西安交通大學
XI'AN JIAOTONG UNIVERSITY

Homework#2

Name: Ensheng Shi (石恩升)

ID Number : 4119105089

I Problem description

Find the minimizer of the higher-dimensional quadratic optimization for the respective consistent convex, bounded convex and convex cases with different initial points by any 3 methods of conjugate gradient method, steepest descent method, Newton's method and Quasi Newton method.

So we should solve three questions:

1. How to generate different matrix corresponding to above three cases?
2. How to implement the optimization methods above?
3. How to set up experiences to compare the convergence and rate of convergence

II Solution & Programming

In this chapter, we will solve the above three questions—considering different cases, implement Steepest descent method, Newton's method and Quasi Newton method and discussing the convergence of these methods later.

2.1 Considering different cases

Suppose the set D is nonempty open convex and f(x): D -> R is 2^{nd} –order

continuously differentiable. If $m|u|^2 \leq u^T \nabla^2 f(x) u$, function f(x) is convex. Thus, $\nabla^2 f(x) - mI \ \ is \ \ RSDP$. We can construct A = A + m*I, where A is RSDP. If Hessian Matrix is RSSDP (real symmetric semi-define positive), f(x) is convex. If Hessian Matrix is neither RSSDP nor RSNP (real symmetric negative positive), f(x) may be bounded convex. Thus, we apply these to generate different matrixes to discuss.

They are simple in python. Take RSSDP for example, you can find other matrix in myutils.py

RSSDP

```
1. def funcConvex(dim):
2.     eigvals = 10 * np.abs(np.random.random((dim)))
3.     A = np.eye(dim)
4.     for i in range(dim):
5.         A[i][i] = eigvals[i]
6.     seed = 1
7.     U = np.float32(ortho_group.rvs(dim=dim, random_state=
   seed))
8.     b = np.random.rand(dim)
9.     A = U.transpose().dot(A).dot(U)
10.     e, v = np.linalg.eig(A)
11.     print(A)
12.     print("################################")
13.     if min(e) != 0:
14.         condA = max(e) / min(e)
15.         print("Conditional number : ", condA)
16.     return A,b
```

The conditional number outputs to tell us whether this matrix is illness or not.

2.2 Implement three methods

In this section, three method, steepest descent method, newton's method

and quasi newton method, is implemented.

To be simple, only the algorithm and python code snippets are showed. First, it is algorithm of steepest descent method and corresponding code snippets, then is newton's method and finally is quasi newton method

Algorithm 2.1. (Steepest descent method).
Step 1: Initialize $x^{(0)}$, k = 0 Step 2: Calculate the residual error vector by $\text{grad}^{(k)} = Ax^{(k)} + b$ Step3: Calculate the direction by $d^{(k)} = -\text{grad}^{(k)}$ Step 4: Calculate the step size by $\alpha_k = \frac{\|d^{(k)}\|_2^2}{d^{(k)T} A d^{(k)}}$ 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:

```
def SteeDesMethod(x, A, b , threshold,dim):
    grad_vec = np.dot(A,x) + b
    grad_length = np.linalg.norm(grad_vec)
    k = 0
    f = 1 / 2 * np.dot(np.dot(x,A),x) + np.dot(b,x)
    data_x = [x]
    data_f = [f]
    data_g = [grad_length]
    alpha = 1
     while  alpha*grad_length > threshold:
         k += 1
         grad = -grad_vec
         # iterate
         alpha = np.dot(grad,grad) / np.dot(np.dot(grad,A),gra
d)
         #print(alpha)
```

```
        x = x + alpha* grad
        #gradient
        grad_vec = np.dot(A, x) + b
        grad_length = np.linalg.norm(grad_vec)
        f =1 / 2 * np.dot(np.dot(x,A),x) + np.dot(b,x)
        #update
        data_x.append(x)
        data_f.append(f)
        data_g.append(grad_length)
```

Algorithm 2.2. (Newton's method).
Step 1: Initialize $x^{(0)}$, k = 0 Step 2: Calculate the residual error vector by $grad^{(k)} = Ax^{(k)} + b$ Step3: Calculate the direction by $d^{(k)} = -grad^{(k)}$ Step 4: Calculate the step size by $H^{(k)} = A^{-1}$ Step 5: Set $x^{(k+1)} = x^{(k)} + H^{(k)}d^{(k)}$. Step 6: Set k=k+1 and go to Step 2.

In Python it would be like this:

```
def NewtonMethod(x,A,b, threshold,dim):
    grad_vec = np.dot(A,x) + b
    grad_length = np.linalg.norm(grad_vec)
    k = 0
    f = 1 / 2 * np.dot(np.dot(x,A),x) + np.dot(b,x)
    data_x = [x]
    data_f = [f]
    data_g = [grad_length]
    delta_norm = 1
     while delta_norm  > threshold:
        k += 1
        grad = grad_vec
        # iterate
        alpha = np.linalg.inv(A)
        #print(alpha)
```

```
        delta = - np.dot(alpha, grad)
        x = x + delta
        #gradient
        grad_vec = np.dot(A, x) + b
        grad_length = np.linalg.norm(grad_vec)
        delta_norm = np.linalg.norm(delta)
        f =1 / 2 * np.dot(np.dot(x,A),x) + np.dot(b,x)
        #uodate
        data_x.append(x)
        data_f.append(f)
        data_g.append(grad_length)
```

Algorithm 2.3. (Quasi-newton's method).

Step 1: Initialize $x^{(0)}$, k = 0, $H^{(0)} = I$

Step 2: Calculate the residual error vector by $grad^{(k)} = Ax^{(k)} + b$

Step3: Calculate the direction by $d^{(k)} = -grad^{(k)}$

Step 4: Calculate the step size by $H^{(k+!)} = H^{(k)} + \frac{\delta_k \delta_k{}^T}{\delta_k{}^T y_k} - \frac{H^{(k)} y_k y_k{}^T H^{(k)}}{y_k{}^T H^{(k)} y_k}$

Step 5: Set $\delta^{(k+1)} = x^{(k+1)} - x^{(k)}, f(x^{(k+1)}) - f(x^{(k)})$,

Step 6: Inexact search to get $\alpha^{(k)}$,

Step 7: Set $x^{(k+1)} = x^{(k)} + \alpha^{(k)} H^{(k)} d^{(k)}$.

Step 8: Set k=k+1 and go to Step 2.

Python code snippets are showed below.

```
def QuasiNewtonMethod(x,  A, b ,threshold, dim):
    #parameter set
    maxk = 1e5
    rho = 0.05
    sigma = 0.4
    epsilon = 1e-5
    k = 0
```

```
    n = np.shape(x)[0]
    #Hessian maitix
     Hk = np.eye(dim)/10
     grad = np.dot(A, x) + b
     grad_length = np.linalg.norm(grad)
     f = 1 / 2 * np.dot(np.dot(x, A), x) + np.dot(b, x)
     #init
     data_x = [x]
     data_f = [f]
     data_g = [grad_length]
     # main loop
     while k < maxk:

         if np.linalg.norm(grad) < epsilon:
             break
         dk = -1.0*np.dot(Hk,grad)

         #DFP update

         xk = x + dk
         sk = xk - x
         yk = np.dot(A,dk)

         if np.dot(sk,yk) > 0:
             Hy = np.dot(Hk,yk)
             sy = np.dot(sk,yk)
             yHy = np.dot(np.dot(yk,Hk),yk)
             #Hk = Hk - np.dot(Hy,Hy)/yHy + np.dot(sk,sk)
  /sy
             Hk = Hk - 1.0 * Hy.reshape((n, 1)) * Hy / yH
  y + 1.0 * sk.reshape((n, 1)) * sk / sy
         k += 1
         x = xk
         grad = np.dot(A, x) + b
         grad_length =  np.linalg.norm(grad)
         f = 1 / 2 * np.dot(np.dot(xk, A), xk) + np.dot(b
  , xk)
         # update
         data_x.append(xk)
         data_f.append(f)
         data_g.append(grad_length)
```

2.3 Program:

Please look through appendix I for the complete code.

III Results

3.1 Consistent convex

In the case that function f(x) is consistent convex, we compare the results of three methods with different initial points. Two kinds of initial point is set: one is near the optimal solution and another is away from optimal solution.

3.1.1 Three methods with initial pointer near x*

The optimal solution x* can be gained by $x^* = A^{-1}b$. So the initial pointer can be initialized by tuning x*. The following picture is matrix.

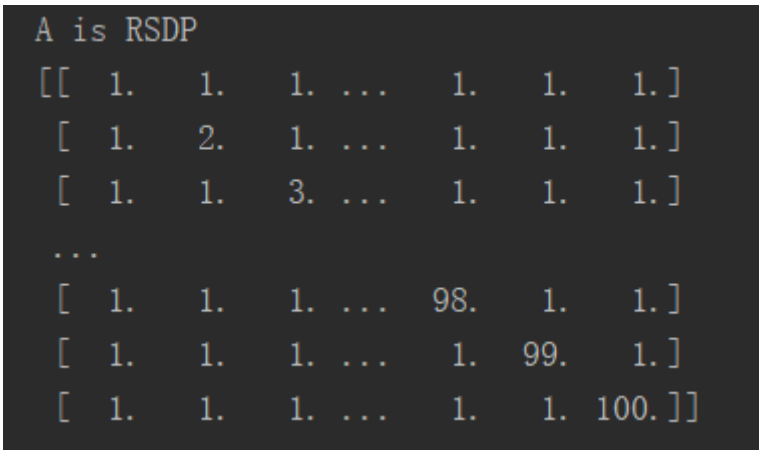


Fig 3.1 the matrix A that is RSDP

The result is showed below:

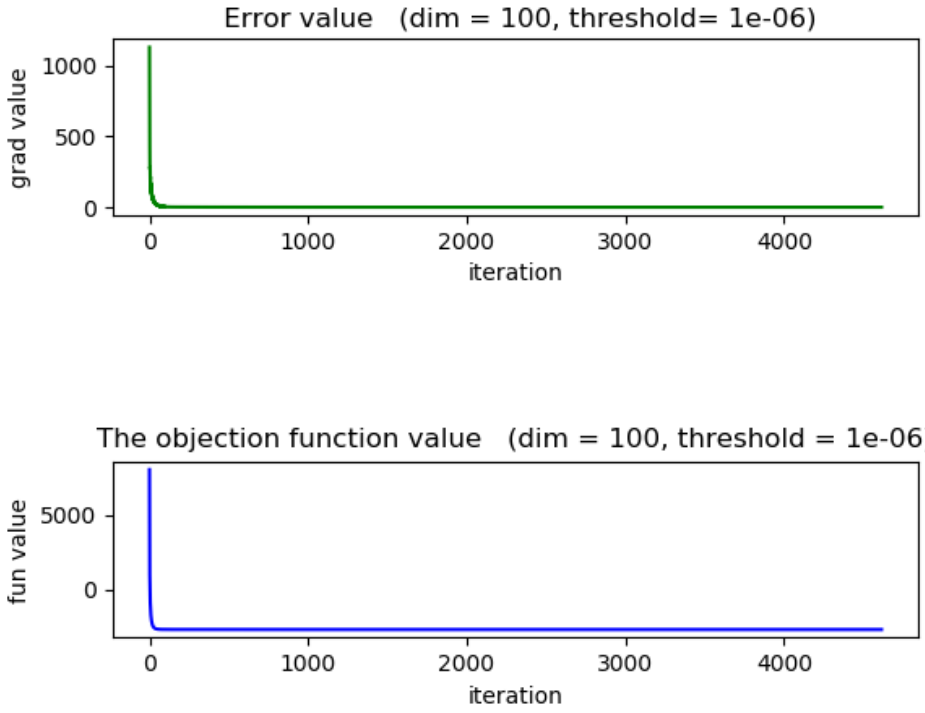


Fig 3.2 This is the result of steepest descent method (SDM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. However, x is converged to x* and the value of function minimum in the special threshold.

Fig 3.2 show the values of gradient and objective function keep decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. However, x is converged to x* and the value of function minimum in the special threshold.

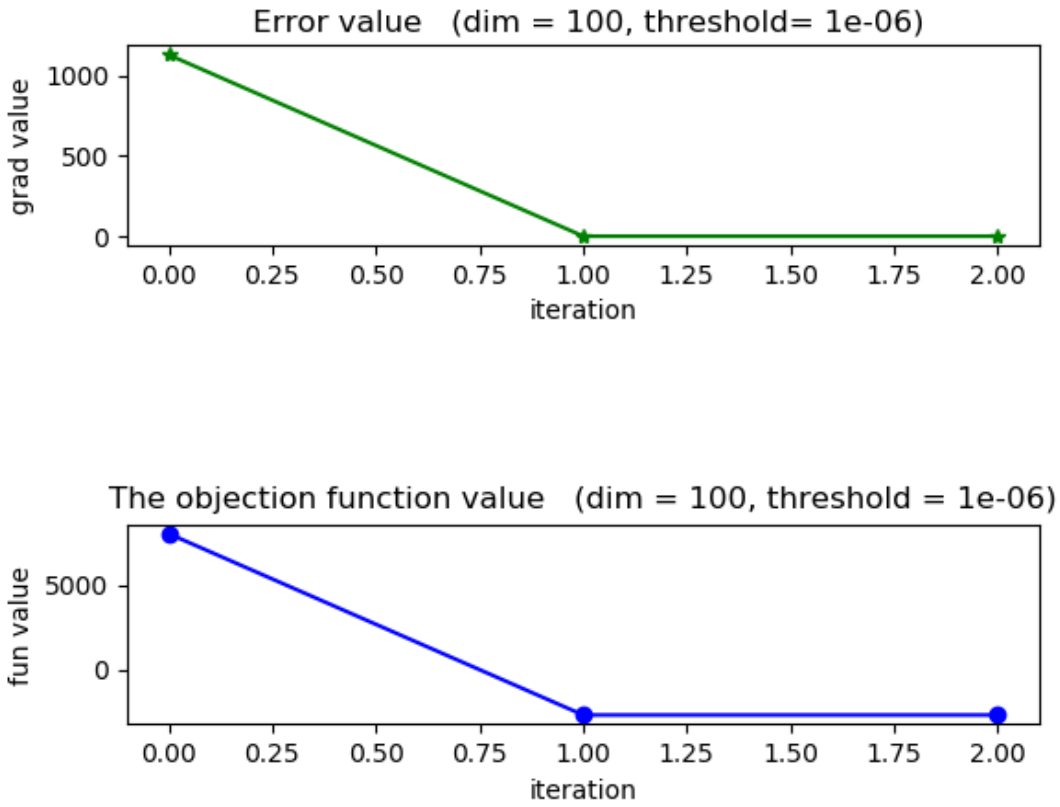


Fig 3.3 This is the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM.

Fig 3.3 shows the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM.

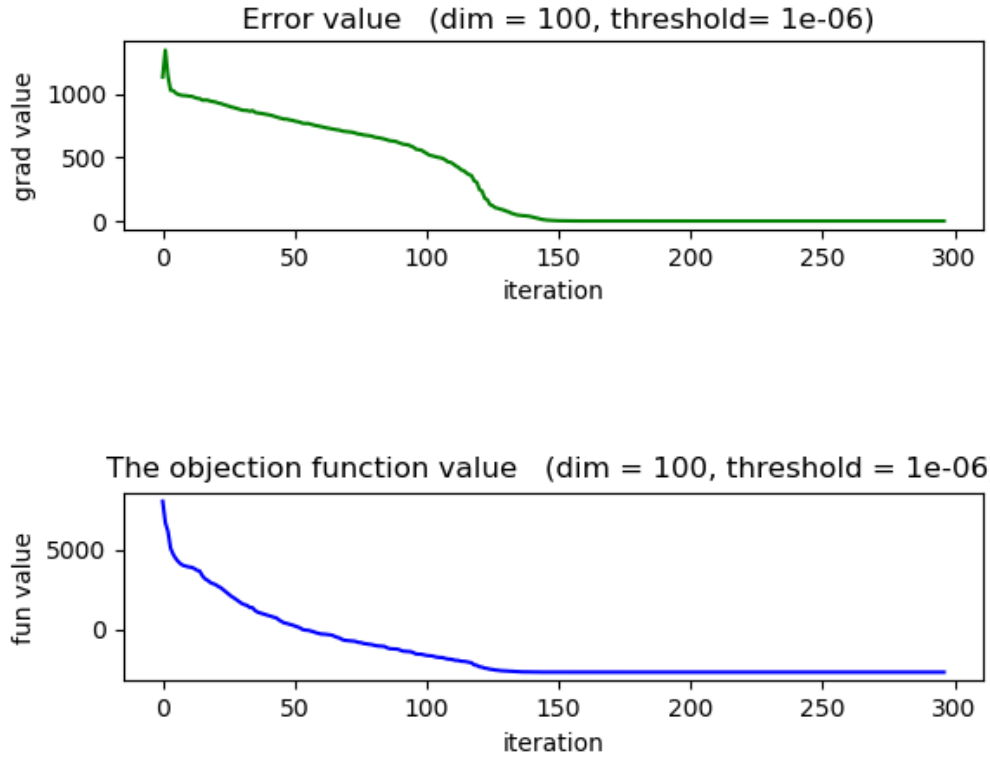


Fig 3.4 This is the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function decrease and finally are the optimal solution. The rate of convergence is fast than SDM but slower than NM

Fig 3.4 shows the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function oscillate and finally are the optimal solution. The rate of convergence is fast than SDM but slower than NM

Table 3.1 The Optimal function value of three methods with the initial point near x*

method	Optimal function value
SDM	-2637.215487036301
NM	-2637.215487056671
QNM	-2637.2154870566683

The function values of three methods are very close. It is reasonable to

think that they all converged to optimal value.

3.1.2 Three methods with initial pointer away from x*

The optimal solution x* can be gained by $x^* = A^{-1}b$. So the initial pointer can be initialized by x* multiplying 56 and plus 100. The result is showed below:

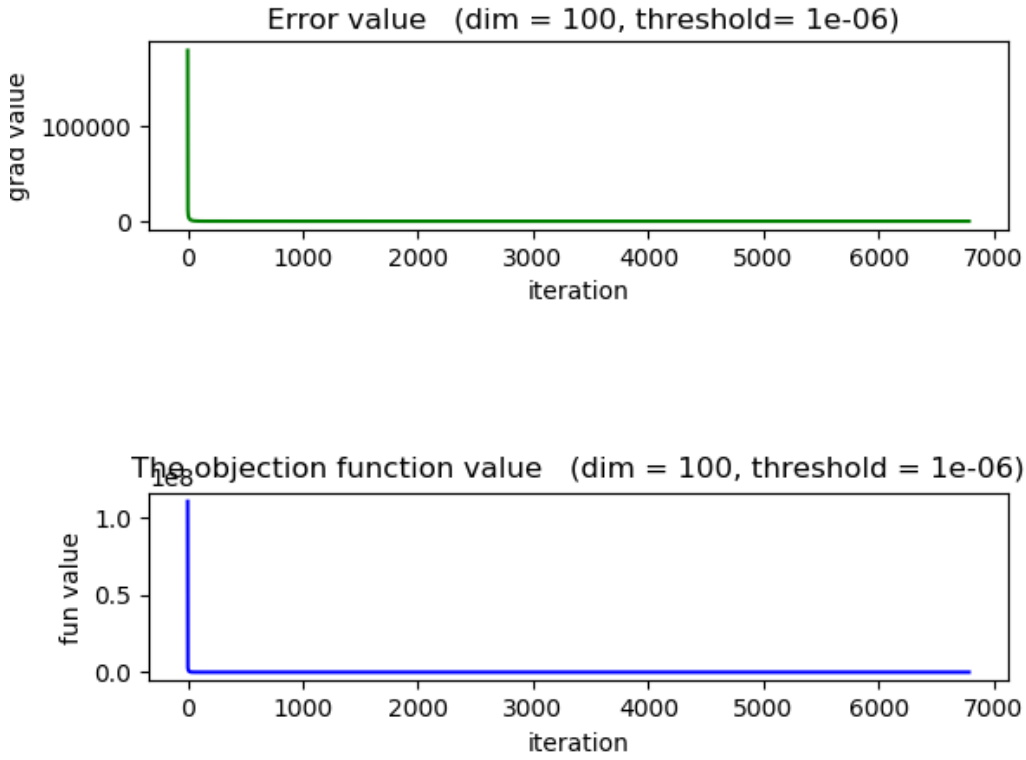


Fig 3.5 This is the result of steepest descent method (SDM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. However, x is converged to x* and the value of function minimum in the special threshold. Time costs is more than before case that initial point near x*.

Fig 3.5 show the values of gradient and objective function keep decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. However, x is converged to x* and the value of function minimum in the special

threshold. Time costs is more than before case that initial point near x*.

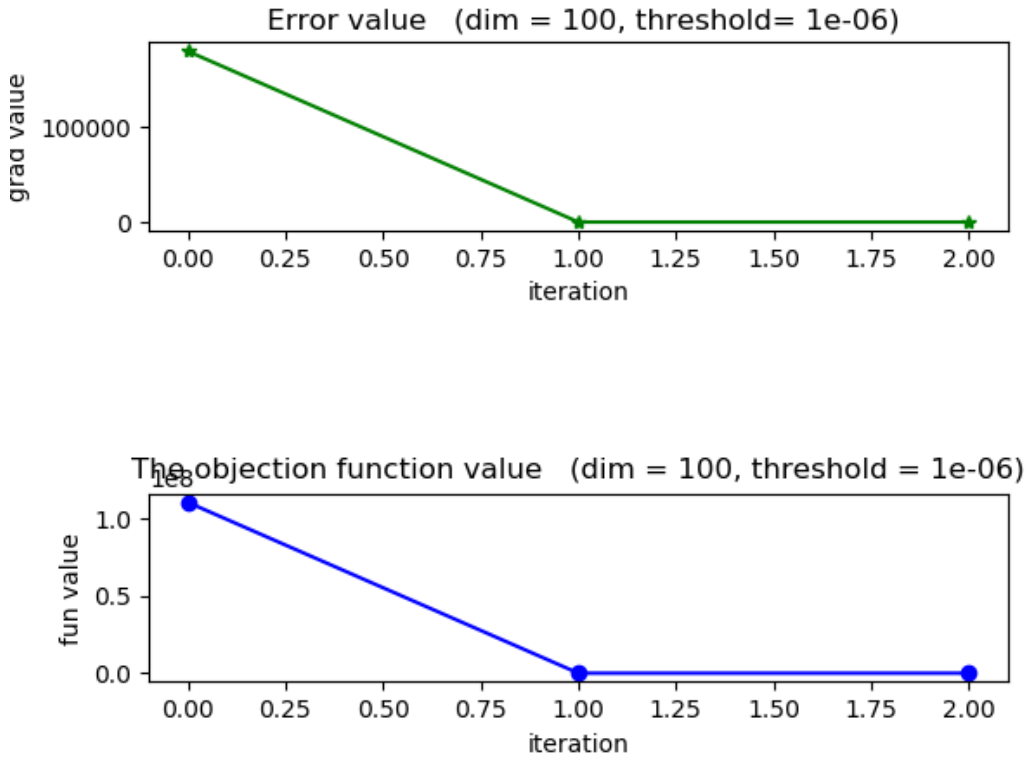
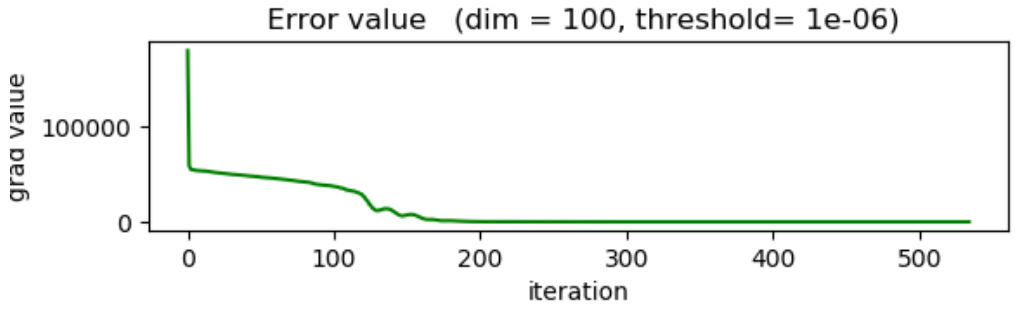


Fig 3.6 This is the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM. There are same iteration times in two different initial point.

Fig 3.6 shows the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM.



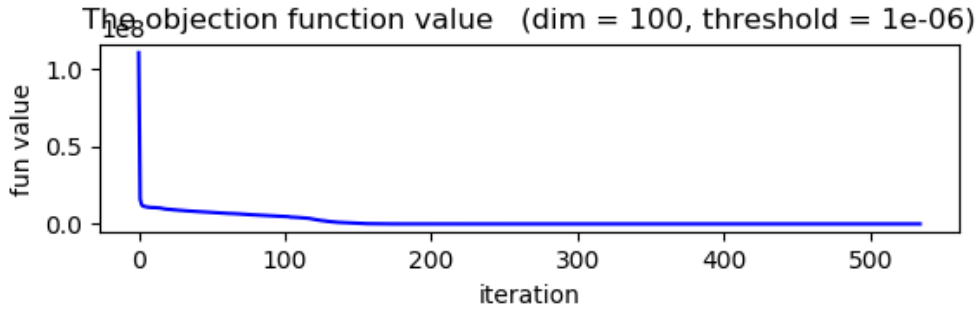


Fig 3.7 This is the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function keep decreasing and finally is the optimal solution. The rate of convergence is fast than SDM but slower than NM. Time costs is more than before case that initial point near x*

Fig 3.7 shows the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function oscillate and finally are the optimal solution. The rate of convergence is fast than SDM but slower than NM. Time costs is more than before case that initial point near x*

Table 3.2 The Optimal function value of three methods with initial point away from x*

method	Optimal function value
SDM	-2637.215487045589
NM	-2637.215487056671
QNM	-2637.215487056672

The function values of three methods are very close. It is reasonable to think that they all converged to optimal value.

For the case that function is consistent convex, three method all converge and can get optimal solution in limit step. Newton's method always performs well with different initial points.

3.2 Bounded convex

3.2.1 Three methods with initial pointer near x*

The optimal solution x* can be gained by $x^* = A^{-1}b$. So the initial pointer can be initialized by tuning x*. The following picture is matrix.

```
A is real-symmetric-semi-define-positive
[[ 1.  0.  0. ...  0.  0.  0.]
 [ 0.  2.  0. ...  0.  0.  0.]
 [ 0.  0.  3. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  8.  0.  0.]
 [ 0.  0.  0. ...  0.  9.  0.]
 [ 1.  2.  3. ...  8.  9. 10.]]
```

Fig 3.8 the matrix A that is RDP

A is not symmetric, So function f(x) is not consistent convex. The surface will be like this and have some local minimizer.

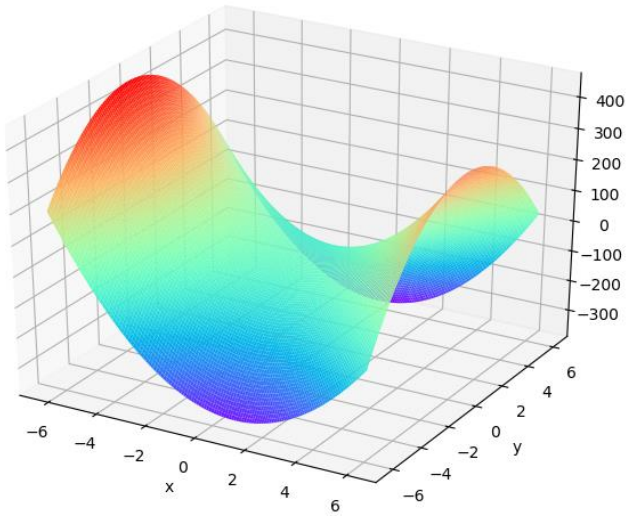


Fig 3.9 Bounded convex

The result is showed below:

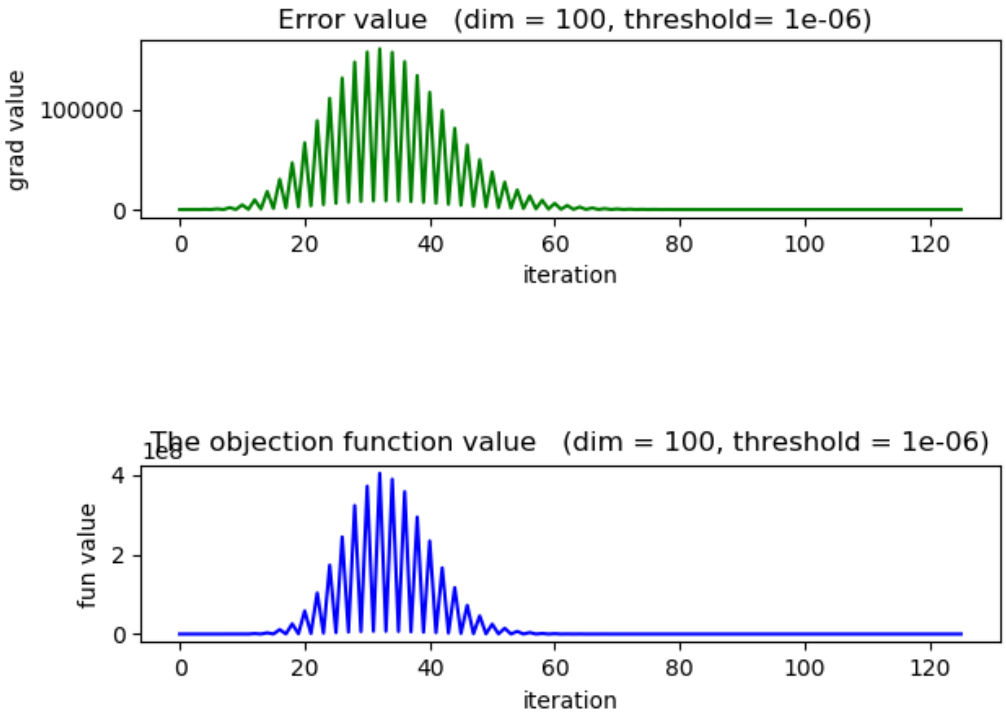


Fig 3.10 This is the result of steepest descent method (SDM) with initial point near to x* .The values of gradient and objective function oscillate and eventually we obtained the local minimum

This is the result of steepest descent method (SDM) with initial point near to x*.The values of gradient and objective function oscillate and eventually

we obtained the local minimum.

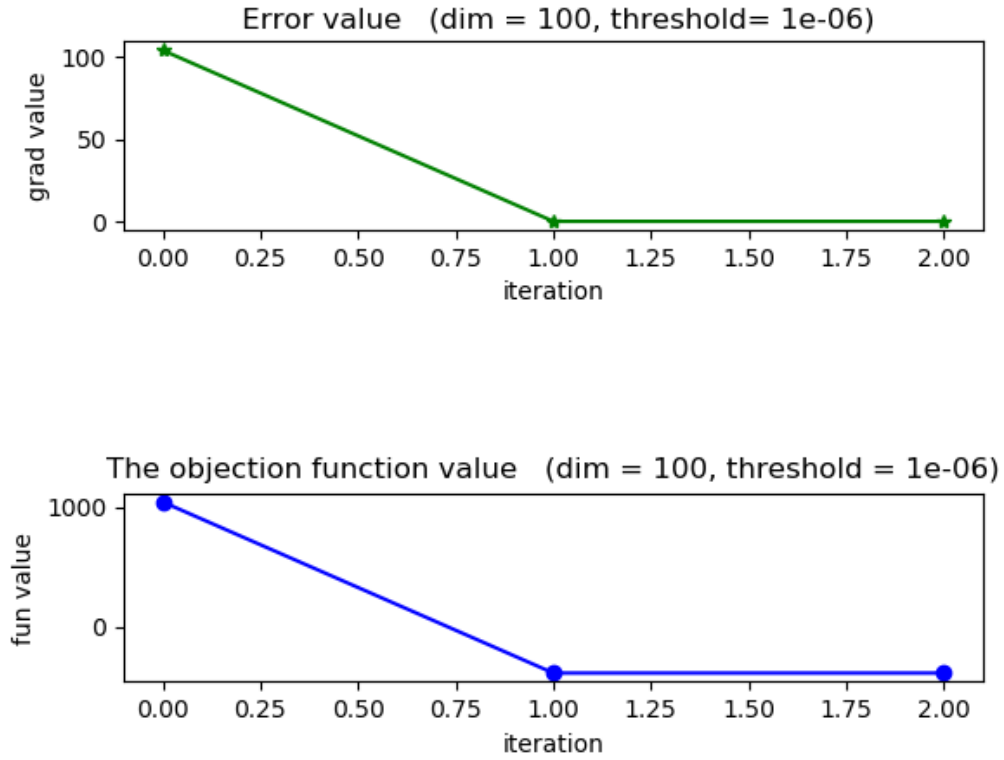


Fig 3.11 This is the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM. More important, the minimum is global.

Fig 3.11 shows the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM. What's more, the minimum is global.

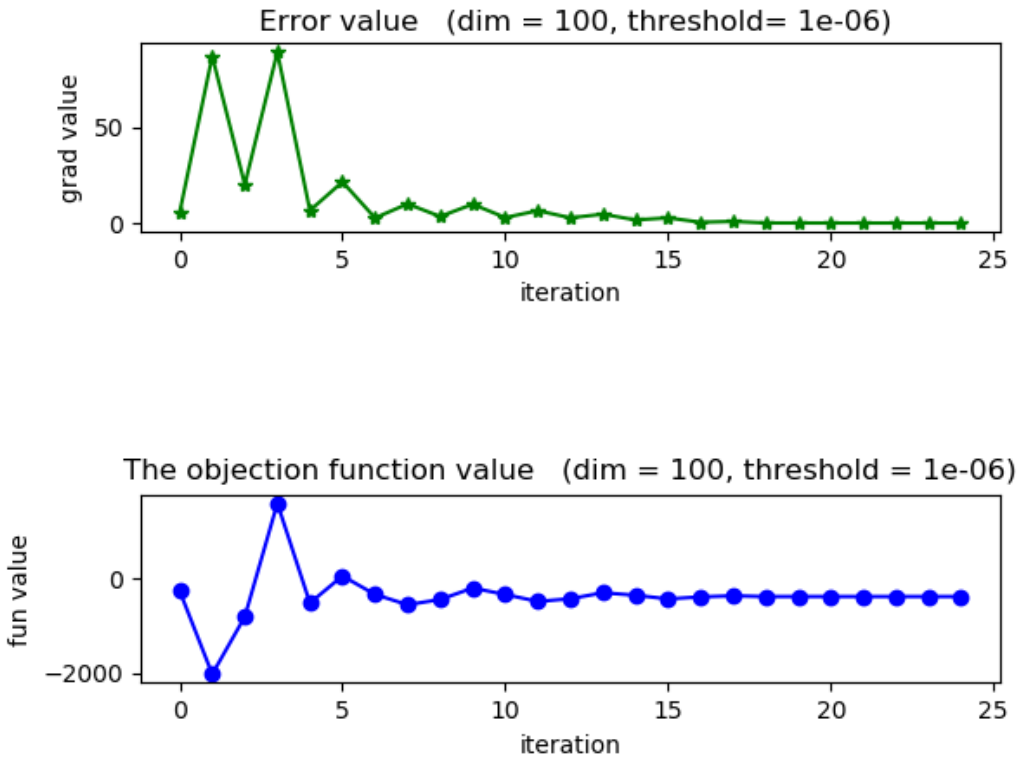


Fig 3.12 This is the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function, the value of gradient oscillates and finally is the optimal solution. The rate of convergence is fast than SDM but slower than NM

Fig 3.12 shows the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function oscillate and finally are the optimal solution. The rate of convergence is faster than SDM but slower than NM

Table 3.3 The Optimal function value of three methods with the initial point near x*

method	Optimal function value
$x^* = A^{-1}b$	-383.767658730158
SDM	-383.7676602327597
NM	-383.76765873015984
QNM	-383.7676580563427

The function values of three methods are very close. It is reasonable to think that they can converged to optimal value in the case that function f(x) is bounded convex and initial point is near x*, but it is not global minimum. In contrast, SDM obtains the local minimum in that case and has an expense iteration times cost. What's more, when we set x* is $x^* = A^{-1}b$, the optimal function values also is closed to others solution, so we can conclude that even $x^* = A^{-1}b$ is not the exact solution because of the precision of computer.

The condition number of A is 10 , so A is not illness. However, the gradient is very large during the iteration, which is the possible reason contribute to local minimum.

3.2.2 Three methods with initial pointer away from x*

The optimal solution x* can be gained by $x^* = A^{-1}b$. So the initial pointer can be initialized by the following code snippets.

```
x0 = -56*np.dot(np.linalg.inv(A),b) + np.random.randn(dim)
```

The result is showed below:

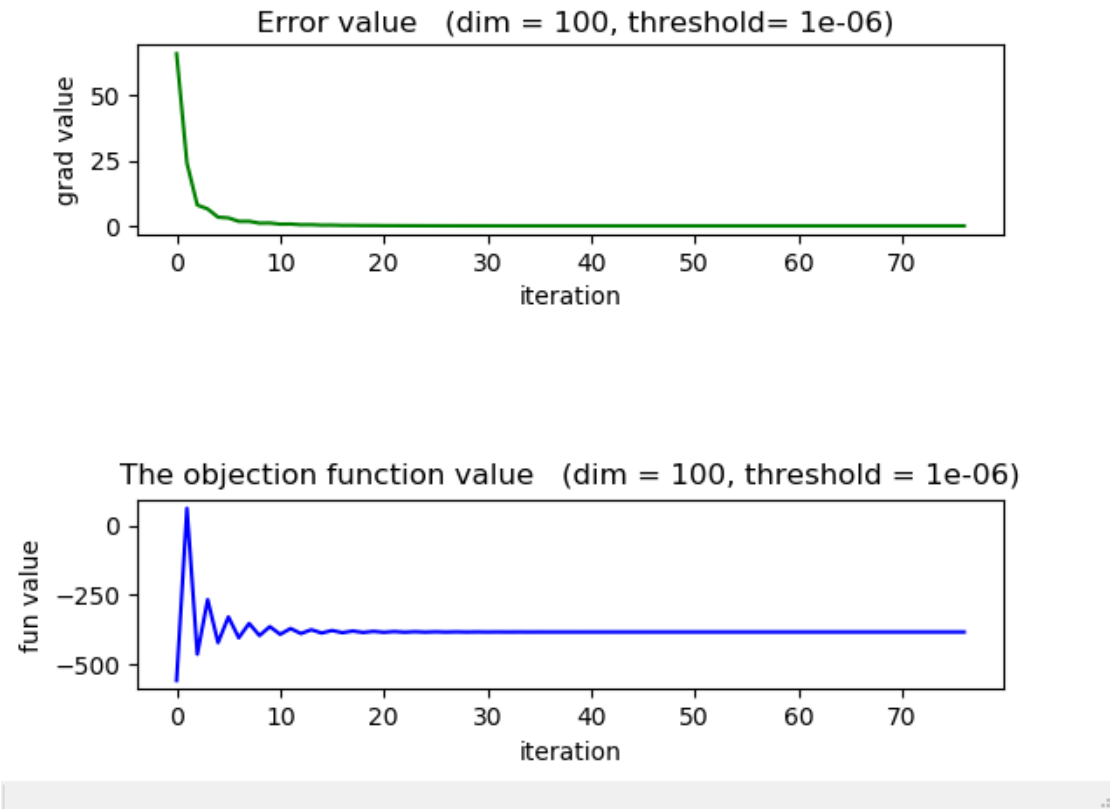


Fig 3.13 This is the result of steepest descent method (SDM) with initial point near to x* . At the beginning, function value decreases fast and when x nears the local x*, the function value changes slowly. Suddenly value of gradient becomes large and gain a smaller value.

Fig 3.13 This is the result of steepest descent method (SDM) with initial point near to x*. At the beginning, function value decreases fast and when x nears the local x*, the function value changes slowly. Suddenly value of gradient becomes large and gain a smaller value.

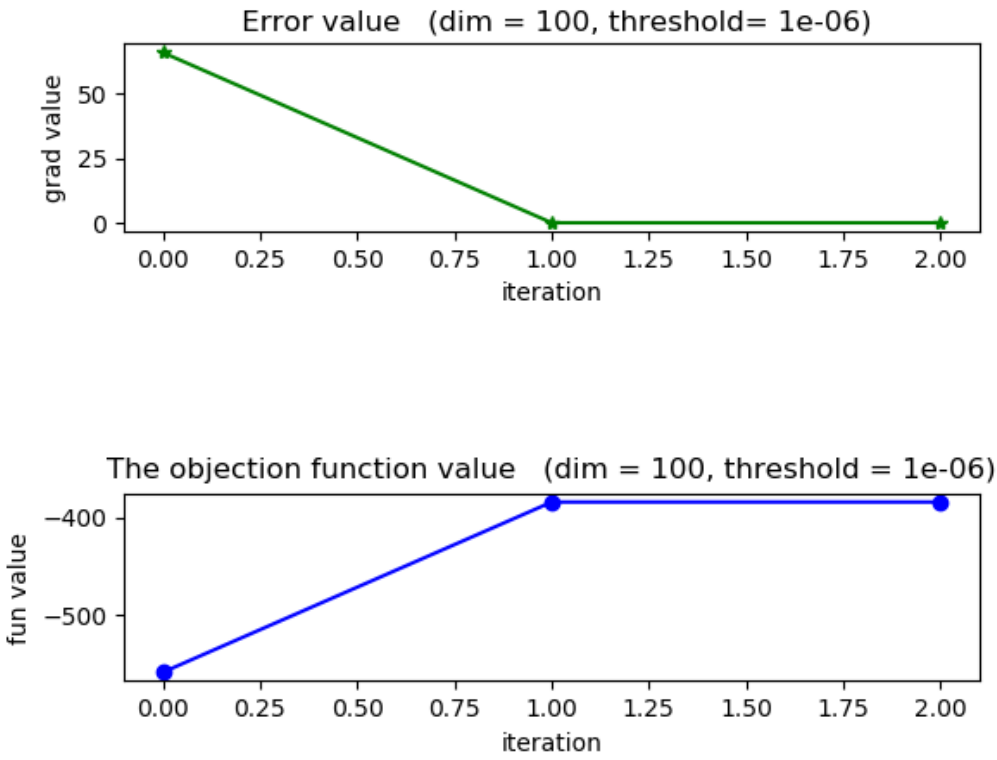


Fig 3.14 This is the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function, the values of gradient keep decreasing until it's zero. The rate of convergence is fast than SDM. There are same iteration times in two different initial point.

Fig 3.14 shows the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function, the values of gradient keep decreasing until it's zero. The rate of convergence is fast than SDM. From the result of function value, we can concluded that NM is converge to a local minimum.

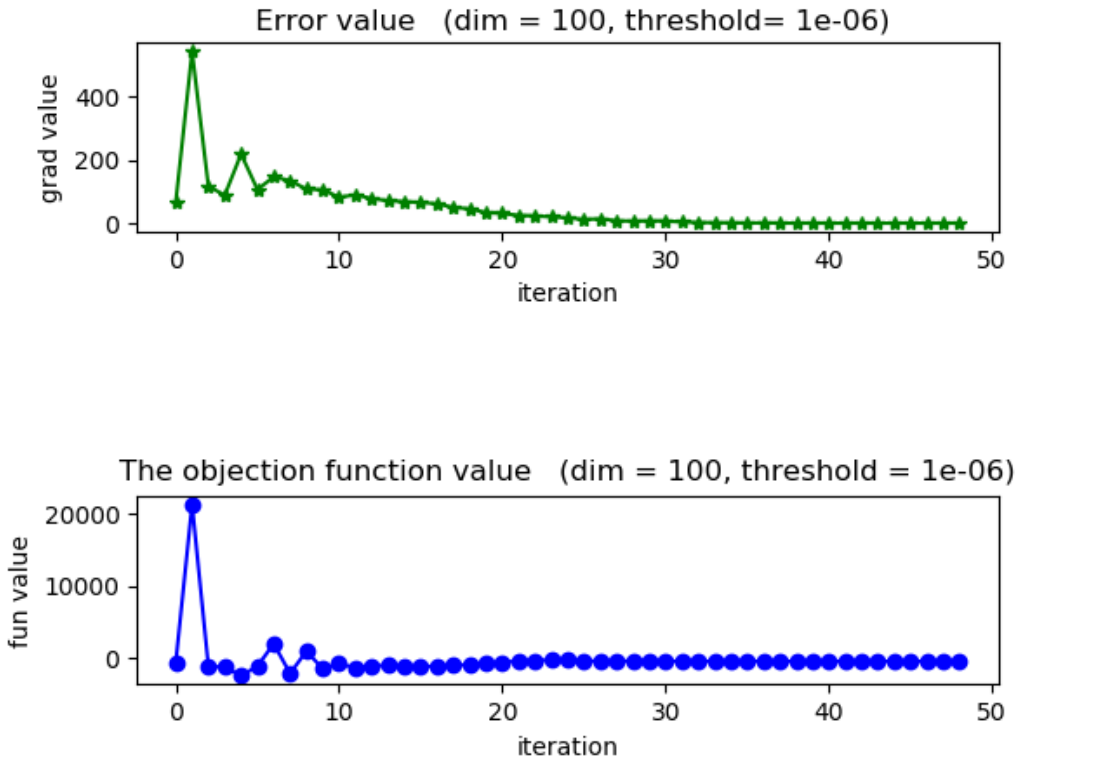


Fig 3.15 This is the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function, the value of gradient oscillates and finally is the optimal solution. The rate of convergence is fast than SDM but slower than NM. Time costs is more than before case that initial point near x*

Fig 3.15 shows the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function, the value of gradient oscillates and finally is the optimal solution. The rate of convergence is fast than SDM but slower than NM. Time costs is more than before case that initial point near x*. From the result of SDM (fig3.12) and NM （fig3.1） , we can concluded that SNM is also converge to a local minimum.

Table 3.4 The Optimal function value of three methods with initial point away from x*

method	Optimal function value
$x^* = A^{-1}b$	-383.767658730158
SDM	-383.76765873015984
NM	-383.767658730158
QNM	-383.7676477608311

For the case that function is bounded convex, three method all converge and can get optimal solution in limit step. NM and QMN can gain local minimum in less iterate times.

3.3 Convex

In the case that function f(x) is convex, comparing the results of three methods with different initial points. Two kinds of initial point is set: one is near the optimal solution and another is away from optimal solution. The optimal solution x* can't be gained by $x^* = A^{-1}b$ because A maybe not a full rank matrix. So the initial pointer can be initialized by random. Then x can be initial from the optimal solution of the result of method with random initial pointer.

3.3.1 Three methods with random initial pointer

The optimal solution x* can't be gained by $x^* = A^{-1}b$ because A maybe not a full rank matrix. So the initial pointer can be initialized by random. The following picture is the matrix A.

```
[[ 5.20876613  0.36722402 -0.51212708 ... -0.20847244  0.16177928
  -0.40833212]
 [ 0.36722402  4.66377708  0.20005786 ... -0.06384606  0.15378633
  -0.06669551]
 [-0.51212708  0.20005786  5.67666576 ...  0.27653419 -0.22666594
  -0.09611904]
 ...
 [-0.20847244 -0.06384606  0.27653419 ...  5.04421696 -0.61365729
  -0.45723277]
 [ 0.16177928  0.15378633 -0.22666594 ... -0.61365729  5.18528027
  -0.12750139]
 [-0.40833212 -0.06669551 -0.09611904 ... -0.45723277 -0.12750139
   5.05282269]]
```

Fig 3.16 the matrix A that is RSSDP

The result is showed below:

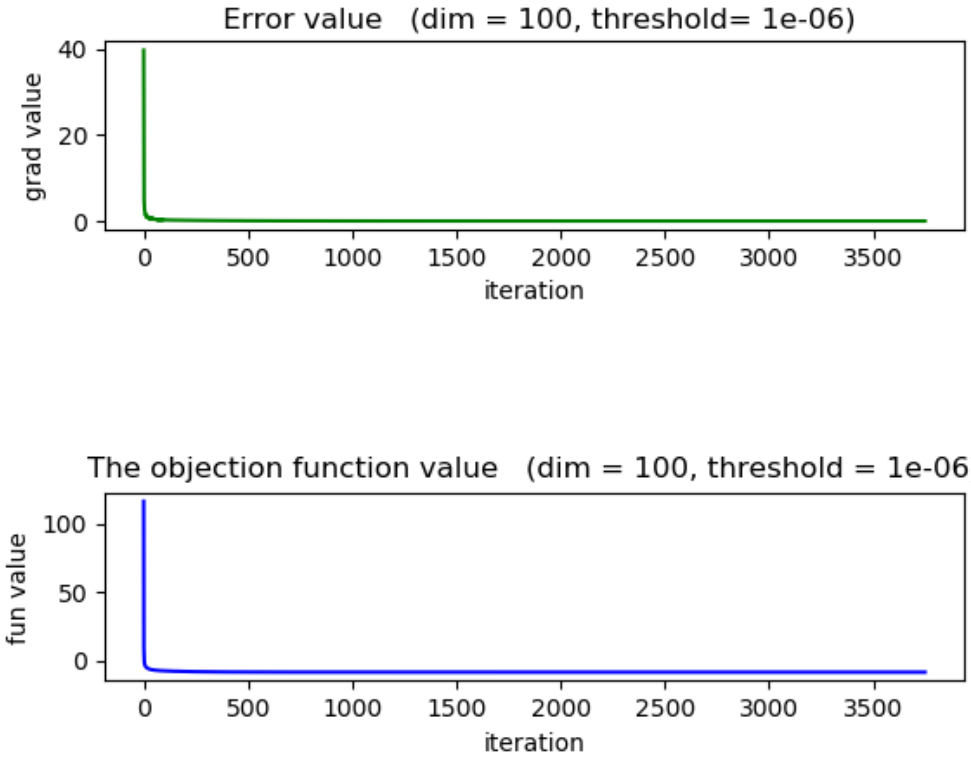


Fig 3.17 This is the result of steepest descent method (SDM) with random initial point.The values of gradient and objective function keep decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. However, x is converged to x* and the value of function minimum in the special threshold.

Fig 3.17 show the values of gradient and objective function keep

decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. However, x is converged to x* and the value of function minimum in the special threshold.

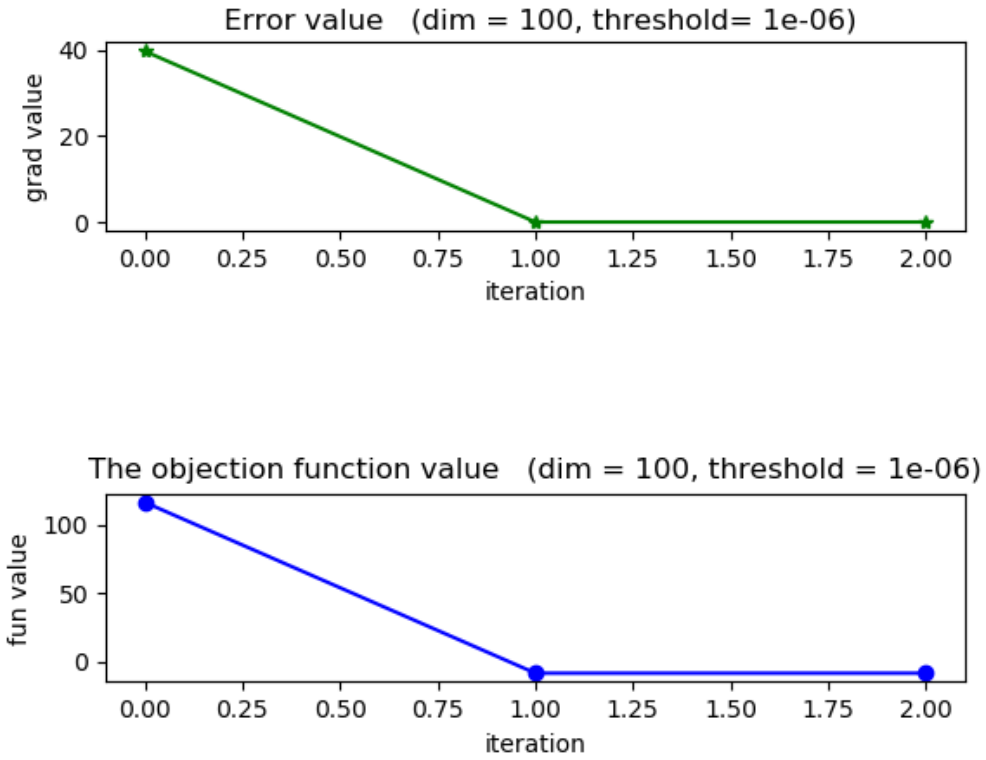


Fig 3.18 This is the result of newton's method (NM) with random initial point .The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM.

Fig 3.18 shows the result of newton's method (NM) with random initial point. The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM.

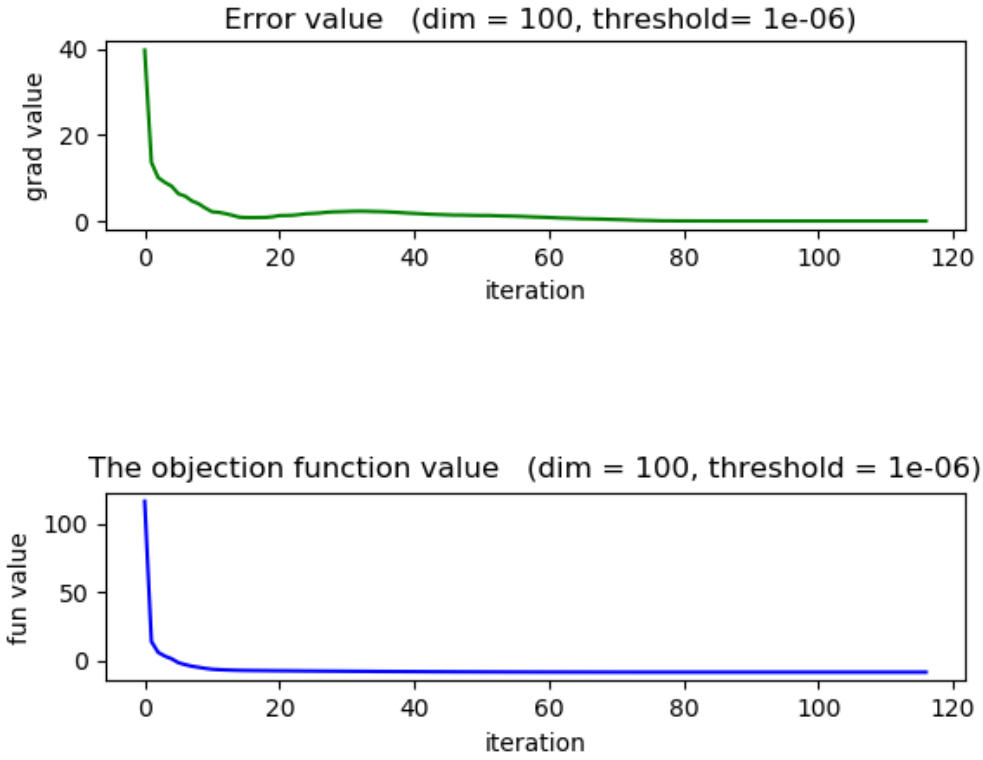


Fig 3.19 This is the result of quasi-newton's method (QNM) with random initial point. The values of gradient and objective function decrease and finally are the optimal solution. The rate of convergence is fast than SDM but slower than NM

Fig 3.19 shows the result of quasi-newton's method (QNM) with random initial point .The values of gradient and objective function oscillate and finally are the optimal solution. The rate of convergence is fast than SDM but slower than NM

Table 3.5 The Optimal function value of three methods with the initial point near x*

method	Optimal function value
SDM	-8.937134551651237
NM	-8.937134551653418

QNM	-8.937134551653351

The function values of three methods are very close. It is reasonable to think that they all converged to optimal value.

3.3.2 Three methods with initial pointer near x*

From the above result, we can gain the optimal solution x*,so we can tune the x* as the initial pointer .

The result is showed below:

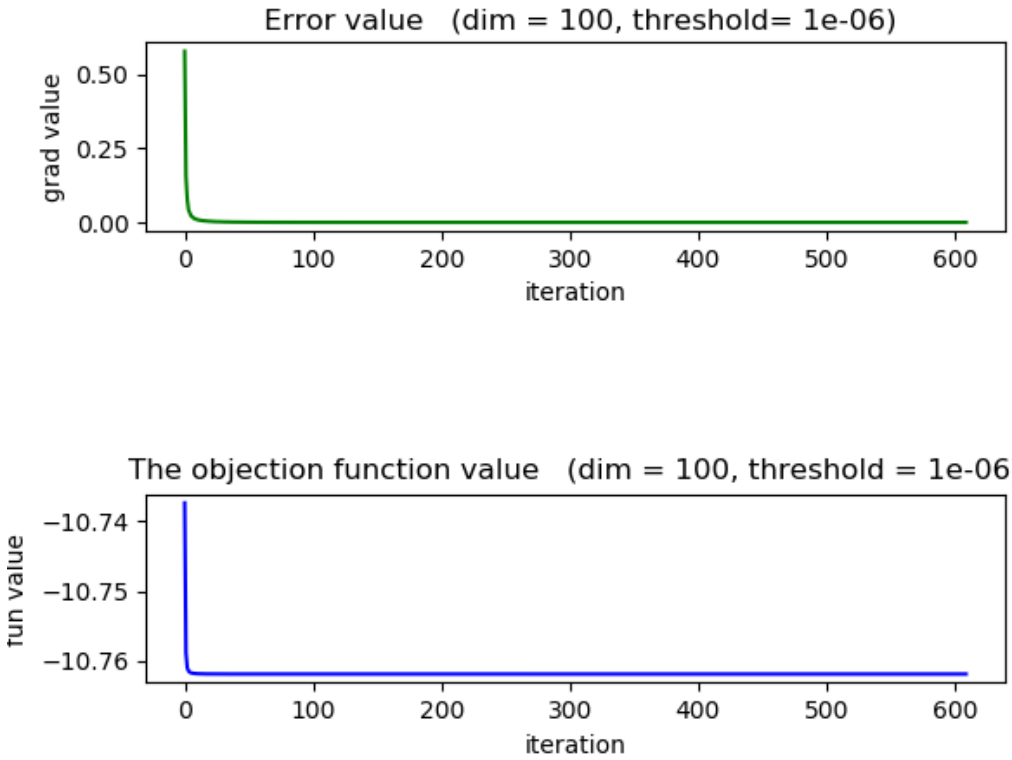


Fig 3.20 This is the result of steepest descent method (SDM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. It is worth mentioning that the iteration times is smaller than before case with random initial point

Fig 3.20 the result of steepest descent method (SDM) with initial point

near to x* .The values of gradient and objective function keep decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. It is worth mentioning that the iteration times is smaller than before case with random initial point.

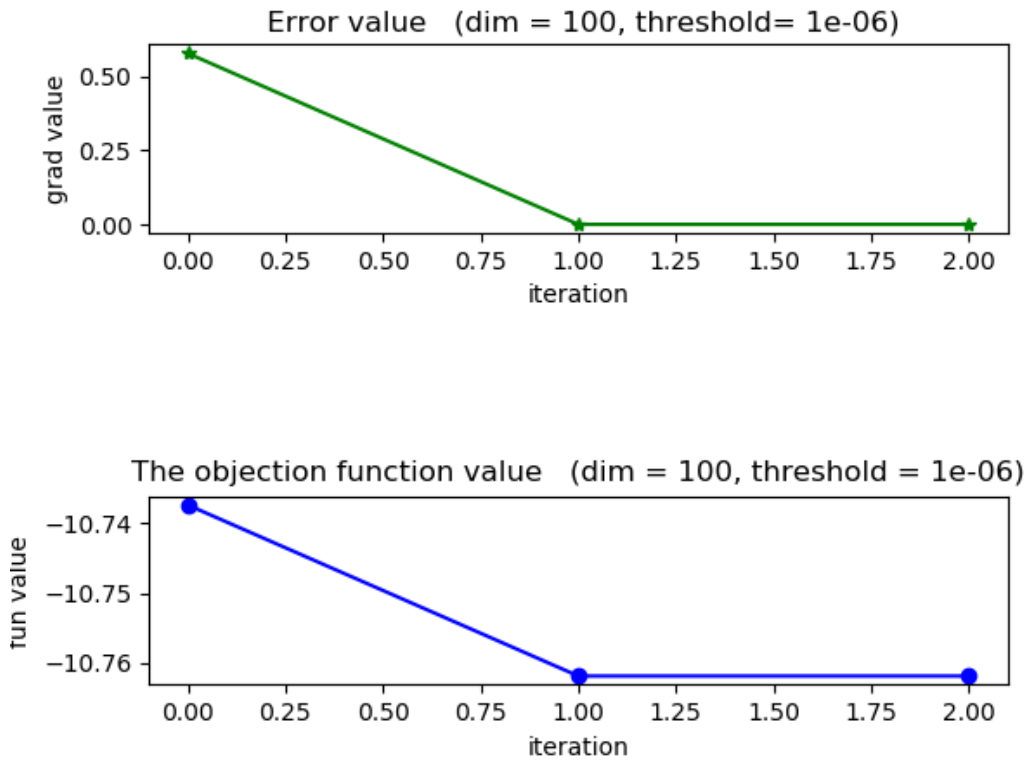


Fig 3.21 This is the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. There are same iteration times with two different initial point

Fig 3.21 the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. There are same iteration times with two different initial point. For the function f(x) is convex, NM always gains the minimum with different initial points. The main reason is that local minimum equals to

global minimum for convex function.

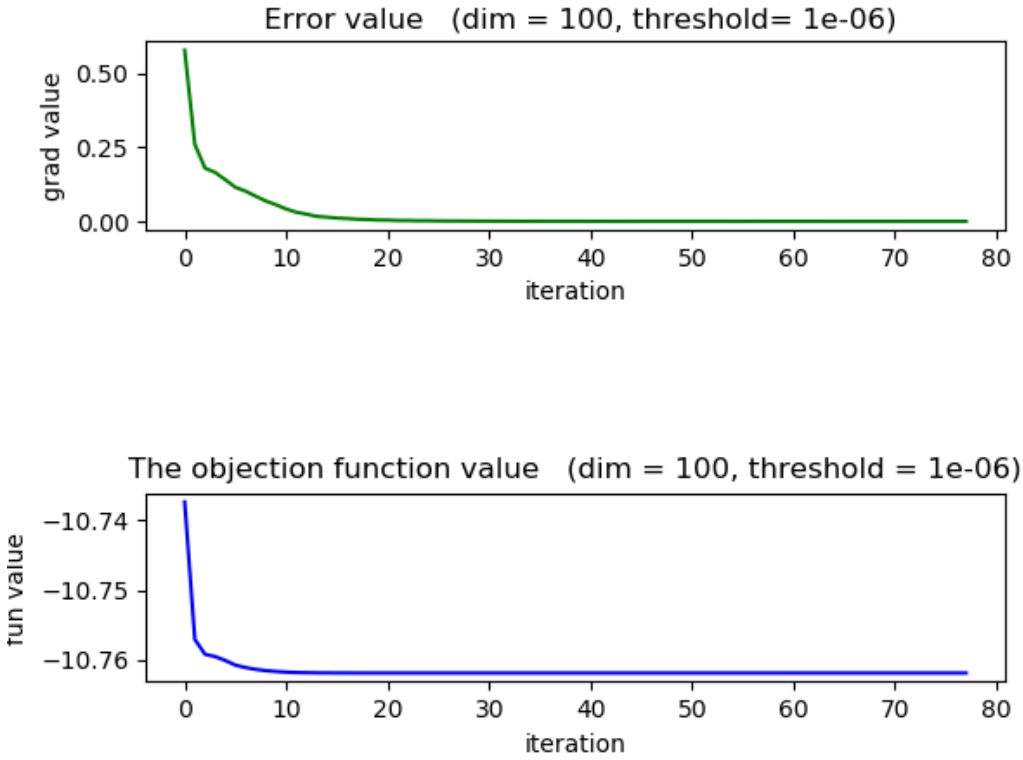


Fig 3.22 This is the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function decrease and finally are the optimal solution. The rate of convergence is fast than SDM but slower than NM. The iteration times also decrease.

Fig 3.22 shows the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function oscillate and finally are the optimal solution. The rate of convergence is fast than SDM but slower than NM. The iteration times also decrease.

For the function f(x) is convex, NM always gains the minimum in two different initial points. The main reason is that local minimum equals to global minimum for convex function.

3.3.3 Three methods with initial pointer away from x*

The initial point x0 ,x0 = x* plus 10* np.random.randn(dim), is away from x*. the following shows the results.

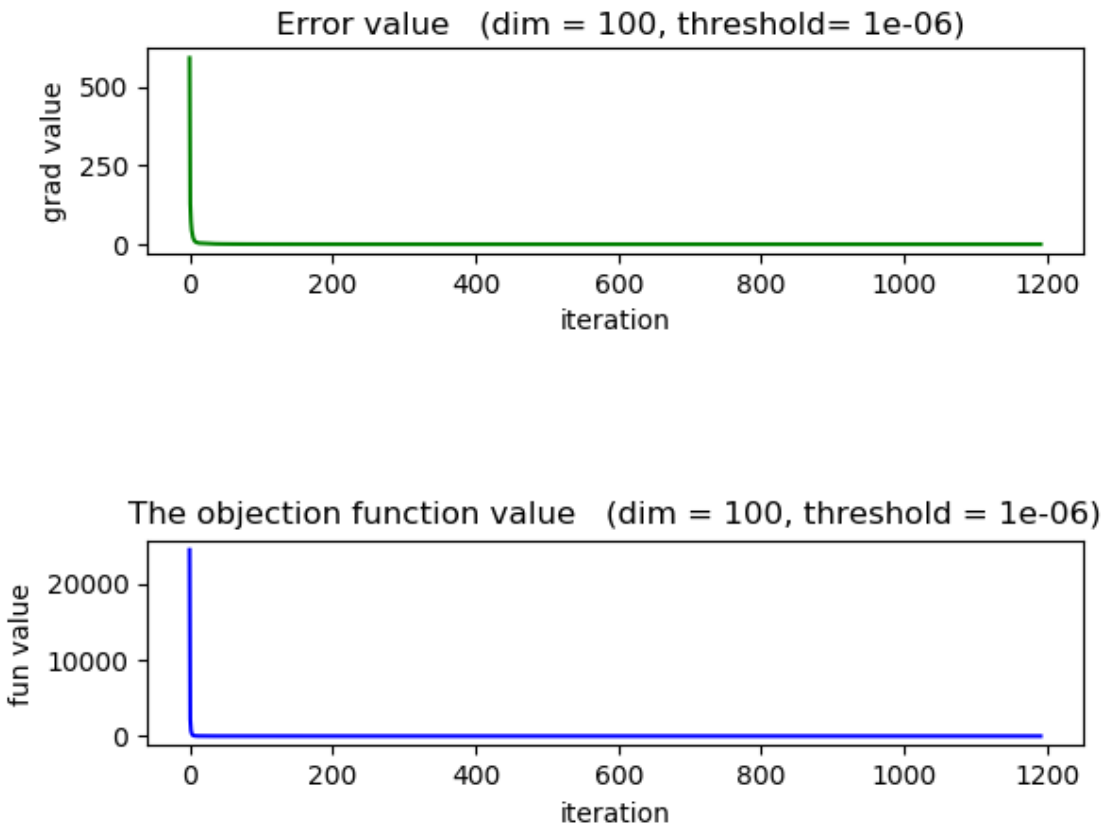


Fig 3.23 This is the result of steepest descent method (SDM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. However, x is converged to x* and the value of function minimum in the special threshold. Time costs is more than before case that initial point near x*.

Fig 3.23 show the values of gradient and objective function keep decreasing until it's zero. At the beginning, function value decreases fast and when x nears the x*, the function value changes slowly. However, x

is converged to x* and the value of function minimum in the special threshold. Time costs is more than before case that initial point near x*.

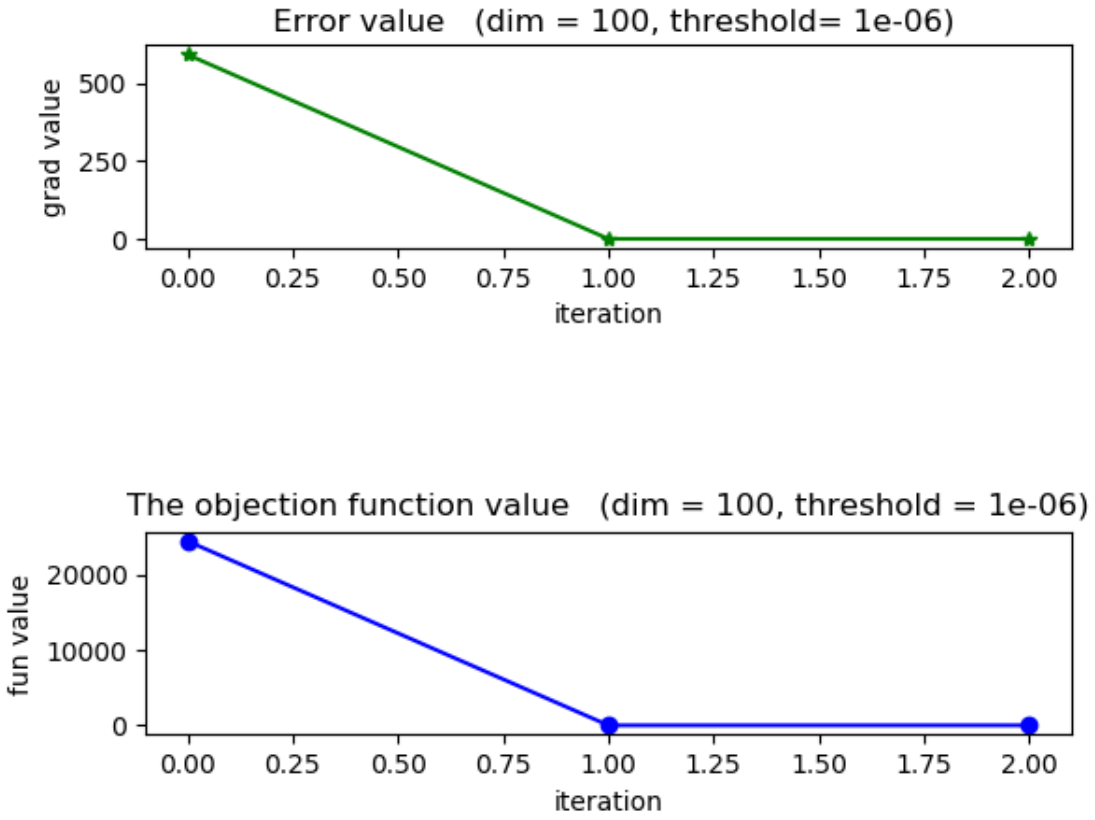


Fig 3.24 This is the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM. There are same iteration times in two different initial point.

Fig 3.24 shows the result of newton's method (NM) with initial point near to x* .The values of gradient and objective function keep decreasing until it's zero. The rate of convergence is fast than SDM.

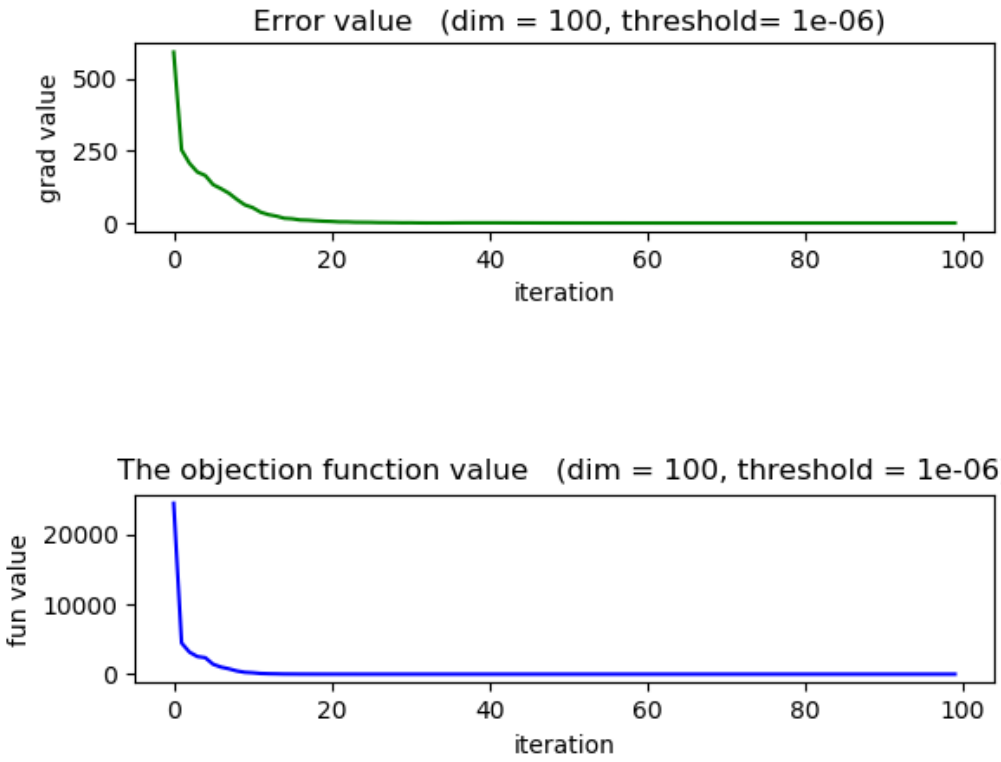


Fig 3.25 This is the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function keep decreasing and finally is the optimal solution. The rate of convergence is fast than SDM but slower than NM. Time costs is more than before case that initial point near x*

Fig 3.25 shows the result of quasi-newton's method (QNM) with initial point near to x* .The values of gradient and objective function oscillate and finally are the optimal solution. The rate of convergence is fast than SDM but slower than NM. Time costs is more than before case that initial point near x*

For the function f(x) is convex, NM always gains the minimum in three different initial points. More iteration times is cost for other methods with initial point away from x*.

3.4 An example

Take low–dimension, dim =2, for example. We plot the path from initial to destination.The results are showed below. The order is SDM, NM and QNM.

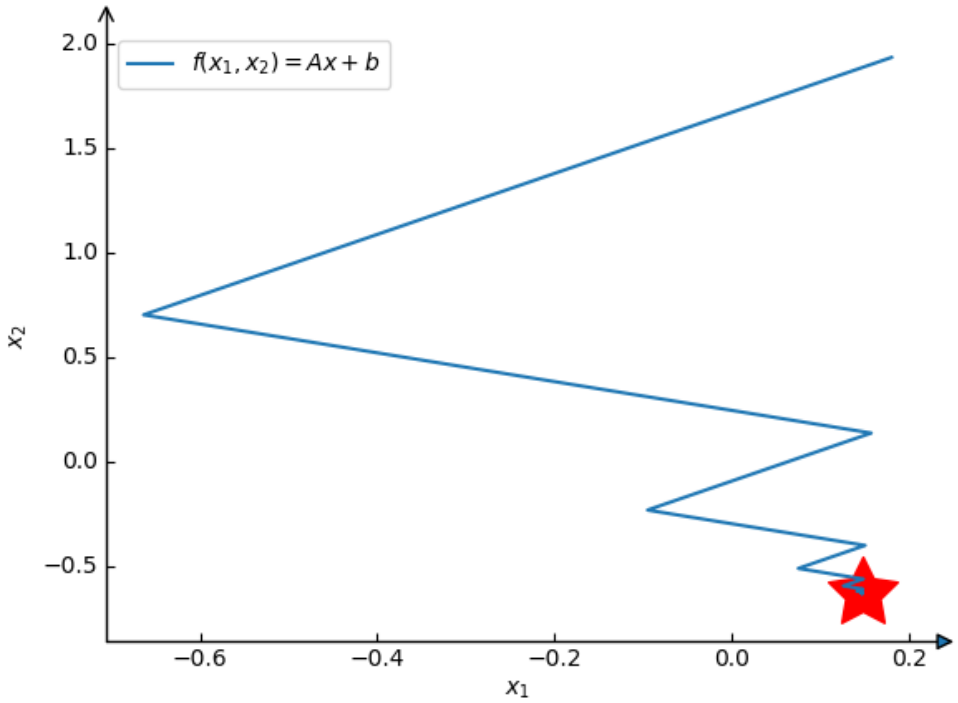


Fig3.26a SDM

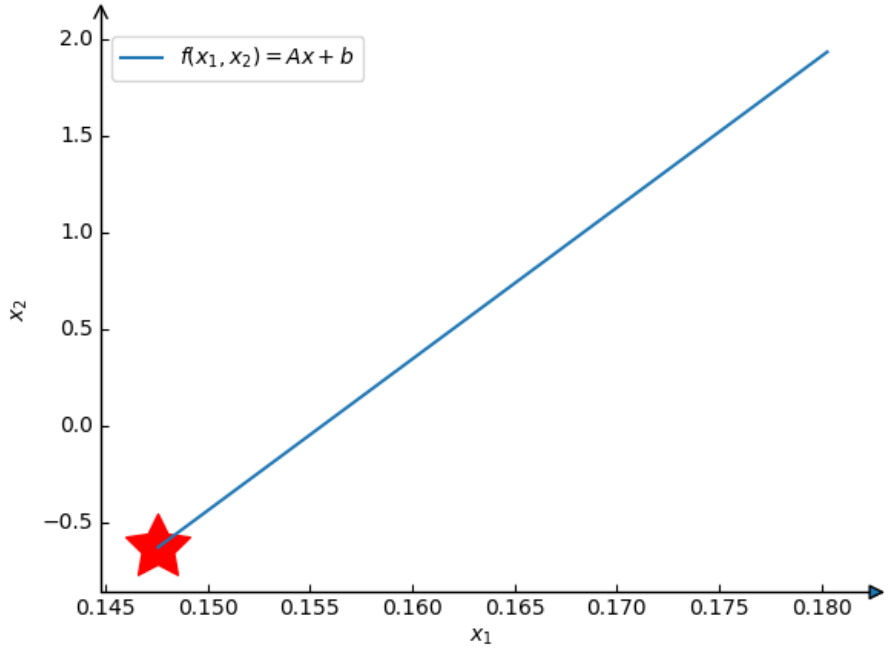


Fig3.26b NM

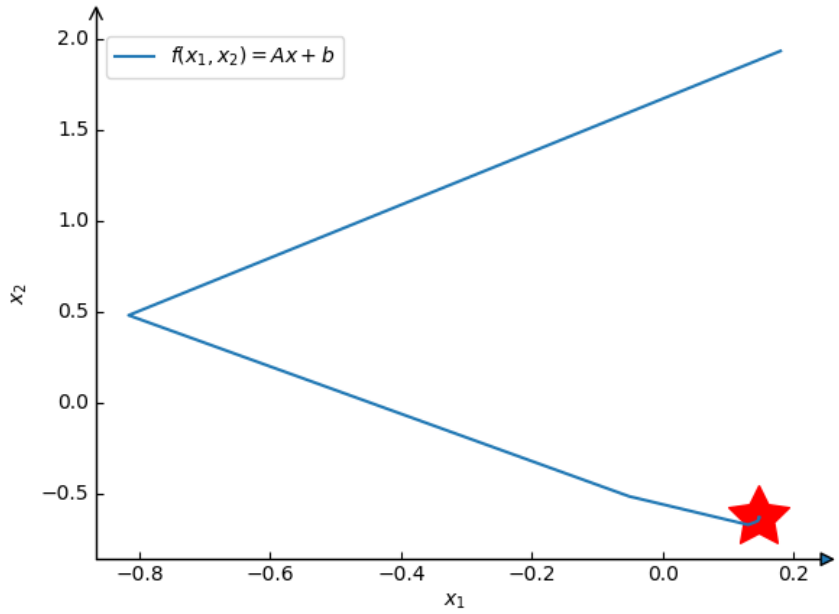


Fig3.26c QNM

For the case that function f(x) is convex, SDM is the slow algorithm from a global perspective even though the direction is the steepest in every iteration. We also found the directions of SDM is orthogonal. NM is the fastest among the three method.

IV Conclusion and acquirement

For the quadratic optimization:

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

Find the minimizer of the higher-dimensional quadratic optimization for the respective consistent convex, bounded convex and convex cases with different initial points by any 3 methods of conjugate gradient method,

steepest descent method, Newton's method and Quasi Newton method. From this homework, we main solve two sub problem:

1. to generate different matrix corresponding to above three cases.
2. to implement the optimization methods above.
3. to set up experiences to compare the convergence and rate of convergence.

We observe the oscillation occurs for bounded convex. Because the conditional number of A is small and the method is true, it is reasonable to think that lager gradient influences the performance of method.

In all experience, the iteration times of SDM are always the most, so we can concluded that SDM is the slow algorithm from a global perspective.

Finally we also find that $A^{-1}b$ is not always, especially the function has many local minimum.

In brief, if f(x) is convex and 2nd –order continuously differentiable. NM is the good choice. In this case, the convergence rate of newton is the fastest. If matrix inversion cost too much resource, QNM is the good choice. We don't recommend SDM because it is really slow especially x is near x*.

Appendix A

1. main

```
import numpy as np
from NewtonMethod import NewtonMethod
from QuasiNewtonMethod import QuasiNewtonMethod
from SteDesMet import SteeDesMethod2 as SteeDesMethod
```

```
from myutils import funcRSDP ,plot3D,funcBound,funcRSSDP,
funcConvex
from Tao import SDA

Dim = 100
x0 = np.random.rand(dim)
threshold = 1e-6
A,b = funcConvex(dim) # convex
SteeDesMethod(x0,A,b, threshold, dim)
SDA(A,b,x0, threshold)
NewtonMethod(x0,A,b, threshold, dim)
QuasiNewtonMethod(x0,A,b, threshold, dim)
```

2. myutils

```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.axisartist as axisartist
from scipy.stats import ortho_group

def func(dim):
    i = 1
    while True:
        np.random.seed(i)
         i =i+1
         #G = np.random.randint( 0,dim,size =[dim, dim])

         G = np.random.randn(dim, dim)
         A = G.T @ G
         e, v = np.linalg.eig(A)
         if e.all() > 0:
             print("A is RSDP")
             condA = max(e) / min(e)
             print("Conditional number :", condA)
             b = np.random.randint(0, 10, size=[dim])
             # print("A is: ", A)
             # print("**********************************
")
             print("eigenvalues: ", e)
             return A,b
def funcRSDP(dim):

    np.random.seed(0)
    A = np.eye(dim)
```

```
28.     for i in np.arange(0,dim):
29.         A[i,:] = 1
30.         A[i][i] =i + 1
31.     m = 1
32.     A = A+m*np.eye(dim)
33.     b = np.random.randint(0, dim, size=[dim])
34.     e, v = np.linalg.eig(A)
35.     print(A)
36.     print("################################")
37.     print(e)
38.     condA = max(e) / min(e)
39.     print("Conditional number : ",    condA)
40.     return A,b
41. 
42. def funcRSSDP(dim):
43.     print("A is real-symmetric-semi-define-positive")
44.     np.random.seed(0)
45.     A = np.eye(dim)
46.     for i in np.arange(0, dim):
47.         A[i, :] = 1
48.         A[i][i] = i
49.     # A[i,:] = (np.arange(0,dim))%10 + 1
50.     # A[:, i] =  A[i,:]
51.     b = np.random.randint(0, 10, size=[dim])
52.     e, v = np.linalg.eig(A)
53.     print(A)
54.     print("################################")
55.     print(e)
56.     #condA = max(e) / min(e)
57.     #print("Conditional number : ", condA)
58.     return A, b
59. 
60. def funcConvex(dim):
61.     eigvals = 10 * np.abs(np.random.random((dim)))
62.     A = np.eye(dim)
63.     for i in range(dim):
64.         A[i][i] = eigvals[i]
65.     seed = 1
66.     U = np.float32(ortho_group.rvs(dim=dim, random_state
    =seed))
67.     b = np.random.rand(dim)
68.     A = U.transpose().dot(A).dot(U)
69.     e, v = np.linalg.eig(A)
70.     print(A)
```

```
71.     print("################################")
72.     if min(e) != 0:
73.         condA = max(e) / min(e)
74.         print("Conditional number : ", condA)
75.     return A,b
76. def funcBound(dim):
77.     print("A is real-symmetric-semi-define-positive")
78.     np.random.seed(0)
79.     A = np.eye(dim)
80.     for i in np.arange(0, dim):
81.
82.         A[i][i] = (i) % 10+1
83.     A[i,:] = (np.arange(0,dim))%10 + 1
84.     # A[:, i] =  A[i,:]
85.     b = np.random.randint(0, 10, size=[dim])
86.     e, v = np.linalg.eig(A)
87.     print(A)
88.     print("################################")
89.     print(e)
90.     condA = max(e) / min(e)
91.     print("Conditional number : ", condA)
92.     return A, b
93. def create_A_b(dimension,matrix_type):
94.     eigvals = 10*np.abs(np.random.random((dimension)))
95.     A = np.eye(dimension)
96.     m = abs(np.random.randint(10))
97.     print(m)
98.     for i in range(dimension):
99.         A[i][i] = eigvals[i]
100.        seed = 1
101.        U= np.float32(ortho_group.rvs(dim=dimension, rand
    om_state=seed))
102.        b = np.random.rand(dimension, 1)
103.        A = U.transpose().dot(A).dot(U)
104.        if matrix_type == "convex":
105.            return A,b
106.        if matrix_type == "consistently convex":
107.            return A+m*np.eye(dimension),b,m
108.        if matrix_type == "bounded convex":
109.            return
110.
111.    def PlotWithAxis(data_x,data_y):
112.
```

```
113.        # plot
114.        fig = plt.figure()
115.        ax = axisartist.Subplot(fig, 111)
116.        fig.add_axes(ax)
117.        ax.axis["bottom"].set_axisline_style("-
    |>", size=1.5)
118.        ax.axis["left"].set_axisline_style("->", size=1.5
    )
119.        ax.axis["top"].set_visible(False)
120.        ax.axis["right"].set_visible(False)
121.
122.        plt.title(r'$Gradient \ method - steepest \ desce
    nt \ method$')
123.        plt.plot(data_x, data_y, label=r'$f(x_1,x_2)=x_1^
    2+2 \cdot x_2^2-2 \cdot x_1 \cdot x_2-2 \cdot x_2$')
124.        #plt.grid(True, color="r")
125.        plt.legend()
126.        plt.scatter(1, 1,marker=(5,1), c="r", s=1000)
127.        #plt.grid(True,color='r', linestyle ='-
    ', linewidth = 2)
128.        plt.xlabel(r'$x_1$', fontsize=20)
129.        plt.ylabel(r'$x_2$', fontsize=20)
130.        plt.show()
131.
132.    def PlotSub(r,func,dim,threshold):#residual error; fu
    nction value, dimension, threshold
133.        plt.figure()
134.        X = np.arange(0, len(r))
135.        length = len(X)
136.        # fig1
137.        ax1 = plt.subplot(3, 1, 1)
138.        if len(X) < 50:
139.            plt.plot(X, r, 'g*-')
140.        else:
141.            plt.plot(X, r, 'g-')
142.        ax1.set_title(" Error value   " + "(dim = " +str(
    dim) +", " +"threshold= " +str(threshold) +")")
143.        ax1.set_xlabel('iteration')
144.        ax1.set_ylabel('grad value')
145.        #plt.xticks(range(length + 1))
146.
147.        # fig2
148.        ax2 = plt.subplot(3, 1, 3)
```

```
149.        if len(X) < 50:
150.            plt.plot(X, func, 'bo-')
151.        else:
152.            plt.plot(X, func, 'b-')
153.        ax2.set_title("The objection function value   " +
    "(dim = " +str(dim)+", " +"threshold = " +str(threshold)
    +")")
154.        ax2.set_xlabel('iteration')
155.        ax2.set_ylabel('fun value')
156.        #plt.xticks(range(length + 1))
157.
158.        plt.show()
159.
160.    from mpl_toolkits.mplot3d import Axes3D
161.    def plot3D(A,b):
162.        fig = plt.figure()
163.        ax = Axes3D(fig)
164.        x = np.arange(-2 * np.pi, 2 * np.pi, 0.1)
165.        y = np.arange(-2 * np.pi, 2 * np.pi, 0.1)
166.        X, Y = np.meshgrid(x, y)
167.
168.        Z = A[0,0]*X**2 + (A[0,1]+A[1,0])*X*Y + A[1,1]*Y*
   *2 + b[0]*X +b[1]*Y
169.
170.        Z = np.array(Z)
171.        plt.xlabel('x')
172.        plt.ylabel('y')
173.        ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cm
   ap='rainbow')
174.        plt.show()
```