**Homework #3**

**孙仲煜 & 3119307082**

**Part 1:Linear constraint**

简约梯度法(reduce conjugate method)处理带有线性约束的非线性规划问题的标准形式如下：

其中，为相应约束的右端项。

由于问题具有与标准线性规划问题相同形式的约束，一个自然的想法就是利用单纯形法的思想来处理线性约束优化问题中的约束。在这里我们设行满秩。

将一个可行解的个最大的正分量定义为基变量，其余的个变量定义为非基变量，即任选的一个基，记，将对应划分为，即和分别由基变量和非基变量组成。从而目标函数可写为：

从而将原问题等价地转化为下列仅带有变量非负约束的极小化问题：

利用复合函数的求导法则可得的梯度，因为是关于简化后的个变量的梯度，故称它为的简约梯度。

下面给出简约梯度法的迭代步骤：

|  |
| --- |
| 简约梯度法 |
| 1. 给定初始基本可行解，将其剖分成，其中为基变量，令. 2. 对应于，将分解成。由复合函数求导法则求得，再分别确定和，令. 3. 若，则停止迭代，输出点；否则，求使得   令.   1. 若，则基向量不变，令，转2。若有某个使，则将换出基，而将中具有最大分量的变量换入基，由此构成新的基向量与非基变量，令，转2。 |

**Section 1: Problem description**

现有二次线性约束优化问题如下：

s.t.

其中*) , ,*

要求解这类带等式约束的二次型优化问题，我们可以用单纯形法将变量分为基变量与非基变量，然后通过等式约束将基变量用非基变量表示并回代入目标函数，达到降维的作用，从而将线性约束优化问题转化为无约束优化问题，然后再使用梯度下降法或牛顿法求解问题。

下面我们选取为基变量，为非基变量，由等式约束易知：

将上述等式回代入目标函数可得降维后的无约束优化问题如下：

其中*)*, ,

下面我们只需求解该无约束优化问题的极小值点即可。

**Section 2: Solution & Programming**

下面我们采用最速下降法求解该无约束优化问题，步骤如下：

1) 令*k = 0*, 取初始点*,* 迭代精度*,* 根据下述方式进行迭代：

2)

3)

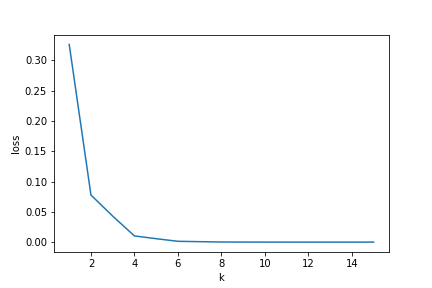
4)

5)

6) 迭代停止条件：< 或 *k+1 = n*, 则停止迭代；否则继续进行如上步骤。

经过15次迭代后，目标函数收敛。

下图为损失函数图，其中横坐标为迭代次数k, 纵坐标loss为。



综上所述，经过15次迭代，目标函数在下述点*x*可取到极小值：

即, 时目标函数取得极小值，则

，

即原目标函数在极小值点：

取得局部极小值:

**Part 2:Nonlinear constraint**

在求解无约束优化问题时，只需保证目标函数不断下降并能最终达到最优解即可。但对于非线性约束优化问题时，仅仅使目标函数下降时不够的，我们还应使迭代点列满足或逐渐满足各个约束条件。因此，要想用某个五约束问题替代原来的约束问题，该无约束问题的目标函数必须是约束问题的目标函数与约束函数的某种组合。传统的罚函数法就是通过给原来的目标函数加上一项由约束函数所构成的惩罚项来生成新的目标函数，以便将约束优化问题转化为无约束优化问题的求解。惩罚项的构造原理是：每当某个点不可行时，就要对其处以惩罚，且惩罚值将随着该点不可行性的提高而增大；但当某点为可行点时，则不做任何惩罚。惩罚项的作用就是随着迭代的进行，迫使迭代点不断逼近并最终位于可行域内，以便找到原约束问题的最优解。构造不同的惩罚项，就产生了许多不同形式的罚函数，并由此导出了可用于求解不同特点约束优化问题的多种罚函数法。

对一般的非线性约束最优化问题，在任一点处，等式约束的违反程度可用来度量，不等式约束的违反程度可用来衡量。依照惩罚项的构造原理，显然可取下列函数作为惩罚项：

这里，为选定的常数。为使具有良好的性质，如连续可微性，通常取，这时连续可微，以下不妨假定。由此，可以定义上述非线性约束优化问题的罚函数为:

这里为惩罚因子。

通过上式我们可以观察到，只要选取较大的，就可以通过求解一个无约束优化问题来寻求最优解。然而，在实际计算中，值的确定往往比较困难，所以我们往往是选取一个单调增的罚参数序列，通过求解一系列无约束优化问题来寻求原问题的最优解。因此，罚函数法的迭代步骤如下：

|  |
| --- |
| 罚函数法 |
| 1) 任选初始点，初始罚因子及精度参数，令.  2) 以为初始点，求罚函数的无约束极小点，记其解为。  3) 若，则取为原约束问题的近似最优解，停止迭代；否则，令，，转2. |

由惩罚项的特点，当趋于无穷时，随着的不断增大，对每个不可行点的惩罚也不断增大并趋向于无穷。因此，在对应于的无约束极小化问题的最优解处，的值应不断减小，从而保证逐步趋于可行并最终达到问题的最优解。

**Section 1: Problem description**

现有二次非线性约束优化问题如下：

其中*) , ,*

求解这类非线性约束优化问题的步骤往往分为两步，第一步是对原约束问题进行简化，观察是否能够将其转化为无约束优化问题；第二步则是对进行简化后的问题选择合适的方法进行求解。通过观察上述非线性等式约束条件，我们发现可通过换元的方法对原目标函数进行降维，再利用最速下降法对新目标函数求极小值。

由等式约束易知：

由此可构造惩罚项如下：

因此上述非线性约束优化问题可用如下罚函数代替：

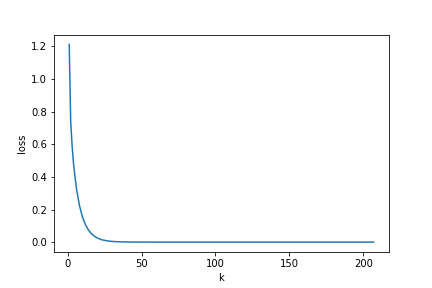
**Section 2: Solution & Programming**

对于罚函数, 用最速下降法求每一个所对应的无约束优化问题的极小值，迭代格式为：

其中为负梯度方向，即最速下降方向，为搜索步长。一般情况下，最优步长的确定要用到线性搜索技术，比如精确线性搜索，但在本次问题中所需求解的目标函数并不适用该搜索，接下来我们使用Goldstein不精确线性搜索来确定步长。

经过207次迭代后，目标函数收敛。

下图为损失函数图，其中横坐标为迭代次数k, 纵坐标loss为。



综上所述，当 时，经过207次迭代，目标函数在下述点*x*可取到极小值：

即 时目标函数取得极小值，即原目标函数在极小值点：

处取得局部极小值:

**附录**

1. #Goldstein不确定搜索函数#
2. **import** numpy as np
3. **import** random
5. **def** goldsteinsearch(f,df,d,x,alpham,rho,t):
7. flag=0
9. a=0
10. b=alpham
11. fk=f(x)
12. gk=df(x)
14. phi0=fk
15. dphi0=np.dot(gk,d)
17. alpha=b\*random.uniform(0,1)
19. **while**(flag==0):
20. newfk=f(x+alpha\*d)
21. phi=newfk
22. **if**(phi-phi0<=rho\*alpha\*dphi0):
23. **if**(phi-phi0>=(1-rho)\*alpha\*dphi0):
24. flag=1
25. **else**:
26. a=alpha
27. b=b
28. **if**(b<alpham):
29. alpha=(a+b)/2
30. **else**:
31. alpha=t\*alpha
32. **else**:
33. a=a
34. b=alpha
35. alpha=(a+b)/2
36. **return** alpha
38. #最速下降法#
39. **import** numpy as np
40. **import** matplotlib.pyplot as plt
41. **import** random
42. **import** linesearch
43. **from** linesearch **import**  goldsteinsearch
45. **def** rosenbrock(x):
46. **return**2\*((1+x3)\*\*2)\*\*2 + x2\*\*2 + 3\*x3\*\*2 + 2\*x2\*(1+x3)\*\*2+x3\*(1+x3)\*\*2+x2\*x3
48. **def** jacobian(x):
49. **return** np.array([-36\*x2\*(x3-x2\*\*2)-22\*(1-x2),13\*(x3-x2\*\*2)])
50. **def** steepest(x0):
52. **print**('初始点为:')
53. **print**(x0,'\n')
54. imax = 20000
55. W=np.zeros((2,imax))
56. W[:,0] = x0
57. i = 1
58. x = x0
59. grad = jacobian(x)
60. delta = sum(grad\*\*2)  # 初始误差

63. **while** i<imax **and** delta>10\*\*(-5):
64. p = -jacobian(x)
65. x0=x
66. alpha = goldsteinsearch(rosenbrock,jacobian,p,x,1,0.1,2)
67. x = x + alpha\*p
68. W[:,i] = x
69. grad = jacobian(x)
70. delta = sum(grad\*\*2)
71. i=i+1
73. **print**("迭代次数为:",i)
74. **print**("近似最优解为:")
75. **print**(x,'\n')
76. **return** W