UID: U6261174

ASS5

1.

The results show as below

```
x: [ 5.00000000e-01  5.00000000e-01  1.11022303e-22 -5.55111505e-23]
optimal objective value: 1.5
PyDev console: starting.
```

The source code

linear_programming.py

linear_programming.py 〉 No Selection

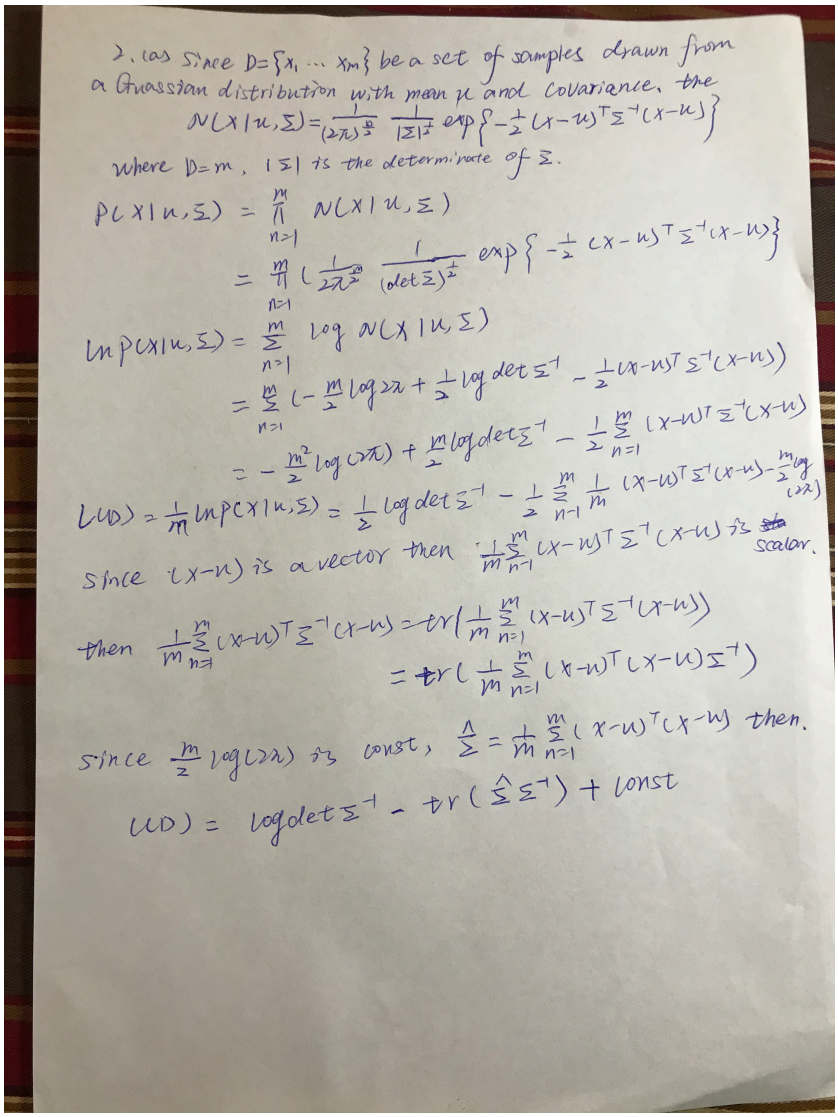
```python
import cvxpy as cvx
# import numpy as np

# Problem data.
n = 4
A = [1, 2, 3, 4]
m1 = [1, 1, 1, 1]
m2 = [1, -1, 1, -1]

# Construct the problem.
x = cvx.Variable(n)
objective = cvx.Minimize(A*x)
constraints = [0 <= x, m1*x == 1, m2*x == 0]
prob = cvx.Problem(objective, constraints)

# The optimal objective value is returned by `prob.solve()`.
result = prob.solve()
# The optimal value for x is stored in `x.value`.
print("x:", x.value)
print("optimal objective value:", result)
```

2(a)



(a) Since $D=\{x_1 \cdots x_m\}$ be a set of samples drawn from a Guassian distribution with mean μ and covariance, the

$$N(x|u,\Sigma)=\frac{1}{(2\pi)^{\frac{D}{2}}}\frac{1}{|\Sigma|^{\frac{1}{2}}}\exp\left\{-\frac{1}{2}(x-u)^T\Sigma^{-1}(x-u)\right\}$$

where $D=m$, $|\Sigma|$ is the determinate of Σ.

$$p(x|u,\Sigma)=\prod_{n=1}^{m} N(x|u,\Sigma)$$

$$=\prod_{n=1}^{m}\left(\frac{1}{2\pi^{\frac{m}{2}}}\frac{1}{(\det\Sigma)^{\frac{1}{2}}}\exp\left\{-\frac{1}{2}(x-u)^T\Sigma^{-1}(x-u)\right\}\right.$$

$$\ln p(x|u,\Sigma)=\sum_{n=1}^{m}\log N(x|u,\Sigma)$$

$$=\sum_{n=1}^{m}\left(-\frac{m}{2}\log 2\pi+\frac{1}{2}\log\det\Sigma^{-1}-\frac{1}{2}(x-u)^T\Sigma^{-1}(x-u)\right)$$

$$=-\frac{m^2}{2}\log(2\pi)+\frac{m}{2}\log\det\Sigma^{-1}-\frac{1}{2}\sum_{n=1}^{m}(x-u)^T\Sigma^{-1}(x-u)$$

$$L(D)=\frac{1}{m}\ln p(x|u,\Sigma)=\frac{1}{2}\log\det\Sigma^{-1}-\frac{1}{2}\sum_{n=1}^{m}\frac{1}{m}(x-u)^T\Sigma^{-1}(x-u)-\frac{m}{2}\log(2\pi)$$

since $(x-u)$ is a vector then $\frac{1}{m}\sum_{n=1}^{m}(x-u)^T\Sigma^{-1}(x-u)$ is scalar.

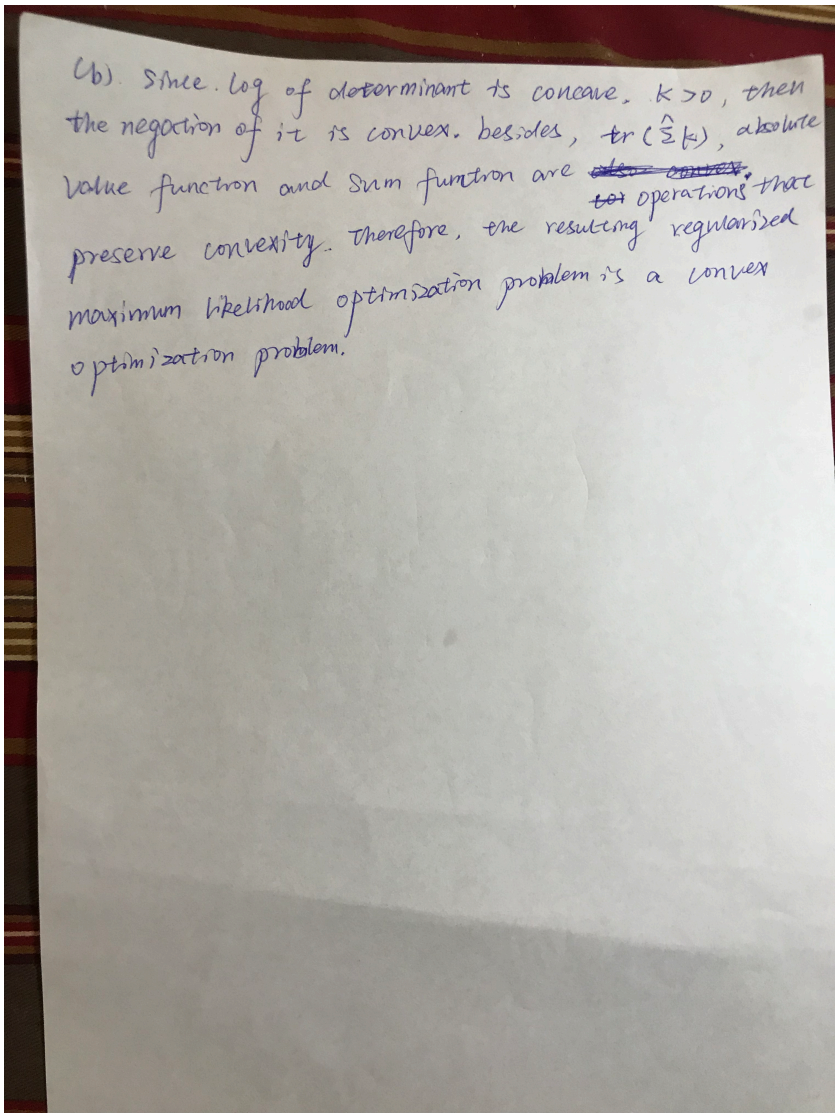
then $\frac{1}{m}\sum_{n=1}^{m}(x-u)^T\Sigma^{-1}(x-u)=tr\left(\frac{1}{m}\sum_{n=1}^{m}(x-u)^T\Sigma^{-1}(x-u)\right)$

$$=tr\left(\frac{1}{m}\sum_{n=1}^{m}(x-u)^T(x-u)\Sigma^{-1}\right)$$

since $\frac{m}{2}\log(2\pi)$ is const, $\hat{\Sigma}=\frac{1}{m}\sum_{n=1}^{m}(x-u)^T(x-u)$ then.

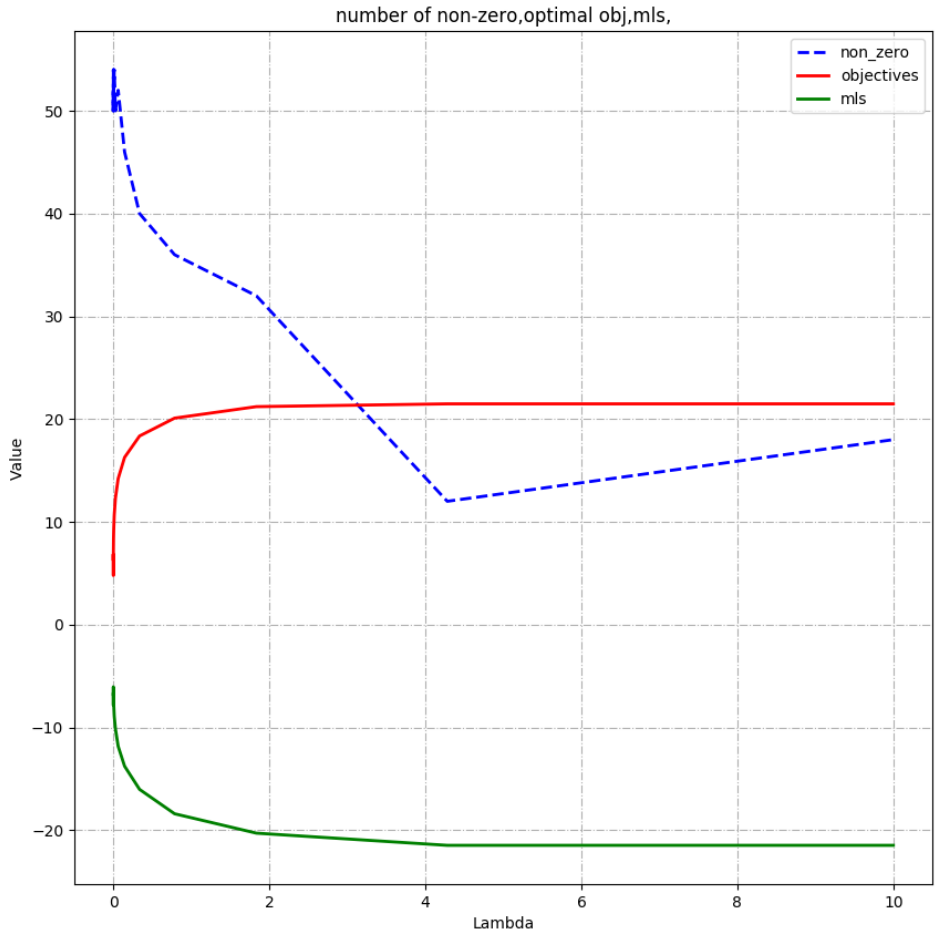
$$L(D)=\log\det\Sigma^{-1}-tr(\hat{\Sigma}\Sigma^{-1})+\text{const}$$

(b)



(b). Since. log of determinant is concave, $k>0$, then the negation of it is convex. besides, $tr(\hat{\Sigma}_k)$, absolute value function and sum function are ~~also convex~~, ~~to~~ operations that preserve convexity. Therefore, the resulting regularized maximum likelihood optimization problem is a convex optimization problem.

(c) Figure 1 shows the results of the 2(c). The blue line represents the number of non-zeros in the inverse covariance matrix as a function of λ. The red line represents the value of optimal objective function. The green line represents the value of log-likelihood function without the regularization. The range of λ in this assignment is 1e-6~10.



Figue1： values change by λ

the source code

RMLE.py

RMLE.py 〉 No Selection

```
import matplotlib.pyplot as plt
import cvxpy as cvx
import numpy as np
import pickle

# Problem data.
X = pickle.load(open('asgn5q2.pkl', 'rb'))
m, n = X.shape
sigmaHat = np.cov(X, rowvar=0)

# non-zero array
non_zeros = np.zeros(20)
# objective array
objectives = np.zeros(20)
# ML array
mls = np.zeros(20)

# off-diagonal matrix
off_dia = 1-np.eye(n, n)
v_dia = np.ones(n)
# values of lambda
lambdas = np.logspace(-6, 1, num = 20)
# print(lambdas)

# Construct the problem.
for i in range(len(lambdas)):
    K = cvx.Variable((n, n), PSD=True)
    # K = cvx.Variable((n, n), symmetric=True)
    objective = cvx.Minimize(-cvx.log_det(K) +
                             cvx.trace(sigmaHat*K) +
                             lambdas[i]*v_dia * cvx.multiply(off_dia, cvx.abs(K)) * v_dia.T)
    constraints = []
    # constraints = [K >> 0]
    prob = cvx.Problem(objective, constraints)
    result = prob.solve()

    non_zeros[i] = np.sum(K.value >= 1e-6)
    objectives[i] = result
    mls[i] = np.log(np.linalg.det(K.value)) - np.trace(np.dot(sigmaHat, K.value))


# show the graph
plt.figure(num=3, figsize=(10,10))
plt.title("number of non-zero,optimal obj,mls")
plt.grid(linestyle = "-.")
plt.xlabel("Lambda")
plt.ylabel("Value")
plt.plot(lambdas, non_zeros, color='blue', linewidth=2.0, linestyle='--', label='non_zero')
plt.plot(lambdas, objectives, color='red', linewidth=2.0, label='objectives')
plt.plot(lambdas, mls, color='green', linewidth=2.0, label='mls')
plt.legend(loc='upper right')
plt.show()
```

3

Figure 2 shows the trade-off curve, the X ordinate represents $\|\hat{x} - x_{\text{corr}}\|_2^2$, the Y ordinate represents $\|D\hat{x}\|_1$.

Figure 3 shows the comparison between the original signal and the denoised shape when λ = 0.24, we can see that the noise has been significantly reduced, the shape has been smoothed very well.

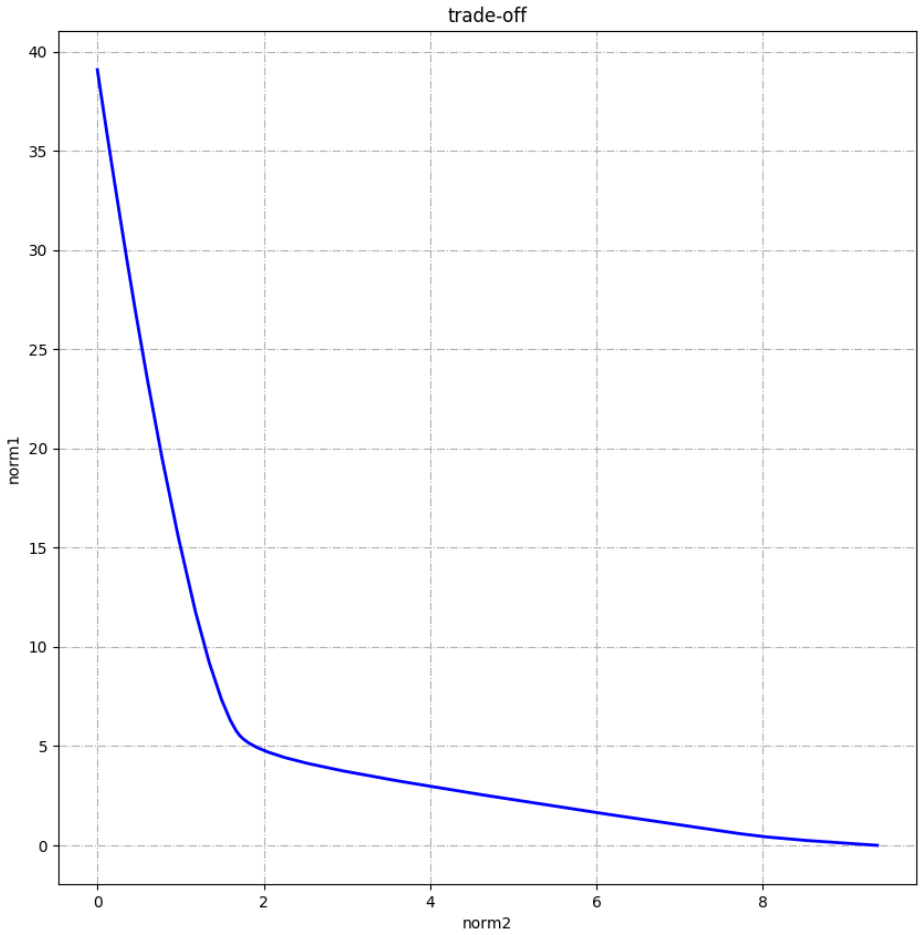


Figure 2: Trade-off

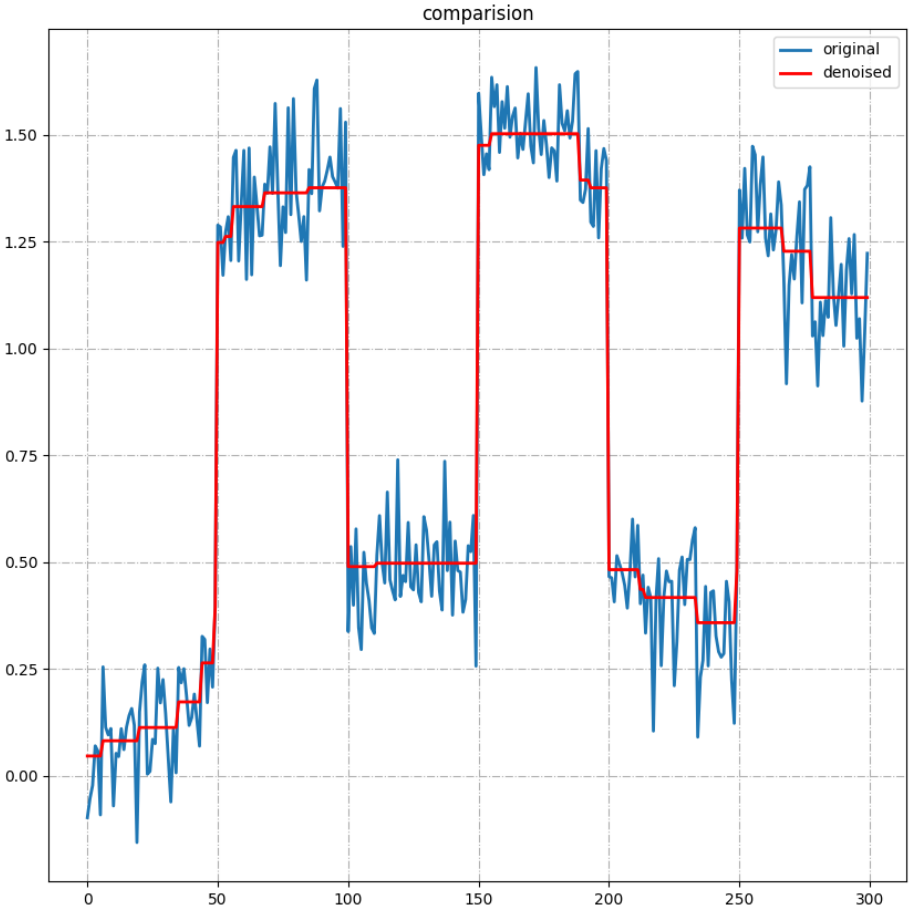


Figure 3: comparison between the original signal and the denoised shape

The source code shows as below:

denoising.py

denoising.py 〉 No Selection

```
import cvxpy as cvx
import numpy as np
import pickle
import matplotlib.pyplot as plt

# Problem data.
x_corr = pickle.load(open('asgn5q3.pkl', 'rb')).reshape(-1)
n = len(x_corr)

# non-zero array
error = np.zeros(50)
# objective array
objectives = np.zeros(50)
# ML array
regular = np.zeros(50)

# values of lambda
lambdas = np.logspace(-5, 2, num = 50)

# Construct the problem
for i in range(len(lambdas)):
    x = cvx.Variable(n)
    objective = cvx.Minimize(cvx.square(cvx.norm(x-x_corr)) +
                             lambdas[i]*cvx.norm(cvx.diff(x), 1))
    constraints = []
    prob = cvx.Problem(objective, constraints)
    result = prob.solve()

    error[i] = np.linalg.norm(x.value - x_corr)
    objectives[i] = result
    regular[i] = np.linalg.norm(np.diff(x.value), 1)

# show the graph
plt.figure(figsize=(10,10))
plt.title("trade-off")
plt.grid(linestyle = "-.")
plt.xlabel("norm2")
plt.ylabel("norm1")
plt.plot(error, regular, color='blue', linewidth=2.0)
plt.show()

# Construct the problem
x = cvx.Variable(n)
objective = cvx.Minimize(cvx.norm(x-x_corr) + 0.24*cvx.norm(cvx.diff(x), 1))
constraints = []
prob = cvx.Problem(objective, constraints)
result = prob.solve()

# show the graph
plt.figure(num=2,figsize=(10,10))
plt.title("comparision")
plt.grid(linestyle = "-.")
plt.plot(x_corr, linewidth=2,label= 'original')
plt.plot(x.value,color = 'red',linewidth=2,label='denoised')
plt.legend(loc='upper right')
plt.show()
```