**Homework #2**

**孙仲煜 & 3119307082**

**Section 1: Problem description**

现有二次型如下:

其中=, ,

.

接下来我将分别使用共轭梯度法、最速下降法以及牛顿迭代法求解上述目标函数极小值。

**Section 2: Solution & Programming**

Method 1: Conjugate Gradient Method

解：

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)

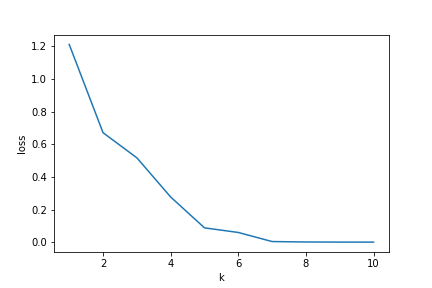
6) 迭代停止条件：< 或 *k+1 = n*, 则停止迭代；否则继续进行如下步骤。

7)

8)

经过10次迭代后，目标函数值收敛。

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，目标函数经过10次迭代后可得极小值点:

]

极小值如下：

Method 2: Steepest Descent Method

解：

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

2)

3)

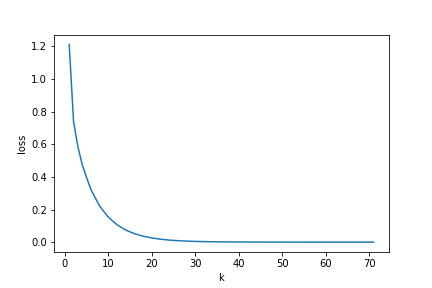
4)

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71次迭代后，目标函数值收敛。

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，目标函数经过71次迭代后可得极小值点:

[[-0.31704858]

[ 0.57577109]

[-0.13709404]

[ 0.11684237]

[ 0.10757146]

[ 0.14610353]

[-0.18187607]

[ 0.86721981]

[ 0.67423084]

[-0.24412767]]

极小值如下：

Method 3: Newton Method

解：

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

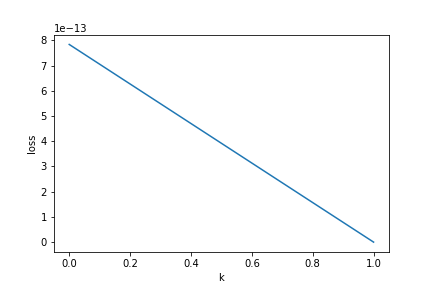
2)

3)

4)

5) 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。

由于这里选取的矩阵*A*为对称正定矩阵，因此*f(x)*的海森矩阵即为矩阵*A*.经过一次迭代即可得到函数极小值点，如下图所示，其中横坐标为迭代次数k, 纵坐标loss为.



综上所述，经过一次迭代后可得极小值点:

[[-0.31704918]

[ 0.57577052]

[-0.13709436]

[ 0.11684474]

[ 0.10757077]

[ 0.1461028 ]

[-0.18187742]

[ 0.86721983]

[ 0.67423075]

[-0.24412747]]

极小值如下：

**附录**

1. #共轭梯度法
2. **import** numpy as np
3. **import** matplotlib.pyplot as plt

6. **def** ConjugateGradient(A, x, b):
7. **global** k\_all, x\_all
9. k = 0
10. r = b - np.dot(A,x)
11. d = r
12. k\_all = []
13. x\_all = []
14. **while** True:
15. k+=1
16. alpha = np.linalg.norm(r)\*\*2 / np.vdot(np.dot(A,d),d)
17. x = x + alpha \* d
18. old\_r = r
19. r = b - np.dot(A,x)
20. f = 1/2 \* np.dot(np.dot(x.T,A),x) - np.dot(b.T, x)
21. **print**("第",k,"次迭代:",f)
22. k\_all.append(k)
23. x\_all.append(x)
25. **if** np.linalg.norm(r) < 1e-6:
26. **print**("conjugate gradient finished!")
27. **return** x
29. **else**:
30. beta = np.linalg.norm(r)\*\*2/np.linalg.norm(old\_r)\*\*2
31. d = r + beta \* d

34. A = np.array([[ 56,   5,  -2,  12,   9,  -5, -10,  11,   1, -12],
35. [  5,  58,  -1,  10,   3, -11,  -4, -16, -17,   5],
36. [ -2,  -1,  67,   9,  10,  -4,  -6,   6,   6,  -1],
37. [ 12,  10,   9,  37,  22,   9,  15,  -1,   1, -9],
38. [  9,   3,  10,  22,  64,  -8,  -4, -11,   3,   0],
39. [ -5, -11,  -4,   9,  -8,  62,   4,   9,  -5,  -1],
40. [-10,  -4,  -6,  15,  -4,   4,  43,   4,   3,  -3],
41. [ 11, -16,   6,  -1, -11,   9,   4,  43, -16,  -3],
42. [  1, -17,   6,   1,   3,  -5,   3, -16,  61,   2],
43. [-12,   5,  -1,  -9,   0,  -1,  -3,  -3,   2,  57]])
45. x = np.array([[1,1,1,1,1,1,1,1,1,1]]).T
46. b = np.array([[2, 6, 3, 8, -1, 9, 2, 13, 15,-9]]).T
47. ConjugateGradient(A, x, b)
49. loss\_all = []
51. **for** x **in** x\_all:
52. loss = np.linalg.norm(x - x\_all[-1])
53. loss\_all.append(loss)
55. plt.plot(k\_all, loss\_all)
56. plt.xlabel("k")
57. plt.ylabel("loss")
58. plt.savefig('ConjugateGradient.png')
60. #最速下降法
61. **import** numpy as np
62. **import** matplotlib.pyplot as plt
64. **def** SteepestDescent(A, x, b):
65. **global** x\_all, k\_all
66. x\_all = []
67. k\_all = []
68. k = 0
69. gradient = np.dot(A, x) - b
71. **while** True:
72. k +=1
73. alpha = np.dot(gradient.T, gradient) / np.dot(np.dot(gradient.T, A), gradient)
74. old\_x = x
75. x = old\_x - alpha \* gradient
76. gradient = np.dot(A, x) - b
77. f = 1/2 \* np.dot(np.dot(x.T,A),x) - np.dot(b.T, x)
78. **print**("第",k,"次迭代:",f)
79. x\_all.append(x)
80. k\_all.append(k)
81. **if** np.linalg.norm(x - old\_x) <1e-6:
82. **print**("Gradient Descent Fishend!")
83. **return** x
85. A = np.array([[ 56,   5,  -2,  12,   9,  -5, -10,  11,   1, -12],
86. [  5,  58,  -1,  10,   3, -11,  -4, -16, -17,   5],
87. [ -2,  -1,  67,   9,  10,  -4,  -6,   6,   6,  -1],
88. [ 12,  10,   9,  37,  22,   9,  15,  -1,   1, -9],
89. [  9,   3,  10,  22,  64,  -8,  -4, -11,   3,   0],
90. [ -5, -11,  -4,   9,  -8,  62,   4,   9,  -5,  -1],
91. [-10,  -4,  -6,  15,  -4,   4,  43,   4,   3,  -3],
92. [ 11, -16,   6,  -1, -11,   9,   4,  43, -16,  -3],
93. [  1, -17,   6,   1,   3,  -5,   3, -16,  61,   2],
94. [-12,   5,  -1,  -9,   0,  -1,  -3,  -3,   2,  57]])
96. x = np.array([[1,1,1,1,1,1,1,1,1,1]]).T
97. b = np.array([[2, 6, 3, 8, -1, 9, 2, 13, 15,-9]]).T
98. SteepestDescent(A, x, b)
100. loss\_all = []
102. **for** x **in** x\_all:
103. loss = np.linalg.norm(x - x\_all[-1])
104. loss\_all.append(loss)
106. plt.plot(k\_all, loss\_all)
107. plt.xlabel("k")
108. plt.ylabel("loss")
109. plt.savefig('SteepestDescent.png')
111. #牛顿法
112. **import** numpy as np
113. **from** sympy **import** \*
114. **import** matplotlib.pyplot as plt
116. **def** Newtown(A, x, b):
117. **global** x\_all, k\_all
118. x\_all = []
119. k\_all = []
120. k = -1
121. gradient =  np.dot(A, x) - b
123. **while** True:
124. k+=1
125. old\_x = x
126. x = old\_x - np.dot(np.linalg.inv(A) , gradient)
127. gradient =  np.dot(A, x) - b
128. f = 1/2 \* np.dot(np.dot(x.T,A),x) - np.dot(b.T, x)
129. **print**("第",k,"次迭代:",x,f)
130. x\_all.append(x)
131. k\_all.append(k)
132. **if** np.linalg.norm(x - old\_x) < 1e-6:
133. **print**("Gradient Descent Fishend!")
134. **return** x,f

137. A = np.array([[ 56,   5,  -2,  12,   9,  -5, -10,  11,   1, -12],
138. [  5,  58,  -1,  10,   3, -11,  -4, -16, -17,   5],
139. [ -2,  -1,  67,   9,  10,  -4,  -6,   6,   6,  -1],
140. [ 12,  10,   9,  37,  22,   9,  15,  -1,   1, -9],
141. [  9,   3,  10,  22,  64,  -8,  -4, -11,   3,   0],
142. [ -5, -11,  -4,   9,  -8,  62,   4,   9,  -5,  -1],
143. [-10,  -4,  -6,  15,  -4,   4,  43,   4,   3,  -3],
144. [ 11, -16,   6,  -1, -11,   9,   4,  43, -16,  -3],
145. [  1, -17,   6,   1,   3,  -5,   3, -16,  61,   2],
146. [-12,   5,  -1,  -9,   0,  -1,  -3,  -3,   2,  57]])
148. x = np.array([[1111,1,1111,1,1,1,1111,1,1,1]]).T
149. b = np.array([[2, 6, 3, 8, -1, 9, 2, 13, 15,-9]]).T
150. Newtown(A, x, b)
152. loss\_all = []
154. **for** x **in** x\_all:
155. loss = np.linalg.norm(x - x\_all[-1])
156. loss\_all.append(loss)
158. plt.plot(k\_all, loss\_all)
159. plt.xlabel("k")
160. plt.ylabel("loss")
161. plt.savefig('Newton.png')