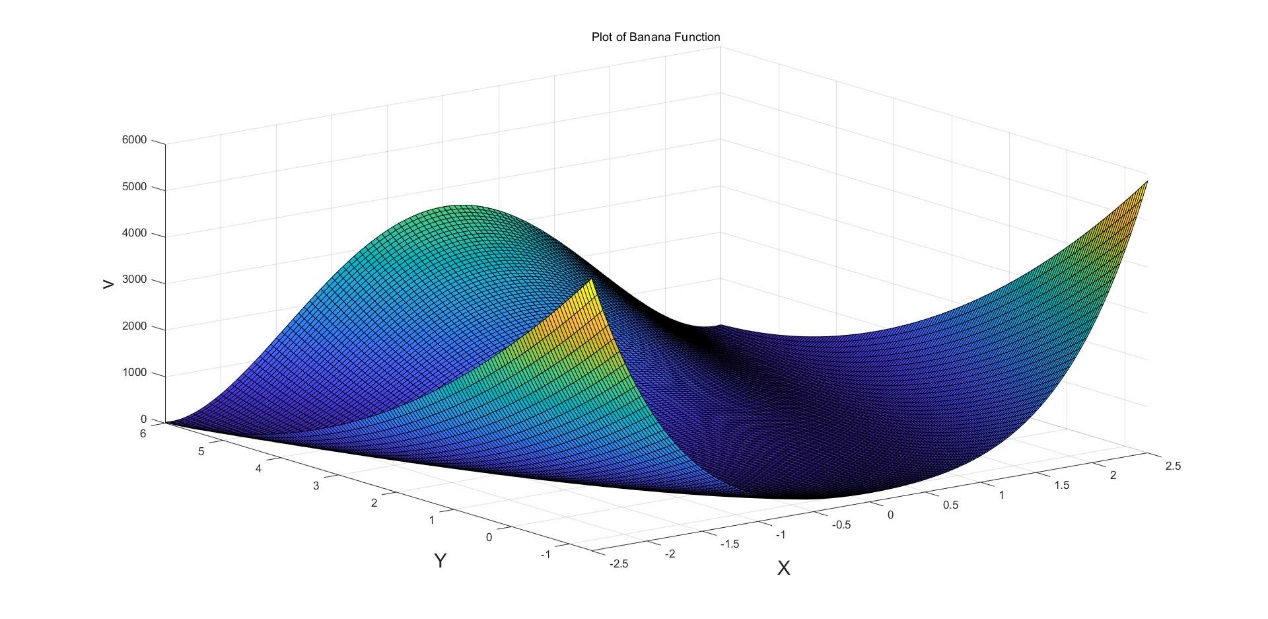
**Optimisation Course Work**

**A1)**

is the stationary point

the eigenvalues of is 0.394 and 1001.6. 802 > 0, all the eigenvalues of are larger than 0, the is positive definite. Thus = (1,1) is the minimizers.

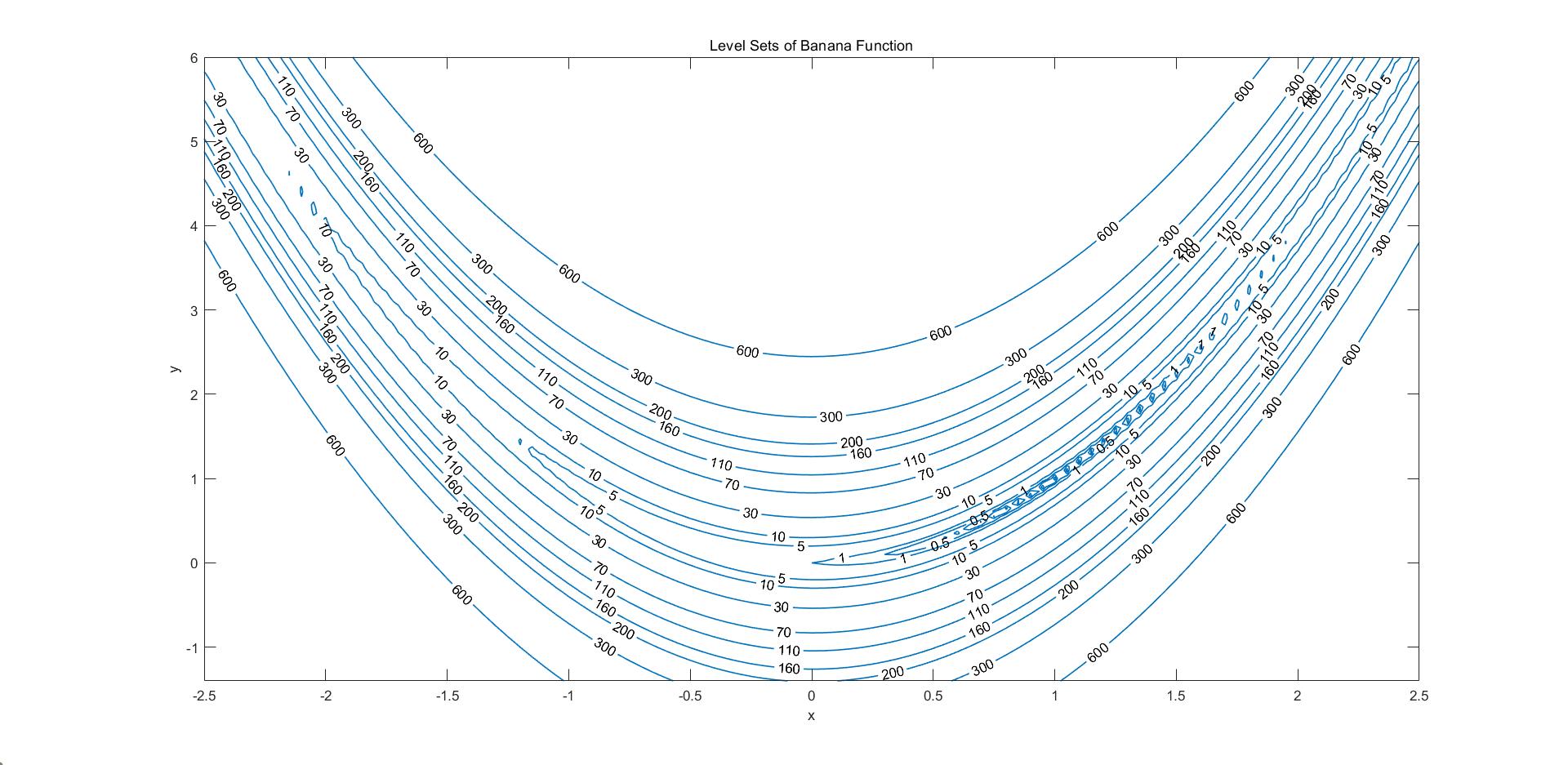
**A2）**In matlab, using meshgrid(X, Y) to construct the function and set the range of the x and y. According to the observation of the function, set X = -2.5: 0.05 :2.5(-2.5 to 2.5 with 0.05 margin) Y = -1.4: 0.05: 6 (-1.4 to 6 with 0.05 margin). Using surf(X,Y,v) to plot the function. As **Fig.1** shows.



**Fig.1**

Set level set L = [0,0.1,0.25,0.5,1,5,10,30,70,110,160,200,300,600];

Using contour(X,Y,v,L,'LineWidth',1,'LineColor','#0072BD','ShowText','on') to plot the level sets. (**Fig.2**)



**Fig.2**

**A3)**

**Armijo Line Search**

**Step 1.** Set

**Step 2.** If

set and STOP. Else go to **Step 3**.

**Step 3.** Set, and go to **Step 2**.

Using **while(1)** to implement this algorithm.

**Matlab code:**

function alphak = armijo(a,sigma,xk,dk,gamma,g)

j=0;

while(1)

alpha = a\*sigma^j;

x = xk+alpha\*dk;

phi=Banana(x)-Banana(xk);

if phi <=gamma\*alpha\*g'\*dk

alphak = alpha;

break;

end

j=j+1;

end

end

**Gradient Method**

**Step 0.** Given .

**Step 1**. Set .

**Step 2.** Compute . If STOP. Else set .

**Step 3.** Compute a step along the direction with any line search method such that

And

**Step 4.** Set Go to **Step 2.**

In this algorithm, because , we set when < **STOP**.

We set the parameter for Armijo =0.5; =0.35; a=1;

Initial point .

**Matlab code:**

%% Gradient Method

x0=[-0.75;1]; xk=x0; k=0; sigma = 0.5; gamma = 0.35; a = 1 ; Accuracy = 1e-5;

while (1)

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

g = gradient(xk);

dk = -g;

if norm(dk) < Accuracy

break;

end

alphak = armijo(a,sigma,xk,dk,gamma,g);

xk = xk+alphak\*dk;

k = k+1;

end

**A4)**

**Newton Method with Armijo Line Search**

**Step 0.** Given and ε > 0.

**Step 1**. Set .

**Step 2.** Compute , If STOP. Else compute . If is singular set and go to **Step 6**

**Step 3:** Compute Newton direction solving the (linear) system

**Step 4**. If

Set and go to **Step 6.**

**Step 5.** If

set ; if

set .

**Step 6.** Make a line search along assuming as initial estimate α = 1. Compute , set and go to **Step 2**

In this algorithm, we set when **STOP**. We set the parameter for Armijo =0.5; =0.35; a=1;

Initial point .Using SET to collect the point trajectory and J to collect the cost for plotting.

**Matlab Code:**

%% Newton's Method with Armijo

x0=; xk=x0; k=0; e=0.2; sigma = 0.5; gamma = 0.35; a = 1 ; Accuracy = 1e-5;

while (1)

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

g = gradient(xk);

if norm(g)==0 %%|| norm(g)<Accuracy

break;

else

H = hessian(xk);

if rank(H)==0

dk=-g;

else

s = -(H)^-1\*g;

if abs(g'\*s)<e\*norm(g)\*norm(s)

dk=-g;

else

if g'\*s<0

dk=s;

elseif g'\*s>0

dk=-s;

end

end

end

end

alphak = armijo(a,sigma,xk,dk,gamma,g);

xk = xk+alphak\*dk;

k = k+1;

end

**Newton Method Without Armijo**

**Step 0.** Given .

**Step 1**. Set .

**Step 2.** Compute

**Step 3**. Set Go to **Step 2.**

In this algorithm, we set when **STOP**. We set the parameter for Armijo =0.5; =0.35; a=1;

Initial point . Using SET to collect the point trajectory and J to collect the cost for plotting. Because it doesn’t use Armijo, so we set .

**Matlab Code:**

%% Newton's Method

x0=[-0.75;1]; xk=x0; k=0; Accuracy = 0;

while (1)

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

H = hessian(xk);

g = gradient(xk);

s = -(H)^-1\*g;

if norm(g)==0

break;

end

xk = xk+s;

k = k+1;

end

**A5)**

**Polak-Ribiere Algorithm**

**Step 0.** Given .

**Step 1**. Set .

**Step 2.** Compute . If STOP. Else let

**Step 3.** Compute performing a line search along .

**Step 4.** Set and go to **Step 2.**

Let initial point , using SET to collect the point trajectory and J to collect the cost for plotting. We set when < **STOP**. We set the parameter for Armijo =0.5; =0.35; a=1;

**Matlab Code:**

%% Polak-Ribiere algorithm

x0=[-0.75;1];

xk=x0;

k=0;

sigma = 0.5; gamma = 0.35; a = 1 ;

Accuracy = 1e-5;

while (1)

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

g = gradient(xk);

if norm(g) < Accuracy

break;

end

if k==0

dk=-g;

else

dk=-g+(dkT(:,k)\*(g'\*(g-gradient(Set(:,k))))/(norm(gradient(Set(:,k))))^2);

end

alphak = armijo(a,sigma,xk,dk,gamma,g);

xk = xk+alphak\*dk;

k = k+1;

dkT(:,k)=dk;

end

**A6)**

**BFGS Method**

**Step 0.** Given .

**Step 1**. Set .

**Step 2.** Compute . If STOP. Else compute with BFGS equation and set

**Step 3.** Compute performing a line search along .

**Step 4.** Set and go to **Step 2.**

**Broyden**

and

Let initial point , using SET to collect the point trajectory and J to collect the cost for plotting. We set when < **STOP**. We set the parameter for Armijo =0.5; =0.35; a=1;

**Matlab Code:**

%% Broyden-Fletcher-Goldfarb-Shanno algorithm

x0=[-0.75;1];

xk=x0;

k=0;phi=1;

sigma = 0.5; gamma = 0.35; a = 1 ;

Accuracy = 1e-5;

while (1)

g = gradient(xk);

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

if norm(g) < Accuracy

break;

end

if k==0

Hk=eye(2);

else

Se=(D\*D')/(D'\*G);

Th=(Hk\* (G\*G')\*Hk)/(G'\*Hk\*G);

Fo=phi\*(vk\*vk');

Hk=Hk+Se-Th+Fo;

end

dk=-Hk\*g;

alphak = armijo(a,sigma,xk,dk,gamma,g);

xk = xk+alphak\*dk;

k = k+1;

Set(:,k+1) = xk;

D=Set(:,k+1)-Set(:,k);

G=gradient(Set(:,k+1))-gradient(Set(:,k));

vk=((G'\*Hk\*G)^1/2)\*((D/(D'\*G))-((Hk\*G)/(G'\*Hk\*G)));

end

**A7)**

**Simplex Method**

**Step 0.** Given .

**Step 1.** Set step size (side length of equilateral triangle of 3 initial points),

Calculating

**Step 2.** Stopping criterion: Exist a , when ,

**STOP**. Else computing .

**Step 3.** Suppose is the vertex where the function *f* attains the maximum value is determined. This vertex is reflected with respect to the center of the simplex, the new vertex , where >0 and repeat the procedure until it is satisfied the stopping criterion.

Let initial point ,, step=0.5,, using SET to collect the point trajectory and J to collect the cost for plotting. is an important constant for this algorithm. If is not small enough, it cannot converge to (1,1). But this algorithm can’t converge to v(1,1) for different step size. This is caused by some new points can be larger than previous one. So, we need improve this algorithm.

**Matlab Code:**

%% Simplex Method

x0=[-0.75;1]; k=0; step = 0.5;

x1 = [x0(1)+step;x0(2)];

x2 = [x0(1)+step/2; x0(2)-sqrt(3)/2\*step];

P = [x0 x1 x2];

I=1;

a = 1.9733 ;

Accuracy = 1e-25;

Set(:,1) = x0; Set(:,2) = x1; Set(:,3) = x2;

while (1)

jk = log((P(1,I)-1)^2 + (P(2,I)-1)^2);

Jk(:,k+1) = jk;

fA = (Banana(P(:,1))+Banana(P(:,2))+Banana(P(:,3)))/3;

criterion=((Banana(P(:,1))-fA)^2+(Banana(P(:,2))-fA)^2+(Banana(P(:,3))-fA)^2)/3;

if criterion < Accuracy

break;

end

Xc = (P(:,1)+P(:,2)+P(:,3))/3;

F = [Banana(P(:,1));Banana(P(:,2));Banana(P(:,3))];

[M,I]=max(F);

Xn = Xc+a\*(Xc-P(:,I));

P(:,I) = Xn;

k=k+1;

Set(:,k+3) = P(:,I);

end

**For the improvement converge properties of the algorithm.**

**One iteration of the simplex algorithm**

note is the worst point.

Generate a trial point x, by reflection

Where , , compute , consider 3 possibilities

1. , which shows that is neither the best nor worst point, replace by .
2. , is the new best point. The reflection is good and generate a point by expansion

, we replace by when ,else, replace by which means the expansion has failed.

1. then assume the polytope is too large and generate a new point by contradiction

Where (contraction coefficient). The contraction is succeeded when . Replace by , else contract again.

Let initial point ,, step=0.8,, using SET to collect the point trajectory and J to collect the cost for plotting. Use Pc to store the new point . With this algorithm, For different step (side length of equilateral triangle of 3 initial points), sequences are always converging to a stationary point of v(1, 1).

,

**Matlab Code:**

%% Simplex Method Improvement

x0=[-0.75;1];

k=0;

step = 0.8;

x1 = [x0(1)+step;x0(2)];

x2 = [x0(1)+step/2; x0(2)-sqrt(3)/2\*step];

P = [x0 x1 x2];

I=1;

beta = 2; gamma = 0.5; alpha = 1.95 ;

Accuracy = 1e-20;

Set(:,1) = x0; Set(:,2) = x1; Set(:,3) = x2;

while (1)

jk = log((P(1,I)-1)^2 + (P(2,I)-1)^2);

Jk(:,k+1) = jk;

fA = (Banana(P(:,1))+Banana(P(:,2))+Banana(P(:,3)))/3;

criterion = ((Banana(P(:,1))-fA)^2+(Banana(P(:,2))-fA)^2+(Banana(P(:,3))-fA)^2)/3;

if criterion < Accuracy

break;

end

Xa = (P(:,1)+P(:,2)+P(:,3))/3;

F = [Banana(P(:,1));Banana(P(:,2));Banana(P(:,3))];

[M,I]=max(F);

Xr = Xa+alpha\*(Xa-P(:,I));

Pc=P;

Pc(:,I) = Xr;

F = [Banana(Pc(:,1));Banana(Pc(:,2));Banana(Pc(:,3))];

if min(F)~=Banana(Xr) && max(F)~=Banana(Xr)

P(:,I) = Xr;

elseif min(F)==Banana(Xr)

Xe= Xr+ beta\*(Xr-Xa);

if Banana(Xe)<Banana(Xr)

P(:,I) = Xe;

else

P(:,I) = Xr;

end

elseif max(F)==Banana(Xr)

Xn=P(:,I);

while(1)

Xc=Xa+gamma\*(Xn-Xa);

if Banana(Xc)<Banana(Xn)

P(:,I) = Xc;

break;

else

Xn=Xc;

end

end

end

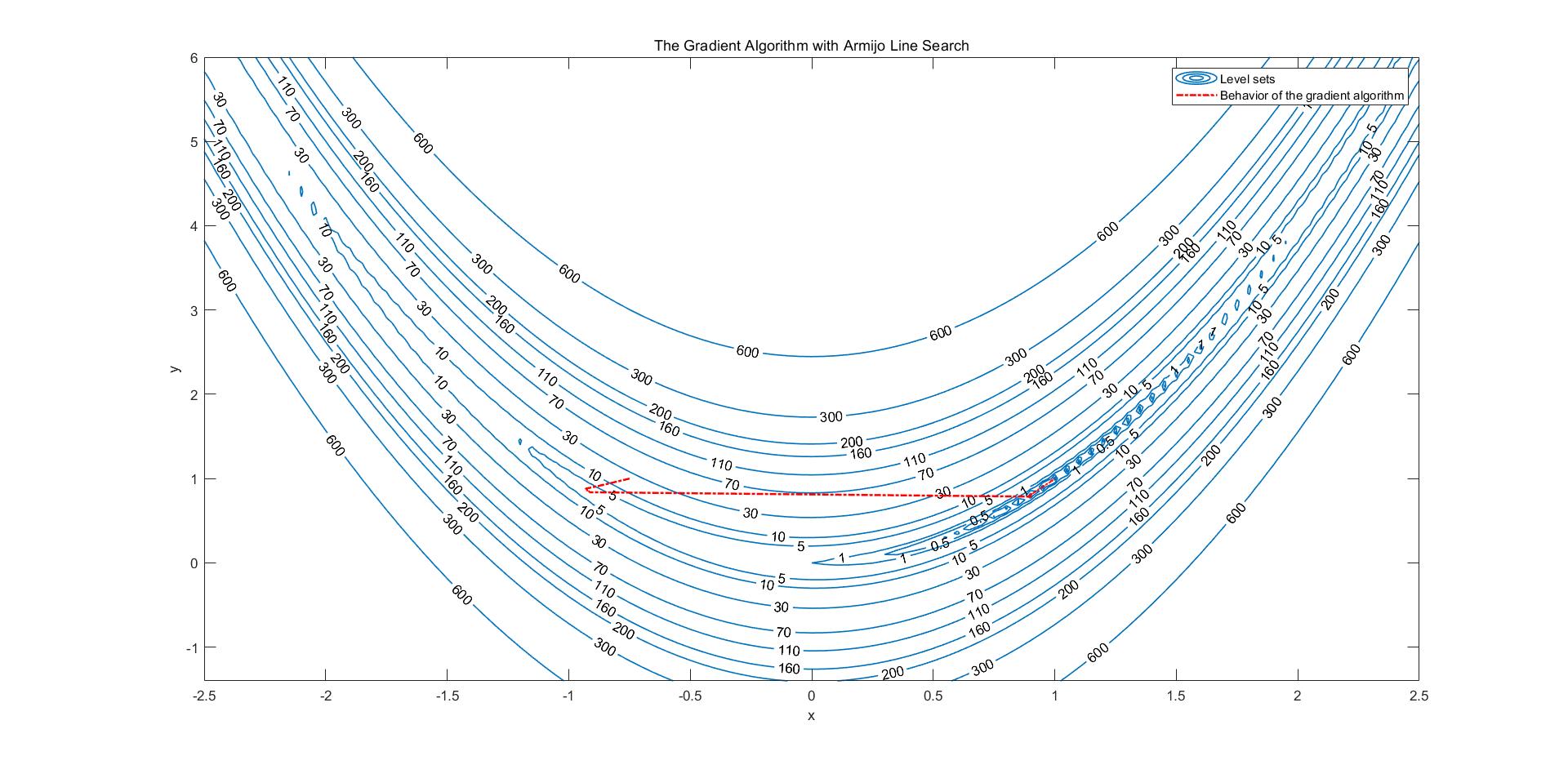
k=k+1;

Set(:,k+3) = P(:,I);

end

**A8a)**

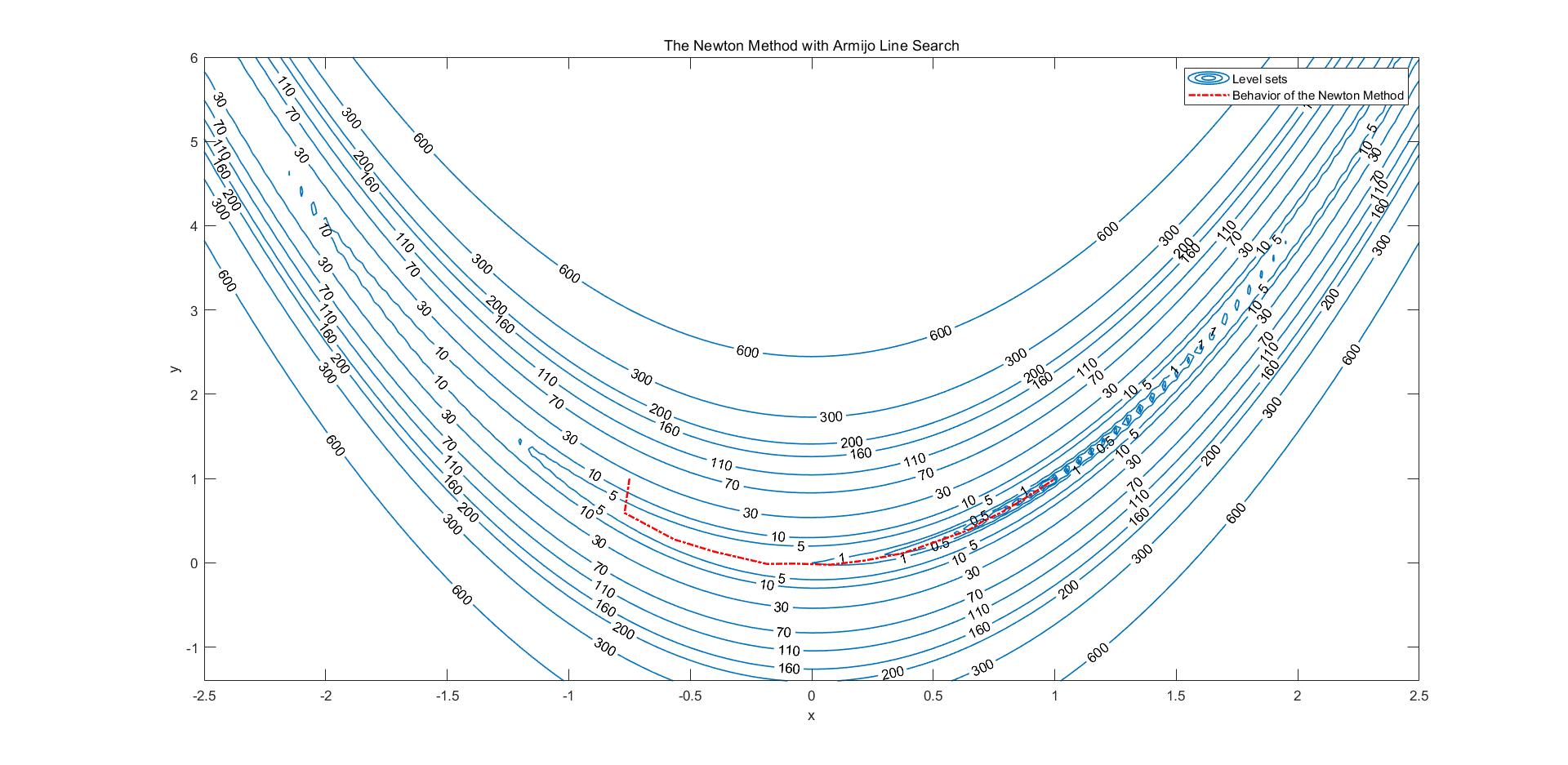
**Gradient Method with Armijo Line Search** **(Fig.3)**



**Fig.3**

**Fig.3** shows that the sequence converging to a stationary point of v(1, 1)

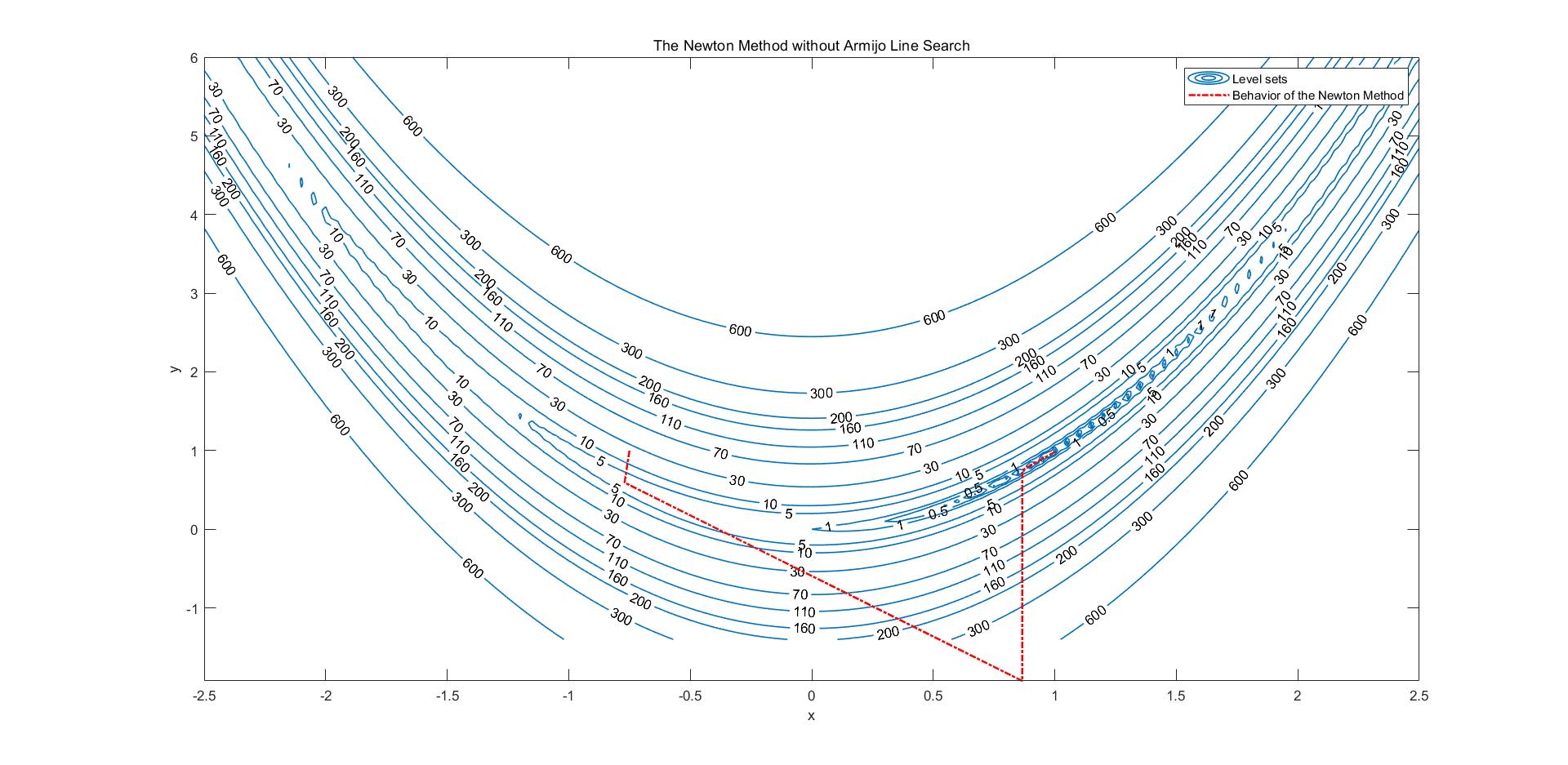
**The Newton Method with Armijo Line Search (Fig.4)**

****

**Fig.4**

**Fig. 4** shows that the sequence exactly converging to a stationary point of v(1, 1)

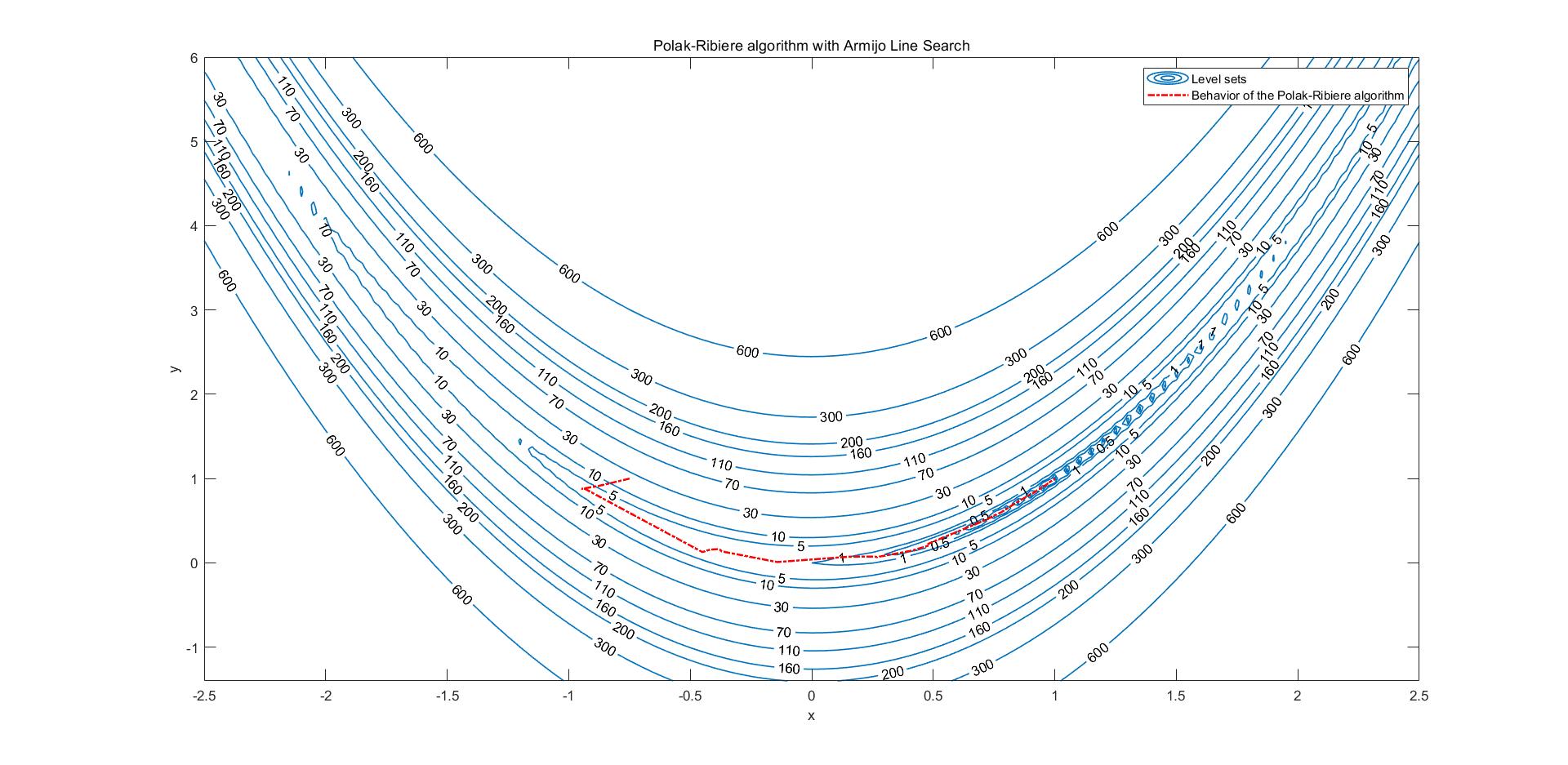
**The Newton Method without Armijo Line Search (Fig.5)**

****

**Fig.5**

**Fig.5** shows that the sequence exactly converging to a stationary point of v(1, 1)

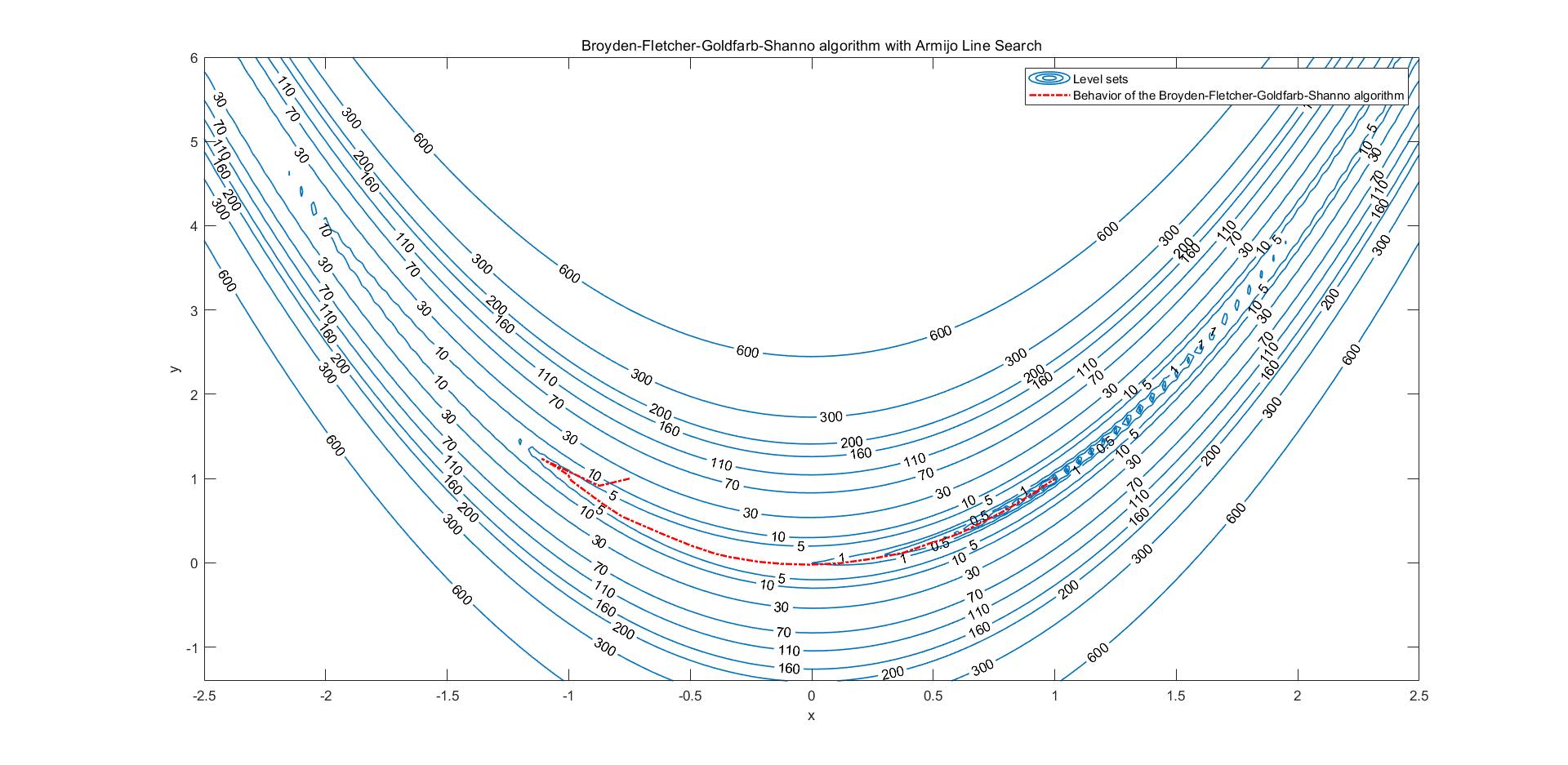
**Polak-Ribere Algorithm with Armijo Line Search (Fig.6)**

****

**Fig.6**

**Fig.6** shows that the sequence converging to a stationary point of v(1, 1)

**Broyden-Fletcher-Goldfarb-Shanno algorithm with Armijo Line Search (Fig.7)**

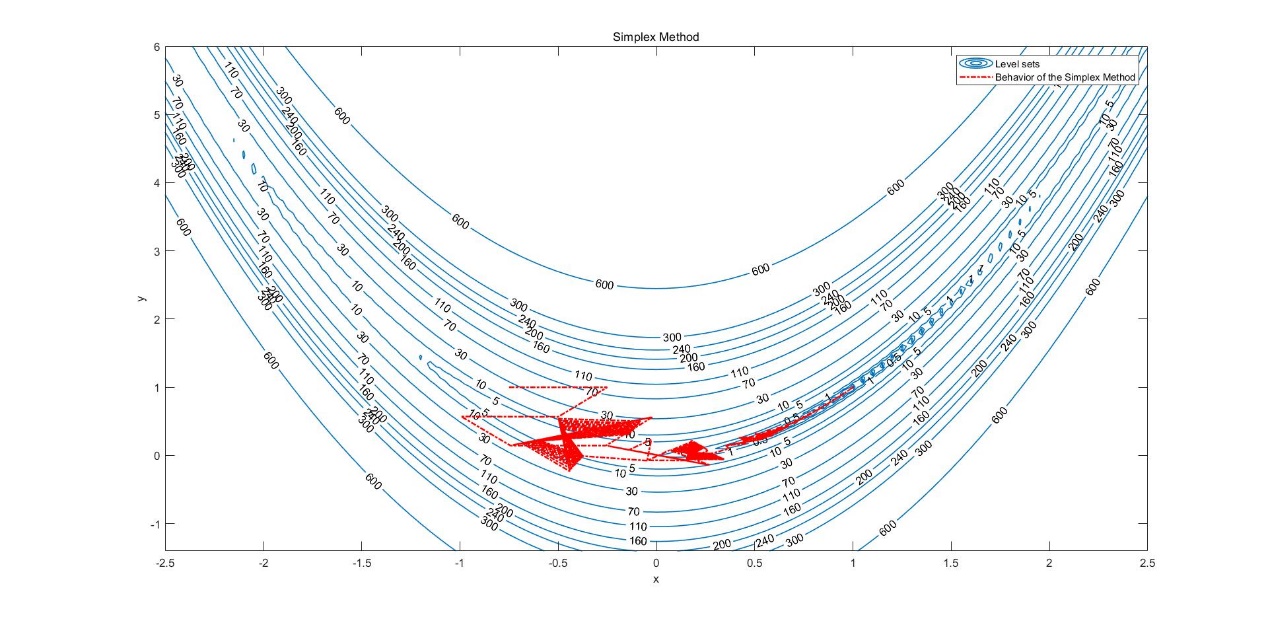
****

**Fig.7**

**Fig.7** shows that the sequence converging to a stationary point of v(1, 1)

**Simplex Method**

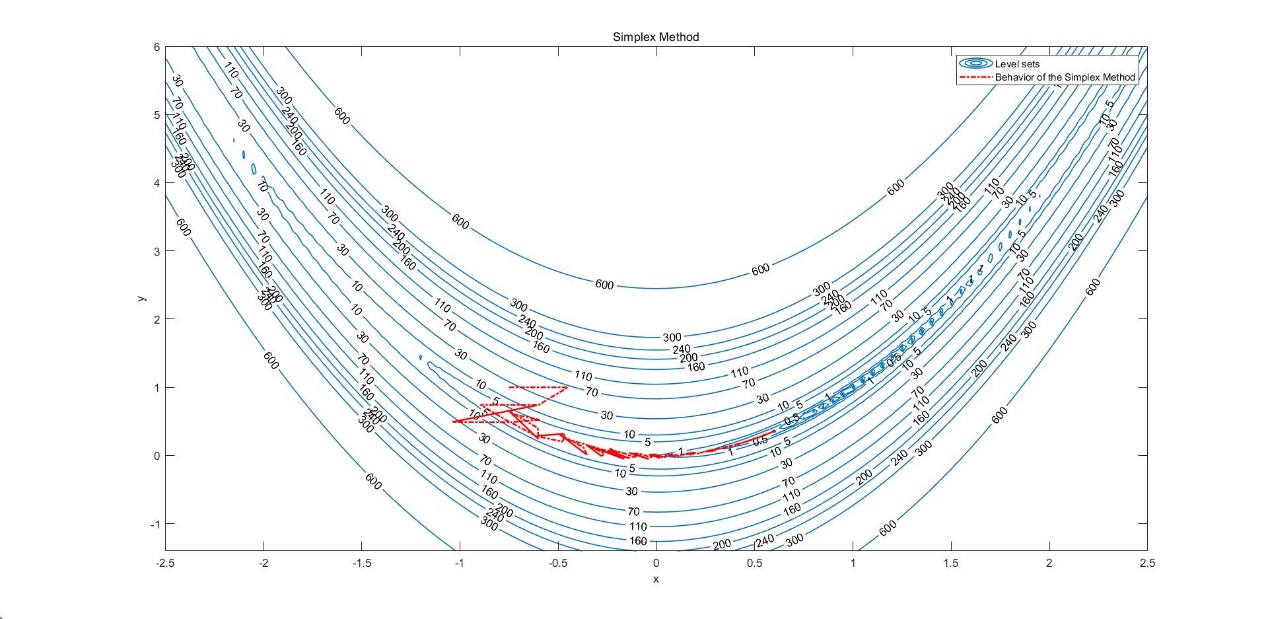
**Step size=0.5（Fig.8）**

****

**Fig.8**

**Fig.8** shows that the sequence converging to a stationary point of v(1, 1)

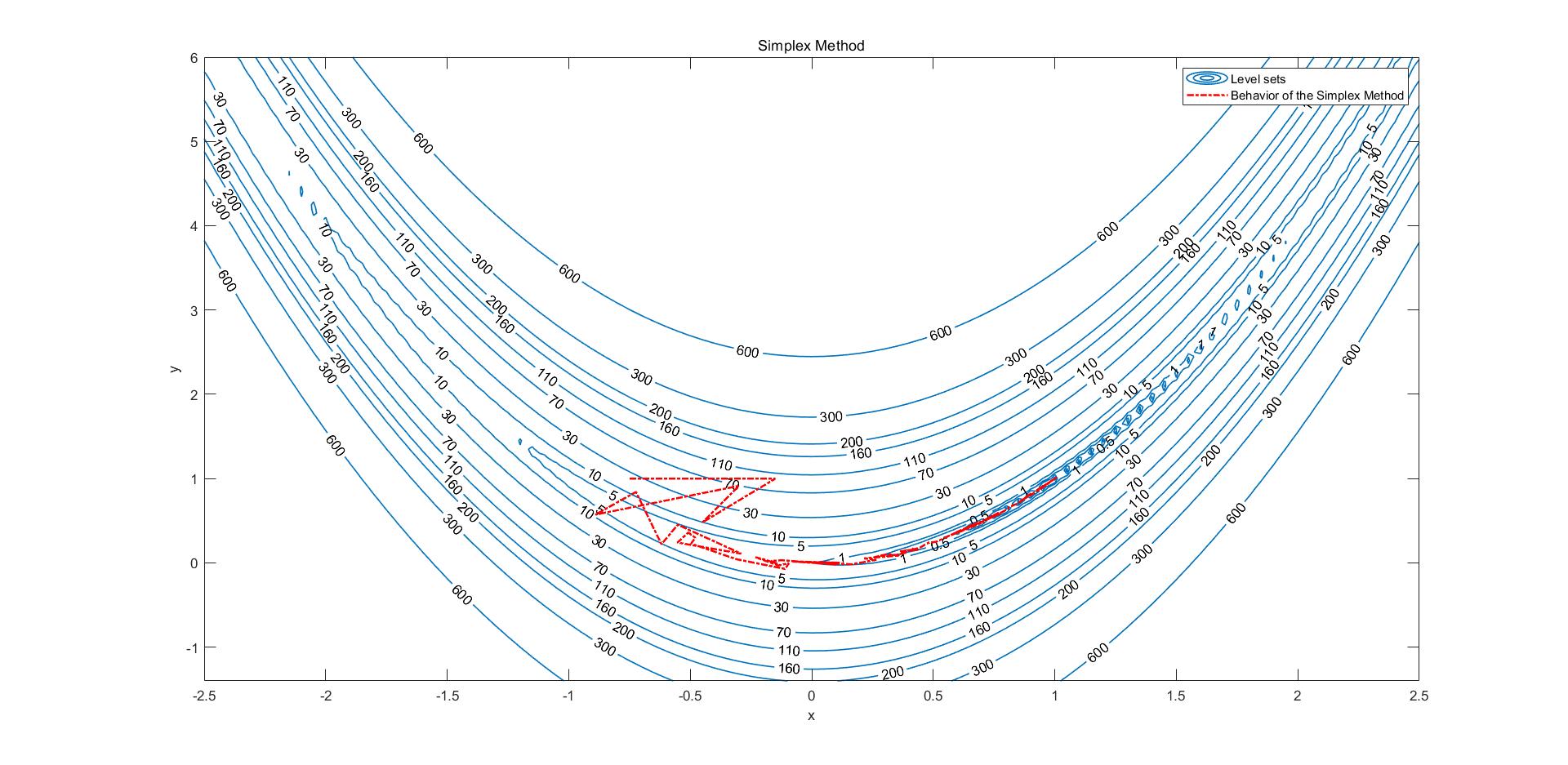
**Step size = 0.3 (Fig.9)**

****

**Fig.9**

**Fig.9** shows that the sequence cannot converging to a stationary point of v(1, 1)

**Improvement Simplex Method (Fig.10)**

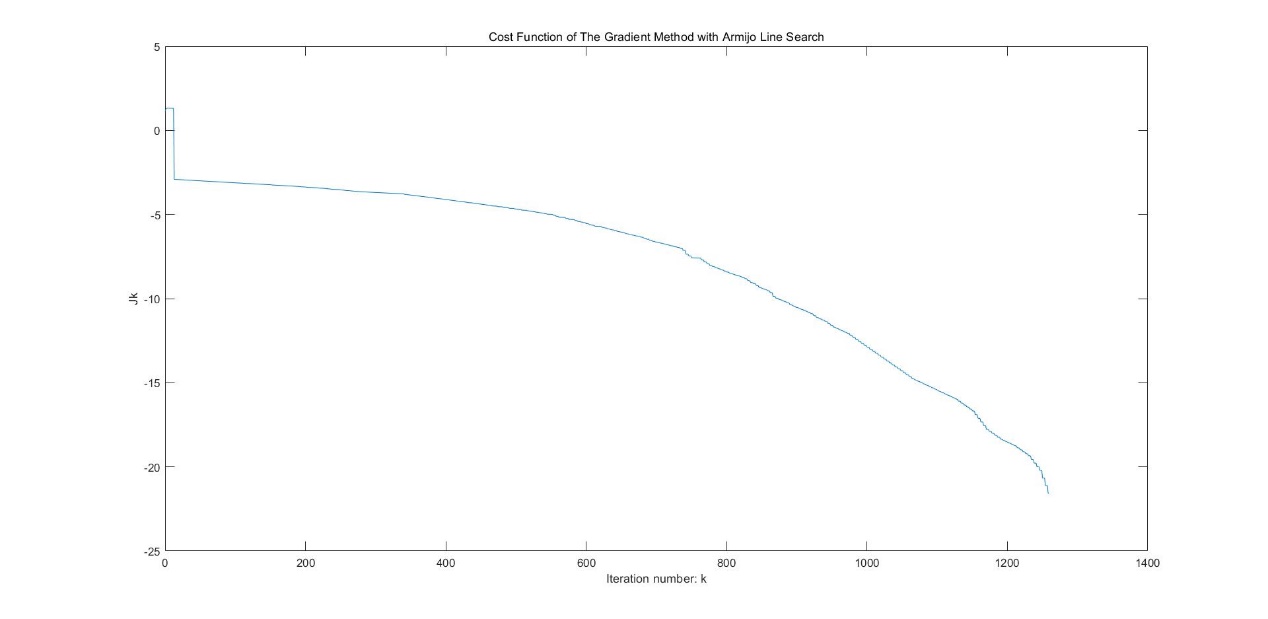
****

**Fig.10**

**Fig.10** shows that the sequence converging to a stationary point of v(1, 1)

**A8b)**

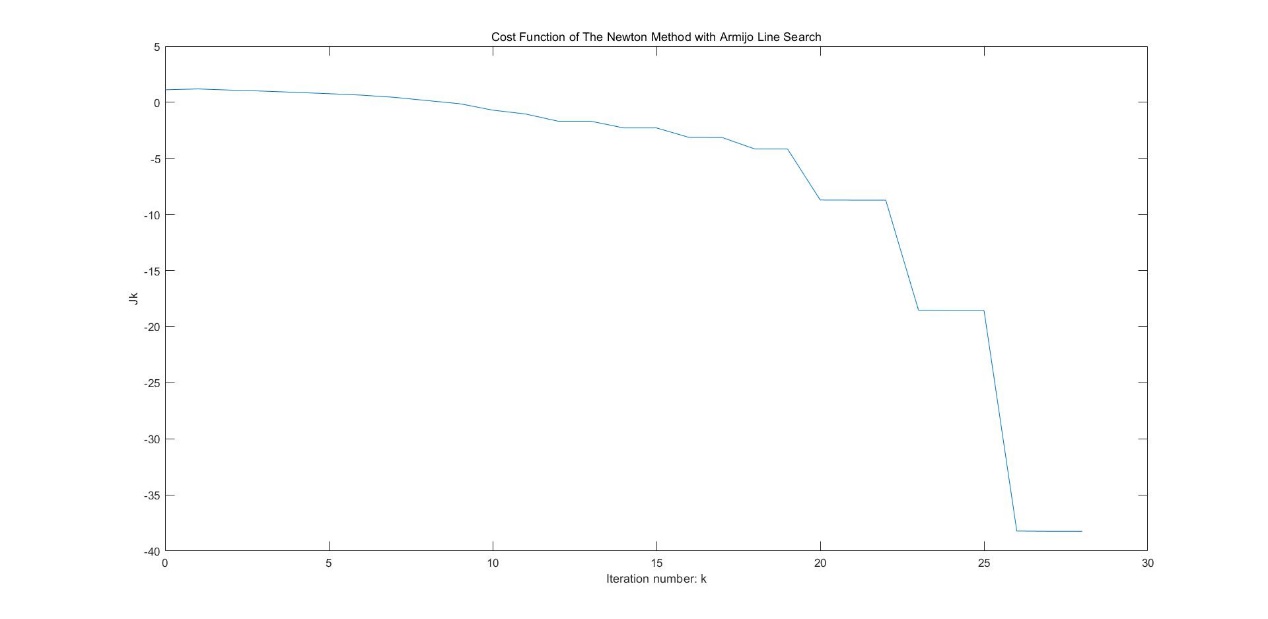
**Cost function of Gradient Method with Armijo Line Search (Fig.11)**

****

**Fig.11**

**Fig.11** shows that the cost become small and approach negative infinite, the sequence is converging to a stationary point of v(1, 1). Convergence speeds up as the number of iterations increases. The iteration number is k=1259. The speed of convergence is Linear for quadradic function.

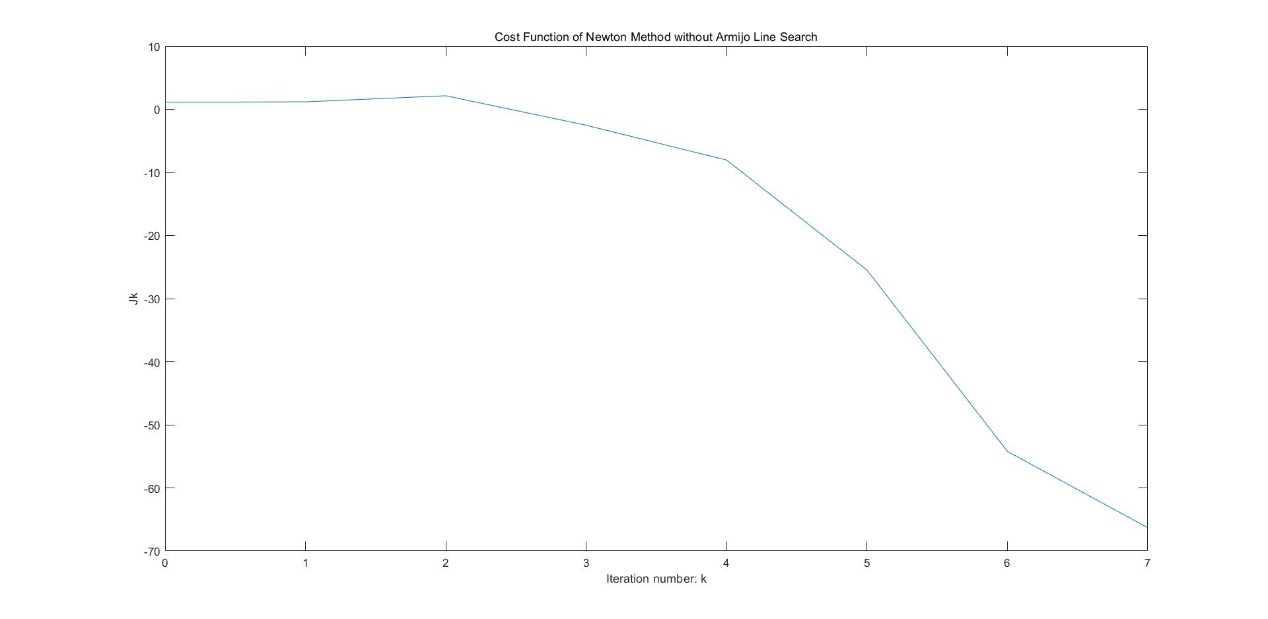
**Cost function of Newton Method with Armijo Line search (Fig.12)**

****

**Fig.12**

**Fig.12** shows that the cost become small and reach negative infinite, the sequence is converging to a stationary point of v(1, 1). Convergence speeds up as the number of iterations increases. The iteration number is k=29.when k=29, goes to infinity. The speed of convergence is quadratic.

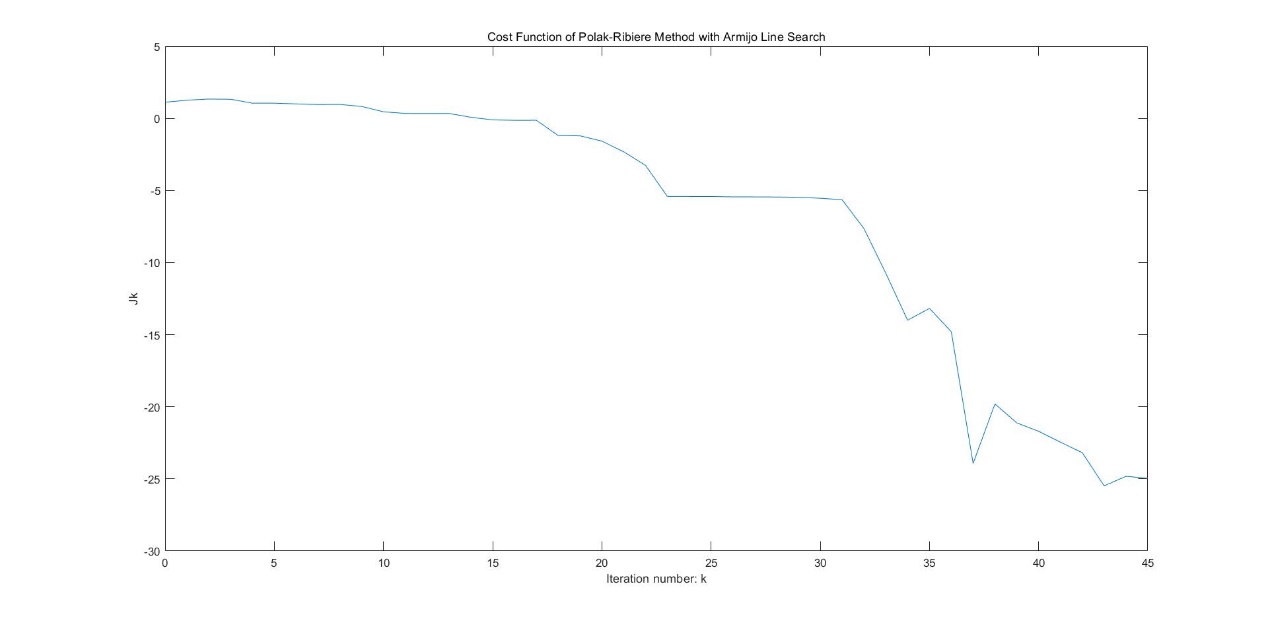
**Cost function of** **Newton Method without Armijo Line search (Fig.13)**

****

**Fig.13**

**Fig.13** shows that the cost become small and reach negative infinite, the sequence is converging to a stationary point of v(1, 1). Convergence speeds up as the number of iterations increases. The iteration number is k=8. When k=8, goes to infinity. The speed of convergence is faster than Newton Method with Armijo Line search, but it cannot converge for all initial points.

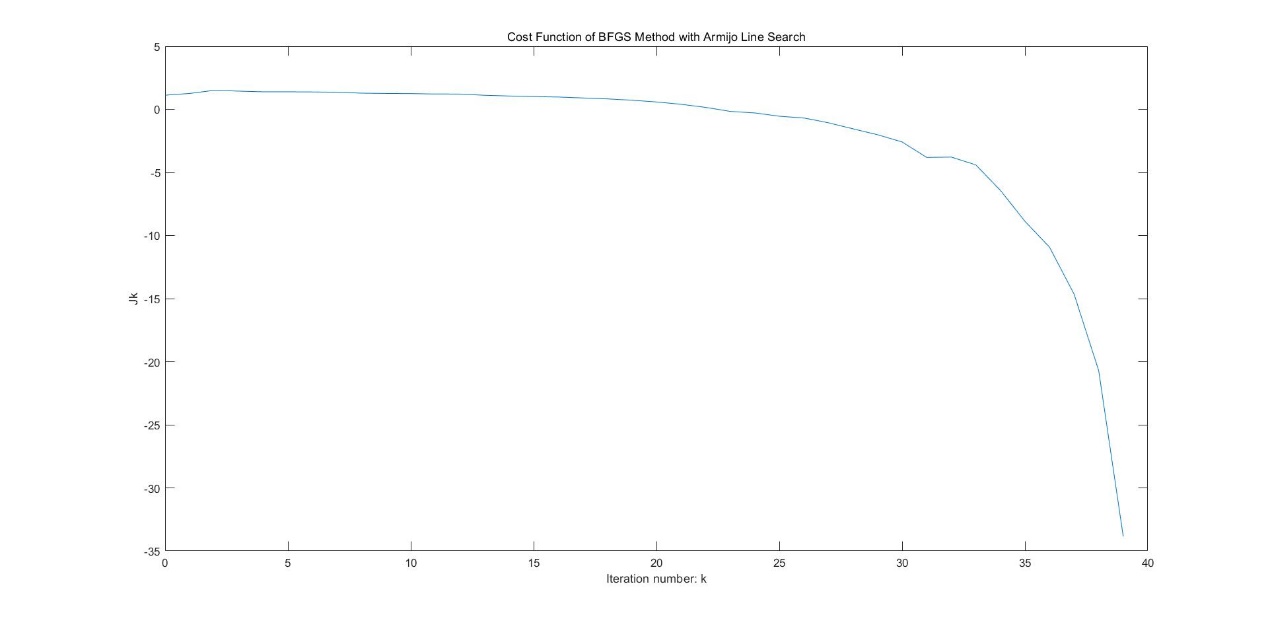
**Cost function of Polak-Ribere algorithm with Armijo Line Search (Fig.14)**

****

**Fig.14**

**Fig.14** shows that the cost become small and approach negative infinite, the sequence is converging to a stationary point of v(1, 1). Convergence speeds up as the number of iterations increases. The iteration number is k=45. The algorithm uses a restart procedure. It is possible to get the quadradic speed of convergence.

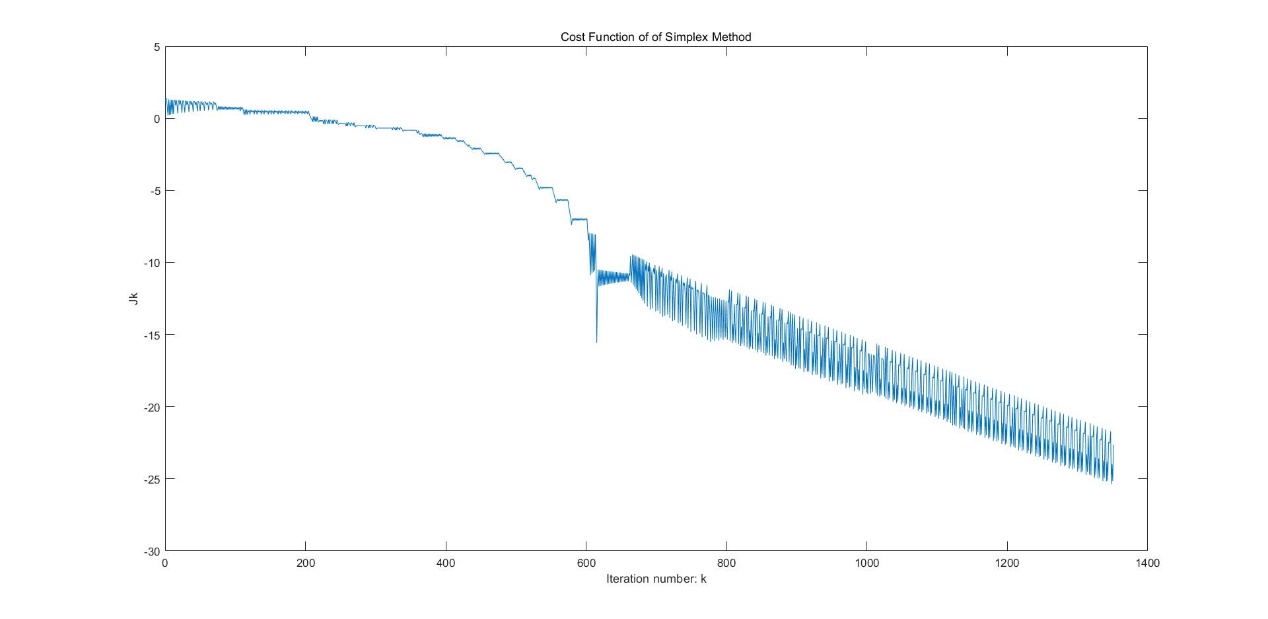
**Cost function of Broyden-Fletcher-Goldfarb-Shanno algorithm with Armijo Line Search (Fig.15)**

****

**Fig.15**

**Fig.15** shows that the cost become small and approach negative infinite, the sequence is converging to a stationary point of v(1, 1). Convergence speeds up as the number of iterations increases. The iteration number is k=39. The speed of convergence is super linear.

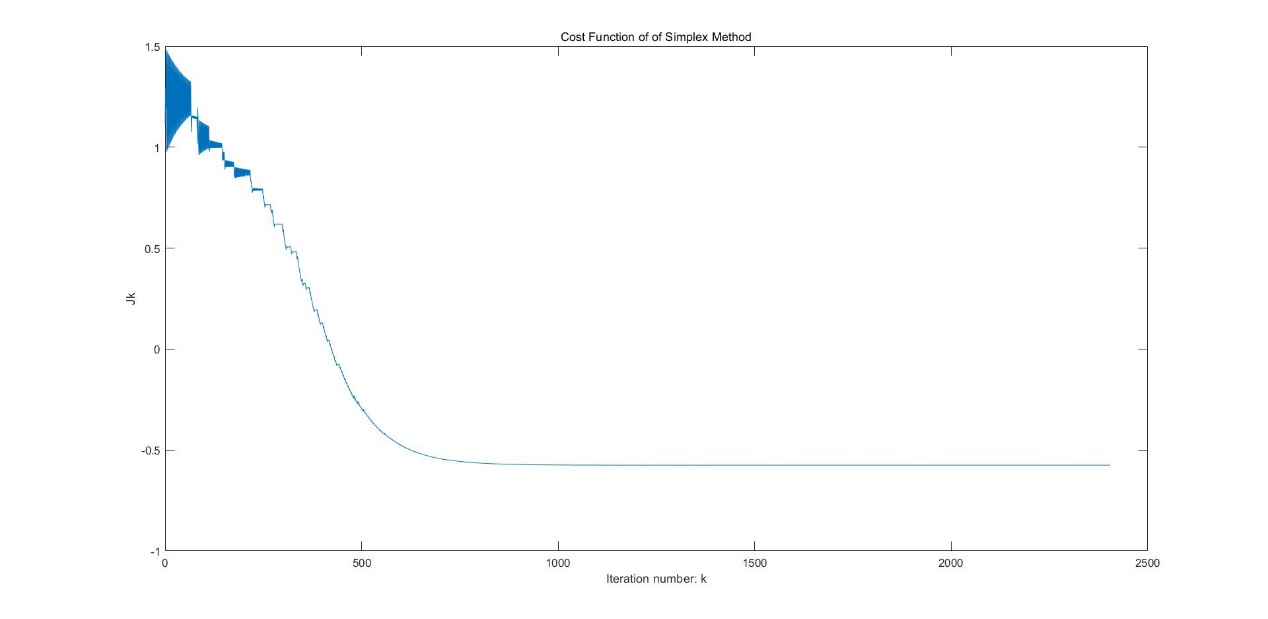
**Cost function of Simplex Method with step=0.5** (side length of equilateral triangle of 3 initial points) **(Fig.16)**

****

**Fig.16**

**Fig.16** shows that the cost become small and approach negative infinite, the sequence is converging to a stationary point of v(1, 1). The iteration number is k=1351. The speed of convergence is linear which is slow.

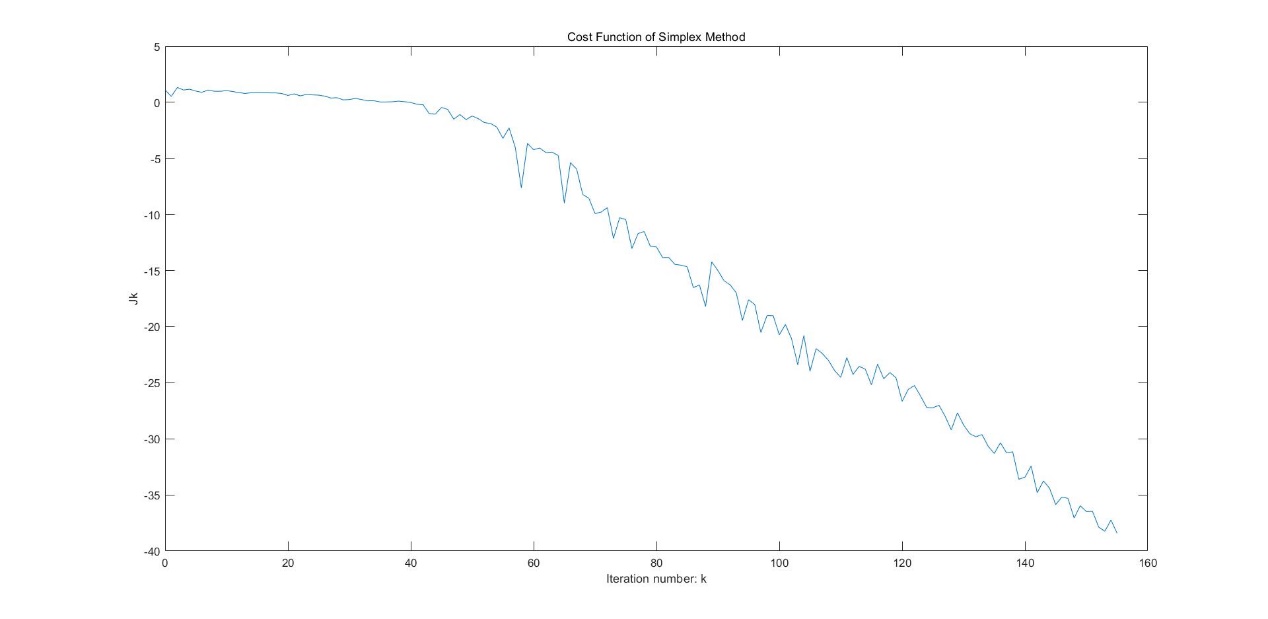
**Cost function of Simplex Method with step=0.3 (**side length of equilateral triangle of 3 initial points**) (Fig.17)**

****

**Fig.17**

**Fig.17** shows that the cost do not approaching negative infinity, the sequence is not converging to a stationary point of v(1, 1).

**Cost function of Improvement Simplex Method (Fig.18)**

****

**Fig.18**

**Fig.18** shows that the cost become small and approach negative infinite after modifying, the sequence is converging to a stationary point of v(1, 1). Convergence speeds up as the number of iterations increases. The iteration number is k=155. The speed of convergence is improved by the modifying.

|  |  |
| --- | --- |
| **Algorithm** | **Iteration Step** |
| Gradient Method with Armijo Line Search | **1259** |
| Newton Method with Armijo Line search | **29** |
| Newton Method without Armijo Line search | **8** |
| Polak-Ribere algorithm with Armijo Line Search | **45** |
| Broyden-Fletcher-Goldfarb-Shanno algorithm with Armijo Line Search | **39** |
| Simplex Method with step=0.5 | **1351** |
| Improvement Simplex Method | **155** |

**B1）**Construct the Lagrangian

The necessary conditions of optimality are

⇒ ⇒ ⇒

The candidate optimal point is

Which satisfying such condition

**B2)** Second order

For s

)

Calculate s

)

Hence the point is constrained local minimizers

**B3)**

Set

⇒

hence ( is the optimal point which gives minimum value of -1.

**B4)** Construct the Lagrangian

Multiply [1,1] on two sides

0 = + [1, 1]

0 =

Construct exact penalty function

(

Which is

Calculate . Stationary points are the solutions of

Stationary point is = (1,1,1).

=

For and sufficiently small. is positive definite.

So unconstrained minimizer of G(x) = (1,1,1) is a solution of the considered constrained optimization problem

**B5)**

The exact augmented Lagrange function with equality constrain

and

Solving equations to get the stationary point

⇒ .

Second order condition:

=

Because It is constructed by 3 components,is sufficiently small, which means is sufficiently large, and is sufficiently large and they all multiply a constant matrix. So is positive definite.

So the unconstrained minimizer of S(x, λ) yields a solution of the considered constrained optimization problem and the corresponding optimal multiplier.

**Appendix (Matlab Code)**

1. **Armijo**

function alphak = armijo(a,sigma,xk,dk,gamma,g)

j=0;

while(1)

alpha = a\*sigma^j;

x = xk+alpha\*dk;

phi=Banana(x)-Banana(xk);

if phi <=gamma\*alpha\*g'\*dk

alphak = alpha;

break;

end

j=j+1;

end

end

1. **Rosenbrock function**

function v = Banana(x)

% compute the value of the function at point x

X = x(1);

Y = x(2);

v = 100\*(Y-X.^2).^2+(1-X).^2;

end

1. **Gradient**

function grad = gradient(x)

syms X Y;

v = 100\*(Y-X.^2).^2+(1-X).^2;

X = x(1);

Y = x(2);

grad = gradient(v);

grad = eval(grad);

end

**4. Hessian Matrix**

function H = hessian(x)

syms X Y;

v = 100\*(Y-X.^2).^2+(1-X).^2;

X = x(1);

Y = x(2);

grad = gradient(v);

H = jacobian(grad);

H = eval(H);

end

**5. A1, A2, function plot and level sets plot**

clear;

%% Banana function

X = -2.5: 0.05 :2.5;

Y = -1.4: 0.05 : 6;

[X, Y] = meshgrid(X, Y);

v = 100\*(Y-X.^2).^2 +(1-X).^2;

figure(1)

surf(X, Y, v );

colormap = jet;

axis([-2.5, 2.5, -1.4, 6, 0, 6000]);

xlabel('X', 'fontsize', 18);

ylabel('Y', 'fontsize', 18);

zlabel('v', 'fontsize', 18);

title('Plot of Banana Function');

%% Level sets

figure(2)

L = [0,0.1,0.25,0.5,1,5,10,30,70,110,160,200,300,600];

contour(X,Y,v,L,'LineWidth',1,'LineColor','#0072BD','ShowText','on');

xlabel('x');

ylabel('y');

title('Level Sets of Banana Function');

**6. A3 Gradient Method**

clear;

%% Gradient Method

x0=[-0.75;1];

xk=x0;

k=0;

sigma = 0.5; gamma = 0.35; a = 1 ;

Accuracy = 1e-5;

while (1)

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

g = gradient(xk);

dk = -g;

if norm(dk) < Accuracy

break;

end

alphak = armijo(a,sigma,xk,dk,gamma,g);

xk = xk+alphak\*dk;

k = k+1;

end

%% Visualization

X = -2.5: 0.05 :2.5;

Y = -1.4: 0.05 : 6;

[X, Y] = meshgrid(X, Y);

v = 100\*(Y-X.^2).^2 +(1-X).^2;

figure(1)

L = [0,0.1,0.25,0.5,1,5,10,30,70,110,160,200,300,600];

contour(X,Y,v,L,'LineWidth',1,'LineColor','#0072BD','ShowText','on');

hold on;

plot(Set(1,:),Set(2,:),'r-.','LineWidth',1.5);

xlabel('x'); ylabel('y');

title('The Gradient Algorithm with Armijo Line Search');

legend('Level sets','Behavior of the gradient algorithm');

figure(2)

Xaxis=(0:k);

plot(Xaxis(1,:),Jk(1,:));

xlabel('Iteration number: k'); ylabel('Jk'); title('Cost Function of The Gradient Method with Armijo Line Search');

**7. A4.1 Newton's Method with Armijo**

clear;

%% Newton's Method With Armijo

x0=[-0.75;1];

xk=x0;

k=0;

e=0.2;

sigma = 0.5; gamma = 0.35; a = 1 ;

Accuracy = 1e-5;

while (1)

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

g = gradient(xk);

if norm(g)==0 %%|| norm(g)<Accuracy

break;

else

H = hessian(xk);

if rank(H)==0

dk=-g;

else

s = -(H)^-1\*g;

if abs(g'\*s)<e\*norm(g)\*norm(s)

dk=-g;

else

if g'\*s<0

dk=s;

elseif g'\*s>0

dk=-s;

end

end

end

end

alphak = armijo(a,sigma,xk,dk,gamma,g);

xk = xk+alphak\*dk;

k = k+1;

end

%% Visualization

X = -2.5: 0.05 :2.5;

Y = -1.4: 0.05 : 6;

[X, Y] = meshgrid(X, Y);

v = 100\*(Y-X.^2).^2 +(1-X).^2;

figure(1)

L = [0,0.1,0.25,0.5,1,5,10,30,70,110,160,200,300,600];

contour(X,Y,v,L,'LineWidth',1,'LineColor','#0072BD','ShowText','on');

hold on;

plot(Set(1,:),Set(2,:),'r-.','LineWidth',1.5);

xlabel('x'); ylabel('y');

title('The Newton Method with Armijo Line Search');

legend('Level sets','Behavior of the Newton Method');

figure(2)

Xaxis=(0:k);

plot(Xaxis(1,:),Jk(1,:));

xlabel('Iteration number: k'); ylabel('Jk'); title('Cost Function of The Newton Method with Armijo Line Search');

**8. A4.2 Newton's Method without Armijo**

clear;

%% Newton's Method

x0=[-0.75;1];

xk=x0;

k=0;

Accuracy = 0;

while (1)

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

H = hessian(xk);

g = gradient(xk);

s = -(H)^-1\*g;

if norm(g)==0

break;

end

xk = xk+s;

k = k+1;

end

%% Visualization

X = -2.5: 0.05 :2.5;

Y = -1.4: 0.05 : 6;

[X, Y] = meshgrid(X, Y);

v = 100\*(Y-X.^2).^2 +(1-X).^2;

figure(1)

L = [0,0.1,0.25,0.5,1,5,10,30,70,110,160,200,300,600];

contour(X,Y,v,L,'LineWidth',1,'LineColor','#0072BD','ShowText','on');

hold on;

plot(Set(1,:),Set(2,:),'r-.','LineWidth',1.5);

xlabel('x'); ylabel('y');

title('The Newton Method without Armijo Line Search');

legend('Level sets','Behavior of the Newton Method');

figure(2)

Xaxis=(0:k);

plot(Xaxis(1,:),Jk(1,:));

xlabel('Iteration number: k'); ylabel('Jk'); title('Cost Function of Newton Method without Armijo Line Search');

**9. A5 Polak-Ribiere Algorithm**

clear;

%% Polak-Ribiere algorithm

x0=[-0.75;1];

xk=x0;

k=0;

sigma = 0.5; gamma = 0.35; a = 1 ;

Accuracy = 1e-5;

while (1)

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

g = gradient(xk);

if norm(g) < Accuracy

break;

end

if k==0

dk=-g;

else

dk=-g+(dkT(:,k)\*(g'\*(g-gradient(Set(:,k))))/(norm(gradient(Set(:,k))))^2);

end

alphak = armijo(a,sigma,xk,dk,gamma,g);

xk = xk+alphak\*dk;

k = k+1;

dkT(:,k)=dk;

end

%% Visualization

X = -2.5: 0.05 :2.5;

Y = -1.4: 0.05 : 6;

[X, Y] = meshgrid(X, Y);

v = 100\*(Y-X.^2).^2 +(1-X).^2;

figure(1)

L = [0,0.1,0.25,0.5,1,5,10,30,70,110,160,200,240,300,600];

contour(X,Y,v,L,'LineWidth',1,'LineColor','#0072BD','ShowText','on');

hold on;

plot(Set(1,:),Set(2,:),'r-.','LineWidth',1.5);

xlabel('x'); ylabel('y');

title('Polak-Ribiere algorithm with Armijo Line Search');

legend('Level sets','Behavior of the Polak-Ribiere algorithm');

figure(2)

Xaxis=(0:k);

plot(Xaxis(1,:),Jk(1,:));

xlabel('Iteration number: k'); ylabel('Jk'); title('Cost Function of Polak-Ribiere Method with Armijo Line Search');

**10. A6 Broyden-Fletcher-Goldfarb-Shanno algorithm**

clear;

%% Broyden-Fletcher-Goldfarb-Shanno algorithm

x0=[-0.75;1];

xk=x0;

k=0;phi=1;

sigma = 0.5; gamma = 0.35; a = 1 ;

Accuracy = 1e-5;

while (1)

g = gradient(xk);

Set(:,k+1) = xk;

jk = log((xk(1)-1)^2 + (xk(2)-1)^2);

Jk(:,k+1) = jk;

if norm(g) < Accuracy

break;

end

if k==0

Hk=eye(2);

else

Se=(D\*D')/(D'\*G);

Th=(Hk\* (G\*G')\*Hk)/(G'\*Hk\*G);

Fo=phi\*(vk\*vk');

Hk=Hk+Se-Th+Fo;

end

dk=-Hk\*g;

alphak = armijo(a,sigma,xk,dk,gamma,g);

xk = xk+alphak\*dk;

k = k+1;

Set(:,k+1) = xk;

D=Set(:,k+1)-Set(:,k);

G=gradient(Set(:,k+1))-gradient(Set(:,k));

vk=((G'\*Hk\*G)^1/2)\*((D/(D'\*G))-((Hk\*G)/(G'\*Hk\*G)));

end

%% Visualization

X = -2.5: 0.05 :2.5;

Y = -1.4: 0.05 : 6;

[X, Y] = meshgrid(X, Y);

v = 100\*(Y-X.^2).^2 +(1-X).^2;

figure(1)

L = [0,0.1,0.25,0.5,1,5,10,30,70,110,160,200,300,600];

contour(X,Y,v,L,'LineWidth',1,'LineColor','#0072BD','ShowText','on');

hold on;

plot(Set(1,:),Set(2,:),'r-.','LineWidth',1.5);

xlabel('x'); ylabel('y');

title('Broyden-Fletcher-Goldfarb-Shanno algorithm with Armijo Line Search');

legend('Level sets','Behavior of the Broyden-Fletcher-Goldfarb-Shanno algorithm');

figure(2)

Xaxis=(0:k);

plot(Xaxis(1,:),Jk(1,:));

xlabel('Iteration number: k'); ylabel('Jk'); title('Cost Function of BFGS Method with Armijo Line Search');

**11. A7 Simplex Method**

clear;

%% Simplex Method

x0=[-0.75;1];

k=0;

step = 0.3;

x1 = [x0(1)+step;x0(2)];

x2 = [x0(1)+step/2; x0(2)-sqrt(3)/2\*step];

P = [x0 x1 x2];

I=1;

a = 1.9733 ;

Accuracy = 1e-25;

Set(:,1) = x0; Set(:,2) = x1; Set(:,3) = x2;

while (1)

jk = log((P(1,I)-1)^2 + (P(2,I)-1)^2);

Jk(:,k+1) = jk;

fA = (Banana(P(:,1))+Banana(P(:,2))+Banana(P(:,3)))/3;

criterion = ((Banana(P(:,1))-fA)^2+(Banana(P(:,2))-fA)^2+(Banana(P(:,3))-fA)^2)/3;

if criterion < Accuracy

break;

end

Xc = (P(:,1)+P(:,2)+P(:,3))/3;

F = [Banana(P(:,1));Banana(P(:,2));Banana(P(:,3))];

[M,I]=max(F);

Xn = Xc+a\*(Xc-P(:,I));

P(:,I) = Xn;

k=k+1;

Set(:,k+3) = P(:,I);

end

%% Visualization

X = -2.5: 0.05 :2.5;

Y = -1.4: 0.05 : 6;

[X, Y] = meshgrid(X, Y);

v = 100\*(Y-X.^2).^2 +(1-X).^2;

figure(1)

L = [0,0.1,0.25,0.5,1,5,10,30,70,110,160,200,240,300,600];

contour(X,Y,v,L,'LineWidth',1,'LineColor','#0072BD','ShowText','on');

hold on;

plot(Set(1,:),Set(2,:),'r-.','LineWidth',1.5);

xlabel('x'); ylabel('y');

title('Simplex Method');

legend('Level sets','Behavior of the Simplex Method');

figure(2)

Xaxis=(0:k);

plot(Xaxis(1,:),Jk(1,:));

xlabel('Iteration number: k'); ylabel('Jk'); title('Cost Function of of Simplex Method');

**12. A7 Modify Simplex Method**

clear;

%% Simplex Method Improvement

x0=[-0.75;1];

k=0;

step = 0.6;

x1 = [x0(1)+step;x0(2)];

x2 = [x0(1)+step/2; x0(2)-sqrt(3)/2\*step];

P = [x0 x1 x2];

I=1;

beta = 2; gamma = 0.5; alpha = 1.95 ;

Accuracy = 1e-35;

Set(:,1) = x0; Set(:,2) = x1; Set(:,3) = x2;

while (1)

jk = log((P(1,I)-1)^2 + (P(2,I)-1)^2);

Jk(:,k+1) = jk;

fA = (Banana(P(:,1))+Banana(P(:,2))+Banana(P(:,3)))/3;

criterion = ((Banana(P(:,1))-fA)^2+(Banana(P(:,2))-fA)^2+(Banana(P(:,3))-fA)^2)/3;

if criterion < Accuracy

break;

end

Xa = (P(:,1)+P(:,2)+P(:,3))/3;

F = [Banana(P(:,1));Banana(P(:,2));Banana(P(:,3))];

[M,I]=max(F);

Xr = Xa+alpha\*(Xa-P(:,I));

Pc=P;

Pc(:,I) = Xr;

F = [Banana(Pc(:,1));Banana(Pc(:,2));Banana(Pc(:,3))];

if min(F)~=Banana(Xr) && max(F)~=Banana(Xr)

P(:,I) = Xr;

elseif min(F)==Banana(Xr)

Xe= Xr+ beta\*(Xr-Xa);

if Banana(Xe)<Banana(Xr)

P(:,I) = Xe;

else

P(:,I) = Xr;

end

elseif max(F)==Banana(Xr)

Xn=P(:,I);

while(1)

Xc=Xa+gamma\*(Xn-Xa);

if Banana(Xc)<Banana(Xn)

P(:,I) = Xc;

break;

else

Xn=Xc;

end

end

end

k=k+1;

Set(:,k+3) = P(:,I);

end

%% Visualization

X = -2.5: 0.05 :2.5;

Y = -1.4: 0.05 : 6;

[X, Y] = meshgrid(X, Y);

v = 100\*(Y-X.^2).^2 +(1-X).^2;

figure(1)

L = [0,0.1,0.25,0.5,1,5,10,30,70,110,160,200,300,600];

contour(X,Y,v,L,'LineWidth',1,'LineColor','#0072BD','ShowText','on');

hold on;

plot(Set(1,:),Set(2,:),'r-.','LineWidth',1.5);

xlabel('x'); ylabel('y');

title('Simplex Method');

legend('Level sets','Behavior of the Simplex Method');

figure(2)

Xaxis=(0:k);

plot(Xaxis(1,:),Jk(1,:));

xlabel('Iteration number: k'); ylabel('Jk'); title('Cost Function of Simplex Method');