中国科学技术大学

学士学位论文

Lasso 求解算法及其在 $\ell_0 TV$ 中的应用

姓　　名：	韩 冰 岩
院　　系：	数学科学学院
专　　业：	数学与应用数学
学　　号：	PB12203121
导　　师：	张举勇　副教授
完成时间：	二〇一六年五月

University of Science and Technology of China

A dissertation for bachelor's degree

Algorithms for Lasso and their applications in $\ell_0 TV$

Author : Bingyan Han

Department : School of Mathematical Sciences

Major : Mathematics and Applied Mathematics

Student ID : PB12203121

Supervisor : Prof. Juyong Zhang

Finished Time : May, 2016

致　谢

在本科的四年里，我所从事的学习和研究，得到了导师以及系里其他老师和同学的帮助和鼓励。在完成论文之际，请容许我对他们表达诚挚的谢意。

首先感谢导师张举勇老师半年来的指导和教诲，在张老师的指导下，我对最优化和图像处理领域有了更进一步的理解。不光是学到的知识本身，张老师治学的严谨态度也使我受益匪浅。

感谢杨周旺老师，他教授我运筹学这门课程，是杨老师的讲解使我对运筹学有了一定的了解。也感谢其他老师的亲切教导，让我能够在四年的本科生涯里学到很多知识。

感谢班主任夏晶老师和唐泽波老师多年的关怀。感谢科大，感谢一同走过来的同学们，在最宝贵年华里，是你们伴随着我的成长。

最后，感谢我家人一贯的鼓励和支持，你们是我追求学业的坚强后盾。

韩冰岩

2016 年 5 月 31 日

目 录

摘 要

本文探讨了 Lasso 和 ℓ_0TV 相关的算法。第一部分关注 Lasso 问题，回顾了 Lasso 的相关性质，然后介绍了近年来涌现出的一些重要算法，包括 GRock、Shotgun 等五个算法。通过在一些数据集上进行实验，对其中一些算法进行了比较。结果表明以坐标下降法为基础，利用并行方法的一些算法表现优异。第二部分关注 ℓ_0TV 的求解，本文尝试推广坐标下降法和并行方法到图像处理模型 ℓ_0TV 的求解上面，得到了一些结论，并探讨了一些未能解决的问题。

关键词： Lasso ℓ_0TV 坐标下降法 并行算法 图像处理

ABSTRACT

This thesis discusses some algorithms for solving the Lasso and ℓ_0TV problems. The first part focuses on the algorithms for solving Lasso . After reviewing some basic but important properties of Lasso, we introduce five famous algorithms in recent years,such as Shotgun and GRock. Then we select some datasets to test the implementations of these algorithms. The result is that algorithms based on coordinate descent and parallel methods outperform other algorithms. The second part is about solving the ℓ_0TV . We try to implement coordinate descent and parallel methods to solve the ℓ_0TV . We present some results we get and discuss some problems which still unsolved.

Keywords: Lasso,ℓ_0TV,Coordinate descent,Parallel algorithms,Image processing

第一章 Lasso 求解算法介绍

本章介绍与 Lasso 有关的算法。首先对 Lasso 的性质进行了简单的回顾。然后介绍了以下几个算法：基于活动集的 FAST-BCDA [11]，基于内点法的 L1_ LS [6]，以及引入了并行的 Shotgun [2]，p-FISTA [9]，GRock [9] 三个算法。最后从数值实验的角度上，给出了这些算法的一个比较，并且特别强调了在算法实现方面的一些重要细节。

第一节 Lasso 的提出及其性质

人们越来越多地接触到规模很大的数据，而且很多数据集具有高维的性质，即对一个样本点，常常可以观测并记录它的很多属性。在这些情况下，最小二乘回归要么没有办法解决，要么解并不理想。后来提出的岭回归较好的解决了共线性的问题，但是在属性的删选 (feature selection) 上表现不佳。通常有些属性没有解释因变量的变化，因而可以剔除掉。1996 年 Tibshirani 提出了 Lasso 模型（式 1.1）。和岭回归比起来，Lasso 的惩罚项使用的是 ℓ_1 范数，所求得的解有稀疏性，即 $\hat{\beta}$ 的很多项会是 0，这正是对属性做删选所需要的。Lasso 在统计和机器学习领域得到了广泛应用。

$$\min_{\beta\in\mathbb{R}^p} \frac{1}{2}\|X\beta - y\|^2 + \lambda\|\beta\|_1 \tag{1.1}$$

这里 $X \in \mathbb{R}^{n\times p}$ 是观测数据，n 个样本，每个样本有 p 个属性, λ 是惩罚系数。记该问题的解为 $\hat{\beta}$。

与直接使用 ℓ_0 范数做惩罚项相比，Lasso 依然具有凸性，便于我们寻找合适的求解算法。在正式介绍求解算法前，先看一下 Lasso 都有哪些性质。

1. Lasso 不一定只有一个解 $\hat{\beta}$, 因为当 $p > n$ 时，目标函数关于 β 不是严格凸的。所以在后面的表述中，$\hat{\beta}$ 可能指的是解的全体所构成的集合；

2. 拟合值 $X\hat{\beta}$ 是唯一的, 这是因为关于 $X\beta$ 是严格凸的；

3. 问题的 KKT 条件是，任何 Lasso 问题的解 $\hat{\beta}$ 都必须满足

$$X^T(y - X\hat{\beta}) = \lambda s, s \in \partial\|\hat{\beta}\|_1 \tag{1.2}$$

这里，

$$s_j \in \begin{cases} \{+1\} & \hat{\beta}_j > 0 \\ \{-1\} & \hat{\beta}_j < 0 \\ [-1, 1] & \hat{\beta}_j = 0 \end{cases} \tag{1.3}$$

由此我们可以得到一个重要结论，由于 $X\hat{\beta}$ 是唯一的，所以 $\hat{\beta}$ 每一项的符号是确定的。在这种意义下，我们也可以称，问题的解是唯一的。因为哪些属性会被选出来，这是不变的；

4. 定义 $E = \{j \in \{1, ..., p\} | \hat{\beta}_j \neq 0\}$。这样 KKT 条件就可以写成

$$X_E^T(y - X_E\hat{\beta}_E) = \lambda s_E \tag{1.4}$$

可以形式地写出 Lasso 问题的解，

$$\hat{\beta}_E = (X_E^T X_E)^+(X_E^T y - \lambda s_E) + \eta, \quad \hat{\beta}_{-E} = 0 \tag{1.5}$$

这里 $(X_E^T X_E)^+$ 是 Moore-Penrose 广义逆，$\eta \in \text{null}(X_E)$，于是也可以认为 $\hat{\beta}$ 是 λ 的函数，在后面的段落里，有时也会使用 $\hat{\beta}(\lambda)$ 这样的符号来强调这一点。对于每一个 $\lambda \in [0, \infty]$，都可以得到对应的解，我们称这些解构成了一条 regularization path，它详尽地刻画了在惩罚项和拟合项之间的权衡。

对 regularization path 的深入了解，产生了下面的算法。

当 $\lambda = \infty$ 时，显然最优解 $\hat{\beta} = 0$。当减小 λ 的取值时，$\hat{\beta}$ 逐渐开始有不为零的项出现。后面我们会看到，$\hat{\beta}(\lambda)$ 其实是一个关于 λ 连续，而且分段线性的函数。这意味着我们只要计算出在 regularization path 上某些节点 $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_r \geq 0$ 的值，其他点的值通过线性插值就可以得到。而这些关键的节点正好对应了集合 E 发生改变的时候。

为了描述算法的方便，本小节假设 X 每一列之间线性无关，这意味着原问题有唯一解。假设不成立的情形也可以类似地推导。

现在我们从 $\lambda = \infty$ 开始，这时 $\hat{\beta} = 0$。把它代入 KKT 条件的等式中可以得到

$$X^T y = \lambda s, s_j \in [-1, 1] \quad \text{所有} \quad j = 1, ..., p. \tag{1.6}$$

等式左边是一个取值固定的向量，而等式右边，在 λ 足够大时，可以满足等式成立。随着 λ 的减小，减到

$$\lambda_1 = \max_{j=1,...,p} |X_j^T y| \tag{1.7}$$

如果继续减下去，等式将不再成立。这时就需要将某个 β_j 取一个合适的非零值。定义 j_1 是使 $|X_j^T y|$ 取到最大值的下标，那么令

$$\begin{aligned} \hat{\beta}_{j_1}(\lambda) &= (X_{j_1}^T X_{j_1})^{-1}(X_{j_1}^T y - \lambda s_{j_1}),\ s_{j_1}\text{=sign}(X_{j_1}^T y) \\ \hat{\beta}_j(\lambda) &= 0, \quad \text{所有} \quad j \neq j_1 \end{aligned} \tag{1.8}$$

可以验证上式在 λ 进一步减小的某一段里，可以使 KKT 条件成立，即

$$X_{j_1}^T(y - X_{j_1}(X_{j_1}^T X_{j_1})^{-1}(X_{j_1}^T y - \lambda s_{j_1})) = \lambda s_{j_1} \tag{1.9}$$

并且

$$|X_j^T(y - X_{j_1}(X_{j_1}^T X_{j_1})^{-1}(X_{j_1}^T y - \lambda s_{j_1}))| \leq \lambda, \quad \text{所有 } j \neq j_1 \tag{1.10}$$

这也验证了之前关于 $\hat{\beta}(\lambda)$ 是一个关于 λ 连续，而且分段线性的函数的结论。

随着 λ 的继续减小，上面的不等式也开始出现不成立的情形，当 $X_j^T(y - X_{j_1}(X_{j_1}^T X_{j_1})^{-1}(X_{j_1}^T y - \lambda s_{j_1}))$ 等于 $\pm\lambda$ 时，是一个临界情形，类似的，可以定义

$$\lambda_2 = \max_{j \neq j_1, s_j \in \{-1,1\}}{}^{+} \frac{X_j^T(I - X_{j_1}(X_{j_1}^T X_{j_1})^{-1}X_{j_1})y}{s_j - X_j^T X_{j_1}(X_{j_1}^T X_{j_1})^{-1}s_{j_1}} \tag{1.11}$$

$\max^+$ 表示是在 $\lambda < \lambda_1$ 的条件下去求最大值。定义 j_2, s_2 是取到最大值的情形，令 $A = \{j_1, j_2\}, s_A = (s_{j_1}, s_{j_2})$，$s$ 其他项保持为零，令

$$\begin{aligned} \hat{\beta}_A(\lambda) &= (X_A^T X_A)^{-1}(X_A^T y - \lambda s_A) \\ \hat{\beta}_j(\lambda) &= 0, \quad \text{所有} \quad j \notin A \end{aligned} \tag{1.12}$$

以此类推，可以一直将 λ 减到想要的值，每一步都可以得到一个显式的更新公式。其中需要注意的是，要检查 $\hat{\beta}_A(\lambda)$ 中的每一项是否会穿过 0，因为在这时 s 就不是一个合理的符号向量了，需要做些修改。利用线性性质，可以给出相应的修改方法。

具体地，可以将算法描述如下:

Data: $y, X, k = 0, \lambda_0 = \infty, A = \emptyset, s_A = \emptyset$
while $\lambda_k > 0$ **do**

1. 从 λ_k 到下一个节点 λ_{k+1} 的解由下式计算：

$$\begin{aligned} \hat{\beta}_A(\lambda) &= (X_A^T X_A)^{-1}(X_A^T y - \lambda s_A) \\ \hat{\beta}_j(\lambda) &= 0, \quad \text{所有} \quad j \notin A \end{aligned} \tag{1.13}$$

2. 计算 λ_{k+1}^{hit}，即 A 需要改变的时候，$\max^+$ 是指在 $< \lambda_k$ 的范围内求最大值。

$$\lambda_{k+1}^{hit} = \max_{j \notin A, s_j \in \{-1,1\}}{}^{+} \frac{X_j^T(I - X_A(X_A^T X_A)^{-1}X_A)y}{s_j - X_j^T X_A(X_A^T X_A)^{-1}s_A} \tag{1.14}$$

3. 检查 λ_{k+1}^{cross}，即 $\hat{\beta}_A(\lambda)$ 中有变为 0 的情形，$\max^+$ 是指在 $< \lambda_k$ 的范围内求最大值。

$$\lambda_{k+1}^{cross} = \max_{j \in A}{}^{+} \frac{[(X_A^T X_A)^{-1}X_A^T y]_j}{[(X_A^T X_A)^{-1}s_A]_j} \tag{1.15}$$

4. 定义 $\lambda_{k+1} = \max\{\lambda_{k+1}^{hit}, \lambda_{k+1}^{cross}\}$
5. 如果 $\lambda_{k+1}^{hit} > \lambda_{k+1}^{cross}$，则将相应取到最值情形时的下标加入到 A 当中去, 并修改 s_A 相应项的正负号; 如果 $\lambda_{k+1}^{hit} < \lambda_{k+1}^{cross}$，则将相应取到最值情形时的下标从 A 当中剔除出去, 并修改 s_A 相应项为 0。
6. 更新 k=k+1.

end
Result: 得到 regualrization path

算法 1.1: Lasso Path Algorithm

Lasso path 算法常常被看成是 Lasso 版本的 least angle regression 算法。不过 least angle regression 是更一般的算法，详尽的讨论可以参看 [3] 。由于 Lasso path 算法每一步要计算矩阵的逆，所以计算负担会比较重。

第二节 FAST-BCDA

为了不引起混淆，我们先引入活动集 (active set) 的定义。

定义 1.2.1. *记 $\hat{\beta} \in \mathbb{R}^p$ 是式 1.1 的一个最优解，那么活动集就是*

$$\bar{\mathcal{A}}(\hat{\beta}) = \big\{i \in \{1, \dots, p\} : \hat{\beta}_i = 0\big\}. \tag{1.16}$$

非活动集就是:

$$\bar{\mathcal{N}}(\hat{\beta}) = I \setminus \bar{\mathcal{A}}(\hat{\beta}) = \big\{i \in \{1, \dots, p\} : \hat{\beta}_i \neq 0\big\}. \tag{1.17}$$

从活动集的观点来看，上一节给出的算法就是一种判定下标 i 进出活动集的一个方法。关于活动集的研究并没有终止，这一节将介绍 M.D.Santis et al. 在 2016 年发表的论文 [11]，这篇文章给出了活动集的一种新的估计方法。

为了表述方便，在这一节我们引入关于 式 1.1 的定义 $g(\beta)$ 和 H:

$$g(\beta) = X^T(X\beta - y), \quad H = X^T X.$$

利用这些符号可以将 $\hat{\beta} \in \mathbb{R}^p$ 是式 1.1的最优解的条件表示为，当且仅当式 1.18 成立。

$$\begin{cases} \hat{\beta}_i > 0, & g_i(\hat{\beta}) + \lambda = 0 \\ \hat{\beta}_i < 0, & g_i(\hat{\beta}) - \lambda = 0 \\ \hat{\beta}_i = 0, & -\lambda \leq g_i(\hat{\beta}) \leq \lambda. \end{cases} \tag{1.18}$$

每一个采用活动集思想的算法，最重要的一步都是估计哪些下标 i 在活动集里面。为了给出活动集的估计，我们将 Lasso(式 1.1) 等价地写成:

$$\begin{aligned} \min_{u,v} \quad & \tfrac{1}{2}\|X(u-v) - y\|^2 + \lambda \textstyle\sum_{i=1}^{p}(u_i + v_i) \\ & u \geq 0, \\ & v \geq 0, \end{aligned} \tag{1.19}$$

这里 $u, v \in \mathbb{R}^p$。记这个问题的最优解是 $(u^\star, v^\star) \in \mathbb{R}^p \times \mathbb{R}^p$。之所以称这两个问题等价，是因为这两个问题的解之间存在 式 1.20 的转化关系。

$$\begin{aligned} u^\star &= \max(0, \hat{\beta}), \\ v^\star &= \max(0, -\hat{\beta}), \\ \hat{\beta} &= u^\star - v^\star. \end{aligned} \tag{1.20}$$

可以知道 式 1.19 的 Lagrangian 函数是：

$$\mathcal{L}(u,v,\phi,\mu)=\frac{1}{2}\|X(u-v)-y\|^2+\lambda\sum_{i=1}^{p}(u_i+v_i)-\phi^T u-\mu^T v,$$

其中 $\phi,\mu\in\mathbb{R}^p$ 是 Lagrangian 乘子。

如果令 $(u^\star,v^\star,\phi^\star,\mu^\star)$ 是 Lagrangian 函数的最优解，那么最优解需要满足：

$$\begin{aligned}\phi_i^\star &= g_i(u^\star-v^\star)+\lambda\ =g_i(\hat{\beta})+\lambda;\\ \mu_i^\star &= \lambda\ -g_i(u^\star-v^\star)=\lambda\ -g_i(\hat{\beta}).\end{aligned}\tag{1.21}$$

从 式 1.21 可以定义 ϕ,μ 和 u,v 之间有以下关系：

$$\begin{aligned}\phi_i(u,v) &= g_i(u-v)+\lambda;\\ \mu_i(u,v) &= \lambda\ -g_i(u-v).\end{aligned}\tag{1.22}$$

利用这两个等式，结合参考文献 [4] 的思想，可以给出非活动集 $\mathcal{N}(u,v)$ 和活动集 $\mathcal{A}(u,v)$ 的估计:

$$\begin{aligned}\mathcal{N}(u,v) &= \{i:u_i>\epsilon\,\phi_i(u,v)\}\cup\{i:v_i>\epsilon\,\mu_i(u,v)\},\\ \mathcal{A}(u,v) &= I\setminus\mathcal{N}(u,v).\end{aligned}\tag{1.23}$$

这里 ϵ 是一个正的标量。为了保证算法的收敛性，参数 ϵ 需要满足 $0<\epsilon<\frac{1}{\lambda_{max}(X^TX)}$，其中 $\lambda_{max}(\cdot)$ 表示矩阵的最大特征值。

式 1.23 给出的估计还是用 u,v 表示的，需要再利用式 1.24 进行转化得到最终结果。

$$u=\max(0,\beta)\qquad 和\qquad v=\max(0,-\beta),\tag{1.24}$$

定义 1.2.2. 取 $\beta\in\mathbb{R}^p$。定义非活动集和活动集的估计分别是

$$\mathcal{N}(\beta)=\{i:\max(0,\beta_i)>\epsilon\,(\lambda+g_i(\beta))\}\cup\{i:\max(0,-\beta_i)>\epsilon\,(\lambda-g_i(\beta))\},\tag{1.25}$$

$$\mathcal{A}(\beta)=I\setminus\mathcal{N}(\beta).\tag{1.26}$$

下面来理解一下新的估计方法究竟是限定了什么样的条件，允许哪些 i 被纳入到活动集里面。假设迭代到了第 k 步，β^k 是相关的迭代结果，而 $i\in I$ 被认为是属于活动集的下标，也就是

$$i\in\mathcal{A}(\beta^k)=\{i:\max(0,\beta_i^k)\le\epsilon\,(\lambda+g_i(\beta^k))\}\cap\{i:\max(0,-\beta_i^k)\le\epsilon\,(\lambda-g_i(\beta^k))\},$$

可以观察到，这个约束能等价地写成

$$\beta_i^k\in[\epsilon(g_i(\beta^k)-\lambda),\epsilon(g_i(\beta^k)+\lambda)]\quad 且\quad -\lambda\le g_i(\beta^k)\le\lambda,\tag{1.27}$$

由此可以对我们的估计给出一个直观的解释，也就是那些满足 β_i^k 充分小而且与 0 相对应的最优性条件也满足时，i 就会被纳入活动集。

另外，除了要鉴别出哪些下标在活动集里面，我们还要对非零的部分进行修改，使得目标函数值能够进一步下降。

算法 1.2 给出了具体步骤。这里面用到几个记号，给定一个矩阵 $Q \in \mathbb{R}^{n\times n}$，定义指标集 $I = \{1, \ldots, n\}$，使用 $Q_{I_jI_j}$ 来表示 Q 的子矩阵，这个子矩阵的行和列的下标都在 $I_j \subseteq I$。

另一个是 $\bar{\mathcal{N}}^k_{ord}$。首先定义一个连续函数 $\Phi_i(\beta)$ 来衡量在 β_i 处对最优性的偏离程度，也就是

$$\Phi_i(\beta) = -\text{mid}\left\{\frac{g_i(\beta)-\lambda}{H_{ii}}, \beta_i, \frac{g_i(\beta)+\lambda}{H_{ii}}\right\}, \tag{1.28}$$

这里 mid$\{a, b, c\}$ 指的是 $a,\ b,\ c$ 的中位数。

根据 $\Phi_i(\beta)$ 的值，递减地排列非活动集 $\mathcal{N}^k$ 中的下标 i，将其中前 s 个最大的下标构成的集合记为 $\bar{\mathcal{N}}^k_{ord}$。然后再将 $\bar{\mathcal{N}}^k_{ord}$ 分成 q 个子集 $I_1, ..., I_q$，每个子集有 r 个下标。算法会分块求解对应的子问题。

Data: $\beta^0 \in \mathbb{R}^n$, 令 $k = 0$.
for $k = 0, 1 \ldots$ **do**
  按照定义 1.2.2 计算 $\mathcal{A}^k, \mathcal{N}^k$, 排序得到 $\bar{\mathcal{N}}^k_{ord}$
  令 $x^{0,k}_{\mathcal{A}^k} = 0$, $x^{0,k}_{\mathcal{N}^k} = \beta^k_{\mathcal{N}^k}$
  for $j = 1, \ldots, q$ **do**
    求 $x^{j,k}_{I_j}, I_j \subseteq \bar{\mathcal{N}}^k_{ord}$, 即下面子问题的解：

$$\min_{w\in\mathbb{R}^r} g_{I_j}(x^{j-1,k})^T(w - x^{j-1,k}_{I_j}) + \frac{1}{2}(w - x^{j-1,k}_{I_j})^T H_{I_jI_j}(w - x^{j-1,k}_{I_j}) + \lambda\|w\|_1 \tag{1.29}$$

    如果 $i \notin I_j$, 则令 $x^{j,k}_i = x^{j-1,k}_i$;
  end
  令 $\beta^{k+1} = x^{q,k}$;
end

算法 1.2: FAST-BCDA

算法的收敛性能够成立，依赖于两个结果，一个是活动集的变化，它通过把活动集里面的对应项设置为 0 来使得目标函数有足够的下降。另一个结果是，在数据集 X 满足一定条件时，即使只对一个子集的变量进行极小化，同时固定其他项不变，也可以使目标函数值有所下降。

记 $\{\beta^k\}$ 是算法产生的结果序列。可以证明，在假设条件成立时，或者存在一个整数 $\bar{k} \geq 0$ 使得 $\beta^{\bar{k}}$ 是 式 1.1 的最优解, 或者序列 $\{\beta^k\}$ 有无限项，它每一个极限点都是 式 1.1 的最优解。

第三节 L1_LS

Kim et al. [6] 的算法属于内点法，同时还利用了牛顿法、PCG 等等，是一个从凸优化的角度提出的算法。

Lasso (式 1.1) 的对偶问题是

$$\begin{aligned} &\max G(\nu) = -(1/2)\nu^T\nu - \nu^T y \\ &\text{subject to} \quad |(X^T\nu)_i| \le \lambda, \quad i = 1, ..., p. \end{aligned} \tag{1.30}$$

由于原问题满足 Slater 条件,strong duality 成立，即原问题和对偶问题的目标函数最优值相等。

令

$$\eta = \frac{1}{2}\|X\beta - y\|^2 + \lambda\|\beta\|_1 - G(\nu) \tag{1.31}$$

称 η 为 dual gap 。

对于任何一个 β, 都可以建立一个对偶问题的可行点如下：

$$\begin{aligned} &\nu = \sqrt{2}s(X\beta - y) \\ &s = \min\{\frac{\lambda}{\sqrt{2}|(X^T(X\beta - y))_i|} \quad | \quad i = 1, ..., n\} \end{aligned} \tag{1.32}$$

然后就可以计算 η 得到 x 离最优解的近似程度，这样就能给出算法的停机准则。

原问题可以等价转化为式 1.33，其中 u, β 都是变量。

$$\begin{aligned} &\min \frac{1}{2}\|X\beta - y\|^2 + \lambda\sum_{i=1}^{p} u_i \\ &\text{subject to} - u_i \le \beta_i \le u_i, i = 1, ..., p \end{aligned} \tag{1.33}$$

内点法通过引入一些边界的惩罚函数，使得新问题的解能差不多满足原问题的约束条件。此处定义,

$$\Phi(\beta, u) = -\sum_{i=1}^{p}\log(u_i + \beta_i) - \sum_{i=1}^{p}\log(u_i - \beta_i) \tag{1.34}$$

然后求解

$$\phi_t(\beta, u) = t(\frac{1}{2}||X\beta - y||^2 + \lambda\sum_{i=1}^{p} u_i) + \Phi(\beta, u) \tag{1.35}$$

这里的 t 是用来调节和问题 (1.33) 的近似程度，如果 (β, u) 满足边界条件的话，t 取的大，对原问题的近似程度更好。但一开始 t 取的太大，利用牛顿法求解时，会发现 $\phi_t(\beta, u)$ 的 Hessian 阵在可行集的边界处变化剧烈。所以恰当的解决方法是，对一列逐渐增大的 t 求解，并用前一个较小的 t 的解作为下一步求解

问题的起点。记上面问题的最优解为 $(\beta^*(t), u^*(t))$。然后求得 $\beta^*(t)$ 对应的对偶可行点 $\nu^*(t) = \sqrt{2}(X\beta^*(t) - y)$, 可以证明下面不等式成立:

$$\eta = \frac{1}{2}||X\beta^* - y||^2 + \lambda||\beta^*||_1 - G(\nu^*) \leq \frac{Cp}{t} \tag{1.36}$$

其中 C 是一个常数。也就是说，通过增加 t 的值，可以保证逼近最优解。

那么，$\phi_t(\beta, u)$ 的求解使用什么方法呢？答案是牛顿法。

通常找下降方向，需要求解下面的方程

$$\nabla^2\phi_t(\beta, u)[\Delta\beta \quad \Delta u]^T = -\nabla\phi_t(\beta, u) \tag{1.37}$$

但 式 1.37 往往不易求得精确解，但可以用 Preconditioned Conjugate Gradient 方法求一个近似的下降方向，减小这一步的计算负担，然后再求步长。当 $||\nabla\phi_t||_2$ 的值很小时，就认为求得了 $(\beta^*(t), u^*(t))$。Matlab 里面有函数 pcg 来实现 PCG。Kim et al. 的代码也利用了这个函数，这在后面的实验中有所体现。

Data: $X, t = 1/\lambda, \beta = 0, u = (1, 1, ..., 1) \in \mathbb{R}^p$
while $\eta/G(\nu)$ 没有充分小 **do**
    更新 t。
    利用 PCG 给出 式 1.37 的近似解，即搜索方向 $(\Delta\beta, \Delta u)$。
    利用线性搜索 (line search) 的方法计算步长 s。
    更新 $(\beta, u) = (\beta, u) + s(\Delta\beta, \Delta u)$。
    利用 式 1.32 计算对偶可行点 ν。
    利用 式 1.31 计算 η。
end

算法 1.3: L1_LS

从数值实验来看，这个算法很稳健，求出的 β, 往往会 100% 非零，尽管其中很大一部分非常接近于 0。这是它最重要的缺点。

第四节　p-FISTA

从这一节开始介绍的算法都引入了并行的理念。p-FISTA 表示的是 parallel-FISTA 。而为了介绍 p-FISTA，首先要介绍 FISTA [1]。同时也要介绍 iterative shrinkage-thresholding algorithm (ISTA)。

可以考虑比 Lasso 更一般一点的模型，式 1.38，

$$\min_{\beta\in\mathbb{R}^p} \frac{1}{2}\|X\beta - y\|^2 + \lambda c(\beta) \tag{1.38}$$

这里要求惩罚项 $c(\cdot)$ 是凸的。而被称作 shrinkage function 的就是

$$\mathcal{T}_\lambda(u) = \arg\min_z \frac{1}{2}\|z - u\|^2 + \lambda c(z) \tag{1.39}$$

当 $c(z)=\|z\|_1$ 时，$\mathcal{T}_\lambda(z)=\text{soft}(z,\lambda)=\text{sign}(z)\cdot(|z|-\lambda)_+$ 。

ISTA 给出的求解 式 1.38 的迭代步骤就是

$$\beta_{k+1}=\mathcal{T}_{\lambda t_k}(\beta_k-t_kX^T(X\beta_k-y)) \tag{1.40}$$

其中 t_k 是一个合理的步长。有时 X^TX 可以快速地计算，例如使用 FFT 。这可以提高 ISTA 的计算性能。

下面来理解 ISTA 是如何提出的。一个非常著名的观察是，ISTA 和梯度方法 (gradient methods) 有很大的关系，ISTA 可以看成是梯度方法的一种推广。

首先考虑一个无约束最优化问题，

$$\min_{x\in\mathbb{R}^n} f(x) \quad ,f \quad \text{连续可微} \tag{1.41}$$

如果用梯度方法解决这个问题，那么迭代序列是这样产生的:

$$\begin{aligned} x_0 &\in \mathbb{R}^n \\ x_k &= x_{k-1}-t_k\nabla f(x_{k-1}) \end{aligned} \tag{1.42}$$

而上面的等式可以从 proximal 方法的观点来重新阐释，因为可以将式 1.42 改写成符合 proximal algorithm 的形式，即 式 1.43。

$$x_k=\arg\min_x\{f(x_{k-1})+\langle x-x_{k-1},\nabla f(x_{k-1})\rangle+\frac{1}{2t_k}\|x-x_{k-1}\|^2\} \tag{1.43}$$

下面对带有 ℓ_1 范数的问题，推广式 1.43。即如果求解

$$\min_{x\in\mathbb{R}^n} f(x)+\lambda\|x\|_1 \tag{1.44}$$

假定可以使用

$$\begin{aligned} x_k &= \arg\min_x\{f(x_{k-1})+\langle x-x_{k-1},\nabla f(x_{k-1})\rangle+\tfrac{1}{2t_k}\|x-x_{k-1}\|^2+\lambda\|x\|_1\} \\ &= \arg\min_x\{\tfrac{1}{2t_k}\|x-(x_{k-1}-t_k\nabla f(x_{k-1})\|^2+\lambda\|x\|_1\} \end{aligned} \tag{1.45}$$

第二个等式成立的原因是常数项可以忽略。

而 ℓ_1 范数是可分的，上面的问题又可以化成一维的问题，给出求解的方法

$$x_k=\mathcal{T}_{\lambda t_k}(x_{k-1}-t_k\nabla f(x_{k-1})) \tag{1.46}$$

正因为如此，我们才会说 ISTA 可以看成是梯度方法的一种推广。

ISTA 有 $O(1/k)$ 的收敛速度。下面要介绍的 FISTA 有 $O(1/k^2)$ 的收敛速度。

可以考虑更一般的模型：

$$\min\{F(x)=f(x)+g(x):x\in\mathbb{R}^n\} \tag{1.47}$$

我们需要以下假定：

1. $g:\mathbb{R}^n\to\mathbb{R}$ 是一个连续的凸函数，它有可能不光滑；

2. $f:\mathbb{R}^n\to\mathbb{R}$ 是 $C^{1,1}$ 的凸函数，也就是连续可微，并且满足:

$$\|\nabla f(x)-\nabla f(y)\|\le L(f)\|x-y\|,\quad x,y\in\mathbb{R}^n \tag{1.48}$$

$\|\cdot\|$ 是欧几里得范数，$L(f)$ 是 ∇f 的 Lipschitz 常数。当 $f(x)=1/2\|Ax-y\|^2$ 时，L 的最小值是 $\lambda_{\max}(A^TA)$。

3. 原问题是有解的。

首先介绍一个结果，

定理 1.4.1. *$f:\mathbb{R}^n\to\mathbb{R}$ 是 $C^{1,1}$ 的凸函数，$L(f)$ 是 ∇f 的 Lipschitz 常数, 那么对任意的 $L\ge L(f)$, 有*

$$f(x)\le f(y)+\nabla f(y)^T(x-y)+\frac{L}{2}\|x-y\|^2 \tag{1.49}$$

证明. 由假设条件可得 $h(x)=\frac{L}{2}x^Tx-f(x)$ 是凸函数，而 $h(x)$ 是凸函数等价于

$$f(x)\le f(y)+\nabla f(y)^T(x-y)+\frac{L}{2}\|x-y\|^2 \tag{1.50}$$

□

有了这个结论之后，我们会看到，可以给出目标函数 $F(x)=f(x)+g(x)$ 的一个近似

$$Q_L(x,y)=f(y)+\nabla f(y)^T(x-y)+\frac{L}{2}\|x-y\|^2+g(x) \tag{1.51}$$

对 $Q_L(x,y)$ 求解要简单一些，可得

$$\begin{aligned} p_L(y) &= \arg\min_{x\in\mathbb{R}^n} Q_L(x,y)\\ &= \arg\min_x\{g(x)+\tfrac{L}{2}\|x-(y-\tfrac{1}{L}\nabla f(y))\|^2\}\end{aligned} \tag{1.52}$$

ISTA 的基本步骤也就可以概括为

$$x_k=p_L(x_{k-1}) \tag{1.53}$$

那么求解 $Q_L(x,y)$ 和求解原问题的差异有多大呢？下面的定理给出了答案。

定理 1.4.2. *取 $y\in\mathbb{R}^n$ 且 $L>0$，使得*

$$F(p_L(y))\le Q(p_L(y),y) \tag{1.54}$$

成立。那么对任意的 $x\in\mathbb{R}^n$, 有

$$F(x)-F(p_L(y))\ge\frac{L}{2}\|p_L(y)-y\|^2+L\langle y-x,p_L(y)-y\rangle. \tag{1.55}$$

Data: 取 $L_0 > 0, \eta > 1, x_0 \in \mathbb{R}^n$, 设初始值 $y_1 = x_0, t_1 = 1$

while 不收敛 **do**

找到最小的非负整数 i_k 使得对于 $\hat{L} = \eta^{i_k} L_{k-1}$ 有

$$F(p_{\hat{L}}(y_k)) \leq Q_{\hat{L}}(p_{\hat{L}}(y_k), y_k)$$

令 $L_k = \eta^{i_k} L_{k-1}$, 计算

$$\begin{aligned} x_k &= p_{L_k}(y_k), \\ t_{k+1} &= \frac{1+\sqrt{1+4t_k^2}}{2}, \\ y_{k+1} &= x_k + (\frac{t_k - 1}{t_{k+1}})(x_k - x_{k-1}) \end{aligned} \tag{1.56}$$

end

Result: x,y

算法 1.4: FISTA

由 定理 1.4.1 可以知道，当 $L \geq L(f)$ 时，式 1.54 就会永远成立。算法会用到 $y = x$ 的情形，代入式 1.55中可以看到，求解 $Q_L(x, y)$ 得到的结果，也会使原问题目标函数值下降。

算法1.4给出了具体步骤，注意到实际应用时，$L(f)$ 不一定很好求，所以要考虑 $L(f)$ 不知道的情形。

算法里面使用了 Nestrov 的加速方法 [8]。另一方面，我们也可以看到，算法里面对 p_L 子问题的求解可以并行地去做。这在的 Z. Peng et al. 的文章 [9] 里面得到了实现，得到的并行算法称为 p-FISTA。

第五节 Shotgun

随着数据集合的规模越来越大，以前的算法在求解速度和解的稀疏程度上有一些不足。近年来，研究人员将并行的方法应用到 Lasso 的求解上，期望能够在求解速度上得到提升，并且求得的解依然保持稀疏性。

Shotgun [2] 就是利用并行和随机的方法，通过对坐标下降法进行修改，得到的一种求解方法。原来的坐标下降法，通过固定大部分变量，只对其它小部分变量进行求解。坐标下降法思想上非常简单，但性质还是十分优异的。

具体到 Lasso 问题，由坐标下降法得到的算法又被成为 Shooting 算法。

利用 subdifferential ，可以得到

$$\begin{aligned} \partial f(\beta_j) &= \partial_j(\tfrac{1}{2}\|X\beta - y\|^2 + \lambda\|\beta\|_1) \\ &= \partial_j(\tfrac{1}{2}\beta^T X^T X\beta - y^T X\beta + \tfrac{1}{2}y^T y + \lambda\textstyle\sum_j |\beta_j|) \end{aligned} \tag{1.57}$$

利用 X^TX 的对称性，记 $A_{(j,:)}$ 为矩阵 A 的第 j 行，可以得到

$$\partial_j f = \begin{cases} (X^TX)_{(j,:)} \cdot \beta - (y^TX)_j + \text{sign}(\beta_j)\lambda & \beta_j \neq 0 \\ (X^TX)_{(j,:)} \cdot \beta - (y^TX)_j + [-\lambda, \lambda] & \beta_j = 0 \end{cases} \tag{1.58}$$

记 $S_j = (X^TX)_{(j,:)} \cdot \beta - (X^TX)_{(j,j)}\beta_j - (y^TX)_j$, 则

$$\hat{\beta}_j = \begin{cases} \frac{\lambda - S_j}{(X^TX)_{(j,j)}} & , S_j > \lambda \\ \frac{-\lambda - S_j}{(X^TX)_{(j,j)}} & , S_j < -\lambda \\ 0 & .\text{其它} \end{cases} \tag{1.59}$$

上面的等式其实和最优性判据一样，Shooting 算法就是通过给出一个初始的 β，迭代地更新 β，直到收敛。

算法 1.5给出了 Shooting 算法的具体步骤。

令 $\beta = 0 \in \mathbb{R}^p$
while 没有收敛 **do**
  1. 随机地从 $\{1, ..., p\}$ 选一个下标 j.
  2. 利用 式 1.59 更新 β_j.
end

算法 1.5: Shooting

后来 Friedman et al. [5] 引入了 pathwise coordinate descent。这个方法就是通过在 Shooting 的外面再加一层循环。我们知道，当 λ 充分大的时候，最优解是 $\hat{\beta} = 0$，先从这个充分大的 λ 开始，求解最优解，然后逐步减小 λ，对每一个小的 λ 利用 Shooting 算法进行求解。然后将得到的结果作为下一个更小的 λ 对应的初始值，这种方法称为 warm start 。可以证明这样能提高收敛速度。

而 Shotgun 的方法就是，将 β 分成 M 块，对每一块同时分别进行求解。

将 β 分成 M 个集合：$\beta_1, \beta_2, ..., \beta_M$
while $\lambda > \lambda_{min}$ **do**
  并行地在 M 个处理器上运行 $Shooting(\beta_m, \beta_0, X, \lambda)$ 算法，其中 β_0 是初始值，β_m 是第 m 个进程负责更新的部分。直到所有处理器上都收敛。
  按对数方式减小 λ，并以上一步结果作为初始值。(warm start)
end

算法 1.6: M 个处理器的 Shotgun

如果不对数据集 X 加什么条件的话，Shotgun 很有可能不收敛。直观的说，如果 X 各列相关性不强的话，Shotgun 收敛的可能性比较大。

第六节 GRock

GRock 是 greedy coordinate-block descent method 的缩写。与 Shotgun 使用随机选取下标的方法不同的是，GRock 要对下标按照某种判断指标来筛选一下，然后再进行更新。原论文 [9] 对更一般的模型进行了考察，这里仅局限于对 Lasso 的看法。

首先注意到惩罚项是可分的，即 $\|\beta\|_1 = \sum_{i=1}^{p} |\beta_i|$，这对于并行化有很大帮助。然后记

$$g(\beta) = X^T(X\beta - y) = [g_1, g_2, ..., g_p]^T.$$

而

$$\frac{1}{2}\|X(\beta + \Delta\beta) - y\|^2 \leq \frac{1}{2}\|X\beta - y\|^2 + g^T\Delta\beta + \frac{1}{2}(\Delta\beta)^T X^T X\Delta\beta.$$

为了分析简便，假定 X^TX 的对角元为 1。对每一个下标 i，定义它的势 (potential) 为

$$d_i = \arg\min_{d\in\mathbb{R}} \lambda \cdot |\beta_i + d| + g_i d + \frac{d^2}{2}. \tag{1.60}$$

记 d_i 组成的向量为 $\mathbf{d} = [d_1, d_2, ..., d_p]^T$。下面我们考虑将数据集按列分成 N 块的情形，即 $X = [X_1, X_2, ..., X_N]$。对于第 j 块, 选出这一块里面 $|d|$ 最大的那一个，记对应的最大值为 m_j，对应的下标为 s_j, 即 $m_j = |d_{s_j}|$。这样我们就有 N 个局部的最大值。每一步的迭代，就对这 N 个局部最大值排序，从当中选出最大的 P 个，对他们对应的 β_{s_j} 进行更新。下面就可以给出算法了。

Data: 数据集 X, 观测值 y, 初始值 $\beta = 0$
while 没有收敛 **do**
  1. 按 d_i 的定义计算每一个分块的 $\mathbf{d}_j$ 向量。
  2. 为每一块找到 m_j, s_j。
  3. 记 N 个局部最大值里，最大的 P 个对应的下标构成的集合为 $\mathcal{P}$ 。
  4. 按照 $\beta_{s_j} \leftarrow \beta_{s_j} + d_{s_j}$ 对每一个 $j \in \mathcal{P}$ 更新相应的值。
end
Result: β

算法 1.7: GRock

如果不对原始的数据集 X 附加一定条件的话，GRock 有可能只在 $P = 1$ 的情形成立。为了证明 $P > 1$ 时的收敛性质，要引入一些条件。

定义矩阵的分块谱半径是

$$\rho_P = \max_{M\in\mathcal{M}} \rho(M) \tag{1.61}$$

$\mathcal{M}$ 是由 X^TX 所有 $P \times P$ 的子矩阵构成的集合，$\rho(M)$ 是一个矩阵的谱半径。

直观的感觉是，如果 X 各个分块的列不相关的话，那 GRock 就可能收敛。反映到 ρ_P 上就是限制它不要太大，当 ρ_P 不太大时，GRock 有 $O(1/k)$ 的收敛性。

而 GRock 有较好的并行性，算法中的第 1 步和第 2 步都可以并行地进行，这两步也是最耗费时间的。

第七节　Lasso 求解算法数值实验

这一部分是对前面的算法进行实验，并做比较。

值得注意的是，比较算法的表现往往比理解算法更为困难。由于种种原因，很难用一个公平而又统一的标准去比较不同的算法。因为比较算法的性能要注意以下问题：

1. 编程问题，不同算法使用不同语言来实现，使用不同的第三方库，这会显著影响性能。p-FISTA、GRock 使用 C 语言编写，用了 GSL 来解决数学计算的问题，用 MPI 解决不同进程间通信和同步的问题，后面会看到使用 GSL 的数据结构有一定的限制。Shotgun 使用 OpenMP 来实现多线程，在对共享数据的保护上，使用汇编语言自己编写了原子操作 CAS (compare and swap), 有效降低了因竞争互斥锁而引起的阻塞时间。L1_LS 利用 Matlab 来实现相应算法，而且可以利用 Matlab 内置的函数，比如 pcg , 通常认为内置函数都是很快的。

2. 有的算法，比如 Shotgun，使用 warm start，pathwise coordinate descent 还可以输出中间问题的解 (有可能精度不是非常高)，如果把它和只对一个惩罚系数求解的算法作比较，显得”不公平”。p-FISTA 还计算了 X^TX 的谱半径，这一步也非常消耗时间。

3. 算法使用的停机准则不同，最后求得的解一般不一样。有的算法比较的是两次迭代之间目标函数下降了多少，如果下降幅度低于某一个阈值，算法停止。有的算法使用固定的迭代次数，显得更加随意一点。不难想象，这些算法最后求得的解一般不一样。

4. 对数据集合的要求不同。利用并行的算法诸如 p-FISTA，GRock，Shotgun 都要求数据集的列与列之间相关性尽可能的小，以保证算法的收敛性。而 FASTBCDA 和 L1_LS 没有这样的要求，因而应用范围广泛一些。

5. 计算机 CPU 核数的限制，使得一些并行算法不能以最优的进程数来运行。我的计算机的 CPU 是双核的，不支持超线程，所以对于一些并行算法，只能选两个进程，并不能选最优的数目，因为进程多了，一方面是不能所有进程都能同时运行，另一方面，进程上下文切换的时间也无法忽略。

我们通常看重的算法性能指标有：时间，解的稀疏程度 (非零元的个数)，目标函数值，惩罚项的值，均方误差。

解的稀疏程度比运行时间长短更重要一些，运行时间长可以多运行一会儿，但解的稀疏程度不好，以后做预测时，观测变量的维数仍然要很高，这无疑是更大的负担，因为要测量更多的属性。

算法比较中，还囊括了 SpaRSA, GPSR_BB , FPC_AS 三个算法，没有在第一章中做详细介绍，可以参看相关文献。

详尽的数值结果请参看附录。从这些结果来看，Shotgun 表现不错，这其实也和数据集的选取有关，我没有选取那些会使 Shotgun 发散的数据集，而且，大部分是稀疏集。由于 p-FISTA 和 GRock 使用 gsl_matrix 这种数据结构来存储 X，在一些特别大的数据集上，导致无法运行，不过在那些可以运行的数据集上，p-FISTA 和 GRock 的表现还是不错的，相信改善程序使用的数据结构可以提高它们的性能。

第二章 对 ℓ_0TV 求解算法的改进

Total Variation 的方法已经被广泛地运用到图像处理当中，在图像去噪、图像风格化等方面有很多应用。本章利用在 Lasso 求解算法里面学到的一些算法思想，尝试对 ℓ_0TV 模型进行求解，并展示一些已经得到的结果和存在的问题。

第一节 ℓ_0TV

图像去噪（image denoising）一直是图像处理领域比较重要的问题之一。由于数据收集或者数据传输时的种种问题，有可能会引入一些噪声。研究者提出了很多模型和算法来解决这个问题。一些热门的相关算法有 $\ell_{02}TV-AOP, \ell_0TV-PDA$ 等等。最近发表的一篇文章 [13] 提出了 ℓ_0TV 的模型。原模型有更广泛的应用，如果将原模型具体到图像去噪的应用上来，并且惩罚项的范数取 $p=1$，可以得到式 2.1。

$$\min_u \|u-b\|_0+\lambda\sum_{i=1}^{n}[|(\nabla_x u)_i|+|(\nabla_y u)_i|] \tag{2.1}$$

这里面 b 表示观测到的图像，u 表示求解的结果。通常来说，灰度图片 b,u 都是二维矩阵，不妨设 $b,u\in\mathbb{R}^{M\times N}$ 表示一个灰度图片，不过为了方便，我们也可以将矩阵“拉直”，后面都假设 $b,u\in\mathbb{R}^n$。记 ∇_x,∇_y 分别是计算图像 u 在 x 轴和 y 轴的离散梯度的算子。惩罚项的范数取 $p=1$ 时被称为 anisotropic TV，$p=2$ 的情形稍复杂一点。

$$(\nabla_x(u))_{i,j}=\begin{cases} u_{i+1,j}-u_{i,j} & \text{if} \quad \text{i<M,} \\ 0 & \text{if} \quad \text{i=M.} \end{cases} \tag{2.2}$$

$$(\nabla_y(u))_{i,j}=\begin{cases} u_{i,j+1}-u_{i,j} & \text{if} \quad \text{j<N,} \\ 0 & \text{if} \quad \text{j=N} \end{cases} \tag{2.3}$$

当然还有其它的离散梯度的取法，这也是可以的。

对于彩色图片，也可以类似定义离散梯度。由于问题模型对于 RGB 三层有可分的性质，所以后面都只针对灰度图片讨论，然后把结果分别作用到 RGB，即可得到彩色图片的结果。

原作者提出的算法是把 ℓ_0TV 问题转化到它的对偶形式，然后利用 ADMM 进行求解。算法实际运行时显得稍微慢一点 (在我的计算机环境下，大概需要 10 s)。为了能尽量提高计算性能，尝试用新的方法求解 式 2.1。

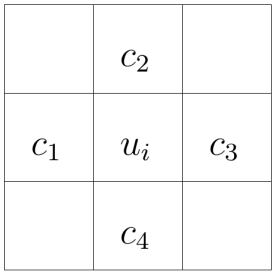

图 2.1 子问题图解

第二节 新算法框架

鉴于坐标下降法在求解 Lasso 时的优异表现，而且 ℓ_0TV 和 Lasso 同样有 ℓ_1 范数的惩罚项，所以考虑用坐标下降法求解 ℓ_0TV。坐标下降法的基本思想就是，固定大部分坐标不动，只对一个或几个坐标进行求解。而求解这些只有几个变量的子问题往往非常容易。具体到问题 式 2.1 就是，只对某一个 u_i 求解，而固定其他下标的变量不动。考虑 u_i 不是图片边缘的情形，结合离散梯度 ∇_x, ∇_y 的定义，将相邻的相关像素点的值记为 c_1, c_2, c_3, c_4，它们都是常数，可以参照图 2.1，得到要求解的子问题是

$$\min_{u_i} \|u_i - b_i\|_0 + |u_i - c_1| + |u_i - c_2| + |u_i - c_3| + |u_i - c_4| \tag{2.4}$$

子问题的解是容易得到的，通过分类讨论 $u_i = b_i$ 或者 $u_i \neq b_i$ 可以得到结果。通常来说求解得到的是一个区间或一个点。另外需要注意的是，其实 式 2.1 也是一个有约束的问题，因为像素的取值是有范围的。但是子问题的解的左右端点基本上就是 b_i, c_1, c_2, c_3, c_4 中的点，所以可以和无约束的问题可以等同。对于那些在图像边缘的像素点，也可以类似地讨论，只不过它们没有像内部的点那样有八个相邻的像素点，它们对应子问题的惩罚项里，只有 c_1, c_2, c_3, c_4 其中的几个而已。

下面讨论坐标 i 的选取。一般来说有以下几种方法，一种是可以顺序地取，按照 $i = 1, ..., n$ 的方法依次求解相应的子问题，这种方法有可能带来一些问题，比如算法更容易得不到最优解等等。另一种方法是随机地取，这种情况往往代表着，对于原始数据了解的并不多。我使用的就是这种方法。此外，如果有一些先验的信息，也可以应用到 i 的选取上面。

不难发现，新算法的并行程度可以很高，因为一个子问题只涉及到邻近的四个点，而一张图片往往是诸如 512×512 的大小，完全满足多个线程并行运算的需要。另一方面，并行算法的一个关键点是，对于共享数据的保护以及不同进程之间的信息的同步。在本算法中，由于子问题的局部性，可以将图片划分为若干块，每一块由一个进程去求解，当取到块与块相邻的那些点时，利用互斥锁进行保护，以避免竞争现象 (race condition) 的发生。也可以让子进程计算的时候，不取边缘的像素点，等所有进程都计算完毕之后，在由一个进程对这些交界处进行

求解。当然，这实际上已经修改了原有算法，影响也比较难以估计，从后面的实际例子可以看到，这样做并没有带来什么问题。而好处就是，可以不用互斥锁，也就没有因为竞争失败而引起的阻塞。

停机准则的选取，可以采用对目标函数值进行记录，如果下降幅度小于某一个阈值时，就停止。也可以采用固定步长。

算法2.1 和算法 2.2 给出了算法的非并行版本和并行版本的具体描述。

Data: 图片 $b \in \mathbb{R}^{M\times N}$, λ , 恢复的结果 u
令 $u = b$。
while 没有收敛 **do**
  1. 按照均匀分布，分别随机选取 $t \in \{1, ..., M\}$,$s \in \{1, ..., N\}$。
  2. 令 $u_i = u[t][s]$，求解子问题 式 2.4 (边界情形的子问题稍有不同)。
end
Result: u

算法 2.1: 基于 CD 的 $\ell_0 TV$ 算法

Data: 图片 $b \in \mathbb{R}^{M\times N}$, λ , 迭代次数 T, 恢复的结果 u，线程个数 P
令 $u = b$。将图片划分为 P 块，每一块由一个线程处理。
P 个线程并行的进行以下步骤:
while 迭代次数小于给定值 T **do**
  1. 按照均匀分布，分别随机选取位于自己分块的 t, s，并且保证 t, s 不是和其他分块相邻的点。
  2. 求解子问题 式 2.4 (边界情形的子问题稍有不同)。
end
P 个线程都结束之后，由一个线程对分块交界的点求解子问题 式 2.4。
Result: u

算法 2.2: 基于 CD 的 $\ell_0 TV$ 并行算法

第三节　算法的收敛性

算法收敛性是一个问题。虽然从后面很多例子可以看到，设计的新算法取得了效果，目标函数值一直都在下降。但不能忽视 $\ell_0 TV$ 使用了 ℓ_0 范数，所以是一个非凸的问题，而且也不光滑。已经有结论证明 ℓ_0 范数的问题是一个 NP-hard 的问题 [7]。另一方面，在一些更好的情形下，都有坐标下降法不收敛的例子。例如图 2.2 里面的函数 (引至维基百科)，其实是一个凸函数，但是不可微，结果坐标下降法就有可能收敛到一个点，在这个点沿任何一个坐标方向，该点的函数值都是最优的，然而由图可以看出，它并不是全局最优的。

另外一个是 Powell 在文章 [10] 中的例子，这是一个连续可微，但是非凸的函数 (见式 2.5)，如果限制只能使用顺序选取下标的方法，并且保证每一步的子问题都是精确求解，反而不能收敛到这个函数在点 $(1, 1, 1)^T$ 和 $(-1, -1, -1)^T$ 的

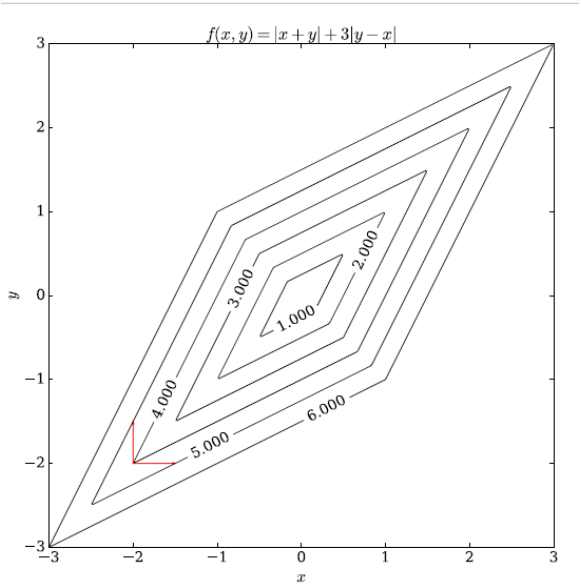

图 2.2 CD 没有收敛到最优解的情形

最值。不过要是随机选取下标进行子问题的求解，这种情形将有很大的概率能够避免。

$$f(x_1,x_2,x_3)=-(x_1x_2+x_2x_3+x_1x_3)+\sum_{i=1}^{3}(|x_i|-1)_+^2 \tag{2.5}$$

关于随机的坐标下降法收敛性的一般结果，目前较好的有 Nesterov 在 [8] 中的结论，他证明了在无约束最优化的情形下，如果目标函数 $f(x)$ 是可微的凸函数，并且 $f(x)$ 的导函数满足 Lipschitz 连续的条件 (对导函数的要求还可以适当放宽，详见参考文献 [8])，那么就可以证明随机的坐标下降法产生的函数值序列的期望值收敛到最优值。其他的相关结果在 [12] 中也可以找到，同样需要 $f(x)$ 是可微的凸函数。目前还没有结论可以保证新算法的收敛性。

第四节　实验结果

通常使用的噪声类型分为两种，random-valued impulse noise 和 salt-and-pepper impulse noise 。第一种是随机选取图片中一定比例的像素点，将这些像素点的取值修改为服从 [0,255] 上均匀分布的取值。第二种噪声则是选取一定比例的像素点之后，将它们的取值要么改成一个可能的最大值，要么改成最小值，取最大值和最小值的比例一般相同。可见这两种噪声的区别在于取值分布的不同，而噪声的位置都是随机的。

这里使用 random-valued impulse noise 做了一次实验，密度是 20%，取 $\lambda=5$。可以得到四张图，图 2.3、图 2.4、图 2.5 、图 2.6 。最后一张图显示了恢复图和原图的区别，黑点代表有区别的地方，颜色越深，差异越大。可以看到，除了在细

节比较多的地方 (比如帽子上的羽毛) 与原图差异较大之外，其它部分基本相同。另外目标函数 式 2.1 的值见 图 2.7。

通常可以使用信噪比 (SNR) 来衡量算法恢复的图片和原来图片的区别，记原图为 u_0，恢复的图片为 u_k，将原图所有像素点的取值平均之后得到的图片记为 $\bar{u}$，假定它们都有 n 个像素点。那么信噪比的定义有以下几种:

$$\begin{aligned} SNR_0(u) &= \frac{n-\|u_0-u_k\|_{0-\epsilon}}{n-\|u_0-u_0\|_{0-\epsilon}} \\ SNR_1(u) &= 10\log_{10}\frac{\|u_0-\bar{u}\|_1}{\|u_k-\bar{u}\|_1} \\ SNR_2(u) &= 10\log_{10}\frac{\|u_0-\bar{u}\|_2^2}{\|u_k-\bar{u}\|_2^2} \end{aligned}$$

这里的 $\|\cdot\|_{0-\epsilon}$ 是对绝对值大于阈值 ϵ 的元素个数进行计数的一种范数。这里取 $\epsilon = 20/255$。

具体到算法的实现上来，由于子问题的解是一个区间，这就面临一个问题，那就是选择区间中的哪一个点才合适。通过实验发现，选取区间的端点和区间的中点相比，没有明显差别，另外也没有什么理论上的相关结果。所以为了加快计算速度，算法的具体实现上使用的是选取端点的方法。算法初始值的选取也很重要，不当的选取有可能导致算法的结果很差。比如使用元素全是一或零的矩阵作为初始值，往往得不到满意的解。通常可以直接取观测到的有噪声的图片即可。

算法的时间上，与迭代次数有关，如果迭代两百万次的话，时间在 0.4s 左右。后来使用并行的方法，同样的迭代次数，大概花费了 0.8s 左右。时间增加的原因可以这样来解释: 一方面，创建任务 (Linux 不区分进程和线程，统称为 task) 是要消耗时间的；另一方面，操作系统的调度机制也是要考虑的因素之一，一个系统之中往往还有其它很多任务在运行。

代码在 https://github.com/hanbingyan/cdtv。关于数值结果可以参考附录部分。

第五节　存在的问题

算法的局部性也会带来一些问题，主要是噪声类型上的问题，实验中都是采用的由文件产生的位置较为随机的噪声。而如果用 图 2.8 中的噪声，也就是在图片上画一条宽度是一个像素的白线，那么即使新算法每次都取白线和白线周围的点，进行子问题求解，这条白线也是去不掉的。问题的原因在于，新算法每次只看一个像素点周围的取值，在这种情况下，它会认为白线的取值是局部最优的，也就是不需要更改的。解决这个问题的办法是，扩大子问题考虑的像素点个数，由原来的只考虑一个像素点改为考虑四个像素点 (参考图 2.9)，然后求解，就可以去掉像白线这样的噪声。但这么做也是有代价的，子问题没有一个显式解，需要通过迭代求解。子问题变得复杂之后，求解速度显著下降。而且对于更粗一些的线条类型的噪声也无能为力。只能尝试进一步扩大考虑的范围。

图 2.3 原图

图 2.4 噪声

图 2.5 恢复

图 2.6 差别

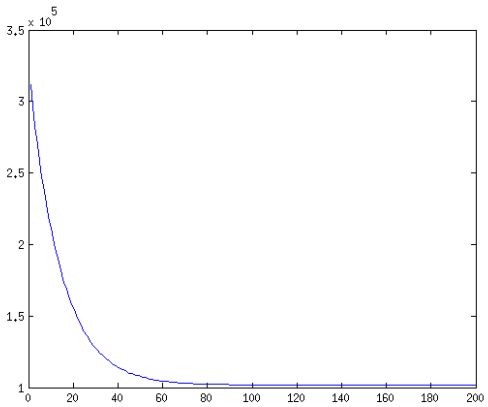

图 2.7 目标函数值，每求解两万个子问题后计算一次

图 2.8 左侧有一条白线

	c_3	c_4	
c_2	u_j	u_k	c_5
c_1	u_i	u_l	c_6
	c_8	c_7	

图 2.9 扩大子问题范围

从实验的结果来看，新算法在噪声密度小于 50% 时与其他算法的表现差不多，而当噪声密度很大时，新算法显得无能为力。另一方面，总的来说新算法在 random-valued impulse noise 上的表现优于在 salt-and-pepper impulse noise 上的表现。

另外一个问题是随机性带来的问题。由于无法事先感知到噪声位于图片的哪一个部分，新算法有可能在一些没有噪声的部分做无用功，特别是噪声所占比例较小的时候，有噪声的部分被取到的概率非常小。这个问题的解决也许只能寄希望于我们事先对噪声的位置能有一些了解，以便对像素点的选取做一些限制。

参考文献

[1] A. Beck and M. Teboulle. *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*. SIAM Journal on Imaging Sciences, 2(1):183-202,2009.

[2] J.K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. *Parallel coordinate descent for L1-regularized loss minimization*. ICML,321-328,2011.

[3] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. *Least angle regression* . Annals of Statistics, 32(2):407-499, 2004.

[4] F. Facchinei and S. Lucidi. *Quadratically and superlinearly convergent algorithms for the solution of inequality constrained minimization problems*. J.Optim.Theory Appl. 85,265-289,1995.

[5] J. Friedman, T. Hastie, and R. Tibshirani. *Regularization paths for generalized linear models via coordinate descent*. Journal of Statistical Software, 33(1):1-22, 2010.

[6] S.J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. *An interior-point method for large-scale L1-regularized least squares*. IEEE Journal of Sel. Top. in Signal Processing, 1(4):606-617, 2007.

[7] B.K.Natarajan. *Sparse approximate solutions to linear systems*. SIAM Journal on Computing. 24(2):227-234,1995.

[8] Y. Nesterov. *Efficiency of coordinate descent methods on huge-scale optimization problems*. SIAM Journal on Optimization, 22(2):341-362, 2012.

[9] Z. Peng, M. Yan, and W. Yin. *Parallel and distributed sparse optimization*. Preprint, UCLA, 2013.

[10] M.J.D. Powell. *On search directions for minimization algorithms*. Mathematical Programming. 4,193-201,1973.

[11] M.D.Santis, S. Lucidi, and F. Rinaldi. *A fast active set block coordinate descent algorithm for ℓ_1-regularized least squares*. SIAM Journal on Optimization,26(1):781-809,2016.

[12] S. Shalev-Shwartz and A. Tewari. *stochastic methods for ℓ_1-regularized loss minimization*. Journal of Machine Learning Research 12,1865-1892,2011.

[13] G. Yuan and B. Ghanem. *ℓ_0TV : A new method for image restoration in the presence of impulse noise*. CVPR.5369 - 5377,2015.

附录 A 数值实验结果

所有的实验都是在同一台计算机上完成的。硬件配置双核 2.2 GHz CPU 和 6GB 内存，软件环境是 Ubuntu 14.04 64bit 和 Matlab 2014a。

第一章使用的数据集 (表 A.1) 来自 http://www.select.cs.cmu.edu/projects/shotgun/datasets/。另外需要特别注意的是，Shotgun 使用了 64 位独有的指令集，所以在 32 位的计算机上会报错。

表 A.2 中，Time 是这样测量的: 对于使用 Matlab 的程序，Matlab 里面有 Run and Time 这个选项，可以得到 m 文件的运行时间 (CPU Time). 运行几次取平均。不过计算结果没有取平均，而是选了其中一个好一些的。使用 C/C++ 的可以在命令行里使用 time ./yourprogram 的方式测量时间。目标函数值指的是 $\|X\beta - y\|^2 + 0.5\|\beta\|_1$ 的值。

表 A.3、表 A.4 是按照与 [13] 同样的方式，以信噪比为依据记录的实验结果，原文章里给出了原算法的相关结果，可以做一个对比。

数据集名称	大小	特点
finance1000	30465× 216842	稀疏，高维数据
Ball64_singlepixcam	1638×4096	不稀疏，高维数据
mug05_12_12	12410×24820	稀疏，高维数据
SparcoProblem603	1024×4096	稀疏，高维数据，来自 GRSR 2D Compressed sensing
peppers05_6_6	32768×65536	稀疏，高维数据

表 A.1 数据集合特点

Finance1000	Time	目标函数值	$\|X\beta - y\|$	惩罚项	β 非零元比例
Shotgun	34.089s	4.3208e4	207.4438	174.9803	3.89%
GRock	GSL	无法分得	足够内存		
pFISTA	GSL	无法分得	足够内存		
L1_LS	20min 无结果	/	/	/	/
SpaRSA	304.557s	2.3175e11	4.8141e5	8.8995e4	100%
FPC_AS	128.79 s	5.7662e8	2.4011e4	7.8124e4	99.83%
GPSR_BB	393.296 s	7.5619e4	274.7133	151.8883	93.86%
SparcoPro603	Time	目标函数值	$\|X\beta - y\|$	惩罚项	β 非零元比例
Shotgun	0.247s	89.7342	4.6303	68.2942	1.22%
GRock	7s	89.7342	4.6303	68.2942	1%
pFISTA	4s	89.7342	4.6302	68.2956	1%
L1_LS	0.083s	649.0677	25.4768	0	0%
SpaRSA	0.194s	95.8899	6.5063	53.5578	0.49%
FPC_AS	0.340s	95.8899	6.5063	53.5578	0.49%
GPSR_BB	0.181s	95.8899	6.5063	53.5579	0.49%
Peppers05_6_6	Time	目标函数值	$\|X\beta - y\|$	惩罚项	β 非零元比例
Shotgun	8.790s	4.5565e3	9.4003	4.4681e3	38.53%
GRock	GSL	无法分得	足够内存		
pFISTA	GSL	无法分得	足够内存		
L1_LS	0.395s	1.4442e5	376.5139	2.6563e3	100%
SpaRSA	1.723s	4.6150e3	16.4409	4.3447e3	31.9%
FPC_AS	2.463s	4.6150e3	16.4409	4.3447e3	31.9%
GPSR_BB	4.987s	4.6150e3	16.4409	4.3447e3	31.9%
Ball64_singlepixcam	Time	目标函数值	$\|X\beta - y\|$	惩罚项	β 非零元比例
Shotgun	3.071s	0.3445	0.1737	0.3143	26.95%
GRock	12s	0.3221	0.0851	0.3148	23%
pFISTA	4s	0.3152	0.0314	0.3142	97%
L1_LS	0.208s	0.3666	0.2271	0.3151	100%
SpaRSA	91.466s	0.3237	0.0994	0.3138	100%
FPC_AS	14.777s	3.3919	1.2173	1.9101	100%
GPSR_BB	96.715s	0.3116	0.0362	0.3103	69.65%
Mug05_12_12	Time	目标函数值	$\|X\beta - y\|$	惩罚项	β 非零元比例
Shotgun	2.460s	1684.5570	4.3618	1.6655e3	41.44 %
GRock	590s	5136.0600	60.9967	1.4154e3	6.64%
pFISTA	180s	1684.9	4.3721	1.6658e3	43.07%
L1_LS	16.070s	1684.5584	4.3599	1.6655e3	100%
SpaRSA	1.811s	1697.8289	7.7531	1.6377e3	36.09%
FPC_AS	1.943s	1697.8289	7.7531	1.6377e3	36.08%
GPSR_BB	4.857s	1697.8290	7.7532	1.6377e3	36.09%

表 A.2 算法结果

图片名称 + 噪声密度	破损图片的 $SNR_0/SNR_1/SNR_2$	恢复图片的 $SNR_0/SNR_1/SNR_2$
walkbridge +10 %	0.92 / 7.69 / 5.38	0.94 /12.21/ 15.08
walkbridge +30 %	0.74 / 2.87 / 0.56	0.90 /8.93/ 11.54
walkbridge +50 %	0.58 / 0.70 / -1.60	0.80 /5.91/ 7.27
walkbridge +70 %	0.41 / -0.78 / -3.08	0.60 /2.58/ 2.67
walkbridge +90 %	0.24 / -1.86 / -4.17	0.34 /-0.04/ -0.79
pepper +10 %	0.69 / 3.89 / 4.99	0.74 /5.42/ 11.31
pepper +30 %	0.57 / 1.84 / 0.94	0.73 /5.26/ 10.57
pepper +50 %	0.45 / 0.45 / -1.13	0.68 /4.52/ 7.85
pepper +70 %	0.33 / -0.62 / -2.55	0.54 /2.54/ 3.33
pepper +90 %	0.21 / -1.46 / -3.60	0.31 /0.24/ -0.38
mandrill +10 %	0.83 / 2.88 / 3.52	0.79 /3.25/ 6.06
mandrill +30 %	0.68 / 0.84 / -0.60	0.72 /2.80/ 4.90
mandrill +50 %	0.53 / -0.52 / -2.63	0.63 /2.12/ 3.31
mandrill +70 %	0.38 / -1.58 / -4.05	0.50 /0.94/ 0.97
mandrill +90 %	0.23 / -2.42 / -5.11	0.32 /-0.58/ -1.62
lenna +10 %	0.92 / 5.43 / 4.85	0.99 /8.81/ 17.48
lenna +30 %	0.75 / 2.11 / 0.20	0.97 /8.07/ 14.18
lenna +50 %	0.58 / 0.23 / -2.02	0.91 /6.36/ 9.08
lenna +70 %	0.41 / -1.06 / -3.46	0.72 /3.13/ 3.39
lenna +90 %	0.24 / -2.06 / -4.56	0.39 /0.14/ -0.67
lake +10 %	0.92 / 6.27 / 6.51	0.98 /8.75/ 16.12
lake +30 %	0.75 / 3.28 / 1.90	0.94 /8.03/ 13.64
lake +50 %	0.58 / 1.52 / -0.30	0.84 /6.28/ 8.94
lake +70 %	0.41 / 0.29 / -1.73	0.58 /3.25/ 3.72
lake +90 %	0.24 / -0.68 / -2.82	0.25 /0.58/ -0.04
jetplane +10 %	0.37 / 2.01 / 2.13	0.39 /3.41/ 8.34
jetplane +30 %	0.32 / 0.03 / -1.96	0.39 /3.21/ 7.40
jetplane +50 %	0.27 / -1.33 / -4.04	0.37 /2.32/ 4.27
jetplane +70 %	0.23 / -2.35 / -5.41	0.28 /0.01/ -0.65
jetplane +90 %	0.18 / -3.17 / -6.45	0.16 /-2.27/ -4.20
blonde +10 %	0.10 / 0.46 / 1.05	0.10 /0.93/ 2.96
blonde +30 %	0.11 / -0.46 / -1.45	0.11 /0.87/ 2.72
blonde +50 %	0.12 / -1.21 / -3.04	0.13 /0.66/ 2.00
blonde +70 %	0.14 / -1.85 / -4.19	0.15 /0.00/ 0.11
blonde +90 %	0.15 / -2.41 / -5.09	0.16 /-1.14/ -2.38
cameraman +10 %	0.91 / 8.21 / 6.18	0.99 /20.50/ 24.92
cameraman +30 %	0.75 / 3.47 / 1.42	0.98 /14.34/ 17.10
cameraman +50 %	0.58 / 1.23 / -0.82	0.90 /8.91/ 9.78
cameraman +70 %	0.41 / -0.23 / -2.29	0.69 /3.74/ 3.47
cameraman +90 %	0.24 / -1.33 / -3.39	0.37 /0.45/ -0.40
barbara +10 %	0.92 / 6.20 / 5.60	0.92 /7.83/ 11.54
barbara +30 %	0.75 / 2.68 / 0.91	0.88 /6.95/ 9.92
barbara +50 %	0.58 / 0.76 / -1.31	0.80 /5.36/ 7.02
barbara +70 %	0.41 / -0.57 / -2.75	0.62 /2.76/ 3.05
barbara +90 %	0.24 / -1.60 / -3.86	0.33 /0.14/ -0.58
boat +10 %	0.92 / 4.61 / 5.09	0.98 /6.79/ 14.05
boat +30 %	0.75 / 1.93 / 0.60	0.95 /6.32/ 12.06
boat +50 %	0.58 / 0.26 / -1.58	0.86 /5.05/ 8.15
boat +70 %	0.41 / -0.93 / -3.01	0.65 /2.44/ 3.06
boat +90 %	0.24 / -1.87 / -4.09	0.32 /-0.19/ -0.82
pirate +10 %	0.92 / 8.36 / 6.20	0.97 /14.75/ 17.65
pirate +30 %	0.74 / 3.60 / 1.46	0.93 /11.02/ 13.67
pirate +50 %	0.57 / 1.36 / -0.78	0.83 /7.02/ 8.04
pirate +70 %	0.40 / -0.11 / -2.26	0.59 /2.95/ 2.81
pirate +90 %	0.23 / -1.19 / -3.34	0.30 /0.24/ -0.63
livingroom +10 %	0.91 / 6.81 / 4.19	0.98 /14.66/ 17.44
livingroom +30 %	0.75 / 2.04 / -0.57	0.95 /10.10/ 12.53
livingroom +50 %	0.58 / -0.17 / -2.77	0.88 /6.34/ 7.29
livingroom +70 %	0.41 / -1.60 / -4.20	0.71 /2.73/ 2.48
livingroom +90 %	0.24 / -2.71 / -5.32	0.42 /-0.32/ -1.35
house +10 %	0.92 / 8.06 / 5.79	1.00 /18.44/ 30.00
house +30 %	0.75 / 3.50 / 1.03	0.99 /15.33/ 20.74
house +50 %	0.58 / 1.31 / -1.20	0.94 /10.16/ 11.20
house +70 %	0.41 / -0.14 / -2.68	0.73 /4.48/ 4.06
house +90 %	0.24 / -1.19 / -3.72	0.36 /0.67/ -0.34

表 A.3 Random-Value Impulse Noise, $\lambda = 5$

图片名称 + 噪声密度	破损图片的 $SNR_0/SNR_1/SNR_2$	恢复图片的 $SNR_0/SNR_1/SNR_2$
walkbridge +10 %	0.90 / 5.44 / 1.82	0.98 /15.21/ 17.74
walkbridge +30 %	0.70 / 0.63 / -2.99	0.91 /8.30/ 7.85
walkbridge +50 %	0.51 / -1.55 / -5.17	0.75 /2.75/ 0.31
walkbridge +70 %	0.31 / -3.02 / -6.64	0.42 /-1.73/ -4.92
walkbridge +90 %	0.11 / -4.10 / -7.72	0.09 /-4.06/ -7.55
pepper +10 %	0.68 / 2.96 / 2.08	0.74 /5.39/ 11.22
pepper +30 %	0.53 / 0.24 / -2.35	0.72 /4.81/ 8.02
pepper +50 %	0.39 / -1.42 / -4.50	0.62 /2.33/ 1.24
pepper +70 %	0.24 / -2.63 / -5.95	0.37 /-1.25/ -4.11
pepper +90 %	0.10 / -3.58 / -7.02	0.09 /-3.53/ -6.91
mandrill +10 %	0.81 / 1.85 / 0.31	0.86 /3.98/ 8.00
mandrill +30 %	0.63 / -0.94 / -4.19	0.77 /2.89/ 3.82
mandrill +50 %	0.45 / -2.63 / -6.34	0.61 /0.27/ -1.98
mandrill +70 %	0.27 / -3.83 / -7.77	0.33 /-2.91/ -6.55
mandrill +90 %	0.09 / -4.78 / -8.85	0.06 /-4.85/ -8.90
lenna +10 %	0.90 / 3.82 / 1.21	0.99 /8.84/ 17.17
lenna +30 %	0.70 / -0.03 / -3.54	0.95 /6.86/ 8.13
lenna +50 %	0.50 / -2.04 / -5.76	0.80 /2.33/ -0.15
lenna +70 %	0.30 / -3.40 / -7.20	0.45 /-1.98/ -5.41
lenna +90 %	0.10 / -4.44 / -8.30	0.08 /-4.40/ -8.16
lake +10 %	0.90 / 5.16 / 3.50	0.98 /8.85/ 16.27
lake +30 %	0.71 / 1.70 / -1.19	0.92 /7.20/ 9.62
lake +50 %	0.51 / -0.22 / -3.42	0.74 /3.60/ 2.14
lake +70 %	0.32 / -1.52 / -4.84	0.43 /-0.04/ -2.78
lake +90 %	0.12 / -2.53 / -5.93	0.10 /-2.37/ -5.63
jetplane +10 %	0.36 / 1.46 / -0.11	0.40 /3.42/ 8.37
jetplane +30 %	0.30 / -0.96 / -4.49	0.39 /2.94/ 5.41
jetplane +50 %	0.24 / -2.52 / -6.65	0.36 /1.01/ -0.69
jetplane +70 %	0.17 / -3.64 / -8.05	0.24 /-1.95/ -5.62
jetplane +90 %	0.11 / -4.53 / -9.12	0.12 /-4.17/ -8.58
blonde +10 %	0.09 / -0.01 / -0.67	0.10 /0.97/ 3.05
blonde +30 %	0.08 / -1.52 / -4.05	0.11 /0.77/ 2.08
blonde +50 %	0.07 / -2.63 / -5.93	0.11 /-0.39/ -1.65
blonde +70 %	0.06 / -3.50 / -7.22	0.09 /-2.59/ -5.84
blonde +90 %	0.05 / -4.24 / -8.22	0.06 /-4.26/ -8.23
cameraman +10 %	0.91 / 6.14 / 2.82	1.00 /20.35/ 24.29
cameraman +30 %	0.73 / 1.40 / -1.93	0.97 /11.88/ 10.94
cameraman +50 %	0.55 / -0.84 / -4.17	0.85 /4.56/ 2.04
cameraman +70 %	0.37 / -2.30 / -5.63	0.55 /-0.59/ -3.57
cameraman +90 %	0.18 / -3.40 / -6.74	0.18 /-3.34/ -6.56
barbara +10 %	0.90 / 4.61 / 2.15	0.97 /9.09/ 13.96
barbara +30 %	0.70 / 0.68 / -2.59	0.89 /6.54/ 7.30
barbara +50 %	0.50 / -1.38 / -4.83	0.73 /2.47/ 0.39
barbara +70 %	0.30 / -2.75 / -6.28	0.41 /-1.42/ -4.47
barbara +90 %	0.10 / -3.81 / -7.38	0.08 /-3.74/ -7.19
boat +10 %	0.90 / 3.41 / 1.81	0.99 /6.85/ 14.22
boat +30 %	0.70 / 0.13 / -2.82	0.94 /5.72/ 8.30
boat +50 %	0.50 / -1.72 / -5.01	0.79 /2.33/ 0.67
boat +70 %	0.30 / -3.02 / -6.47	0.44 /-1.70/ -4.77
boat +90 %	0.10 / -4.02 / -7.56	0.08 /-4.00/ -7.47
pirate +10 %	0.91 / 6.48 / 3.07	0.98 /16.86/ 19.51
pirate +30 %	0.73 / 1.70 / -1.70	0.93 /10.01/ 9.76
pirate +50 %	0.54 / -0.53 / -3.94	0.80 /4.21/ 2.02
pirate +70 %	0.36 / -1.98 / -5.39	0.50 /-0.34/ -3.24
pirate +90 %	0.18 / -3.08 / -6.48	0.17 /-2.91/ -6.15
livingroom +10 %	0.90 / 4.32 / 0.31	0.99 /16.53/ 19.18
livingroom +30 %	0.70 / -0.45 / -4.45	0.95 /8.83/ 7.55
livingroom +50 %	0.51 / -2.67 / -6.68	0.80 /2.01/ -1.18
livingroom +70 %	0.31 / -4.11 / -8.11	0.46 /-2.75/ -6.46
livingroom +90 %	0.11 / -5.22 / -9.22	0.09 /-5.26/ -9.19
house +10 %	0.90 / 6.00 / 2.33	0.99 /17.49/ 23.00
house +30 %	0.70 / 1.35 / -2.44	0.95 /10.82/ 9.94
house +50 %	0.50 / -0.85 / -4.68	0.80 /4.23/ 1.39
house +70 %	0.31 / -2.30 / -6.13	0.47 /-0.57/ -3.94
house +90 %	0.11 / -3.37 / -7.20	0.10 /-3.21/ -6.88

表 A.4 Salt-and-Pepper Impulse Noise, $\lambda = 2$

图 A.1　原图

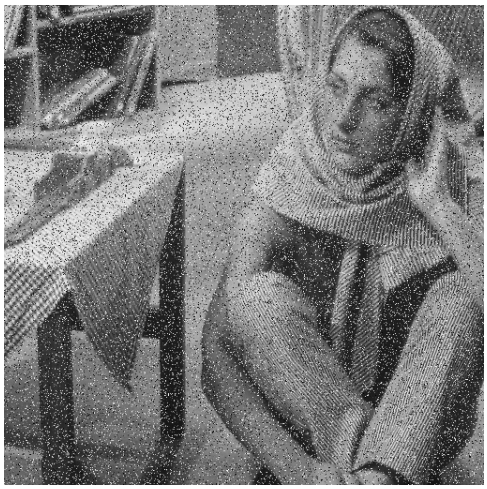

图 A.2　噪声

图 A.3　恢复

图 A.4　差别

图 A.5　原图

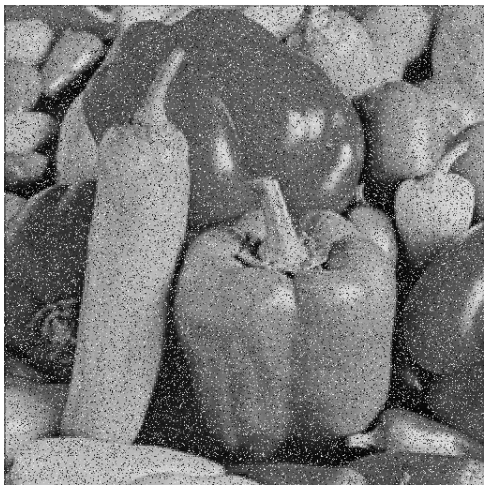

图 A.6　噪声

图 A.7　恢复

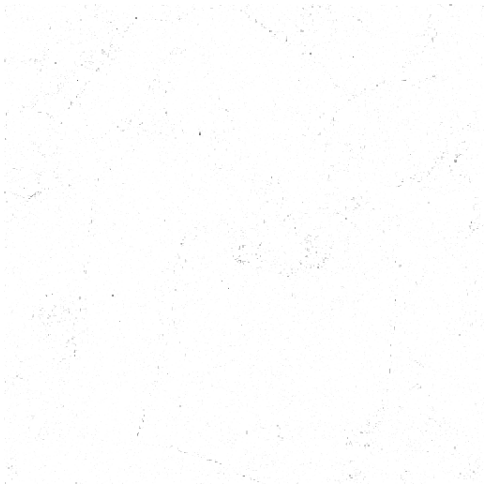

图 A.8　差别