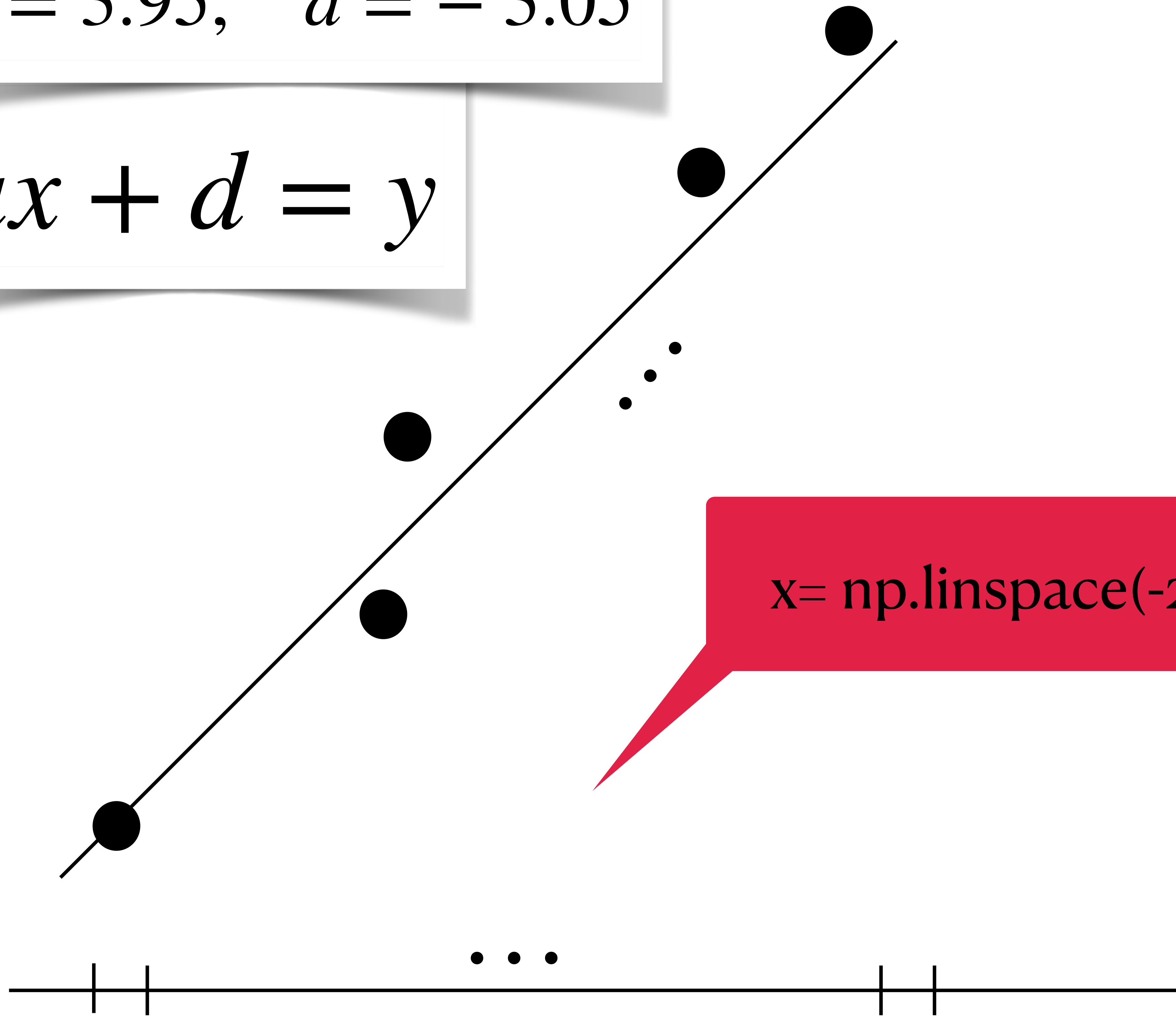
可以擴充到更多點？

在$[-2\pi, 2\pi]$的區間取50點

對應到直線上，取值加雜訊

$$a = 3.95, \quad d = -3.05$$

$$ax + d = y$$

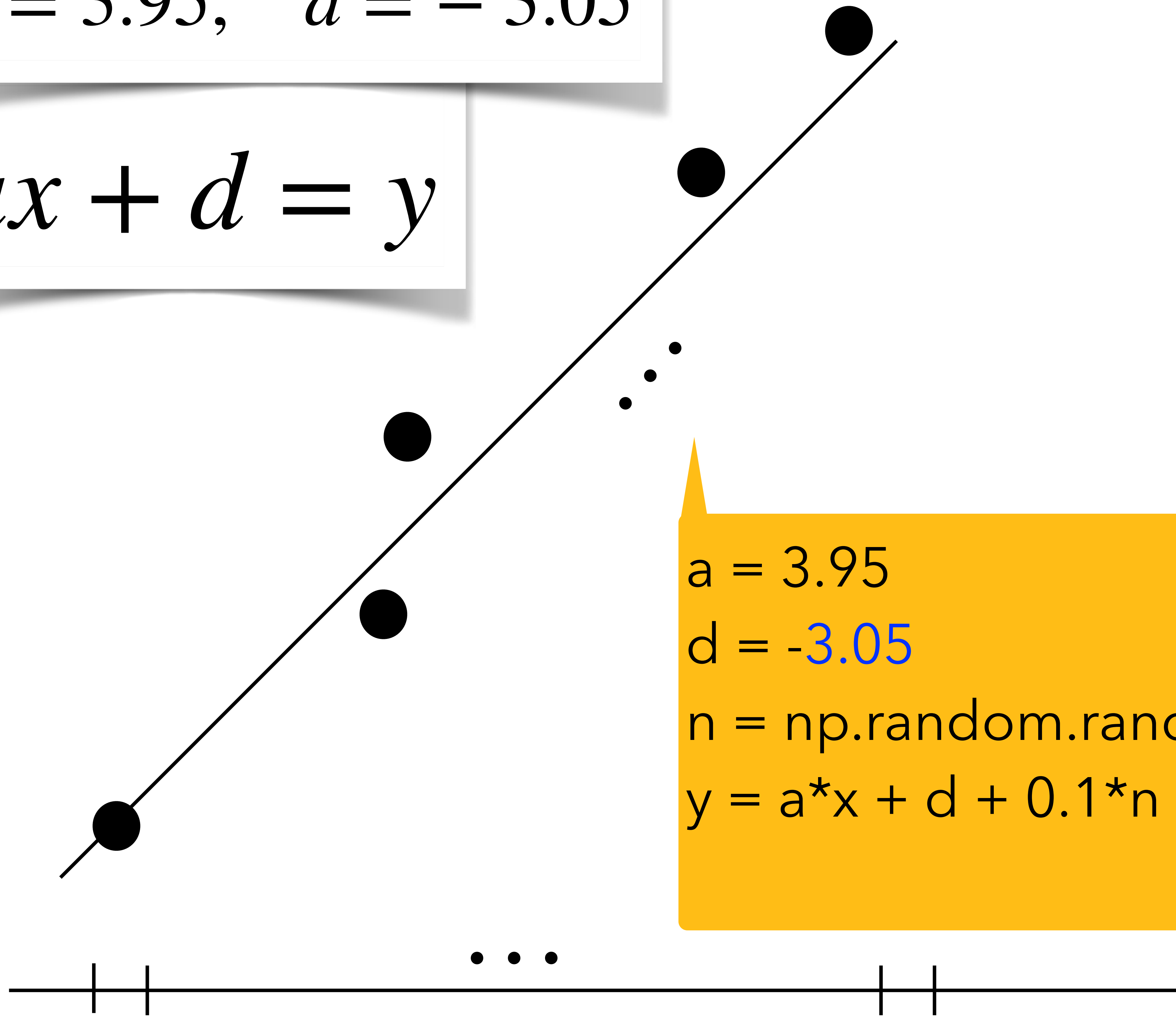


x= np.linspace(-2*np.pi,2*np.pi)

$$[-2\pi, 2\pi]$$

$$a = 3.95, \quad d = -3.05$$

$$ax + d = y$$



```
a = 3.95
d = -3.05
n = np.random.rand(1,50)-0.5
y = a*x + d + 0.1*n
```

$$[-2\pi, 2\pi]$$

```
x= np.linspace(-2*np.pi,2*np.pi)
```

```
a = 3.95
d = -3.05
n = np.random.rand(1,50)-0.5
x = np.linspace(-2*np.pi,2*np.pi)
y = a*x + d + 0.1 * n
```

```
vx = np.matrix(x)
vy = np.matrix(y)
```

```
A = np.hstack((np.transpose(vx), np.ones((50,1))))
b = np.transpose(vy)
```

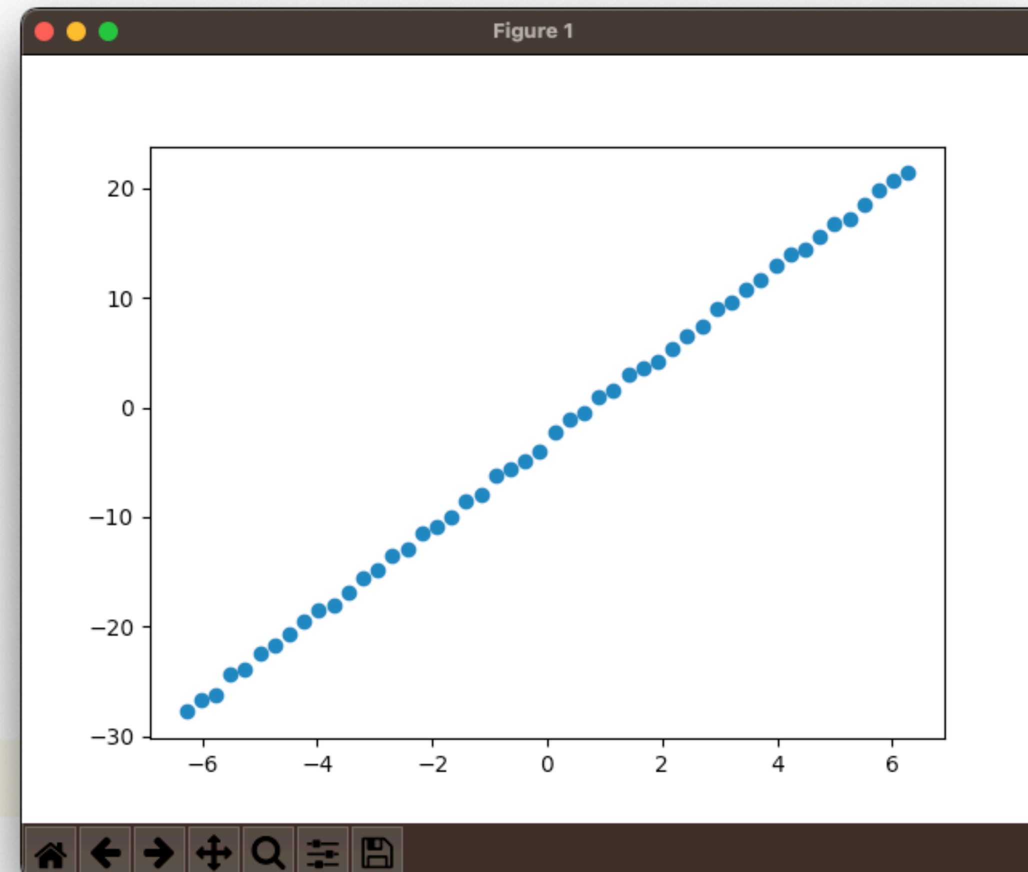
如何使用步驟三、步驟四求直線參數？

ex2022_ex1 › demo_noiseData.py

```python
import numpy as np
import matplotlib.pyplot as plt

a = 3.95
d = -3.05
x = np.linspace(-2*np.pi,2*np.pi)
print(np.shape(x))
n = np.random.rand(1,50)-0.5
y = a * x+d+ n

plt.figure()
plt.scatter(x, y, marker = 'o')
plt.show()
```



```
AT = np.transpose(A)
B = AT @ A
invB = inv(B)
ans = invB @ (AT @ b)
```

A plane with seven parameters

Sampling data

- $y = ax_1 + bx_2 + cx_3 + dx_4 + ex_5 + fx_6 + g$

```
import numpy as np
from numpy.linalg import inv
a = 3.95
b = 1.2
c = -2.5
d = -3.05
e = 2
f = -1
g = -1.5
dim = 5
dataSize = 500
x = np.random.rand(6,500)
print(x.shape)
y = np.array([[a,b,c,d,e,f]]) @ x + g
print(y.shape)
```

$$
\overbrace{\begin{bmatrix} x_1[1] & x_2[1] & x_3[1] & x_4[1] & x_5[1] & 1 \\ x_1[2] & x_2[2] & x_3[2] & x_4[1] & x_5[2] & 1 \\ \cdots & & & & & \\ x_{[}500] & x_2[500] & x_3[500] & x_4[500] & x_5[500] & 1 \end{bmatrix}}^{A}
\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix}
=
\overbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{500} \end{bmatrix}}^{y}
$$

$$
A^t \ A \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix} = A^t \ y
$$

$$A^t \; A \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix} = A^t \; y$$

$$\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{bmatrix} = (A^t A)^{-1} A^t \; y$$

Solve by matrix inversion

$$A = \begin{bmatrix} x_1[1] & x_2[1] & x_3[1] & x_4[1] & x_5[1] & 1 \\ x_1[2] & x_2[2] & x_3[2] & x_4[1] & x_5[2] & 1 \\ \dots & & & & & \\ x_{[}500] & x_2[500] & x_3[500] & x_4[500] & x_5[500] & 1 \end{bmatrix}$$

```
A = np.hstack((np.transpose(x),np.ones((dataSize,1))*g))
print(A.shape)
AT = np.transpose(A)
ans = inv(AT @ A) @ ( AT @ np.transpose(y))
print(ans)
```

$$(A^t A)^{-1} A^t \ y$$

```python
A = np.hstack((np.transpose(x),np.ones((dataSize,1))*g))
print(A.shape)
AT = np.transpose(A)
ans = inv(AT @ A) @ ( AT @ np.transpose(y))
print(ans)
```

thon Console

```
[[ 3.95]
 [ 1.2 ]
 [-2.5 ]
 [-3.05]
 [ 2.  ]
 [-1.  ]
 [ 1.  ]]
```

Solve by conjugate gradient method

```
import numpy as np
def conjgrad(A, b, x0, tol):
    r = b - A @ x0
    p = r
    rsold = np.transpose(r) @ r
    x = x0
    count = 0
    while np.sqrt(rsold) > tol:
        Ap = A @ p
        alpha = rsold /(np.transpose(p) @ Ap)
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = np.transpose(r) @ r
        p = r + (rsnew/rsold) * p
        rsold = rsnew
        count += 1
        if count % 2000 == 0:
            print('loop ', count)
            break
    return x
```

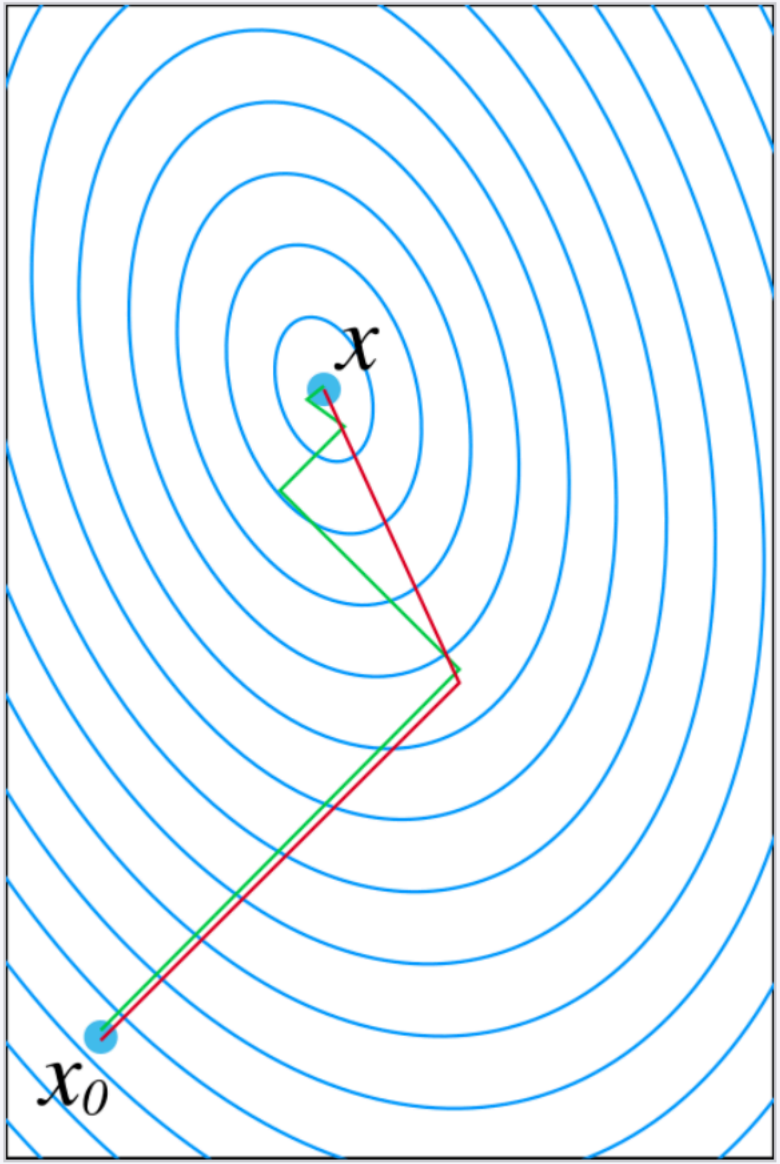
conjgrad.py

Conjugate gradient method

Article Talk

From Wikipedia, the free encyclopedia

In mathematics, the **conjugate gradient method** is an algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is positive-semidefinite. The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods such as the Cholesky decomposition. Large sparse systems often arise when numerically solving partial differential equations or optimization problems.



A comparison of the convergence of gradient descent with optimal step size (in green) and conjugate vector (in red) for minimizing a quadratic function associated with a given linear system. Conjugate gradient, assuming exact arithmetic, converges in at most *n* steps, where *n* is the size of the matrix of the system (here *n* = 2).

```python
from conjgrad import *
b = np.transpose(y)
ans =  conjgrad(AT @ A,AT @ np.transpose(y), np.zeros((7,1)), 10**-8)
print(ans)
```

```
[[ 3.95]
 [ 1.2 ]
 [-2.5 ]
 [-3.05]
 [ 2.  ]
 [-1.  ]
 [ 1.  ]]
```

```python
from conjgrad import *
import time
n = 10000
m = 30
AA = np.random.randn(n,m)
A = AA @ np.transpose(AA) + np.eye(n)
x = np.random.rand(n,1)
b = A@x
t = time.time()
ans = conjgrad(A,b,np.zeros((n,1)),10**-8)
tt = t = time.time() - t
print('mean abs error : ', np.mean(abs(A @ ans - b)))
print('execution time :', tt)
```

A is symmetric
A : 10000x10000

conjgrad is accurate and fast

```
mean abs error :  1.5202749636955558e-11
execution time : 0.701566934585571З
```

```
from conjgrad import *
import time
n = 10000
m = 30
AA = np.random.randn(n,m)
A = AA @ np.transpose(AA) + np.eye(n)
x = np.random.rand(n,1)
b = A@x
t = time.time()
ans = inv(A)*b
tt = t = time.time() - t
print('mean abs error : ', np.mean(abs(A @ ans - b)))
print('execution time :', tt)
```

A is symmetric

A : 10000x10000

inv is slow

execution time : 26.37588119506836