成绩	

中国矿业大学数学学院

实验报告

课程名称： 实用优化算法

实验名称： 实验三：外罚函数法

姓名学号： 胡钧耀　06192081

实验时间： 2021 年 10 月 31 日

《实用优化算法》实验报告

实验名称：实验三（外罚函数法算法）

1 实验目的

复习理论课学习的外罚函数法，通过上机利用 MATLAB 数学软件进行外罚函数编程，结合前面实验学习的黄金分割法和梯度下降法，对具体问题具体分析，培养 MATLAB 编程与上机调试能力，加强计算机与数学学科联系。

2 实验内容

$$\begin{cases} \min\ f_1(x)=(x_1-2)^4+(x_1-2x_2)^2 \\ s.t.\ x_1^2-x_2=0 \end{cases} \tag{1}$$

$$\begin{cases} \min\ f_2(x)=\dfrac{3}{2}x_1^2+x_2^2+\dfrac{1}{2}x_3^2-x_1x_2-x_2x_3+x_1+x_2+x_3 \\ s.t.\ x_1+2x_2+x_3-4=0 \end{cases} \tag{2}$$

3 算法设计

设计如下算法程序结构：

Algorithm 1 外罚函数法（等式约束）

初始化：选取初始点x_0，初始罚因子σ_1，精度$\epsilon_1>0$, $\epsilon>0$.

计算罚函数$P_0=P(x_0,\ \sigma_0=1)$及梯度$\nabla P_0=\nabla P(x_0,\ \sigma_0)$.

令$f_{stop}=0$, $k=1$.

while $f_{stop}=0$ do

以x_{k-1}为初始点，

$$\|\nabla P(x,\ \sigma_{k-1})\|\le\epsilon_1$$

为终止条件，使用共轭梯度法求解问题

$$\min\ P(x,\ \sigma_{k-1}).$$

设解为$x_k=x_k(\sigma_k)$.

if $\|c(x_k)\|\le\epsilon$ then

令$f_{stop}=1$;

else

令$\sigma_{k+1}=10\sigma_k$, $k=k+1$.

end if

end while

输出：x_k.

设计如下程序文件结构：

exterior_penalty.m 实现本次实验外罚函数法主程序。函数输入为：需要求解的函数，需要求解的函数的梯度，罚函数，发函数的梯度，起始点坐标。函数输出为：该函数最小值对应自变量值，该函数最小值。

Conjugate_Gradient.m 是根据实验 2 的共轭梯度算法进行改进后的算法。函数输入为：需要求解的函数，需要求解的函数的梯度，罚函数，罚函数的梯度，罚函数参数，起始点坐标。函数输出为：该函数最小值对应自变量值。

golden_section.m 是根据实验 1 的黄金分割算法进行改进后的算法，便于计算出每一次一维搜索的步长。函数输入为：需要求解的函数，需要求解的函数的梯度，罚函数参数，起始点坐标，步长。函数输出为：共轭下降法中步长参数。

f1.m 保存问题 1 函数，f2.m 保存问题 2 函数。

st_f1.m 保存问题 1 函数的条件，st_f2.m 保存问题 2 函数的条件。

grad_f1.m 保存问题 1 函数的梯度，grad_f2.m 保存问题 2 函数的梯度。

grad_st_f1.m 保存问题 1 条件梯度，grad_st_f2.m 保存问题 2 条件梯度。

p.m 保存一维搜索的搜索步骤。

funp.m 保存计算罚函数的计算步骤。

grad_p.m 保存计算罚函数梯度的计算步骤。

根据参考文献学习[1]，可设置将函数以及对应函数梯度作为参数直接放入算法函数进行计算。函数输入为：需要求解的函数，需要求解的函数的梯度，罚函数，发函数的梯度，起始点坐标。函数输出为：该函数最小值对应自变量值，该函数最小值。

4 程序代码

exterior_penalty.m

```
function [final_x, final_y]
    = exterior_penalty(funf, gradf, func, gradc, x)

epsilon = 10e-5;
sigma = 1;
k = 1;
stop = 0;

while stop == 0
    x = Conjugate_Gradient
        (funf, gradf, func, gradc, sigma, x);
    y = norm(func(x));
    if norm(func(x)) <= epsilon
        stop = 1;
    else
        sigma = 10 * sigma;
        k = k + 1;
    end
end

final_x = x;
final_y = funf(x);
```

Conjugate_Gradient.m

```
function x_star = Conjugate_Gradient
        (funf, gradf, func, gradc, sigma, x_old)
epsilon_1 = 1e-7;
n = length(x_old);
beta = 0;
d = 0;
kk = 0;

f = funp(funf, func, x_old, sigma);
g = grad_p(gradf, func, gradc, x_old, sigma);

if norm(g) <= epsilon_1
    run = 0;
else
    run = 1;
end
while run == 1 && kk < 10000
    d = -g+beta*d;
    alpha = golden_section(funf, func, sigma, x_old, d);
    x_new = x_old + alpha*d;
    g = grad_p(gradf, func, gradc, x_new, sigma);
    if norm(g) <= epsilon_1
        run = 0;
    else
        if mod(kk,n+1) == 0
            beta = 0;
        else
            beta =
        norm(grad_p(gradf, func, gradc, x_new, sigma))^2
        /norm(grad_p(gradf, func, gradc, x_old, sigma))^2;
        end
        kk = kk + 1;
        x_old = x_new;
    end
end
x_star = x_old;
```

golden_section.m

```
function final = golden_section(funf, func, sigma, x, d)
epsilon = 1e-7;
a = 0;
b = 2;
alp1 = a + 0.382*(b-a);
alp2 = a + 0.618*(b-a);
while abs(b-a) > epsilon
    p1 = p(funf, func, sigma, alp1,x,d);
    p2 = p(funf, func, sigma, alp2,x,d);
    if p1 >= p2
        a = alp1;
        alp1 = alp2;
        alp2 = a + 0.618*(b-a);
    else
        b = alp2;
        alp2 = alp1;
        alp1 = a + 0.382*(b-a);
    end
end
final = (a+b) / 2;
```

f1.m

```
function f= f1(x)
f = (x(1)-2)^4+(x(1)-2*x(2))^2;
end
```

f2.m

```
function f= f2(x)
f = 1.5*x(1)^2+x(2)^2+0.5*x(3)^2
    -x(1)*x(2)-x(2)*x(3)+x(1)+x(2)+x(3);
end
```

st_f1.m

```
function cx = st_f1(x)
cx1 = x(1)^2-x(2);
cx = [cx1];
end
```

st_f2.m

```
function cx = st_f2(x)
cx1 = x(1)+2*x(2)+x(3)-4;
cx = [cx1];
end
```

grad_f1.m

```
function gf = grad_f1(x)
gf = [2*x(1) - 4*x(2) + 4*(x(1) - 2)^3;
    8*x(2) - 4*x(1)];
end
```

grad_f2.m

```
function gf = grad_f2(x)
gf = [3*x(1) - x(2) + 1;
    2*x(2) - x(1) - x(3) + 1;
    x(3) - x(2) + 1];
end
```

grad_st_f1.m

```
function gcx = grad_st_f1(x)
gcx1 = [2*x(1); -1];
gcx = [gcx1];
end
```

grad_st_f2.m

```
function gcx = grad_st_f2(x)
gcx1 = [1;2;1];
gcx = [gcx1];
end
```

funp.m

```
function px = funp (funf, func, x, sigma)
fx=funf(x);
cx=func(x);
px=fx+sigma*cx'*cx;
end
```

p.m

```
function px = p(funf, func, sigma, alp, x, d)
px = funp(funf, func,x+alp*d,sigma);
end
```

grad_p.m

```
function gpx = grad_p(gradf, func, gradc, x, sigma)
gfx=gradf(x);
cx=func(x);
gcx=gradc(x);
gpx=gfx+2*sigma*gcx*cx;
end
```

5 运行结果

命令行输入

```
>>funf=@f1;gradf=@grad_f1;func=@st_f1;gradc=@grad_st_f1;
x=[0;0];
>>[final_x,final_y] = exterior_penalty(funf, gradf, func,
gradc, x)

final_x =
    0.9456
    0.8941
final_y =
    1.9461

>>funf=@f2;gradf=@grad_f2;func=@st_f2;gradc=@grad_st_f2;
x=[0;0;0];
>>[final_x,final_y] = exterior_penalty(funf, gradf, func,
gradc, x)

final_x =
    0.3889
    1.2222
    1.1667
final_y =
    3.2777
```

6 结果分析

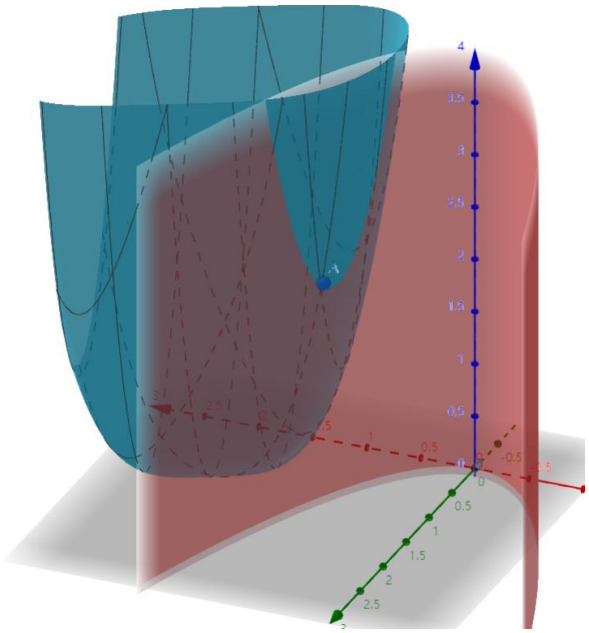

图 1 GeoGebrea 绘图验证结果（问题一）

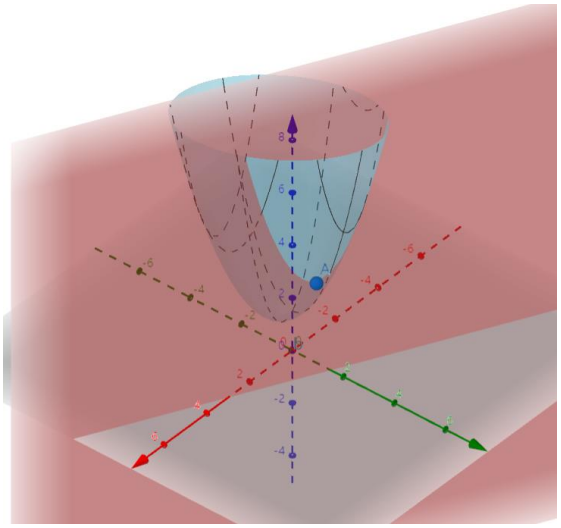

图 2 GeoGebrea 绘图验证结果（问题二）

对计算结果进行分析，与绘图观察直接得到的结果十分相近（蓝色为函数无约束全局图形，与红色曲面的交线上的点是满足原函数约束条件的点，其最小值点与计算结果接近）。对于问题二，由于有三个变量，第三个变量z设为a，每次切换a的大小作图，观察z值（蓝色为函数无约束全局图形，与红色曲面的交线上的点是满足原函数约束条件的点，x, y不变时，$a=1.1$左右有最小值，其最小值点与计算结果接近）有两个极小值，因此需要选取其中的最小值作为因此可以认为全局极小值，求解得到两个数值相等。

$$\min\ f_1 = f_1(0.9456, 0.8941) = 1.9461 \tag{3}$$

$$\min\ f_2 = f_2(0.3889, 1.2222, 1.6667) = 3.2777 \tag{4}$$

7 参考文献

[1] Matlab 中如何将（自定义）函数作为参数传递给另一个函数.VVingerfly.博客园.https://www.cnblogs.com/VVingerfly/p/4793131.html