Practical Optimization Method: Homework 1

钟琦 3210103612

Question 1

由矩阵范数定义可知，

$$\begin{aligned}\|B^{-1}\| &= \max_{x\neq 0}\frac{\|B^{-1}x\|}{\|x\|}\\ &= \max_{y\neq 0}\frac{\|y\|}{\|By\|}\\ &= \frac{1}{\min_{y\neq 0}\frac{\|By\|}{\|y\|}}\\ &\geq \frac{1}{\frac{\|Bx\|}{\|x\|}} = \frac{\|x\|}{\|Bx\|}\end{aligned}$$

$$\therefore \|Bx\| \geq \|x\|/\|B^{-1}\|$$

Question 2

(a)

$$\begin{aligned}(A^{-1}-\frac{A^{-1}ab^TA^{-1}}{1+b^TA^{-1}a})\overline{A} &= (A^{-1}-\frac{A^{-1}ab^TA^{-1}}{1+b^TA^{-1}a})(A+ab^T)\\ &= E-\frac{A^{-1}ab^T}{1+b^TA^{-1}a}+A^{-1}ab^T-\frac{A^{-1}ab^TA^{-1}ab^T}{1+b^TA^{-1}a}\\ &= E-\frac{1}{1+b^TA^{-1}a}[A^{-1}ab^T-A^{-1}ab^T(1+b^TA^{-1}a+A^{-1}ab^TA^{-1}ab^T]\\ &= E-\frac{1}{1+b^TA^{-1}a}(A^{-1}ab^TA^{-1}ab^T-A^{-1}ab^Tb^TA^{-1}a)\end{aligned}$$

$\because b^TA^{-1}a$ 是一个数

$$\therefore A^{-1}a(b^TA^{-1}a)b^T-A^{-1}ab^T(b^TA^{-1}a=(b^TA^{-1}a)A^{-1}ab^T-(b^TA^{-1}a)A^{-1}ab^T=0$$

$$\therefore (A^{-1}-\frac{A^{-1}ab^TA^{-1}}{a+b^TA^{-1}a})\overline{A}=E$$

$$\therefore \overline{A}^{-1}=A^{-1}-\frac{A^{-1}ab^TA^{-1}}{1+b^TA^{-1}a}$$

(b)

令 $a = s_k - H_k y_k$, $b = \frac{s_k - H_k y_k}{(s_k - H_k y_k)^T y_k}$，则 $H_{k+1} = H_k + ab^T$

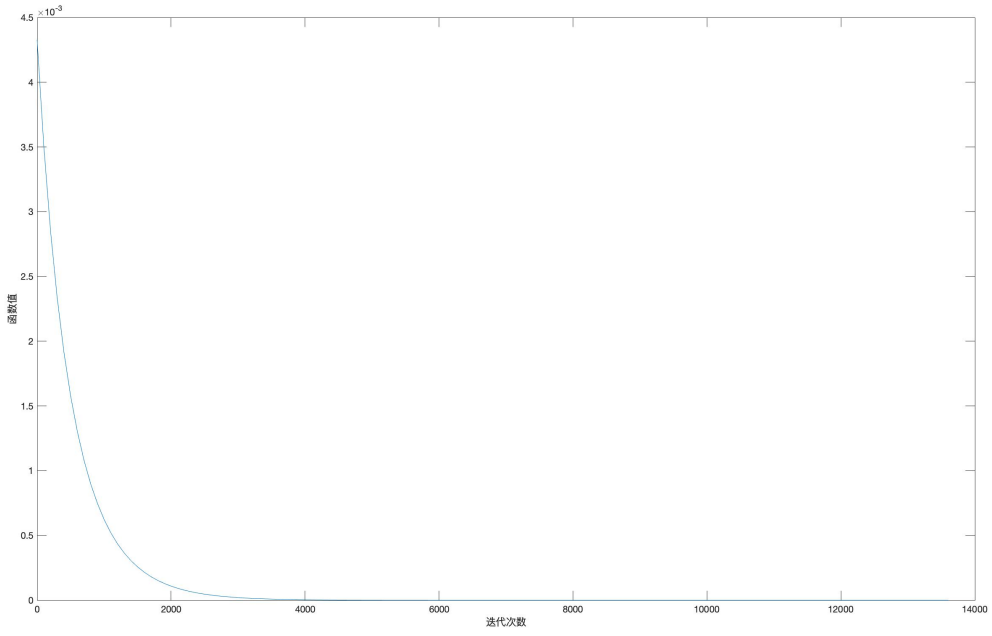
$$
\begin{aligned}
\therefore H_{k+1}^{-1} &= H_k^{-1} - \frac{H_k^{-1}(s_k - H_k y_k)\frac{(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k} H_k^{-1}}{1 + \frac{(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k} H_k^{-1}(s_k - H_k y_k)} \\
&= B_k - \frac{B_k(s_k - H_k y_k)(s_k - H_k y_k)^T B_k^T}{(s_k - H_k y_k)^T y_k + (s_k - H_k y_k)^T B_k (s_k - H_k y_k)} \\
&= B_k - \frac{(B_k s_k - y_k)(B_k s_k - y_k)^T}{(s_k - H_k y_k)^T y_k + (s_k - H_k y_k)^T (B_k s_k - y_k)} \\
&= B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{-(s_k - H_k y_k)^T B_k^T s_k} \\
&= B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \\
&= B_{k+1}
\end{aligned}
$$

Question 3

3.1 Rosenbrock Function

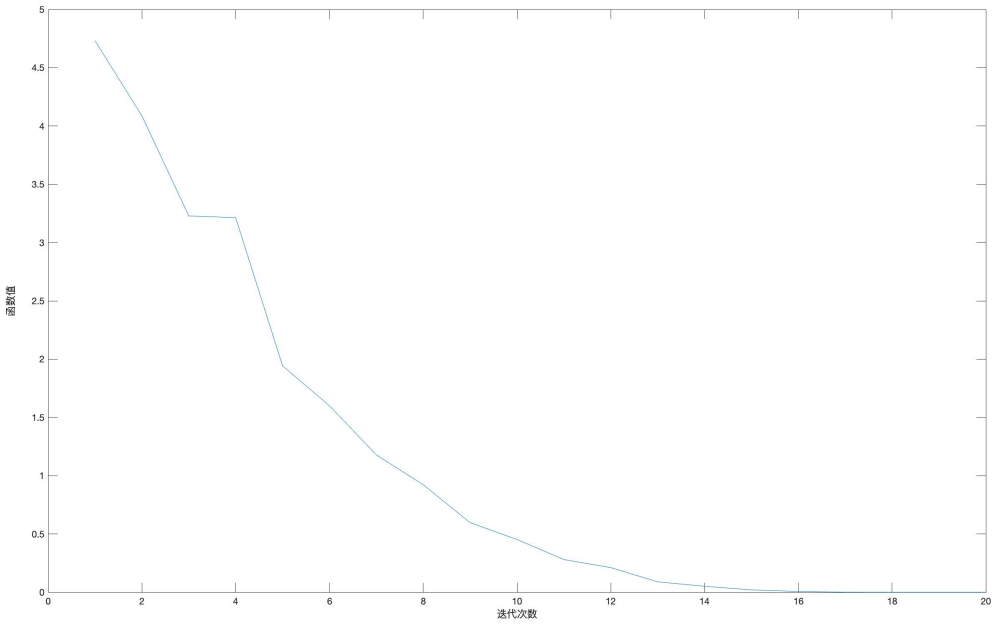
3.1.1 Steepest Descent Method

利用 Chapter 3 课件 P4 中列出的最速下降法，编写 C++ 代码（见附录），设置 $\epsilon = 10^{-6}$，对 α 的线搜索中满足 sufficient decrease condition 的 $c = 10^{-4}$，得到经过 13755 次迭代后在 $x = (0.9999992191, 0.9999984335)^T$ 处取得最小值 0.0000000000。函数值随迭代次数变化曲线如下图所示，由图可知迭代次数超过 4000 次后函数值已趋于稳定。



3.1.2 Newtons Method

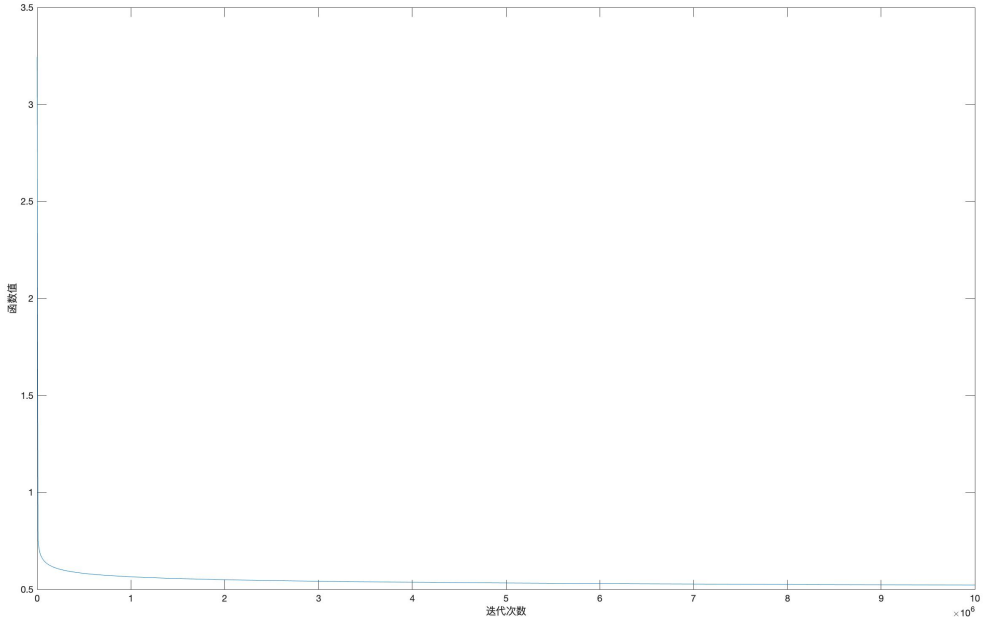
利用 Chapter 3 课件 P17 中列出的带步长因子的牛顿法，对 ϵ 和 c 进行相同设置，得到经过 20 次迭代后在 $x = (0.9999999999, 0.9999999999)^T$ 处取得最小值 0.0000000000。此时，Hessian 矩阵为 $\begin{pmatrix} 801.9999999046 & -399.9999999760 \\ -399.9999999760 & 200.0000000000 \end{pmatrix}$，是正定矩阵。由下图可知，下降速度非常快。



3.2 Beale Function

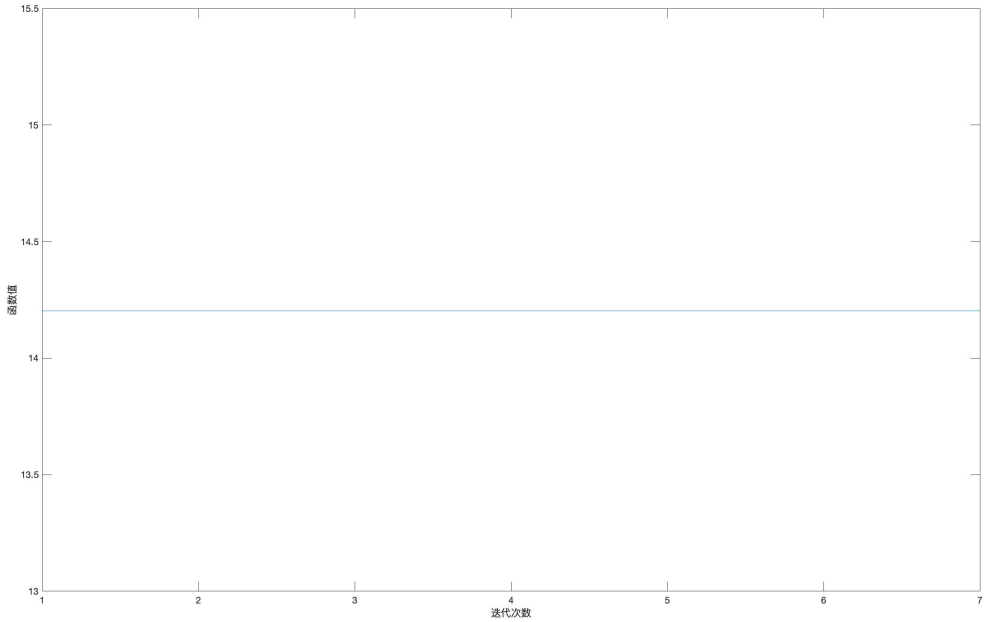
3.2.1 Steepest Descent Method

更换函数，其余操作步骤均不变，得到经过 10000000 次迭代后仍然未找到最小值点，循环停止，此时 $x = (-21.3656512618, x2 = 1.0445675837)^T$，取到函数值 $f(x) = 0.5221702403$。由下图可知，经过 10000000 次迭代后仍然有下降趋势。



3.2.2 Newtons Method

由 Newton 法，经过 7 次迭代后显示在 $x = (-0.0000000033, 1.0000000000)^T$ 处取得最小值 14.2031250000，此时 Hession 矩阵为 $\begin{pmatrix} 0.0000000000 & 25.5750000000 \\ 25.5750000000 & -0.0000001187 \end{pmatrix}$，不是正定矩阵，所以实际上未取到最小值，需要用到修正 Newton 法。



3.3 总结

由上述两个函数测试可知，Newton 法普遍收敛速度较最速下降法快，然而，Newton 法只适用于部分函数，应用时需要检查其 Hessian 矩阵是否正定，最速下降法理论上在有限步内总能找到最小值，数值上可靠性较高，但是总体下降速度较慢，对参数选择和算力要求较高。

Question 4

(1) $x^{(0)}$ =(cos 70°,sin 70°,cos 70°,sin 70°)

σ	Newton 算法	迭代次数	最小值点	函数值
1	pure	10	(0.00000001,-0.00000006, 0.00000016, -0.00000001)	0.00000
10^4	pure	46	(0.00000000,-0.00000000, 0.00000000, -0.00000000)	0.00000
1	with line search	11	(0.00000007, 0.00000000, 0.00000000, 0.00000010)	0.00000
10^4	with line search	19	(0.00000000, -0.00000000, -0.00000000, 0.00000000)	0.00000

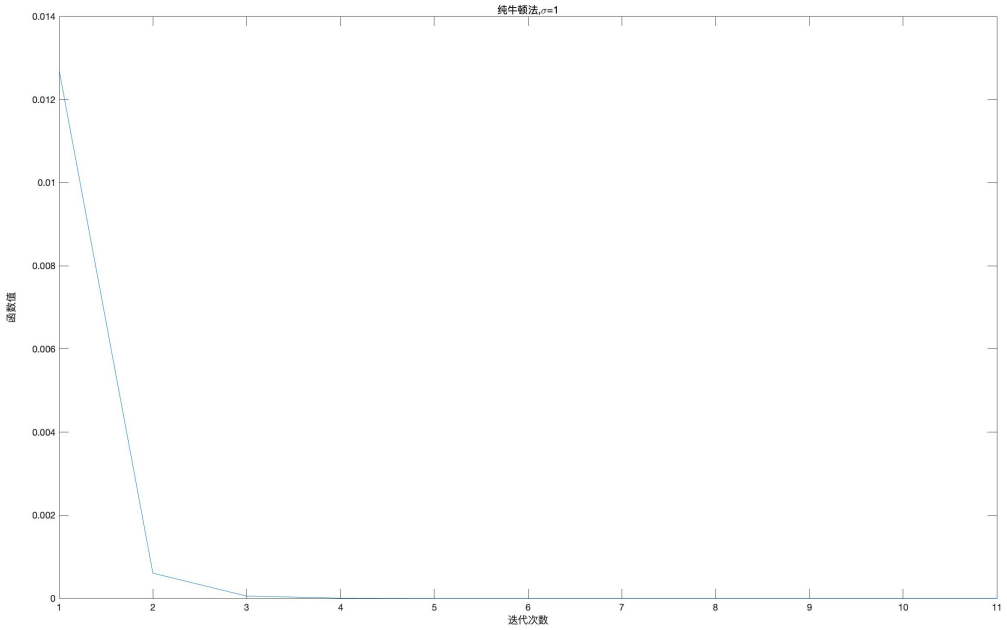
(2) $x^{(0)}$ =(cos 50°,sin 50°,cos 50°,sin 50°)

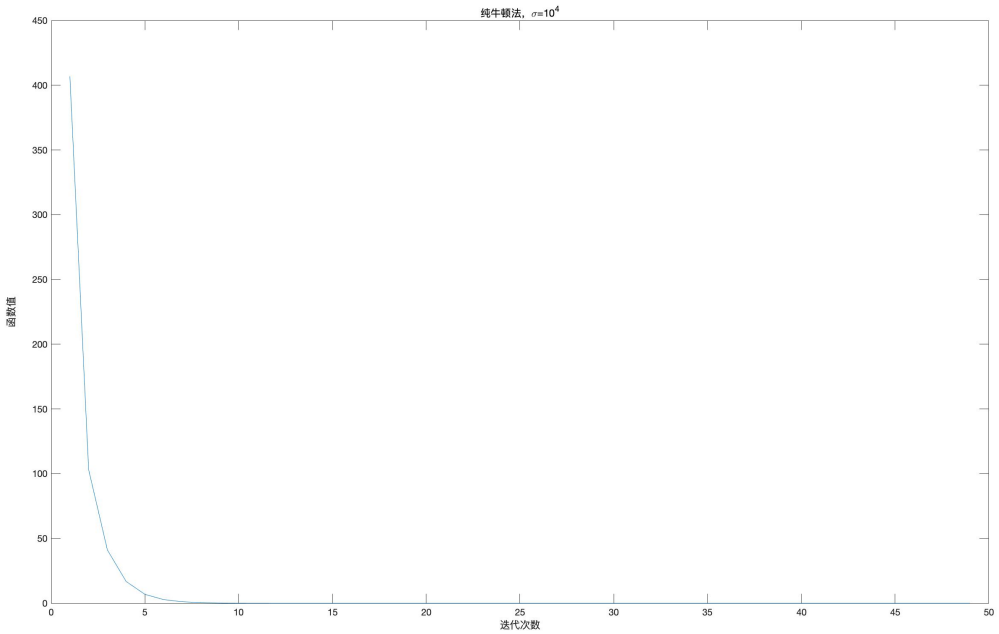
σ	Newton 算法	迭代次数	最小值点	函数值
1	pure	11	(0.00000001, -0.00000007, 0.00000020, -0.00000001)	0.00000
10^4	pure	49	(0.00000000,-0.00000000, 0.00000000, -0.00000000)	0.00000
1	with line search	11	(0.00000009, -0.00000000, 0.00000000, 0.00000011)	0.00000
10^4	with line search	20	(0.00000000,-0.00000000,-0.00000000,0.00000000)	0.00000

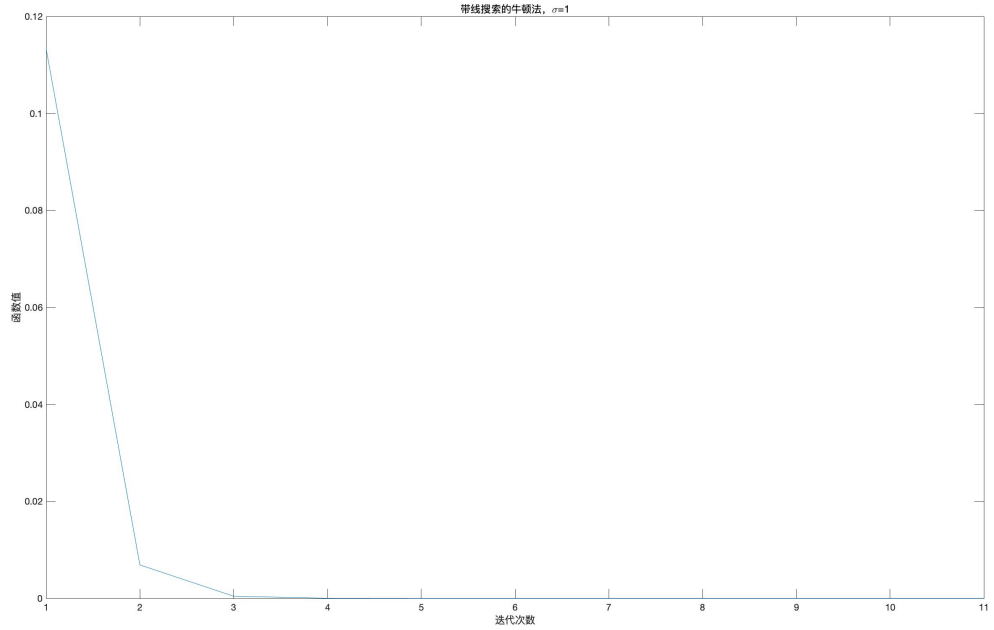
有上表可知，不同的初始点对纯牛顿法和带线搜索的牛顿法迭代速率影响不大，σ 的大小对这两种牛顿法的影响较大。

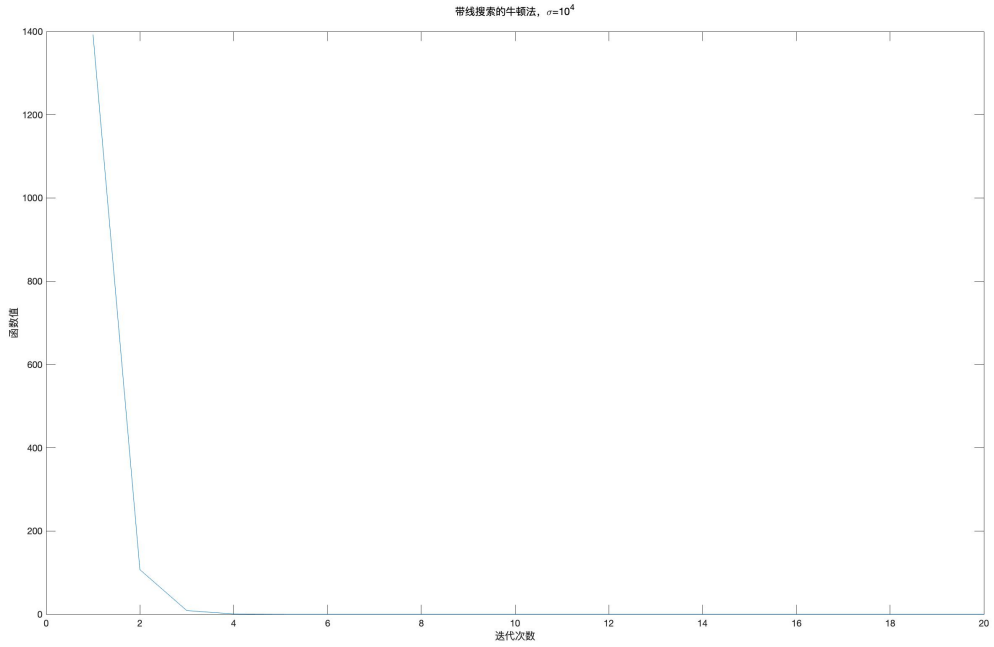
数值结果方面，两种牛顿法在有限步内都能得到相似的最小值点，没有显著差异；**收敛速度方面**，在 σ 较小时，两种牛顿法收敛速度没有显著差异，在 σ 较大时，带线搜索的牛顿法收敛速度显著高于纯牛顿法收敛速度。

以 **(2)** 中初始值为例，下面四张图分别是纯牛顿法在 $\sigma = 1$ 和 $\sigma = 10^4$、带线搜索的牛顿法在 $\sigma = 1$ 和 $\sigma = 10^4$ 处的收敛情况。


纯牛顿法,σ=1
函数值
迭代次数
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
1
2
3
4
5
6
7
8
9
10
11


纯牛顿法，σ=10^4
函数值
迭代次数
450
400
350
300
250
200
150
100
50
0
0
5
10
15
20
25
30
35
40
45
50



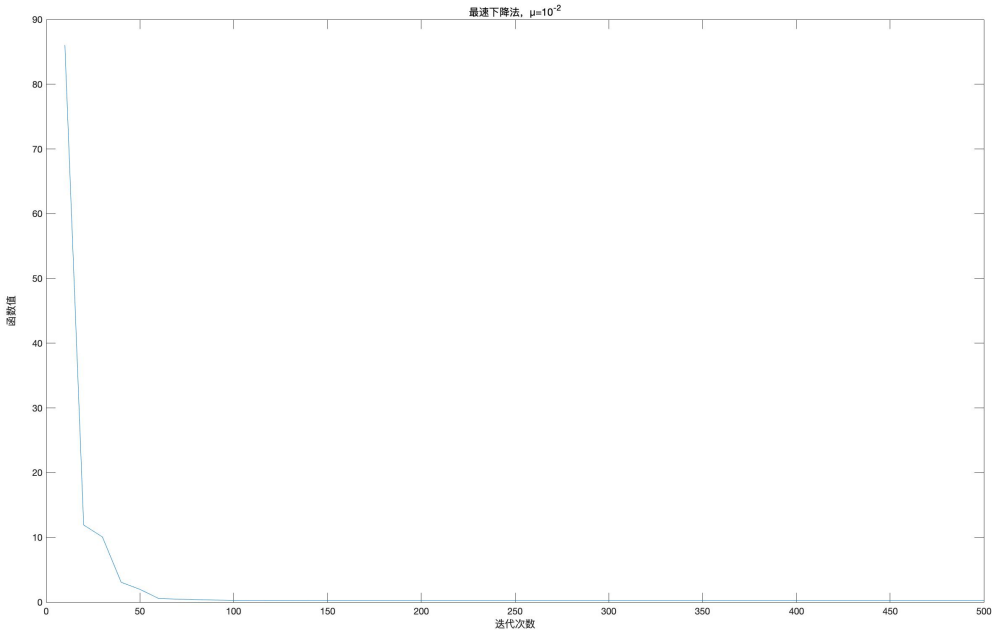


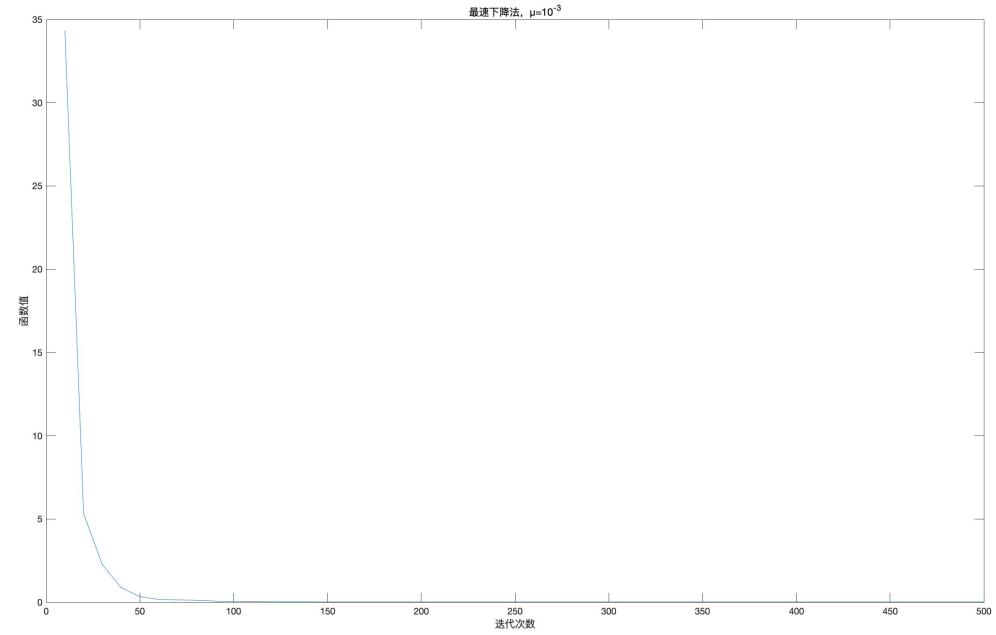
另外，当带线搜索的牛顿法选择参数与 **Question 3** 中相同时，不论 σ 的大小，其收敛速度与纯牛顿法相似。而当选择参数 $\alpha_0 = 0.8$，$AlphaRate = 0.618$ 时，当 $\sigma = 10^4$ 时收敛速度有显著提升（即上表中数据）。由此可知，参数的选择在优化进程中起着重要的作用。

Question 5

(1) 最速下降法

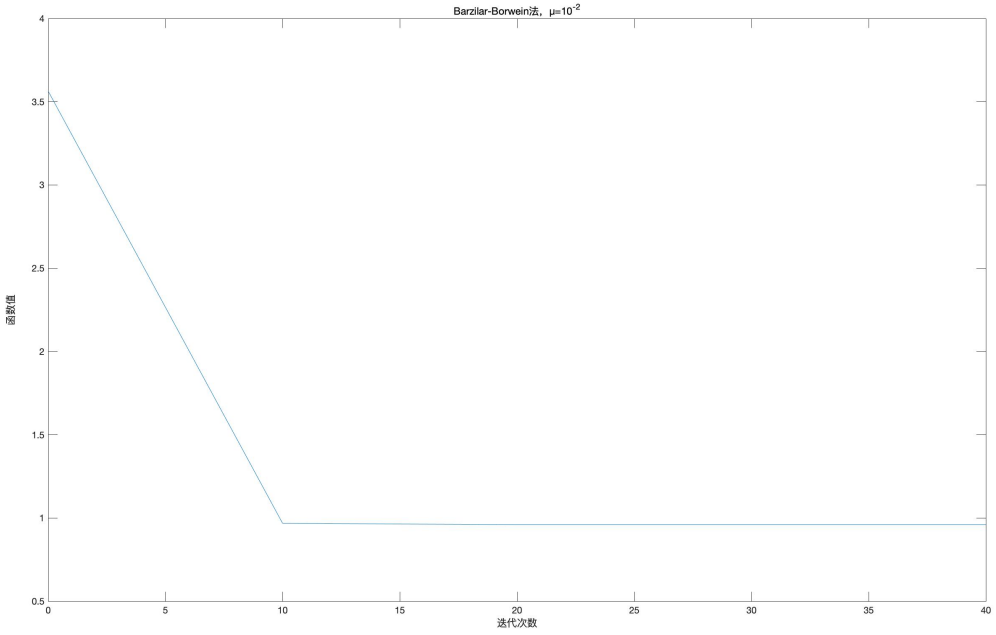
使用最速下降法对该问题进行求解，随机产生矩阵 A，向量 b，以及初始稀疏向量 $x^{(0)}$。运行结果如下（由于最速下降法运行速度较慢，限制最大迭代次数 500 次），平均迭代次数均超过 500 次，但由图中能发现迭代次数超过 100 后函数值已趋于稳定。

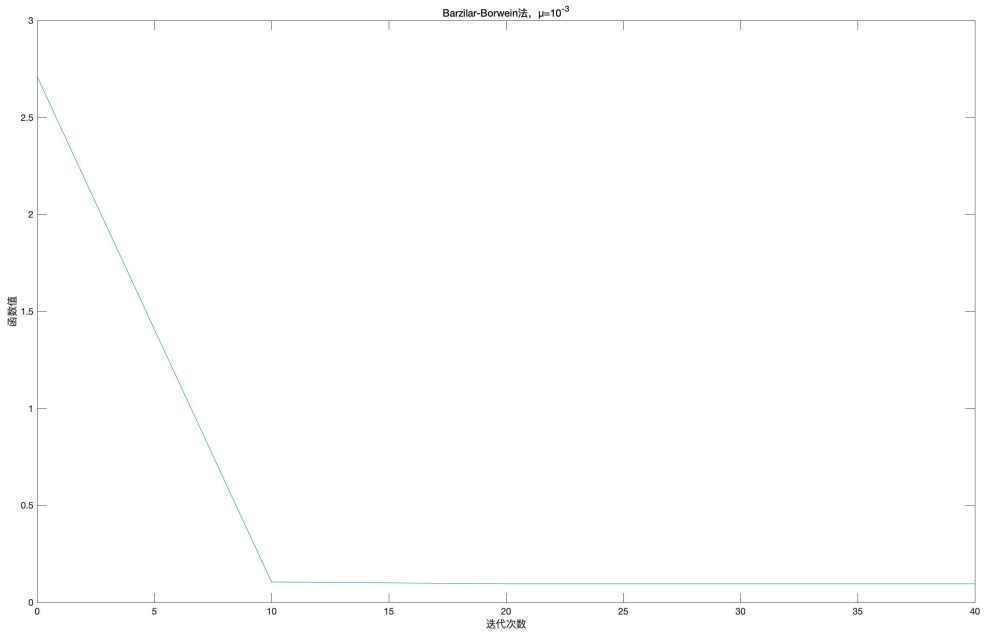




(2)Barzilar-Borwein 法

运行结果如下，$\mu = 10^{-2}$ 和 $\mu = 10^{-3}$ 时平均迭代次数均为 42 次。





(3) 总结

对于 LASSO 问题, 最速下降法和 Barzilar-Borwein 法都能得到理想的优化值, 但采用 Barzilar-Borwein 法可以有更快的收敛速度，求解性能更好。

A 附录

A.1 Question 3 最速下降法代码

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;

double f(double x1,double x2){
        double a=x2-x1*x1,b=1-x1,ans;
        ans=100*a*a+b*b;
        return ans;
}

double g1(double x1,double x2){
        double a=x2-x1*x1,b=1-x1;
        double ans=-400*a*x1-2*b;
        return ans;
}

double g2(double x1,double x2){
        double ans=200*(x2-x1*x1);
        return ans;
}

double norm(double a,double b){
        double ans=sqrt(a*a+b*b);
        return ans;
}

double LineSearch(double x1,double x2,double p1,double p2){
        double alpha=1,c=1e-4;
        double y1=x1-alpha*p1,y2=x2-alpha*p2;
        while(f(y1,y2)>f(x1,x2)-c*alpha*(p1*p1+p2*p2)){
                alpha*=0.5;
                y1 = x1 - alpha * p1;
                y2 = x2 - alpha * p2;
        }
        return alpha;
}

```

```
39 void SteepestDescent(double x1,double x2,double e){
40         int iter=0;
41         while(iter<=100000){
42                 double p1,p2;
43                 if(iter==0){
44                         p1=g1(x1,x2);
45                         p2=g2(x1,x2);
46                 }
47                 double alpha=LineSearch(x1,x2,p1,p2);
48                 x1=x1-alpha*p1;
49                 x2=x2-alpha*p2;
50                 p1=g1(x1,x2);
51                 p2=g2(x1,x2);
52                 if(norm(p1,p2)<e){
53                         break;
54                 }
55                 iter++;
56         }
57         cout<<"The number of iterations is "<<iter<<endl;
58         cout <<"Minimum found at x1 = "<<fixed<<setprecision(10)<<x1 << ", x2
       = " << x2  << endl;
59         cout<<"The value of the function is "<<f(x1,x2)<<endl;
60 }
61
62 int main(){
63         double e=1e-6;
64         double x1=-1.2,x2=1;
65         SteepestDescent(x1,x2,e);
66         return 0;
67 }
```

A.2 Question 3 Newton 法代码

```
1 #include <iostream>
2 #include <cmath>
3 #include <Eigen/Dense>
4 #include <iomanip>
5
6 using namespace std;
7 using namespace Eigen;
8
9 double f(Vector2d x){
```

```
10          double a=x(1)-x(0)*x(0),b=1-x(0),ans;
11          ans=100*pow(a,2)+pow(b,2);
12          return ans;
13  }
14
15  Vector2d grad(Vector2d x){
16          Vector2d ans;
17          ans(0)=-400*(x(1)-x(0)*x(0))*x(0)-2*(1-x(0));
18          ans(1)=200*(x(1)-x(0)*x(0));
19          return ans;
20  }
21
22
23  Matrix2d hessian(Vector2d x){
24          Matrix2d hess;
25          hess(0,0)=-400*(x(1)-3*x(0)*x(0))+2;
26          hess(0,1)=-400*x(0);
27          hess(1,0)=-400*x(0);
28          hess(1,1)=200;
29          return hess;
30  }
31
32  double LineSearch(Vector2d x,Vector2d p){
33          double alpha=1,c=1e-4;
34          Vector2d y=x+alpha*p;
35          while(f(y)>f(x)+c*alpha*(grad(x)(0)*p(0)+grad(x)(1)*p(1))){
36                  alpha*=0.5;
37                  y=x+alpha*p;
38          }
39          return alpha;
40  }
41
42  void Newton(Vector2d x,double e){
43          int iter=0;
44          while(true){
45                  Vector2d p=-hessian(x).inverse()*grad(x);
46                  double alpha=LineSearch(x,p);
47                  x+=alpha*p;
48                  if(grad(x).norm()<e){
49                          break;
50                  }
51                  iter++;
52          }
```

```
53          cout<<"The number of iterations is "<<iter<<endl;
54          cout <<"Minimum found at x1 = "<<fixed<<setprecision(10)<<x(0) << ",
        x2 = " << x(1)  << endl;
55          cout<<"The value of the function is "<<f(x)<<endl;
56          cout<<"hess:"<<endl;
57                  cout<<hessian(x)(0,0)<<"  "<<hessian(x)(0,1)<<endl;
58                  cout<<hessian(x)(1,0)<<"  "<<hessian(x)(1,1)<<endl;
59                  cout<<endl;
60  }
61
62  int main(){
63          double e=1e-6;
64          Vector2d x(-1.2,1);
65          Newton(x,e);
66          return 0;
67  }
```

A.3 Question 4 纯 Newton 法代码

```
1   #include <iostream>
2   #include <cmath>
3   #include <Eigen/Dense>
4   #include <iomanip>
5   #define theta 1.0
6   #define PI 3.1415926535
7
8   using namespace std;
9   using namespace Eigen;
10
11  double f(Vector4d x){
12          double x1=x(0),x2=x(1),x3=x(2),x4=x(3),ans;
13          double a=1/2*(x1*x1+x2*x2+x3*x3+x4*x4);
14          double b=x1*(5*x1+x2+1/2*x4)+(x1+4*x2+1/2*x3)*x2+(1/2*x2+3*x3)*x3
        +(1/2*x1+2*x4)*x4;
15          ans=a+theta/4*b;
16          return ans;
17  }
18
19  Vector4d grad(Vector4d x){
20          Vector4d ans;
21          double x1=x(0),x2=x(1),x3=x(2),x4=x(3);
22          ans(0)=x1+theta/4*(5*x1+x2+1/2*x4+5*x1+x2+1/2*x4);
```

```
23          ans(1)=x2+theta/4*(x1+4*x2+x1+4*x2+1/2*x3+1/2*x3);
24          ans(2)=x3+theta/4*(1/2*x2+3*x3+1/2*x2+3*x3);
25          ans(3)=x4+theta/4*(1/2*x1+2*x4+1/2*x1+2*x4);
26          return ans;
27  }
28
29  Matrix4d hessian(Vector4d x){
30          double x1=x(0),x2=x(1),x3=x(2),x4=x(3);
31          Matrix4d hess;
32          hess(0,0)=1+theta/2*5;
33          hess(0,1)=theta/2;
34          hess(0,2)=0;
35          hess(0,3)=theta/4;
36          hess(1,0)=theta/2;
37          hess(1,1)=1+2*theta;
38          hess(1,2)=theta/4;
39          hess(1,3)=0;
40          hess(2,0)=0;
41          hess(2,1)=theta/4;
42          hess(2,2)=1+3/2*theta;
43          hess(2,3)=0;
44          hess(3,0)=theta/4;
45          hess(3,1)=0;
46          hess(3,2)=0;
47          hess(3,3)=1+theta;
48          return hess;
49  }
50
51
52  void Newton(Vector4d x,double e){
53          int iter=0;
54          while(iter<=100000){
55                  Vector4d p=-hessian(x).inverse()*grad(x);
56                  double alpha=1;
57                  x+=alpha*p;
58                  if(grad(x).norm()<e){
59                          break;
60                  }
61                  iter++;
62          }
63          cout<<"The number of iterations is "<<iter<<endl;
64          cout <<"Minimum found at x1 = "<<fixed<<setprecision(8)<<x(0) << ", x2
        = " << x(1)  <<  ", x3 = " << x(2)  << ", x4 = " << x(3)  <<endl;
```

```
65          cout<<"The value of the function is "<<f(x)<<endl;
66          cout<<"hess:"<<endl;
67                  cout<<hessian(x)(0,0)<<"  "<<hessian(x)(0,1)<<"  "<<hessian(x)
    (0,2)<<"  "<<hessian(x)(0,3)<<endl;
68                  cout<<hessian(x)(1,0)<<"  "<<hessian(x)(1,1)<<"  "<<hessian(x)
    (1,2)<<"  "<<hessian(x)(1,3)<<endl;
69                  cout<<hessian(x)(2,0)<<"  "<<hessian(x)(2,1)<<"  "<<hessian(x)
    (2,2)<<"  "<<hessian(x)(2,3)<<endl;
70                  cout<<hessian(x)(3,0)<<"  "<<hessian(x)(3,1)<<"  "<<hessian(x)
    (3,2)<<"  "<<hessian(x)(3,3)<<endl;
71  }
72
73  int main(){
74          double e=1e-6;
75          double ang=50.0*PI/180.0;
76          double a=cos(ang),b=sin(ang);
77          Vector4d x(a,b,a,b);
78          Newton(x,e);
79          return 0;
80  }
```

A.4 Question 4 带线搜索的 Newton 法代码

```
1   #include <iostream>
2   #include <cmath>
3   #include <Eigen/Dense>
4   #include <iomanip>
5   #define theta 1.0
6   #define PI 3.1415926535
7
8   using namespace std;
9   using namespace Eigen;
10
11  double f(Vector4d x){
12          double x1=x(0),x2=x(1),x3=x(2),x4=x(3),ans;
13          double a=1/2*(x1*x1+x2*x2+x3*x3+x4*x4);
14          double b=x1*(5*x1+x2+1/2*x4)+(x1+4*x2+1/2*x3)*x2+(1/2*x2+3*x3)*x3
    +(1/2*x1+2*x4)*x4;
15          ans=a+theta/4*b;
16          return ans;
17  }
18
```

```
19 Vector4d grad(Vector4d x){
20         Vector4d ans;
21         double x1=x(0),x2=x(1),x3=x(2),x4=x(3);
22         ans(0)=x1+theta/4*(5*x1+x2+1/2*x4+5*x1+x2+1/2*x4);
23         ans(1)=x2+theta/4*(x1+4*x2+x1+4*x2+1/2*x3+1/2*x3);
24         ans(2)=x3+theta/4*(1/2*x2+3*x3+1/2*x2+3*x3);
25         ans(3)=x4+theta/4*(1/2*x1+2*x4+1/2*x1+2*x4);
26         return ans;
27 }
28
29 Matrix4d hessian(Vector4d x){
30         double x1=x(0),x2=x(1),x3=x(2),x4=x(3);
31         Matrix4d hess;
32         hess(0,0)=1+theta/2*5;
33         hess(0,1)=theta/2;
34         hess(0,2)=0;
35         hess(0,3)=theta/4;
36         hess(1,0)=theta/2;
37         hess(1,1)=1+2*theta;
38         hess(1,2)=theta/4;
39         hess(1,3)=0;
40         hess(2,0)=0;
41         hess(2,1)=theta/4;
42         hess(2,2)=1+3/2*theta;
43         hess(2,3)=0;
44         hess(3,0)=theta/4;
45         hess(3,1)=0;
46         hess(3,2)=0;
47         hess(3,3)=1+theta;
48         return hess;
49 }
50
51 double LineSearch(Vector4d x,Vector4d p){
52         double alpha=0.8,c=1e-4;
53         Vector4d y=x+alpha*p;
54         while(f(y)>f(x)+c*alpha*(grad(x)(0)*p(0)+grad(x)(1)*p(1)+grad(x)(2)*p
       (2)+grad(x)(3)*p(3))){
55                 alpha*=0.618;
56                 y=x+alpha*p;
57         }
58         return alpha;
59 }
60
```

```
61 void Newton(Vector4d x,double e){
62         int iter=0;
63         while(iter<=100000){
64                 Vector4d p=-hessian(x).inverse()*grad(x);
65                 double alpha=LineSearch(x,p);
66                 x+=alpha*p;
67                 if(grad(x).norm()<e){
68                         break;
69                 }
70                 iter++;
71         }
72         cout<<"The number of iterations is "<<iter<<endl;
73         cout <<"Minimum found at x1 = "<<fixed<<setprecision(8)<<x(0) << ", x2
       = " << x(1)  <<  ", x3 = " << x(2)  << ", x4 = " << x(3)  <<endl;
74         cout<<"The value of the function is "<<f(x)<<endl;
75         cout<<"hess:"<<endl;
76                 cout<<hessian(x)(0,0)<<"  "<<hessian(x)(0,1)<<"  "<<hessian(x)
      (0,2)<<"  "<<hessian(x)(0,3)<<endl;
77                 cout<<hessian(x)(1,0)<<"  "<<hessian(x)(1,1)<<"  "<<hessian(x)
      (1,2)<<"  "<<hessian(x)(1,3)<<endl;
78                 cout<<hessian(x)(2,0)<<"  "<<hessian(x)(2,1)<<"  "<<hessian(x)
      (2,2)<<"  "<<hessian(x)(2,3)<<endl;
79                 cout<<hessian(x)(3,0)<<"  "<<hessian(x)(3,1)<<"  "<<hessian(x)
      (3,2)<<"  "<<hessian(x)(3,3)<<endl;
80 }
81
82 int main(){
83         double e=1e-6;
84         double ang=50.0*PI/180.0;
85         double a=cos(ang),b=sin(ang);
86         Vector4d x(a,b,a,b);
87         Newton(x,e);
88         return 0;
89 }
```

A.5 Question 5 最速下降法代码

```
1 #include <iostream>
2 #include <cmath>
3 #include <iomanip>
4 #include <Eigen/Sparse>
5 #include <Eigen/Dense>
```

```
#define miu 0.001
using namespace std;
using namespace Eigen;

double AbsoluteValue(double a){
    if(a>=0){
        return a;
    }else{
        return -1*a;
    }
}

double L(VectorXd x){
    double delta=0.001*miu;
    double ret=0;
    for(int i=0;i<1024;i++){
        double value=x(i);
        if(AbsoluteValue(value)<delta){
            ret+=1.0/(2*delta)*value*value;
        }else{
            ret+=AbsoluteValue(value)-delta/2.0;
        }
    }
    return ret;
}

double f(MatrixXd A,VectorXd x,VectorXd b){
        VectorXd c(b.size());
        c=A*x-b;
        double M=c.norm();
        double ans=M*M/2.0+miu*L(x);
        return ans;
}

VectorXd g(MatrixXd A,VectorXd x,VectorXd b){
    VectorXd ans(x.size());
    for(int i=0;i<1024;i++){
        ans(i)=0;
    }
    VectorXd Ax=A*x-b;
    for(int i=0;i<1024;i++){
        for(int j=0;j<512;j++){
            ans(i)+=A(j,i)*Ax(j);
```

```
49          }
50      }
51      return ans;
52  }
53
54  double Linesearch(MatrixXd A,VectorXd x,VectorXd b,VectorXd p){
55          double alpha=1,c=1e-4;
56          VectorXd y=x-alpha*p;
57          while(f(A,y,b)>f(A,x,b)-c*alpha*p.norm()*p.norm()){
58              alpha*=0.618;
59              y=x-alpha*p;
60          }
61          return alpha;
62  }
63
64  void SteepestDescent(MatrixXd A,VectorXd x,VectorXd b,double e){
65      int iter=0;
66      while(iter<=500){
67          VectorXd p(1024);
68          if(iter==0){
69              p=g(A,x,b);
70          }
71          double alpha=Linesearch(A,x,b,p);
72          x=x-alpha*p;
73          p=g(A,x,b);
74          if(p.norm()<e){
75              break;
76          }
77          iter++;
78          if(iter%10==0){
79              //cout<<"The number of iterations is "<<iter<<endl;
80              //cout<<"The value of the function is "<<f(A,x,b)<<endl;
81              cout<<f(A,x,b)<<endl;
82          }
83      }
84      cout<<f(A,x,b)<<endl;
85      cout<<iter<<endl;
86      //cout<<"The number of iterations is "<<iter<<endl;
87      //cout<<"The value of the function is "<<f(A,x,b)<<endl;
88  }
89
90  int main(){
91      double e=1e-6;
```

```
    int m=512,n=1024;
    MatrixXd A=MatrixXd::Random(m,n);

    SparseVector<double> y(n);
    for (int i = 0; i < n; i++) {
        if (rand() / (double)RAND_MAX < 0.1) {
            y.insert(i) = rand() / (double)RAND_MAX;
        }
    }
    VectorXd x(n);
    x=y.toDense();

    VectorXd b=VectorXd::Random(m);
    SteepestDescent(A,x,b,e);
    return 0;
}
```

A.6 Question 5 Barzilar-Borwein 法代码

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <Eigen/Sparse>
#include <Eigen/Dense>
#define miu 0.001
using namespace std;
using namespace Eigen;

double AbsoluteValue(double a){
    if(a>=0){
        return a;
    }else{
        return -1*a;
    }
}

double L(VectorXd x){
    double delta=0.001*miu;
    double ret=0;
    for(int i=0;i<1024;i++){
        double value=x(i);
        if(AbsoluteValue(value)<delta){
```

```
24              ret+=1.0/(2*delta)*value*value;
25          }else{
26              ret+=AbsoluteValue(value)-delta/2.0;
27          }
28      }
29      return ret;
30  }
31
32  double f(MatrixXd A,VectorXd x,VectorXd b){
33          VectorXd c(b.size());
34          c=A*x-b;
35          double M=c.norm();
36          double ans=M*M/2.0+miu*L(x);
37          return ans;
38  }
39
40  VectorXd g(MatrixXd A,VectorXd x,VectorXd b){
41      VectorXd ans(x.size());
42      for(int i=0;i<1024;i++){
43          ans(i)=0;
44      }
45      VectorXd Ax=A*x-b;
46      for(int i=0;i<1024;i++){
47          for(int j=0;j<512;j++){
48              ans(i)+=A(j,i)*Ax(j);
49          }
50      }
51      return ans;
52  }
53
54  double Linesearch(MatrixXd A,VectorXd x,VectorXd b,VectorXd p){
55          double alpha=1,c=1e-4;
56          VectorXd y=x-alpha*p;
57          while(f(A,y,b)>f(A,x,b)-c*alpha*p.norm()*p.norm()){
58              alpha*=0.618;
59              y=x-alpha*p;
60          }
61          return alpha;
62  }
63
64
65  void SteepestDescent(MatrixXd A,VectorXd x,VectorXd b,double e){
66      int iter=0;
```

```
67      double sum=0;
68      VectorXd Pre_x(x.size()),s(x.size()),Pre_p(x.size()),y(x.size());
69      while(iter<=1000){
70          VectorXd p(x.size());
71          if(iter==0){
72              p=g(A,x,b);
73          }
74          double alpha;
75          if(iter==0){
76              alpha=Linesearch(A,x,b,p);
77          }else{
78              p=g(A,x,b);
79              s=Pre_x-x;
80              y=Pre_p-p;
81              sum=s.dot(y);
82              alpha=sum/y.dot(y);
83          }
84          Pre_x=x;
85          Pre_p=p;
86          x=x-alpha*p;
87          if(p.norm()<e){
88              break;
89          }
90          iter++;
91          if(iter%10==0){
92              //std::cout<<"The number of iterations is "<<iter<<endl;
93              //std::cout<<"The value of the function is "<<f(A,x,b)<<endl;
94              cout<<f(A,x,b)<<endl;
95          }
96      }
97      //std::cout<<"The number of iterations is "<<iter<<endl;
98      //std::cout<<"The value of the function is "<<f(A,x,b)<<endl;
99      cout<<f(A,x,b)<<endl;
100     cout<<iter<<endl;
101 }
102
103 int main(){
104     double e=1e-3;
105     int m=512,n=1024;
106     MatrixXd A=MatrixXd::Random(m,n);
107
108     SparseVector<double> y(n);
109     for (int i = 0; i < n; i++) {
```

```
110         if (rand() / (double)RAND_MAX < 0.1) {
111             y.insert(i) = rand() / (double)RAND_MAX;
112         }
113     }
114     VectorXd x(n);
115     x=y.toDense();
116 
117     VectorXd b=VectorXd::Random(m);
118     SteepestDescent(A,x,b,e);
119     return 0;
120 }
```