[Q1]

In data fitting, the task is to find a model, from a family of potential models, that best fits some observed data and prior information. Here the variables are the parameters in the model, and the constraints can represent prior information or required limits on the parameters (such as nonnegativity). The objective function might be a measure of misfit or prediction error between the observed data and the values predicted by the model, or a statistical measure of the unlikeliness or implausibility of the parameter values. The optimization problem is to find the model parameter values that are consistent with the prior information and give the smallest misfit or prediction error with the observed data.

In device sizing in electronic design, which is the task of choosing the width and length of each device in an electronic circuit. Here the variables represent the widths and lengths of the devices. The constraints represent a variety of engineering requirements, such as limits on the device sizes imposed by the manufacturing process, timing requirements that ensure that the circuit can operate reliably at a specified speed, and a limit on the total area of the circuit. A common objective in a device sizing problem is the total power consumed by the circuit. The optimization problem is to find the device sizes that satisfy the design requirements (on manufacturability, timing, and area) and are most power efficient.

[Q4]

4. let's denote $x = [x_1, \cdots, x_n]$

$$D = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \quad \text{where } \lambda_i > 0$$

$$x^T D x = \sum_{i=1}^{n} \lambda_i x_i^2 \le \max(\{x_i\}_{i=1}^{n}) \sum_{i=1}^{n} x_i^2 \qquad (1)$$

Since $\|x\|_2 = 1$, we have $\sum_{i=1}^{n} x_i^2 = 1$, and

$$x^T D x = \sum_{i=1}^{n} \lambda_i x_i^2 \le \max(\{x_i\}_{i=1}^{n}) \sum_{i=1}^{n} x_i^2 = \lambda_j \sum_{i=1}^{n} x_i^2 = \lambda_j \quad (2)$$

where $\lambda_j = \max(\{x_i\}_{i=1}^{n})$ and $\lambda_j > \{x_i\}_{i=1, i\neq j}^{n}$

~~Hence, equation (2) is equal to~~,

$$(2) \iff \sum_{i=1, i\neq j}^{n} \lambda_i x_i^2 + \lambda_j x_j^2 \le \lambda_j$$

$$\iff \sum_{i=1, i\neq j}^{n} \lambda_i x_i^2 \le (1 - x_j^2) \lambda_j$$

$$\iff \sum_{i=1, i\neq j}^{n} \lambda_i x_i^2 \le \left(\sum_{i=1, i\neq j}^{n} x_i^2\right) \cdot \lambda_j$$

$$\iff \sum_{i=1, i\neq j}^{n} \lambda_i x_i^2 \le \sum_{i=1, i\neq j}^{n} \lambda_j x_i^2$$

$$\iff \sum_{i=1, i\neq j}^{n} (\lambda_i - \lambda_j) x_i^2 \le 0$$

Clearly, when maximize $\sum_{i=1, i\neq j}^{n} (\lambda_i - \lambda_j) x_i^2 \le 0$,

we can only get 0, namely when equality holds.

Since $\{\lambda_i\}_{i=1}^{n} < \lambda_j$, when equality holds, we must have,

$$\{x_i\}_{i=1, i\neq j}^{n} = 0 \text{ and } x_j = 1$$

namely, the maximizer $\hat{x}$ has a 1 in the entry corresponding to the largest diagonal element (λ_j) of D, and is 0 elsewhere.

[Q5]

let's denote x_1 as ton of pulp for towels, x_2 as ton of pulp for paper
then optimization problem can be

$$\underset{x_1,\, x_2}{\text{maximize}} \quad 10x_1 + 100x_2$$

$$\text{subject to} \quad \begin{aligned} & x_1 + x_2 \le 8000 \\ & x_1 \geq 200 \\ & x_2 \geq 0 \\ & 10x_1 + 35x_2 \le 100\,000 \end{aligned}$$

This is a linear optimization problem.

[Q6]

6. let's denote y_i is row vector of $y \in R^M$

a_i is row vector of A $(M \times N)$

we have $\| y_i - a_i x \|_1 \le t$ $\quad i = 1, \cdots, M$

thus, we have linear program,

$$
\begin{array}{ll}
\text{minimize} & t \\
\text{subject to} & a_i^T x - t \le y_i \,, \quad i = 1, \cdots, k \\
 & -a_i^T x - t \le -y_i \,, \quad i = 1, \cdots, k
\end{array}
$$

[Q7]

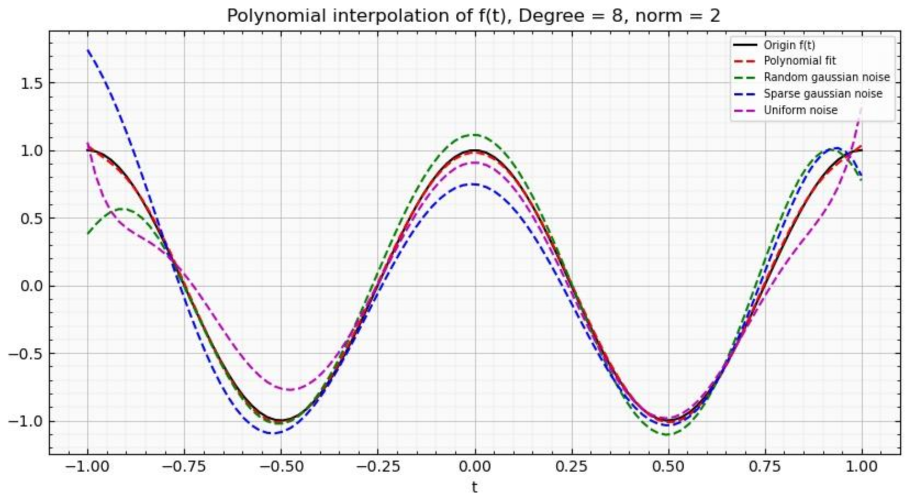
$$\min_{x \in R^N,\, u_m} \quad x^T I x + \sum_{m=1}^{M} u_m$$

$$\text{Subject to} \quad a_m^T x \leq u_m$$

$$z_m \leq u_m$$

[Q8]
(a)

In the picture below, I draw the polynomial interpolation of f(t) under different situations, Where the black solid line indicates the origin f(t) and the red dashed line indicates polynomial fit without any noise, green dashed line indicates polynomial fit with random gaussian noise, blue dashed line indicates polynomial fit with sparse gaussian noise and purple dashed line indicates polynomial fit with uniform noise.



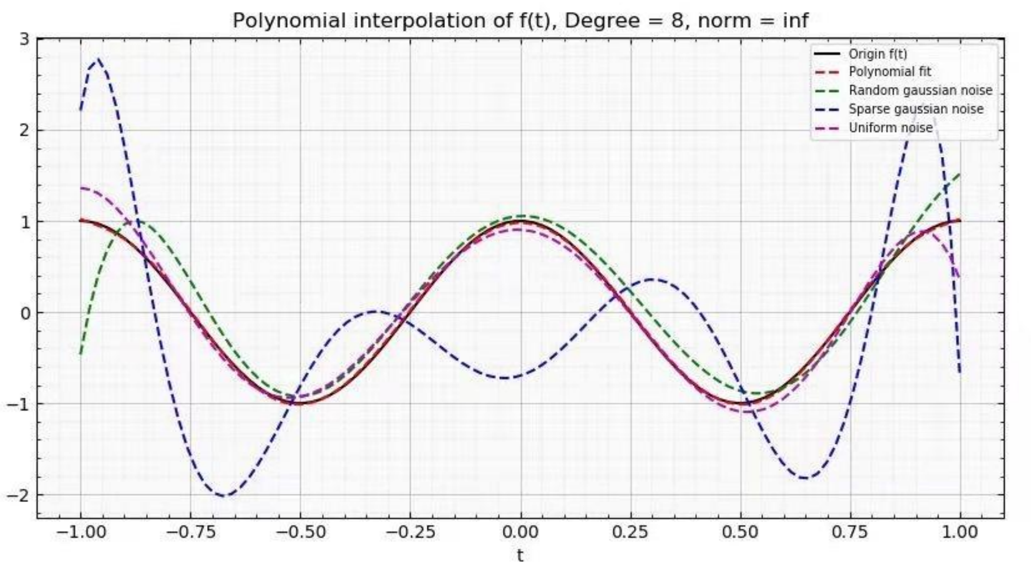
The corresponding norm residual is listed in the table below, clearly, random gaussian noise has least influences on fit.

Norm/Noise type	**Random gaussian noise**	**Sparse gaussian noise**	**Uniform noise**
2	1.48	1.88	1.59

And the optimal x value is listed below under different situations, note that all the results retain two decimal places

Noise type	**The optimal x value**								
Random gaussian noise	1.11	-0.15	-20.57	-0.10	61.47	1.85	-61.08	-1.40	19.65
Sparse gaussian noise	0.75	-0.22	-16.13	1.36	44.35	-0.96	-37.69	-0.64	10.00
Uniform noise	0.91	-0.03	-18.46	-1.46	62.02	3.30	-74.68	-1.67	31.4

(b)
For norm infinity, I draw the plot below. Graphically, uniform noise makes the smallest influence on fit, random gaussian noise seems fit better than sparse gaussian noise and worse than uniform noise.



The corresponding norm residual is listed in the table below, clearly, uniform noise has least influences on the fit.

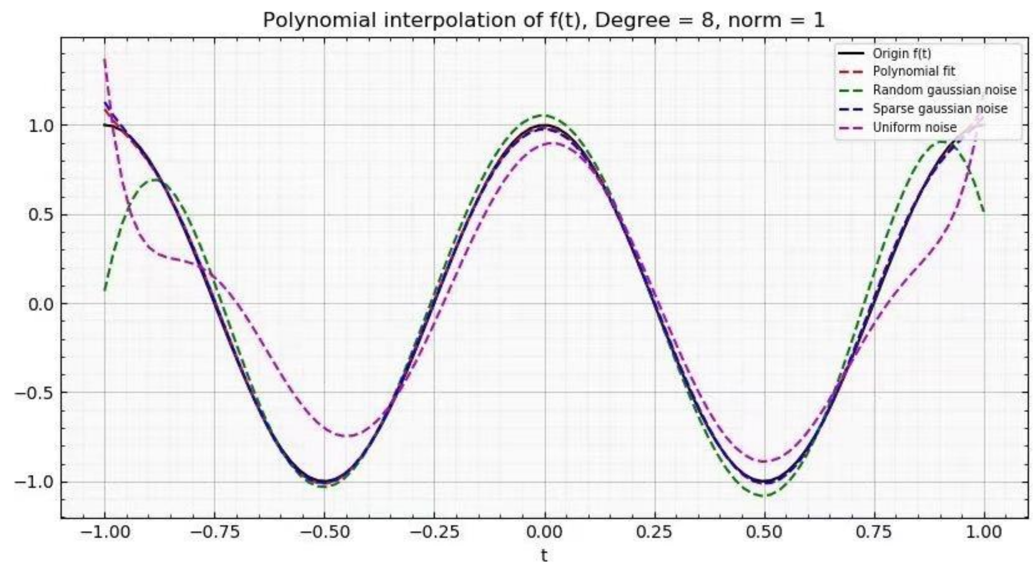
Norm/Noise type	**Random gaussian noise**	**Sparse gaussian noise**	**Uniform noise**
inf	1.47	1.80	0.64

Besides, the optimal x value is listed below under different situations, note that all the results retain two decimal places

Noise type	**The optimal x value**								
Random gaussian noise	1.05	0.23	-16.66	0.44	43.37	-6.05	-32.91	6.37	5.67
Sparse gaussian noise	-0.70	1.45	19.92	-11.56	-135.07	28.13	247.40	-19.46	-130.79
Uniform noise	0.90	-0.20	-16.3	-0.46	42.34	3.15	-32.59	-3.00	6.51

(c)

For norm 1, similarly, I draw the plot below. Graphically, sparse gaussian noise seems to make the smallest influences on the fit, while random gaussian noise and uniform noise don't fit so well.



The corresponding norm residual is listed in the table below, clearly, sparse gaussian noise has least influences on the fit.

Norm/Noise type	**Random gaussian noise**	**Sparse gaussian noise**	**Uniform noise**
1	9.38	1.28	17.67

Quantatively, the optimal x value is listed below under different situations, note that all the results retain two decimal places.

Noise type	**The optimal x value**								
Random gaussian noise	1.05	-0.12	-19.88	0.35	58.67	-0.29	-55.64	0.28	16.1
Sparse gaussian noise	0.98	-0.00	-18.81	-0.04	56.82	0.22	-58.54	-0.22	20.64
Uniform noise	0.89	0.67	-18.48	-5.85	65.68	11.08	-84.51	-5.99	37.69

(d)

Here, I calculate the corresponding norm residual for different cases

Norm/Noise type	**Random gaussian noise**	**Sparse gaussian noise**	**Uniform noise**
1	9.38	1.28	17.67
2	1.48	1.88	1.59
inf	1.47	1.80	0.64

The norm 1 is best suited for the Sparse Gaussian noise case, the inf norm is best suited for the uniform noise and norm inf is best suited for Random gaussian noise.

[Q9]

(a)

After setting the optimization problem and use CVXPY package to model it, there are 1000 predicted x value. In the program, I only print values which are greater than 1e-5(1×10^{-5}), and only ten results are left. Note that all results are preserved two decimal places.

All the results are shown in the picture,

```
/Users/gexueren/opt/anaconda3/python.app/Contents/MacOS/python /Users/gexueren/Desktop/6270/assignment/hw01/hw01_probo9.py
The optimal value is -2.554929424943638e-12
The optimal x is
[0.42, 0.47, 0.39, 0.97, 0.38, 0.11, 0.11, 0.76, 0.84, 0.39]
Predicted number of infected people
[[66], [83], [241], [347], [365], [500], [559], [620], [662], [968]]
True number of infected people
[66, 83, 241, 347, 365, 500, 559, 620, 662, 968]
The norm of the residual is  0.0
===========================================================================
```

From this picture, we can see clearly the value of optimal x and their predicted indexes. Meanwhile, I print the true indexes of infected people. Compare the two results, we can clearly see that all the predictions are 100 percent correct.

(b)

Here is how I get the optimal k:

Firstly, I find the optimal x whose value is greater than 1e-5. Secondly, if the number of optimal values is not equal setting number of infected individuals which is 10 in this case, then the optimization fails.

Not only I decrease K from 10 to 0 and find the lower bound, but also increase K from 10 to 100 and find the upper bound.

The final answer I got from the program is that **the lower bound of K is 3 and the upper bound of K is 97.**

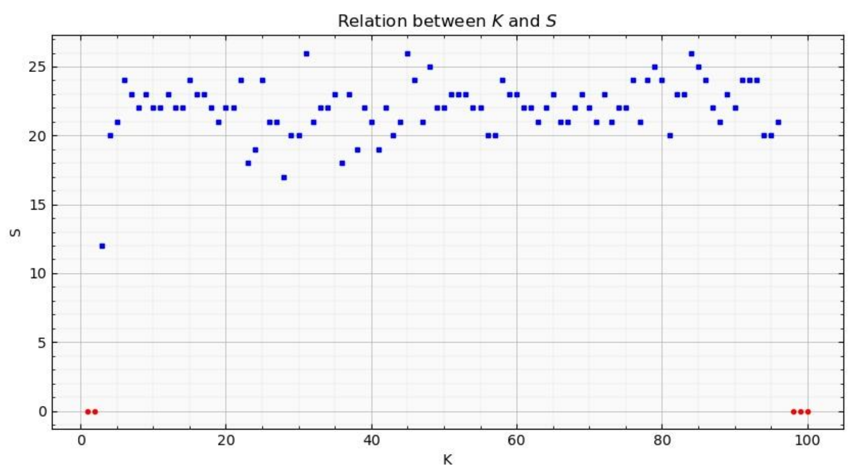
(c)

Here is how I do it:

I set 2 for loops in the program, the outer loop sets the value of K (number of batches), and the inner loop sets the value of S (number of infected individuals). Note that the range of S is [10, 1000] and the range of K is [1, 100].

And for most K, it's able to find a maximum s which the optimization solution exists. And for some certain cases, there is no solution.

In order to show it graphically, I set the initial value of S is equal to 0, and replace it with the optimal solution if exists. To be more clear, **the red dots in picture indicates that there's no optimal solution and the blue dots indicates the maximum S which the optimal solution exists.**



Clearly, we can see that the model is robust because when K ranges from 4 to 96, the optimal S always around 23. When K is lower than 4 or higher than 97, there is no optimal solution for S.

Appendix (Code for Problem 8 and 9)
Problem 8
```
'''
Author: Xueren Ge
Time: 2021/1/23
Descrption:
This script find the optimal solution for different norm
'''
import numpy as np
from matplotlib import pyplot as plt
import cvxpy as cp

if __name__ == '__main__':
    # Define function
    f_opt = lambda t: np.cos(2 * np.pi * t)

    # Number of samples
    M = 100

    # Degree + 1
    N = 9

    # Choose sample location
    tt = np.linspace(-1, 1, M)

    # Compute "clean" function
    ff = f_opt(tt)

    # Form "observation matrix"
    A = np.zeros((M, N))
    for i in np.arange(N):
        A[:, i] = tt ** i

    # Compute and plot least squares polynomial fit for "noise free" samples
    xhat = np.linalg.pinv(A) @ ff
    fhat = A @ xhat

    # plt.clf()
    # plt.plot(tt, ff, 'b')
    # plt.plot(tt, fhat, 'r--')
    # plt.xlabel('t')
    # plt.title('Polynomial interpolation of f(t), Degree = {}'.format(N -
1))
    # plt.legend(('Original f(t)', 'Polynomial fit'))
    # plt.show()

    ## Construct noisy observations
    np.random.seed(2020)  # Set random seed so results are repeatable

    # Random gaussian noise
    noise1 = np.random.randn(M)
    noise1 = 5 * noise1 / np.linalg.norm(noise1)
    y1 = ff + noise1
    xhat_1 = np.linalg.pinv(A) @ y1
    yhat_1 = A @ xhat_1
    print('Random gaussian noise: ', list(round(i, 2) for i in xhat_1))
```

```
    print('\n')

    # Sparse gaussian noise
    K = 15
    Gamma = np.random.choice(M, K, 0)
    noise2 = np.zeros(M)
    noise2[Gamma] = np.random.randn(K)
    noise2 = 5 * noise2 / np.linalg.norm(noise2)
    y2 = ff + noise2
    xhat_2 = np.linalg.pinv(A) @ y2
    yhat_2 = A @ xhat_2
    print('Sparse gaussian noise: ', list(round(i, 2) for i in xhat_2))
    print('\n')

    # Uniform noise
    noise3 = np.random.rand(M) - 0.5
    noise3 = 5 * noise3 / np.linalg.norm(noise3)
    y3 = ff + noise3
    xhat_3 = np.linalg.pinv(A) @ y3
    yhat_3 = A @ xhat_3
    print('Uniform noise: ', list(round(i, 2) for i in xhat_3))
    print('\n')


    fid = plt.figure(figsize=(10, 5))
    Axes = plt.subplot(1, 1, 1)
    Axes.axes.tick_params(which='both', direction='in', top=True, right=True)
    plt.minorticks_on()
    Axes.set_facecolor((0, 0, 0, 0.02))
    plt.plot(tt, ff, color='k', label='Origin f(t)')
    plt.plot(tt, fhat, 'k--', color='r', label='Polynomial fit')
    plt.plot(tt, yhat_1, 'k--', color='g', label='Random gaussian noise')
    plt.plot(tt, yhat_2, 'k--', color='b', label='Sparse gaussian noise')
    plt.plot(tt, yhat_3, 'k--', color='m', label='Uniform noise')
    plt.grid(True, which='major', linewidth=0.5)
    plt.grid(True, which='minor', linewidth=0.1)
    plt.xlabel('t')
    plt.title('Polynomial interpolation of f(t), Degree = {}, norm =
{}'.format(N - 1, 2))
    plt.legend(loc='upper right', fontsize='x-small')
    plt.savefig('/Users/gexueren/Desktop/6270/assignment/hw01/2.jpg')



    '''
    This part solves the norm infinity and norm 1 problem
    '''
    # Define and solve the CVXPY problem.
    y_dic = [ff, y1, y2, y3]
    for j in ["inf", 1]:
        yhat_norm_inf = []
        print("\n\nThis part is for norm: ", j)
        for i in range(4):
            x = cp.Variable(N)
            cost = cp.norm(A @ x - y_dic[i], j)
            prob = cp.Problem(cp.Minimize(cost))
            prob.solve()
```

```
            yhat_norm_inf.append(A @ x.value)
            # Print result.
            print("The optimal value is", round(prob.value, 2))
            print("The optimal x is")
            print(list(round(i, 2) for i in x.value))
            print("The norm of the residual is ", round(cp.norm(A @ x -
y_dic[i], p=j).value, 2))

print('=============================================================================
=====')

        fid = plt.figure(figsize=(10, 5))
        Axes = plt.subplot(1, 1, 1)
        Axes.axes.tick_params(which='both', direction='in', top=True,
right=True)
        plt.minorticks_on()
        Axes.set_facecolor((0, 0, 0, 0.02))
        plt.plot(tt, ff, color='k', label='Origin f(t)')
        plt.plot(tt, yhat_norm_inf[0], 'k--', color='r', label='Polynomial
fit')
        plt.plot(tt, yhat_norm_inf[1], 'k--', color='g', label='Random
gaussian noise')
        plt.plot(tt, yhat_norm_inf[2], 'k--', color='b', label='Sparse
gaussian noise')
        plt.plot(tt, yhat_norm_inf[3], 'k--', color='m', label='Uniform
noise')
        plt.grid(True, which='major', linewidth=0.5)
        plt.grid(True, which='minor', linewidth=0.1)
        plt.xlabel('t')
        plt.title('Polynomial interpolation of f(t), Degree = {}, norm =
{}'.format(N - 1, j))
        plt.legend(loc='upper right', fontsize='x-small')

plt.savefig('/Users/gexueren/Desktop/6270/assignment/hw01/{}.jpg'.format(j))
```

Problem 9

```
'''
Author: Xueren Ge
Time: 2021/1/24
Descrption:
This script models the spread of virus and how
to find the optimal Number of infected individuals
and optimal Number of splits of each sample
'''

import numpy as np
import cvxpy as cp
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings("ignore")

if __name__ == '__main__':
    ## PROBLEM 9
    # Construct the problem
    N = 1000 # Population size
    S = 10 # Number of infected individuals
```

```
    M = 100 # Number of tests
    K = 10 # Number of splits of each sample

    # Define x0
    ind0 = np.random.choice(N,S,0) # index subset
    x0 = np.zeros(N)
    x0[ind0] = np.random.rand(S)

    # Define A
    A = np.zeros((M,N))
    for i in np.arange(N):
        ind = np.random.choice(M,K,0)
        A[ind,i] = 1

    y = A @ x0

    x = cp.Variable(N)

    constraints = [x >= 0]

    cost = cp.norm(A @ x - y, 1)
    prob = cp.Problem(cp.Minimize(cost), constraints)
    prob.solve()
    print("The optimal value is", prob.value)
    print("The optimal x is")
    print(list(round(i, 2) for i in x.value[x.value > 1e-5]))
    print("Predicted number of infected people")
    print(np.argwhere(x.value > 1e-5).tolist())
    print('True number of infected people')
    print(sorted(ind0))
    print("The norm of the residual is ", round(cp.norm(A @ x - y,
p=1).value, 2))

print('=============================================================================
=====\n\n')




    '''
    This part solves (b) in Problem 9
    '''
    print('if we decrease K from 10')
    optimal_k = 10
    for k in range(10, -1, -1):

        ## PROBLEM 9
        # Construct the problem
        N = 1000  # Population size
        S = 10  # Number of infected individuals
        M = 100  # Number of tests

        # Define x0
        ind0 = np.random.choice(N, S, 0)  # index subset
        x0 = np.zeros(N)
        x0[ind0] = np.random.rand(S)
```

```
        # Define A
        A = np.zeros((M, N))
        for i in np.arange(N):
            ind = np.random.choice(M, k, 0)
            A[ind, i] = 1

        y = A @ x0
        x = cp.Variable(N)

        constraints = [x >= 0]

        cost = cp.norm(A @ x - y, 1)
        prob = cp.Problem(cp.Minimize(cost), constraints)
        prob.solve()

        if np.argwhere(x.value > 1e-5).shape[0] != len(ind0):
            break
        else:
            optimal_k = k
    print('the optimal k = ', optimal_k)
    print('\n')
    print('If we increase K from 10')
    optimal_k = 10
    for k in range(11, 101):
        ## PROBLEM 9
        # Construct the problem
        N = 1000  # Population size
        S = 10  # Number of infected individuals
        M = 100  # Number of tests

        # Define x0
        ind0 = np.random.choice(N, S, 0)  # index subset
        x0 = np.zeros(N)
        x0[ind0] = np.random.rand(S)

        # Define A
        A = np.zeros((M, N))
        for i in np.arange(N):
            ind = np.random.choice(M, k, 0)
            A[ind, i] = 1

        y = A @ x0
        x = cp.Variable(N)

        constraints = [x >= 0]

        cost = cp.norm(A @ x - y, 1)
        prob = cp.Problem(cp.Minimize(cost), constraints)
        prob.solve()

        if np.argwhere(x.value > 1e-5).shape[0] != len(ind0):
            break
        else:
            optimal_k = k
    print('the optimal k = ', optimal_k)

print('======================================================================
```

```
=====\n\n')


    '''
    This part solves (c) in Problem 9
    '''
    points = []
    for k in range(1, 101):
        temp_max_s = 0
        for s in range(10, 1001):
            ## PROBLEM 9
            # Construct the problem
            N = 1000  # Population size
            M = 100  # Number of tests

            # Define x0
            ind0 = np.random.choice(N, s, 0)  # index subset
            x0 = np.zeros(N)
            x0[ind0] = np.random.rand(s)
            # Define A
            A = np.zeros((M, N))
            for i in np.arange(N):
                ind = np.random.choice(M, k, 0)
                A[ind, i] = 1
            y = A @ x0
            x = cp.Variable(N)

            constraints = [x >= 0]

            cost = cp.norm(A @ x - y, 1)
            prob = cp.Problem(cp.Minimize(cost), constraints)
            prob.solve()
            # if there is a solution
            if np.argwhere(x.value > 1e-5).shape[0] == len(ind0) and
temp_max_s < s:
                temp_max_s = s
            else:
                print('current k: {}, current s: {}'.format(k, temp_max_s))
                points.append((k, temp_max_s))
                break

    fid = plt.figure(figsize=(10, 5))
    Axes = plt.subplot(1, 1, 1)
    Axes.axes.tick_params(which='both', direction='in', top=True, right=True)
    plt.minorticks_on()
    Axes.set_facecolor((0, 0, 0, 0.02))
    plt.plot([point[0] for point in points[2:-4]], [point[1] for point in
points[2:-4]], 'bs', markersize=3, label='Maximum S')
    plt.plot([point[0] for point in points[:2]], [point[1] for point in
points[:2]], 'ro', markersize=3, label='No Solution')
    plt.plot([point[0] for point in points[-3:]], [point[1] for point in
points[-3:]], 'ro', markersize=3)
    plt.grid(True, which='major', linewidth=0.5)
    plt.grid(True, which='minor', linewidth=0.1)
    plt.title("Relation between $K$ and $S$")
    plt.xlabel("K")
```

```
    plt.ylabel("S")
    plt.savefig('D:\gexueren\prob09.jpg')
```