Module 3-1:
Linear Algebra I (Practice)

國立中山大學電機系
魏家博

Outline

1. Linear Equations
2. Least Squares Problem
   - Normal Equation
   - Gradient Descent
3. Point Set Registration
   - Matrix Calculus
   - Gradient Descent

Exercise 1: Solve Linear Equations

Solve $\mathbf{Ax} = \mathbf{b}$ with

a) $\mathbf{A} = \begin{bmatrix} 3 & 2 & -1 \\ 6 & -1 & 3 \\ 1 & 10 & -2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} -7 \\ -4 \\ 2 \end{bmatrix}$

b) $\mathbf{A} = \begin{bmatrix} 4 & -1 & 3 \\ 21 & -4 & 18 \\ -9 & 1 & -9 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 5 \\ 7 \\ -8 \end{bmatrix}$

c) $\mathbf{A} = \begin{bmatrix} 7 & -4 & 1 \\ 3 & 2 & -1 \\ 5 & 12 & -5 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} -15 \\ -5 \\ -5 \end{bmatrix}$

判斷上面線性方程式是否有解，若有解，再進一步判斷是唯一解或無窮多解。

Useful Propositions

Proposition 1. The following three statements are equivalent.

1. The linear system $\mathbf{Ax} = \mathbf{b}$ is consistent.
2. The vector $\mathbf{b}$ can be expressed as a linear combination of $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n$.
3. $\mathrm{rank}(\mathbf{A}) = \mathrm{rank}([\mathbf{A} \quad \mathbf{b}])$

Proposition 2. Suppose $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$. The following three statements are equivalent.

1. The linear system $\mathbf{Ax} = \mathbf{b}$ has at most one solution for every $\mathbf{b} \in \mathbb{R}^m$.
2. The column vectors of $\mathbf{A}$ are linearly independent.
3. $\mathrm{rank}(\mathbf{A}) = n$

Steps for Exercise 1

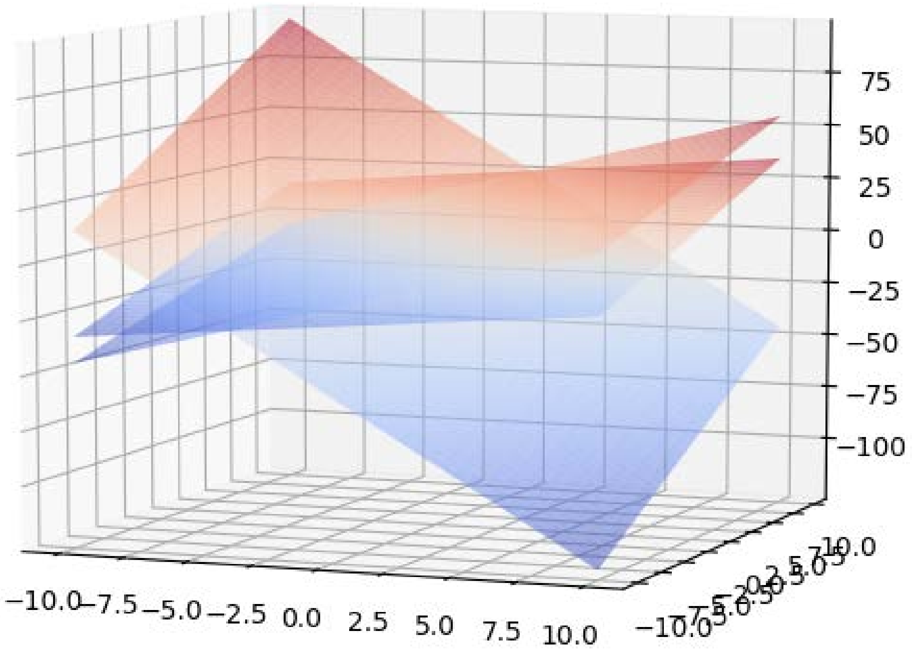
1. Open exercise_1a_start.py
2. Use np.array() to create array $\mathbf{A}$ and array $\mathbf{b}$.
3. Use np.linalg.matrix_rank() to calculate the rank of $\mathbf{A}$.
4. Use np.hstack() to create array $[\mathbf{A}, \mathbf{b}]$.
5. Use np.linalg.matrix_rank() to calculate the rank of $[\mathbf{A}, \mathbf{b}]$.
6. Use **Proposition 1** to determine whether $\mathbf{Ax} = \mathbf{b}$ is consistent.
7. If it is consistent, then use **Proposition 2** to determine whether $\mathbf{Ax} = \mathbf{b}$ has only one solution.
   - You can use A.shape[1] to get the column number of $\mathbf{A}$
8. If it has only one solution, use np.linalg.solve() to obtain the solution of $\mathbf{Ax} = \mathbf{b}$.

Steps for Exercise 1

9. Open exercise_1b_function.py
10. Write a function `solve_linear_equation()` that achieves the following requirements
    - Input: Array $\mathbf{A}$ and array $\mathbf{b}$
    - Output: 1 if $\mathbf{Ax} = \mathbf{b}$ has only one solution
    - 0 if $\mathbf{Ax} = \mathbf{b}$ has infinitely many solutions
    - -1 if $\mathbf{Ax} = \mathbf{b}$ has no solutions
11. Test your function with
    - `input_id = 'case1'`
    - `input_id = 'case2'`
    - `input_id = 'case3'`

Steps for Exercise 1

12. Open exercise_1c_draw.py
13. Draw the plane $\mathbf{Ax} = \mathbf{b}$ in 3D space as shown in the following figure.



Exercise 2: Least Squares Problem

Consider

$$\min_{\mathbf{x}} f(\mathbf{x}) = \min_{\mathbf{x}} ||\mathbf{Ax} - \mathbf{b}||_2^2 = \min_{\mathbf{x}} (\mathbf{Ax} - \mathbf{b})^T (\mathbf{Ax} - \mathbf{b})$$

where $\mathbf{A} = \begin{bmatrix} 1 & -2 \\ 1 & -2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

1. Use the normal equation to find the optimal solution, i.e., solve

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b} \implies \begin{bmatrix} 1 & -2 \\ 1 & -2 \end{bmatrix}^T \begin{bmatrix} 1 & -2 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ 1 & -2 \end{bmatrix}^T \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

   Find all x_1, x_2 that satisfy the above equation. In fact, there are infinitely many optimal solutions for this exercise.

Steps for Exercise 2

2. Open exercise_2b_gradient_descent.py
3. Implement the following pseudo code for the gradient descent method.

```
x = [−2,2]^T # initial condition
alpha = 0.02 #learning rate
max_iter = 1000
f = ||Ax − b||_2^2
for k in range(max_iter):
  print k, (x1,x2), f
        as shown in the right table
  x_prev = x
  x = x - alpha * ∇f(x)
  # ∇f(x) = 2(A^T Ax − A^T b)
  if ||x − x_prev|| ≤ 10^−8 then break
  f = ||Ax − b||_2^2
```

k	(x_1, x_2)	$f(\mathbf{x})$
0	(-2, 2)	113.00
1	(-1.4, 0.8)	41.00
2	(-1.,04, 0.08)	15.08
3	(-0.69, -0.61)	2.39
4	(-0.62, -0.77)	1.18
5	(-0.57, -0.86)	0.74
⋮	⋮	⋮
37	(-0.5, -1.0)	0.5

最佳解
印出小數點後4位

最小值
印出小數點後8位

Steps for Exercise 2

4. Hints for the pseudo code:
   - You can use np.dot(A, b) or A.dot(b) to perform matrix multiplication $\mathbf{Ab}$
   - You can use A.T to obtain $\mathbf{A}^T$
   - You can use np.linalg.norm(v) to calculate $\|\mathbf{v}\|_2$
   - You can use x**2 to calculate $\mathbf{x}^2$
   - Learn how to set the decimal precision for print(). There are two styles.
     - Old style (Python 2.7): http://interactivepython.org/runestone/static/pip/StringFormatting/interpolation.html
     - New style (Python 3.7): https://pyformat.info/

Steps for Exercise 2

5. Answer the following questions.
   a) How many iterations are required to reach the optimal solution? 需要跑幾次迴圈才能找到最佳解?
   b) Try different learning rates of alpha. How to adjust the learning rate for faster speed of convergence? 如何調整 alpha 使得所需迴圈次數愈少愈好(即較快達到收斂)？
   c) Try different initial points of x. Do different initial points give rise to different to optimal solutions? 不同初始點 x 會使演算法找到不同最佳解嗎?
   d) Does the minimum vary with the values of the initial point? 不同初始點 x 會使演算法找到不同最小值嗎?
   e) Is the minimum a local minimum or a global minimum?

Steps for Exercise 2

6. Open exercise_2c_draw.py
7. Let ax = plt.gca(). Use ax.scatter() to plot the point x of each iteration in the 3D space as shown in Figure 1 (draw the red points)
8. Draw all optimal solutions calculated in Step 1 of Exercise 2 as shown in Figure 2 (draw the blue line).

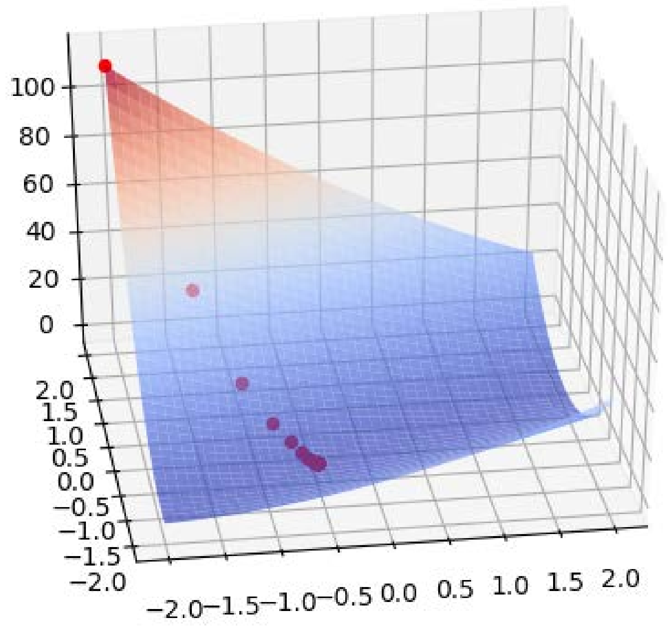


Figure 1

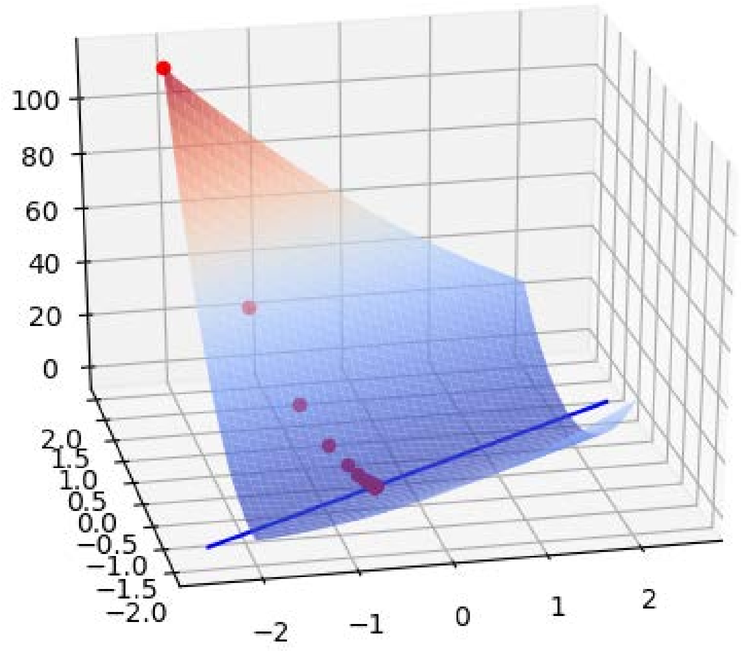


Figure 2

Exercise 3: Point Set Registration

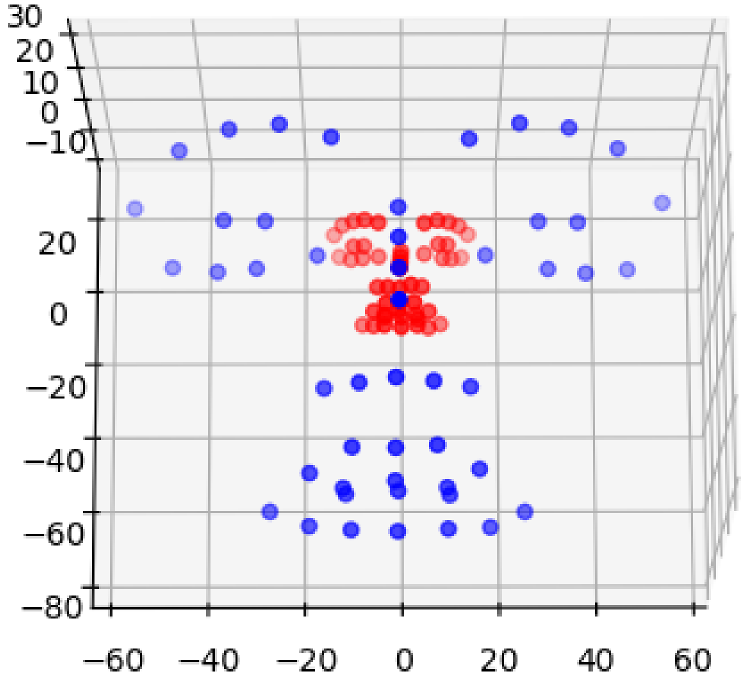


Figure 1

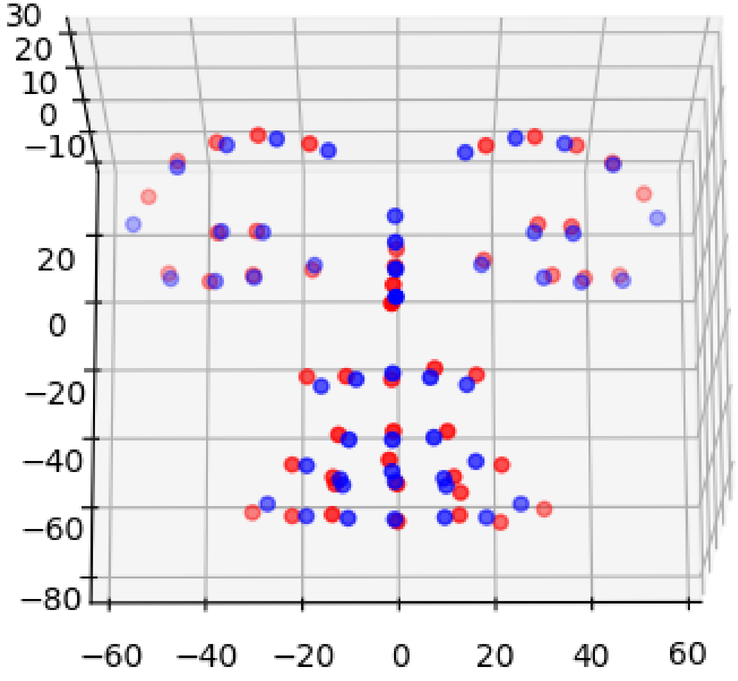


Figure 2

Exercise 3: Point Set Registration

Let $\mathbf{P}$ and $\mathbf{Q}$ be two given matrices in $\mathbb{R}^{3\times 49}$. Each column of $\mathbf{P}$ (or $\mathbf{Q}$) represents a point in $\mathbb{R}^3$. Thus, column vectors of $\mathbf{P}$ (or $\mathbf{Q}$) represent 49 points in $\mathbb{R}^3$.

In Figure 1, the blue points are column vectors of $\mathbf{P}$, while the red points are column vectors of $\mathbf{Q}$. Our goal is to search for a scaling factor $s \in \mathbb{R}$ such that points represented by $s\mathbf{Q}$ are as close as possible to points represented by $\mathbf{P}$, as shown in Figure 2. To be precise, we aim to solve the following minimization problem

$$\min_s \sum_{i=1}^{49} \|\mathbf{p}_i - s\mathbf{q}_i\|_2^2 = \min_s \sum_{i=1}^{49} (\mathbf{p}_i - s\mathbf{q}_i)^T(\mathbf{p}_i - s\mathbf{q}_i) = \min_s f(s)$$

where $\mathbf{p}_\mathrm{i} \in \mathbb{R}^3$ is the ith column of $\mathbf{P}$, and $\mathbf{q}_\mathrm{i} \in \mathbb{R}^3$ is the ith column of $\mathbf{Q}$.

Steps for Exercise 3

$$f(s) = \sum_{i=1}^{49} (\mathbf{p}_i - s\mathbf{q}_i)^T (\mathbf{p}_i - s\mathbf{q}_i)$$

1. Calculate $\partial f / \partial s$. Set $\partial f / \partial s = 0$ to compute the optimal solution $\hat{s}$. (變數 s 是純量，僅需用到高中微積分)
2. Open exercise_3a_point_set_registration.py
3. Compute $\hat{s} = \left(\sum_{i=1}^{49} \mathbf{p}_i^T \mathbf{q}_i\right)\left(\sum_{i=1}^{49} \mathbf{q}_i^T \mathbf{q}_i\right)^{-1}$

   After computing $\hat{s}$, you can use `draw_landmarks`$(\mathbf{P}, \hat{s}\mathbf{Q})$ to visualize the result.

Steps for Exercise 3

4. Compute the mean distance error $d(s)$ for $s = 1$ and $s = \hat{s}$, where the mean distance error $d(s)$ is defined as

$$d(s) = \frac{1}{49}\left(\sum_{i=1}^{49} \sqrt{(\mathbf{p}_i - s\mathbf{q}_i)^T(\mathbf{p}_i - s\mathbf{q}_i)}\right)$$

   a) Using the for loop

   b) Without using the for loop (this method is called vectorization)

   Calculate $\mathbf{E} = \mathbf{P} - s\mathbf{Q}$

   You only need np.sum(), np.sqrt(), element-wise multiply * to complete this task.

$$\underbrace{\begin{bmatrix} | & | & & | \\ e_1 & e_2 & \cdots & e_{49} \\ | & | & & | \end{bmatrix}\begin{bmatrix} | & | & & | \\ e_1 & e_2 & \cdots & e_{49} \\ | & | & & | \end{bmatrix}}_{\mathbf{E} * \mathbf{E}} = \begin{bmatrix} | & | & & | \\ h_1 & h_2 & \cdots & h_{49} \\ | & | & & | \end{bmatrix} \xrightarrow{\text{np.sum(, axis=0)}} [s_1 \quad s_2 \quad \cdots \quad s_{49}] \xrightarrow{\text{np.sum(np.sqrt()) / 49}} d(s)$$

Exercise 4: Point Set Registration

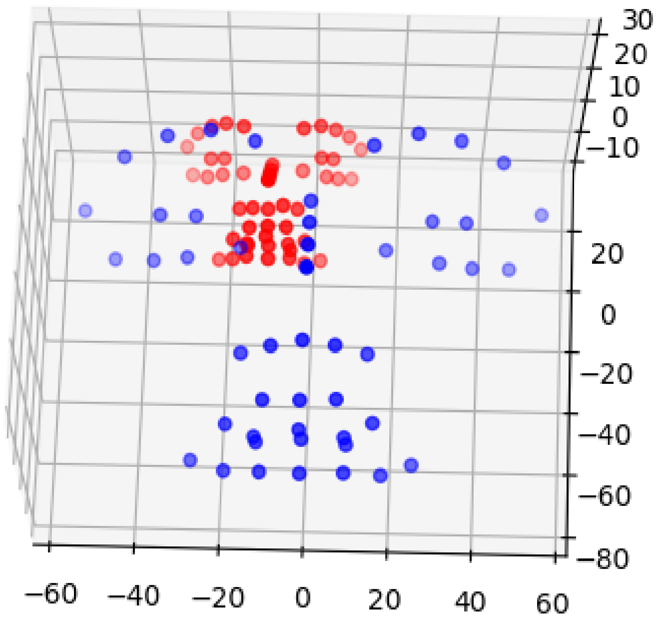


Figure 3

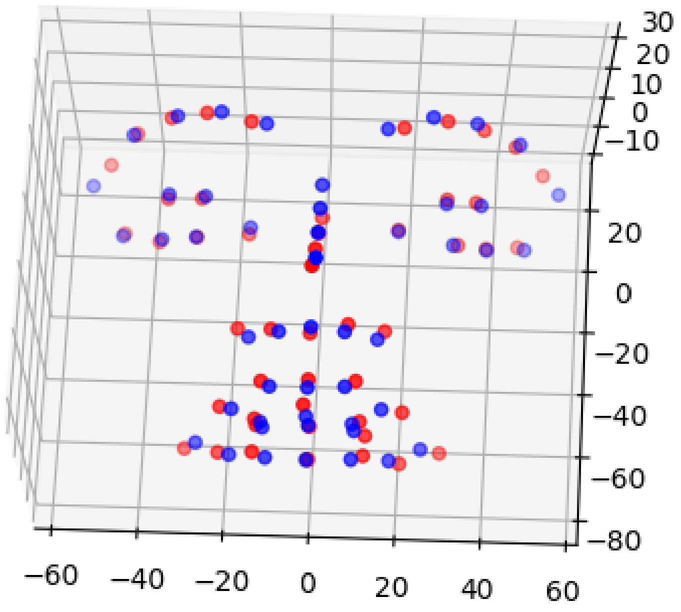


Figure 4

Exercise 4: Point Set Registration

Let $\mathbf{P}$ and $\mathbf{Q}$ be two given matrices in $\mathbb{R}^{3\times 49}$. Each column of $\mathbf{P}$ (or $\mathbf{Q}$) represents a point in $\mathbb{R}^3$. Thus, column vectors of $\mathbf{P}$ (or $\mathbf{Q}$) represent 49 points in $\mathbb{R}^3$.

In Figure 3, the blue points are column vectors of $\mathbf{P}$, while the red points are column vectors of $\mathbf{Q}$. Our goal is to search for a scaling factor $s \in \mathbb{R}$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$ such that $s\mathbf{q}_i + \mathbf{t}$ is as close as possible to $\mathbf{p}_i$ for $i = 1,2,\ldots,49$, as shown in Figure 4. To be precise, we aim to solve the following minimization problem

$$\min_{s,\mathbf{t}} \sum_{i=1}^{49} \|\mathbf{p}_i - s\mathbf{q}_i - \mathbf{t}\|_2^2 = \min_{s,\mathbf{t}} \sum_{i=1}^{49} (\mathbf{p}_i - s\mathbf{q}_i - \mathbf{t})^T(\mathbf{p}_i - s\mathbf{q}_i - \mathbf{t}) = \min_{s,\mathbf{t}} f(s)$$

where $\mathbf{p}_\mathrm{i} \in \mathbb{R}^3$ is the ith column of $\mathbf{P}$, and $\mathbf{q}_\mathrm{i} \in \mathbb{R}^3$ is the ith column of $\mathbf{Q}$.

Steps for Exercise 4

$$f(s, \mathbf{t}) = \sum_{i=1}^{49} (\mathbf{p}_i - s\mathbf{q}_i - \mathbf{t})^T (\mathbf{p}_i - s\mathbf{q}_i - \mathbf{t})$$

1. Define $\mathbf{u} = [s, \mathbf{t}^T]^T$. Calculate $\partial f / \partial \mathbf{u}$.
   Set $\partial f / \partial \mathbf{u} = 0$ to compute the optimal solution $\hat{\mathbf{u}}$.
2. Open exercise_4a_point_set_registration.py
3. Compute $\hat{\mathbf{u}} = \left(\sum_{i=1}^{49} \mathbf{r}_i^T \mathbf{r}_i\right)^{-1} \sum_{i=1}^{49} \mathbf{r}_i^T \mathbf{p}_i$ where $\mathbf{r}_i = [\mathbf{q}_i, \mathbf{I}_3]$

   Note that $\mathbf{I}_3$ = np.eye(3)

   You might need np.column_stack() to create $[\mathbf{q}_i, \mathbf{I}_3]$

   You can use np.inv() to compute the inverse of the matrix $\sum_{i=1}^{49} \mathbf{r}_i^T \mathbf{r}_i$.

Steps for Exercise 4

4. Suppose $\hat{\mathbf{u}} = [s, \mathbf{t}^T]^T$, you can use `draw_landmarks`$(\mathbf{P}, \mathbf{R})$ to visualize the result, where $\mathbf{R} = s\mathbf{Q}$ + np.tile($\mathbf{t}$, 49)

5. Compute the mean distance error $d(s, \mathbf{t})$ for

   a) $s = 1, \mathbf{t} = [0,0,0]^T$

   b) $s = \hat{\mathbf{u}}[0], \; \mathbf{t} = \hat{\mathbf{u}}[1:]$

   The mean distance error $d(s, \mathbf{t})$ is defined as

$$d(s, \mathbf{t}) = \frac{1}{49}\left(\sum_{i=1}^{49} \sqrt{(\mathbf{p}_i - s\mathbf{q}_i - \mathbf{t})^T(\mathbf{p}_i - s\mathbf{q}_i - \mathbf{t})}\right)$$