Assignment 3

1) $(a_i, b_i),\ i=1, \dots m$

feat label

$w(a_i) \approx b_i,\ i=1 \dots m$

↓

degree p polynomial

$b_i = w_p a_i^p + w_{p-1} a_i^{p-1} + \dots + w_1 a_i + w_0$

a) ~~$b_i = w_p a_i^p + w_{p-1} a_i^{p-1} + \dots + w_1 a_i + w_0$~~

a_i

?? ~~$m \times (p+1)\ (p+1) \times 1\ m \times 1$~~

b) $Ax = d$, $A = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}$, $d = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \overset{A}{\begin{bmatrix} a_1^p & a_1^{p-1} & \dots & 1 \\ a_2^p & a_2^{p-1} & \dots & 1 \\ \vdots \\ a_m^p & a_m^{p-1} & \dots & 1 \end{bmatrix}} \overset{x}{\begin{bmatrix} w_p \\ w_{p-1} \\ \vdots \\ w_0 \end{bmatrix}}$

$a_i\ w\ b_i$

$w(a_i) \approx b_i$

$m \times (p+1)$ $(p+1) \times 1$ $m \times 1$

?? in term of a_i

2) a) $\min_w \|x - Tw\|^2$, T is $n \times r$ matrix, orthonormal columns

?? not involve a matrix inverse $T^T = T^{-1}$

$w = (T^T T)^{-1} T^T x = (T^T)^{-1} T^T T^T x = T T^T T^T x$

b) $X: n \times p$, $[x_1, x_2 \dots x_p]$, $W = [w_1\ w_2 \dots w_p]$

$\min_{w_i} \|x_i - Tw_i\|_2^2$, $X \approx TW$

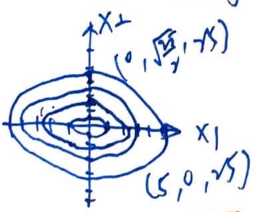
$W = (T^T T)^{-1} T^T X$?? → $W = T T^T T^T X$

4) $Q = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ a) $x^T Q x = \underset{1\times 2}{[x_1\ x_2]} \underset{2\times 2}{\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}} \underset{2\times 1}{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}} = \overset{[x_1\ 2x_2]}{\cancel{\begin{bmatrix} x_1 \\ 2x_2 \end{bmatrix}}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \boxed{x_1^2 + 2x_2^2 > 0 \to Q > 0}$

(PD)

b) $y = x^T Q x$, $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to y = \underset{1\times 2}{[x_1\ x_2]} \underset{2\times 2}{\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}} \underset{2\times 1}{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}$

$\to y = [x_1\ 2x_2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 + 2x_2^2$



$(1, 1, 5)$ $(0, \sqrt{\frac{5}{2}}, 5)$ 1.58 $(0, \sqrt{\frac{9}{2}}, 9)$ 2.1

$(1, 0, 1)$ $(\sqrt{5}, 0, 5)$ 2.23 $(3, 0, 9)$

$(0, \frac{1}{\sqrt{2}}, 1)$ 0.7 $(5, 0, 25)$

$(0, \sqrt{\frac{25}{2}}, 25)$ 3.53

5. $P > 0$, $Q > 0$ are $n \times n$ matrices

$QPQ > 0$??? symmetric PD

↑

$x^T QPQ x > 0$ $P^T = P$, $Q^T = Q$

$a^T P a > 0$, if $a = Qx$

$\to (Qx)^T PQx > 0 \to x^T Q^T PQx > 0 \to x^T QPQx > 0$

least-square

February 21, 2024

1 1. Polynomial fitting

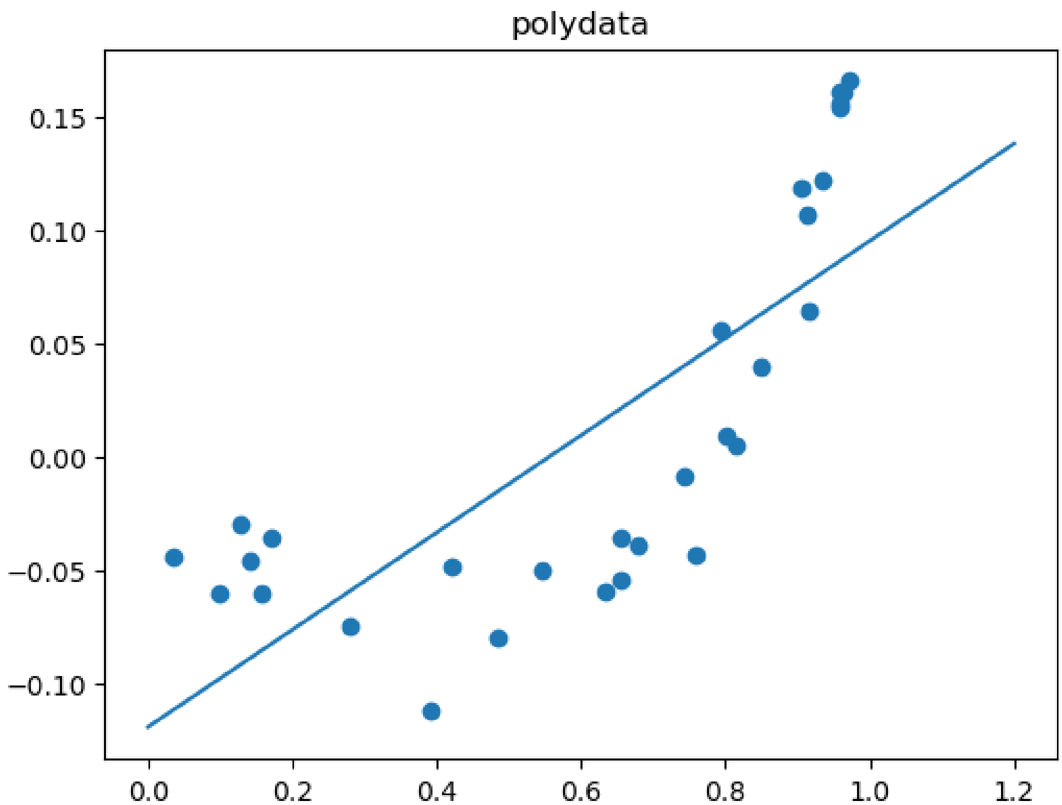
1.1 c)

[15]:
```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
import sympy
from sympy.abc import x

in_data = loadmat('polydata.mat')
print([key for key in in_data]) # -- use this line to see the keys in the␣
↪dictionary data structure

a = in_data['a']
b = in_data['b']
n_a = np.size(a)
# print(n_a)
# print(np.size(b))

# plt.scatter(a,b)
# plt.show()
```

```
['__header__', '__version__', '__globals__', 'a', 'b']
```

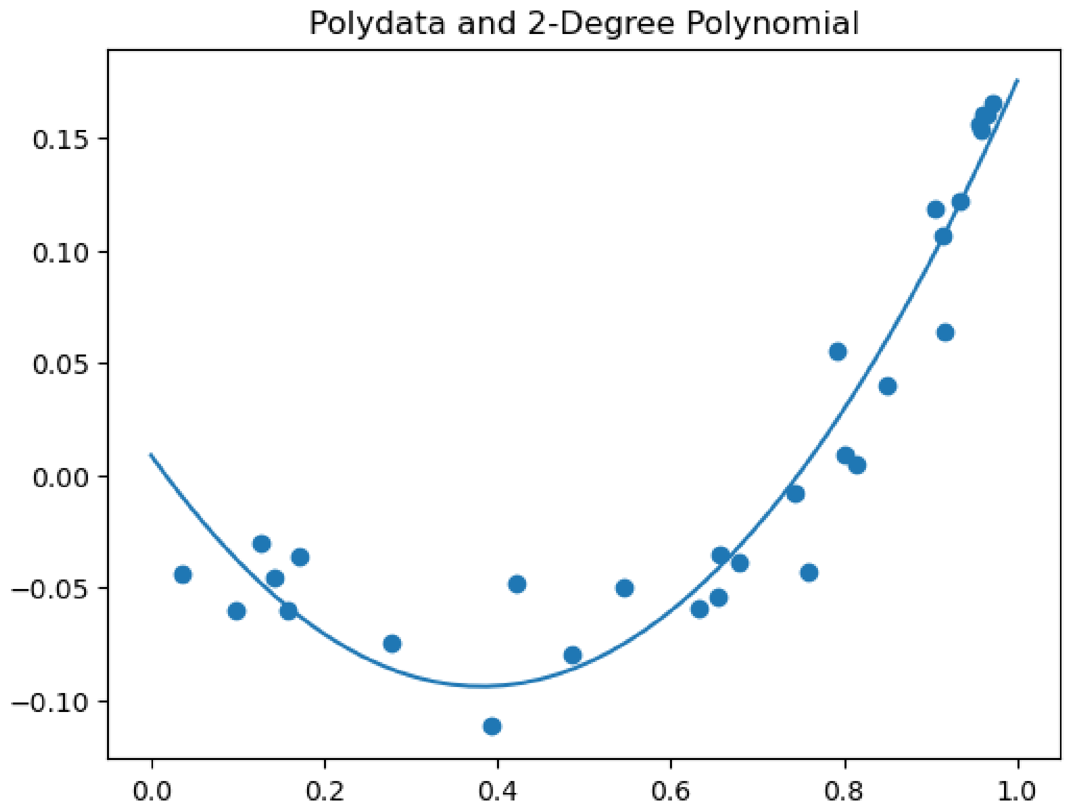
[16]:
```
# 1-degree polynomial (add 1 as the constant)
a_1 = np.hstack((a, np.ones((n_a,1))))

# least squares as a loss function
# w = (X^T X)^(-1)X^T y
w = np.linalg.inv(a_1.transpose()@a_1)@a_1.transpose()@b
# print(w[0][0], w[1][0])
w1 = w[0][0]
w2 = w[1][0]
f = w1* x + w2
X = np.linspace(0, 1.2)
Y = sympy.lambdify(x, f, "numpy")(X)
```

```
plt.plot(X, Y)
plt.scatter(a,b)
plt.title('Polydata and 1-Degree Polynomial')
plt.show()
```



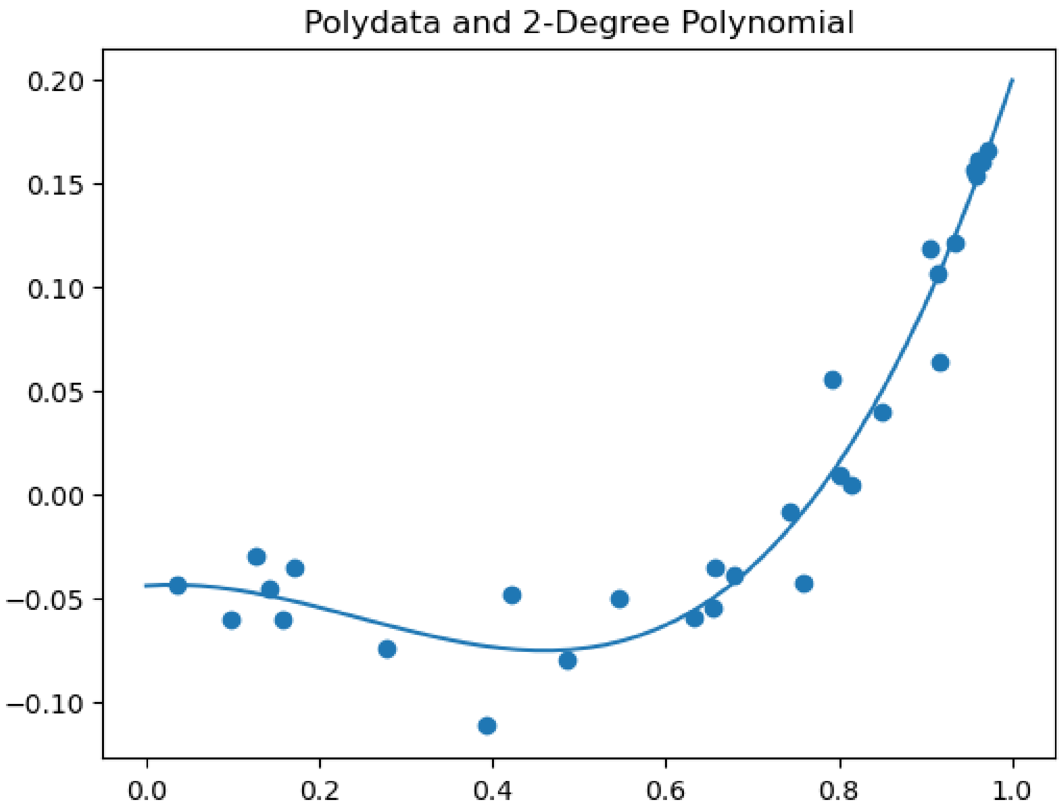
[20]:
```
# 2-degree polynomial
a_2 = np.hstack((a**2, a, np.ones((n_a,1))))

# least squares as a loss function
# w = (X^T X)^(-1)X^T y
w = np.linalg.inv(a_2.transpose()@a_2)@a_2.transpose()@b
# print(w)
w1 = w[0][0]
w2 = w[1][0]
w3 = w[2][0]
f = w1*x**2 + w2*x + w3
X = np.linspace(0, 1)
Y = sympy.lambdify(x, f, "numpy")(X)

plt.plot(X, Y)
plt.scatter(a,b)
```

```
plt.title('Polydata and 2-Degree Polynomial')
plt.show()
```



[21]:
```
# 3-degree polynomial
a_3 = np.hstack((a**3, a**2, a, np.ones((n_a,1))))

# least squares as a loss function
# w = (X^T X)^(-1)X^T y
w = np.linalg.inv(a_3.transpose()@a_3)@a_3.transpose()@b
# print(w)
w1 = w[0][0]
w2 = w[1][0]
w3 = w[2][0]
w4 = w[3][0]
f = w1*x**3 + w2*x**2 + w3*x + w4
X = np.linspace(0, 1)
Y = sympy.lambdify(x, f, "numpy")(X)

plt.plot(X, Y)
plt.scatter(a,b)
plt.title('Polydata and 2-Degree Polynomial')
```

```
plt.show()
```


Polydata and 2-Degree Polynomial
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
0.0
0.2
0.4
0.6
0.8
1.0

gram-schmidt-orth

February 21, 2024

1 3. Movies Rating

[57]:
```
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

def gram_schmidt(B):
    """Orthogonalize a set of vectors stored as the columns of matrix B."""
    # Get the number of vectors.
    m, n = B.shape
    # Create new matrix to hold the orthonormal basis
    U = np.zeros([m,n])
    for j in range(n):
        # To orthogonalize the vector in column j with respect to the
        # previous vectors, subtract from it its projection onto
        # each of the previous vectors.
        v = B[:,j].copy()
        for k in range(j):
            v -= np.dot(U[:, k], B[:, j]) * U[:, k]
        if np.linalg.norm(v)>1e-10:
            U[:, j] = v / np.linalg.norm(v)
    return U

# if __name__ == '__main__':
#     B1 = np.array([[1.0, 1.0, 0.0], [2.0, 2.0, 0.0], [2.0, 2.0, 1.0]])
#     A1 = gram_schmidt(B1)
#     print(A1)
#     A2 = gram_schmidt(np.random.rand(4,2)@np.random.rand(2,5))
#     print(A2.transpose()@A2)
```

1.0.1 a) Is the first basis vector you obtain equal to t1?

Yes, it is the same as t1.

[58]:
```
in_data = loadmat('movie.mat')
# print([key for key in in_data])
```

```
X = in_data['X']
# print(X.shape[0])
X_aug = np.hstack((np.ones((X.shape[0],1)), X))
# print(X_aug.shape)
# print(X_aug)

X_gram = gram_schmidt(X_aug)
print(X_gram)
print(5 ** (-1/2))
print(X_gram[0][0] == 5 ** (-1/2))
```

```
[[ 4.47213595e-01 -3.65148372e-01 -6.32455532e-01 -5.16397779e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00  1.26565425e-14]
 [ 4.47213595e-01  5.47722558e-01  3.16227766e-01 -3.87298335e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00  5.00000000e-01]
 [ 4.47213595e-01 -3.65148372e-01  2.80866677e-16  6.45497224e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00  5.00000000e-01]
 [ 4.47213595e-01  5.47722558e-01 -3.16227766e-01  3.87298335e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -5.00000000e-01]
 [ 4.47213595e-01 -3.65148372e-01  6.32455532e-01 -1.29099445e-01
   0.00000000e+00  0.00000000e+00  0.00000000e+00 -5.00000000e-01]]
0.4472135954999579
True
```

1.0.2 b) rank-1 approximation

[59]:
```
# t1 = 5 ** (-1/2) * np.array([[1],[1],[1],[1],[1]])
# print(t1)
t1 =  np.array([X_gram[:, 0]]).transpose()
print(t1)

# least squares as a loss function
# w = (X^T X)^(-1)X^T y
W = np.linalg.inv(t1.transpose()@t1)@t1.transpose()@X
print('W:\n', W)

t1W = t1 @ W
print('t1W:\n', t1W)

print('X - t1W:\n', X - t1W)
# # e = d^T d - d^T A (A^T A)^(-1) A^T d
# e = X.transpose()@X - X.transpose() @ t1 @ np.linalg.inv(t1.transpose() @ t1)␣
↪@ t1.transpose() @ X
# print(e) # --> ???
```

```
[[0.4472136]
 [0.4472136]
```

```
 [0.4472136]
 [0.4472136]
 [0.4472136]]
W:
 [[13.41640786 12.96919427  8.04984472 15.20526225 17.44133022  9.8386991
   5.81377674]]
t1W:
 [[6.  5.8 3.6 6.8 7.8 4.4 2.6]
 [6.  5.8 3.6 6.8 7.8 4.4 2.6]
 [6.  5.8 3.6 6.8 7.8 4.4 2.6]
 [6.  5.8 3.6 6.8 7.8 4.4 2.6]
 [6.  5.8 3.6 6.8 7.8 4.4 2.6]]
X - t1W:
 [[-2.   1.2 -1.6  1.2 -0.8 -0.4 -0.6]
 [ 3.  -2.8  1.4 -0.8  2.2  0.6  2.4]
 [-2.   2.2 -0.6  0.2 -1.8 -0.4 -1.6]
 [ 3.  -3.8  2.4 -1.8  1.2  0.6  1.4]
 [-2.   3.2 -1.6  1.2 -0.8 -0.4 -1.6]]
```

1.0.3 c) rank-2 approximation

[60]:
```
T2 =  np.array(X_gram[:, :2])
# print(T)

# least squares as a loss function
# w = (X^T X)^(-1)X^T y
W = np.linalg.inv(T2.transpose()@T2)@T2.transpose()@X
print('W:\n', W)

T2W = T2 @ W
print('T2W:\n', T2W)

print('X - T2W:\n', X - T2W)
```

```
W:
 [[13.41640786 12.96919427  8.04984472 15.20526225 17.44133022  9.8386991
   5.81377674]
 [ 5.47722558 -6.02494813  3.46890953 -2.37346442  3.10376116  1.09544512
   3.46890953]]
T2W:
 [[4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [9.         2.5        5.5        5.5        9.5        5.
  4.5       ]
 [4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [9.         2.5        5.5        5.5        9.5        5.
  4.5       ]
```

```
 [4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]]
X - T2W:
 [[-8.88178420e-16 -1.00000000e+00 -3.33333333e-01  3.33333333e-01
   3.33333333e-01  0.00000000e+00  6.66666667e-01]
 [-1.77635684e-15  5.00000000e-01 -5.00000000e-01  5.00000000e-01
   5.00000000e-01 -8.88178420e-16  5.00000000e-01]
 [-8.88178420e-16  0.00000000e+00  6.66666667e-01 -6.66666667e-01
  -6.66666667e-01  0.00000000e+00 -3.33333333e-01]
 [-1.77635684e-15 -5.00000000e-01  5.00000000e-01 -5.00000000e-01
  -5.00000000e-01 -8.88178420e-16 -5.00000000e-01]
 [-8.88178420e-16  1.00000000e+00 -3.33333333e-01  3.33333333e-01
   3.33333333e-01  0.00000000e+00 -3.33333333e-01]]
```

1.0.4 d) rank-3 approximation

[61]:
```
T3 =  np.array(X_gram[:, :3])
# print(T3)

# least squares as a loss function
# w = (X^T X)^(-1)X^T y
W = np.linalg.inv(T3.transpose()@T3)@T3.transpose()@X
print('W:\n', W)

T3W = T3 @ W
print('T3W:\n', T3W)

print('X - T3W:\n', X - T3W)
```

```
W:
 [[ 1.34164079e+01  1.29691943e+01  8.04984472e+00  1.52052622e+01
   1.74413302e+01  9.83869910e+00  5.81377674e+00]
 [ 5.47722558e+00 -6.02494813e+00  3.46890953e+00 -2.37346442e+00
   3.10376116e+00  1.09544512e+00  3.46890953e+00]
 [-1.33226763e-15  1.58113883e+00 -3.16227766e-01  3.16227766e-01
   3.16227766e-01 -8.88178420e-16 -3.16227766e-01]]
T3W:
 [[4.         7.         2.53333333 7.46666667 6.46666667 4.
  1.53333333]
 [9.         3.         5.4        5.6        9.6        5.
  4.4       ]
 [4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [9.         2.         5.6        5.4        9.4        5.
  4.6       ]
 [4.         9.         2.13333333 7.86666667 6.86666667 4.
  1.13333333]]
X - T3W:
```

```
[[-2.66453526e-15 -8.88178420e-16 -5.33333333e-01  5.33333333e-01
   5.33333333e-01 -1.77635684e-15  4.66666667e-01]
 [-1.77635684e-15 -8.88178420e-16 -4.00000000e-01  4.00000000e-01
   4.00000000e-01 -8.88178420e-16  6.00000000e-01]
 [-1.77635684e-15  0.00000000e+00  6.66666667e-01 -6.66666667e-01
  -6.66666667e-01 -8.88178420e-16 -3.33333333e-01]
 [-1.77635684e-15 -8.88178420e-16  4.00000000e-01 -4.00000000e-01
  -4.00000000e-01 -8.88178420e-16 -6.00000000e-01]
 [-8.88178420e-16  0.00000000e+00 -1.33333333e-01  1.33333333e-01
   1.33333333e-01  0.00000000e+00 -1.33333333e-01]]
```

1.0.5 e) Suppose you interchange the order of Jake and Jennifer so that Jennifer's ratings are in the first column of X and Jake's ratings are in the second column. Does the rank-2 approximation change? Why or why not? Does the rank-3 approximation change? Why or why not?

Yes, both of them only alter the order of the first two columns, but the values within these columns remain identical. This is because we utilize the same taste matrix (T) to compute X and the version of X with the first two columns swapped.

[62]:
```
X_swap = X.copy()
X_swap[:, [1, 0]] = X_swap[:, [0, 1]]
# print(X)
# print(X_swap)



# X_swap = X_aug.copy()
# X_swap[:, [1, 0]] = X_swap[:, [0, 1]]
# print(X_aug)
# print(X_swap)

# X_gram2 = gram_schmidt(X_swap)
# print(X_gram)
# print(X_gram2)
```

[63]:
```
# least squares as a loss function
# w = (X^T X)^(-1)X^T y
W = np.linalg.inv(T2.transpose()@T2)@T2.transpose()@X_swap
# print('W:\n', W)

T2W_swap = T2 @ W
print('T2W_swap:\n', T2W_swap)
print('T2W:\n', T2W)
```

```
T2W_swap:
 [[8.         4.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [2.5        9.         5.5        5.5        9.5        5.
```

```
 4.5        ]
 [8.         4.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [2.5        9.         5.5        5.5        9.5        5.
  4.5        ]
 [8.         4.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]]
T2W:
 [[4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [9.         2.5        5.5        5.5        9.5        5.
  4.5        ]
 [4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [9.         2.5        5.5        5.5        9.5        5.
  4.5        ]
 [4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]]
```

[64]:
```
# least squares as a loss function
# w = (X^T X)^(-1)X^T y
W = np.linalg.inv(T3.transpose()@T3)@T3.transpose()@X_swap
# print('W:\n', W)

T3W_swap = T3 @ W
print('T3W_swap:\n', T3W_swap)
print('T3W:\n', T3W)
```

```
T3W_swap:
 [[7.         4.         2.53333333 7.46666667 6.46666667 4.
  1.53333333]
 [3.         9.         5.4        5.6        9.6        5.
  4.4        ]
 [8.         4.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [2.         9.         5.6        5.4        9.4        5.
  4.6        ]
 [9.         4.         2.13333333 7.86666667 6.86666667 4.
  1.13333333]]
T3W:
 [[4.         7.         2.53333333 7.46666667 6.46666667 4.
  1.53333333]
 [9.         3.         5.4        5.6        9.6        5.
  4.4        ]
 [4.         8.         2.33333333 7.66666667 6.66666667 4.
  1.33333333]
 [9.         2.         5.6        5.4        9.4        5.
  4.6        ]
```

```
[4.         9.         2.13333333 7.86666667 6.86666667 4.
 1.13333333]]
```