兰州大学

本科毕业论文

论文题目（中文） 基于稀疏图码的离群值去除

论文题目（英文） Outliers Removal Based on Sparse-Graph Codes

学生姓名 庄启源

指导教师 李朋

学　　院 数学与统计学院

专　　业 数学（基础理论班）

年　　级 2020 级

兰州大学教务处

诚信责任书

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所取得的成果。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人、集体已经发表或未发表的论文。

本声明的法律责任由本人承担。

论文作者签名：_____ 日 期：_____

关于毕业论文（设计）使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用毕业论文（设计）的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州大学可以将本毕业论文（设计）的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本毕业论文（设计）。本人离校后发表、使用毕业论文（设计）或与该毕业论文（设计）直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

本毕业论文（设计）研究内容：

☑可以公开

□ 不宜公开，已在学位办公室办理保密申请，解密后适用本授权书。

（请在以上选项内选择其中一项打“√”）

论文作者签名：_____ 导师签名：_____

日 期：_____ 日 期：_____

基于稀疏图码的离群值去除

中文摘要

随着压缩感知理论研究的推进以及在工业界的大量应用，众多解码方法被提出以更精确快速地重构稀疏度为 K 的高维稀疏信号 $\mathbf{x} \in \mathbb{R}^N$。为了更好地解决现实会遇到的信号恢复问题，我们研究在无噪声情形下当观测信号存在少量离群值时稀疏信号的支撑集恢复。本文提出了一种离群值去除算法能够有效消除离群值在信号恢复中的巨大偏差影响。在恢复过程中，我们应用稀疏图码相关理论，借由 DFT 矩阵和左正则二部图给出精心设计的测量矩阵。由此实现的重构算法仅需对观测值进行几次简单的解码迭代即可精确恢复稀疏系数。我们借助测量次数、恢复时间、精确恢复率等指标来评估信号重构算法。特别地，本文展示了在无噪声情况下，我们的框架能够高效且稳定地恢复存在离群值的观测信号。本文的所有源代码和数据公开在*https://github.com/waterEand/thesis_lzu*。

关键词：压缩感知；离群值；稀疏信号恢复；稀疏图码

Outliers Removal Based on Sparse-Graph Codes

Abstract

With the advancement of research in compressed perception theory and its numerous applications in industry, numerous decoding methods have been proposed to fast reconstruct a high-dimensional K-sparse signal $\mathbf{x} \in \mathbb{R}^N$ accurately. In order to solve a signal recovery problem that may be encountered in reality more comprehensively, we study the support recovery of sparse signals from the observations containing a few outliers without noise. In this work, an outlier removal algorithm is proposed to effectively eliminate the influence of large deviations of outliers in signal recovery. Through recovery, the measurement matrix is carefully designed through sparse-graph codes combined with the DFT matrix and the left regular bipartite graph. A reconstruction algorithm is also implemented, by which the sparse coefficients can be recovered in a few iterations by performing simple error decoding over the observations. The signal reconstruction algorithm is evaluated by measuring times, time of recovery, rate of accurate recovery, etc. Specifically, it is shown that in the absence of noise, our framework is able to efficiently recover observed signals in the presence of outliers steadily. All source codes and data of our work are available at *https://github.com/waterEand/thesis_lzu*.

Keywords: Compressed sensing; Outliers; Sparse signal recovery; Sparse-graph codes

目　录

第一章 绪 论

1.1 问题介绍

1.1.1 压缩感知问题

压缩感知（Compressed Sensing, CS）[1] 是一种利用信号稀疏性的信号采样理论。在有噪声情形，这一问题可被表达为：对一个长度为 N 的稀疏信号 $\mathbf{x}$，使用一个随机测量矩阵采样后得到一个长度为 M 的观测向量，再由该观测值估计初始信号。信号的压缩观测过程可表示为：

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{w} \tag{1}$$

这一过程被称为编码。其中 $\mathbf{A}$ 是一个维数为 $M \times N$ 的已知测量矩阵（观测矩阵），$\mathbf{w}$ 是一个可能的噪声向量，$\mathbf{y}$ 是观测值。

通常情况下，若 $\mathbf{x}$ 没有特殊的结构或额外的信息，我们无法通过远少于该信号维度的测量次数（采样）来恢复 $\mathbf{x}$。但是，如果信号是稀疏的或可压缩的，即 $\mathbf{x}$ 向量中只有 K 个非零元素（$\mathbf{x}$ 的稀疏度为 K）且 $K << N$，那么我们就可以由极少次采样精确地恢复原信号。这一过程也被称为解码。这种通过低维观测值重建高维信号的压缩感知方法被应用在各个领域中，如医疗成像 [2]、天体物理学成像 [3]、语音与图像处理 [4] 等。

1.1.2 次线性时间的支撑集恢复算法

在压缩感知问题中，一个好的测量矩阵设计和高效的解码算法是极其重要的。为达到这一目的，我们可以从两个角度的问题入手：要求能够保证信号精确恢复所需的观测次数 M（即测量矩阵 $\mathbf{A}$ 的行数）尽可能小；构建合适的测量矩阵 $\mathbf{A}$ 使得恢复信号所需的时间复杂度尽可能小。

为同时达到这两个问题的最优，研究人员利用信号的稀疏性，提出了大量测量矩阵设计思路和信号重构算法来从低维观测值中恢复信号。其中，大部分相关文献提出的稀疏信号恢复模型都主要基于 ℓ_2/ℓ_1 范数或 ℓ_1/ℓ_1 范数的近似误差指标，而基于支撑集恢复 [5] 的工作就相对较少。对于在不同观测情况下精确恢复信号支撑集的充要条件，学者们采取了最优解码器 [6,7]、ℓ_1 极小 [8]、贪心算法 [9] 等策略进行研究。

其中，Wainwright [7] 提出，在高斯噪声下，当测量矩阵的元素服从独立同分布（i.i.d.）的高斯分布时，$O(K\log(N/K))$ 次测量 (即 $M = O(K\log(N/K))$) 对于信号的支撑恢复是充分且必要的。Bakshi、Jaggi [10] 等人提出的 SHO-FA 算法使有噪声情形下的信号恢复仅需 $O(K)$ 次测量且编码解码时间复杂度分别为 $O(N)$ 和 $O(K)$，这一结果达到了信息论中的阶次最优。Li、Yin [11] 等人建立了一种新的压缩感知框架，能够在无噪声情况下，用 $2K$ 次

测量精确地恢复任意稀疏度为 K 的信号，解码时间复杂度达到 $O(K)$；在有噪声情形，用 $O(K\log(N/K))$ 次测量能达到同等效果，解码时间复杂度为 $O(K\log(N/K))$。并且文献 [11] 提出的模型在 K 为次线性时，即存在 $\delta \in (0,1)$ 使得 $K = O(N^{\delta})$，测量次数和计算时间都与信号维数 N 成次线性关系，实现了概率保证下两者同时的阶次最优。

1.1.3 基于稀疏图码的分治法

我们通过通信系统中的稀疏图码来看待压缩感知问题，并用简单的“分治法”（分而治之）来处理稀疏信号。如图1.1的 (a) 所示，我们用不同的颜色表示稀疏向量 **x** 中的不同元素，其中红色、绿色、蓝色方格表示非零元素，白色方格表示零元素。在测量矩阵中，灰色方格表示随机生成的元素。相应生成的观测向量中每个元素则是红、绿、蓝颜色的混合。

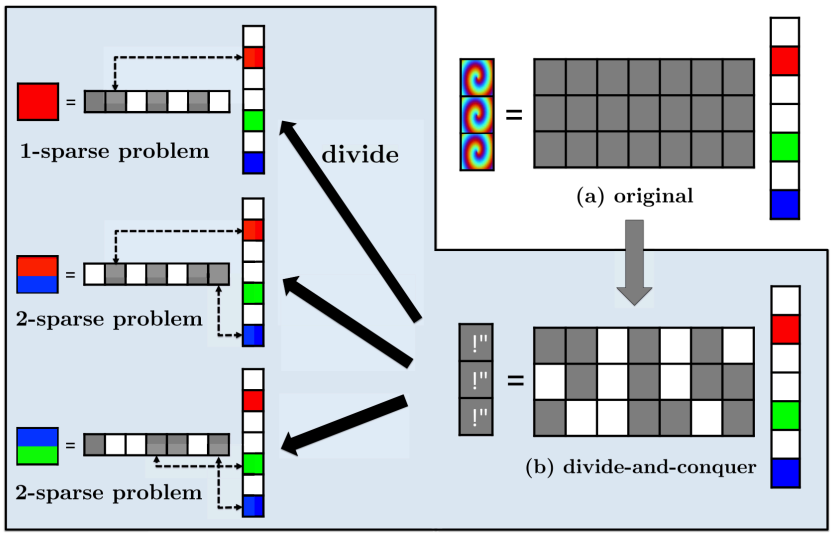

图 1.1 “分治法”的概念抽象图。子图 (a) 描绘一个稀疏度为 3 的恢复问题，其中测量矩阵着灰色，表示元素是随机生成的。由于不同颜色成分（红色、绿色、蓝色）随机混合，导致得到的观测结果呈现混合色。在子图 (b) 中，我们通过在每行中放置三个零元素来稀疏化测量矩阵，即图中的白色方格。由此产生的测量矩阵将稀疏度为 3 的恢复问题分解为多个稀疏度小于 3 的子问题。若子问题的观测值是单一颜色，我们可以立即恢复这一元素。在这一图中，第一个测量中的红色元素可以被立即恢复，之后再从第二个红蓝元素中把红色剥离，恢复蓝色，以此类推。

在图1.1的 (b) 中，我们用零元素（图中的空白格）来稀疏化测量矩阵的每一行。这样设计的测量矩阵就导致了观测到的向量 **y** 中包含的元素一部分是单一颜色，一部分是几种颜色的混合。我们的设计理念就是把稀疏信号中的非零元分散到多个单一颜色的测量中（例如第一个测量中的红色），然后从把这些单一颜色从混合颜色中剥离出来（例如第二个测量中的红蓝混合和第三个测量中的蓝绿混合），如此迭代逐步解码其他颜色的未知元素，这也就是“分治”的过程。稀疏图码在这里的作用本质上是将一般的稀疏信号重构问题分解为多个可以轻松解决的子问题，最后再把这些子问题的结果融合以恢复所有非零元素。这样的

设计兼具了稀疏图编码在测量成本（容量逼近）和计算复杂性（基于快速剥离的解码）方面的特性，从而使稀疏测量矩阵达到了低测量次数和低计算成本的效果。

1.2 研究意义与目的

考虑到压缩感知常被应用在图像采样的任务中，且采样过程可能会出现探头损坏，导致得到的观测值会出现离群值（Outliers）的情况，本文研究在无噪声情形下，观测值中存在部分离群值的稀疏信号的解码恢复。该观测过程可表示为：

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{f} \tag{2}$$

其中 $\mathbf{f}$ 是一个维数为 M 的离群向量，$\mathbf{f}$ 中的非零元素（离群值）个数 T 远小于 M，满足 $|\mathrm{supp}(\mathbf{f})| = T = \eta M$ $(0 \le \eta < \frac{1}{2})$，且这些离群值的分布方差显著大于 $\mathbf{x}$ 中的非零元素。我们的目标是尽可能去除 $\mathbf{y}$ 的非零元素中的所有离群值，并着眼于精确地恢复稀疏度为 K 长度为 N 的信号的支撑集（稀疏体系）以及相对应的稀疏系数。

支撑集恢复结果的一个经典误差评价指标是信号支撑集无法被精确恢复的概率，即 $\Pr(\mathrm{supp}(\hat{\mathbf{x}}) \neq \mathrm{supp}(\mathbf{x}))$，其中 $\mathrm{supp}(\mathbf{x}) \coloneqq \{k : x[k] \neq 0,\ 0 \le k \le N-1\}$。也就是说，对于任意给定的稀疏度为 K 长度为 N 的信号 $\mathbf{x}$，我们设计一个测量矩阵 $\mathbf{A}$，并计算出一个估计向量 $\hat{\mathbf{x}}$，并且希望这一个估计的支撑集与原信号 $\mathbf{x}$ 的支撑集完全匹配的概率接近于 1。

同时，我们也要恢复 $\mathbf{x}$ 中的稀疏系数，也就是要恢复信号中非零元的精确值。对于这一恢复，我们目标得到较强的 ℓ_∞ 和 ℓ_1 范数保证。这里，我们采取非均匀保证 [4]，也就是采用单个测量矩阵来恢复特定的单个稀疏信号。相对地，均匀保证是指仅用一个测量矩阵恢复一次生成的所有稀疏信号（多个），这里不做讨论。

兼顾以上两个要求，我们制定评价指标为信号被精确恢复的概率，具体定义将在第四章给出。我们的目的就是提高这一概率，并与其他经典模型比较。

1.3 本文的贡献

在本文中，我们基于稀疏图码把稀疏度为 K 的信号恢复问题转化为求解多个稀疏度为 1 维数为 N 的信号恢复问题，并进一步探讨存在离群值情况下的问题，提出一种离群值去除算法。在最后的数值实验中，存在少量离群值下的信号精确恢复率基本达到 90% 以上，与传统的 PLAD 算法 [12, 13] 相比具有极大的恢复效率优势，且对不同的情况更具一般性，也不会因算法参数变化产生极大波动。

第二章　测量矩阵的设计

在本章中，我们将从章节1.1.3描述的理念出发，借助简单例子介绍测量矩阵 **A** 的具体设计方法。

我们考虑一个长度 $N=16$，稀疏度 $K=4$ 的稀疏信号 **x**。其中包含有非零元素 $x[1]=1$，$x[4]=4$，$x[8]=2$，$x[13]=7$。接下来我们构建一个左节点数为 16，右节点数为 9 的二部图。如图2.1所示，这个图有以下性质：

- 每个左节点 $x[k]$ 都连接至少一个右节点。
- 每个右节点 y_r 的值是与其相连接的的左节点值的和。

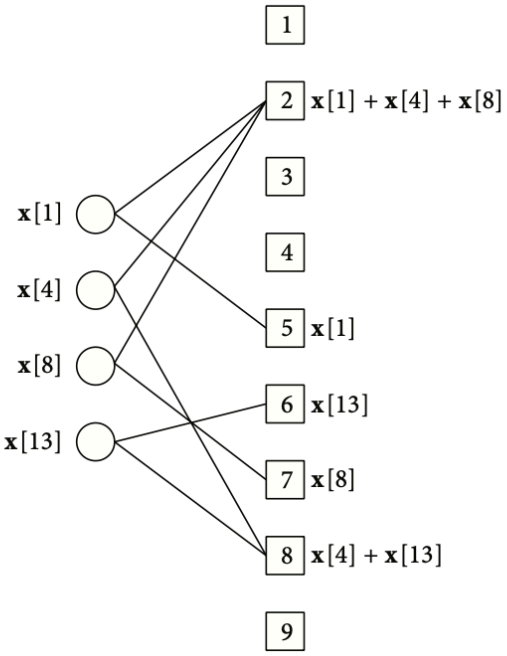

图 2.1　稀疏测量二部图

这样就形成了对于长度为 16 的稀疏信号 **x**（左节点）的 9 次测量：

$$
\begin{aligned}
&y_1 = y_3 = y_4 = y_9 = 0,\\
&y_2 = x[1]+x[4]+x[8],\\
&y_5 = x[1],\\
&y_6 = x[13],\\
&y_7 = x[8],\\
&y_8 = x[4]+x[13].
\end{aligned}
$$

根据稀疏测量二部图的连通性，我们将与右节点分为以下三种类型：

1. **零节点**：若一个右节点不包含任何非零元素，则称其为零节点，如图2.1的 y_1。
2. **单节点**：若一个右节点仅包含一个非零元素，则称其为单节点，如图2.1的 y_5。
3. **多节点**：若一个右节点包含一个以上的非零元素，则称其为多节点，如图2.1的 y_2。

其实单节点就是章节1.1.3描述的单一颜色的元素，多节点就是混合颜色。我们从单节点入手进行逐步剥离解码，具体算法将在第3.3章介绍。

为了判断观测向量中的元素（右节点）属于哪种节点，我们给每个左节点不同的 2 维向量加权，于是每个右节点可被表示为几个二维向量之和。为此我们设计一个列数与信号 $\mathbf{x}$ 长度相同的检测矩阵 $\mathbf{S}$：

$$\mathbf{S} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W & W^2 & W^3 & W^4 & \dots & W^{15} \end{bmatrix}, \tag{3}$$

其中 $W = e^{i\frac{2\pi}{N}}$ 是一个 N 次单位根，这里的例子中 $N = 16$。$\mathbf{S}$ 是一个 16×16 的 DFT 矩阵的前两行，通过 $\mathbf{S}$ 检测节点的具体算法将在章节3.3介绍。

除此之外，我们还需要定义一个编码矩阵 $\mathbf{H}$ 来表示二部图左右节点的连接，$\mathbf{H}$ 在这个例子中是一个 9×16 的由 0 和 1 组成的邻接矩阵。

为了将 $\mathbf{H}$ 和 $\mathbf{S}$ 结合在一起形成完整的测量矩阵 $\mathbf{A}$，下面我们介绍**行张量算子** $\boxtimes$（row-tensor operator）的定义。一般情况下，检测矩阵 $\mathbf{S} = [\mathbf{s}_0, \cdots, \mathbf{s}_{N-1}] \in \mathbb{C}^{M_2\times N}$，邻接矩阵 $\mathbf{H} = [\mathbf{h}_0, \cdots, \mathbf{h}_{N-1}] \in \mathbb{C}^{M_1\times N}$。行张量运算 $\mathbf{H}\boxtimes\mathbf{S}$ 本质上就是将矩阵 $\mathbf{H}$ 的每一行通过与矩阵 $\mathbf{S}$ 中每个相应的列进行逐个元素的增广。我们得到的行张量积是一个维数为 $M_1M_2\times N$ 的矩阵：

$$\mathbf{H}\boxtimes\mathbf{S} = [\mathbf{h}_0\otimes\mathbf{s}_0, \cdots, \mathbf{h}_{N-1}\otimes\mathbf{s}_{N-1}],$$

其中 $\otimes$ 是一个标准的克罗内克积（Kronecker product）。下面我们举一个简单例子来具象化这一过程。我们设定

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}, \tag{4}$$

$$\mathbf{S} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 & W^4 & W^5 & W^6 \end{bmatrix}, \tag{5}$$

这里 $W = e^{i\frac{2\pi}{7}}$。那么我们就可以计算出行张量积：

$$\mathbf{H} \boxtimes \mathbf{S} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & W & 0 & W^3 & 0 & W^5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & W & 0 & W^3 & 0 & 0 & W^6 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & W^3 & W^4 & 0 & W^6 \end{bmatrix}. \tag{6}$$

在本例中，$\mathbf{H}$ 有三行，分别与 $\mathbf{H} \boxtimes \mathbf{S}$ 的前两行，中间两行，后两行对应。

用这种方式，我们可以生成一个测量矩阵 $\mathbf{A} = \mathbf{H} \boxtimes \mathbf{S} \in \mathbb{C}^{M_1 M_2 \times N}$，这样得到观测值 $\mathbf{y} \in \mathbb{C}^{M_1 M_2}$。

定义 2.1 (测量矩阵) 设 $M = RP$ 且 $P, R \in \mathbb{N}^*$。给定一个 $R \times N$ 的编码矩阵 $\mathbf{H}$ 和一个 $P \times N$ 的检测矩阵 $\mathbf{S}$，那么 $M \times N$ 的测量矩阵 $\mathbf{A}$ 就可被定义为：

$$\mathbf{A} = \mathbf{H} \boxtimes \mathbf{S}, \tag{7}$$

其中 $\boxtimes$ 是行张量算子，并且编码矩阵和检测矩阵满足下述条件：

- 编码矩阵 $\mathbf{H} = [H_{r,n}]_{R \times N}$ 是一个关于二部图 $\mathcal{G}$ 的邻接矩阵。图 $\mathcal{G}$ 包含 N 个左节点 $V_1 := [N]$ 和 R 个右节点 $V_2 := [R]$，边的集合为 $\mathcal{E} := V_1 \times V_2$。
- 检测矩阵 $\mathbf{S} := [\mathbf{s}_0, \cdots, \mathbf{s}_{N-1}]$ 是一个 $N \times N$ 的 DFT 矩阵的前 P 行。

第三章 求解算法

本章将介绍在测量矩阵设计完毕的基础上，离群值去除以及信号恢复问题的具体算法求解。

3.1 节点类型检测

我们继续借用图2.1的例子来模拟算法。由公式 3 可知，我们可以通过 $\mathbf{S}$ 把每个右节点都记为一个二维向量 $\mathbf{y}_r = [y_r[0], y_r[1]]^T$，我们称每个这样的向量为一个**测量对**（measurement bin）。在上述例子中，右节点 1，2，5 的测量就可以被记为：

$$\begin{aligned}
&\mathbf{y}_1 = \mathbf{0},\\
&\mathbf{y}_2 = x[1]\times\begin{bmatrix}1\\ W\end{bmatrix} + x[4]\times\begin{bmatrix}1\\ W^5\end{bmatrix} + x[8]\times\begin{bmatrix}1\\ W^8\end{bmatrix},\\
&\mathbf{y}_5 = x[1]\times\begin{bmatrix}1\\ W\end{bmatrix}.
\end{aligned}$$

于是通过对测量对的检测，我们能够有效地确定右节点是零节点、单节点还是多节点。我们由右节点 1，2，5 的例子介绍节点检测的方法：

1. **零节点对**：参考右节点 1。若测量对是一个全 0 向量，即 $\mathbf{y}_r = \mathbf{0}$，则该节点为零节点。
2. **单节点对**：参考右节点 5。这个测量对满足比值检测：

$$\hat{k} = \frac{\angle y_5[1]/y_5[0]}{2\pi/16} = 1,$$

其中 $\hat{k}$ 为整数，则该节点就是单节点。于是

$$\hat{x}[\hat{k}] = y_5[0].$$

我们也就从中顺利恢复了信号 $\mathbf{x}$ 中的非零元素 x_1。另一种检测方法是，这样的单节点对的两个元素的模一定相等，即 $|y_3[0]| = |y_3[1]|$。因为 W 是单位根，$|W^r| = 1$。

3. **多节点对**：参考右节点 2。对这个测量对进行比值检测：

$$\hat{k} = \frac{\angle y_2[1]/y_2[0]}{2\pi/16} = 4.85995.$$

并且两个元素的模并不相等，$|y_2[1]| \neq |y_2[0]|$。在这种比值检测的结果 $\hat{k}$ 不为非零整数或两个元素不同模的情况下，该节点为多节点。

综上，要判断任意非零节点对 y_r 的类型，我们只需计算

$$\hat{k} = \frac{\angle y_r[1]/y_r[0]}{2\pi/N}, \tag{8}$$

, 当 $\hat{k}$ 为整数，则该节点是单节点；反之则为多节点。

3.2 离群值去除

离群值（outliers），也被称为异常值，是指与其他观察结果显著不同的数据点。存在离群值情况下，压缩观测过程可被描述为 $\mathbf{y} = \mathbf{Ax} + \mathbf{f}$。其中 $\mathbf{f} = [f_1, f_2, \ldots, f_M]^T$，且 $|\text{supp}(\mathbf{f})| = T = \eta M$ $(0 \le \eta < \frac{1}{2})$。我们设初始稀疏度为 K 的稀疏信号 $\mathbf{x}$ 中的非零元素服从均值为 0，标准差为 1 的高斯分布，即 $x_i \sim N(0,1), i \in [N]$。稀疏度为 η 的离群向量 $\mathbf{f}$ 中的非零元素满足 $f_j \sim N(0,100), j \in [M]$。

从中我们可以显然得到：$\mathbf{f}$ 中大部分非零元素的绝对值远大于 $\mathbf{x}$ 中元素的绝对值。为了尽可能剔除这些离群值，又考虑到 $\mathbf{y}$ 的稀疏性，我们可以尝试取 $\mathbf{y}$ 的非零元素绝对值的某个分位数 θ_p(p 是一个分位数算子，常取 0.5、0.75)，并且设定一个倍率 α 得到一个分界数 $\alpha \cdot \theta_p$，抹去 $\mathbf{y}$ 的非零元素中所有绝对值大于 $\alpha \cdot \theta_p$ 的数的集合。这里“抹去”的定义是将这些非零元素设为 0，也就是这些元素不参与信号重构的过程。

用数学语言描述，我们把 $\mathbf{y}$ 的有效非零元素限制在一个指标集 $S = \{s : |y_s| < \alpha\theta_p(\{|y_j| : j \in \text{supp}(\mathbf{y})\})\}$ 上，其他指标的对应元素全部设为 0。例如，当 $\alpha = 1, p = \frac{1}{2}$ 时，我们就抹去所有绝对值大于非零元素绝对值中位数的数。这样就理论上去除了全部绝对值过大的离群值，接下来对被限制的 $\mathbf{y}$ 应用信号重构算法即可。

由于离群向量 $\mathbf{f}$ 的稀疏度 T 是未知的，我们无法简单地抹去绝对值最大的前 T 个元素，因此采取上述设置比例或分界数的方式。我们又知道 $\mathbf{y}$ 和 $\mathbf{f}$ 都是稀疏的，所以 $\mathbf{y}$ 和 $\mathbf{f}$ 的支撑集大概率不会存在元素位置重合的情况，不会抹去需要参与信号恢复的非离群值。因此，这样直接设为 0 的方式是合理且可取的。算法伪代码如下：

算法 1 离群值去除算法

1: **输入**：一个观测值 $\mathbf{y} \in \mathbb{C}^{2R}$（测量对形式），常数 α。
2: 首先计算得到 $\mathbf{y}$ 的非零元素模的中位数 $med = \textbf{Median}(\{|\mathbf{y}_j[0]| : \mathbf{y}_j[0] \neq 0, \mathbf{y}_j \in \mathbf{y}\})$;
3: **for** $r = 1$ to R **do**
4: **if** $|\mathbf{y}_r[0]| > \alpha \cdot med$ **then**
5: 将 $\mathbf{y}_r$ 抹去为 $\mathbf{0}$ 向量;
6: **end if**
7: **end for**

3.3 信号恢复

本节将聚焦于离群值已去除的信号恢复。经过上述铺垫，我们按照以下步骤解码即可（参考1.1.3）：

1) 选取二部图中所有使得右节点度数为 1 的边（找到所有单节点）
2) 去除（剥离）所有这些边以及与之对应的左右节点对。
3) 去除（剥离）步骤2)中剥离的左节点的其他未被剥离的边。
4) 找到步骤3)中去除的左节点连接的所有右节点，将这些左节点的值从右节点的值中减去。

伪代码如下：

算法 2 剥离解码器

输入：一个观测值 $\mathbf{y} \in \mathbb{C}^{2R}$ （测量对形式），待恢复信号 $\hat{\mathbf{x}} = \mathbf{0}$。
for $i = 1, 2, \ldots$ 直到 $\mathbf{y}^{(i)} = \mathbf{0}$ 或找不到单节点 **do**
    for $r = 1$ to R **do**
        由公式 8 计算 $\hat{k}$，判断 $\mathbf{y}_r^{(i)}$ 是否为单节点对；
        if $\mathbf{y}_r^{(i)}$ 是单节点 **then**
            $\hat{\mathbf{x}}[\hat{k}] = \mathbf{y}_r^{(i)}[0]$；
            for $r' = 1$ to R **do**
                找到连接左节点 $\hat{k}$ 的右节点 r'；
                剥离过程 $\mathbf{y}_{r'}^{(i+1)} = \mathbf{y}_{r'}^{(i)} - \hat{x}[\hat{k}]\mathbf{s}_{\hat{k}}$，这里 $\mathbf{s}_{\hat{k}}$ 是检测矩阵 $\mathbf{S}$ 的第 $\hat{k}$ 列；
            end for
        else
            继续下一个节点对 $\mathbf{y}_{r+1}^{(i)}$。
        end if
    end for
end for

第四章 数值实验结果

本章我们对无噪声条件下存在离群值的信号恢复问题进行数值模拟，从实验角度出发论证理论内容。在本章的实验中，我们通过上一章描述的剥离算法对不同情况的离群值问题进行求解。先验证算法在不同情形下的高效性，再与其他算法做比较来证明我们算法的优越性，之后比较观测离群值去除算法参数不同时的恢复效果。

我们将实验的参数统一设置为：$N = 200$；初始稀疏信号 $\mathbf{x} \in \mathbb{R}^N$ 且其中元素服从 $x_i \sim N(0,1)$；邻接矩阵 $\mathbf{H} \in \{0,1\}^{M\times N}$；检测矩阵 $\mathbf{S}$ 为 $N\times N$ 的 DFT 矩阵前两行；离群向量 $\mathbf{f}$ 中的非零元素服从 $f_j \sim N(0,100)$。本文中数值实验均由 Python3.11.3 实现，在 macOS Sonoma 14.2.1 操作系统下，用一块 Apple M2 Pro（CPU）芯片完成运行。

为了衡量信号恢复的误差，我们定义评价指标**精确恢复率**为多次重复实验中恢复结果误差 $err < thr\_err$ 的次数与总实验次数的比值。这里 $err = \frac{||x-\hat{x}||_2}{||x||_2}$，其中 x 是初始信号，$\hat{x}$ 为恢复结果；thr_err 是界定一次信号恢复是否成功的阈值。在实验的大多数情况下我们取 $thr\_err = 10^{-7}$；而由于我们将对比的传统算法（PLAD 算法）无法达到这样的准确度，我们在对比实验中取 $thr\_err = 10^{-2}$。当 $err < thr\_err$，我们称信号被精确恢复一次。

4.1 信号恢复效率

在本节中我们先通过几个实验，用信号精确恢复所需测量次数和恢复用时两个角度来评估我们算法的恢复精确度及高效性。

4.1.1 不同测量次数下的恢复效率

在第一个实验中，测量值数目 M 作为自变量。我们设定 $\mathbf{x}$ 的稀疏度为 2；$\mathbf{H}$ 为每列有且仅有三个 1 的矩阵，即表示正则度为 3 的左正则二部图；离群值比例 η 为 0.01；离群值去除算法中 $\alpha = 2, p = \frac{1}{2}$；重复实验次数为 500 次。结果如下：

表 4.1 不同测量次数下信号恢复时间及恢复效果

测量值数目 M	10	20	40	80	100	120
精确恢复率	70.0%	84.5%	90.5%	92.5%	95.5%	95.0%
平均用时（毫秒）	48.55	79.25	89.70	95.40	115.25	126.50

由表4.1可知仅需 100 次的测量就可 95% 精确地恢复存在一个离群值的稀疏度为 2 的

信号 $\mathbf{x} \in \mathbb{R}^{200}$，且在 500 次随机实验中每次恢复所需时间仅 0.115 秒左右。

4.1.2 不同信号稀疏度下的恢复效率

在第二个实验中，信号稀疏度 K 作为自变量。我们设定 $M = 100$；离群值去除算法中 $\alpha = 2, p = \frac{3}{4}$；其他与上一个实验相同。结果如下：

表 4.2 不同信号稀疏度下信号精确恢复的概率

稀疏度 K	1	2	3	4	5
精确恢复率	98.0%	95.5%	92.2%	91.7%	88.8%
平均用时（毫秒）	124.0	129.0	129.6	125.4	130.4

在表4.2我们可以看到信号稀疏度越小，恢复越精确；平均每次信号恢复时间为 0.124 ~ 0.130 秒，总体上稀疏度 K 越大恢复所需时间越长。

4.1.3 不同离群值比例下的数值效果

在离群向量 $\mathbf{f}$ 中 $|\mathrm{supp}(\mathbf{f})| = \eta M$。显然，随着离群值比例 η 的增大，信号恢复的难度也越来越大。本次实验中我们探索离群值比例变化给恢复效果带来的变化程度，设定信号稀疏度 $K = 2$，离群值去除算法中 $\alpha = 1, p = \frac{1}{2}$，其他与上一小节相同。由于本次实验中恢复时间基本相同（最大差值不超过 5 毫秒），我们不再列举平均每次恢复用时。实验结果如下：

表 4.3 不同离群值比例下信号精确恢复的概率

离群值比例 η	0.01	0.02	0.03	0.04	0.05
精确恢复率	91.2%	90.4%	88.0%	85.6%	82.8%

这里信号恢复效果较差，是因为离群值比例变化下，中位数倍率并没有变化。若对每一个 η 下的中位数倍率 α 进行微调，我们能得到每个精确恢复率都高于 90% 的效果。由表4.3可见当离群值比例 η 大于 0.02 时，算法恢复效果显著降低，且精确恢复率下降速度极快。

4.2 离群值去除算法的参数大小对数值效果的影响

本节实验中我们探究离群值去除算法的参数大小（包括 α 和 p）对恢复效果的影响。在直觉上，离群值比例越高，倍率 α 就要更小（或 p 更小）来保证去除所有离群值。这里我们选取 $\mathbf{x}$ 的稀疏度 $K = 2$，分位数 θ_p 中的分位数算子 $p = 0.5$，$\mathbf{H}$ 和评价指标均与上述实验

相同。这里只讨论倍率 α 的影响，是因为由章节3.2可知，离群值去除的分界数是 $\alpha\theta_p$，因此模拟的数值实验中 α 的变化和 p 的变化是等价的。我们通过离群值比例 η 不同的实验，对精确恢复率随倍率 α 的变化进行展示。结果如下图：

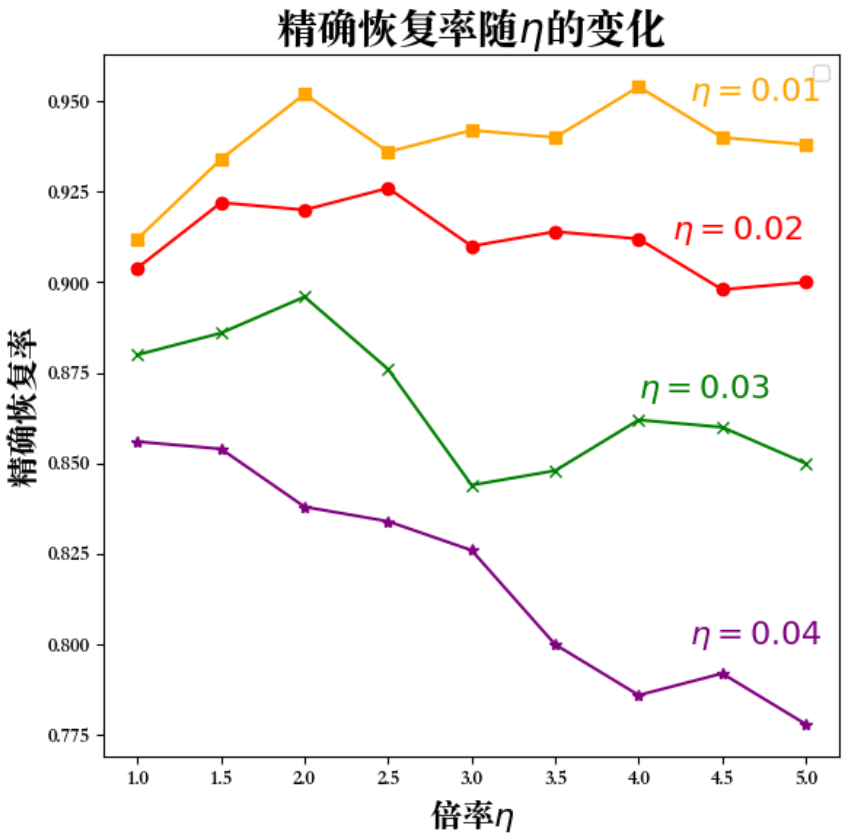

图 4.1　不同离群值比例下的恢复效果随算法参数变化展示。图中横坐标表示不同的倍率 α，纵坐标表示精确恢复率，四条曲线分别表示离群值比例取 1%,2%,3%,4% 的情形。

由图4.1可知，随离群值比例增大，我们需要调小倍率 α（或调小 p）来保证较好的恢复效果。当离群值比例超过 0.03 时，精确恢复概率无法达到期望。

因此，当我们在解决离群值极少的任务时，可以选择较大的倍率 α 或分位数算子 p；而在大多数情况下，选择小的倍率显然是更加稳定的。

4.3　与其他信号恢复算法的比较

为了说明我们提出的算法的存在意义与优越性，我们将算法与其他传统算法对比，比较同等情况下两者的精确恢复率和恢复所需时间。我们选择对比算法为 ℓ_1-PLAD 算法（Penalized Least Absolute Deviation），因为这是信号恢复领域非常经典且能够解决离群值问题的算法。其他算法如 LASSO、DS、RPDS 等算法都无法去除偏差较大的离群值，故不做比

较。ℓ_1-PLAD 算法的原理可以简单描述为：

$$\min_{\mathbf{x}\in\mathbb{R}^N} ||\mathbf{y}-\mathbf{A}\mathbf{x}||_1+\lambda||\mathbf{x}||_1 \tag{9}$$

其中 $\mathbf{A}$ 是一个 $M\times N$ 的高斯测量矩阵。

在本节实验中，我们设置 $\mathbf{x}$ 的稀疏度为 3，$M=100$，离群值比例取 0.02 和 0.03，编码矩阵 H 和重复实验次数与上一节相同。我们的算法中 $\alpha=2, p=\frac{1}{2}$；ℓ_1-PLAD 算法中迭代次数为 2000 次，λ 作为自变量，且 $\lambda\in[0.006, 0.009]$（经检验，λ 超出这一区间的 PLAD 算法精确恢复率无法高于 50%）。另外与本文其他实验不同的是，这里 $thr\_err=10^{-2}$，因为 PLAD 算法无法达到我们的算法能达到的 $err<10^{-7}$ 的精度。实验结果如下（下表中间部分的数值为精确恢复率）：

表 4.4 我们的离群值去除算法与 PLAD 算法的对比

应用算法	我们的算法	$\text{PLAD}_{\lambda=0.006}$	$\text{PLAD}_{\lambda=0.007}$	$\text{PLAD}_{\lambda=0.008}$	$\text{PLAD}_{\lambda=0.009}$
$\eta=0.02$	92.0%	78.2%	94.6%	91.2%	79.8%
$\eta=0.03$	89.8%	47.8%	66.4%	92.2%	92.0%
平均用时（毫秒）	129.0	1729.9	1268.3	1692.2	2374.5

由此可见 PLAD 算法的平均恢复所需用时远高于本文提出的算法，在最好情况下也有 10 倍的差距，因此 PLAD 显然无法用于实时的存在离群值的信号恢复任务。

另外，PLAD 算法虽然在一定条件下能达到更加精确的结果，但是总体恢复效果受参数 λ 大小的影响极严重。例如，由表4.4，倘若我们取 $\lambda=0.007$，这一算法在离群值只占 0.02 比例时恢复效果极好，而一旦离群值比例轻微变化，上升到 0.03，精确恢复率就急剧下降到小于 70%。而由4.2节可知，我们提出的算法受参数大小影响并不严重。因此，我们的算法相较 PLAD 更能胜任多情形下（更一般）的快速信号恢复任务。

参考文献

[1] Donoho D. Compressed sensing[J]. IEEE Transactions on Information Theory, 2006, 52(4):1289–1306.

[2] Lustig M, Donoho D, Pauly J M. Sparse mri: The application of compressed sensing for rapid mr imaging[J]. Magnetic resonance in medicine, 2007, 58(6):1182—1195.

[3] Wiaux Y, Jacques L, Puy G, et al. Compressed sensing imaging techniques for radio interferometry[J]. Monthly Notices of the Royal Astronomical Society, 2009, 395(3):1733–1742.

[4] Elad M. Sparse and redundant representations: from theory to applications in signal and image processing[M]. Springer Science & Business Media, 2010.

[5] Gilbert A, Indyk P. Sparse recovery using sparse matrices[J]. Proceedings of the IEEE, 2010, 98(6):937–947.

[6] Gastpar M, Bresler Y. On the necessary density for spectrum-blind nonuniform sampling subject to quantization[C]. In 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100), volume 1, 348–351 vol.1, 2000.

[7] Wainwright M J. Information-theoretic limits on sparsity recovery in the high-dimensional and noisy setting[Z], 2007.

[8] Candès E J, Plan Y. Near-ideal model selection by 1 minimization[J]. The Annals of Statistics, 2009, 37(5A):2145 – 2177.

[9] Cai T T, Wang L. Orthogonal matching pursuit for sparse signal recovery with noise[J]. IEEE Transactions on Information Theory, 2011, 57(7):4680–4688.

[10] Bakshi M, Jaggi S, Cai S, et al. Sho-fa: Robust compressive sensing with order-optimal complexity, measurements, and bits[J]. IEEE Transactions on Information Theory, 2016, 62(12):7419–7444.

[11] Li X, Yin D, Pawar S, et al. Sub-linear time support recovery for compressed sensing using sparse-graph codes[J]. IEEE Transactions on Information Theory, 2019, 65(10):6580–6619.

[12] Yang J, Zhang Y. Alternating direction algorithms for ℓ_1-problems in compressive sensing[J]. SIAM Journal on Scientific Computing, 2011, 33(1):250–278.

[13] Wang L. The ℓ_1 penalized lad estimator for high dimensional linear regression[J]. Journal of Multivariate Analysis, 2013, 120:135–151.

附　　录

A.1　主算法代码

主程序文件

```
# peeling_decoder_vector.py

import random
import time
import numpy as np
import scipy as sp
import cmath
from signal_vectors import signal_vec, outliers
from scipy.sparse import csr_matrix
from numpy import linalg as LA

# parameters to be modified
n = 200 # N
R = 100 # M
k = 2 # sparsity of x
density_H = 3
num_outliers = 1
med_times = 2
percent = 50 # p
var_outliers = 100 # variance of outliers
thr_err = 1e-7
num_epoch = 500

def input_vector(n, k):
    x = signal_vec(n, k)
    return np.squeeze(x)

def H_measure(row, col, k):
    res = np.zeros((row, col))
    for i in range(col):
        rvs = sp.stats.bernoulli(1).rvs
        S = sp.sparse.random(row, 1, density=k/row, data_rvs=rvs)
```

```
            res[:, i] = S.toarray().reshape((row, ))
        return res

    def S_matrix(n, W):
        S = np.zeros((2, n), dtype=complex)
        for i in range(n):
            S[:, i] = np.array([[1], [W**i]], dtype=complex).reshape(2, )
        return S

    def kron_product(a, b):
        res = np.empty(0)
        for elem in a:
            res = np.append(res, np.dot(elem, b.T))
        return res.T

    def B_matrix(H, S, n):
        B = np.empty((2 * H.shape[0], H.shape[1]), dtype=complex)
        for i in range(n):
            B[:, i] = kron_product(H[:, i], S[:, i])
        return B

    def isSingleton(y, j, n):
        if (y[2*j].real == 0):
            return 0
        l_hat = cmath.phase(y[2*j+1]/y[2*j]) / (2*np.pi/n)
        if (abs(round(l_hat) - l_hat) < 0.001):
            return l_hat
        return 0

    def add_outliers(y_hat, y, num_outliers, med_times, var, R, W):
        for i in range(num_outliers):
            rand_row = random.randint(0, R-1)
            out_val = np.random.normal(0, var_outliers)
            y[rand_row] += out_val
            y_hat[rand_row*2] += out_val
            y_hat[rand_row*2+1] += out_val * W
        y_supp = np.transpose(np.nonzero(y_hat))
        y_supp_elem = []
        for index in y_supp:
            if index%2 == 0:
```

```
73                  y_supp_elem.append(abs(y_hat[index][0]))
74          y_supp_elem = np.array(y_supp_elem)
75          med = np.percentile(y_supp_elem, percent)
76          for index in y_supp:
77              if index%2 == 0:
78                  if abs(y_hat[index]) > med_times*med:
79                      y_hat[index] = 0
80                      y_hat[index+1] = 0.
81          return y_hat
82
83      def peeling_decoder():
84          total_err = 0.
85          num_success = 0
86          start_time = time.time()
87
88          for epoch in range(num_epoch):
89              x = input_vector(n, k)
90              W = cmath.exp(2 * np.pi * cmath.sqrt(-1) / n)
91              H = H_measure(R, n, density_H) # (R, n)
92              S = S_matrix(n, W)
93              B = B_matrix(H, S, n)
94              y = np.dot(H, x)
95              y = np.squeeze(y)
96              y_hat = np.dot(B, x) # y_hat = np.zeros(R*2, dtype=complex)
97              y_hat = np.squeeze(y_hat)
98              y_hat = add_outliers(y_hat, y, num_outliers, med_times,
                    var_outliers, R=R, W=W)
99              x_hat = np.zeros(n)
100
101             err = 1.
102             while (err > thr_err):
103                 flag = False
104                 for r in range(R):
105                     if isSingleton(y_hat, r, n) != 0:
106                         flag = True
107                         l_hat = round(isSingleton(y_hat, r, n))
108                         x_hat[l_hat] = y_hat[2*r].real
109                         y_hat[2*r] = 0
110                         y_hat[2*r+1] = 0
111                         H[r, l_hat] = 0
```

```
                        for j in range(R):
                            if (H[j, l_hat] > 0 and j != r):
                                r1 = j
                                H[j, l_hat] = 0
                                y_hat[r1*2] = y_hat[r1*2] - x_hat[l_hat]
                                y_hat[r1*2+1] = y_hat[r1*2+1] - x_hat[
                                    l_hat] * np.power(W, l_hat)
                                break
                    else:
                        continue
                if (np.all(y_hat.real < 1e-5)):
                    break
                elif(not flag):
                    print("There is no single-ton but still multi-ton in
                        y_hat!\n")
                    break

            err = LA.norm(x_hat - x) / LA.norm(x)
            if err < thr_err:
                num_success += 1

            if (epoch+1) % 100 == 0:
                print("time of succ in {} epochs: ".format(epoch+1),
                    num_success)

        end_time = time.time()
        print("The average time cost per-epoch is :, (end_time -
            start_time)/num_epoch)

    if __name__ == '__main__':
        peeling_decoder()
```

A.2 对比算法代码

用于对比的 ℓ_1-PLAD 算法文件

```
# PLAD.py

import numpy as np
import time
from signal_vectors import signal_vec, outliers
```

```
6
7     n = 200
8     m = 100
9     lam = 0.006
10    k = 2
11    thr_err = 1e-2
12    num_epoch = 500
13
14    def A_Gauss(m, n, mean, var):
15        Matrix = np.random.normal(mean, var**0.5, size=[m, n])
16        Matrix = np.mat(Matrix)
17        return Matrix
18
19    def SoftThreshold(b, lambd):
20        xx = np.maximum(np.abs(b) - lambd, 0)
21        return np.multiply(np.sign(b),xx)
22
23    def PLAD(lambd, iterN, m, n, A, x0, e):
24        b = A * x0 + e
25        x = np.zeros((n, 1), dtype=complex)
26        z = np.zeros((m, 1), dtype=complex)
27        AA = np.dot(A.T, A)
28        a, v = np.linalg.eig(AA)
29        L = np.max(a)
30        t = 1 / L ** 2
31        for k in range(iterN):
32            alfa = 1/L
33            x = SoftThreshold(x - alfa * A.T * z, alfa * lambd)
34            z = z + t * (A * x -b)
35            for i in range(m):
36                if z[i,0] > t:
37                    z[i, 0] = t
38                if z[i,0] < -t:
39                    z[i, 0] = -t
40        return x, x0, np.linalg.norm((x-x0), ord=2) / np.linalg.norm(x0,
              ord=2), max(np.abs(A.T*e))
41
42    if __name__ == '__main__':
43        num_succ_PLAD = 0
44        time_sum = 0.
```

```
45
46          for epoch in range(num_epoch):
47              A = A_Gauss(m, n, 0, 1 / m)
48              x0 = signal_vec(n, k)
49              e = outliers(m,  3, 100)
50
51              start_time = time.time()
52
53              _, _, E_PLAD, _ = PLAD(lam, 2000, m, n, A, x0, e)
54
55              end_time = time.time()
56              time_sum += (end_time - start_time)
57
58              if E_PLAD < thr_err:
59                  num_succ_PLAD += 1
60
61              if (epoch + 1) % 100 == 0:
62                  print("time of succ in {} epochs: ".format(epoch+1),
                        num_succ_PLAD)
63
64          print("The average time cost per-epoch is :, time_sum/num_epoch)
```

A.3 信号初始化代码

信号及离群值初始化文件

```
1   # signal_vectors.py
2
3   import numpy as np
4   import scipy as sp
5
6   # k: signal vectors are k-sparse
7   def signal_vec(n, k, var=1):
8       rvs = sp.stats.norm(loc=0, scale=var).rvs
9       S = sp.sparse.random(n, 1, density=k/n, data_rvs=rvs)
10      return S.toarray()
11
12  def outliers(m, k, var=100):
13      rvs = sp.stats.norm(loc=0, scale=var).rvs
14      S = sp.sparse.random(m, 1, density=k / m, data_rvs=rvs)
15      return S.toarray()
```

致　　谢

毕业论文（设计）成绩表

导师评语 建议成绩_____　　指导教师（签字）_____
答辩委员会意见 答辩委员会负责人（签字）_____
成绩_____　　学院（盖章）_____ 年　月　日