DATA130026.01 Optimization Assignment 10

Due Time: at the beginning of the class, Jun. 4, 2024
姓名: 雍崔扬
学号: 21307140051

Problem 1

Let $f(x) := \|x\|_2^\beta$, where $\beta > 0$ is given.
Suppose you use Pure Newton's method (with stepsize 1) to minimize f,
and initial point $x^{(0)} \neq 0_n$.

(a) If $\beta > \frac{3}{2}$ and $\beta \neq 2$, then $\{x^{(k)}\}$ converges to 0 linearly.
Explain why we do not have local quadratic convergence shown in the class.

- **引理 6.3.6:(Sherman-Morrison-Woodbury 公式)**
  FDU 高等线性代数 专题B. Shermann–Morrison–Woodbury 公式
  若 $\begin{cases} A \in \mathbb{C}^{n\times n} \\ U, V \in \mathbb{C}^{n\times k} \\ B \in \mathbb{C}^{k\times k} \\ n \geq k \end{cases}$ 且 A, B 均为可逆矩阵,
  则 $A + UBV^H$ 可逆的充要条件是 $B^{-1} + V^H A^{-1} U$ 可逆.
  此时逆矩阵 $(A + UBV^H)^{-1} = A^{-1} - A^{-1}(B^{-1} + V^H A^{-1} U)^{-1} V^H A^{-1}$
  (当 $k \ll n$ 时适用于求解逆矩阵 $(A + UBV^H)^{-1}$ 以及相应的线性系统 $(A + UBV^H)x = d$)
  - **Sherman-Morrison-Woodbury 公式的实用形式:**
    取 $\begin{cases} B = I_k \\ k = 1 \end{cases}$, 记 $u, v \in \mathbb{C}^n$
    则我们有 $(A + uv^H)^{-1} = A^{-1} - \frac{A^{-1}uv^H A^{-1}}{1 + v^H A^{-1} u}$ 成立.
- **Solution:**
  目标函数 $f(x) = \|x\|_2^\beta = (x^T x)^{\frac{\beta}{2}} \ \ (\beta > 0)$
  梯度 $\nabla f(x) = \frac{\beta}{2}(x^T x)^{\frac{\beta}{2}-1} \cdot 2x = \beta (x^T x)^{\frac{\beta}{2}-1} x$
  Hessian 矩阵:
  $$\begin{aligned}\nabla^2 f(x) &= \beta\{(x^T x)^{\frac{\beta}{2}-1} \cdot I_n + x \cdot [(\frac{\beta}{2} - 1)(x^T x)^{\frac{\beta}{2}-2} \cdot 2x]^T\} \\ &= \beta\{(x^T x)^{\frac{\beta}{2}-1} \cdot I_n + (\beta - 2)(x^T x)^{\frac{\beta}{2}-2} x x^T\} \\ &= \beta (x^T x)^{\frac{\beta}{2}-1}\{I_n + (\beta - 2)(x^T x)^{-1} x x^T\}\end{aligned}$$
  对 $I_n + (\beta - 2)(x^T x)^{-1} x x^T$ 应用 **Sherman-Morrison-Woodbury 公式**可得:
  $$\begin{aligned}\{I_n + (\beta-2)(x^Tx)^{-1}xx^T\}^{-1} &= I_n^{-1} - \frac{I_n^{-1}((\beta-2)(x^Tx)^{-1}xx^T)I_n^{-1}}{1 + x^T I_n^{-1} \cdot (\beta-2)(x^Tx)^{-1}x} \\ &= I_n - \frac{(\beta-2)(x^Tx)^{-1}}{1 + (\beta-2)} xx^T \\ &= I_n - \frac{\beta-2}{\beta-1}(x^Tx)^{-1}xx^T\end{aligned}$$
  因此 **Hessian 矩阵的逆矩阵**为:
  $$\begin{aligned}(\nabla^2 f(x))^{-1} &= \{\beta(x^Tx)^{\frac{\beta}{2}-1}[I_n + (\beta-2)(x^Tx)^{-1}xx^T]\}^{-1} \\ &= \frac{1}{\beta}(x^Tx)^{1-\frac{\beta}{2}} \cdot \{I_n + (\beta-2)(x^Tx)^{-1}xx^T\}^{-1} \\ &= \frac{1}{\beta}(x^Tx)^{1-\frac{\beta}{2}} \cdot \{I_n - \frac{\beta-2}{\beta-1}(x^Tx)^{-1}xx^T\}\end{aligned}$$

因此 **Newton 法的搜索方向**为：

$$
\begin{aligned}
d(x) &= -(\nabla^2 f(x))^{-1}\nabla f(x) \\
&= -\frac{1}{\beta}(x^Tx)^{1-\frac{\beta}{2}}\cdot\{I_n - \frac{\beta-2}{\beta-1}(x^Tx)^{-1}xx^T\}\cdot\beta(x^Tx)^{\frac{\beta}{2}-1}x \\
&= -\{I_n - \frac{\beta-2}{\beta-1}(x^Tx)^{-1}xx^T\}\cdot x \\
&= -x + \frac{\beta-2}{\beta-1}x
\end{aligned}
$$

因此**纯 Newton 法的迭代公式**为：(纯牛顿法步长 $t_k \equiv 1$)

$$
\begin{aligned}
x^{(k+1)} &= x^{(k)} + d^{(k)} \\
&= x^{(k)} - x^{(k)} + \frac{\beta-2}{\beta-1}x^{(k)} \\
&= \frac{\beta-2}{\beta-1}x^{(k)}
\end{aligned}
$$

我们定义误差函数 $e(x) = \|x - x_\star\|_2$，

其中全局最优值 $x_\star$ 显然为 0_n

于是我们有：

$$
\begin{aligned}
\mu &= \lim_{k\to\infty}\frac{e_{k+1}}{e_k} \\
&= \lim_{k\to\infty}\frac{\|x^{(k+1)} - x_\star\|}{\|x^{(k)} - x_\star\|} \\
&= \lim_{k\to\infty}\frac{\|x^{(k+1)}\|}{\|x^{(k)}\|} \\
&= \lim_{k\to\infty}\frac{\|\frac{\beta-2}{\beta-1}x^{(k)}\|}{\|x^{(k)}\|} \\
&= |\frac{\beta-2}{\beta-1}|
\end{aligned}
$$

当 $\beta > \frac{3}{2}$ 且 $\beta \neq 2$ 时我们有 $\mu = |\frac{\beta-2}{\beta-1}| \in (0,1)$

表明 $\{x^{(k)}\}$ **线性收敛**于 $x_\star = 0_n$

- 为什么这里的纯 Newton 法没有表现出课上提到的**局部二次收敛性**？

  这是因为目标函数 $f(x) = \|x\|_2^\beta$ $(\beta > 1, \beta \neq 2)$ 不是强凸函数，不满足该定理的条件：

  - $\nabla^2 f(x) = \beta(x^Tx)^{\frac{\beta}{2}-1}\{I_n + (\beta-2)(x^Tx)^{-1}xx^T\} = \beta\|x\|_2^{\beta-2}I_n + \beta(\beta-2)\|x\|_2^{\beta-4}xx^T$

  显然不存在 $\mu > 0$ 使得 $\nabla^2 f(x) \succeq \mu I_n$ 对于任意 $x \in \mathbb{R}^n$ 恒成立.

> **定理 6.3.1：(纯牛顿法的局部二阶收敛, 江如俊教授 Note 6 Theorem 2.1)**
> 若 $f : \mathbb{R}^n \mapsto \mathbb{R}$ 二阶连续可微，且满足：
>
> - f 是 μ-强凸的 (即 $f(x) - \frac{\mu}{2}x^Tx$ 是凸函数，其中 $\mu > 0$)
>   也即 $\nabla^2 f(x) \succ \mu I_n$ 对任意 $x \in \mathbb{R}^n$ 恒成立.
> - $\nabla^2 f$ 是 L-Lipschitz 连续的 (其中 $L > 0$)
>   即 $\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq L\|x - y\|$ 对任意 $x, y \in \mathbb{R}^n$ 恒成立.
>
> 则对于任意 $k = 0, 1\ldots$ 我们都有 $\|x^{(k+1)} - x_\star\| \leq \frac{L}{2\mu}\|x^{(k)} - x_\star\|$ 成立.
> 因此我们有 $\lim_{k\to\infty}\frac{e_{k+1}}{e_k} = \lim_{k\to\infty}\frac{\|x^{(k+1)} - x_\star\|}{\|x^{(k)} - x_\star\|} \leq \frac{L}{2\mu}$ 成立，即 $\{x^{(k)}\}$ 至少**次线性收敛**于 $x_\star$
>
> - 进一步，若起始点 $x^{(0)}$ 与唯一的全局最小点 $x_\star$ 足够接近 (具体来说，是 $\|x^{(0)} - x_\star\| \leq \frac{\mu}{L}$)
>   则我们有 $e_k = \|x^{(k)} - x_\star\| \leq \frac{2\mu}{L}(\frac{1}{2})^{2^k}$ 成立.
>   于是 $\lim_{k\to\infty}\frac{e_{k+1}}{e_k^2} = \frac{\frac{2\mu}{L}(\frac{1}{2})^{2^{k+1}}}{(\frac{2\mu}{L}(\frac{1}{2})^{2^k})^2} = \frac{L}{2\mu} > 0$，说明 $\{x^{(k)}\}$ **二阶收敛**于 $x_\star$

(b) If $0 < \beta < 1$, then the method diverges.

- **Solution:**
  根据 (a) 的结论，我们有 $\begin{cases} x_\star = 0 \\ x^{(k+1)} = \frac{\beta-2}{\beta-1}x^{(k)} \\ \mu = \lim\limits_{k\to\infty}\frac{e_{k+1}}{e_k} = \lim\limits_{k\to\infty}\frac{\|x^{(k+1)}-x_\star\|}{\|x^{(k)}-x_\star\|} = |\frac{\beta-2}{\beta-1}| \end{cases}$
  当 $0 < \beta < 1$ 时，我们有 $\mu = |\frac{\beta-2}{\beta-1}| = \frac{2-\beta}{1-\beta} \in (1,\infty)$
  表明 $x^{(k)} \to +\infty$
  (或者我们根据 $\lim\limits_{k\to\infty}\|x^{(k)}\| = \lim\limits_{k\to\infty}(\frac{2-\beta}{1-\beta})^k\|x^{(0)}\| = \infty$ 得到)
  而全局最优值 $x_\star = 0_n$
  这说明当 $0 < \beta < 1$ 时，纯 Newton 法发散.

Problem 2

An engineer has decided to verify numerically that
the exponential function $x \to \exp(x) = e^x$ grows faster than any polynomial.
In order to do so, the engineer studies the optimization problem

$$\min f(x) = x^\alpha - e^x \tag{1}$$

where α is the highest power of the polynomial (we assume it is an even, positive integer number).

The engineer uses **Pure Newton method** to solve the problem.
The engineer argues that if the exponential function grows faster than any polynomial,
then the sequence $\{x_k\}$ generated by the method should diverge to infinity,
because the objective function f can be decreased indefinitely by increasing the value of x.

(a) State the Newton iteration explicitly for the given problem (1).

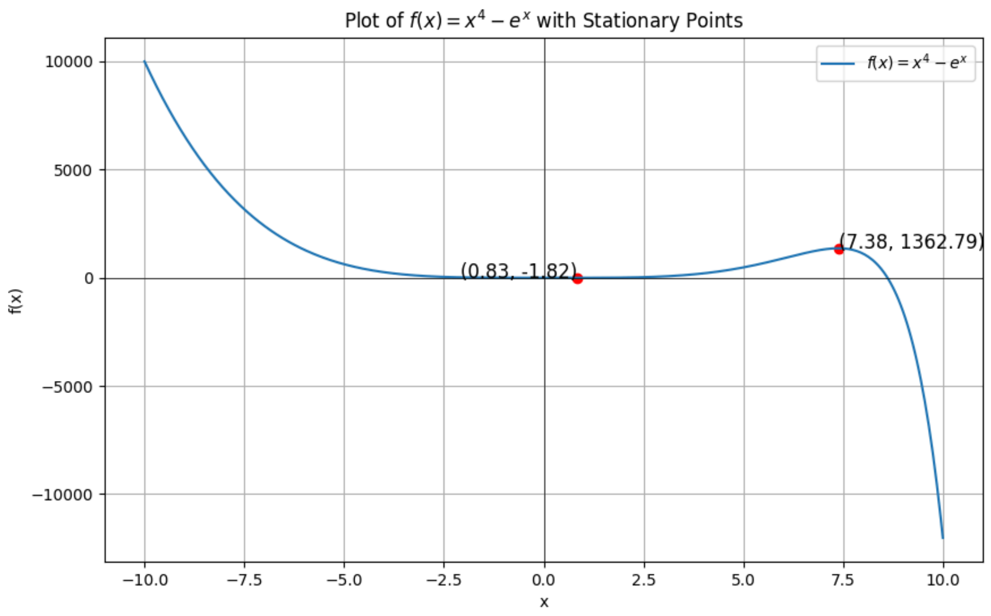
- **Solution:**
  目标函数 $f(x) = x^\alpha - e^x$
  梯度 $\nabla f(x) = \alpha x^{\alpha-1} - e^x$
  Hessian 矩阵 $\nabla^2 f(x) = \alpha(\alpha-1)x^{\alpha-2} - e^x$
  Newton 法搜索方向 $d(x) = -(\nabla^2 f(x))^{-1}\nabla f(x) = -\frac{\alpha x^{\alpha-1}-e^x}{\alpha(\alpha-1)x^{\alpha-2}-e^x}$
  纯牛顿法 (步长 $t_k \equiv 1$) 迭代公式:
  $x^{(k+1)} = x^{(k)} + d^{(k)} = x^{(k)} - \frac{\alpha(x^{(k)})^{\alpha-1}-e^{x^{(k)}}}{\alpha(\alpha-1)(x^{(k)})^{\alpha-2}-e^{x^{(k)}}}$

(b) Construct a numerical example (that is, choose a value of $\alpha \in \{2,4,\dots\}$ and a starting point of the Newton algorithm) illustrating the engineer's error in reasoning.

- **Solution:**
  工程师的想法是错误的,
  因为目标函数 $f(x) = x^\alpha - e^x \ \ (\alpha = 2,4,\dots)$ 并非凸函数,
  所以如果初始点 $x^{(0)}$ 选取的不好时，纯 Newton 法会收敛到最近的局部最小点 (如果存在的话)

例如 $\alpha = 4$ 就构成一个反例:
我们首先说明 $f(x) = x^4 - e^x$ 存在局部最小点:
$\begin{cases}\nabla f(x) = 4x^3 - e^x \\ \nabla^2 f(x) = 12x^2 - e^x\end{cases}$ 根据 $\begin{cases}\nabla f(0) = -1 < 0 \\ \nabla f(1) = 4 - e > 0\end{cases}$ 可知 f 在 $(0,1)$ 至少存在一个局部最小点.

我们可以使用 Python 代码验证:



$\alpha = 4$ 时的迭代公式是 $x^{(k+1)} = x^{(k)} - \frac{4(x^{(k)})^3 - e^{x^{(k)}}}{12(x^{(k)})^2 - e^{x^{(k)}}}$
若我们取初始点 $x^{(0)}$ 为 $\nabla f(x) = 4x^3 - e^x = 0$ 在区间 $(0,1)$ 的解 (这是一个局部最小点),
则我们有 $x^{(k)} \equiv x^{(0)}$
此时纯 Newton 法收敛到 $f(x^{(0)})$，而不是 $-\infty$

(c) Find the error in the engineer's reasoning and formally explain it.

- **Solution:**
$\begin{cases}\nabla f(x) = \alpha x^{\alpha-1} - e^x \\ \nabla^2 f(x) = \alpha(\alpha-1)x^{\alpha-2} - e^x\end{cases}$
当 $\alpha \geq 4$ 时，我们有 $\begin{cases}\nabla f(0) = -e < 0 \\ \nabla f(1) = \alpha - e \geq 4 - e > 0\end{cases}$
表明 f 在区间 $(0,1)$ 至少存在一个局部最小点，记为 $x_\star$.
它满足 $\alpha x_\star^{\alpha-1} = e^{x_\star}$
根据 $\begin{cases}\nabla^2 f(x_\star) = \alpha(\alpha-1)x_\star^{\alpha-2} - e^{x_\star} \\ \qquad = \alpha(\alpha-1)x_\star^{\alpha-2} - \alpha x_\star^{\alpha-1} \\ \qquad = \alpha x_\star^{\alpha-2} \cdot (\alpha - 1 - x_\star) \\ \qquad > 0 \\ \nabla^2 f(\alpha) = \alpha(\alpha-1)\alpha^{\alpha-2} - e^\alpha \\ \qquad = \frac{\alpha-1}{\alpha}\alpha^\alpha - e^\alpha \\ \qquad > 0\end{cases}$ 可知 f 在区间 $(x_\star, \alpha)$ 上是凸函数
因此只要纯 Newton 法的初始点 $x^{(0)}$ 取在区间 $(x_\star, \alpha)$ 上,
点列 $\{x^{(k)}\}$ 最终都会收敛到 $x_\star$ 而不是 $+\infty$

当 α 不断增大时，这个区间 $(x_\star, \alpha)$ 还会不断向右延伸，
所以任何预设的与 α 无关的初始点 $x^{(0)}$ 产生的点列 $\{x^{(k)}\}$，
都将在 α 足够大时收敛到 α 对应的局部最小点 $x_\star$，
因此我们说工程师的想法过于理想化了，无法实际应用.

Problem 3

Consider the following logistic regression problem:

$$\min_{w\in\mathbb{R}^n} \frac{1}{m}\sum_{i=1}^{m}\log(1+\exp(-b_i w^T a_i)) + \frac{1}{100m}\|w\|^2$$

where a_i, b_i are given.
Test your algorithms on three real datasets from libsvm: `a9a.test`, `CINA.test` and `ijcnn1.test`.
You may use the following MATLAB code to generate the data:

```
dataset = 'a9a.test';
[b, A] = libsvmread(dataset);
[m, n] = size(A);
mu = 1e-2/m;
```

(a) Use the zero vector as starting point.
Terminate your code when the Euclidean norm of the gradient is smaller than 10^{-6}.
Implement Newton's method and inexact Newton (Newton-CG) method (with three different inexact rules) to solve this problem and compare the convergence rate.
The three different tolerance rules for inexact Newton methods are $\begin{cases}\eta = \min(0.5, \|g\|)\|g\| \\ \eta = \min(0.5, \sqrt{\|g\|})\|g\| \\ \eta = 0.5\|g\|\end{cases}$.
You need to use Armijo rule line search for both methods.
Plot the results (four lines) on gradient norm in one figure, and on function values $l(x_k) - l(x^*)$, for each dataset. ($l(x^*)$ can be obtained by running Newton's method for higher accuracy.)

Solution:
考虑 **Logistic 回归**的例子：
$\min f(x) = \frac{1}{m}\sum\limits_{i=1}^{m}\log(1+\exp(-b_i a_i^T x)) + \lambda\|x\|^2$
其中标签 $b_i \in \{-1, 1\}$，正则化系数 $\lambda = \frac{1}{100m}$

记 $A = [a_1, \dots, a_m] \in \mathbb{R}^{n\times m}$，则目标函数可以表示为：
$f(x) = \frac{1}{m}1_m^T \log\{1_m + \exp[-b\circ(A^T x)]\} + \lambda\|x\|^2$

记 $p(x) = 1_m + \exp[-b\circ(A^T x)]$ (其中 $\oslash$ 代表点除)
记 $\sigma(x) = p^{-1}(x) = 1_m \oslash p(x)$ (这实际上相当于 Logistic 函数)
容易知道：

$$\begin{aligned}\nabla p(x) &= -A\cdot\text{diag}\{\exp[-b\circ(A^T x)]\circ b\} \\ &= -A\cdot\text{diag}\{(p(x)-1_m)\circ b\}\end{aligned}$$

因此我们有：

$$\begin{aligned}\nabla\sigma(x) &= \nabla p^{-1}(x) \\ &= \nabla p(x)\cdot\text{diag}\{-p^{-2}(x)\} \\ &= -A\cdot\text{diag}\{(p(x)-1_m)\circ b\}\cdot -\text{diag}\{p^{-2}(x)\} \\ &= A\cdot\text{diag}\{(1_m-\sigma(x))\circ\sigma(x)\circ b\}\end{aligned}$$

则我们有:

- 目标函数:
$$\begin{aligned} f(x) &= \frac{1}{m}1_m^T\log(p(x)) + \lambda\|x\|^2 \\ &= -\frac{1}{m}1_m^T\log(\sigma(x)) + \lambda\|x\|^2 \end{aligned}$$
- 梯度函数:
$$\begin{aligned} \nabla f(x) &= -\frac{1}{m}\nabla\sigma(x)\cdot(\sigma^{-1}(x)\circ 1_m) + 2\lambda x \\ &= -\frac{1}{m}A\cdot\mathrm{diag}\{(1_m-\sigma(x))\circ\sigma(x)\circ b\}\cdot\mathrm{diag}(\sigma^{-1}(x))1_m + 2\lambda x \\ &= -\frac{1}{m}A\cdot\mathrm{diag}\{(1_m-\sigma(x))\circ b\}1_m + 2\lambda x \\ &= -\frac{1}{m}A\cdot\{(1_m-\sigma(x))\circ b\} + 2\lambda x \end{aligned}$$
- Hessian 矩阵:
$$\begin{aligned} \nabla^2 f(x) &= -\frac{1}{m}\nabla\sigma(x)\cdot[-A\cdot\mathrm{diag}(b)]^T + 2\lambda I_n \\ &= -\frac{1}{m}A\cdot\mathrm{diag}\{(1_m-\sigma(x))\circ\sigma(x)\circ b\}\cdot[-\mathrm{diag}(b)A^T] + 2\lambda I_n \\ &= \frac{1}{m}A\cdot\mathrm{diag}\{(1_m-\sigma(x))\circ\sigma(x)\circ b^2\}\cdot A^T + 2\lambda I_n \\ &= \frac{1}{m}A\cdot\mathrm{diag}\{(1_m-\sigma(x))\circ\sigma(x)\circ 1_m\}\cdot A^T + 2\lambda I_n \quad (\text{label } b^2 = 1_m) \\ &= \frac{1}{m}A\cdot\mathrm{diag}\{(1_m-\sigma(x))\circ\sigma(x)\}\cdot A^T + 2\lambda I_n \end{aligned}$$

代码实现:

- 复合线性变换的 **Logistic 函数**:
$\sigma(x) = p^{-1}(x) = 1_m \oslash p(x) = 1_m \oslash \{1_m + \exp[-b\circ(A^Tx)]\}$

```
function logistic = Logistic(x,A,b)
    logistic = 1./(1 + exp(-b.*(A'*x)));
end
```

- **目标函数:**
$f(x) = -\frac{1}{m}1_m^T\log(\sigma(x)) + \lambda\|x\|^2$ (其中 $\lambda = \frac{1}{100m}$)

```
function object = Object(x,A,b)
    [~, m] = size(A);
    lambda = 0.01/m;
    logistic = Logistic(x,A,b);
    object = -(1/m)*(sum(log(logistic))) + lambda * (x'*x);
end
```

- **梯度:**
$\nabla f(x) = -\frac{1}{m}A\cdot\{(1_m-\sigma(x))\circ b\} + 2\lambda x$ (其中 $\lambda = \frac{1}{100m}$)

```
function grad = Gradient(x,A,b)
    [~, m] = size(A);
    lambda = 0.01/m;
    logistic = Logistic(x,A,b);
    grad = -(1/m)*(A*(b.*(1-logistic))) + 2*lambda*x;
end
```

- **Hessian 矩阵:**
$\nabla^2 f(x) = \frac{1}{m} A \cdot \mathrm{diag}\{(1_m - \sigma(x)) \circ \sigma(x)\} \cdot A^T + 2\lambda I_n$ (其中 $\lambda = \frac{1}{100m}$)

```
function hess = Hessian(x,A,b)
    [n, m] = size(A);
    lambda = 0.01/m;
    logistic = Logistic(x,A,b);
    vector = (1-logistic).*logistic;
    stack_vector = repmat(vector', n, 1);
    % 在行维度和列维度上分别重复 vector 的转置 n 次和 1 次, 构造 nxm 的矩阵
stack_vector
    % 用 (A.*stack_vector)*A' 代替 A*diag(vector)*A', 如果 m>n 的话可以节省空间
    % 而且将一次矩阵乘法变为矩阵点乘，时间复杂度也降低
    hess = (1/m)*((A.*stack_vector)*A') + 2*lambda*diag(ones(n,1));
end
```

Armijo 步长准则:
固定初始步长 $\hat{t}$ 和回溯因子 $\beta \in (0,1)$，以及尺度因子 $\alpha \in (0,1)$
我们不断试验步长 $\beta^m \hat{t} \ \ (m = 0, 1, \ldots)$ 直至找到第一个非负整数 m，
使得 $f(x^{(k)} + (\beta^m \hat{t}) d^{(k)}) - f(x^{(k)}) \le \alpha(\beta^m \hat{t}) \nabla f(x^{(k)})^T d^{(k)}$ 成立.
这不但保证了每步迭代有下降， 还保证了下降的程度是充分大的.
我们通常取 $\begin{cases} \hat{t} = 1 \\ \beta \in [0.1, 0.5] \\ \alpha \in [10^{-5}, 10^{-1}] \end{cases}$ 这里我们取 $\begin{cases} \hat{t} = 1 \\ \beta = 0.5 \\ \alpha = 10^{-4} \end{cases}$

```
function stepsize = Armijo(x,A,b,object,grad,direction)
    alpha = 1e-4;
    beta = 0.5;
    stepsize = 1;
    x_new = x + stepsize*direction;
    while Object(x_new,A,b) > (object + alpha * stepsize * (direction' * grad))
        stepsize = stepsize * beta;
        x_new = x + stepsize*direction;
    end
end
```

精确 (阻尼) Newton 法:

- **搜索方向** $d^{(k)} = -(\nabla^2 f(x^{(k)}))^{-1} \nabla f(x^{(k)})$

```
direction = -hess\grad;
```

- **步长准则：回溯线搜索 (本例使用 Armijo 准则)**

```
stepsize = Armijo(x,A,b,object,grad,direction);
```

非精确 (阻尼) Newton 法:

- **搜索方向:** 使用**共轭梯度法 (CG)** 近似求解线性系统 $\nabla^2 f(x^{(k)})d + \nabla f(x^{(k)}) = 0_n$

```
direction = CG(hess,grad,CG_max_iter,CG_tol_rule);
```

  - **共轭梯度法的阈值准则 (非精确规则):**
$$\begin{cases}\eta = \min(0.5, \|g\|)\|g\| \\ \eta = \min(0.5, \sqrt{\|g\|})\|g\| \\ \eta = 0.5\|g\|\end{cases}$$

```
function CG_tol = Inexact_Rule(ng, CG_tol_rule)
% 'ng' for 'norm of gradient'
% 'CG_tol' for 'tolerance of CG method'
    tol = zeros(3,1);
    tol(1) = min(0.5,ng)*ng;
    tol(2) = min(0.5,sqrt(ng))*ng;
    tol(3) = 0.5*ng;
    CG_tol = tol(CG_tol_rule);
end
```

  - **共轭梯度法 (Conjugate Gradient, CG):**
    它用于求解大型稀疏对称正定线性系统 $Ax = b$
    对应到 Newton-CG 算法中，这里的 $\begin{cases}A = \nabla^2 f(x^{(k)}) \\ b = -\nabla f(x^{(k)})\end{cases}$ 其中 k 是 Newton 迭代的计数.
    CG 算法求解的 x 便是 Newton 第 k 步迭代的搜索方向 $d^{(k)}$.

    为避免混淆，我们记 CG 算法的迭代计数为 m
    它产生的一系列 CG 搜索方向 $p^{(0)}, \dots, p^{(m)}$ 是共轭的，满足 $(d^{(i)})^T A d^{(j)} = 0 \ \ (\forall\, i \neq j)$

    - 初始化:
      - 起始点设为 $x^{(0)} = 0_n$ (通常设为零向量)
      - 初始残差 $r^{(0)} = b - Ax^{(0)}$
      - 初始搜索方向 $p^{(0)} = r^{(0)}$
    - 迭代：对于 $m = 0, 1, 2, \dots$ 进行如下步骤，直至收敛.
      - 计算步长 $t_m = \frac{(r^{(m)})^T r^{(m)}}{(p^{(m)})^T A p^{(m)}}$
      - 更新 CG 解 $x^{(m+1)} = x^{(m)} + t_m p^{(m)}$
      - 计算新的残差
        $r^{(m+1)} = b - Ax^{(m+1)} = b - A(x^{(m)} + t_m d^{(m)}) = r^{(m)} - t_m A p^{(m)}$
      - 检查收敛，即检查 $\|r^{(m+1)}\| \le \eta \|b\|$ 是否成立，若成立则迭代终止.
        阈值 η 一般有三种代表性的选择 (参考**定理 6.3.5 非精确牛顿法的收敛性**的内容):
        - $\eta = 0.5$ (对应的 Newton-CG 算法线性收敛)
        - $\eta = \min\{0.5, \|b\|\}$ (对应的 Newton-CG 算法超线性收敛)
        - $\eta = \min\{0.5, \sqrt{\|b\|}\}$ (对应的 Newton-CG 算法二阶收敛)
      - 若仍未收敛，则计算 $\beta_k = \frac{(r^{(m+1)})^T r^{(m+1)}}{(r^{(m)})^T r^{(m)}}$
        并更新搜索方向 $p^{(m+1)} = r^{(m)} + \beta_k p^{(m)}$

求解大型稀疏对称正定线性系统 $Ax + g = 0_n$ 的共轭梯度算法的 Matlab 代码：

```
function x = CG(A, g, CG_max_iter, CG_tol_rule)
    % CG solve Ax + g = 0
    x = zeros(size(g));
    ng = norm(g);
    % Tolerance for convergence
    CG_tol = Inexact_Rule(ng, CG_tol_rule);
    r = g;
    p = -r;

    for iter = 1:CG_max_iter
        % Compute r' * r
        rr = r' * r;
        Ap = A * p;
        stepsize = rr / (p' * Ap);
        % Update solution x
        x = x + stepsize * p;
        % Update residual r
        r = r + stepsize * Ap;
        % Compute the norm of the new residual
        nr = norm(r);

        % Check for convergence
        if nr <= CG_tol
            break;
        end

        beta = (nr^2) / rr;
        % Update search direction p
        p = -r + beta * p;
    end

    % Print the total number of iterations
    fprintf('Number of CG iteration: %d\t Norm of residual: %f\t', iter,
nr);

end
```

- **步长准则：回溯线搜索 (本例使用 Armijo 准则)**

```
stepsize = Armijo(x,A,b,object,grad,direction);
```

因此 Newton 法迭代器的代码可以为：

```
function [history,i] =
Newton_Iterator(x,A,b,exact_or_not,epsilon,CG_max_iter,CG_tol_rule)
    % 设置默认值
    if nargin < 6
        CG_tol_rule = 1;
        CG_max_iter = 1000;
    end
    if nargin < 5
        epsilon = 1e-6;
    end
    % 函数主体
    history = zeros(1001,2);
    i = 1;
    while i <= 1001
        object = Object(x,A,b);
        grad = Gradient(x,A,b);
        hess = Hessian(x,A,b);

        % 记录优化历史
        history(i,1) = object;
        history(i,2) = norm(grad);

        % 达到精度要求时停止迭代
        if history(i,2) < epsilon
            break;
        end

        % 精确 Newton 法和非精确 Newton 法的关键区别便是搜索方向的计算
        if exact_or_not
            direction = -hess\grad;
        else
            direction = CG(hess,grad,CG_max_iter,CG_tol_rule);
        end

        % 根据 Armijo 步长准则选取步长
        stepsize = Armijo(x,A,b,object,grad,direction);

        % 迭代公式
        x = x + stepsize * direction;
        fprintf('%d iteration done!\n',i);

        % 更新索引
        i = i + 1;
    end
end
```

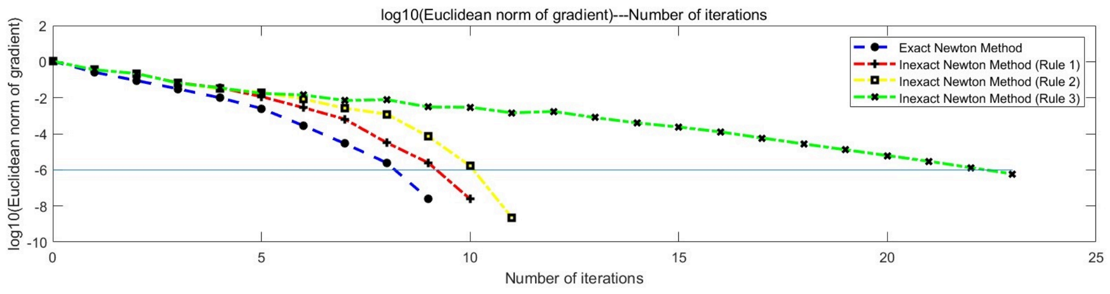
实验结果:

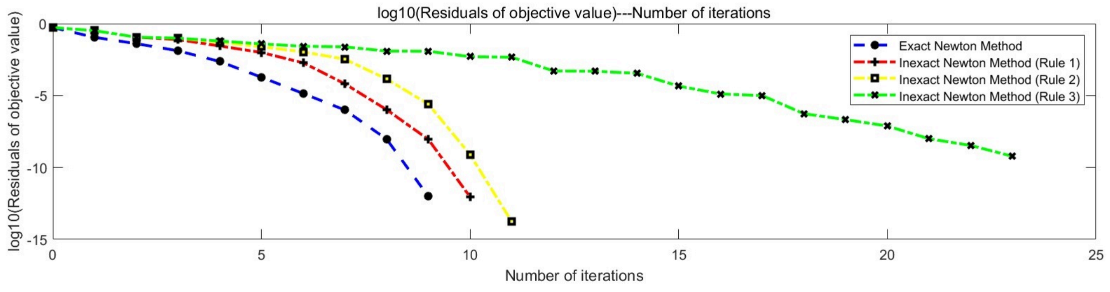
- a9a.test

  ```
  Dataset size:    (m,n) = (16281,122)
  Exact Newton:   0.31880 (7 iteration)
  Inexact Newton (Rule 1):    0.31880 (8 iteration)
  Inexact Newton (Rule 2):    0.31880 (10 iteration)
  Inexact Newton (Rule 3):    0.31880 (18 iteration)
  optimal:    0.31880
  ```

- CINA.test

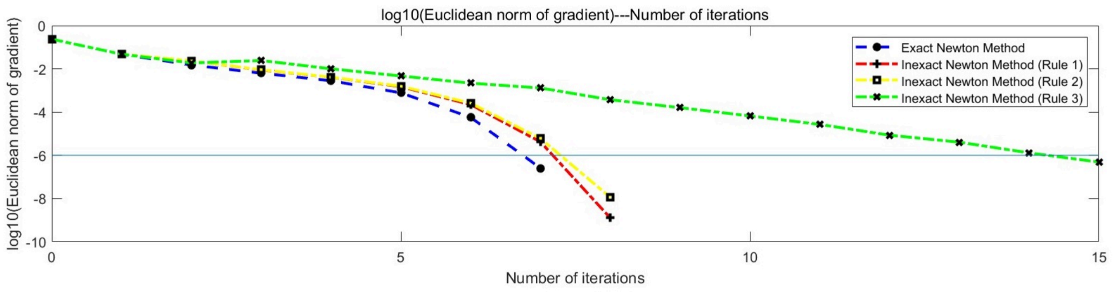
```
Dataset size:    (m,n) = (3206,132)
Exact Newton:   0.15931 (9 iteration)
Inexact Newton (Rule 1):    0.15931 (10 iteration)
Inexact Newton (Rule 2):    0.15931 (11 iteration)
Inexact Newton (Rule 3):    0.15931 (23 iteration)
optimal:    0.15931
```

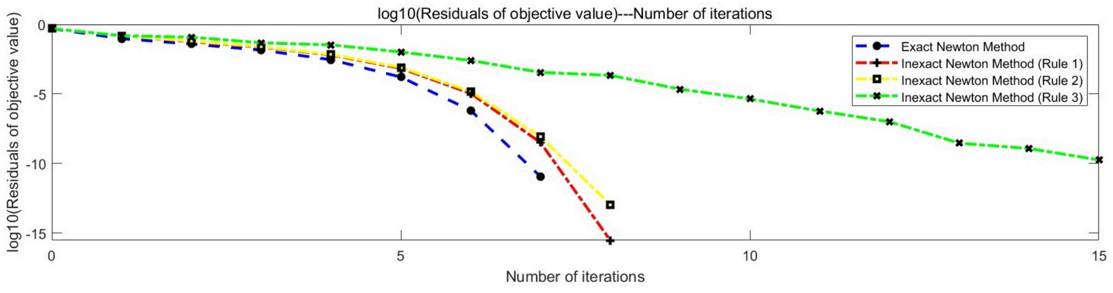




- ijcnn1.test

```
Dataset size:    (m,n) = (91701,22)
Exact Newton:   0.19566 (7 iteration)
Inexact Newton (Rule 1):    0.19566 (8 iteration)
Inexact Newton (Rule 2):    0.19566 (8 iteration)
Inexact Newton (Rule 3):    0.19566 (15 iteration)
optimal:    0.19566
```





(b) Implement gradient descent with Armijo rule line search and constant step size gradient descent with at least two small fixed step sizes such that the algorithms can converge.
Terminate your code with maximum iteration number 1000.
Plot the results in one figure for each dataset.

Solution:
梯度法：

- **搜索方向** $d^{(k)} = -\nabla f(x^{(k)})$

  ```
  direction = -grad;
  ```

- **步长准则：**
  - **固定步长：** 经过实验我们发现 0.05 和 0.10 两个步长比较合适
  - **Armijo 步长准则** (参考前文)

梯度法的迭代器代码如下：

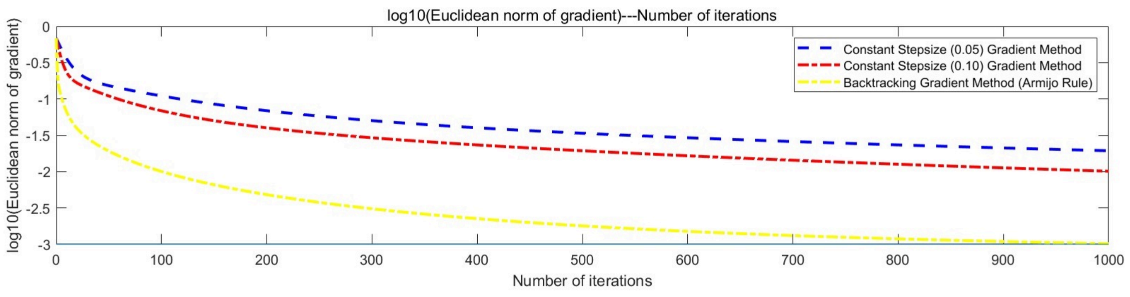
```
function [history,i] =
Gradient_Iterator(x,A,b,constant_stepsize,epsilon,max_iter)
    % 设置默认值
    if nargin < 6
        max_iter = 1000;
    end
    if nargin < 5
        epsilon = 1e-6;
    end
    if nargin < 4
        constant_stepsize = 0;
    end

    % 函数主体
    history = zeros(max_iter+1,2);
    i = 1;
    while i <= max_iter + 1
        object = Object(x,A,b);
        grad = Gradient(x,A,b);

        % 记录优化历史
        history(i,1) = object;
        history(i,2) = norm(grad);

        % 达到精度要求时停止迭代
        if history(i,2) < epsilon
            break;
        end

        % 梯度法搜索方向
        direction = -grad;

        % 若给定的固定步长为 0, 则根据 Armijo 步长准则选取步长
        if constant_stepsize == 0
            stepsize = Armijo(x,A,b,object,grad,direction);
        else
```

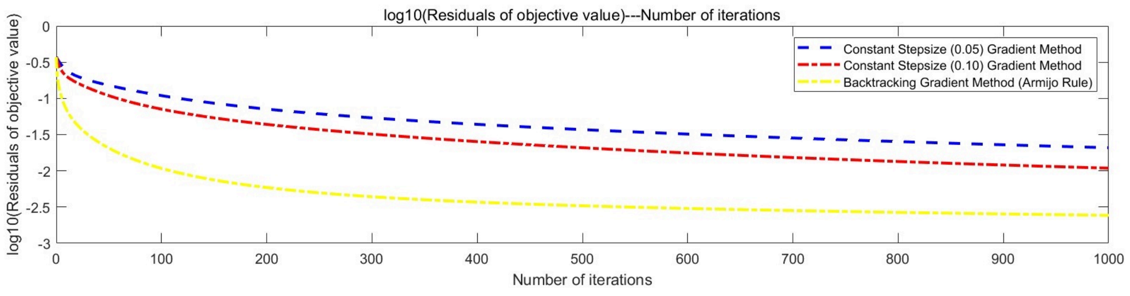
```
            % 若给定的固定步长不为 0, 则以其为固定步长
            stepsize = constant_stepsize;
        end

        % 迭代公式
        x = x + stepsize * direction;

        % 设置输出语句方便调试
        if mod(i,500)==0
           fprintf('Number of Iteration:\t%d\n',i);
        end

        % 更新索引
        i = i + 1;
    end
end
```

实验结果:

- a9a.test

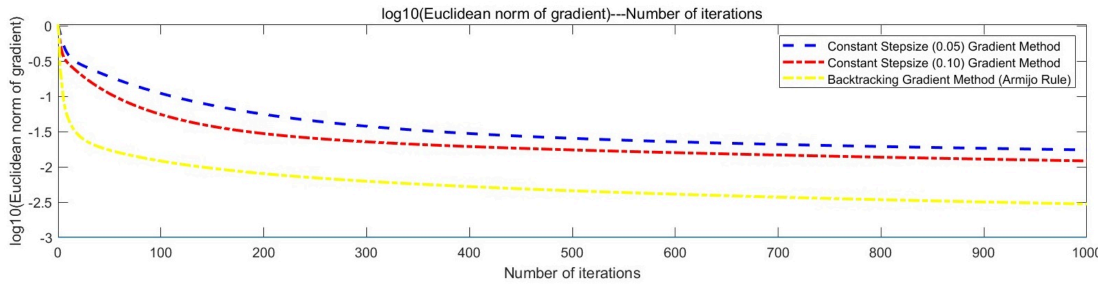
```
Dataset size:    (m,n) = (16281,122)
Constant Stepsize (0.05) Gradient Method:   0.33963 (1000 iteration)
Constant Stepsize (0.10) Gradient Method:   0.32968 (1000 iteration)
Backtracking Gradient Method (Armijo Rule): 0.32123 (1000 iteration)
optimal:    0.31880
```

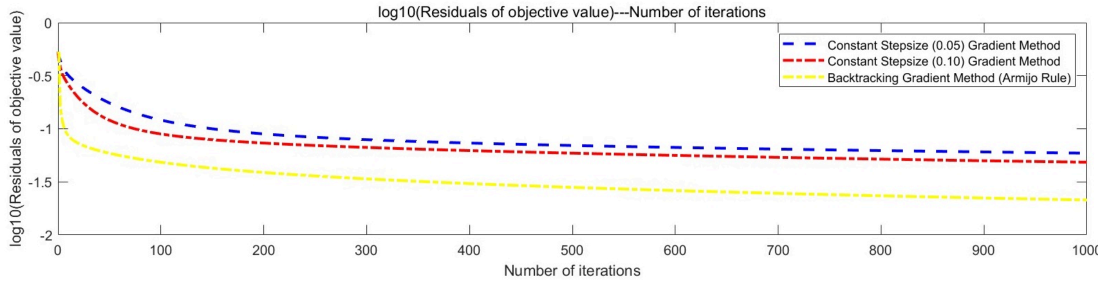




- CINA.test

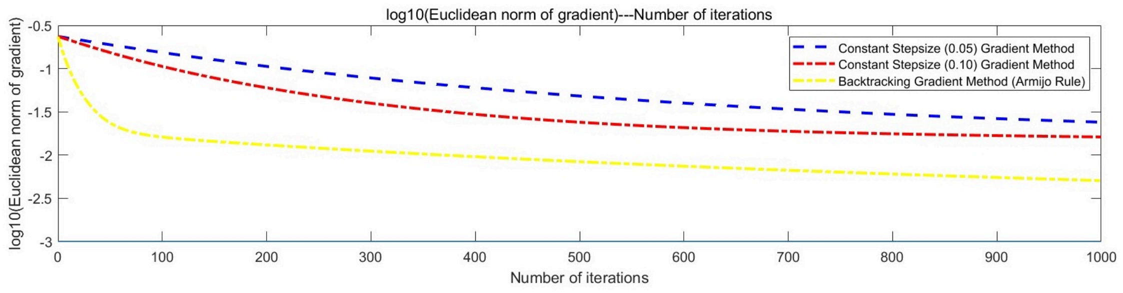
```
Dataset size:    (m,n) = (3206,132)
Constant Stepsize (0.05) Gradient Method:   0.21824 (1000 iteration)
Constant Stepsize (0.10) Gradient Method:   0.20774 (1000 iteration)
Backtracking Gradient Method (Armijo Rule): 0.18065 (1000 iteration)
optimal:    0.15931
```

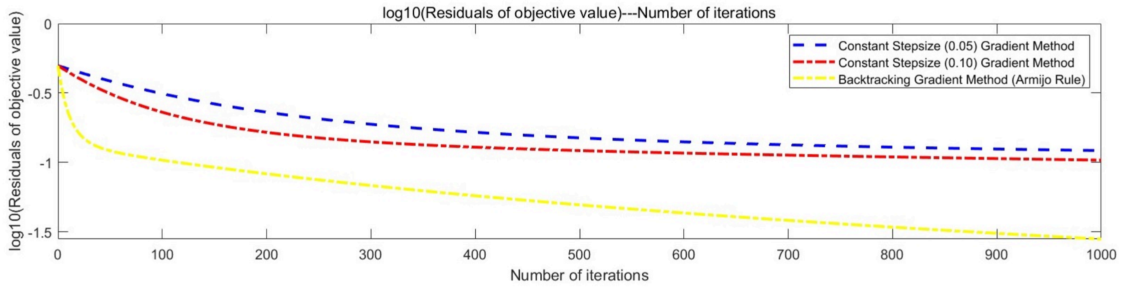




- ijcnn1.test

```
Dataset size:    (m,n) = (91701,22)
Constant Stepsize (0.05) Gradient Method:   0.31720 (1000 iteration)
Constant Stepsize (0.10) Gradient Method:   0.29934 (1000 iteration)
Backtracking Gradient Method (Armijo Rule): 0.22365 (1000 iteration)
optimal:    0.19566
```


log10(Euclidean norm of gradient)---Number of iterations
log10(Euclidean norm of gradient)
Number of iterations
Constant Stepsize (0.05) Gradient Method
Constant Stepsize (0.10) Gradient Method
Backtracking Gradient Method (Armijo Rule)


log10(Residuals of objective value)---Number of iterations
log10(Residuals of objective value)
Number of iterations
Constant Stepsize (0.05) Gradient Method
Constant Stepsize (0.10) Gradient Method
Backtracking Gradient Method (Armijo Rule)