西安交通大學
XI'AN JIAOTONG UNIVERSITY

Homework#1

Find the minimizer of a higher-dimensional quadratic objective function by Conjugate Gradient Method Programming

---

Name: Ensheng Shi (石恩升)

ID Number :4119105089

I Problem description

Find the minimizer of a higher-dimensional quadratic objective function by Conjugate Gradient Method (CGM). This can be formulated as :

$$\min_x f(x) = \frac{1}{2} x^T A x - b^T x$$

Where A & b should be generated by yourself and dimension n either can be fixed or is a input argument.

Note：

1. In order to use conjugate gradient method well, A,(n*n), must be an real symmetric positive definite(RSPD) matrix .
2. b and x are both n dimensional vector.

So we should solve two questions:

1. How to generate a RSPD matrix?
2. How to implement the iteration process of CGM? In detail , how to iterate the value of x(k), d(k), r(k), α(k)? (ps : this symbol can be found in p15 of lecture#3 ppt)

II Solution & Programming

In this chapter, we will solve the above two questions—generating RSPD matrix and implementing the CGM by programming.

2.1 Generating RSPD matrix

As we know, a matrix is RSPD when it’s eigenvalues are all greater than

zero.

A'*A will give a positive definite matrix if A is of full column rank. If A is of rank < n then A'A will be positive semidefinite (but not positive definite);

A is A non-singular matrix If A is A strictly diagonally dominant matrix.

So we can firstly generate strictly diagonally dominant matrix and then generate positive definite matrix. Noted M = A'*A must be symmetric because M' = = A'*A =M

In Python it would be as simple as

```
1. # Generate a random dim*dim matrix,all value in matix are less than 10
2. G = np.random.randint(0,10,size = [dim,dim])
3. # obtain strictly diaginally dominant matrix
4. G = G + 30*np.eye(dim)
5. # Multiply by its tranpose
6. A = G.T @ G
```

2.2 Implementing the CGM

Last class(26/9/2019), we get the processing of conjugate gradient method(CGM)

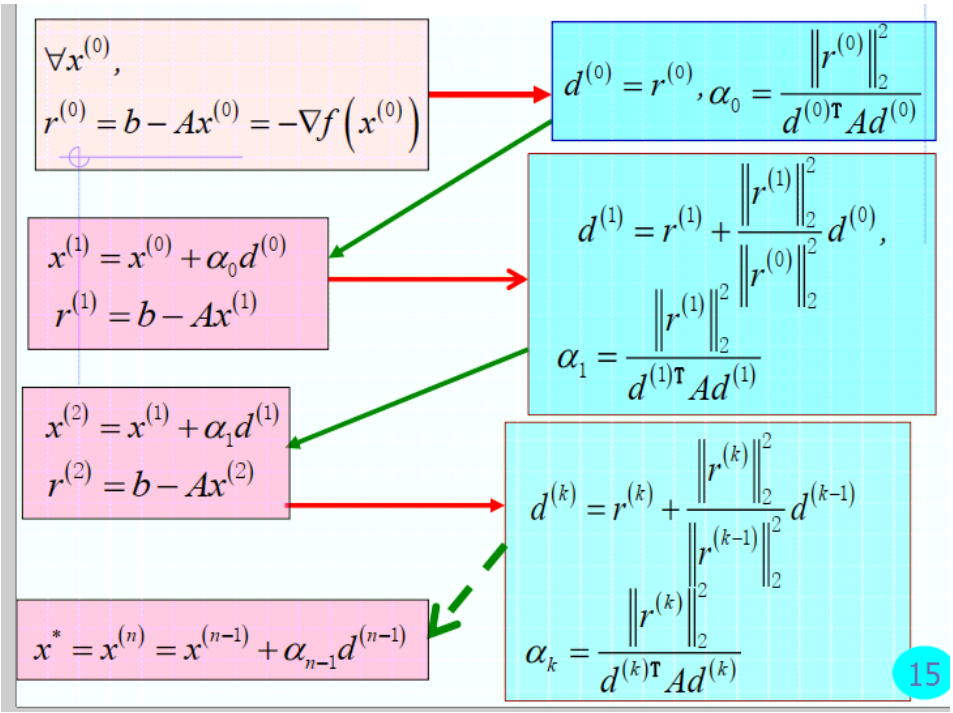


Fig 1, the processing of CGM (picture is made by Prof. Xiaoe Ruan)

Algorithm 2.1. (CG method).
Step 1: Initialize $x^{(0)}$, k = 0 Step 2: Calculate the residual error vector by $r^{(k)} = b - Ax^{(k)}$ Step3: Calculate the direction by $d^{(k)} = \begin{cases} r^{(0)} & k = 0 \\ r^{(k)} + \frac{\left\|r^{(k)}\right\|_2^2}{\left\|r^{(k-1)}\right\|_2^2} d^{(k-1)} & k > 1 \end{cases}$ Step 4: Calculate the step size by $\alpha_k = \frac{\left\|r^{(k)}\right\|_2^2}{d^{(k)\mathrm{T}} A d^{(k)}}$ Step 5: Set $x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}$. Step 6: Set k=k+1 and go to Step 2.

In Python it would be like this:

```
for k in range(0, dim):
    r[k] = b.T - A @ x[k]
    if k == 0:
        d[k] =  r[k]
    else:
        d[k] = r[k] + np.linalg.norm(r[k]) ** 2 / np.linalg.norm(r[k-1])**2 * d[k-1]
    alpha[k] =np.linalg.norm(r[k],2) **2  / (d[k].T @ A @ d[k])
    x[k+1] =  x[k] + alpha[k]*d[k]
```

But There is the true zero computer because of the way computer store number. So we should set threshold as terminator.

Code snippet shows below

```
for k in range(0, dim+1):
    r[k] = b.T - A @ x[k]
    if k == 0:
        d[k] =  r[k]
    else:
        d[k] = r[k] + np.linalg.norm(r[k]) ** 2 / np.linalg.norm(r[k-1])**2 * d[k-1]
    alpha[k] =np.linalg.norm(r[k],2) **2  / (d[k].T @ A @ d[k])
    x[k+1] =  x[k] + alpha[k]*d[k]
    func[k] = 1 / 2 * x[k].T @ A @ x[k] - b.T @ x[k]
    if np.linalg.norm(r[k] ) < threshold:
        length = k;
        print("dim = ",dim,"the number of interation is ",length)
        break
```

Later we will discuss effect of threshold on result.

2.3 Program:

Please look through appendix I for the complete code.

III Results

3.1 A brief example

When we run the program, it will remind you to type the dimension value. Take dimension = 5 for example:

The console will output A, the eigenvalues of A, B and $A^{-1}b$ and the best solution we obtain.

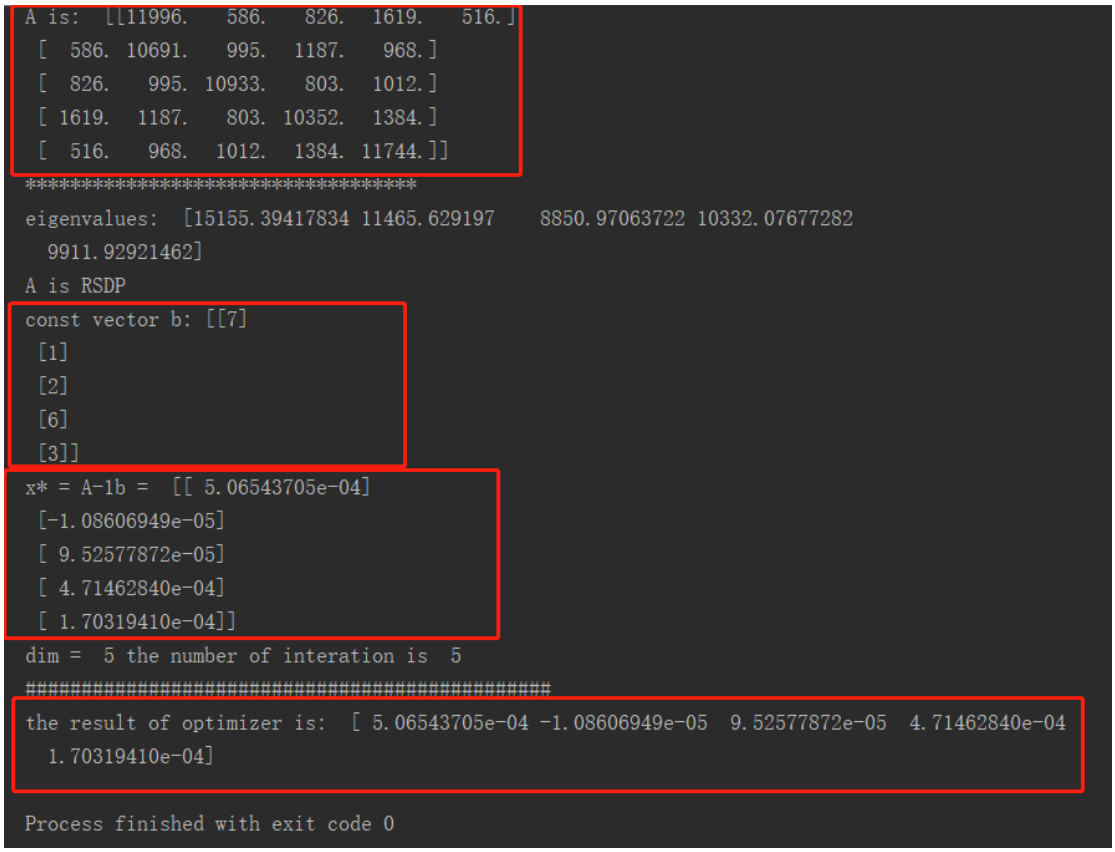


Fig 3.1 the output of console, RSPD matix A, the eigenvalues of A, const vector b and theoretical optimal solution A^(-1) b and the result of optimizer.

We can notice that there are similar values between $A^{-1}b$ and the result gained CGM. To illustrate the convergence of this method , the values of error and objective function will be visualized in fig3.2, the

values of error and objective function keep decreasing until it's zero. At n iterations gets the minimizer. Please find more examples in appendix.

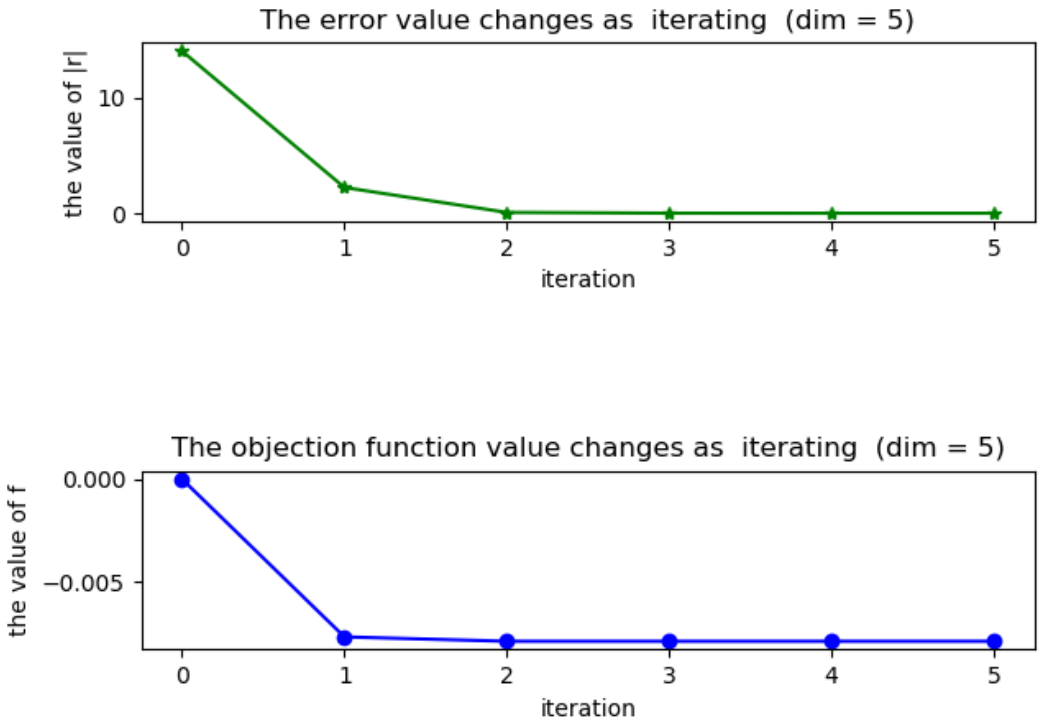
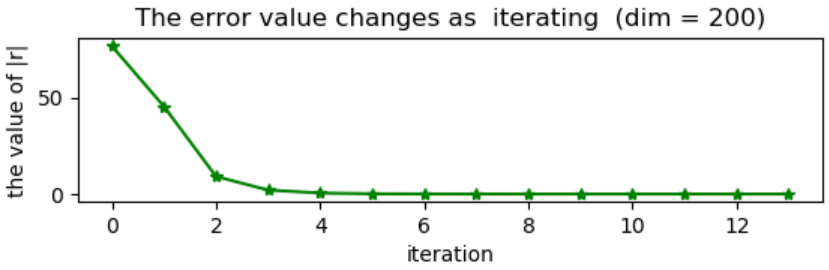
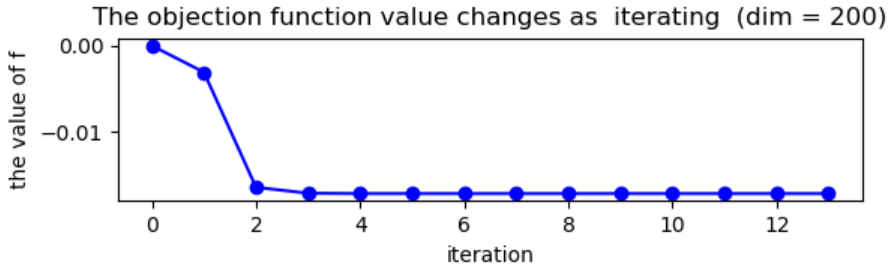


Fig 3.2 the values of error and objective, the values of error keep decreasing until it's zero. At n iterations gets the minimizer.

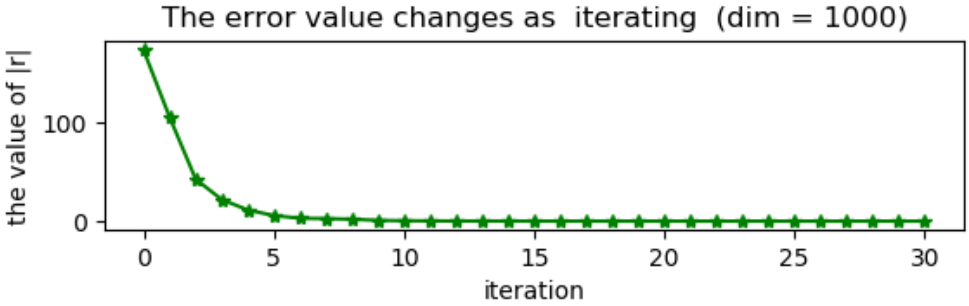
3.2 Scaling of dimensions

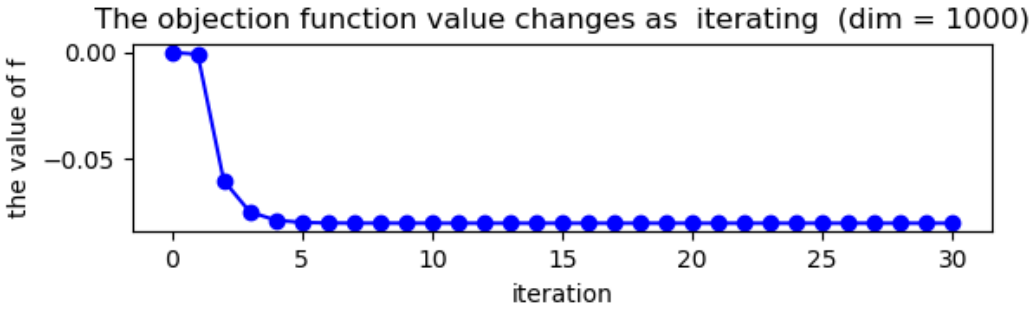
As we know, GCM can solve the high-dimension quadratic optimization problem. How the results are when dimension is different and becomes higher? To answer this question, we set dim = 10,50,100,200,500,1000. Fig 3.3 shows the a portion more in appendix.





(a) Dim =200





(b) dim =1000

Fig 3.3 the values of error and objective, the values of error change in different dimensions. Both of them keep decrease and eventually converge in various dimensions by GCM

3.3 Change α

When we Change **α** to a constant vector, the values of error is unusually large. Because the step-size is not non-trivial, it effect the independence among residual errors r^k and the conjugate gradient among the directions d^k (k=0,1,2...n).

1. When we set **α = 1,** $r^{(k)} = b - Ax^{(k)}, d^{(k)} = r^{(k)} + d^{(k-1)}$(k>1), then $< d^{(k)}, r^{(k-1)} >= 0$ will not true.

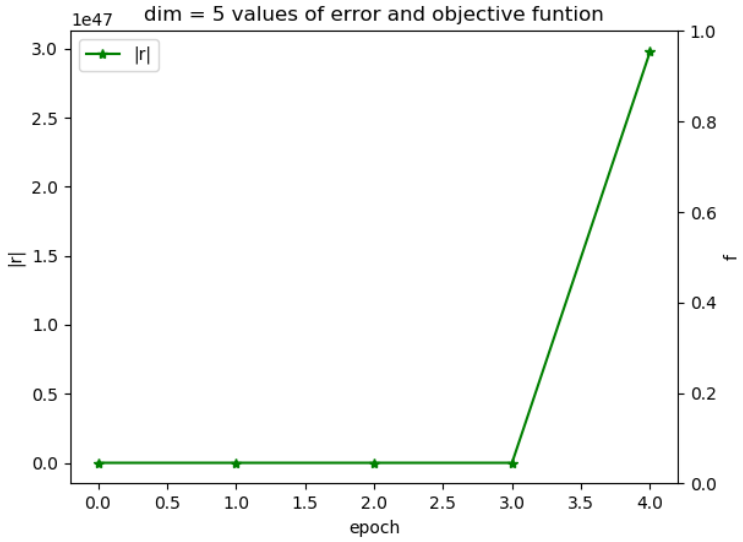


Fig 3.4 dim = 3, α = 1， the values of error keep increasing and result is so terrible

2. Let's make A smaller and set **α =** $\mathbf{10^{-5}}$, error value is closed to zero when it is over 300 epochs. The full landscape shows in Fig3.5.

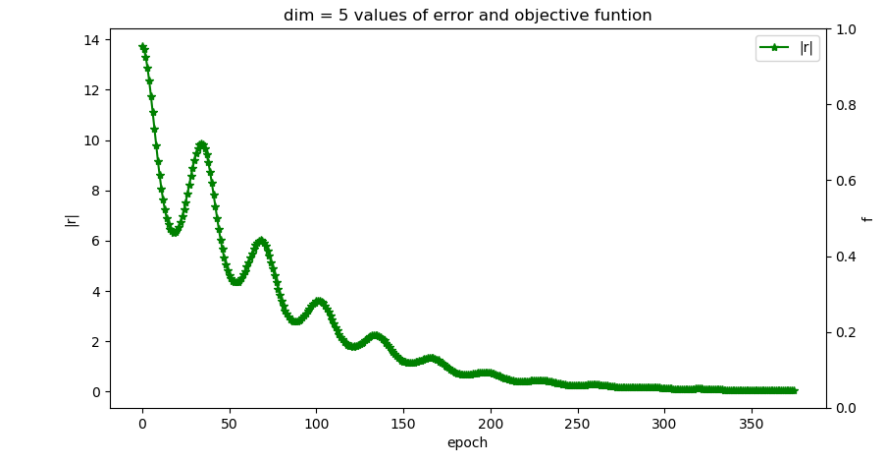


Fig 3.5 dim = 5, $\alpha = 10^{-5}$, error value oscillates and eventually converges to zeros

From Fig 3.4 , we conclude that error value oscillates and eventually converges to zeros. Although we evaluate the error in different A, the result still show that we should set α carefully. If setting it too lager, the error value will not converge and if setting it smaller, It will take more time to be converged. In brief, the value of alpha is changed, the $d^0, d^1 \ldots d^n$, are not conjugate directions with respect to A.

3.4 Ill-conditioned matrix A

When A is a RSAP but ill-conditioned matrix, error values will oscillate and the value of objective function doesn't keep decreasing. Fig 3.6 shows the result.

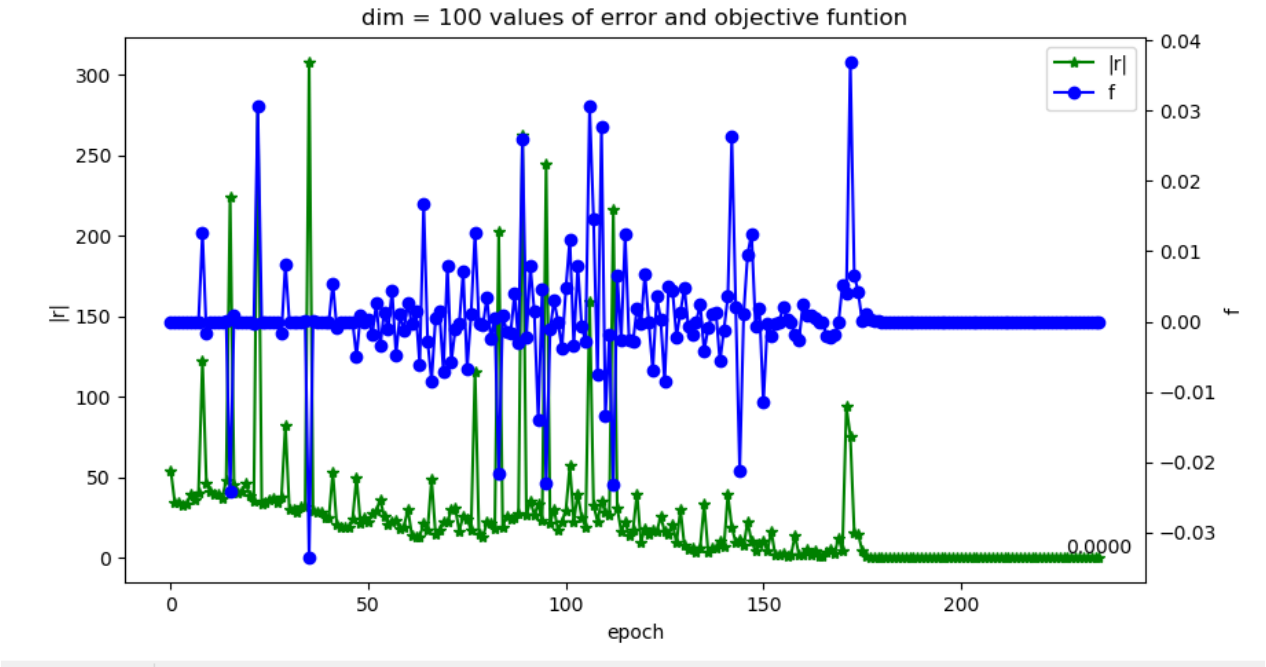


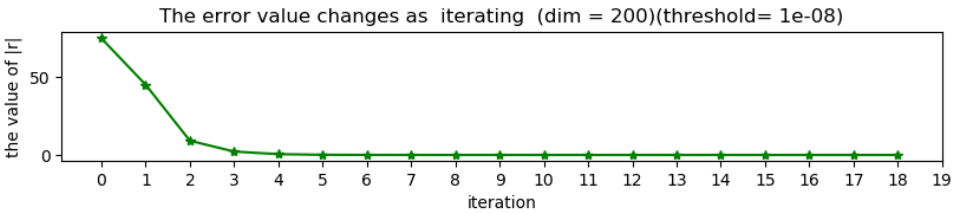
Fig 3.6 dim = 100 , error value oscillates and it is not converged

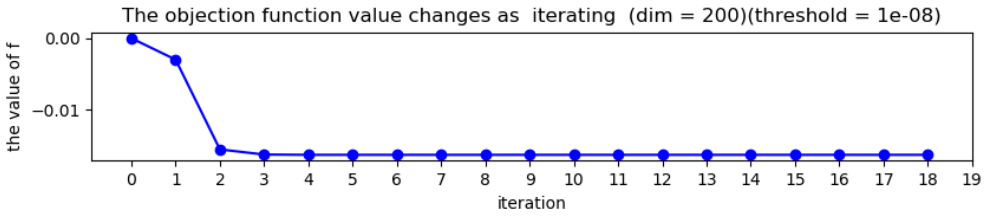
Because our computer do not do the exact calculation, a little disturbance will make the error oscillate. So we should avoid to construct the ill-conditioned matrix.

3.3 Change threshold

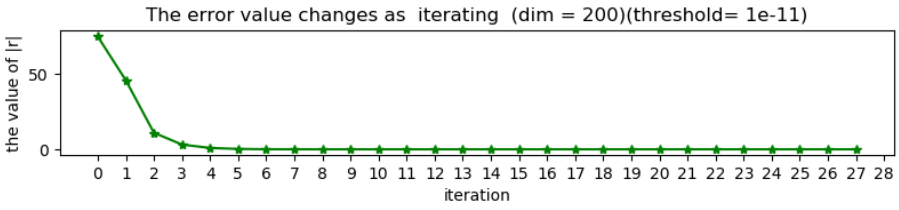
Because computer aren't able to store the precise fractional value, it is not the true zero. Thus, we set a threshold, if absolute value of variable is less than it , it is can be regarded as zeros. Here we can know that the size threshold effect the times of iteration and hence the finally result x* and f*.

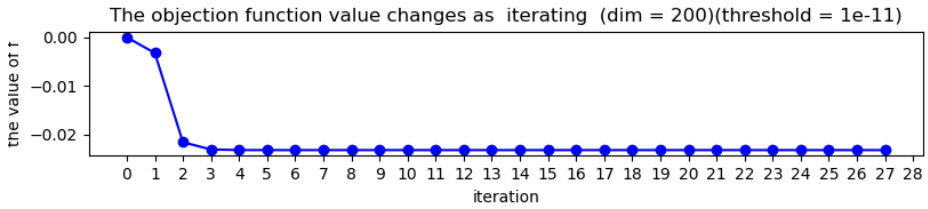
We contrast the different threshold and the result is showed in Fig3.7





(c) Threshold = 10^-8





(d) Threshold = 10^-11

Fig 3.7 The results in different threshold. We observe that time of iteration increases as the size threshold becomes lager.

From Fig **3.7** we conclude that as the size threshold increases, time of iteration increases. It is obvious because it threshold determines the precision. The program terminate early in the low precision.

IV Conclusion and acquirement

For the quadratic optimization:

$$\min_x f(x) = \frac{1}{2} x^T Ax - b^T x .$$

Strat from arbitrary x_0 and search along conjugate direction $d^0, d^1 \ldots d^n$ w.s.t RSPD A. At most n epoch gets the minimizer. What's more , when we change the α value, we can't promise that neither we get the good result within n iterations or it is converged.

From this homework, we main solve two subproblem:

1. To generate RSPD matrix?
2. To implement the iteration process of CGM. In detail , to iterate the value of x(k), d(k), r(k), α(k).

and we setting different parameter values to evaluate the goodness of CGM by changing the step-size. Later maybe we could change the directions or apply CGM to other optimization to evaluate it's performance.

Appendix A

1. Main()

```
#!/usr/bin/env python
#!-*-coding:utf-8 -*-
'''
@version: python3.7
@author: 'enshi'
@license: Apache Licence
@contact: *******@qq.com
@site:
@software: PyCharm
@file: Main.py
@time: 10/8/2019 9:02 AM
'''
import numpy as np
```

```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import host_subplot
def CGM(dim ,threshold = 10**-5):

    #1. generate maxtix A and a vector b
    # A = GT*G
    # f(x) = 1/2 x(T)Ax - b(T)x

    # Generate a random dim*dim matrix
    G = np.random.randint(0,10,size = [dim,dim])
    # obtain strictly diaginally dominant matrix
    G = G + 240*np.eye(dim)
    # Multiply by its tranpose
    A = G.T @ G
    e,v = np.linalg.eig(A)

    b = np.random.randint(0,10,size = [dim,1])
    print("A is: ",A  )
    print("**********************************")
    print("eigenvalues: " ,e)
    if e.all() > 0:
        print("A is RSDP")
    print("const vector b:" ,b)
    print("x* = A-1b = ",np.linalg.inv(A)@b)
    #2. init  & iterate
    # r(k) = b - Ax(k)
    # d(k) = r(k) +  |r(k)|/|r(k-1)|*d(k-1)
    # α(k) = |r(k)|/(d(k)(T)Ad(k))
    x = np.zeros([dim+2,dim])
    r = np.zeros([dim+1,dim])
    d = np.zeros([dim+1,dim])
    alpha =  np.zeros([dim+1,1])
    func =  np.zeros([dim+1,1])

    length = dim;
    for k in range(0, dim+1):
        r[k] = b.T - A @ x[k]
```

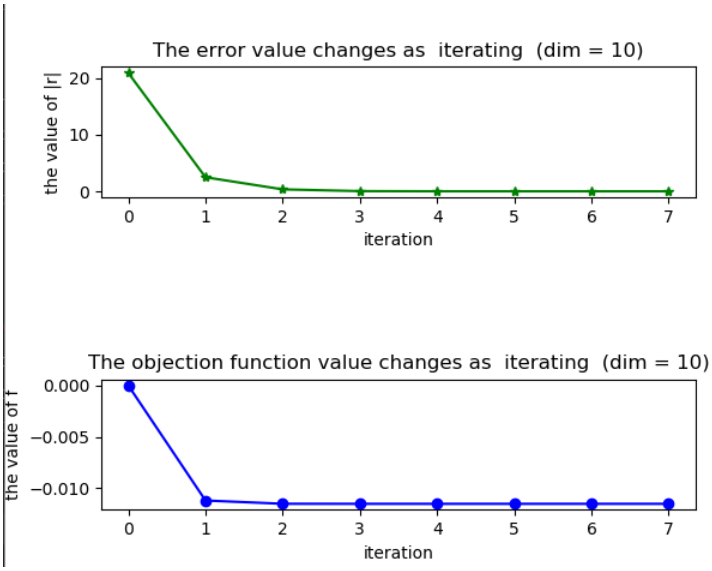
```
        if k == 0:
            d[k] =  r[k]
        else:
            d[k] = r[k] + np.linalg.norm(r[k]) ** 2 / np.linalg.norm(r[k-1])**2 * d[k-1]
        alpha[k] =np.linalg.norm(r[k],2) **2  / (d[k].T @ A @ d[k])
        x[k+1] =  x[k] + alpha[k]*d[k]
        func[k] = 1 / 2 * x[k].T @ A @ x[k] - b.T @ x[k]
        if np.linalg.norm(r[k] ) < threshold:
            length = k;
            print("dim = ",dim,"the number of interation is ",length)
            break
    return r[0:length+1,],func[0:length+1],x[0:length+2,]


def PlotTwin(r,func):
    r2norm = np.linalg.norm(r,2,1)
    print("################################################")
    #print("2 norm of r is : ",r2norm)
    #print("the values of object function",func)

    X = np.arange(0,len(r2norm))
    ax1 =  host_subplot(111)
    ax2 = ax1.twinx()
    ax1.plot(X, r2norm, 'g*-',label = "|r|")
    ax2.plot(X, func, 'bo-',label = "f")

    # set digital label

    for a, b in zip(X[-1:], r2norm[-1:]):
        plt.text(a, b, '%.4f' % b, ha='center', va='bottom', fontsize=10)

    ax1.set_xlabel('epoch')
    ax1.set_ylabel('|r|')
    ax2.set_ylabel('f')
    #plt.xticks(range(length+1))
    ax1.legend() # 显示图例
    plt.title("dim = "+str(dim)+" values of error and objective funtion")
```

```
    plt.show()


def PlotSub(r,func,dim):
    plt.figure()
    r2norm = np.linalg.norm(r, 2, 1)
    X = np.arange(0, len(r2norm))
    length = len(X)
    # fig1
    ax1 = plt.subplot(3, 1, 1)
     plt.plot(X, r2norm, 'g*-')
     ax1.set_title("The error value changes as  iterating  " + "(dim = " +st
r(dim) +")" )
     ax1.set_xlabel('iteration')
     ax1.set_ylabel('the value of |r|')
     #plt.xticks(range(length + 1))

     # fig2
     ax2 = plt.subplot(3, 1, 3)
     plt.plot(X, func, 'bo-')
     ax2.set_title("The objection function value changes as  iterating  " +
"(dim = " +str(dim)+")")
     ax2.set_xlabel('iteration')
     ax2.set_ylabel('the value of f')
     #plt.xticks(range(0,length + 1),10)

     plt.show()


if __name__ ==  '__main__':
     dim = 200
     r,func,x = CGM(dim )
     #PlotTwin(r,func)
     PlotSub(r,func,dim)
     # 3.result
     #print("x: ",x)
     print("################################################")
     print("the result of optimizer is: ",x[-1])
```

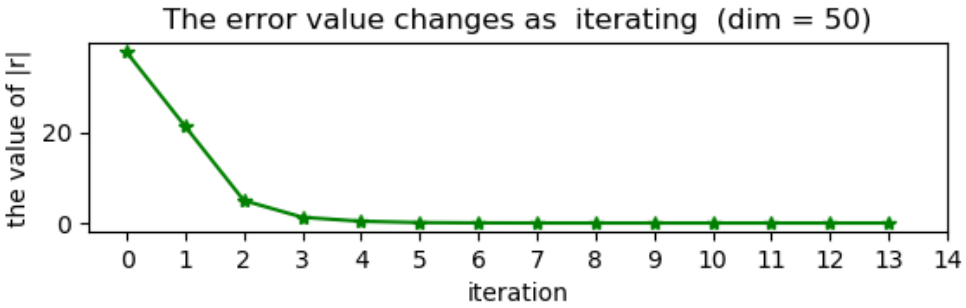
```
126. plt.show()
```

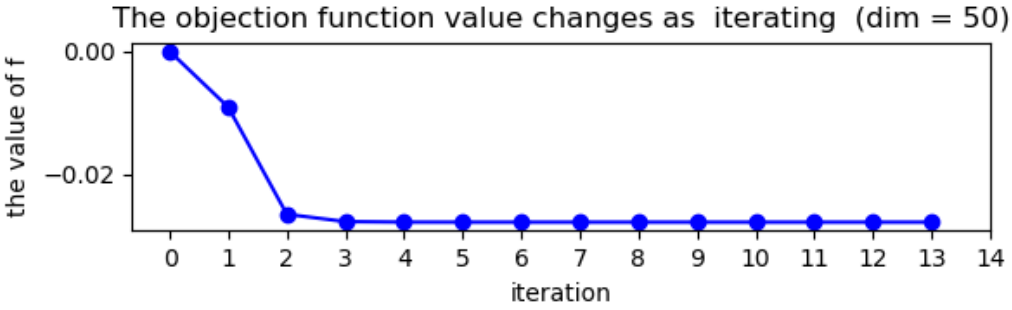
2.Dim 10 , 50，100，500，1000

Dim =10

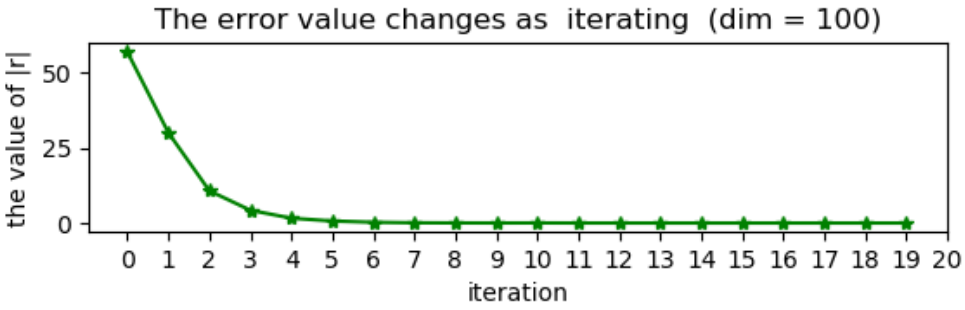


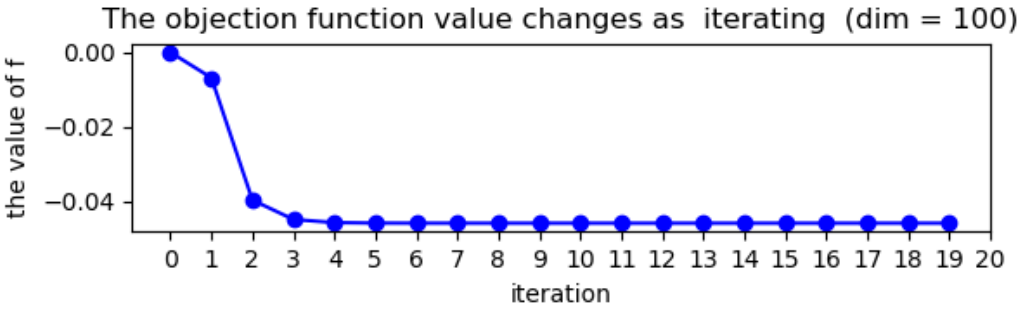
dim = 50



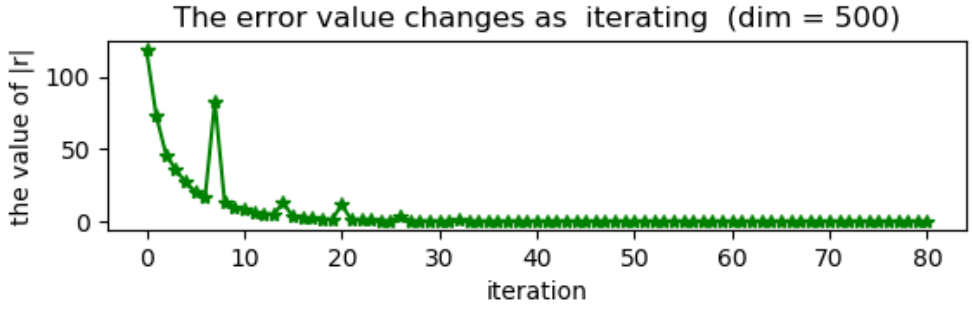


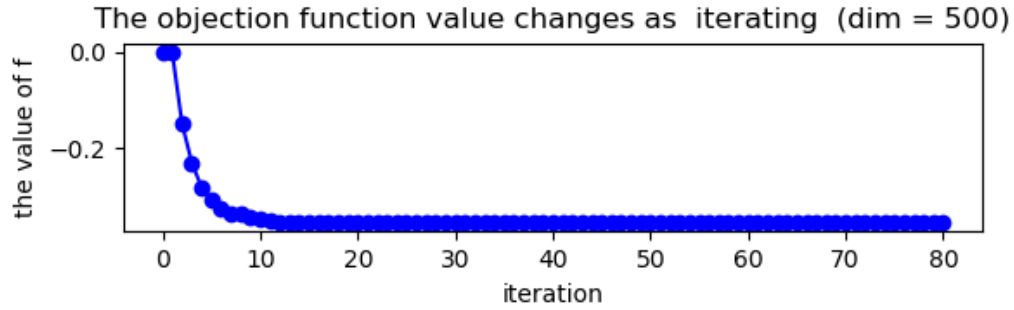
dim = 100


The error value changes as iterating (dim = 100)
the value of |r|
50
25
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
iteration


The objection function value changes as iterating (dim = 100)
the value of f
0.00
−0.02
−0.04
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
iteration

4 dim = 500


The error value changes as iterating (dim = 500)
the value of |r|
100
50
0
0 10 20 30 40 50 60 70 80
iteration


The objection function value changes as iterating (dim = 500)
the value of f
0.0
−0.2
0 10 20 30 40 50 60 70 80
iteration