Numerical Optimization, 2020 Fall

Homework 7

Due on 14:59 NOV 26, 2020

请尽量使用提供的 tex 模板, 若手写作答请标清题号并拍照加入文档.

1 收敛速率

分别构造具有次线性，线性，超线性和二阶收敛速率的序列的例子。[10 pts] **解**

- 次线性: $\{\frac{1}{k}\}$
- 线性: $\{\frac{1}{2^k}\}$
- 超线性: $\{\frac{1}{k!}\}$
- 二阶: $\{1+(0.5)^{2^k}\}$

2 梯度下降法的收敛性分析

考虑如下优化问题:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad f(\boldsymbol{x}), \tag{1}$$

其中目标函数 f 满足一下性质:

- 对任意 $\boldsymbol{x}$, $f(\boldsymbol{x}) \geq \underline{f}$。
- ∇f 是 Lipschitz 连续的，即对于任意的 $\boldsymbol{x},\boldsymbol{y}$，存在 $L>0$ 使得

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L\|\boldsymbol{x}-\boldsymbol{y}\|_2.$$

若采用梯度下降法求解问题(1)，记所产生的迭代点序列为 $\{\boldsymbol{x}^k\}$。迭代点的更新为 $\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k$。试证明以下问题。

(i) 在一点 $\boldsymbol{x}^k$ 处给定一个下降方向 $\boldsymbol{d}^k$，即 $\boldsymbol{d}^k$ 满足 $\langle \nabla f(\boldsymbol{x}^k), \boldsymbol{d}^k\rangle < 0$。试证明：对于充分小的 $\alpha>0$，有 $f(\boldsymbol{x}+\alpha\boldsymbol{d}^k) < f(\boldsymbol{x}^k)$ 成立。[10 pts]

(ii) 假设存在 $\delta>0$ 使得 $-\frac{\langle \nabla f(\boldsymbol{x}^k), \boldsymbol{d}^k\rangle}{\|\nabla f(\boldsymbol{x}^k)\|_2\|\boldsymbol{d}^k\|_2} > \delta$。证明回溯线搜索会有限步终止，并给出对应步长 α^k 的下界。[10 pts]

(iii) 根据上一问结果证明 $\lim_{k\to\infty}\|\nabla f(\boldsymbol{x}^k)\|_2 = \mathbf{0}$。[10 pts]

(iv) 令 $\boldsymbol{d}^k = -\nabla f(\boldsymbol{x}^k)$，采用固定步长 $\alpha^k \equiv \alpha = \frac{1}{L}$。试证明该设定下梯度下降法的全局收敛性。[20 pts]

解

1. 由于 Δf 是 Lipschitz 连续的，并且 $\langle \nabla f(x^k), d^k \rangle < 0$，因此一定存在 $\alpha_0 \to 0$，使得对于 $\forall \alpha \in (0, \alpha_0]$ 有
$$\langle \nabla f(x^k + \alpha d^k), d^k \rangle < 0.$$
根据中值定理可得, 存在 $a' \in (0, \alpha_0]$ 使得
$$f(x^k + \alpha' d^k) = f(x^k) + \nabla f(x^k + \alpha' d^k)^T d^k.$$
综上，我们可以得到:
$$f(x^k + \alpha' d^k) = f(x^k) + \nabla f(x^k + \alpha' d^k)^T d^k < f(x^k).$$
Q.E.D.

2. 由 Wolf conditions：
$$\begin{aligned} f(x^k + \alpha^k d^k) &\le f(x^k) + c_1 \alpha^k \nabla f(x^k)^T d_k,\ c_1 \in (0,1) \\ \nabla f(x^k + \alpha^k d^k)^T &\ge c_2 \nabla f(x^k)^T d^k,\ c_2 \in (c_1, 1). \end{aligned}$$
再根据更新公式
$$x^{k+1} = x^k + \alpha d^k,$$
可得
$$(\nabla f(x^{k+1}) - \nabla f(x^k))^T d^k \ge (c_2 - 1) \nabla f(x^k)^T d^k.$$
由 Lipschitz 连续条件可以得到:
$$(\nabla f(x^{k+1}) - \nabla f(x^k))^T d^k \le \alpha^k L \|d^k\|_2^2.$$
综合以上两式可以得到 α^k 的下界:
$$\alpha^k \ge \frac{c_2 - 1}{L} \frac{\nabla f(x^k)^T d^k}{\|d^k\|_2^2} \ge \frac{c_2 - 1}{L} \delta \|\mathfrak{f}\|_2^2 \quad (\text{根据符号性质可以保证}).$$
下说明回溯线搜索算法会有限步终止。由于衰减系数 $\gamma \in (0,1)$，α^k 存在下界，因此一定存步长选择次数最多执行 ℓ 次，ℓ 满足
$$a^0 \gamma^l \le \frac{c_2 - 1}{L} \delta \|\mathfrak{f}\|_2^2, \quad a^0 \gamma^{\ell - 1} > \frac{c_2 - 1}{L} \delta \|\mathfrak{f}\|_2^2.$$
因此回溯线搜索算法会有限步终止，α^k 的下界也已求得 $\frac{c_2 - 1}{L} \delta \|\mathfrak{f}\|_2^2$.

3. α^k 的下界带入到 Wolf condition 中：
$$\begin{aligned} f(x^{k+1}) &\le f(x^k) - c_1 \alpha^k \nabla f(x^k)^T d_k \\ &\le f(x^k) - c_1 \frac{c_2 - 1}{L} \delta^2 \|\nabla f(x^k)\|_2^2. \end{aligned}$$

进一步我们可以得到

$$f(x^{k+1}) \le f(x^0) - c_1 \frac{c_2 - 1}{L} \delta^2 \sum_{j=0}^{k} \|\nabla f(x^j)\|_2^2.$$

对其取极限形式，再根据 $f(x^0) - f(x^{k+1})$ 的有界性:

$$c_1 \frac{c_2 - 1}{L} \delta^2 \sum_{j=0}^{k} \|\nabla f(x^j)\|_2^2 \le \infty$$

一个无穷数列收敛，因此我们有

$$\lim_{k \to \infty} \|\nabla f(x^k)\|_2^2 = 0$$

4. 将 $d^k = -\nabla f(x^k)$ 带入到 $-\frac{\langle \nabla f(\boldsymbol{x}^k), \boldsymbol{d}^k \rangle}{\|\nabla f(\boldsymbol{x}^k)\|_2 \|\boldsymbol{d}^k\|_2}$ 中可以得到，存在 $\delta = \frac{1}{2}$ 使得

$$\delta = \frac{1}{2} < -\frac{\langle \nabla f(\boldsymbol{x}^k), \boldsymbol{d}^k \rangle}{\|\nabla f(\boldsymbol{x}^k)\|_2 \|\boldsymbol{d}^k\|_2} = 1$$

并且步长

$$\alpha^k = \frac{1}{L} \ge \frac{1 - c_2}{L} = \frac{c_2 - 1}{L} \frac{\nabla f(x^k)^T d^k}{\|d^k\|_2^2}$$

满足 (2) 中的步长下界（也满足 Wolf 条件），因此根据 (2)(3) 问中的结论有

$$\delta^2 \sum_{j=0}^{k} \|\nabla f(x^j)\|_2^2 \le \infty \Rightarrow \lim_{k \to \infty} \|\nabla f(x^k)\|_2^2 = 0$$

即满足全局收敛性。

3 编程题

考虑求解如下优化问题:

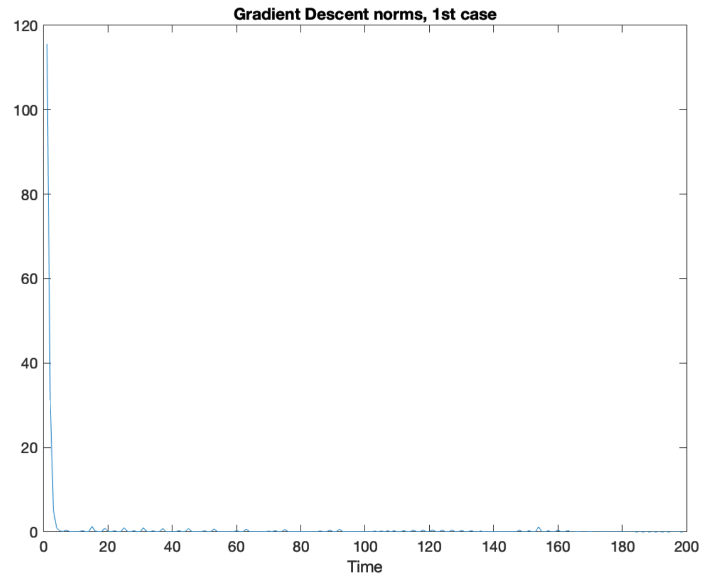
$$\min_{x_1, x_2} \quad 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \tag{2}$$

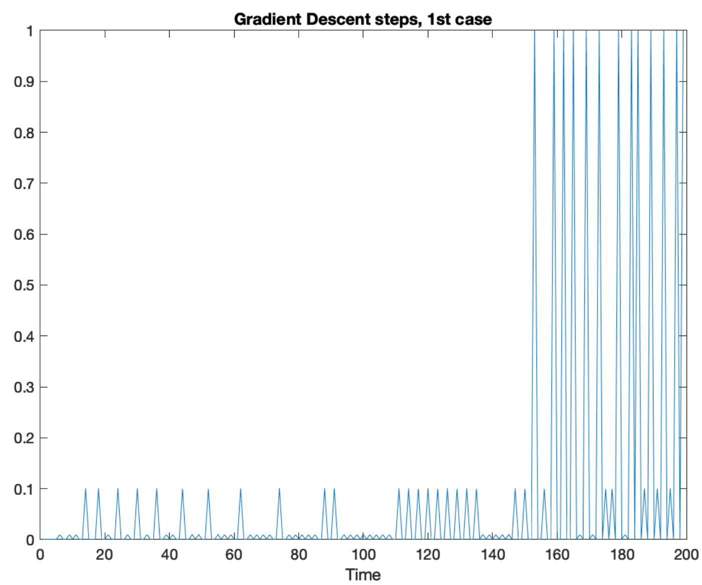
分别用**梯度下降法**和**牛顿法**结合 Armijo 回溯搜索编程求解该问题。分别考虑用 $\boldsymbol{x}^0 = [1.2, 1.2]^T$ 和 $\boldsymbol{x}^0 = [-1.2, 1]^T$(较困难) 作为初始点启动算法。

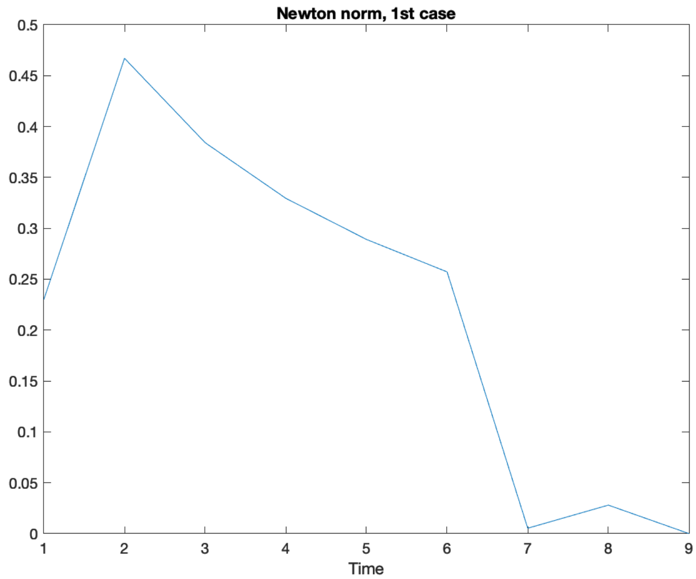
要求: 对于两种初始点，分别画出两种算法步长 α^k 和 $\|\nabla f(\boldsymbol{x}^k)\|_\infty$ 随迭代步数 k 变化的曲线。(编程可使用 matlab 或 python 完成，请将代码截图贴在该文档中。) [40pts]

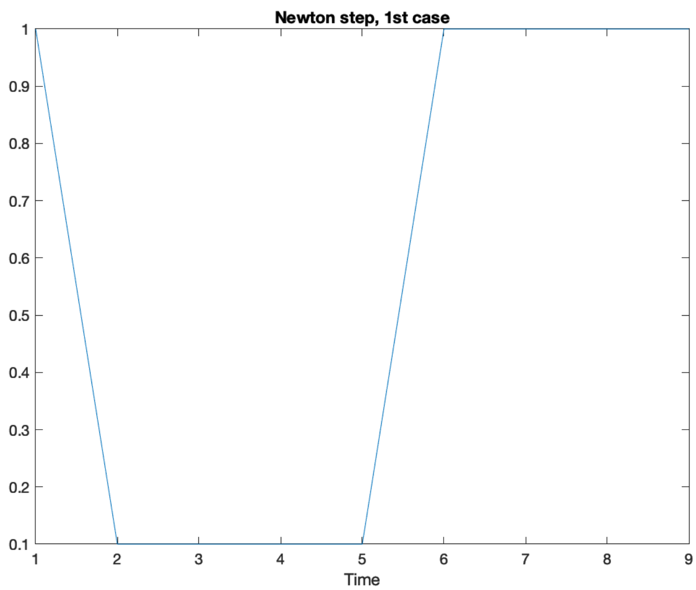
(Hint: 步长初始值 $\alpha_0 = 1$, 参数 c_1 可选为 10^{-4}，终止条件为 $\|\nabla f(\boldsymbol{x}^k)\|_\infty \le 10^{-4}$.) **解**

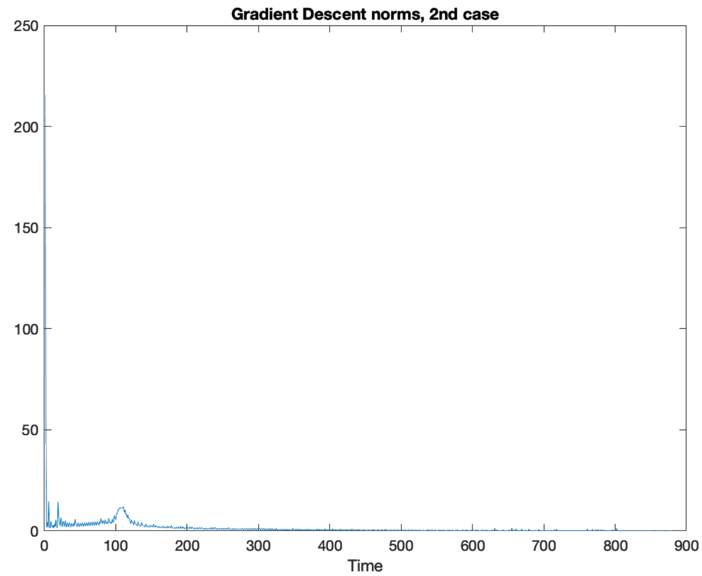
- GD1: $(x_1, x_2) = (1.000046543809064, 1.000096895389840)$, objective $= 3.614589398332103e - 09$.
- Newton1: $(x_1, x_2) = (1.000002952400574, 1.000005898886329)$, objective $= 8.720177978515248e - 12$.
- GD2: $(x_1, x_2) = (0.999950301741476, 0.999902794047657)$, objective $= 2.948692780318477e - 09$.
- Newton2: $(x_1, x_2) = (0.999999999996608, 0.999999999985182)$, objective $= 6.466763285974932e - 21$.

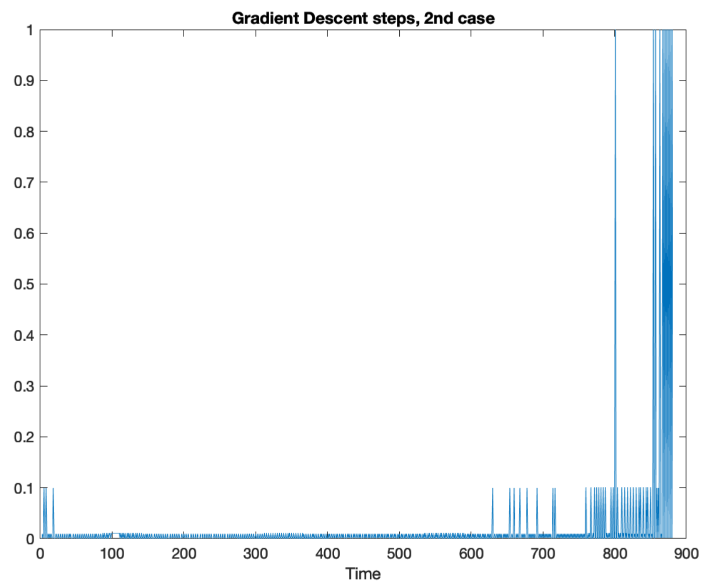

Gradient Descent norms, 1st case
Time

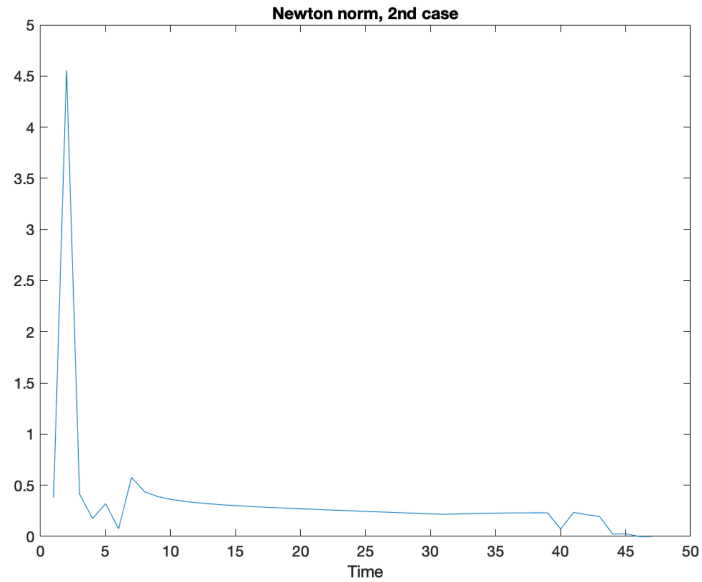

Gradient Descent steps, 1st case
Time

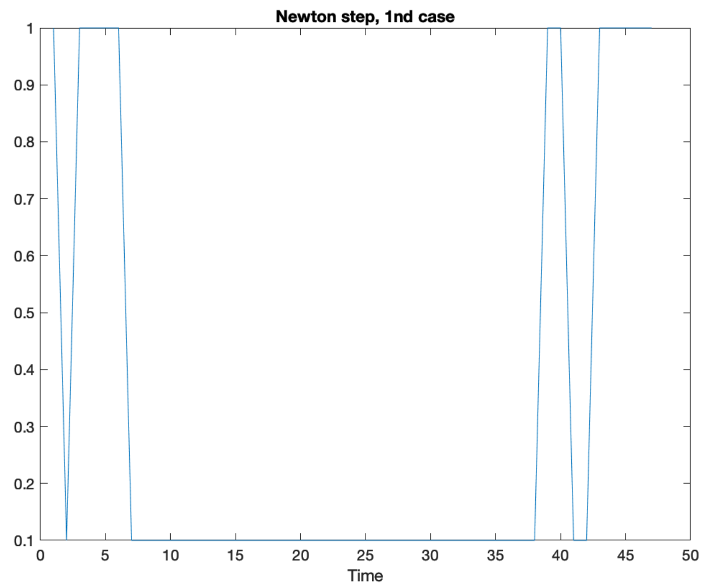

Newton norm, 1st case
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
3
4
5
6
7
8
9
Time


Newton step, 1st case
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9
Time


Gradient Descent norms, 2nd case
250
200
150
100
50
0
0
100
200
300
400
500
600
700
800
900
Time


Gradient Descent steps, 2nd case
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
100
200
300
400
500
600
700
800
900
Time





Appendices

x1 = 1.2;

x2 = 1.2;

alpha = 1;

c1 = 1e-4;

gamma = 1e-1;

gradients = [];

steps = [];

gradient = 0.1;

while norm(gradient,'inf') > 1e-4

```
    step = alpha;
    gradient = [-400*(x2-x1^2)*(x1)-2*(1-x1); 200*(x2-x1^2)];
    newx1 = x1 - step * gradient(1);
    newx2 = x2 - step * gradient(2);
    while objective(newx1, newx2) > objective(x1,x2) + c1 * step * gradient' * -gradient
        step = step * gamma;
        newx1 = x1 - step * gradient(1);
        newx2 = x2 - step * gradient(2);
    end
    x1 = newx1;
    x2 = newx2;
    steps = [steps, step];
    %pause(0.1)
    norm_grad = norm(gradient, "inf");
    gradients = [gradients, norm_grad];
end
x1
x2

plot(gradients);xlabel('Time');title('Gradient Descent norms, 1st case');
plot(steps);xlabel('Time');title('Gradient Descent steps, 1st case');

x1 = 1.2;
x2 = 1.2;
alpha = 1;
c1 = 1e-4;
gamma = 1e-1;
gradients = [];
steps = [];
gradient = 0.1;
while norm(gradient,'inf') > 1e-4
    step = alpha;
    gradient = [-400*(x2-x1^2)*(x1)-2*(1-x1); 200*(x2-x1^2)];
    Hess = [-400*(x2-x1^2)-400*x1*(-2*x1)+2,-400*x1;-400*x1,200];
    gradient = Hess \ gradient;
    newx1 = x1 - step * gradient(1);
    newx2 = x2 - step * gradient(2);
    while objective(newx1, newx2) > objective(x1,x2) + c1 * step * gradient' * -gradient
        step = step * gamma;
        newx1 = x1 - step * gradient(1);
        newx2 = x2 - step * gradient(2);
    end
```

```
    x1 = newx1;
    x2 = newx2;
    steps = [steps, step];
    %pause(0.1)
    norm_grad = norm(gradient, "inf");
    gradients = [gradients, norm_grad];
end
x1
x2

plot(gradients);xlabel('Time');title('Newton norm, 1st case');
plot(steps);xlabel('Time');title('Newton step, 1st case');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x1 = -1.2;
x2 = 1;
alpha = 1;
c1 = 1e-4;
gamma = 1e-1;
gradients = [];
steps = [];
gradient = 0.1;
while norm(gradient,'inf') > 1e-4
    step = alpha;
    gradient = [-400*(x2-x1^2)*(x1)-2*(1-x1); 200*(x2-x1^2)];
    newx1 = x1 - step * gradient(1);
    newx2 = x2 - step * gradient(2);
    while objective(newx1, newx2) > objective(x1,x2) + c1 * step * gradient' * -gradient
        step = step * gamma;

        newx1 = x1 - step * gradient(1);
        newx2 = x2 - step * gradient(2);
    end
    x1 = newx1;
    x2 = newx2;
    steps = [steps, step];
    %pause(0.1)
    norm_grad = norm(gradient, "inf");
    gradients = [gradients, norm_grad];
end

plot(gradients);xlabel('Time');title('Gradient Descent norms, 2nd case');
plot(steps);xlabel('Time');title('Gradient Descent steps, 2nd case');
```

```
x1 = -1.2;
x2 = 1;
alpha = 1;
c1 = 1e-4;
gamma = 1e-1;
gradients = [];
steps = [];
gradient = 0.1;
while norm(gradient,'inf') > 1e-4
    step = alpha;
    gradient = [-400*(x2-x1^2)*(x1)-2*(1-x1); 200*(x2-x1^2)];
    Hess = [-400*(x2-x1^2)-400*x1*(-2*x1)+2,-400*x1;-400*x1,200];
    gradient = Hess \ gradient;
    newx1 = x1 - step * gradient(1);
    newx2 = x2 - step * gradient(2);
    while objective(newx1, newx2) > objective(x1,x2) + c1 * step * gradient' * -gradient
        step = step * gamma;
        newx1 = x1 - step * gradient(1);
        newx2 = x2 - step * gradient(2);
    end
    x1 = newx1;
    x2 = newx2;
    steps = [steps, step];
    %pause(0.1)
    norm_grad = norm(gradient, "inf");
    gradients = [gradients, norm_grad];
end

plot(gradients);xlabel('Time');title('Newton norm, 2nd case');
plot(steps);xlabel('Time');title('Newton step, 1nd case');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [y] = objective(x1,x2)
%OBJECTIVE Summary of this function goes here
%   Detailed explanation goes here
y = 100 * (x2-x1^2)^2 + (1-x1)^2;
end
```