

# Lab Instructions - session 6

## Least Squares

### Least Squares

Look at the following code.

**least\_squares.py**

```
import numpy as np

x_true = np.array([3, 1.5, -1.0, 2.4, -3, -.1, 2.2, 4.1, -3.2, 1.0])
n = x_true.size # no. of unknowns

m = 20 # no of equations (measurements)
A = np.random.randn(m,n)

# create the measurments
y_true = A @ x_true

# add noise to the measurments
sigma = 0.01
measurement_noise = sigma * np.random.randn(m)
y_noisy = y_true + measurement_noise

# we have access to the matrix "A" and noisy measurements "y_noisy".
# Frome these, we intend to estimate "x_true" using least squares
x_est = np.linalg.inv(A.T@A) @ A.T @ y_noisy
# x_est = np.linalg.solve(A.T@A, A.T @ y_noisy)
# x_est = np.linalg.lstsq(A,y_noisy)[0]

# measure the distance between the estimated unknowns "x_est"
# and the ture ones "x_true"
print('error=', np.linalg.norm(x_est - x_true))
```

- Explain the code.
- Use the alternative method `np.linalg.solve(A.T@A, A.T @ y_noisy)` and check if you get a similar `x_est`. Why this is equivalent to the least squares solution `np.linalg.inv(A.T@A) @ A.T @ y_noisy`?
- You can also use the numpy function `np.linalg.lstsq` to do least squares. Verify that it gives the same result.

## Task 1

Put the above in a loop to repeat it 100 times and report the average error. Afterward, keep increasing  $m$ , the number of equations (measurements). How does increasing the number of equations affect the average error? How do you explain this?

## Back to the Face Models

From the previous lab, remember trying to find the  $\mathbf{a}$ ,  $\mathbf{b}$ , and  $\mathbf{c}$  to reconstruct **TargetFace2** as a linear combination of **Face1**, **Face2**, and **Face3**. To do that, we first created an overdetermined system of 136 equations in 3 unknowns  $\mathbf{F} \mathbf{x} = \mathbf{t}$ , where  $\mathbf{F}$  and  $\mathbf{t}$  were obtained by

```
face1 = Face1.ravel()
face2 = Face2.ravel()
face3 = Face3.ravel()
t = TargetFace.ravel()
F = np.stack((face1, face2, face3), axis=1)
```

In the previous Lab session, we chose 3 out of 136 equations, randomly or otherwise, to find  $\mathbf{x} = [\mathbf{a}, \mathbf{b}, \mathbf{c}]^T$  as the solution to a system of 3 equations and 3 unknowns. You observed that this approach failed when the target face was noisy.

```
NoisyTargetFace = TargetFace + 3 * np.random.randn(*TargetFace2.shape)
```

Here, we intend to use all the 136 equations to solve for  $\mathbf{x} = [\mathbf{a}, \mathbf{b}, \mathbf{c}]^T$ .

## Task 2

Use the least squares solution to solve  $\mathbf{F} \mathbf{x} = \mathbf{t}$  for a noisy target  $\mathbf{t}$ . Use the formula you learned in class. Compare the solution against randomly selecting 3 points.

**task2.py**

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

from face_data import Face1, Face2, Face3, TargetFace2, edges

def plot_face(plt, X, edges, color='b'):
    "plots a face"

    plt.plot(X[:,0], X[:,1], 'o', color=color, markersize=3)
```

```
for i,j in edges:
    xi = X[i,0]
    yi = X[i,1]
    xj = X[j,0]
    yj = X[j,1]

    # draw a line between X[i] and X[j]
    plt.plot((xi,xj), (yi,yj), '-', color=color)
plt.axis('square')
plt.xlim(-100,100)
plt.ylim(-100,100)

TargetFace = TargetFace2
NoisyTargetFace = TargetFace + 3 * np.random.randn(*TargetFace.shape)

face1 = Face1.ravel()
face2 = Face2.ravel()
face3 = Face3.ravel()
t = NoisyTargetFace.ravel();

F = np.stack((face1, face2, face3), axis=1)

for i in range(5):
    inds = np.random.choice(range(136), 3, replace=False)

    a1,b1,c1 = # solve 3 random equations
    a2,b2,c2 = # least squares solution

    Face_rnd = a1 * Face1 + b1 * Face2 + c1 * Face3
    Face_lsq = a2 * Face1 + b2 * Face2 + c2 * Face3

    plot_face(plt, NoisyTargetFace, edges, color='k')
    plot_face(plt, Face_rnd, edges, color='g')
    plot_face(plt, Face_lsq, edges, color='b')

plt.show()
```

- What do you conclude by comparing **Face\_rnd** with **Face\_lsq**?
- Plot **Face\_lsq** against **TargetFace** instead of **NoisyTargetFace**. What do you observe?
- Which one do you think is closer to **TargetFace**? **Face\_lsq** or **NoisyTargetFace**? Notice that we constructed **Face\_lsq** from the noisy target **NoisyTargetFace**. Why do you think this happens?
- Confirm the above numerically, by computing the sum of squared differences between the elements of pairs of matrices.

- Use `numpy.linalg.lstsq` to solve the least squares problem.