《最优化算法》第二次作业

***Assignment 8.26：***

求解***Rosenbrock***函数的极小点：

Note:初始点终止条件为

**Matlab代码如下：**

function A\_8\_26\_2()

clear;clc;

epsilon\_x =10^(-6);

epsilon\_g =10^(-4);

max\_iter=10000;

xnew=[-2,2];

for k = 1:max\_iter

xcurr=xnew;

g\_curr=grad(xcurr);

if norm(g\_curr) <= epsilon\_g

disp('Terminating: Norm of gradient less than');

disp(epsilon\_g);

k=k-1;

break;

end %if

alpha=linesearch\_secant(xcurr,-g\_curr);

xnew = xcurr-alpha.\*g\_curr;

if norm(xnew-xcurr) <= epsilon\_x\*norm(xcurr)

disp('Terminating: Norm of difference between iterates less than');

disp(epsilon\_x);

break;

end %if

if k == max\_iter

disp('Terminating with maximum number of iterations');

end %if

end %for

disp('Final point =');

disp(xnew');

disp('Number of iterations =');

disp(k);

%------------------------------------------------------------------

%--------------利用割线发迭代alpha value-------------------

function alpha=linesearch\_secant(x,d)

%Line search using secant method

epsilon=10^(-4); %line search tolerance

max = 100; %maximum number of iterations

alpha\_curr=0;

alpha=0.001;

dphi\_zero=d\*grad(x)';

dphi\_curr=dphi\_zero;

i=0;

while abs(dphi\_curr)>epsilon\*abs(dphi\_zero),

alpha\_old=alpha\_curr;

alpha\_curr=alpha;

dphi\_old=dphi\_curr;

dphi\_curr=d\*grad(x+alpha\_curr.\*d)';

alpha=(dphi\_curr\*alpha\_old-dphi\_old\*alpha\_curr)/(dphi\_curr-dphi\_old);

i=i+1;

if (i >= max)&&(abs(dphi\_curr)>epsilon\*abs(dphi\_zero)),

disp('Line search terminating with number of iterations:');

disp(i);

break;

end

end %while

end

%————定义目标函数————

% function y=fun(x)

% y = 100\* (x(2)-x(1))^2 + (1-x(1))^2;

% end

%————定义目标函数的一阶导数————

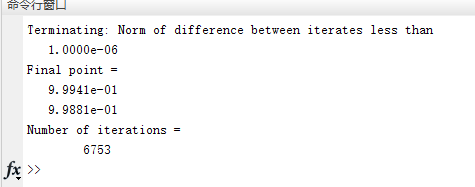
function y=grad(x)

y =[100\*(4\*x(1)^3-4\*x(1)\*x(2))+2\*x(1)-2,200\*(x(2)-x(1)^2)];

end

end

运行结果：



可以看到在迭代6753次之后接近目标极小点：

***Assignment：Correction Example\_4\_2.m***

1. 运行原程序：

% 牛顿法：基于精确一维搜索的步长

function example\_4\_2()

%————初始点————

x=[1.6,1.5];

k=0;

print\_index=0; % 程序结束后，输出文字的指示数

%————停止规则的参数————

f\_current = fun(x); % 计算当前函数值

f\_change = f\_current; % 初始化差值

%————梯度下降法————

while f\_change>0.000000001 %设置条件，两次计算的值之差小于某个数，跳出循环

k=k+1;

F=hfun(x);

if min(eig(F))<0

print\_index=1;

fprintf('黑塞矩阵不正定\n')

break

end

%det(F)

if det(F)<0.0001

print\_index=1;

fprintf('黑塞矩阵奇异\n')

break

end

iF=inv(hfun(x)); %黑塞矩阵的逆矩阵

gfun(x)

dk=-[dot(iF(1,:),gfun(x)),dot(iF(2,:),gfun(x))]; %牛顿法的搜索方向

alpha(k)=exactminfun(x,dk); % 下降步长

x = x + alpha(k).\*dk; % 修正的牛顿法迭代公式

f\_new = fun(x); % 计算下一个迭代点的函数值

f\_change = f\_current - f\_new; % 计算两次函数值之差

if f\_change<0

print\_index=1;

fprintf('目标函数不再下降\n')

break

end

f\_current = f\_new; % 赋给当前的函数值

plot(x(1),x(2),'ro','markersize',7) % 标记当前的位置

drawnow;pause(0.2);

x\_record(k,:)=x;

end

plot(x\_record(:,1),x\_record(:,2),'m')

hold off

if print\_index==0

fprintf('在迭代%d次后找到函数最小值为%e，对应的x值为[%e,%e].\n',k,f\_current,x(1),x(2))

%alpha

end

%————定义目标函数————

function y=fun(x)

y = (x(1)-1)^2 + 4\* (x(2)-x(1)^2)^2;

end

%————定义目标函数的一阶导数————

function y=gfun(x)

y =[2\*x(1)-2-16\*(x(2)-x(1)^2)\*x(1), 8\*(x(2)-x(1)^2)];

end

%————定义目标函数的二阶导数————

function y=hfun(x)

y =[2-16\*x(2)+48\*x(1)^2, -16\*x(1);...

-16\*x(1), 8 ];

end

%————定义一维函数————

function y=phi(alpha,x,dk)

xx(1)=x(1)+ alpha\*dk(1);

xx(2)=x(2)+ alpha\*dk(2);

y = (xx(1)-1)^2 + 4\* (xx(2)-xx(1)^2)^2;

end

%————定义一维函数的一阶导数————

function y=gphi(alpha,x,dk)

xx(1)=x(1)+ alpha\*dk(1);

xx(2)=x(2)+ alpha\*dk(2);

y=(2\*xx(1)-2-16\*(xx(2)-xx(1)^2)\*xx(1))\*dk(1) + (8\*(xx(2)-xx(1)^2))\*dk(2);

end

%————定义一维函数的二阶导数————

function y=hphi(alpha,x,dk)

xx(1)=x(1)+ alpha\*dk(1);

xx(2)=x(2)+ alpha\*dk(2);

y = (2-16\*xx(2)+48\*xx(1)^2)\*dk(1)\*dk(1)-2\*16\*xx(1)\*dk(1)\*dk(2)+8\*dk(2)\*dk(2);

end

%————精确一维搜索:牛顿法————

function a=exactminfun(x,dk)

m=0;

alpha0=1; %初始 步长

current=phi(alpha0,x,dk);

x= x + alpha0.\*dk;

change= current;

while change>0.000001

alpha0=alpha0-gphi(alpha0,x,dk)/hphi(alpha0,x,dk);

x= x + alpha0.\*dk;

new=phi(alpha0,x,dk);

change = current - new;

current = new;

if change<0

break

end

m=m+1;

if m>200

break;

end

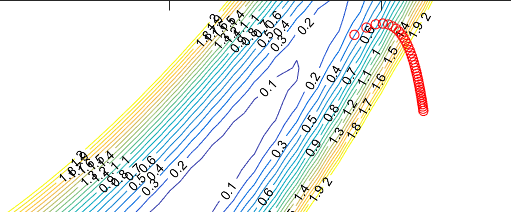
end

a=alpha0;

end

end

运行程序之后发现：



程序运行一段时间后发现迭代点的位置几乎没有发生变化。

1. 对程序中的关键参数进行中间输出查看，程序到底是什么问题：

dk=-[dot(iF(1,:),gfun(x)),dot(iF(2,:),gfun(x))]; %牛顿法的搜索方向

alpha(k)=exactminfun(x,dk); % 下降步长

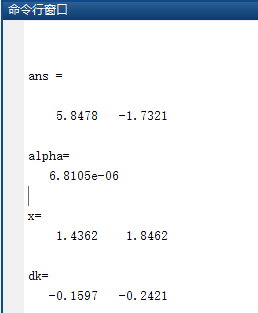
x = x + alpha(k).\*dk; % 修正的牛顿法迭代公式

disp(alpha(k));

disp(x);

disp(dk);

由于Newton法的核心参数任然在于：的选取上，因此将运行中间过程的进行输出查看；可以看到：



在迭代几乎不动的地方，很小，而从Newton法的原理上：

***.***

因此，此时由于很小，已经不符合Newton法的条件，所以此时的迭代是没有任何意义的。

1. **Correction**

if alpha(k)>=correction

x = x + alpha(k).\*dk; % 修正的牛顿法迭代公式

else

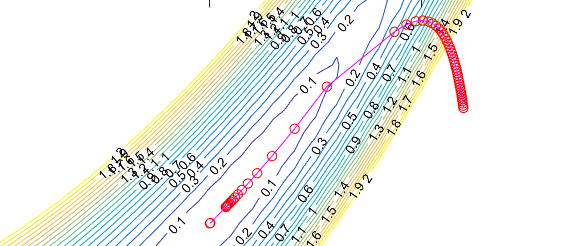
x = x + dk;

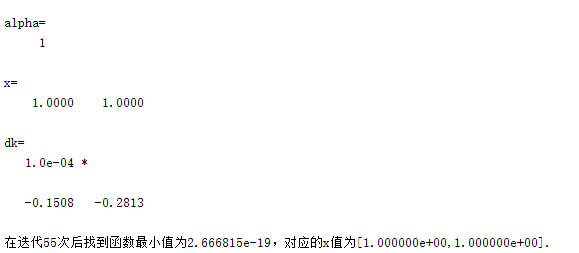
end %if

对的迭代过程进行修正，当很小时直接将迭代式改为：

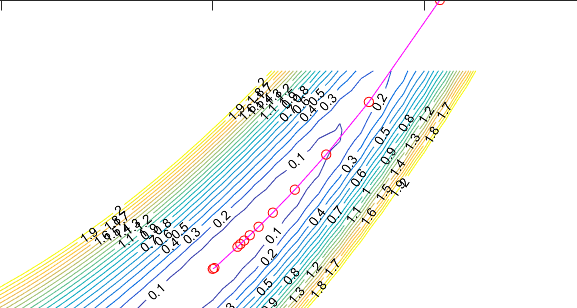
下面是Correction取不同值的情况：

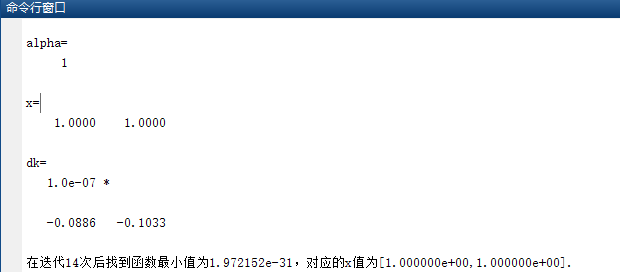
1. When Correction=0.01:



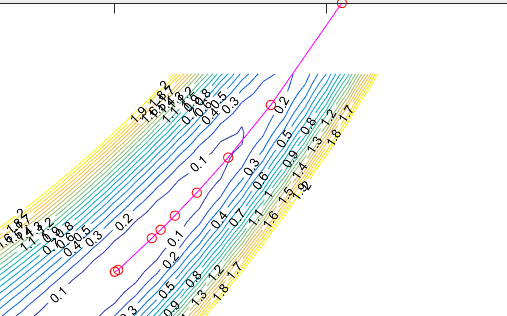


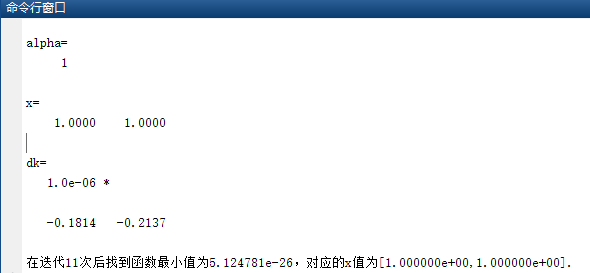
1. 继续调整：Correction=0.1



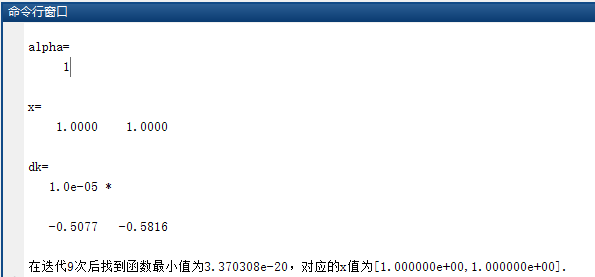


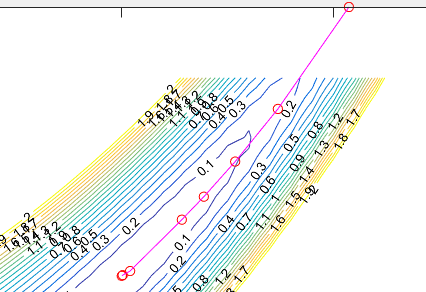
1. 继续调整：Correction=0.2



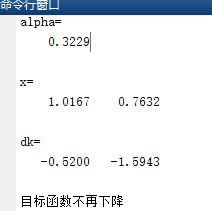


1. 继续调整：Correction=0.3





1. 继续调整：Correction=0.4



可以看到此时明显的效果。

====================================================

总结：因此在使用Newton法进行迭代的时候，应该关注Newton法的使用条件：***.***