最优化大作业

张晋

北京航空航天大学，数学与系统科学学院

2017 年 12 月 24 日

目录

1 Problem 5.8

1.1 重要参数

$$\begin{aligned} f(x_1,x_2) =& -9x_1-10x_2 \\ & -\mu\big[\ln(-x_1-x_2+100)+\ln(-x_1+x_2+50)+\ln(x_1)+\ln(x_2)\big] \end{aligned}$$

$$\begin{aligned} \boldsymbol{g}(x_1,x_2) &= \nabla f(x_1,x_2) \\ &= \begin{bmatrix} -9-\dfrac{\mu}{x_1}+\dfrac{\mu}{100-x_1-x_2}+\dfrac{\mu}{50-x_1+x_2} \\ -10-\dfrac{\mu}{x_2}+\dfrac{\mu}{100-x_1-x_2}-\dfrac{\mu}{50-x_1+x_2} \end{bmatrix} \end{aligned}$$

$$\begin{aligned} & \boldsymbol{G}[x_1,x_2] \\ =& \nabla \boldsymbol{g}(x_1,x_2) \\ =& \begin{bmatrix} \dfrac{1}{{x_1}^2}+\dfrac{1}{(100-x_1-x_2)^2}+\dfrac{1}{(50-x_1+x_2)^2} & \dfrac{1}{(100-x_1-x_2)^2}-\dfrac{1}{(50-x_1+x_2)^2} \\ \dfrac{1}{(100-x_1-x_2)^2}-\dfrac{1}{(50-x_1+x_2)^2} & \dfrac{1}{{x_2}^2}+\dfrac{1}{(100-x_1-x_2)^2}+\dfrac{1}{(50-x_1+x_2)^2} \end{bmatrix} \end{aligned}$$

1.2 算法伪代码

Algorithm 1 Backtracking-Armijo Line Search

```
Choose ᾱ > 0,   γ, ρ ∈ (0, 1)
Set α = ᾱ
while φ(α) > φ(0) + ρφ'(0)α do
    Set α = γα
end while
return  α as α_k
```

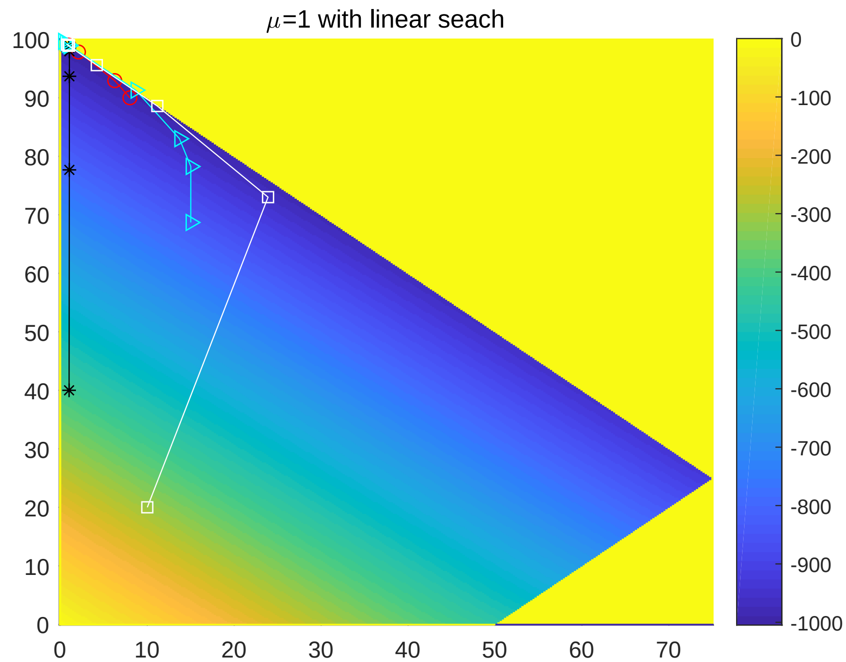
Algorithm 2 Newton method without Armijo Line Searchfor problem(5.8)

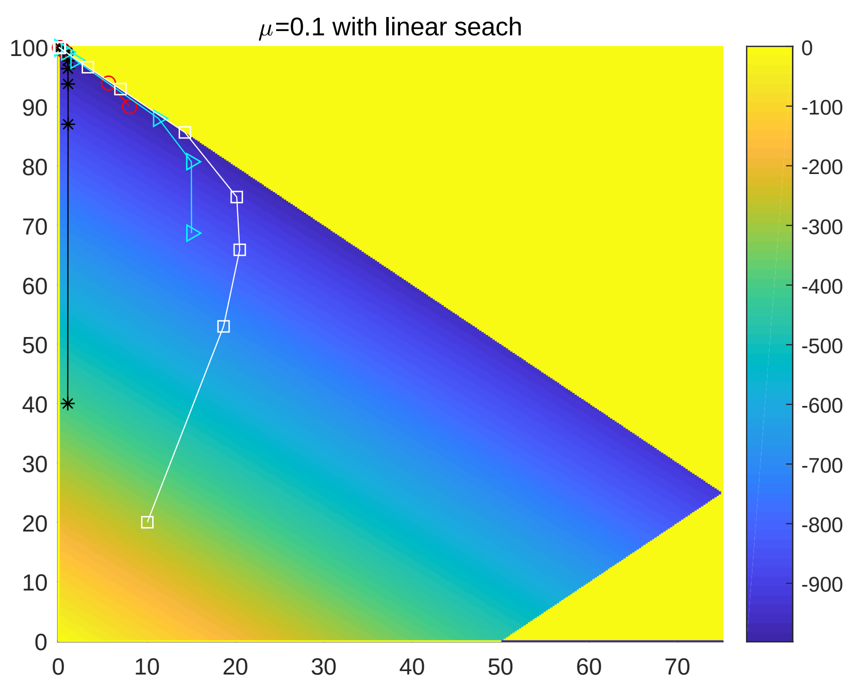
Given $\boldsymbol{x}^{(0)}$ and compute $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$
Compute $\boldsymbol{G}(\boldsymbol{x}) = \nabla \boldsymbol{g}(\boldsymbol{x})^T$
Set $\boldsymbol{g}^{(0)} = \boldsymbol{g}(\boldsymbol{x}^{(0)}), k = 0$
while $\|\boldsymbol{g}^{(k)}\|_2 > \epsilon$ **do**
    Set $\boldsymbol{s}^{(k)} = -{\boldsymbol{G}^{(k)}}^{-1}\boldsymbol{g}^{(k)}$
    Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{s}^{(k)}$
    Set k=k+1
end while

Algorithm 3 Newton method with Armijo Line Searchfor problem(5.8)

Given $\boldsymbol{x}^{(0)}$ and compute $\boldsymbol{g}(\boldsymbol{x}) = \nabla f(\boldsymbol{x})$
Compute $\boldsymbol{G}(\boldsymbol{x}) = \nabla \boldsymbol{g}(\boldsymbol{x})^T$
Set $\boldsymbol{g}^{(0)} = \boldsymbol{g}(\boldsymbol{x}^{(0)}), k = 0$
while $\|\boldsymbol{g}^{(k)}\|_2 > \epsilon$ **do**
    Set $\boldsymbol{s}^{(k)} = -{\boldsymbol{G}^{(k)}}^{-1}\boldsymbol{g}^{(k)}$
    Compute α_k by Line Search(**Algorithm** 1)
    Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k\boldsymbol{s}^{(k)}$
    Set k=k+1
end while

1.3 迭代点运动轨迹展示





1.4 总结分析

本题中 Armijo 线搜索的参数为 $\gamma = 0.5, \rho = 0.01$

没加线搜索和越界判定之前，牛顿法总是一两次就冲到了定义域外，然后无法收敛。

然后我加了个越界判定: 若下一步的迭代点不在定义域内，则将步长 α 缩小一半，牛顿法很好地收敛到了全局最优点。

```
%Check函数检查是否越界，以缩短步长ak
while(Check(x+ak*double(p')))
    ak=0.5*ak;
    xk=x+ak*double(p');
end
```

而后在越界判定的基础上加了个线搜索：

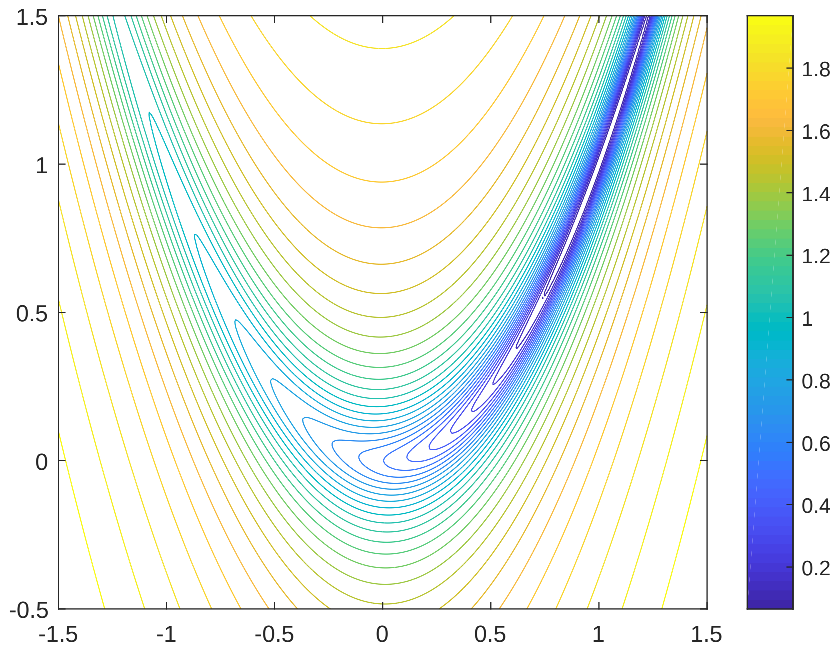
```
%采用Armijo法则计算近似步长ak
while(F(xk(1),xk(2)) > (F(x(1),x(2))+0.01*double(p'*g)*ak)
||Check(x+ak*double(p')))%Check函数检查是否越界
    ak=0.5*ak;
    xk=x+ak*double(p');
end
```

然而得到的结果与加上线搜索之前一样，可见在迭代的前期，越界判定起到了一定类似于线搜索缩短步长的效果，而到了迭代的后期，牛顿法的基本步长已经满足 Armijo 法则，还原成了基本牛顿法。

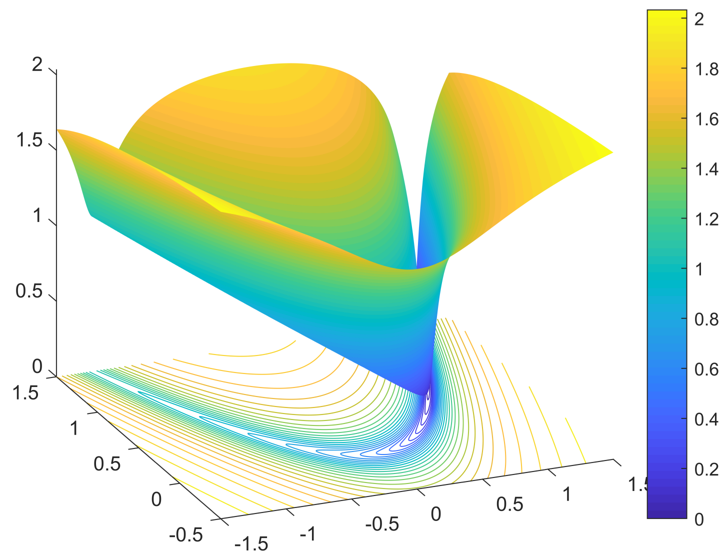
2 Problem5.9

2.1 Rosenbrock 函数图像

首先画出 Rosenbrock 函数的图像及等值线如下[1]:



[1] 由于 Rosenbrock 函数过于陡峭，因此对原函数进行了取对数处理，以便于观察其特点。



2.2 算法伪代码

Armijo 线搜索法参看上节的 (**Algorithm** 1)

带线搜索的 Newton 法算法参看上节的 (**Algorithm** 3)

Algorithm 4 Steepest-denscent-Armijo method for problem(5.9)

Given $\boldsymbol{x}^{(0)}$ and G
Set $\boldsymbol{p}^{(0)} = -\boldsymbol{g}^{(0)}, k = 0$
while $\|\boldsymbol{g}^{(k)}\| > \epsilon$ **do**
  Compute α_k by Line Search(**Algorithm** 1)
  Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}$
  Set $\boldsymbol{g}^{(k+1)} = g(\boldsymbol{x}^{(k+1)})$
  Set $\boldsymbol{p}^{(k)} = -\boldsymbol{g}^{(k)}$
  k=k+1
end while

2.3 计算结果展示

本题中 Armijo 线搜索的参数为 $\gamma = 0.5, \rho = 0.01$, 并设置最大搜索步长为 200.

然后分别以梯度下降法和牛顿法迭代，并画出等高线、运动轨迹、迭代值如下:

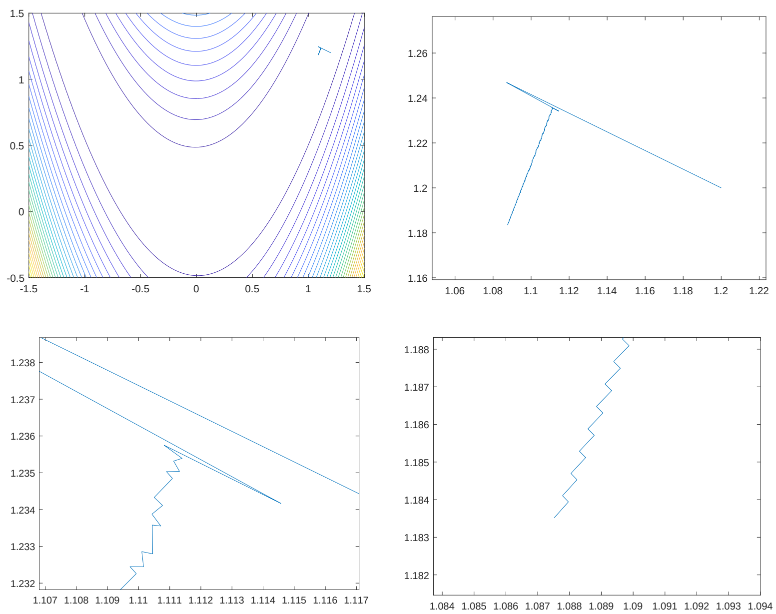


图 1: Steepest-denscent in (1.2,1.2)

```
%Result for Steepest-denscent in (1.2,1.2)
Step[1]:  x=[ 1.200000 1.200000 ] optim_fx=5.800000
Step[2]:  x=[ 1.087109 1.246875 ] optim_fx=0.430975
Step[3]:  x=[ 1.114571 1.234166 ] optim_fx=0.019689
...
...
...
Step[198]:  x=[ 1.083582 1.174725 ] optim_fx=0.007019
Step[199]:  x=[ 1.083742 1.174500 ] optim_fx=0.007013
Step[200]:  x=[ 1.083580 1.174500 ] optim_fx=0.006998
%最速下降法，共迭代 200 步
%最优解：
x=[ 1.083580e+00 1.174500e+00 ] optim_fx=0.006998
```

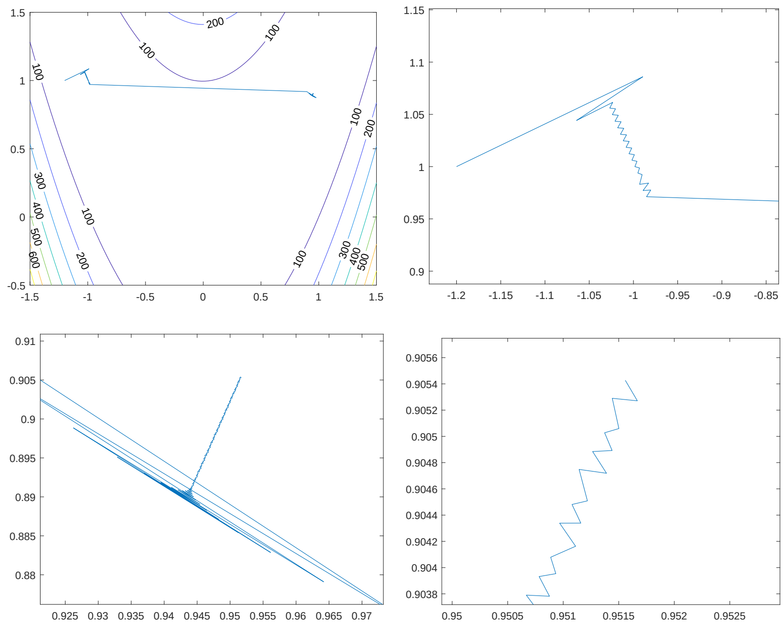


图 2: Steepest-denscent in (-1.2,1)

```
%Result for Steepest-denscent in (-1.2,1)
Step[1]:  x=[ -1.200000 1.000000 ] optim_fx=24.200000
Step[2]:  x=[ -0.989453 1.085938 ] optim_fx=5.101113
Step[3]:  x=[ -1.026893 1.065055 ] optim_fx=4.119416
Step[4]:  x=[ -1.027979 1.056815 ] optim_fx=4.112700
Step[5]:  x=[ 0.984742 1.049394 ] optim_fx=0.635080
Step[6]:  x=[ 1.015421 1.033832 ] optim_fx=0.000995
...
...
...
Step[195]:  x=[ 1.005171 1.010402 ] optim_fx=0.000027
Step[196]:  x=[ 1.005177 1.010389 ] optim_fx=0.000027
Step[197]:  x=[ 1.005163 1.010386 ] optim_fx=0.000027
Step[198]:  x=[ 1.005169 1.010373 ] optim_fx=0.000027
Step[199]:  x=[ 1.005155 1.010370 ] optim_fx=0.000027
Step[200]:  x=[ 1.005161 1.010357 ] optim_fx=0.000027
%最速下降法,共迭代 200 步
%最优解:
x=[ 1.005161e+00 1.010357e+00 ] optim_fx=0.000027
```

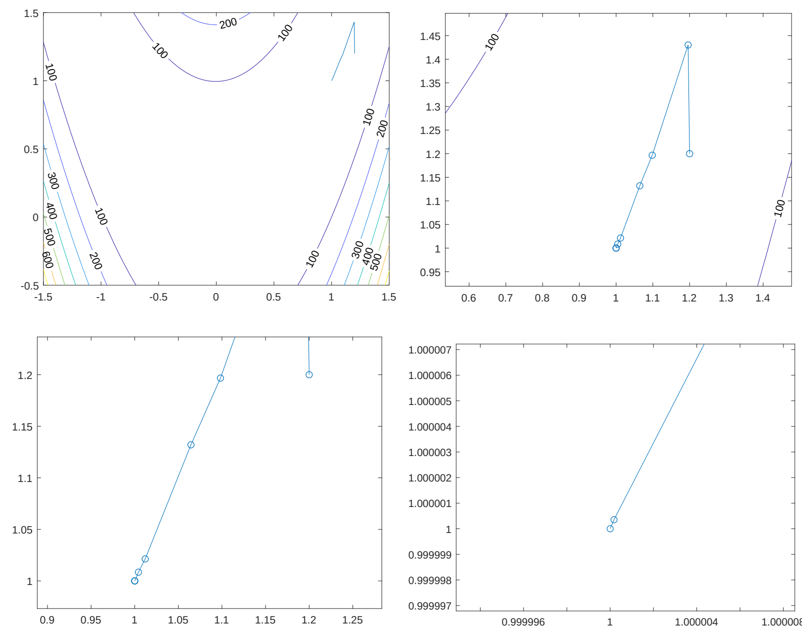


图 3: Newton-Armijo in (1.2,1.2)

```
%Result for Newton-Armijo in (1.2,1.2)
Step[1]: x=[ 1.200000 1.200000 ] optim_fx=5.800000
Step[2]: x=[ 1.195918 1.430204 ] optim_fx=0.038384
Step[3]: x=[ 1.098284 1.196688 ] optim_fx=0.018762
Step[4]: x=[ 1.064488 1.131993 ] optim_fx=0.004289
Step[5]: x=[ 1.011992 1.021372 ] optim_fx=0.000903
Step[6]: x=[ 1.004261 1.008481 ] optim_fx=0.000019
Step[7]: x=[ 1.000050 1.000083 ] optim_fx=0.000000
Step[8]: x=[ 1.000000 1.000000 ] optim_fx=0.000000
Step[9]: x=[ 1.000000 1.000000 ] optim_fx=0.000000
%牛顿 Armijo 回溯法,,共迭代 9 步
%最优解:
x=[ 1.000000e+00 1.000000e+00 ] optim_fx=0.000000
```

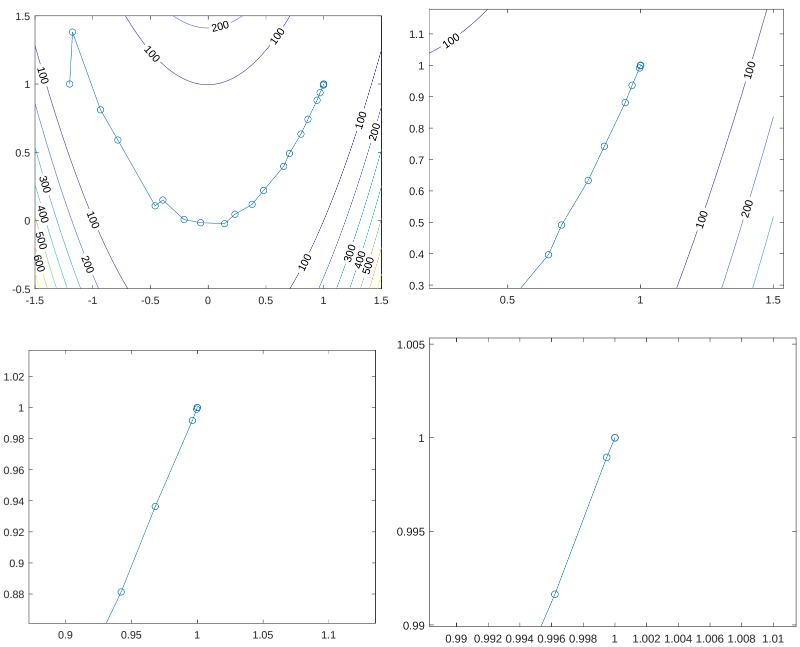


图 4: Newton-Armijo in (-1.2,1)

```
%Result for Newton-Armijo in (-1.2,1)
Step[1]:  x=[ -1.200000 1.000000 ] optim_fx=24.200000
Step[2]:  x=[ -1.175281 1.380674 ] optim_fx=4.731884
Step[3]:  x=[ -0.932981 0.811211 ] optim_fx=4.087399
Step[4]:  x=[ -0.782540 0.589736 ] optim_fx=3.228673
Step[5]:  x=[ -0.459997 0.107563 ] optim_fx=3.213898
Step[6]:  x=[ -0.393046 0.150002 ] optim_fx=1.942585
...
...
...
Step[17]:  x=[ 0.942079 0.881336 ] optim_fx=0.007169
Step[18]:  x=[ 0.967992 0.936337 ] optim_fx=0.001070
Step[19]:  x=[ 0.996210 0.991639 ] optim_fx=0.000078
Step[20]:  x=[ 0.999479 0.998948 ] optim_fx=0.000000
Step[21]:  x=[ 0.999999 0.999998 ] optim_fx=0.000000
Step[22]:  x=[ 1.000000 1.000000 ] optim_fx=0.000000
Step[22]: x=[ 1.000000 1.000000 ] optim_fx=0.000000
%牛顿 Armijo 回溯法,,共迭代 22 步
%最优解:
x=[ 1.000000e+00 1.000000e+00 ] optim_fx=0.000000
```