**Homework #3**

**孙仲煜 & 3119307082**

**Part 1:Linear constraint**

**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)

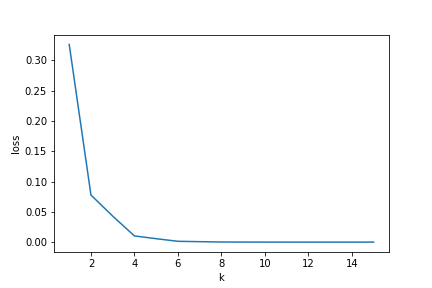
4)

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**

**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：

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：

下面我们只需求解该二元无约束优化问题的极小值点即可。

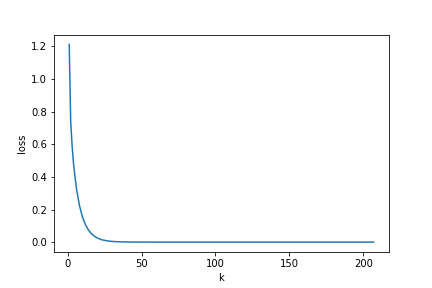
**Section 2: Solution & Programming**

对于一个二元函数, 用最速下降法求其极小值迭代格式为：

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，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