

最优化第十二次作业

张晋 15091060

2017 年 12 月 4 日

5.9 首先画出 Rosenbrock 函数的图像，本题中 Armijo 线搜索的参数为 $\gamma = 0.5, \rho = 0.01$.

最后分别以梯度下降法和牛顿法迭代，并画出等高线、运动轨迹、迭代值如下：

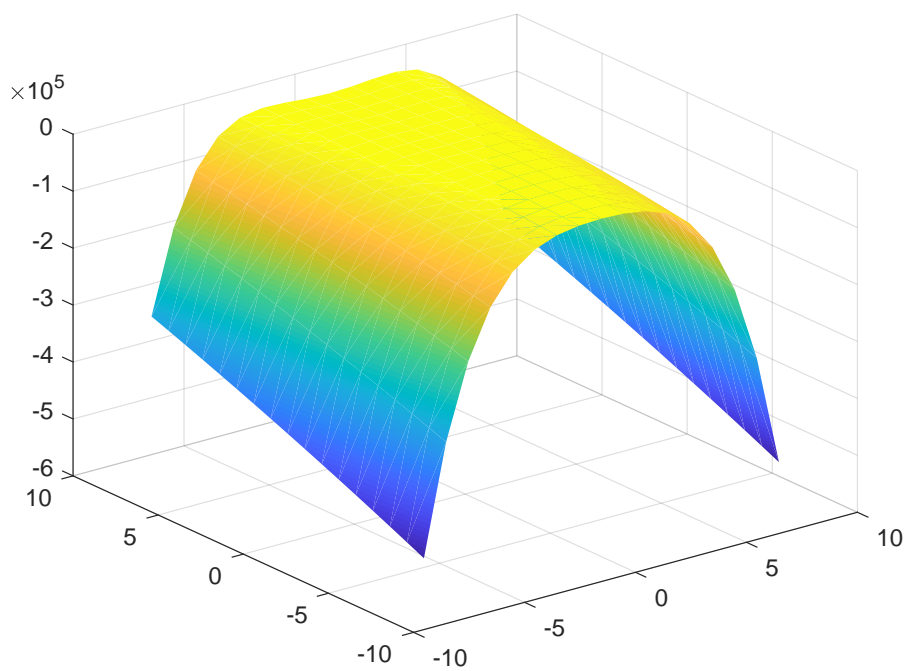


图 1: Rosenbrock 函数图像

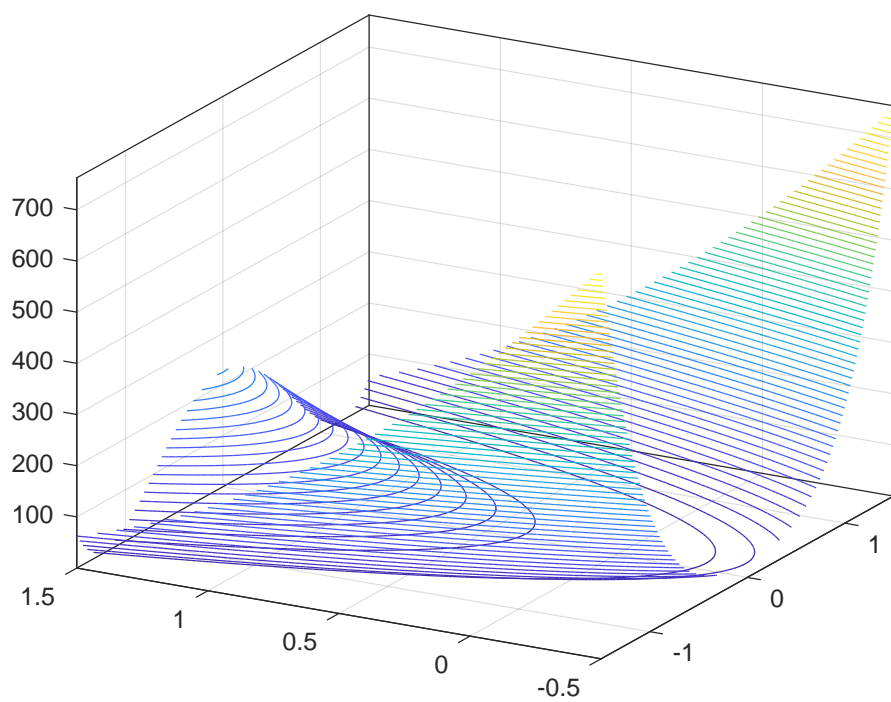


图 2: 在 $(1,1)$ 处附近的三维等高线

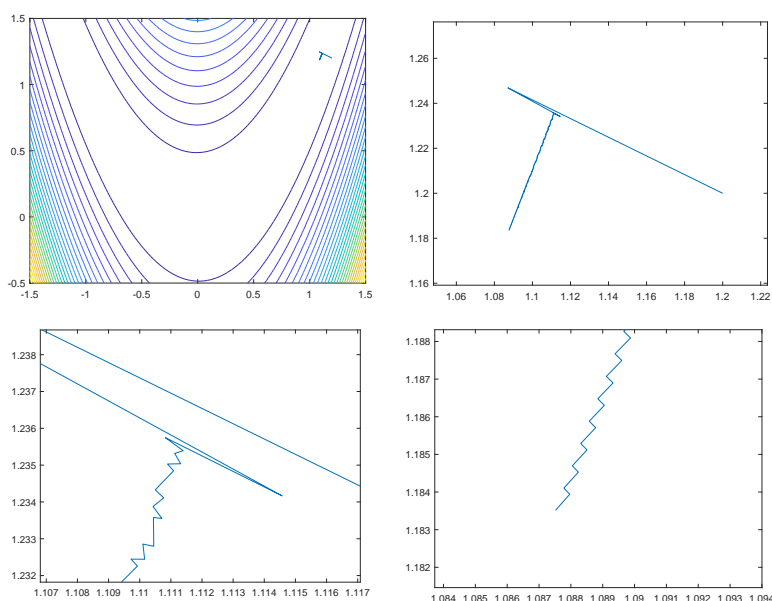


图 3: Steepest-descent in (1.2,1.2)

```

Step[190]: x=[ 1.088846 1.186477 ] optim_fx=0.007973
Step[191]: x=[ 1.089052 1.186303 ] optim_fx=0.007937
Step[192]: x=[ 1.088577 1.185882 ] optim_fx=0.007924
Step[193]: x=[ 1.088779 1.185710 ] optim_fx=0.007889
Step[194]: x=[ 1.088310 1.185288 ] optim_fx=0.007874
Step[195]: x=[ 1.088507 1.185118 ] optim_fx=0.007841
Step[196]: x=[ 1.088044 1.184696 ] optim_fx=0.007825
Step[197]: x=[ 1.088236 1.184529 ] optim_fx=0.007793
Step[198]: x=[ 1.087780 1.184104 ] optim_fx=0.007776
Step[199]: x=[ 1.087965 1.183941 ] optim_fx=0.007745
Step[200]: x=[ 1.087517 1.183515 ] optim_fx=0.007727

```

最速下降法,共迭代 200 步

结果：

```
x=[ 1.087517e+00 1.183515e+00 ] optim_fx=0.007727
```

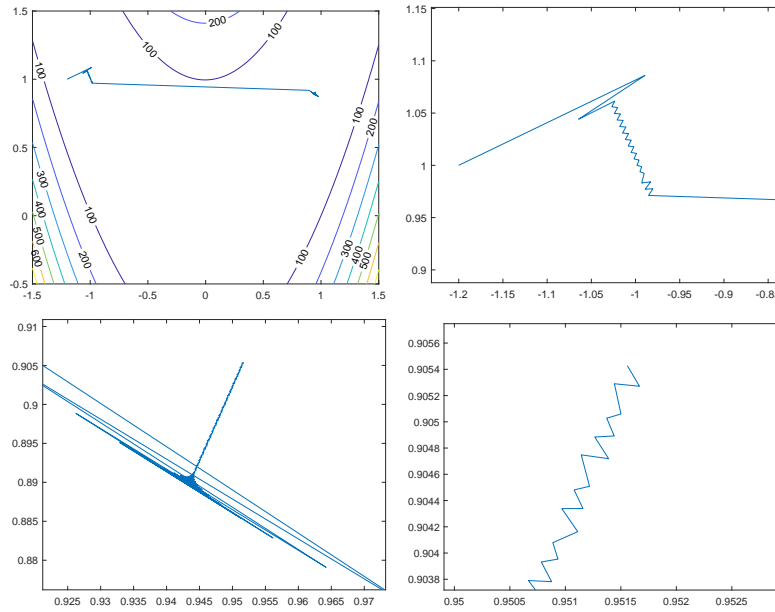


图 4: Steepest-descent in $(-1.2,1)$

Step[190]: $x=[0.951079\ 0.904481]$ $\text{optim_fx}=0.002394$
 Step[191]: $x=[0.951218\ 0.904509]$ $\text{optim_fx}=0.002389$
 Step[192]: $x=[0.951143\ 0.904748]$ $\text{optim_fx}=0.002388$
 Step[193]: $x=[0.951390\ 0.904719]$ $\text{optim_fx}=0.002381$
 Step[194]: $x=[0.951265\ 0.904884]$ $\text{optim_fx}=0.002375$
 Step[195]: $x=[0.951440\ 0.904892]$ $\text{optim_fx}=0.002370$
 Step[196]: $x=[0.951373\ 0.905027]$ $\text{optim_fx}=0.002365$
 Step[197]: $x=[0.951501\ 0.905060]$ $\text{optim_fx}=0.002361$
 Step[198]: $x=[0.951442\ 0.905290]$ $\text{optim_fx}=0.002358$
 Step[199]: $x=[0.951668\ 0.905271]$ $\text{optim_fx}=0.002352$
 Step[200]: $x=[0.951559\ 0.905427]$ $\text{optim_fx}=0.002347$

最速下降法,共迭代 200 步

结果：

$x=[9.515588e-01\ 9.054274e-01]$ $\text{optim_fx}=0.002347$

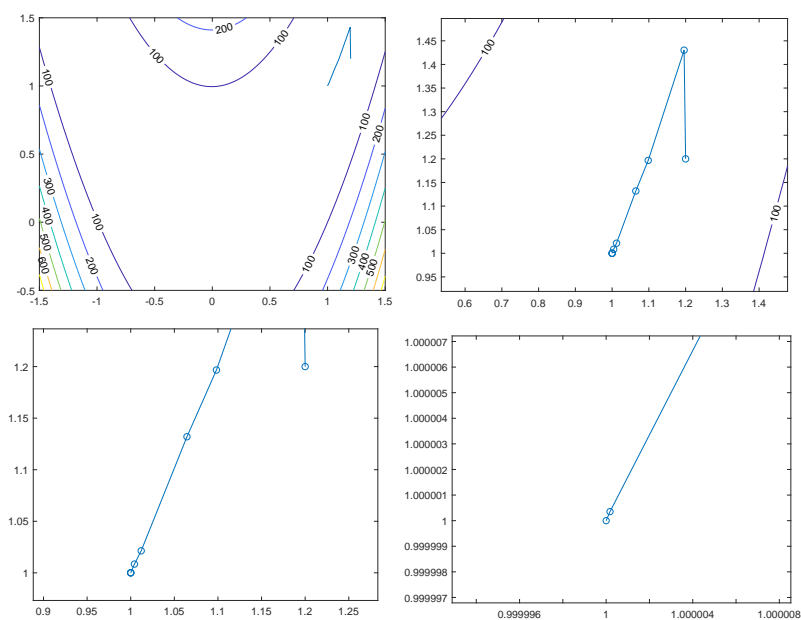


图 5: Newton-Armijo in (1.2,1.2)

```

Step[1]: x=[ 1.200000 1.200000 ] optim_fx=5.800000
Step[2]: x=[ 1.195918 1.430204 ] optim_fx=0.038384
Step[3]: x=[ 1.098284 1.196688 ] optim_fx=0.018762
Step[4]: x=[ 1.064488 1.131993 ] optim_fx=0.004289
Step[5]: x=[ 1.011992 1.021372 ] optim_fx=0.000903
Step[6]: x=[ 1.004261 1.008481 ] optim_fx=0.000019
Step[7]: x=[ 1.000050 1.000083 ] optim_fx=0.000000
Step[8]: x=[ 1.000000 1.000000 ] optim_fx=0.000000
Step[9]: x=[ 1.000000 1.000000 ] optim_fx=0.000000

```

牛顿 Armijo 回溯法,共迭代 9 步

结果 :

```
x=[ 1.000000e+00 1.000000e+00 ] optim_fx=0.000000
```

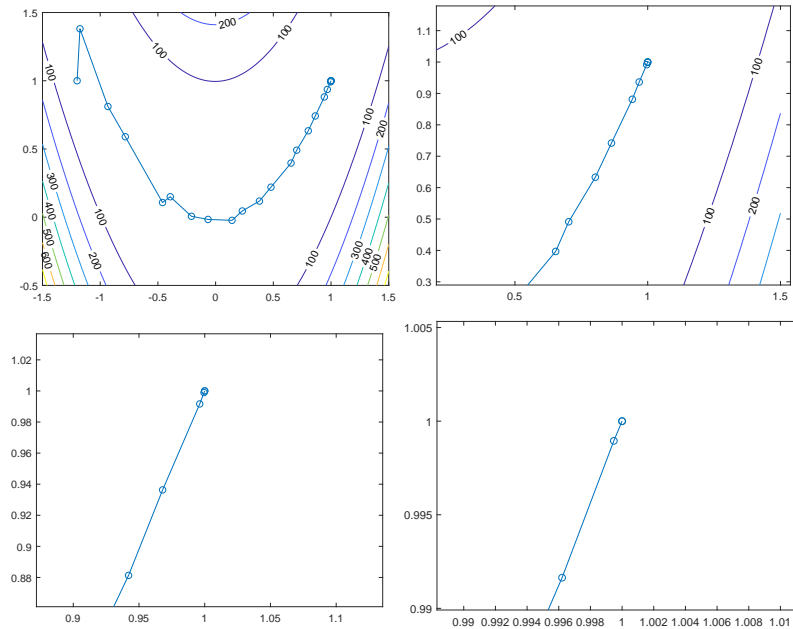


图 6: Newton-Armijo in $(-1.2, 1)$

```

Step[15]: x=[ 0.802786 0.633221 ] optim_fx=0.051535
Step[16]: x=[ 0.863491 0.741931 ] optim_fx=0.019993
Step[17]: x=[ 0.942079 0.881336 ] optim_fx=0.007169
Step[18]: x=[ 0.967992 0.936337 ] optim_fx=0.001070
Step[19]: x=[ 0.996210 0.991639 ] optim_fx=0.000078
Step[20]: x=[ 0.999479 0.998948 ] optim_fx=0.000000
Step[21]: x=[ 0.999999 0.999998 ] optim_fx=0.000000
Step[22]: x=[ 1.000000 1.000000 ] optim_fx=0.000000
牛顿 Armijo 回溯法,,共迭代 22 步
结果：
x=[ 1.000000e+00 1.000000e+00 ] optim_fx=0.000000

```

5.19 迭代次数和求解的值如下:

n=5	n=8	n=12	n=20
k=6	k=19	k=35	k=66
x	x	x	x
5.00E+00	5.90E-11	-9.61E+00	-1.10E+01
-1.20E+02	-6.97E-11	8.15E+02	1.05E+03
6.30E+02	-5.16E-10	-1.65E+04	-2.40E+04
-1.12E+03	1.12E-09	1.36E+05	2.20E+05
6.30E+02	3.17E-10	-5.36E+05	-9.65E+05
	-6.55E-10	1.03E+06	1.99E+06
	-6.57E-10	-6.43E+05	-1.25E+06
	4.59E-10	-6.58E+05	-1.34E+06
		8.04E+05	8.83E+05
		6.63E+05	1.69E+06
		-1.24E+06	3.88E+05
		4.66E+05	-1.31E+06
			-1.71E+06
			-5.28E+05
			1.21E+06
			2.00E+06
			9.45E+05
			-1.43E+06
			-2.65E+06
			1.89E+06

Step	$x^{(k)}$
1	(0, 0, 0, 0)
2	(0.9045, 0, -1.8090, -2.0225)
3	(-0.4472, -1.8944, -3.3416, -2.7889)

5.21

$$G = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}$$

$$P = \begin{bmatrix} g \\ G * g \\ G^2 * g \\ G^3 * g \end{bmatrix}^T = \begin{bmatrix} -1 & 0 & 2 & \sqrt{5} \\ -2 & -1 & 4 - \sqrt{5} & 2\sqrt{5} - 2 \\ -3 & \sqrt{5} - 4 & 11 - 4\sqrt{5} & 5\sqrt{5} - 8 \\ -\sqrt{5} - 2 & 6\sqrt{5} - 16 & 34 - 14\sqrt{5} & 14\sqrt{5} - 27 \end{bmatrix}$$

计算 $\text{MatrixRank}[P]=2$, 故可知其独立向量只有两个

附录代码

```

1  %绘制Rosenbrock函数图形
2  x=[-8:1:8];
3  y=x;
4  [X,Y]=meshgrid(x,y);
5  [row,col]=size(X);
6  for l=1:col
7      for h=1:row
8          z(h,l)=Rosenbrock([X(h,l),Y(h,l)]);
9      end
10 end
11 surf(X,Y,z);
12 shading interp
13
14 function result=Rosenbrock(x)
15 %Rosenbrock 函数
16 %输入x,给出相应的y值,在x=(1,1) 处有全局极小点0,为得到最大值,返回值取
    相反数
17 [row,col]=size(x);
18 if row>1
19     error('输入的参数错误');
20 end
21 result=100*(x(1,2)-x(1,1)^2)^2+(x(1,1)-1)^2;
22 result=-result;
23 end

```

```

1  %本实验采用最速下降法, 步长为精确步长, 测试函数为Rosenbrock函数
2  %%
3  %用符号表达式定义目标函数
4  clc;
5  clear;
6  syms x1 x2;
7  X=[x1,x2];
8  f=100*(X(2)-X(1)^2)^2+(1-X(1))^2;
9  F=eval(['@(x1,x2)',vectorize(f)]);
10 fx=diff(f,x1); %求f对x1偏导数
11 fy=diff(f,x2); %求f对x2偏导数

```

```

12 fxx=diff(fx,x1); %求二阶偏导数 对x1再对x1
13 fxy=diff(fx,x2); %求二阶偏导数 对x1再对x2
14 fyx=diff(fy,x1); %求二阶偏导数 对x2再对x1
15 fyy=diff(fy,x2); %求二阶偏导数 对x2再对x2
16 Gradient=[fx,fy]; %计算梯度表达式
17 Hesse=[fxx,fxy,fyx,fyy];
18 x=[-1.2,1]; %定义初始点
19
20 %%
21 N=200; %总迭代次数
22 e=0.000001;
23 P=zeros(N,2); %储存点的轨迹
24 OPT=zeros(N,2); %储存最优值下降的轨迹
25 g=subs(Gradient,[x1 x2],[x(1) x(2)]);
26 step=1;
27 P(step,:)=x;
28 optim_fx=subs(f,[x1 x2],[x(1) x(2)]);
29 fprintf('Step[%d]: x=[ %f %f ] optim_fx=%f\n',step,x(1),x(2),double
    (optim_fx));
30 OPT(step,:)=optim_fx;
31 %%
32 while (norm(g)>e && step < N) %当g的2-范数小于特定值时，或迭代
    次数到达上限时，停止迭代
33     step=step+1;
34     %计算目标函数点x(k)处一阶导数值
35     g=subs(Gradient,[x1 x2],[x(1) x(2)]);
36     %计算目标函数点x(k)处Hesse矩阵
37     G=subs(Hesse,[x1 x2],[x(1) x(2)]);
38     %计算目标函数点x(k)处搜索方向p
39     %p=-G\g';
40     p=-g;
41     %点x(k)处的搜索步长
42
43     ak=1;
44     % ak=Alpha(p,g,G);
45     xk=x+ak*double(p');
46     %采用Armijo法则计算近似步长ak
47     while(F(xk(1),xk(2)) > (F(x(1),x(2))+0.1*double(p'*g)*ak))
48         ak=0.5*ak;

```

```

49     xk=x+ak*double(p');
50     end
51     x=x+double(ak*p');
52     %输出结果
53     optim_fx=subs(f,[x1 x2],[x(1) x(2)]);
54     fprintf('Step[%d]: x=[ %f %f] optim_fx=%f\n',step,x(1),x(2),
        double(optim_fx));
55     P(step,:)=x;
56     OPT(step,:)=optim_fx;
57     g=subs(Gradient,[x1 x2],[x(1) x(2)]);
58 end
59 %输出结果
60 optim_fx=subs(f,[x1 x2],[x(1) x(2)]);
61 fprintf('\n最速下降法,共迭代 %d 步\n结果: \n x=[ %d %d] optim_fx
    =%f\n',step,x(1),x(2),double(optim_fx));
62 P(step+1:N,:)=[]; %删去P中的多余空间\
63 figure;
64 plot(P(:,1),P(:,2))
65 hold on;
66 xx =linspace(-1.5,1.5);
67 yy = linspace(-0.5,1.5);
68 [X,Y] = meshgrid(xx,yy);
69 Z=F(X,Y);
70 contour(X,Y,Z,'ShowText','on')
71 figure;
72 plot(OPT)
73 %%
74 % 最速下降法子程序, 计算精确步长ak
75 function a=Alpha(p,g,G)
76 a=-(p'*g)/(g'*G*g);
77 a=double(a);
78 end

```

```

1 %牛顿Armijo回溯法
2 %%
3 %用符号表达式定义目标函数
4 clc;
5 clear;

```

```

6  syms x1 x2;
7  X=[x1,x2];
8  f=100*(X(2)-X(1)^2)^2+(1-X(1))^2;
9
10 F=eval(['@(x1,x2)',vectorize(f)]);
11 fx=diff(f,x1); %求f对x1偏导数
12 fy=diff(f,x2); %求f对x2偏导数
13 fxx=diff(fx,x1); %求二阶偏导数 对x1再对x1
14 fxy=diff(fx,x2); %求二阶偏导数 对x1再对x2
15 fyx=diff(fy,x1); %求二阶偏导数 对x2再对x1
16 fyy=diff(fy,x2); %求二阶偏导数 对x2再对x2
17 Gradient=[fx,fy]; %计算梯度表达式
18 Hesse=[fxx,fxy;fyx,fyy];
19 x=[-1.2,1]; %定义初始点
20
21 %%
22 N=100; %总迭代次数
23 e=0.000001;
24 P=zeros(N,2); %储存点的轨迹
25 OPT=zeros(N,2); %储存最优值下降的轨迹
26 g=subs(Gradient,[x1 x2],[x(1) x(2)]);
27 step=1;
28 P(step,:)=x;
29 optim_fx=subs(f,[x1 x2],[x(1) x(2)]);
30 fprintf('Step[%d]: x=[ %f %f ] optim_fx=%f\n',step,x(1),x(2),double
    (optim_fx));
31 OPT(step,:)=optim_fx;
32 %%
33
34 while (norm(g)>e && step < N) %当g的2-范数小于特定值时，或迭代
    次数到达上限时，停止迭代
35     step=step+1;
36     %计算目标函数点x(k)处一阶导数值
37     g=subs(Gradient,[x1 x2],[x(1) x(2)]);
38     %计算目标函数点x(k)处Hesse矩阵
39     G=subs(Hesse,[x1 x2],[x(1) x(2)]);
40     %计算目标函数点x(k)处搜索方向p
41     p=-inv(G)*g;
42     %p=-g;

```

```

43     %点 $x(k)$ 处的搜索步长
44     % $a_k = \text{Alpha}(p, g, G)$ ;
45     % $a_k = 1$ ;
46      $a_k = 1$ ;
47      $x_k = x + a_k * \text{double}(p')$ ;
48     %采用Armijo法则计算近似步长 $a_k$ 
49     while( $F(x_k(1), x_k(2)) > (F(x(1), x(2)) + 0.01 * \text{double}(p' * g) * a_k)$ )
50          $a_k = 0.5 * a_k$ ;
51          $x_k = x + a_k * \text{double}(p')$ ;
52     end
53      $x = x + \text{double}(a_k * p')$ ;
54     %输出结果
55     optim_fx=subs(f,[x1 x2],[x(1) x(2)]);
56     fprintf('Step[%d]: x=[ %f %f] optim_fx=%f\n',step,x(1),x(2),
57         double(optim_fx));
57     P(step,:)=x;
58     OPT(step,:)=optim_fx;
59     g=subs(Gradient,[x1 x2],[x(1) x(2)]);
60 end
61 %输出结果
62 optim_fx=subs(f,[x1 x2],[x(1) x(2)]);
63 fprintf('\n牛顿Armijo回溯法,共迭代 %d 步\n结果: \n x=[ %d %d]
64     optim_fx=%f\n',step,x(1),x(2),double(optim_fx));
64 P(step+1:N,:)=[]; %删去P中的多余空间
65 figure;
66 plot(P(:,1),P(:,2),'-o')
67 hold on;
68 xx = linspace(-1.5,1.5);
69 yy = linspace(-0.5,1.5);
70 [X,Y] = meshgrid(xx,yy);
71 Z=F(X,Y);
72 contour(X,Y,Z,'ShowText','on')
73 figure;
74 plot(OPT)
75 %%
76 % 计算精确步长 $a_k$ 
77 function a=Alpha(p,g,G)
78      $a = -(p' * g) / (p' * G * p)$ ;
79 end

```

```

80
81 %采用Armijo法则计算近似步长ak
82 function a=Armijo(x,p,g)
83     a=1;
84     xk=x+a*double(p);
85     while(F(xk(1),xk(2)) > F(x(1),x(2))+0.01*doubl(p'*g)*a)
86         a=0.9*a;
87         xk=xk+a*double(p);
88     end
89 end

```

```

1 %Conj_Grad 共轭梯度法
2 clc;
3 clear;
4 N=4;
5 G=zeros(N,N);
6 b=[-1,0,2,5^(0.5)]';
7 x0=zeros(N,1);
8 for i= 1:N
9     G(i,i) = 2;
10 end
11
12 for i= 1:N-1
13     G(i+1,i)=-1;
14     G(i,i+1)=-1;
15 end
16 x = x0;x'
17 g = G*x+b;
18 p = -g;
19 k = 0;
20 while 1
21     if norm(g, 2)<1e-6
22         break
23     end
24     k = k + 1;
25
26     d=G*p;
27     a=(g'*g)/(p'*d);

```

```
28
29     x = x+a*p;x'
30     t=g+a*d;
31     beta=(t'*t)/(g'*g);
32     g=t;
33     p=-g+beta*p;
34
35 end
```