|  |  |
| --- | --- |
| **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。

1. 实验内容

|  |  |
| --- | --- |
|  | () |
|  | () |

1. 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*，*初始罚因子, 精度.  计算罚函数及梯度.  令.  while  do  以为初始点,    为终止条件, 使用共轭梯度法求解问题  .  设解为.  if  then  令;  else  令.  end if  end while  输出：. |

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。

1. 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 |

1. 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** |

1. 结果分析

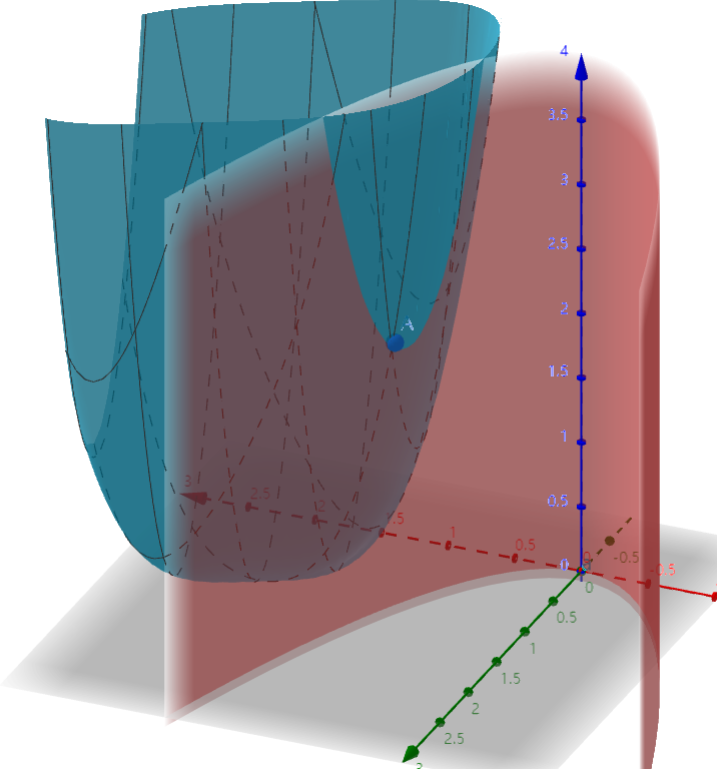


图 1 GeoGebrea绘图验证结果（问题一）

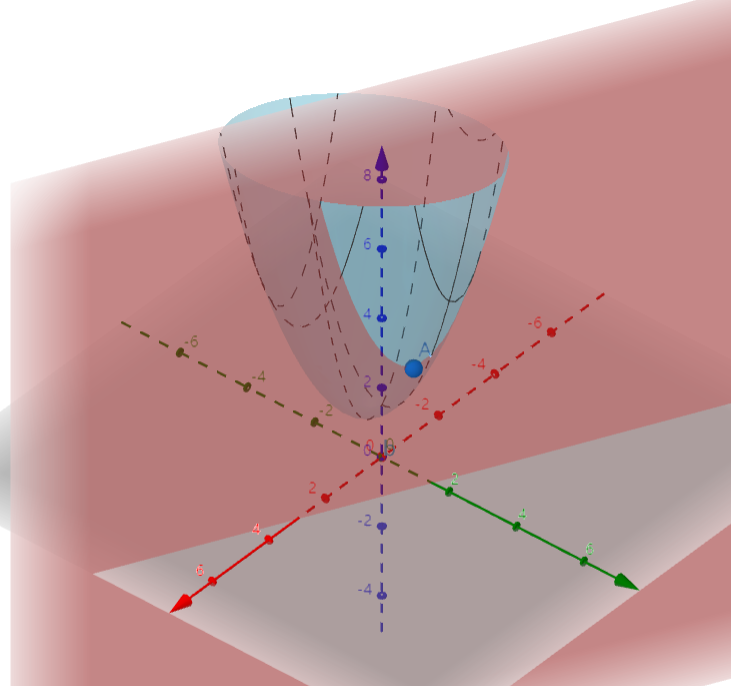


图 2 GeoGebrea绘图验证结果（问题二）

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，第三个变量设为，每次切换的大小作图，观察值（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，不变时，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。

|  |  |
| --- | --- |
|  | () |
|  | () |

1. 参考文献
2. 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