第 4 章　稀疏表示系数的求解

4.1　稀疏表示的基本概念

稀疏编码的概念最早来自神经生物学。人眼视觉感知机理的研究表明，人眼视觉系统(human visual system，HVS)可看成是一种合理而高效的图像处理系统。在人眼视觉系统中，从视网膜到大脑皮层存在一系列细胞，以“感受野”模式描述。感受野是视觉系统信息处理的基本结构和功能单元，是视网膜上可引起或调制视觉细胞响应的区域，它们被视网膜上相应区域的光感受细胞所激活，对时空信息进行处理。神经生理研究已表明：在初级视觉皮层(primary visual cortex)下细胞的感受野具有显著的方向敏感性，单个神经元仅对处于其感受野中的刺激做出反应，即单个神经元仅对某一频段的信息呈现较强的反映，如特定方向的边缘、线段、条纹等图像特征，其空间感受野被描述为具有局部性、方向性和带通特性的信号编码滤波器[34]。

生物学家提出，哺乳类动物在长期的进化中，生成了能够快速、准确、低代价地表示自然图像的视觉神经方面的能力。可以直观地想象，人们的眼睛每看到的一幅图像都是上亿像素的，而每一幅图像都只用很少的代价重建与存储，这种方式叫做稀疏编码(sparse coding，SC)，也叫稀疏表示(sparse representation，SR)。从数学的角度来说，稀疏编码的目的是在大量的数据集中，选取很小的一部分作为元素来重建新的数据。从实际应用角度来看，一般场景下信号采集设备仅仅能够采集信号本身。用于稀疏表示的字典和稀疏表示系数均为未知项。因此，稀疏表示的主要难点包括两个方面：稀疏表示字典的生成和稀疏表示系数的计算。

从稀疏表示字典生成情况来看，在稀疏表示理论未提出前，正交字典和双正交字典因为其数学模型简单而被广泛地应用，然而他们有一个明显的缺点就是自适应能力差，不能灵活全面地表示信号。1993 年，Mallat[35]基于小波分析提出了信号可以用一个超完备字典进行表示，从而开启了稀疏表示的先河。经研究发现，信号在稀疏表示过程中，可以根据信号的自身特点自适应地选择合适的超完备字典。

稀疏表示系数的计算方法首先是由 Mallat 提出的，也就是众所周知的匹配追踪(matching pursuit，MP)算法，该算法是一个迭代算法，简单且易于实现，因此得到了广泛的应用。随后，Tropp 等基于 MP 算法，提出了正交匹配追踪(orthogonal matching pursuit，OMP)算法[36]，OMP 算法相较于 MP 算法，其收敛速度更快。在此后的研究中，为了改进 OMP 算法，学者们提出了各种不同的优秀算法[37]。

4.1.1 稀疏表示研究的关键问题

信号稀疏表示主要包括两个阶段：字典构建阶段和稀疏编码阶段，因此，稀疏表示的两大主要任务就是字典的训练和信号的稀疏表示系数求解。

稀疏表示字典的构建，一般可以选择分析字典和学习字典这两种不同类型的字典。常用的分析字典包括：小波字典、超完备离散余弦变换字典和曲波字典等，用这类字典进行信号的稀疏表示时，虽然简单易实现，但信号的表达形式单一且不具备自适应性；反之，学习字典的自适应能力强，能够更好地适应不同的图像数据。近年来，有众多关于学习字典的稀疏表示算法，如最优方向(method of optimal directions，MOD)算法，基于超完备字典稀疏分解的 K-SVD 算法[37]等。

稀疏表示系数的求解，是稀疏表示过程中研究的另一个重要问题。在假设其字典已知的情况下，稀疏表示系数的求解为不确定方程组的求解，所以通过添加不同的范数约束来使解去逼近一个确定值，本章将着重介绍一些稀疏表示系数的求解算法，包括经典的 MP 算法、OMP 算法，还有一些迭代收缩算法等。字典构建阶段的相关问题将在第 5 章进行介绍。

4.1.2 稀疏表示的模型

假设 $\boldsymbol{b}\in R^n$ 为信号采集器采集到的信号，$\boldsymbol{A}\in R^{n\times m}$ 为过完备字典，$\boldsymbol{x}\in R^m$ 为稀疏表示系数。这三个变量之间的关系为

$$\boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.1}$$

其中，$\boldsymbol{x}$ 应当是稀疏的，即 $\boldsymbol{x}$ 的 m 个元素中只有少数为非 0 值。当 $\boldsymbol{A}$ 为已知时，$\boldsymbol{x}$ 就可以在 $\boldsymbol{A}$ 张成的空间中利用 $\boldsymbol{A}$ 的少数原子来表示信号 $\boldsymbol{b}$。

稀疏表示是针对不确定方程组的求解问题而提出的。对不确定方程组 $\boldsymbol{b}=\boldsymbol{A}\boldsymbol{x}$，$\boldsymbol{A}\in R^{n\times m}$ 且 $n<m$，在 $\boldsymbol{A}$ 为满秩的条件下，方程数目小于未知数的个数，方程组有无数组解，此问题称为欠定问题。此时，解决此类问题的其中一种思路是：给不确定的解增加正则化约束，缩小不确定解的范围，从而使得方程组的不确定解逼近一个确定的解。稀疏表示模型是将解的稀疏性作为不确定性方程组的一种约束，从而使方程组有唯一解。

稀疏表示有以下三种常用模型。

第一种模型：

$$\min_{\boldsymbol{x}}\|\boldsymbol{x}\|_2,\ \text{s.t.}\ \ \boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.2}$$

第二种模型：

$$\min_{\boldsymbol{x}}\|\boldsymbol{x}\|_0,\ \text{s.t.}\ \ \boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.3}$$

第三种模型：

$$\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_1, \text{ s.t. } \boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.4}$$

其中，$\boldsymbol{A}$ 表示字典，$\boldsymbol{x}$ 为需要恢复的高分辨率目标图像，$\boldsymbol{b}$ 为观测到的低分辨率图像。

4.1.3　稀疏表示的应用

目前，稀疏表示主要应用于自然信号形成的图像、音频以及文本等，在应用方面，可大体划分为两类：基于重建的应用和基于分类的应用[34]。

基于重建的应用主要有图像去噪、压缩与超分辨率、合成孔径雷达(synthetic qperture radar，SAR)成像、缺失图像重建以及音频修复等。这些应用主要将目标的特征用若干参数来表示，这些特征构成稀疏向量，稀疏向量由稀疏表示方法求解，稀疏向量还可以对数据或图像进行重建。在这些应用中，观测数据一般含有噪声，可以用稀疏表示所提供的参数对噪声进行去除。

基于分类的应用主要是用表示对象主要的或本质的特征构造稀疏向量，这些特征具有类间的强区分性。首先利用稀疏表示方法得到这些特征的稀疏向量，再根据稀疏向量与某标准值之间的距离，或不同稀疏向量之间的距离进行判别，从而完成模式识别或分类的任务，如盲源分离、人脸识别、文本检测等都应用了这一原理。

4.2　稀疏表示系数的求解方法

本节将介绍一些经典的稀疏表示系数的具体求解算法。

对于传统的 l_2 范数约束下的稀疏表示模型，稀疏表示系数的求解可通过求解如下表达式得到：

$$\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_2, \text{ s.t. } \boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.5}$$

式中，$\boldsymbol{x}$ 为需要恢复的高分辨率目标图像，在 $\boldsymbol{b}=\boldsymbol{A}\boldsymbol{x}$ 成立的条件下，求未知变量 $\boldsymbol{x}$，此稀疏表示系数求解的典型方法为最小二乘法。

对于 l_0 范数约束下的系数表示模型，稀疏表示系数的求解可以表示为

$$\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_0, \text{ s.t. } \boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.6}$$

上式中各个字母的表示与上边描述的 l_2 范数约束下的模型中的表示一致。

此稀疏表示系数 $\boldsymbol{x}$ 求解的典型方法是匹配追踪算法，为了提高算法的精确性，在该算法的基础上引入正交匹配追踪算法。另外，为了进一步降低算法复杂度，接下来介绍最小二乘匹配追踪算法、Cholesky 快速正交匹配算法，以及块正交匹配算法。

由于上述的 l_0 范数约束为非凸函数，其求解过程是一个 NP 难问题，计算复杂。为了方便求解，常常将其近似改造为 l_1 范数的模型来进行求解：

$$\min_{x}\|\boldsymbol{x}\|_1,\ \text{s.t.}\ \ \boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.7}$$

上式中各个字母的表示与上边描述的 l_2 范数约束下的模型中的表示一致。

此稀疏表示系数 $\boldsymbol{x}$ 求解的常用方法包括基追踪算法、迭代重加权最小二乘算法、最小角度回归算法。

下面将具体介绍以上三种约束下的稀疏表示系数 $\boldsymbol{x}$ 的求解方法。此外，还要介绍一类经典的算法——迭代收缩算法，以及这类算法中的一些具体算法，如可分离代理算法、并行坐标迭代收缩算法等。

4.2.1 l_2 范数系数的求解方法

1. 最小二乘法的基本原理

最小二乘法(又称最小平方法)是一种广泛应用的数学优化方法。它通过最小化误差的平方和寻找数据的最佳匹配函数。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。最小二乘法还可用于曲线拟合和其他一些优化问题。

以最简单的一元线性模型来解释最小二乘法。什么是一元线性模型呢？监督学习中，若预测变量为离散变量，称之为分类；若预测变量为连续变量，称之为回归。回归分析中，如果只包含一个自变量和一个因变量，且二者的关系可以用一条直线近似表示，这种回归分析称之为一元线性回归分析。如果回归分析中包含两个或者两个以上的自变量，且因变量和自变量之间是线性关系，则称之为多元线性回归分析。对于二维空间线性是一条直线，对于三维空间线性是一个平面，对于多维空间线性是一个超平面。

对于一元线性回归模型，假设从总体中获取了 n 组观察值。对于平面中的这 n 个点，分别表示为 $(t_1,g_1),(t_2,g_2),\cdots,(t_n,g_n)$，可以使用无数条曲线来拟合。要求样本回归函数尽可能好地拟合这组值。综合起来看，这条直线处于样本数据的中心位置最合理。选择最佳拟合曲线的标准可以确定为：使总的拟合误差(即总残差)达到最小，有以下三个标准可以选择。

(1)用“残差和最小”确定直线位置是一个途径。但很快发现计算“残差和”存在相互抵消的问题。设直线方程为 $kt+v=g$，k 和 v 就是未知的直线斜率和截距。根据 n 个点求得的直线方程为

$$\begin{cases} kt_1+v=g_1' \\ kt_2+v=g_2' \\ \cdots \\ kt_n+v=g_n' \end{cases} \tag{4.8}$$

残差和的目标函数为

$$\begin{aligned}\min_{k,v} f(k,v) &= \min_{k,v}[(g_1 - g_1') + (g_2 - g_2') + \cdots + (g_n - g_n')] \\ &= \min_{k,v}[(g_1 - kt_1 - v) + (g_2 - kt_2 - v) + \cdots + (g_n - kt_n - v)]\end{aligned} \tag{4.9}$$

(2)用“残差绝对值和最小”确定直线位置也是一个途径，但绝对值在求取极值时方法较为复杂。

残差绝对值和的目标函数为

$$\begin{aligned}\min_{k,v} f(k,v) &= \min_{k,v}\left[\left|g_1 - g_1'\right| + \left|g_2 - g_2'\right| + \cdots + \left|g_n - g_n'\right|\right] \\ &= \min_{k,v}\left[\left|g_1 - kt_1 - v\right| + \left|g_2 - kt_2 - v\right| + \cdots + \left|g_n - kt_n - v\right|\right]\end{aligned} \tag{4.10}$$

(3)最小二乘法的原则是以“残差平方和最小”确定直线位置。用最小二乘法除了计算比较方便外，得到的估计量还具有优良特性，这种方法对异常值非常敏感。

残差平方和的目标函数为

$$\begin{aligned}\min_{k,v} f(k,v) &= \min_{k,v}[(g_1 - g_1')^2 + (g_2 - g_2')^2 + \cdots + (g_n - g_n')^2] \\ &= \min_{k,v}[(g_1 - kt_1 - v)^2 + (g_2 - kt_2 - v)^2 + + \cdots + \left(g_n - kt_n - v\right)^2]\end{aligned} \tag{4.11}$$

当所求的拟合函数并非直线，而变成超平面时，同样可以使用残差平方和作为目标函数。

假设有如下线性方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases} \tag{4.12}$$

上述方程组又可表示为

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \ , \quad (i = 1,2,3,\cdots,m) \tag{4.13}$$

其中，a_{ij} 表示在矩阵 $\boldsymbol{A}$ 中第 i 行第 j 列的数据，x_j 表示向量 $\boldsymbol{x}$ 的第 j 个元素，m 代表方程的个数，n 代表未知数的个数，将上述表达式写成矩阵和向量的形式为

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{4.14}$$

其中，$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$，$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$，$\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$。

引入残差平方和函数 $\boldsymbol{S}$：

$$\begin{aligned}\boldsymbol{S}(\boldsymbol{x}) &= \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2 = (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) \\ &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}+\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b}\end{aligned} \tag{4.15}$$

对 $\boldsymbol{S}(\boldsymbol{x})$ 做微分求最值，可得

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} \tag{4.16}$$

若矩阵 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 非奇异，则 $\boldsymbol{x}$ 有唯一解：

$$\hat{\boldsymbol{x}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} \tag{4.17}$$

以上线性方程组的求解过程即为利用最小二乘法求解线性方程组解的过程。另外，关于最小二乘法的证明，可通过以下实例来展示。

设 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$，已知 $\boldsymbol{A}=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}$，$\boldsymbol{x}=\begin{bmatrix}x_1\\ x_2\end{bmatrix}$，$\boldsymbol{b}=\begin{bmatrix}b_1\\ b_2\end{bmatrix}$。对于此问题来说，证明最小二乘法，实际上就是证明：

$$\frac{\partial\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2}{\partial\boldsymbol{x}} = 2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) \tag{4.18}$$

具体证明过程如下。

首先将 $\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2$ 展开为如下形式：

$$\begin{aligned}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}-(\boldsymbol{A}\boldsymbol{x})^{\mathrm{T}}\boldsymbol{b}+\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b} \\ &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}-2\boldsymbol{b}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b}\end{aligned} \tag{4.19}$$

将 $\boldsymbol{A}=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}$，$\boldsymbol{x}=\begin{bmatrix}x_1\\ x_2\end{bmatrix}$，$\boldsymbol{b}=\begin{bmatrix}b_1\\ b_2\end{bmatrix}$ 代入，

$$\begin{aligned}\text{原式} &= \begin{bmatrix}x_1 & x_2\end{bmatrix}\begin{bmatrix}a_{11} & a_{21}\\ a_{12} & a_{22}\end{bmatrix}\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}-2\begin{bmatrix}b_1 & b_2\end{bmatrix}\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}b_1 & b_2\end{bmatrix}\begin{bmatrix}b_1\\ b_2\end{bmatrix} \\ &= \begin{bmatrix}a_{11}x_1+a_{12}x_2 & a_{21}x_1+a_{22}x_2\end{bmatrix}\begin{bmatrix}a_{11}x_1+a_{12}x_2\\ a_{21}x_1+a_{22}x_2\end{bmatrix}-2\begin{bmatrix}b_1a_{11}+b_2a_{21} & b_1a_{12}+b_2a_{22}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} \\ &\quad +b_1^2+b_2^2 \\ &= (a_{11}x_1+a_{12}x_2)^2+(a_{21}x_1+a_{22}x_2)^2-2b_1a_{11}x_1-2b_2a_{21}x_1-2b_1a_{12}x_2-2b_2a_{22}x_2+b_1^2+b_2^2 \\ &= a_{11}^2x_1^2+a_{12}^2x_2^2+2a_{11}a_{12}x_1x_2+a_{21}^2x_1^2+a_{22}^2x_2^2+2a_{21}a_{22}x_1x_2-2b_1a_{11}x_1-2b_2a_{21}x_1- \\ &\quad 2b_1a_{12}x_2-2b_2a_{22}x_2+b_1^2+b_2^2\end{aligned} \tag{4.20}$$

将 $(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})$ 对 $\boldsymbol{x}$ 求偏导数，实际上是将原式分别对 x_1 和 x_2 求偏导数，有

$$\frac{\partial (Ax-b)^{\mathrm{T}}(Ax-b)}{\partial x}=\begin{bmatrix} 2a_{11}{}^2x_1+2a_{11}a_{12}x_2+2a_{21}{}^2x_1+2a_{21}a_{22}x_2-2b_1a_{11}-2b_2a_{21} \\ 2a_{12}{}^2x_2+2a_{11}a_{12}x_1+2a_{22}{}^2x_2+2a_{21}a_{22}x_1-2b_1a_{12}-2b_2a_{22} \end{bmatrix}$$

$$=\begin{bmatrix} 2a_{11}{}^2x_1+2a_{11}a_{12}x_2+2a_{21}{}^2x_1+2a_{21}a_{22}x_2 \\ 2a_{12}{}^2x_2+2a_{11}a_{12}x_1+2a_{22}{}^2x_2+2a_{21}a_{22}x_1 \end{bmatrix}-\begin{bmatrix} 2b_1a_{11}+2b_2a_{21} \\ 2b_1a_{12}+2b_2a_{22} \end{bmatrix}$$

$$=2\begin{bmatrix} (a_{11}{}^2+a_{21}{}^2)x_1+(a_{11}a_{12}+a_{21}a_{22})x_2 \\ (a_{11}a_{12}+a_{21}a_{22})x_1+(a_{12}{}^2+a_{22}{}^2)x_2 \end{bmatrix}-2\begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{4.21}$$

$$=2\begin{bmatrix} a_{11}{}^2+a_{21}{}^2 & a_{11}a_{12}+a_{21}a_{22} \\ a_{11}a_{12}+a_{21}a_{22} & a_{12}{}^2+a_{22}{}^2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}-2\begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$=2\begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}-2\begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$=2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})$$

即得

$$\frac{\partial (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})}{\partial \boldsymbol{x}}=2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) \tag{4.22}$$

原式得证。

2. 最小二乘法求解 l_2 范数

假设目标函数 $\boldsymbol{L}(\boldsymbol{x})$ 表达式为

$$\boldsymbol{L}(\boldsymbol{x})=\|\boldsymbol{x}\|_2^2+\boldsymbol{\lambda}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) \tag{4.23}$$

设 $\boldsymbol{x}=\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$，$\boldsymbol{A}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$，$\boldsymbol{b}=\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$，$\boldsymbol{\lambda}=\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$。

则 $\boldsymbol{L}(\boldsymbol{x})$ 可以写成

$$\boldsymbol{L}(x)=[x_1 \quad x_2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}+[\lambda_1 \quad \lambda_2]\left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}-\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right) \tag{4.24}$$

$$=x_1{}^2+x_2{}^2+\lambda_1(a_{11}x_1+a_{12}x_2-b_1)+\lambda_2(a_{21}x_1+a_{22}x_2-b_2)$$

对 $\boldsymbol{L}(\boldsymbol{x})$ 求偏导，并令偏导数等于 0，有

$$\frac{\partial \boldsymbol{L}(\boldsymbol{x})}{\partial \boldsymbol{x}}=\begin{bmatrix} \dfrac{\partial \boldsymbol{L}(\boldsymbol{x})}{\partial x_1} \\ \dfrac{\partial \boldsymbol{L}(\boldsymbol{x})}{\partial x_2} \end{bmatrix}=\begin{bmatrix} 2x_1+\lambda_1a_{11}+\lambda_2a_{21} \\ 2x_2+\lambda_1a_{12}+\lambda_2a_{22} \end{bmatrix}=2\boldsymbol{x}+\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda}=\boldsymbol{0} \tag{4.25}$$

其中，$\boldsymbol{x}=\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$，$\boldsymbol{A}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$。很明显，此问题为凸优化问题。因此求此问题的

最优值即为求凸函数的驻点，故，

$$\hat{\boldsymbol{x}}_{\text{opt}} = -\frac{1}{2}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} \tag{4.26}$$

将 $\hat{\boldsymbol{x}}_{\text{opt}}$ 代入 $\boldsymbol{Ax}=\boldsymbol{b}$，可得

$$\boldsymbol{A}\hat{\boldsymbol{x}}_{\text{opt}} = -\frac{1}{2}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{b} \Rightarrow \boldsymbol{\lambda} = -2(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b} \tag{4.27}$$

则有

$$\hat{\boldsymbol{x}}_{\text{opt}} = -\frac{1}{2}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b} = \boldsymbol{A}^{+}\boldsymbol{b} \tag{4.28}$$

$\hat{\boldsymbol{x}}_{\text{opt}}$ 为最小二乘解，是二范数约束情况下的最优解。

同理，推广到带映射矩阵的形式，应用最小二乘法求解 l_2 范数的过程如下：

$$\boldsymbol{L}(\boldsymbol{x}) = \|\boldsymbol{Bx}\|_2^2 + \boldsymbol{\lambda}^{\mathrm{T}}(\boldsymbol{Ax}-\boldsymbol{b}) \qquad \boldsymbol{J}(\boldsymbol{x}) = \|\boldsymbol{Bx}\|_2^2 \tag{4.29}$$

假设 $\boldsymbol{x}=\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$，$\boldsymbol{A}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$，$\boldsymbol{b}=\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$，$\boldsymbol{\lambda}=\begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}$，映射矩阵 $\boldsymbol{B}=\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$。

$\boldsymbol{L}(\boldsymbol{x})$ 可以改写为

$$\begin{aligned}
\boldsymbol{L}(\boldsymbol{x}) &= (\boldsymbol{Bx})^{\mathrm{T}}(\boldsymbol{Bx}) + \boldsymbol{\lambda}^{\mathrm{T}}(\boldsymbol{Ax}-\boldsymbol{b}) \\
&= \left(\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)^{\mathrm{T}}\left(\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) + [\lambda_1 \quad \lambda_2]\left(\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right) \\
&= (b_{11}x_1 + b_{12}x_2)^2 + (b_{21}x_1 + b_{22}x_2)^2 + \lambda_1(a_{11}x_1 + a_{12}x_2 - b_1) + \lambda_2(a_{21}x_1 + a_{22}x_2 - b_2)
\end{aligned} \tag{4.30}$$

$\boldsymbol{L}(\boldsymbol{x})$ 对 $\boldsymbol{x}$ 求偏导数并令偏导数等于 0，有

$$\begin{aligned}
\frac{\partial \boldsymbol{L}(\boldsymbol{x})}{\partial \boldsymbol{x}} &= \begin{bmatrix} \dfrac{\partial \boldsymbol{L}(\boldsymbol{x})}{\partial x_1} \\ \dfrac{\partial \boldsymbol{L}(\boldsymbol{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2(b_{11}x_1 + b_{12}x_2)b_{11} + 2(b_{21}x_1 + b_{22}x_2)b_{21} + \lambda_1 a_{11} + \lambda_2 a_{21} \\ 2(b_{11}x_1 + b_{12}x_2)b_{12} + 2(b_{21}x_1 + b_{22}x_2)b_{22} + \lambda_1 a_{12} + \lambda_2 a_{22} \end{bmatrix} \\
&= \begin{bmatrix} 2({b_{11}}^2 + {b_{21}}^2)x_1 + 2(b_{11}b_{12} + b_{21}b_{22})x_2 + \lambda_1 a_{11} + \lambda_2 a_{21} \\ 2({b_{12}}^2 + {b_{22}}^2)x_2 + 2(b_{11}b_{12} + b_{21}b_{22})x_1 + \lambda_1 a_{12} + \lambda_2 a_{22} \end{bmatrix} \\
&= 2\boldsymbol{B}^{\mathrm{T}}\boldsymbol{Bx} + \boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} = \boldsymbol{0}
\end{aligned} \tag{4.31}$$

可得

$$\hat{\boldsymbol{x}} = -\frac{1}{2}(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda} \tag{4.32}$$

将 $\hat{\boldsymbol{x}}$ 代入原方程 $\boldsymbol{Ax}=\boldsymbol{b}$ 中，有

$$-\frac{1}{2}\boldsymbol{A}(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\lambda}=\boldsymbol{b} \tag{4.33}$$

进一步求出 $\boldsymbol{\lambda}$：

$$\boldsymbol{\lambda}=-2(\boldsymbol{A}(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b} \tag{4.34}$$

将 $\boldsymbol{\lambda}$ 代入 $\hat{\boldsymbol{x}}$ 表达式，得到最小二乘解 $\hat{\boldsymbol{x}}_{\mathrm{opt}}$：

$$\hat{\boldsymbol{x}}_{\mathrm{opt}}=(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}(\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b} \tag{4.35}$$

$\hat{\boldsymbol{x}}_{\mathrm{opt}}$ 为最小二乘解，为二范数约束情况下的最优解。

3. 最小二乘法的应用

(1) 最小二乘法在超定方程组问题中的应用。

如果发生 $\boldsymbol{Ax}=\boldsymbol{b}$ 无解的情况，即 $\boldsymbol{e}=\boldsymbol{b}-\boldsymbol{Ax}\neq\boldsymbol{0}$，此时使得 $\boldsymbol{e}$ 最小的解 $\hat{\boldsymbol{x}}$ 为最小二乘解。例如，求最接近点 (0,6)，(1,0) 和 (2,0) 的直线。该问题的求解过程如下。

假设三个点都在所求直线 $y=C+Dx$ 上，可得方程组：

$$\begin{cases}C+D\cdot 0=6\\C+D\cdot 1=0\\C+D\cdot 2=0\end{cases} \tag{4.36}$$

令 $\boldsymbol{A}=\begin{bmatrix}1&0\\1&1\\1&2\end{bmatrix}$，$\boldsymbol{x}=\begin{bmatrix}C\\D\end{bmatrix}$，$\boldsymbol{b}=\begin{bmatrix}6\\0\\0\end{bmatrix}$，式 (4.36) 可以写成如下 $\boldsymbol{Ax}=\boldsymbol{b}$ 形式：

$$\begin{bmatrix}1&0\\1&1\\1&2\end{bmatrix}\begin{bmatrix}C\\D\end{bmatrix}=\begin{bmatrix}6\\0\\0\end{bmatrix}\rightarrow\boldsymbol{Ax}=\boldsymbol{b} \tag{4.37}$$

设该方程组的解为 $\hat{\boldsymbol{x}}=\begin{bmatrix}\hat{C}\\\hat{D}\end{bmatrix}$，则 $\hat{\boldsymbol{x}}$ 满足方程 $\boldsymbol{A}\hat{\boldsymbol{x}}=\boldsymbol{b}$，两边同时乘以 $\boldsymbol{A}^{\mathrm{T}}$ 可得：

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\hat{\boldsymbol{x}}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} \tag{4.38}$$

将 $\boldsymbol{A}$、$\hat{\boldsymbol{x}}$ 和 $\boldsymbol{b}$ 代入上式可得：

$$\begin{bmatrix}1&1&1\\0&1&2\end{bmatrix}\begin{bmatrix}1&0\\1&1\\1&2\end{bmatrix}\begin{bmatrix}\hat{C}\\\hat{D}\end{bmatrix}=\begin{bmatrix}1&1&1\\0&1&2\end{bmatrix}\begin{bmatrix}6\\0\\0\end{bmatrix} \tag{4.39}$$

由式 (4.39) 可解得：

$$\hat{\boldsymbol{x}}=\begin{bmatrix}5\\-3\end{bmatrix} \tag{4.40}$$

故对于(0,6)、(1,0)和(2,0)三个点最接近的直线为 $y=5-3x$，这就是用最小二乘法来解决超定方程组的过程。

(2)最小二乘法在线性回归中的应用。

最小二乘法在回归模型中具有重要的应用，最常用的是普通最小二乘法(ordinary least square，OLS)：所选择的回归模型应该使所有观察值的误差平方和达到最小，即使得平方损失函数达到最小值。

样本线性回归模型为

$$\begin{aligned} \boldsymbol{Y}_i &= \hat{\boldsymbol{\beta}}_0 + \hat{\boldsymbol{\beta}}_1 \boldsymbol{X}_i + \boldsymbol{e}_i \\ \Rightarrow \boldsymbol{e}_i &= \boldsymbol{Y}_i - \hat{\boldsymbol{\beta}}_0 - \hat{\boldsymbol{\beta}}_1 \boldsymbol{X}_i \end{aligned} \tag{4.41}$$

其中，已知样本 $(\boldsymbol{X}_i, \boldsymbol{Y})_i$，$\boldsymbol{e}_i$ 为样本 $(\boldsymbol{X}_i, \boldsymbol{Y}_i)$ 的误差，求 $\boldsymbol{\beta}_0$ 和 $\boldsymbol{\beta}_1$ 的值使拟合效果最佳。

平方损失函数表示为

$$\boldsymbol{Q} = \sum_{i=1}^{n} \boldsymbol{e}_i^2 = \sum_{i=1}^{n} (\boldsymbol{Y}_i - \hat{\boldsymbol{Y}}_i)^2 = \sum_{i=1}^{n} (\boldsymbol{Y}_i - \hat{\boldsymbol{\beta}}_0 - \hat{\boldsymbol{\beta}}_1 \boldsymbol{X}_i)^2 \tag{4.42}$$

则当 $\boldsymbol{Q}$ 取最小值时，确定 $\boldsymbol{\beta}_0$ 和 $\boldsymbol{\beta}_1$ 的值，就变成了一个求极值问题，可以通过求导数的方法得到。对 $\boldsymbol{Q}$ 求 $\boldsymbol{\beta}_0$ 和 $\boldsymbol{\beta}_1$ 的偏导数并令偏导数等于 0 得

$$\begin{cases} \dfrac{\partial \boldsymbol{Q}}{\partial \boldsymbol{\beta}_0} = 2(\boldsymbol{Y}_i - \hat{\boldsymbol{\beta}}_0 - \hat{\boldsymbol{\beta}}_1 \boldsymbol{X}_i)(-1) = 0 \\ \dfrac{\partial \boldsymbol{Q}}{\partial \boldsymbol{\beta}_1} = 2(\boldsymbol{Y}_i - \hat{\boldsymbol{\beta}}_0 - \hat{\boldsymbol{\beta}}_1 \boldsymbol{X}_i)(-\boldsymbol{X}_i) = 0 \end{cases} \tag{4.43}$$

解得

$$\hat{\boldsymbol{\beta}}_0 = \frac{n\sum \boldsymbol{X}_i \boldsymbol{Y}_i - \sum \boldsymbol{X}_i \sum \boldsymbol{Y}_i}{n\sum \boldsymbol{X}_i^2 - \left(\sum \boldsymbol{X}_i\right)^2} \tag{4.44}$$

$$\hat{\boldsymbol{\beta}}_1 = \frac{\sum \boldsymbol{X}_i^2 \sum \boldsymbol{Y}_i - \sum \boldsymbol{X}_i \sum \boldsymbol{X}_i \boldsymbol{Y}_i}{n\sum \boldsymbol{X}_i^2 - \left(\sum \boldsymbol{X}_i\right)^2} \tag{4.45}$$

即求得平方误差函数的极值点。

通常在 MATLAB 软件中对最小二乘法进行仿真，具体操作和结果如下。

polyfit 函数是 MATLAB 中用于进行曲线拟合的一个函数。其数学基础是最小二乘法曲线拟合原理：已知离散点上的数据集，即已知在点集上的函数值，构造一个解析函数，使在原离散点上尽可能接近给定的值。一次函数线性拟合使用 $\text{polyfit}(x,y,1)$；多次函数线性拟合使用 $\text{polyfit}(x,y,n)$，n 为次数。

例如，当 $x=[1.00,2.00,3.00,4.00,5.00,6.00]$，$y=[3.50,5.55,6.00,6.65,\ 8.50,10.00]$，在 MATLAB 中执行如下代码：

```
x=[1.00 2.00 3.00 4.00 5.00 6.00];
y=[3.50 5.55 6.00 6.65 8.50 10.00];
r=corrcoef(x,y) ;
a=polyfit(x,y,1);
x1=0:0.1:8;
P=polyval(a,x1);
figure(1);hold on;plot(x,y,'r*',x1,P,'b');
```

得到的一次线性拟合曲线如图 4.1 所示。

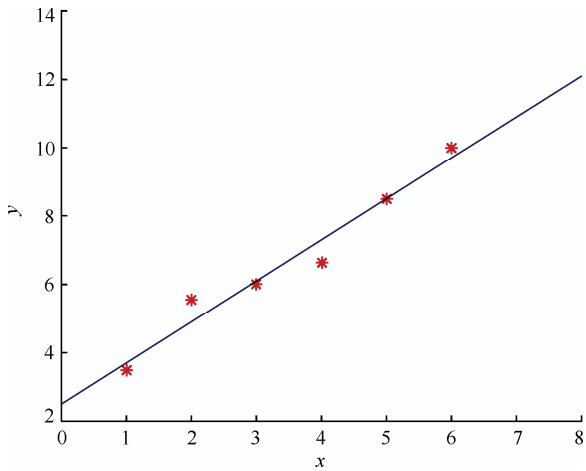

图 4.1　最小二乘法 MATLAB 实验拟合曲线

图 4.1 中，“*”为原始值 (x,y)；实线为最小二乘法拟合曲线。从图是可以看出，对于一元线性回归模型，最小二乘法能够得到效果较好的拟合曲线。

4.2.2　l_0 范数系数的求解方法

由第 2 章已知，l_0 范数问题是一个 NP 难问题，故而 l_0 范数系数的求解是一个开放问题，其求解方法不是唯一的。本节主要介绍几种 l_0 范数系数求解方法。

图像的稀疏表示：将原始图像分块，每个小块大小为 $n\times n$。将每一个小块作为一个样本 $\boldsymbol{b}$，在字典 $\boldsymbol{A}$ 上对样本 $\boldsymbol{b}$ 进行稀疏分解，获得样本 $\boldsymbol{b}$ 的稀疏向量 $\boldsymbol{x}$，$\boldsymbol{x}$ 为样本 $\boldsymbol{b}$ 在字典 $\boldsymbol{A}$ 上的分解系数，或者称为稀疏系数。按同样的方法对所有小块进行稀疏分解后，得到原始图像的稀疏矩阵。将字典 $\boldsymbol{A}$ 的数据线性组合可以重建出样本 $\boldsymbol{b}$：

$$\boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.46}$$

字典 $\boldsymbol{A}$ 中的每一个向量称之为一个原子。如果在某个过完备字典上，将某一样本进行稀疏分解后获得的稀疏矩阵中含有许多零元素，则样本 $\boldsymbol{b}$ 可以被稀疏表示，或者可以说样本 $\boldsymbol{b}$ 具有稀疏性。通常情况下，稀疏度函数用 l_0 范数表示，图像的稀疏表示模型为

$$\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_0, \text{ s.t. } \boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.47}$$

1. 匹配追踪算法

稀疏表示的目标是：满足既定稀疏条件情况下，对目标函数进行优化，得到图像稀疏表示系数。经典稀疏表示算法有匹配追踪(MP)算法、正交匹配追踪[37](OMP)算法和基追踪(basis pursuit，BP)算法等。

匹配追踪算法的数学模型：

$$\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_0, \text{ s.t. } \boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.48}$$

其中，$\boldsymbol{x}$ 为需要求解的稀疏表示系数，$\boldsymbol{A}$ 为已知的字典，$\boldsymbol{b}$ 为输入信号。

匹配追踪算法是一种贪婪算法，其主要思想是每一次迭代寻找一个与当前稀疏表示残差具有最小角度的原子参与表示。贪婪算法在求解问题时，总是选择出当前迭代或当前步骤最好的结果。或者说，在考虑最优解时，并不是从整体出发寻找最优解，而是思考眼前利益，寻找的是当前的局部最优解。贪婪算法的主要思想是基于某个问题的初始解，在此解的基础上一步步寻找或计算。根据已知的优化标准，保证取得每一步或每次迭代的最优解。

在这种情况下，每一步都达到最优解，但合起来最终并不一定能达到全局最优解，这也是匹配追踪算法的缺点所在。

如图 4.2 所示，p 为二维空间内一点，我们的目标是从原点出发，寻找一条到达目标点 p 的最优路线。贪婪算法不能同时考虑两个坐标轴，而是只选择一条最好的坐标轴，然后沿着该坐标轴进行优化，优化完成后再考虑另外一条坐标轴。

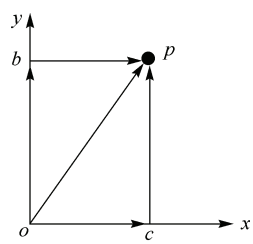

图 4.2　二维状态下的局部最优解与全局最优解

贪婪算法的第一步是，在两条坐标轴 x 和 y 上寻找一条最优路线，已知坐标轴上有两条路线 $\vec{ob}$ 和 $\vec{oc}$，显然，较之 ob，oc 间距离更短，故而选择 $\vec{oc}$ 作为第一步

的最优解；第二步以点 c 为原点，继续在两个坐标轴上选择最佳路线，得到 $\vec{cp}$ 为最优解，故而贪婪算法得到的最优解为 $\vec{oc}+\vec{cp}$。但实际上，在图 4.2 中可以更直观地看出，全局最优解为原点 o 到点 p 间的射线 $\vec{op}$。贪婪算法寻找的路线明显是一条折线而不是最优解 $\vec{op}$，只观察局部信息，不考虑全局，这是贪婪算法最大的缺点。但是，虽然贪婪算法并不一定能得到一个全局最优解，但可以得到一个次优解，这个次优解也是一个在工程中可以接受的解。若想得到一个全局最优解，往往需要很大的计算成本。但在实际情况中，计算成本又是一个不得不考虑的因素，所以为了降低成本，在大多数情况下会用次优解近似代替全局最优解来简化计算。

MP 法作为贪婪算法的一种，迭代过程也是遵循局部最优解的原则进行的。根据图 4.3 所示的运算过程图，我们可以直观了解 MP 算法的实质：利用原子向量的线性运算去逐渐逼近信号向量，经过不断的迭代，最终达到最优解。

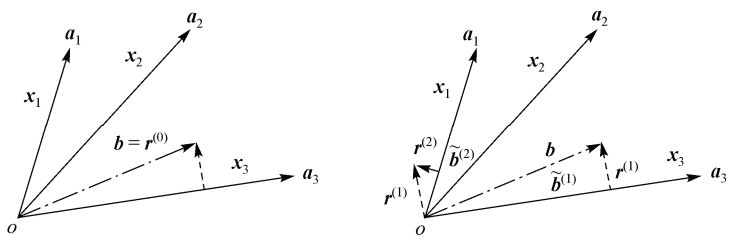

图 4.3　MP 算法运算过程简单实例

MP 算法的基本流程为：初始化稀疏表示残差向量，然后在所有原子中找一个与残差最为相似的原子，将残差在该原子上的投影作为逼近量，接下来用残差减去逼近量，对残差进行更新。再对更新后的残差寻找最近原子，继续迭代上述步骤，直到达到最优解。

如图 4.3 所示，假设有三个原子 $\boldsymbol{a}_1,\boldsymbol{a}_2,\boldsymbol{a}_3$，即矩阵 $\boldsymbol{A}=[\boldsymbol{a}_1,\boldsymbol{a}_2,\boldsymbol{a}_3]$，$\boldsymbol{b}$ 为数据向量。模型的优化目标是用矩阵 $\boldsymbol{A}$ 中的原子去逐渐逼近 $\boldsymbol{b}$，也就是说我们期望 $x_1\boldsymbol{a}_1+x_2\boldsymbol{a}_2+x_3\boldsymbol{a}_3=\boldsymbol{b}$。其中，$x_1,x_2,x_3$ 为表示系数。匹配追踪算法实际上就是利用 x_1,x_2,x_3 对 $\boldsymbol{a}_1,\boldsymbol{a}_2,\boldsymbol{a}_3$ 进行线性组合得到数据向量 $\boldsymbol{b}$，具体流程如下。

第一步，如图 4.3(a)所示，从数据向量 $\boldsymbol{b}$ 出发，令初始残差向量 $\boldsymbol{r}^{(0)}=\boldsymbol{b}$，在 $\boldsymbol{a}_1,\boldsymbol{a}_2,\boldsymbol{a}_3$ 中找到与向量 $\boldsymbol{b}$ 最相似的向量。根据相似度准则，相似向量寻找方法有两个：欧氏距离和内积的方法。内积的方法就是求两向量之间的夹角，夹角越小相似度越高，即

$$\cos\theta=\frac{\boldsymbol{a}\cdot\boldsymbol{b}}{|\boldsymbol{a}|.|\boldsymbol{b}|} \tag{4.49}$$

其中，θ 表示两向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的夹角，$\boldsymbol{a}\cdot\boldsymbol{b}$ 表示两个向量的内积，$|\boldsymbol{a}|$ 和 $|\boldsymbol{b}|$ 分别表示两个向量的模值，即两个向量的长度。图 4.3(a)中，很明显，$\boldsymbol{a}_3$ 与 $\boldsymbol{b}$ 的夹角最小，说

明在 $\boldsymbol{a}_1,\boldsymbol{a}_2,\boldsymbol{a}_3$ 中， $\boldsymbol{a}_3$ 与残差向量 $\boldsymbol{b}$ 最相似。

第二步，如图 4.3 (b) 所示，用 $\boldsymbol{a}_3$ 逼近 $\boldsymbol{b}$，计算新的残差。具体做法是将 $\boldsymbol{b}$ 在 $\boldsymbol{a}_3$ 上做投影，该投影为原子 $\boldsymbol{a}_3$ 对于残差 $\boldsymbol{b}$ 的逼近量，令其为 $\tilde{\boldsymbol{b}}^{(1)}$，假设投影长度为 x_3，则 $\tilde{\boldsymbol{b}}^{(1)}=x_3\boldsymbol{a}_3$，然后用第一次的残差 $\boldsymbol{b}$ 减去逼近量 $\tilde{\boldsymbol{b}}^{(1)}$ 得到第一次更新的残差，即 $\boldsymbol{r}^{(1)}=\boldsymbol{b}-\tilde{\boldsymbol{b}}^{(1)}$。

第三步，如图 4.3 (b) 所示，将更新后的残差 $\boldsymbol{r}^{(1)}$ 平移到原点 o 处，在其余原子中重新寻找最相似原子，然后残差向最相似原子投影，得到逼近量，用该残差减去逼近量得到下一次的残差。具体做法是：平移后的残差 $\boldsymbol{r}^{(1)}$ 与剩余原子中的 $\boldsymbol{a}_1$ 最相似，将 $\boldsymbol{r}^{(1)}$ 在 $\boldsymbol{a}_1$ 上做投影，该投影为原子 $\boldsymbol{a}_1$ 对于残差 $\boldsymbol{r}^{(1)}$ 的逼近量，令其为 $\tilde{\boldsymbol{b}}^{(2)}$，假设投影长度为 x_1，则 $\tilde{\boldsymbol{b}}^{(2)}=x_1\boldsymbol{a}_1$，然后用残差 $\boldsymbol{r}^{(1)}$ 减去逼近量 $\tilde{\boldsymbol{b}}^{(2)}$ 得到第二次更新的残差，即 $\boldsymbol{r}^{(2)}=\boldsymbol{r}^{(2)}-\tilde{\boldsymbol{b}}^{(2)}=\boldsymbol{b}-\tilde{\boldsymbol{b}}^{(1)}-\tilde{\boldsymbol{b}}^{(2)}$。

按照上述步骤依次迭代进行，直到达到最优解为止。

根据图 4.3 可知，寻找初始残差 $\boldsymbol{b}$ 的相似原子 $\boldsymbol{a}_3$ 后，进行第一次投影得到 x_3，此时得到的残差仍然较大，即 $\tilde{\boldsymbol{b}}^{(1)}=x_3\boldsymbol{a}_3\neq\boldsymbol{b}$，不满足约束条件。寻找下一个相似原子，将迭代后的残差平移到原点 o 位置，找到与其最相似的原子 $\boldsymbol{a}_1$，残差在 $\boldsymbol{a}_1$ 上投影得到 x_1，此时 $\tilde{\boldsymbol{b}}^{(2)}=x_1\boldsymbol{a}_1$，期望目标相当于 $\tilde{\boldsymbol{b}}^{(1)}+\tilde{\boldsymbol{b}}^{(2)}=x_3\boldsymbol{a}_3+x_1\boldsymbol{a}_1\approx\boldsymbol{b}$。此时仍有残差，但加入 $\boldsymbol{a}_1$ 后残差明显比单纯用 $\boldsymbol{a}_3$ 做投影的残差要小，即 $x_3\boldsymbol{a}_3+x_1\boldsymbol{a}_1$ 比 $x_3\boldsymbol{a}_3$ 更接近于 $\boldsymbol{b}$。接下来如果满足停止条件，可以停止迭代。如果不满足，按照上述步骤进行后续迭代，直至迭代后的残差足够小达到最优解为止。

匹配追踪算法是一种收敛的算法，随着原子的不断加入，残差不断变小，越接近于目标解。在匹配追踪算法中，每一次只能寻找一个最相似原子进行逼近，然而，上述问题中的最优解应该在整体考虑 $\boldsymbol{a}_1,\boldsymbol{a}_2,\boldsymbol{a}_3$ 的情况下，调整 x_1,x_2,x_3 的值，使 $x_1\boldsymbol{a}_1+x_2\boldsymbol{a}_2+x_3\boldsymbol{a}_3$ 尽可能接近于 $\boldsymbol{b}$。因此，匹配追踪算法作为一种贪婪算法，其得到的解并不一定是全局最优解。

匹配追踪 (MP) 算法具体流程如下。

输入： $n\times m$ 维的字典 $\boldsymbol{A}$， n 维数据向量 $\boldsymbol{b}$。

输出：稀疏表示系数 $\boldsymbol{x}$。

初始化：稀疏表示残差 $\boldsymbol{r}^{(0)}=\boldsymbol{b}$， $\boldsymbol{x}^{(0)}=[\ ]$。

执行如下步骤直到满足停止条件，迭代次数 $J=1,2,3,\cdots$。

①选择原子号码 Λ_J，选择的标准是与上一次迭代的残差内积绝对值达到最大：

$$\Lambda^{(J)}\in\arg\max_{\varpi\in\Omega}\left|\left\langle\boldsymbol{r}^{(J-1)},\boldsymbol{A}(\varpi)\right\rangle\right|$$

其中， Ω 是 $\boldsymbol{A}$ 中全体原子号码。

②计算新的向量 $\boldsymbol{b}$ 的逼近和新的残差：

稀疏表示系数：$x^{(J)}(\Lambda^{(J)})=\left\langle \boldsymbol{r}^{(J-1)},\boldsymbol{A}(\Lambda^{(J)})\right\rangle$

向量 $\boldsymbol{b}$ 的逼近：$\boldsymbol{b}^{(J)}=\tilde{\boldsymbol{b}}^{(J-1)}+\left\langle \boldsymbol{r}^{(J-1)},\boldsymbol{A}(\Lambda^{(J)})\right\rangle \boldsymbol{A}(\Lambda^{(J)})$

残差：$\boldsymbol{r}^{(J)}=\boldsymbol{r}^{(J-1)}-\left\langle \boldsymbol{r}^{(J-1)},\boldsymbol{A}(\Lambda^{(J)})\right\rangle \boldsymbol{A}(\Lambda^{(J)})$

匹配追踪算法中，当前选择的原子 $\boldsymbol{A}(\Lambda^{(J)})$ 与当前稀疏表示残差 $\boldsymbol{r}^{(J)}$ 正交。

2. 正交匹配追踪算法

正交匹配追踪法，顾名思义，就是正交的匹配追踪算法。相比于 MP 算法，OMP 算法的改进之处在于：在迭代的每一步都对选择的全部原子进行正交化处理。这就使得在相同精度要求下，OMP 算法的收敛速度比 MP 算法的收敛速度更快。

正交匹配追踪算法的数学模型与匹配追踪算法相同。

$$\min_{\boldsymbol{x}}\|\boldsymbol{x}\|_0,\ \text{s.t.}\ \boldsymbol{b}=\boldsymbol{A}\boldsymbol{x} \tag{4.50}$$

其中，$\boldsymbol{x}$ 为稀疏表示系数，$\boldsymbol{A}$ 为字典，$\boldsymbol{b}$ 为输入信号。

由已经描述过的 MP 算法可知，如果信号(残差)在已选择的原子上进行投影是非正交的，那么就会导致每次的迭代结果并不是最优解而是次优解，需要多次迭代才能达到收敛的效果。例如，在二维空间中，用矩阵 $\boldsymbol{A}=[\boldsymbol{a}_1,\boldsymbol{a}_2]$ 中的向量去逼近残差 $\boldsymbol{b}$，经过 MP 算法迭代会发现，该算法总是在 $\boldsymbol{a}_1$ 和 $\boldsymbol{a}_2$ 上反复迭代，即 $\boldsymbol{b}=\boldsymbol{x}_1\boldsymbol{a}_1+\boldsymbol{x}_2\boldsymbol{a}_2+\boldsymbol{x}_3\boldsymbol{a}_1+\boldsymbol{x}_4\boldsymbol{a}_2+\cdots$，这就是残差在已选择的原子上进行垂直投影的非正交性导致的。推广到高维空间中，$\boldsymbol{A}=\{\boldsymbol{a}_1,\ \boldsymbol{a}_2,\cdots,\ \boldsymbol{a}_n\}$，定义 $V_K=\text{span}\{\boldsymbol{a}_{r^{(0)}},\boldsymbol{a}_{r^{(1)}},\cdots,\boldsymbol{a}_{r^{(k-1)}}\}$，$V_K$ 为原子 $\boldsymbol{a}_{r^{(0)}},\boldsymbol{a}_{r^{(1)}},\cdots,\boldsymbol{a}_{r^{(k-1)}}$ 张成的空间。构造 MP 算法的一种表达形式为：$P_V\boldsymbol{r}=\sum_n \boldsymbol{a}_n x_n$。此处 $P_V\boldsymbol{r}$ 表示残差 $\boldsymbol{r}$ 在空间 V 上的一个正交投影操作。匹配追踪算法的第 k 次迭代的结果可以表示为：$\boldsymbol{b}=\sum_{i=1}^{k}\boldsymbol{a}_i x_{n_i}+\boldsymbol{r}^{(k)}=\boldsymbol{b}^{(k)}+\boldsymbol{r}^{(k)}$，其中，$\boldsymbol{b}^{(k)}=\tilde{\boldsymbol{b}}^{(1)}+\tilde{\boldsymbol{b}}^{(2)}+\cdots+\tilde{\boldsymbol{b}}^{(k)}=\sum_{i=1}^{k}\tilde{\boldsymbol{b}}^{(k)}$ 为第 k 次的垂直投影分量，当且仅当 $\boldsymbol{r}^{(k)}$ 垂直于张成的平面 V_K 时，$\boldsymbol{b}^{(k)}$ 为最优的 k 次迭代近似值。由于 MP 算法仅能保证 $\boldsymbol{r}^{(k)}$ 垂直于 $\boldsymbol{a}_k$，所以 $\boldsymbol{b}^{(k)}$ 一般情况下为次优解。$\boldsymbol{b}^{(k)}$ 为 k 个项的线性表示，这个组合的值作为最优近似值，只有在第 k 个残差与 $\boldsymbol{b}^{(k)}$ 垂直时，才能保证 $\boldsymbol{b}^{(k)}$ 为最优解。当第 k 个残差与 $\boldsymbol{b}^{(k)}$ 正交时，此残差 $\boldsymbol{r}^{(k)}$ 与 $\boldsymbol{b}^{(k)}$ 的任何一项都线性无关，那么，第 k 个残差在后面的分解过程中，不会再出现 $\boldsymbol{b}^{(k)}$ 中已经出现的项，为最优解。但通常情况下，残差 $\boldsymbol{r}^{(k)}$ 往往不能满足这个条件，MP 算法仅能保证第 k 个残差和 $\boldsymbol{a}_k$ 正交。因此，OMP 算法在稀疏表示系数的计算环节上，对 MP 算法进行了改进，保证了第 k 次更新的残差 $\boldsymbol{r}^{(k)}$ 和 $\boldsymbol{b}^{(k)}$ 正交。

算法运算过程如下。

如图 4.4 所示，OMP 算法的具体步骤如下。

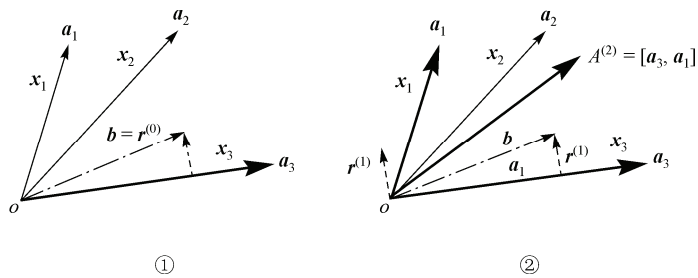

图 4.4　OMP 算法运算过程简单实例

第一步：初始化。首先让稀疏表示残差 $\boldsymbol{r}^{(0)}=\boldsymbol{b}$，然后让参与稀疏表示的原子号码为空集，即此时还未从字典 $\boldsymbol{A}$ 中选出任何原子，接下来令参与稀疏表示的原子组成的矩阵 $\boldsymbol{A}^{(J)}=[\ ]$ 为空集。

第二步：选原子。在字典 $\boldsymbol{A}$ 中选择与残差向量 $\boldsymbol{r}^{(0)}=\boldsymbol{b}$ 最相似的原子 $\boldsymbol{a}_3$，接下来将残差在该原子上做投影，按照匹配追踪算法的迭代方法得到新的残差向量 $\boldsymbol{r}^{(1)}$。

第三步：把残差 $\boldsymbol{r}^{(1)}$ 平移到原点处，比较原子 $\boldsymbol{a}_1,\boldsymbol{a}_2$ 与残差 $\boldsymbol{r}^{(1)}$ 的内积，内积越大相似度越高，选出与残差 $\boldsymbol{r}^{(1)}$ 最相似的原子 $\boldsymbol{a}_1$。接下来就是正交匹配追踪算法对于匹配追踪算法的改进部分：MP 算法中，残差 $\boldsymbol{r}^{(1)}$ 在原子 $\boldsymbol{a}_1$ 上做投影，可得到新的残差向量 $\boldsymbol{r}^{(2)}$，按照此方法进行逐步迭代；OMP 算法中将原子 $\boldsymbol{a}_3$ 和 $\boldsymbol{a}_1$ 作为入选原子，形成一个新的矩阵 $\boldsymbol{A}^{(2)}=[\boldsymbol{a}_3,\boldsymbol{a}_1]$，对残差与矩阵 $\boldsymbol{A}^{(2)}$ 求最小二乘。MP 算法是将残差直接在原子上做投影，得到一个表示系数，并迭代得到新的残差，而 OMP 算法则是第二步入选的原子 $\boldsymbol{a}_1$ 与已经入选的原子 $\boldsymbol{a}_3$ 组合到一起，形成新的矩阵 $\boldsymbol{A}^{(2)}$，求残差与 $\boldsymbol{A}^{(2)}$ 的最小二乘，得到 $\boldsymbol{x}^{(2)}=(\boldsymbol{A}^{(2)\mathrm{T}}\boldsymbol{A}^{(2)})^{-1}\boldsymbol{A}^{(2)\mathrm{T}}\boldsymbol{b}$。在这种情况下，每一次迭代的残差都与前面选定的所有原子正交。

第四步：计算稀疏表示残差 $\boldsymbol{r}^{(2)}=\boldsymbol{b}-\boldsymbol{A}^{(2)}\boldsymbol{x}^{(2)}$。将 $\boldsymbol{r}^{(2)}$ 作为新的残差，再利用如第二步所述的方法进行原子选择，继续迭代直到满足停止条件。

从上面的步骤可以看出，OMP 算法与 MP 算法之间的不同之处主要有两点。

(1) OMP 算法的稀疏表示系数更新方式为 $\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$，其中，$\boldsymbol{A}^{(J)}$ 为第 J 次迭代后入选的所有原子构成的矩阵。稀疏表示每次都重新计算每一个入选原子所乘的稀疏表示系数；MP 算法的稀疏表示系数更新方式为仅仅更新当前迭代新入选的那一个原子的稀疏表示系数，而前面迭代入选原子的稀疏表示系数在后面迭代过程中保持不变。

(2) OMP 算法的残差计算方式为 $\boldsymbol{r}^{(J)}=\boldsymbol{b}-\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}$，该方式为输入信号与当前稀疏表示逼近向量之间的残差；而 MP 算法的残差计算方式为 $\boldsymbol{r}^{(J)}=\boldsymbol{r}^{(J-1)}-\left\langle\boldsymbol{r}^{(J-1)},\boldsymbol{A}(\Lambda^{(J)})\right\rangle\boldsymbol{A}(\Lambda^{(J)})$，该方式为前一次迭代的残差与前一次迭代残差在当前入选原子上的投影之间的差值。

从这两个环节的不同之处可以看出，MP 算法的步骤更为片面，仅仅根据当前迭代生成的一些值确定稀疏表示系数，而 OMP 算法的步骤能够利用到前面迭代步骤中生成的参数整体更新稀疏表示系数。

在 OMP 算法中，残差与原子正交性数学推导公式如下。

由于 $\boldsymbol{r}^{(J)}=\boldsymbol{b}-\boldsymbol{b}^{(J)}$，$\boldsymbol{b}^{(J)}=\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}$，故有，

$$\begin{aligned}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{r}^{(J)}&=\boldsymbol{A}^{(J)\mathrm{T}}(\boldsymbol{b}-\boldsymbol{b}^{(J)})\\&=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}-\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}\\&=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}-(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}\\&=\boldsymbol{0}\end{aligned}\tag{4.51}$$

其中，J 为迭代次数，$\boldsymbol{b}^{(J)}$ 为第 J 次迭代的投影向量，$\boldsymbol{x}^{(J)}$ 为第 J 次迭代的稀疏表示系数。$\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{r}^{(J)}=\boldsymbol{0}$，说明残差 $\boldsymbol{r}^{(J)}$ 是与矩阵 $\boldsymbol{A}^{(J)}$ 中原子组成的空间垂直的向量，该残差与整个空间是正交的，即第 J 次迭代的残差与之前选定的所有原子皆正交。

OMP 算法具体流程如下。

输入：$n\times m$ 维的字典 $\boldsymbol{A}$，n 维数据向量 $\boldsymbol{b}$。

输出：稀疏表示系数 $\boldsymbol{x}$。

初始化：稀疏表示残差 $\boldsymbol{r}^{(0)}=\boldsymbol{b}$，参与稀疏表示的原子号码集合 $\boldsymbol{\Phi}$ 为空集，参与稀疏表示的原子组成的矩阵 $\boldsymbol{A}^{(J)}=[\ ]$ 为空集。

执行如下步骤直到满足停止条件，迭代次数 $J=1,2,3,\cdots$。

(1) 选择原子号码 $\Lambda^{(J)}$，选择的标准是与上一次迭代的残差内积达到最大，$\Lambda^{(J)}\in\arg\max\limits_{\varpi\in\Omega}\left|\left\langle\boldsymbol{r}^{(J-1)},\boldsymbol{A}(\varpi)\right\rangle\right|$，其中，$\Omega$ 是 $\boldsymbol{A}$ 中的全体原子号码。$\boldsymbol{\Phi}^{(J)}=[\Lambda^{(1)}\quad\Lambda^{(2)}\quad\cdots\quad\Lambda^{(J)}]$，$\boldsymbol{A}^{(J)}=[\boldsymbol{A}(\Lambda^{(1)})\quad\boldsymbol{A}(\Lambda^{(2)})\quad\cdots\quad\boldsymbol{A}(\Lambda^{(J)})]$。

(2) 计算新的向量 $\boldsymbol{b}$ 的逼近和新的残差：

稀疏表示系数选择原则：$\boldsymbol{x}^{(J)}=\arg\min\limits_{\boldsymbol{\theta}}\left\|\boldsymbol{b}-\boldsymbol{A}^{(J)}\boldsymbol{\theta}\right\|_2$；

稀疏表示系数计算方法：$\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$；

向量 $\boldsymbol{b}$ 的逼近：$\boldsymbol{b}^{(J)}=\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}$；

残差：$\boldsymbol{r}^{(J)}=\boldsymbol{b}-\boldsymbol{b}^{(J)}$。

虽然 OMP 算法相比于 MP 算法有所改进，但仍存在一些问题。在运用公式 $\Lambda^{(J)}\in\arg\max\limits_{\varpi\in\Omega}\left|\left\langle\boldsymbol{r}^{(J-1)},\boldsymbol{A}(\varpi)\right\rangle\right|$ 求最相似原子时，残差 $\boldsymbol{r}^{(J-1)}$ 需要与 $\boldsymbol{A}$ 中所有原子求内积，算法复杂度相对较高。特别是当 $\boldsymbol{A}$ 的维数较高时，计算极为复杂。另外一个就是在求稀疏表示系数 $\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$ 时，当 $\boldsymbol{A}$ 的维数较高时求解 $(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}$，先对矩阵做乘法运算，乘法过程本身具有一定的算法复杂度，接下来再对其求逆矩

阵，求逆过程本身也是具有一定算法复杂度的运算过程，所以求解稀疏表示系数的计算过程的运算复杂度很大。

综上所述，OMP 算法存在的最大问题是存在两个复杂度较高的环节。

(1) 选择原子的过程计算 $\Lambda^{(J)} \in \arg\max_{\varpi \in \Omega} \left| \left\langle \boldsymbol{r}^{(J-1)}, \boldsymbol{A}(\varpi) \right\rangle \right|$，该过程要用每一次迭代所获得的残差与字典中的原子求内积，寻找最相似的原子。

(2) 用最小二乘法求解稀疏表示系数 $\boldsymbol{x}^{(J)} = (\boldsymbol{A}^{(J)\mathrm{T}} \boldsymbol{A}^{(J)})^{-1} \boldsymbol{A}^{(J)\mathrm{T}} \boldsymbol{b}$ 的求逆过程和矩阵的乘法都是复杂度较高的过程。

算法复杂度的高低决定该算法的应用范围，算法复杂度越低，算法应用的适用范围越广。虽然 OMP 算法能得到较好的稀疏表示效果，但会带来较高的算法复杂度，这也导致运算速度问题成为 OMP 算法应用的瓶颈。

下面通过对 MP 算法和 OMP 算法设计简单实验来展示这两种算法的性能。

创建一个大小为 30×50 的随机矩阵 $\boldsymbol{A}$，其原子服从正态分布。对该矩阵的列进行归一化处理，使其 l_2 范数为 1。生成的稀疏向量 $\boldsymbol{x}$，随机选择从 0 到 10 均匀分布的原子个数。每个原子数执行 1000 次测试，并给出平均结果。

在测试一种近似算法的成功性时，有很多方法可以定义所得解 $\hat{\boldsymbol{x}}$ 与理想解 $\boldsymbol{x}$ 之间的距离。这里定义一种 l_2 范数形式的错误率：$l_{2\text{-error}} = \dfrac{\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2}{\|\boldsymbol{x}\|^2}$，MP 算法及 OMP 算法稀疏表示实验结果对比图如图 4.5 所示。

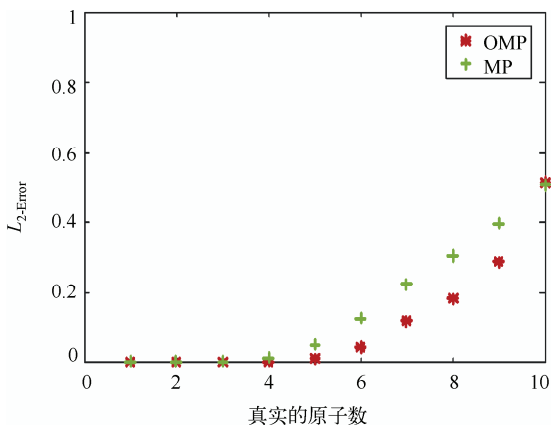

图 4.5　MP 算法与 OMP 算法实验结果对比图

由误差分析及图 4.5 实验结果可知，随着原子数的增加，在相同原子数的情况下，与 MP 算法相比，OMP 算法求解稀疏表示后所得的误差更小，说明在同等条件下，OMP 算法的稀疏表示更为精确，表明了 OMP 算法的改进和优势所在。

3. 最小二乘正交匹配追踪算法

在 OMP 算法中，复杂度问题是其应用的瓶颈，针对复杂度的问题，OMP 算法有两个比较难的运算环节，第一个是寻找最相似原子的过程，第二个是运用最小二乘法寻找稀疏表示系数的过程。对于第二个问题，比较复杂的环节就是求逆问题，本节所讲最小二乘正交匹配追踪(least squares orthogonal matching pursuit，LS-OMP)算法就是为了解决运用最小二乘法求解稀疏表示过程中的求逆问题而提出的。

LS-OMP 算法的主要思想是根据前一步迭代的逆矩阵和当前迭代的逆矩阵之间的相关性，得到一个算法复杂度比较低的求逆运算方法：通过前一次迭代的逆矩阵，直接求解当前的逆矩阵，大大简化了算法的复杂度。

LS-OMP 算法是基于以下分析提出的：稀疏表示系数计算式 $\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$ 中，$(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}$ 和 $(\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)})^{-1}$ 之间的关系为：矩阵 $\boldsymbol{A}^{(J)\mathrm{T}}$ 比矩阵 $\boldsymbol{A}^{(J-1)\mathrm{T}}$ 多一行数据，也就是说，矩阵 $\boldsymbol{A}^{(J)\mathrm{T}}$ 是在矩阵 $\boldsymbol{A}^{(J-1)\mathrm{T}}$ 的基础上添加了一个行向量。同理，矩阵 $\boldsymbol{A}^{(J)}$ 在迭代过程中，在矩阵 $\boldsymbol{A}^{(J-1)}$ 的基础上，下一次迭代时又进入一个原子，相当于在 $\boldsymbol{A}^{(J-1)}$ 上面加了一列。基于这个原理，提出以下思考：如果已知 $(\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)})^{-1}$，能否由 $(\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)})^{-1}$ 推导出一个简单的算法，通过该算法将 $(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}$ 求出？LS-OMP 算法就是基于这种思想。这种方法比直接求 $(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}$ 算法复杂度要低得多。

其中，上述过程中的最小二乘步骤可以用下式计算：

$$\begin{bmatrix}\boldsymbol{M} & \boldsymbol{b}\\ \boldsymbol{b}^{\mathrm{T}} & \mathrm{c}\end{bmatrix}^{-1}=\begin{bmatrix}\boldsymbol{M}^{-1}+\rho\boldsymbol{M}^{-1}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{M}^{-1} & -\rho\boldsymbol{M}^{-1}\boldsymbol{b}\\ -\rho\boldsymbol{b}^{\mathrm{T}}\boldsymbol{M}^{-1} & \rho\end{bmatrix} \tag{4.52}$$

其中，$\rho=1/(c-\boldsymbol{b}^{\mathrm{T}}\boldsymbol{M}^{-1}\boldsymbol{b})$。$\begin{bmatrix}\boldsymbol{M} & \boldsymbol{b}\\ \boldsymbol{b}^{\mathrm{T}} & \mathrm{c}\end{bmatrix}$ 中 $\boldsymbol{M}$ 为 $\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)}$，$\begin{bmatrix}\boldsymbol{M} & \boldsymbol{b}\\ \boldsymbol{b}^{\mathrm{T}} & \mathrm{c}\end{bmatrix}$ 为在 $\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)}$ 中加了一行和一列。相当于，对于 $\boldsymbol{A}^{(J)}$ 和 $\boldsymbol{A}^{(J-1)}$，已知 $\boldsymbol{A}^{(J-1)}$，若在此迭代步骤中，入选原子为 $\boldsymbol{a}$，则 $\boldsymbol{A}^{(J)}=[\boldsymbol{A}^{(J-1)},\boldsymbol{a}]$，$\boldsymbol{A}^{(J)}$ 即为在矩阵 $\boldsymbol{A}^{(J-1)}$ 的基础上加一列。同样地，$\boldsymbol{A}^{(J)\mathrm{T}}=\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\\ \boldsymbol{a}^{\mathrm{T}}\end{bmatrix}$。此时，将 $\boldsymbol{A}^{(J)}$ 和 $\boldsymbol{A}^{(J)\mathrm{T}}$ 相乘，$\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}=\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\\ \boldsymbol{a}^{\mathrm{T}}\end{bmatrix}\cdot[\boldsymbol{A}^{(J-1)},\boldsymbol{a}]$。首先，将 $\boldsymbol{A}^{(J-1)\mathrm{T}}$ 和 $\boldsymbol{A}^{(J-1)}$ 相乘得到 $\begin{bmatrix}\boldsymbol{M} & \boldsymbol{b}\\ \boldsymbol{b}^{\mathrm{T}} & \mathrm{c}\end{bmatrix}$ 的左上角元素 $\boldsymbol{M}$，$\boldsymbol{a}^{\mathrm{T}}$ 和 $\boldsymbol{A}^{(J-1)}$ 相乘得到左下角元素 $\boldsymbol{b}^{\mathrm{T}}$，$\boldsymbol{A}^{(J-1)\mathrm{T}}$ 和 $\boldsymbol{a}$ 相乘得到右上角元素 $\boldsymbol{b}$，最后将 $\boldsymbol{a}^{\mathrm{T}}$ 与 $\boldsymbol{a}$ 相乘得到右下角元素 c。即为

$$\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}=\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\\ \boldsymbol{a}^{\mathrm{T}}\end{bmatrix}\cdot[\boldsymbol{A}^{(J-1)},\boldsymbol{a}]=\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{a}\boldsymbol{A}^{(J-1)\mathrm{T}}\\ \boldsymbol{a}^{\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{a}^{\mathrm{T}}\boldsymbol{a}\end{bmatrix}=\begin{bmatrix}\boldsymbol{M} & \boldsymbol{b}\\ \boldsymbol{b}^{\mathrm{T}} & \mathrm{c}\end{bmatrix} \tag{4.53}$$

上式中，所有原子为归一化原子，因此，有 $\boldsymbol{a}^{\mathrm{T}}\boldsymbol{a}=1$。在得到 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$ 之后，接下来就要对其进行求逆操作。相当于，将 $\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)}$ 看作 $\boldsymbol{M}$，将 $\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{a}\boldsymbol{A}^{(J-1)\mathrm{T}}\\ \boldsymbol{a}^{\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{a}^{\mathrm{T}}\boldsymbol{a}\end{bmatrix}$ 用 $\begin{bmatrix}\boldsymbol{M} & \boldsymbol{b}\\ \boldsymbol{b}^{\mathrm{T}} & \mathrm{c}\end{bmatrix}$ 代替之后，可通过公式 $\begin{bmatrix}\boldsymbol{M} & \boldsymbol{b}\\ \boldsymbol{b}^{\mathrm{T}} & \mathrm{c}\end{bmatrix}^{-1}=\begin{bmatrix}\boldsymbol{M}^{-1}+\rho\boldsymbol{M}^{-1}\boldsymbol{b}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{M}^{-1} & -\rho\boldsymbol{M}^{-1}\boldsymbol{b}\\ -\rho\boldsymbol{b}^{\mathrm{T}}\boldsymbol{M}^{-1} & \rho\end{bmatrix}$，得到 $(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}$。在此迭代过程中，计算出 $\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)}$，便可得到 $\boldsymbol{M}^{-1}$，$\boldsymbol{b}$ 可根据 $\boldsymbol{a}\boldsymbol{A}^{(J-1)\mathrm{T}}$ 计算得到，$\boldsymbol{a}$ 为归一化原子，则 $c=\boldsymbol{a}^{\mathrm{T}}\boldsymbol{a}=1$。需要求解的目标函数可以写为

$$\min_{\boldsymbol{x}^{(J-1)},z}\left\|[\boldsymbol{A}^{(J-1)}\ \boldsymbol{A}(\Lambda^{(J)})]\cdot\begin{bmatrix}\boldsymbol{x}^{(J-1)}\\ z\end{bmatrix}-\boldsymbol{b}\right\|_2^2 \tag{4.54}$$

上式中，$\boldsymbol{A}^{(J-1)}$ 表示前 $J-1$ 次入选的所有原子形成的矩阵，$\boldsymbol{x}^{(J-1)}$ 表示前 $J-1$ 次的稀疏表示系数，$\boldsymbol{A}(\Lambda^{(J)})$ 表示第 J 次入选的原子，$\Lambda^{(J)}$ 表示第 J 次入选原子所在位置，z 为第 J 次入选原子的稀疏表示系数。运用前面推导最小二乘法证明公式 $\dfrac{\partial\left\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\right\|_2^2}{\partial\boldsymbol{x}}=2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})$，并令其等于零，可得

$$\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\\ \boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\end{bmatrix}[\boldsymbol{A}^{(J-1)}\ \boldsymbol{A}(\Lambda^{(J)})]\begin{bmatrix}\boldsymbol{x}^{(J-1)}\\ \mathrm{z}\end{bmatrix}-\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\\ \boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\end{bmatrix}\boldsymbol{b}=\boldsymbol{0} \tag{4.55}$$

得到第 J 次的稀疏表示系数：

$$\begin{bmatrix}\boldsymbol{x}^{(J-1)}\\ z\end{bmatrix}=\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}(\Lambda^{(J)})\\ \boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\boldsymbol{A}(\Lambda^{(J)})\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{b}\\ \boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\boldsymbol{b}\end{bmatrix} \tag{4.56}$$

求稀疏表示系数的复杂环节是求 $\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}(\Lambda^{(J)})\\ \boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\boldsymbol{A}(\Lambda^{(J)})\end{bmatrix}^{-1}$，相当于求 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$ 的逆矩阵。使用式(4.52)求 $\begin{bmatrix}\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}(\Lambda^{(J)})\\ \boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)} & \boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\boldsymbol{A}(\Lambda^{(J)})\end{bmatrix}^{-1}$，我们得到了矩阵 $[\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}]^{-1}$。找到与第 $J-1$ 次迭代后的残差最相近的原子，作为第 J 次迭代的入选原子，若该原子与 $\boldsymbol{A}^{(J-1)}$ 正交，则需要求逆的矩阵变为块对角阵，稀疏表示系数为 $\boldsymbol{x}^{(J-1)}=[\boldsymbol{A}^{(J-1)}]^{+}\boldsymbol{b}$ 且 $z=\boldsymbol{A}(\Lambda^{(J)})^{\mathrm{T}}\boldsymbol{b}/\left\|\boldsymbol{A}(\Lambda^{(J)})\right\|_2^2$。如果第 J 次迭代的入选原子与 $\boldsymbol{A}^{(J-1)}$ 不正交，使用式(4.56)进行求解可以大大降低算法复杂度。

从 MP 算法到 OMP 算法，再到 LS-OMP 算法，纵观这一系列算法的发展史，新算法总是对原算法的补充完善。MP 算法求稀疏表示系数不精确，进而提出

OMP 算法来改进精确度问题，OMP 算法虽然改进了精确度问题，但其中有两个环节算法复杂度较高：一个是找原子，也就是在很多原子中找一个与残差最相似的原子；第二个就是求逆的问题，本节讲到的 LS-OMP 算法就是为解决求逆的问题而提出的。

4. Cholesky 快速正交匹配追踪算法

Cholesky 快速正交匹配追踪算法和 LS-OMP 算法一样，解决的仍然是第二个问题：$\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$ 中算法复杂度较高的求逆矩阵的问题。OMP 算法得到 $\boldsymbol{A}^{(J)}$ 以后，用最小二乘法去求解稀疏表示系数，通过式 $\boldsymbol{x}^{(J)}=\arg\min_{\boldsymbol{\theta}}\left\|\boldsymbol{b}-\boldsymbol{A}^{(J)}\boldsymbol{\theta}\right\|_2$ 计算新的 $\boldsymbol{b}$ 的逼近量和新的残差，用 $\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$ 求解稀疏表示系数。显而易见，该过程复杂度要求较高，故而衍生出 Cholesky 快速正交匹配追踪算法。

Cholesky 快速正交匹配追踪算法将上述问题进行了转化，在 $\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}=\boldsymbol{b}$ 等式两边同时左乘 $\boldsymbol{A}^{(J)\mathrm{T}}$。$\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}=\boldsymbol{b}$ 即为最小二乘法需要解决的关键问题，Cholesky 快速正交匹配追踪算法在 $\boldsymbol{A}^{(J)}$ 前面又乘以 $\boldsymbol{A}^{(J)\mathrm{T}}$，得到：

$$\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b} \tag{4.57}$$

然后将 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$ 分别用一个上三角矩阵 $\boldsymbol{L}^{(J)}$ 和一个下三角矩阵 $\boldsymbol{L}^{(J)\mathrm{T}}$ 代替，将 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}$ 用 $\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}$ 替换。已得到 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$，然后令 $\boldsymbol{\alpha}^{(J)}=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$，则 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$ 可改写为

$$\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{\alpha}^{(J)} \tag{4.58}$$

注意观察式(4.57)和式(4.58)之间的相关性，会发现两式的解其实是一样的，都是求稀疏表示系数 $\boldsymbol{x}^{(J)}$，两式形式也是一样，差别则在于 $\boldsymbol{L}^{(J)}$ 和 $\boldsymbol{L}^{(J)\mathrm{T}}$ 分别为上三角矩阵和下三角矩阵。至于这里为什么要换成上三角矩阵和下三角矩阵，则是由于用到之前学过的另外一个方法——高斯消去法：当线性方程等式左边的矩阵是上三角矩阵和下三角矩阵时，可以一直回代直至解出所有未知数，Cholesky 快速正交匹配追踪算法的创新点就在于此，将 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$ 这两个普通矩阵替换成上三角矩阵和下三角矩阵，就可以用高斯消去法的快速方法迭代求解稀疏表示系数 $\boldsymbol{x}^{(J)}$。

高斯消去法首先令 $\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{q}$，则

$$\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{L}^{(J)}\boldsymbol{q}=\boldsymbol{\alpha}^{(J)} \tag{4.59}$$

通过 $\boldsymbol{L}^{(J)}\boldsymbol{q}=\boldsymbol{\alpha}^{(J)}$ 首先将 $\boldsymbol{q}$ 解出，然后根据 $\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{q}$ 中的已知条件解出稀疏表示系数 $\boldsymbol{x}^{(J)}$，Cholesky 快速正交匹配追踪算法实际上就是将方程 $\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}=\boldsymbol{b}$ 转换为另一种形式 $\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{\alpha}^{(J)}$，然后利用快速算法进行求解。此方法之所以称为 Cholesky 快速正交匹配追踪算法，是由于运用 Cholesky 分解定理解决了整个方法最为关键的部分：如何将 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}$ 换成 $\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}$，即怎样将两个普通矩阵相乘改写为一个上三角矩阵和一个下三角矩阵相乘的问题。

Cholesky 分解的具体方法和步骤如下。

假设 $\boldsymbol{W}^{(J-1)}=\boldsymbol{L}^{(J-1)}\boldsymbol{L}^{(J-1)\mathrm{T}}$，$\boldsymbol{W}^{(J)}=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$，上三角矩阵乘下三角矩阵 $\boldsymbol{L}^{(J-1)}\boldsymbol{L}^{(J-1)\mathrm{T}}$ 等于 $\boldsymbol{W}^{(J-1)}$，通过上一小节可知 $\boldsymbol{W}^{(J)}=\begin{bmatrix}\boldsymbol{W}^{(J-1)} & \boldsymbol{v}\\ \boldsymbol{v}^{\mathrm{T}} & l\end{bmatrix}$ 定然成立。这个方法其实就是用前一次迭代的 $\boldsymbol{L}^{(J-1)}$ 求当前的 $\boldsymbol{L}^{(J)}$，假设 $\boldsymbol{W}^{(J)}=\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}$，$\boldsymbol{L}^{(J)}$ 和 $\boldsymbol{L}^{(J-1)}$ 之间可以写为

$$\boldsymbol{W}^{(J)}=\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}=\begin{pmatrix}\boldsymbol{L}^{(J-1)} & \boldsymbol{0}\\ \boldsymbol{w}^{\mathrm{T}} & \sqrt{l-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}\end{pmatrix}\begin{pmatrix}\boldsymbol{L}^{(J-1)\mathrm{T}} & \boldsymbol{w}\\ \boldsymbol{0}^{\mathrm{T}} & \sqrt{l-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}\end{pmatrix} \tag{4.60}$$

可以看出：$\boldsymbol{L}^{(J)}=\begin{pmatrix}\boldsymbol{L}^{(J-1)} & \boldsymbol{0}\\ \boldsymbol{w}^{\mathrm{T}} & \sqrt{l-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}\end{pmatrix}$，$\boldsymbol{L}^{(J-1)}\boldsymbol{w}=\boldsymbol{v}$，且 $\boldsymbol{W}^{(J)}=\begin{bmatrix}\boldsymbol{W}^{(J-1)} & \boldsymbol{v}\\ \boldsymbol{v}^{\mathrm{T}} & l\end{bmatrix}$。通过 $\boldsymbol{A}^{(J-1)}$ 乘以第 J 次新入选的原子 $\boldsymbol{a}$ 可计算出 $\boldsymbol{v}$，且经过前一次迭代得到 $\boldsymbol{L}^{(J-1)}$，通过式 $\boldsymbol{L}^{(J-1)}\boldsymbol{w}=\boldsymbol{v}$ 可算出 $\boldsymbol{w}$，由 $\boldsymbol{L}^{(J)}=\begin{pmatrix}\boldsymbol{L}^{(J-1)} & \boldsymbol{0}\\ \boldsymbol{w}^{\mathrm{T}} & \sqrt{l-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}\end{pmatrix}$ 可得 $\boldsymbol{L}^{(J)}$，其中，$l=1$。由此得到三角矩阵 $\boldsymbol{L}^{(J)}$。

OMP 算法中需要计算的 $\boldsymbol{W}^{(J)}=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$ 每次迭代都要更新最后一行和一列。而 Cholesky 分解方法让这一类矩阵仅仅计算最后一行，Cholesky 分解定理令 $\boldsymbol{W}^{(J-1)}=\boldsymbol{L}^{(J-1)}\boldsymbol{L}^{(J-1)\mathrm{T}}$，在 $\boldsymbol{W}^{(J-1)}$ 的基础上增加一行和一列，得到 $\boldsymbol{W}^{(J)}=\begin{bmatrix}\boldsymbol{W}^{(J-1)} & \boldsymbol{v}\\ \boldsymbol{v}^{\mathrm{T}} & l\end{bmatrix}$，其中，$\boldsymbol{v}=\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}(\Lambda^{(J)})$。然后对 $\boldsymbol{W}^{(J)}$ 进行 Cholesky 分解得到：$\boldsymbol{W}^{(J)}=\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}=\begin{pmatrix}\boldsymbol{L}^{(J-1)} & \boldsymbol{0}\\ \boldsymbol{w}^{\mathrm{T}} & \sqrt{l-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}\end{pmatrix}\begin{pmatrix}\boldsymbol{L}^{(J-1)\mathrm{T}} & \boldsymbol{w}\\ \boldsymbol{0}^{\mathrm{T}} & \sqrt{l-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}\end{pmatrix}$，其中，令 $\boldsymbol{L}^{(J-1)}\boldsymbol{w}=\boldsymbol{v}$，通过 $\boldsymbol{v}=\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}(\Lambda^{(J)})$ 得到 $\boldsymbol{v}$，继而求出 $\boldsymbol{w}$，然后将 $\boldsymbol{w}$ 代入 $\boldsymbol{L}^{(J)}=\begin{pmatrix}\boldsymbol{L}^{(J-1)} & \boldsymbol{0}\\ \boldsymbol{w}^{\mathrm{T}} & \sqrt{l-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}\end{pmatrix}$ 得到第 J 次迭代的上三角矩阵 $\boldsymbol{L}^{(J)}$。

Cholesky 快速正交匹配追踪算法具体流程如下。

输入：$n\times m$ 维的字典 $\boldsymbol{A}$，n 维的数据向量 $\boldsymbol{b}$。

输出：稀疏表示系数 $\boldsymbol{x}$。

初始化：原子号码集合 $\boldsymbol{\Omega}$ 设为空集，$\boldsymbol{L}=[1]$，$\boldsymbol{r}=\boldsymbol{b}$，$\boldsymbol{\alpha}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$。

执行如下步骤直到满足停止条件，迭代次数 $J=1,2,3,\cdots$。

①在第 J 步寻找与第 $J-1$ 步的稀疏表示残差内积最大的原子，并将该原子的号码 $\Lambda^{(J)}$ 加入原子号码集合 $\Lambda^{(J)}\in\arg\max_{\varpi\in\Omega}\left|\left\langle\boldsymbol{r}^{(J-1)},\boldsymbol{A}(\varpi)\right\rangle\right|$。

②计算新的向量 $\boldsymbol{b}$ 的逼近量和新的残差：$\boldsymbol{x}^{(J)}=\arg\min_{\boldsymbol{\theta}}\left\|\boldsymbol{b}-\boldsymbol{A}^{(J)}\boldsymbol{\theta}\right\|_2$；

稀疏表示系数：$\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$。

由 $\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}=\boldsymbol{b}$ 得到 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$，并由 Cholesky 分解定理转化为 $\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{\alpha}^{(J)}$，使用高斯消去法求解方程 $\boldsymbol{L}^{(J)}\boldsymbol{q}=\boldsymbol{\alpha}^{(J)}$ 和 $\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{q}$。其中，$\boldsymbol{\alpha}^{(J)}$ 是选择 $\boldsymbol{\alpha}$ 里号码与 Ω 对应的元素。

虽然 Cholesky 快速正交匹配追踪算法大大解决了 OMP 算法中算法复杂度大的问题，但此方法仍然只是解决了 OMP 算法中第二个复杂问题，也就是求逆的难题。关于 OMP 算法中另外一个算法复杂度比较高的问题仍然没有解决，那就是寻找与残差内积最大的原子过程中的复杂度问题。

5. 块正交匹配追踪算法

无论是 LS-OMP 算法还是 Cholesky 快速正交匹配追踪算法，都是为了解决 OMP 算法的第二个难题提出的，但关于 OMP 算法中寻找最佳匹配原子的问题仍没有解决。通过 $\Lambda^{(J)}\in\arg\max_{\varpi\in\Omega}\left|\left\langle\boldsymbol{r}^{(J-1)},\boldsymbol{A}(\varpi)\right\rangle\right|$ 可知，寻找最佳原子的过程需要用残差向量与所有的原子对比相似性，找到最佳匹配原子。于是有人提出了块正交匹配追踪（batch orthogonal matching pursuit，B-OMP）算法来解决原子匹配的复杂度问题。

学习 B-OMP 算法之前，首先要弄清几个变量的概念，以便对 B-OMP 算法有个更好的了解。令 $\boldsymbol{\alpha}^{(0)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$，即 $\boldsymbol{\alpha}^{(0)}$ 为系数矩阵 $\boldsymbol{A}$ 的转置乘以初始残差 $\boldsymbol{b}$。令 $\boldsymbol{G}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$，$\boldsymbol{G}$ 是一个确定的矩阵，为系数矩阵 $\boldsymbol{A}$ 的转置乘以它本身。$\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$ 为第 J 次入选的原子的转置乘以 $\boldsymbol{A}$ 中第 J 次入选原子本身。$\boldsymbol{G}^{(J)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}^{(J)}$ 表示系数矩阵 $\boldsymbol{A}$ 的转置乘以第 J 次入选的原子。$\boldsymbol{G}^{(J),(J)}=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$，表示第 J 次入选原子的转置乘以第 J 次入选的原子。

假如系数矩阵 $\boldsymbol{A}$ 中有三个原子，令 $\boldsymbol{A}=[\boldsymbol{a}_1\quad\boldsymbol{a}_2\quad\boldsymbol{a}_3]$，$\boldsymbol{A}^{\mathrm{T}}=[\boldsymbol{a}_1^{\mathrm{T}}\quad\boldsymbol{a}_2^{\mathrm{T}}\quad\boldsymbol{a}_3^{\mathrm{T}}]$。$\boldsymbol{G}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$，则 $\boldsymbol{G}=\begin{bmatrix}\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_1 & \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_2 & \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_3\\ \boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{a}_1 & \boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{a}_2 & \boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{a}_3\\ \boldsymbol{a}_3^{\mathrm{T}}\boldsymbol{a}_1 & \boldsymbol{a}_3^{\mathrm{T}}\boldsymbol{a}_2 & \boldsymbol{a}_3^{\mathrm{T}}\boldsymbol{a}_3\end{bmatrix}$。接下来通过实例，具体了解一下 $\boldsymbol{A}^{(J)}$ 与 $\boldsymbol{G}^{(J)}$、$\boldsymbol{G}^{(J),(J)}$ 之间的关系。当第 J 次迭代时，入选的原子为 $\boldsymbol{a}_1$ 和 $\boldsymbol{a}_3$ 时，则 $\boldsymbol{G}^{(J)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}^{(J)}=\begin{bmatrix}\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_1 & \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_3\\ \boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{a}_1 & \boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{a}_3\\ \boldsymbol{a}_3^{\mathrm{T}}\boldsymbol{a}_1 & \boldsymbol{a}_3^{\mathrm{T}}\boldsymbol{a}_3\end{bmatrix}$，即为 $\boldsymbol{A}^{\mathrm{T}}$ 乘以原子 $\boldsymbol{a}_1$ 和 $\boldsymbol{a}_3$ 组成的矩阵，$\boldsymbol{G}^{(J),(J)}=\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}=\begin{bmatrix}\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_1 & \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_3\\ \boldsymbol{a}_3^{\mathrm{T}}\boldsymbol{a}_1 & \boldsymbol{a}_3^{\mathrm{T}}\boldsymbol{a}_3\end{bmatrix}$。从上述关系可以得出，如果把 $\boldsymbol{G}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 计算出来后，第 J 次迭代的 $\boldsymbol{G}^{(J)}$ 和 $\boldsymbol{G}^{(J),(J)}$ 则不需要专门计算，只需要在 $\boldsymbol{G}$ 中选择入选原子的相应位置即可，这

比乘法运算的算法复杂度要低得多。B-OMP 的主要思想就是首先计算出 $\boldsymbol{G}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$，然后接下来的环节就是在 $\boldsymbol{G}$ 中选择元素。但具体是怎么操作的呢？

首先，已知 $\boldsymbol{\alpha}^{(J)}$ 为变量，假设：

$$\boldsymbol{\alpha}^{(J)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}^{(J-1)} \tag{4.61}$$

其中，$\boldsymbol{r}^{(J-1)}$ 为第 $J-1$ 次迭代的残差。找原子环节，就是在 $\boldsymbol{A}$ 中，找到与残差 $\boldsymbol{r}^{(J-1)}$ 最相近的原子，即将残差与系数矩阵所有原子求内积，找到与残差内积最大的原子。$\boldsymbol{\alpha}^{(J)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}^{(J-1)}$ 求得的 $\boldsymbol{\alpha}^{(J)}$ 为向量，实际上就是在向量中寻找一个最大的值，如下公式所示：

$$\Lambda^{(J)}\in\arg\max_{\varpi\in\Omega}\left|\left\langle\boldsymbol{r}^{(J-1)},\boldsymbol{A}(\varpi)\right\rangle\right| \tag{4.62}$$

$\boldsymbol{r}^{(J-1)}$ 为残差，残差和原子 $\boldsymbol{A}(\varpi)$ 求内积，然后找到一个内积最大的原子，这便是第一个算法复杂度高的问题——找原子问题。B-OMP 算法令 $\boldsymbol{\alpha}^{(J)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}^{(J-1)}$，$\boldsymbol{r}^{(J-1)}$ 可表示为

$$\boldsymbol{r}^{(J-1)}=\boldsymbol{b}-\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} \tag{4.63}$$

其中，$\boldsymbol{b}$ 为目标信号，也就是最终要逼近的信号，$\boldsymbol{A}^{(J-1)}$ 为第 $(J-1)$ 次入选的原子，$\boldsymbol{x}^{(J-1)}$ 为第 $(J-1)$ 次迭代的稀疏表示系数。接下来对式(4.61)进行变形，将式(4.63)代入式(4.61)并展开，得

$$\begin{aligned}\boldsymbol{\alpha}^{(J)}&=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)})\\&=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}\end{aligned} \tag{4.64}$$

由于设定 $\boldsymbol{\alpha}^{(0)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$，$\boldsymbol{G}^{(J)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}^{(J)}$ 可得

$$\boldsymbol{\alpha}^{(J)}=\boldsymbol{\alpha}^{(0)}-\boldsymbol{G}^{(J-1)}\boldsymbol{x}^{(J-1)} \tag{4.65}$$

$\boldsymbol{\alpha}^{(0)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$ 为初始设定值，可以直接计算出，$\boldsymbol{G}^{(J-1)}$ 和 $\boldsymbol{x}^{(J-1)}$ 也可以通过计算得到，进一步可以得到 $\boldsymbol{\alpha}^{(J)}$，找到 $\boldsymbol{\alpha}^{(J)}$ 中最大原子的号码即为第 J 次的入选原子号码，进而找到最佳入选原子。

接下来对残差的计算公式进行推导，残差可通过 $\boldsymbol{G}$ 直接计算。$\boldsymbol{G}$ 为字典矩阵 $\boldsymbol{A}$ 的转置乘以 $\boldsymbol{A}$ 本身，$\boldsymbol{A}$ 矩阵为已知的固定值，则 $\boldsymbol{G}$ 也为固定值。$\boldsymbol{G}$ 一旦计算出来，我们只需要选择一些原子的号码便可得到残差，这比 OMP 算法中反复求内积算法的复杂度要低。另一个需要计算的参量就是 $\left\|\boldsymbol{r}^{(J)}\right\|_2^2$，也就是求残差的长度。已知：

$$(\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}=0 \tag{4.66}$$

且

$$\begin{aligned}\boldsymbol{r}^{(J)}&=\boldsymbol{b}-\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}\\&=\boldsymbol{b}-\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}-\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}+\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}\\&=\boldsymbol{r}^{(J-1)}-\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}+\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}\end{aligned} \tag{4.67}$$

根据二范数计算公式，有 $\boldsymbol{r}^{(J)}$ 的二范数平方为

$$\begin{aligned}\left\|\boldsymbol{r}^{(J)}\right\|_2^2 &= (\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{r}^{(J)} \\ &= (\boldsymbol{r}^{(J)})^{\mathrm{T}}(\boldsymbol{r}^{(J-1)} - \boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)} + \boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}) \\ &= (\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{r}^{(J-1)} - (\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)} + (\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}\end{aligned} \tag{4.68}$$

其中，$(\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{r}^{(J-1)}$ 为前一次的残差和当前残差之间的内积，由 $(\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)} = 0$，将上式进行化简，得

$$\begin{aligned}\left\|\boldsymbol{r}^{(J)}\right\|_2^2 &= (\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{r}^{(J-1)} + (\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} \\ &= (\boldsymbol{r}^{(J-1)} - \boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)} + \boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)})^{\mathrm{T}}\boldsymbol{r}^{(J-1)} + (\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} \\ &= \left\|\boldsymbol{r}^{(J-1)}\right\|_2^2 - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{r}^{(J-1)} + (\boldsymbol{x}^{(J-1)})^{\mathrm{T}}(\boldsymbol{A}^{(J-1)})^{\mathrm{T}}\boldsymbol{r}^{(J-1)} + (\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}\end{aligned} \tag{4.69}$$

由于 $(\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)})^{\mathrm{T}}$ 和 $\boldsymbol{r}^{(J-1)}$ 正交，所以 $(\boldsymbol{x}^{(J-1)})^{\mathrm{T}}(\boldsymbol{A}^{(J-1)})^{\mathrm{T}}\boldsymbol{r}^{(J-1)} = 0$，进一步化简得

$$\left\|\boldsymbol{r}^{(J)}\right\|_2^2 = \left\|\boldsymbol{r}^{(J-1)}\right\|_2^2 - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{r}^{(J-1)} + (\boldsymbol{r}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} \tag{4.70}$$

把 $\boldsymbol{r}^{(J-1)} = \boldsymbol{b} - \boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}$ 和 $\boldsymbol{r}^{(J)} = \boldsymbol{b} - \boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}$ 代入式(4.70)，得

$$\begin{aligned}\left\|\boldsymbol{r}^{(J)}\right\|_2^2 &= \left\|\boldsymbol{r}^{(J-1)}\right\|_2^2 - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}(\boldsymbol{b} - \boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}) + (\boldsymbol{b} - \boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} \\ &= \left\|\boldsymbol{r}^{(J-1)}\right\|_2^2 - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{b} + (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} \\ &\quad + \boldsymbol{b}^{T}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} \\ &= \left\|\boldsymbol{r}^{(J-1)}\right\|_2^2 - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{b} + \boldsymbol{b}^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}\end{aligned} \tag{4.71}$$

将 $\boldsymbol{b} = \boldsymbol{r}^{(J)} + \boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)} = \boldsymbol{r}^{(J-1)} + \boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}$ 代入式(4.71)得

$$\begin{aligned}\left\|\boldsymbol{r}^{(J)}\right\|_2^2 &= \left\|\boldsymbol{r}^{(J-1)}\right\|_2^2 - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}(\boldsymbol{r}^{(J)} + \boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}) + (\boldsymbol{r}^{(J-1)} + \boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)})^{T}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} \\ &= \left\|\boldsymbol{r}^{(J-1)}\right\|_2^2 - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{r}^{(J)} - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)} \\ &\quad + (\boldsymbol{r}^{(J-1)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} + (\boldsymbol{x}^{(J-1)})^{\mathrm{T}}(\boldsymbol{A}^{(J-1)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}\end{aligned} \tag{4.72}$$

由于 $(\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)})^{\mathrm{T}}$ 和 $\boldsymbol{r}^{(J)}$ 正交，$(\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)})^{\mathrm{T}}$ 和 $\boldsymbol{r}^{(J-1)}$ 正交，所以 $(\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{r}^{(J)}=0$，且 $(\boldsymbol{r}^{(J-1)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)}=0$，因此有

$$\begin{aligned}\left\|\boldsymbol{r}^{(J)}\right\|_2^2 &= \left\|\boldsymbol{r}^{(J-1)}\right\|_2^2 - (\boldsymbol{x}^{(J)})^{\mathrm{T}}(\boldsymbol{A}^{(J)})^{\mathrm{T}}\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)} + (\boldsymbol{x}^{(J-1)})^{\mathrm{T}}(\boldsymbol{A}^{(J-1)})^{\mathrm{T}}\boldsymbol{A}^{(J-1)}\boldsymbol{x}^{(J-1)} \\ &= \left\|\boldsymbol{r}^{(J-1)}\right\|_2^2 - (\boldsymbol{x}^{(J)})^{\mathrm{T}}\boldsymbol{G}^{(J),(J)}\boldsymbol{x}^{(J)} + (\boldsymbol{x}^{(J-1)})^{\mathrm{T}}\boldsymbol{G}^{(J-1),(J-1)}\boldsymbol{x}^{(J-1)}\end{aligned} \tag{4.73}$$

B-OMP 算法本质上就是根据残差之间的关系，求解稀疏表示系数的过程。B-OMP 算法的具体流程如下。

输入： $n\times m$ 维的字典 $\boldsymbol{A}$，n 维的数据向量 $\boldsymbol{b}$。

输出： 稀疏表示系数 $\boldsymbol{x}$。

初始化： 原子号码集合 Ω 设为空集，$\boldsymbol{L}=[1]$，$\boldsymbol{r}=\boldsymbol{b}$，$\boldsymbol{\alpha}^{(0)}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$，$\boldsymbol{G}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$。

执行如下步骤直到满足停止条件，迭代次数 $J=1,2,3,\cdots$。

①在第 J 步寻找与 $J-1$ 步的残差内积最大的原子，并将该原子的号码 $\Lambda^{(J)}$ 加入原子号码集合：

$$\Lambda^{(J)}\in\arg\max_{\varpi\in\Omega}\left|\boldsymbol{\alpha}^{(J)}(\varpi)\right| \tag{4.74}$$

作为第 J 次入选的原子号码，$\boldsymbol{\Phi}^{(J)}=[\Lambda^{(1)}\quad\Lambda^{(2)}\quad\cdots\quad\Lambda^{(J)}]$ 为入选原子号码的集合，$\boldsymbol{A}^{(J)}=[\boldsymbol{A}(\Lambda^{(1)})\quad\boldsymbol{A}(\Lambda^{(2)})\quad\cdots\quad\boldsymbol{A}(\Lambda^{(J)})]$ 为所有入选的原子。

②通过解下列方程得到 $\boldsymbol{w}$：

$$\boldsymbol{L}^{(J-1)}\boldsymbol{w}=\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}(\Lambda^{(J)}) \tag{4.75}$$

$$\boldsymbol{L}^{(J)}=\begin{pmatrix}\boldsymbol{L}^{(J-1)} & \boldsymbol{0}\\ \boldsymbol{w}^{\mathrm{T}} & \sqrt{1-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}\end{pmatrix} \tag{4.76}$$

$$\boldsymbol{A}^{(J)}=[\boldsymbol{A}^{(J-1)}\quad\boldsymbol{A}(\Lambda^{(J)})] \tag{4.77}$$

其中，$\boldsymbol{L}^{(J)}$ 为第 J 次迭代的上三角矩阵。

③通过求解下列方程得到 $\boldsymbol{L}^{(J)}$：

$$\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{\alpha}^{(0)}(\boldsymbol{\Phi}^{(J)}) \tag{4.78}$$

其中，$\boldsymbol{\alpha}^{(0)}(\boldsymbol{\Phi}^{(J)})$ 是选择的 $\boldsymbol{\alpha}^{(0)}$ 中号码与 $\boldsymbol{\Phi}^{(J)}$ 对应的元素。

④对推导得到的 $\left\|\boldsymbol{r}^{(J)}\right\|_2^2=\left\|\boldsymbol{r}^{(J-1)}\right\|_2^2-(\boldsymbol{x}^{(J)})^{\mathrm{T}}\boldsymbol{G}^{(J),(J)}\boldsymbol{x}^{(J)}+(\boldsymbol{x}^{(J-1)})^{\mathrm{T}}\boldsymbol{G}^{(J-1),(J-1)}\boldsymbol{x}^{(J-1)}$ 和 $\boldsymbol{\alpha}^{(J)}=\boldsymbol{\alpha}^{(0)}-\boldsymbol{G}^{(J-1)}\boldsymbol{x}^{(J-1)}$ 进行化简，令

$$\boldsymbol{\beta}^{(J)}=\boldsymbol{G}^{(J),(J)}\boldsymbol{x}^{(J)} \tag{4.79}$$

进一步降低算法复杂度，可得

$$\boldsymbol{\alpha}^{(J)}=\boldsymbol{\alpha}^{(0)}-\boldsymbol{\beta}^{(J-1)} \tag{4.80}$$

$$\left\|\boldsymbol{r}^{(J)}\right\|_2^2=\left\|\boldsymbol{r}^{(J-1)}\right\|_2^2-(\boldsymbol{x}^{(J)})^{\mathrm{T}}\boldsymbol{\beta}^{(J)}+(\boldsymbol{x}^{(J-1)})^{\mathrm{T}}\boldsymbol{\beta}^{(J-1)} \tag{4.81}$$

在上述 B-OMP 算法具体流程中，可以明显发现，在选择原子号码的方法上与之前的算法有明显不同。前文中的算法都是对向量求内积寻找最佳原子，但 B-OMP 算法中，仅仅是在计算出了 $\boldsymbol{G}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 之后，从中选择最相似原子的对应位置即可，大大降低了算法复杂度。在 B-OMP 算法中，关于计算稀疏表示系数过程中求逆问题的算法复杂度问题，采用的仍是 Cholesky 快速正交匹配追踪算法的方法。令

$\boldsymbol{L}^{(J)}\boldsymbol{L}^{(J)\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{\alpha}^{(0)}(\boldsymbol{\Phi}^{(J)})$，求解 $\boldsymbol{\alpha}^{(J)}$，首先需要计算出 $\boldsymbol{\beta}^{(J)}$，从 $\boldsymbol{\beta}^{(J)}$ 的定义式可以看出，寻找最佳原子的过程实际上就是在总体中寻找最佳元素，来代替复杂的乘法运算。$\boldsymbol{\alpha}^{(J)}=\boldsymbol{\alpha}^{(0)}-\boldsymbol{\beta}^{(J)}$ 中，已知 $\boldsymbol{\alpha}^{(0)}$ 和 $\boldsymbol{\beta}^{(J)}$，可直接经过简单计算得到 $\boldsymbol{\alpha}^{(J)}$，最佳匹配值即为 $\boldsymbol{\alpha}^{(J)}$ 中的最大值，这样避免了每次求残差之后，都要将残差与所有原子做内积，并且这个过程需要反复迭代，寻找最相似原子的过程就会相当麻烦。对于式 (4.81)，求得 $\boldsymbol{\beta}^{(J)}$ 后，结合其他已知条件，残差长度的求解就会变得比较简单。

6. 算法复杂度分析

算法复杂度是指算法中所有的运算次数的总和，有的时候用等价无穷小来估计，有的时候直接计算准确次数。从算法复杂度的估算中，可以看出算法哪些步骤最为耗时，并且可以看出算法的复杂度主要受到哪些参数影响。同一问题可用不同算法解决，而一个算法的质量优劣将影响到算法乃至程序的效率。算法分析的目的在于选择合适算法和对算法进行合理改进。本节中，通过对前面介绍的 Cholesky 正交匹配追踪算法和 B-OMP 算法的算法复杂度的分析和对比，进一步说明对于同一个问题，应用不同的方法实现时，能够达到不同的效果。因此，对于一个算法的改进，算法复杂度也是一个应当重要考虑的因素。

1) Cholesky 正交匹配追踪算法复杂度分析

Cholesky 正交匹配追踪算法中，$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}$ 的复杂度为 $2mn$，其中，乘法和加法各 mn 次，K 次迭代共需要 $2Kmn$ 次运算。接下来寻找 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}$ 最大的绝对值，其中需要求 n 次绝对值，n 次比较运算寻找最大值，总共 $2n$ 次运算，K 次迭代共需要 $2Kn$ 次运算。当第 J 次迭代 $\boldsymbol{L}$ 大小为 $c\times c$，计算 $\boldsymbol{w}$ 使用回代法需要 c^2 次运算，使用回代法求解方程 $\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{\gamma}^{(J)}=\boldsymbol{\alpha}^{(J)}$ 需要 $2c^2$ 次运算，这两个回代共需要 $3c^2$ 次运算。第一次迭代时，$\boldsymbol{L}$ 的大小为 1×1，第二次迭代时，$\boldsymbol{L}$ 的大小为 2×2，以此类推，K 次迭代共需要 $3(1^2+2^2+\cdots+K^2)=3K(K+1)(2K+1)/6$ 次运算。计算 $\boldsymbol{r}^{(J)}$ 需要 m 次减法运算，K 次迭代共需要 Km 次运算。计算 $\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}$ 和 $\boldsymbol{A}^{(J-1)\mathrm{T}}\boldsymbol{A}(\Lambda^{(1)})$ 分别需要 cm 次运算，K 次迭代共需要 $2(1+2+\cdots+K)m=K(K+1)m$ 次运算。因此，Cholesky 正交匹配追踪算法的总运算复杂度为

$$T_{\text{Cholesky-omp}}=2Kmn+2Kn+3K(K+1)(2K+1)/6+Km+K(K+1)m \tag{4.82}$$

2) B-OMP 算法复杂度分析

块正交匹配追踪算法中，假设 $\boldsymbol{G}$ 是事先计算好的。该算法在计算 $\boldsymbol{\alpha}^0=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$ 需要 $2mn$ 次运算。接下来寻找 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}$ 最大的绝对值，其中，需要求 n 次绝对值，n 次比较运算寻找最大值，总共 $2n$ 次运算，K 次迭代共需要 $2Kn$ 次运算。当第 J 次迭代 $\boldsymbol{L}$ 大小为 $c\times c$，计算 $\boldsymbol{w}$ 使用回代法需要 c^2 次运算。使用回代法求解方程 $\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{x}^{(J)}=\boldsymbol{\alpha}^{(0)}(\boldsymbol{\Phi})$ 需要 $2c^2$ 次运算。这两个回代共需要 $3c^2$ 次运算，第一次迭代时，$\boldsymbol{L}$ 的大小为 1×1，

第二次迭代时，$\boldsymbol{L}$ 的大小为 2×2，以此类推，K 次迭代共需要 $3(1^2+2^2+\cdots+K^2)=3K(K+1)(2K+1)/6$ 次运算。计算 $\boldsymbol{\beta}$ 需要 $2cn$ 次运算，K 次迭代共需要 $2(1+2+\cdots+K)n=K(K+1)n$ 次运算，更新 $\boldsymbol{\alpha}^{(J)}$ 需要 n 次运算，K 次迭代共需要 Kn 次运算。因此，B-OMP 算法的总算法复杂度为

$$T_{\text{batch-omp}}=2mn+2Kn+3K(K+1)(2K+1)/6+Kn+K(K+1)n \tag{4.83}$$

3) Cholesky 正交匹配追踪算法和 B-OMP 算法的对比

假设字典列数 $n=2m$，$K=\sqrt{m}/2$，$O(\cdot)$ 表示等价无穷小操作。计算可得：

$$\begin{aligned}T_{\text{Cholesky-omp}}&=O(2Kmn+2Kn+3K(K+1)(2K+1)/6+Km+K(K+1)m)\\&\approx 2m^{2.5}\end{aligned} \tag{4.84}$$

$$\begin{aligned}T_{\text{B-omp}}&=O(2mn+2Kn+3K(K+1)(2K+1)/6+Kn+K(K+1)n)\\&\approx 4.25m^{2}\end{aligned} \tag{4.85}$$

为了进一步展示块正交匹配追踪算法和 Cholesky 正交匹配追踪算法复杂度的不同，进行一个简单实验，如表 4.1 所示，两个算法随着样本个数的增加算法的执行时间不同，其中，每个样本大小设置为 256，字典大小为 256×512，算法执行时间越长，算法复杂度越高。从表 4.1 可以看出，随着样本数的增加，块正交匹配追踪算法明显降低了 Cholesky 正交匹配追踪算法的算法复杂度。从上述内容来看，块正交匹配追踪算法能够有效减小算法复杂度。减小算法复杂度的方法关键在于将算法复杂度较大的步骤进行拆分和转化，这就是改进算法的意义和考虑的角度。

表 4.1　在不同样本个数的情况下算法的执行时间(样本大小为 256，字典大小为 256×512)

样本个数	Cholesky 正交匹配追踪算法	B-OMP 算法	算法复杂度减小倍数
1	2.14	67.4	0.03
10	21.43	70.2	0.31
10^2	214.3	97.9	2.19
10^3	2142.7	374.8	5.72
10^5	214272.0	30，838.3	6.95

4.2.3　l_1 范数和 l_p 范数的求解方法

尽管 4.2.2 节中提出许多解决 l_0 范数约束的算法，但在找原子的时候，还是在做筛选操作，这种环节往往都是时间复杂度很大的，因此本节提出一种新的思路，将 l_0 范数替换为其他目标函数。用凸目标函数去替换非凸的目标函数，将非凸的问题转化为一个凸问题，再用凸优化的方法解决这个问题，避免了 l_0 范数本身存在的 NP 难问题。在实际情况中往往用 l_1 范数约束近似 l_0 范数约束进行求解，转变后的模型

如下所示：

$$\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_1 \quad \text{s.t.} \quad \boldsymbol{A}\boldsymbol{x}=\boldsymbol{b} \tag{4.86}$$

本节中将介绍 l_1 范数的求解问题。除了 l_0 范数、l_2 范数、l_1 范数之外，还有 l_p 范数。l_p 范数计算式为

$$\|\boldsymbol{x}\|_p=\sqrt[p]{x_1^p+x_2^p+\cdots+x_m^p} \tag{4.87}$$

l_0 范数、l_2 范数、l_1 范数都是特殊形式的 l_p 范数。上式中，p 值不同，l_p 范数对稀疏解的逼近程度也不一样，得到的解可能是稀疏的，也可能是不稀疏的，分别有以下几种情况：

当 $1<p<2$ 时，$\|\boldsymbol{x}\|_1$ 不能近似等价于 $\|\boldsymbol{x}\|_0$，所以得到的解不是稀疏解；

当 $p=1$ 时，$\|\boldsymbol{x}\|_1$ 近似等价于 $\|\boldsymbol{x}\|_0$，得到的解为稀疏解；

当 $0<p<1$ 时，$\|\boldsymbol{x}\|_1$ 也可近似等价于 $\|\boldsymbol{x}\|_0$，可以求得稀疏解。

前面已经讲过 l_2 范数和 l_0 范数的具体求解方法，接下来介绍一下 l_1 范数和 l_p 范数的求解方法。

1. 基追踪算法

求解 l_1 范数问题可以使用基追踪算法。基追踪算法的目标就是把 l_1 范数问题最终变为线性规划问题，转化为线性规划问题之后，就可以用单纯形法和对偶法等这些简单方法求解。

基追踪算法目标函数：

$$\min_{\boldsymbol{x}} \|\boldsymbol{x}\|_1 \quad \text{s.t.} \quad \boldsymbol{A}\boldsymbol{x}=\boldsymbol{b} \tag{4.88}$$

它的基本思想是：把非线性规划转化为线性规划。具体做法为：令 $\boldsymbol{x}=\boldsymbol{u}-\boldsymbol{v}$，其中，$\boldsymbol{u}$ 和 $\boldsymbol{v}$ 中的元素全部为非负数。例如，当 $\boldsymbol{x}=[3\quad -5\quad 0\quad 8]^{\mathrm{T}}$，$\boldsymbol{u}=[3\quad 0\quad 0\quad 8]^{\mathrm{T}}$，则 $\boldsymbol{v}=[0\quad 5\quad 0\quad 0]^{\mathrm{T}}$，也就说，$\boldsymbol{x}=\boldsymbol{u}-\boldsymbol{v}$，且 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 中元素全部为非负数。$\boldsymbol{u}$ 中的元素是 $\boldsymbol{x}$ 中的正元素，负元素的位置上都是 0，$\boldsymbol{v}$ 中的元素是 $\boldsymbol{x}$ 中的负元素的绝对值，正元素的位置都是 0。

假设 $\|\boldsymbol{x}\|_1$ 为

$$\|\boldsymbol{x}\|_1=\mathbf{1}^{\mathrm{T}}[\boldsymbol{u}+\boldsymbol{v}]=\mathbf{1}^{\mathrm{T}}\boldsymbol{z} \tag{4.89}$$

其中，$\boldsymbol{z}=[\boldsymbol{u}^{\mathrm{T}},\boldsymbol{v}^{\mathrm{T}}]^{\mathrm{T}}\in\boldsymbol{R}^{2n}$，$\mathbf{1}$ 为元素全为 1 的矩阵。以 $\boldsymbol{x}=[3\quad -5\quad 0\quad 8]^{\mathrm{T}}$ 为例，$\|\boldsymbol{x}\|_1=3+5+8$，同样 $\mathbf{1}^{\mathrm{T}}[\boldsymbol{u}+\boldsymbol{v}]=\begin{bmatrix}1\\1\\1\\1\end{bmatrix}^{\mathrm{T}}\times\left(\begin{bmatrix}3\\0\\0\\8\end{bmatrix}+\begin{bmatrix}0\\5\\0\\0\end{bmatrix}\right)=3+5+8$，故而，假设成立。则有

$$\boldsymbol{Ax}=\boldsymbol{A}(\boldsymbol{u}-\boldsymbol{v})=[\boldsymbol{A},-\boldsymbol{A}]\begin{bmatrix}\boldsymbol{u}\\ \boldsymbol{v}\end{bmatrix}=[\boldsymbol{A},-\boldsymbol{A}]\boldsymbol{z} \tag{4.90}$$

式(4.88)中目标函数变为线性规划问题：

$$\min_{\boldsymbol{z}}\mathbf{1}^{\mathrm{T}}\boldsymbol{z}\quad \text{s.t.}\,\boldsymbol{b}=[\boldsymbol{A},-\boldsymbol{A}]\boldsymbol{z}\ ,\ \boldsymbol{z}>\mathbf{0} \tag{4.91}$$

这样，我们面对的不再是一个最小化的 l_1 范数问题，而是一个具有经典线性规划结构的新问题。线性规划的目标函数是线性函数，约束也是线性约束。将 l_1 范数约束问题变成普通线性规划问题，接下来运用最优化中的单纯形法或对偶法就能解决这个问题，这就是基追踪算法的主要思想。

2. 迭代重加权最小二乘算法

接下来介绍求解 l_1 范数约束问题的另外一种方法，迭代重加权最小二乘[37](iterative-reweighed-least-squares，IRLS)算法，该算法主要解决的仍然是最小化 l_1 范数问题，目标函数仍为

$$\min_{\boldsymbol{x}}\|\boldsymbol{x}\|_1\ \ \text{s.t.}\ \ \boldsymbol{Ax}=\boldsymbol{b} \tag{4.92}$$

接下来，对目标函数进行变形。IRLS 算法用拉格朗日乘子，将约束和目标函数组合到一起，变成以下问题：

$$\min_{\boldsymbol{x}}\lambda\|\boldsymbol{x}\|_1+\|\boldsymbol{b}-\boldsymbol{Ax}\|_2^2 \tag{4.93}$$

令 $\boldsymbol{X}=\mathrm{diag}(|\boldsymbol{x}|)$，$l_1$ 范数约束问题则为 $\|\boldsymbol{x}\|_1\equiv\boldsymbol{x}^{\mathrm{T}}\boldsymbol{X}^{-1}\boldsymbol{x}$，上式可转化成函数：

$$\min_{\boldsymbol{x}}\ \ \lambda\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}+\|\boldsymbol{b}-\boldsymbol{Ax}\|_2^2 \tag{4.94}$$

接下来对函数求偏导数，并令其等于零，有

$$\frac{\partial(\lambda\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}+\|\boldsymbol{b}-\boldsymbol{Ax}\|_2^2)}{\partial\boldsymbol{x}}=0 \tag{4.95}$$

将 $\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}$ 展开，得到：

$$\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}=\frac{1}{\boldsymbol{X}^{(J-1)}(1,1)}\boldsymbol{x}(1)^2+\frac{1}{\boldsymbol{X}^{(J-1)}(2,2)}\boldsymbol{x}(2)^2+\cdots+\frac{1}{\boldsymbol{X}^{(J-1)}(m,m)}\boldsymbol{x}(m)^2 \tag{4.96}$$

将式(4.96)代入式(4.95)，可得

$$\frac{\partial\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}}{\partial\boldsymbol{x}}=\begin{bmatrix}2\dfrac{1}{\boldsymbol{X}^{(J-1)}(1,1)}\boldsymbol{x}(1)\\ 2\dfrac{1}{\boldsymbol{X}^{(J-1)}(2,2)}\boldsymbol{x}(2)\\ \vdots\\ 2\dfrac{1}{\boldsymbol{X}^{(J-1)}(m,m)}\boldsymbol{x}(m)\end{bmatrix} \tag{4.97}$$

对式(4.97)进行简化，改写为

$$\frac{\partial \boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}}{\partial \boldsymbol{x}} = 2(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x} \tag{4.98}$$

已知 $\dfrac{\partial \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_2^2}{\partial \boldsymbol{x}} = 2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})$，结合式(4.95)和式(4.98)，可得到：

$$\frac{\partial(\lambda \boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x} + \|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2)}{\partial \boldsymbol{x}} = 2\lambda(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x} + 2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) = 0 \tag{4.99}$$

进一步化简得到：

$$(\lambda(\boldsymbol{X}^{(J-1)})^{-1} + \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} \tag{4.100}$$

将前一次迭代的 $\boldsymbol{x}$ 对角化，得到 $\boldsymbol{X}^{(J-1)}$，然后通过式(4.89)计算求出稀疏表示系数 $\boldsymbol{x}$，这就是 IRLS 算法关于求解 l_1 范数约束的主要思路。

用 IRLS 算法解决 l_p 范数约束问题，目标函数变为

$$\boldsymbol{L}(\boldsymbol{x}) = \|\boldsymbol{x}\|_p^p + \lambda^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) \tag{4.101}$$

令 $2-2q=p$，IRLS 算法解决 l_p 范数约束问题思路基本与解决 l_1 范数约束问题相同，假设经过 J 次迭代，令 $\boldsymbol{X}^{(J-1)} = \mathrm{diag}(|\boldsymbol{x}^{(J-1)}|^q)$，对 $|\boldsymbol{x}^{(J-1)}|^q$ 对角化处理，则有

$$\begin{aligned}\boldsymbol{X}^{(J-1)} &= \mathrm{diag}(|\boldsymbol{x}^{(J-1)}|^q) \\ &= \begin{bmatrix} \left|\boldsymbol{x}^{(J-1)}(1)\right|^q & & & 0 \\ & \left|\boldsymbol{x}^{(J-1)}(2)\right|^q & & \\ & & \ddots & \\ 0 & & & |\boldsymbol{x}^{(J-1)}(m)|^q \end{bmatrix}\end{aligned} \tag{4.102}$$

则 $(\boldsymbol{X}^{(J-1)})^{-1}$ 为

$$(\boldsymbol{X}^{(J-1)})^{-1} = \begin{bmatrix} \left|\boldsymbol{x}^{(J-1)}(1)\right|^{-q} & & & 0 \\ & \left|\boldsymbol{x}^{(J-1)}(2)\right|^{-q} & & \\ & & \ddots & \\ 0 & & & \left|\boldsymbol{x}^{(J-1)}(m)\right|^{-q} \end{bmatrix} \tag{4.103}$$

那么，$\|(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}\|_2^2$ 可写为

$$\left\|(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}\right\|_2^2=\left\|\begin{bmatrix}\left|\boldsymbol{x}^{(J-1)}(1)\right|^{-q} & & & 0\\ & \left|\boldsymbol{x}^{(J-1)}(2)\right|^{-q} & & \\ & & \ddots & \\ 0 & & & \left|\boldsymbol{x}^{(J-1)}(m)\right|^{-q}\end{bmatrix}\begin{bmatrix}\left|\boldsymbol{x}^{(J-1)}(1)\right|\\ \left|\boldsymbol{x}^{(J-1)}(2)\right|\\ \vdots\\ \left|\boldsymbol{x}^{(J-1)}(m)\right|\end{bmatrix}\right\|_2^2 \tag{4.104}$$

$$=\left\|\begin{bmatrix}\left|\boldsymbol{x}^{(J-1)}(1)\right|^{1-q}\\ \left|\boldsymbol{x}^{(J-1)}(2)\right|^{1-q}\\ \vdots\\ \left|\boldsymbol{x}^{(J-1)}(m)\right|^{1-q}\end{bmatrix}\right\|_2^2$$

由于 $2-2q=p$，结合式(4.104)，故而有

$$\left\|(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}\right\|_2^2\approx\|\boldsymbol{x}\|_{2-2q}^{2-2q}=\|\boldsymbol{x}\|_p^p \tag{4.105}$$

所以式(4.104)中 $(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}$ 的二范数和 $\boldsymbol{x}$ 的 p 范数是等价的，即

$$\|\boldsymbol{x}\|_p^p=\left\|(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}\right\|_2^2 \tag{4.106}$$

因此式(4.101)可改写为

$$\boldsymbol{L}(\boldsymbol{x})=\left\|(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\right\|_2^2+\lambda^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}) \tag{4.107}$$

将 $\left\|(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\right\|_2^2$ 展开，有

$$\begin{aligned}\left\|(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\right\|_2^2&=((\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x})^{\mathrm{T}}((\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x})\\&=\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{+\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\\&=\boldsymbol{x}^{\mathrm{T}}((\boldsymbol{X}^{(J-1)})^{+})^2\boldsymbol{x}\\&=\left\|(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\right\|_2^2\end{aligned} \tag{4.108}$$

将上式继续展开，得到

$$\begin{aligned}\left\|(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\right\|_2^2=&((\boldsymbol{X}^{(J-1)})^{+}(1,1))^2\boldsymbol{x}(1)^2+((\boldsymbol{X}^{(J-1)})^{+}(2,2))^2\boldsymbol{x}(2)^2\\&+\cdots+((\boldsymbol{X}^{(J-1)})^{+}(m,m))^2\boldsymbol{x}(m)^2\end{aligned} \tag{4.109}$$

对式(4.109)求偏导数：

$$\frac{\partial\left\|(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\right\|_2^2}{\partial\boldsymbol{x}}=\begin{bmatrix}2((\boldsymbol{X}^{(J-1)})^{+}(1,1))^2\boldsymbol{x}(1)\\ 2((\boldsymbol{X}^{(J-1)})^{+}(2,2))^2\boldsymbol{x}(2)\\ \vdots\\ 2((\boldsymbol{X}^{(J-1)})^{+}(m,m))^2\boldsymbol{x}(m)\end{bmatrix} \tag{4.110}$$

整理成简单形式为

$$\frac{\partial\left\|(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\right\|_2^2}{\partial\boldsymbol{x}}=2((\boldsymbol{X}^{(J-1)})^{+})^2\boldsymbol{x} \tag{4.111}$$

接下来，求偏导数 $\frac{\partial L(\boldsymbol{x})}{\partial\boldsymbol{x}}$，并令其等于零。由于 $\boldsymbol{L}(\boldsymbol{x})=\left\|(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\right\|_2^2+\lambda^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x})$，因而：

$$\frac{\partial L(\boldsymbol{x})}{\partial\boldsymbol{x}}=2((\boldsymbol{X}^{(J-1)})^{+})^2\boldsymbol{x}-\boldsymbol{A}^{\mathrm{T}}\lambda \tag{4.112}$$

令偏导数 $\frac{\partial L(\boldsymbol{x})}{\partial\boldsymbol{x}}=\boldsymbol{0}$，有

$$\frac{\partial L(\boldsymbol{x})}{\partial\boldsymbol{x}}=\boldsymbol{0}=2((\boldsymbol{X}^{(J-1)})^{+})^2\boldsymbol{x}-\boldsymbol{A}^{\mathrm{T}}\lambda \tag{4.113}$$

整理得

$$\boldsymbol{x}^{(J)}=0.5(\boldsymbol{X}^{(J-1)})^2\boldsymbol{A}^{\mathrm{T}}\lambda \tag{4.114}$$

将式(4.114)代入 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ 中，有

$$0.5\boldsymbol{A}(\boldsymbol{X}^{(J-1)})^2\boldsymbol{A}^{\mathrm{T}}\lambda=\boldsymbol{b} \tag{4.115}$$

由上式可解得

$$\lambda=2(\boldsymbol{A}(\boldsymbol{X}^{(J-1)})^2\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b} \tag{4.116}$$

则迭代方程可写为

$$\boldsymbol{x}_k=\boldsymbol{X}_{k-1}^2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{X}_{k-1}^2\boldsymbol{A}^{\mathrm{T}})^{+}\boldsymbol{b} \tag{4.117}$$

在这种情况下，根据上述推导过程可知，IRLS 实际上是通过迭代方程不断地去计算、去迭代，来对 l_p 约束问题进行优化，在得到 $\boldsymbol{A}$、$\boldsymbol{b}$ 之后，将其代入迭代方程去求解使得 $\|\boldsymbol{x}\|_p^p$ 最小。IRLS 算法是一个极其有趣的算法，它最主要的思想就是用迭代方程的形式去求 l_1 范数或者是 l_p 范数的一个最小化问题。

IRLS 算法的具体流程如下。

任务：找到近似于该条件下的稀疏表示系数 $\boldsymbol{x}$：$\min\limits_{\boldsymbol{x}}\|\boldsymbol{x}\|_p^p$ s.t. $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$。

初始化：初始化 $k=0$，并设置初始近似 $\boldsymbol{x}_0=\boldsymbol{1}$，初始权重矩阵 $\boldsymbol{X}_0=\boldsymbol{I}$。

主迭代：将 k 增加 1，并应用以下步骤。

①利用如下公式对稀疏表示系数 $\boldsymbol{x}_k$ 进行更新：

$$\boldsymbol{x}_k=\boldsymbol{X}_{k-1}^2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{X}_{k-1}^2\boldsymbol{A}^{\mathrm{T}})^{+}\boldsymbol{b}$$

②权重更新：利用步骤①生成的 $\boldsymbol{x}_k$ 更新对角权重矩阵 $\boldsymbol{X}_k$：$\boldsymbol{X}_k(i,j)=\left|\boldsymbol{x}_k(j)\right|^{1-p/2}$。

③停止条件：如果达到停止条件，例如，$\|\boldsymbol{x}_k-\boldsymbol{x}_{k-1}\|_2$ 小于某个预定阈值，则停止。否则，进行下一次迭代。

输出：稀疏表示系数是 $\boldsymbol{x}_k$ 。

3. 角度回归算法

最小角度回归(least angle regression，LARS)算法，这类算法极其精妙。最小角度回归的思想是对 $\boldsymbol{b}$，也就是目标信号进行中心化，中心化以后不断调整 $\boldsymbol{x}$，也就是调整稀疏表示系数的大小，然后一直到它跟另外一个原子之间平行，接下来将该平行原子计入入选原子中。具体操作如图 4.6 所示。

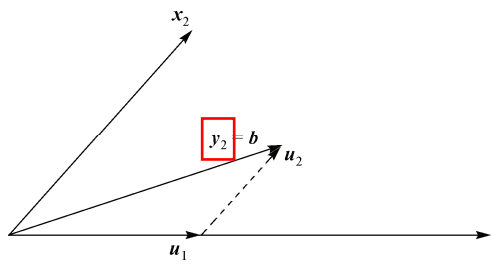

图 4.6　LARS 算法具体操作流程示例图

假设 $\bar{y}_2$ 为信号 $\boldsymbol{b}$，$\bar{y}_2$ 首先在所有原子里面寻找与其距离最近的原子，我们之前的方法是用信号 $\boldsymbol{b}$ 与最近原子间进行投影得到投影长度，LARS 算法虽然也是在原子上进行投影，但是其投影长度是不断变化的，新的残差会随着投影长度的变化而变化。假设残差 $\bar{y}_2$ 在原子 $\boldsymbol{x}_1$ 上进行投影，投影长度 u_1 不断变化，使新的残差长度 u_2 随着投影长度的变化而变化。在变化的过程中，当 u_2 与某个原子平行时，假设该原子为 $\boldsymbol{x}_2$，则将原子 $\boldsymbol{x}_2$ 纳入入选原子中。接下来用入选的两原子 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 进行稀疏表示，且下一次的变化方向为这两个原子的角平分线方向。同样，再次找到与此次投影后的残差平行的原子 $\boldsymbol{x}_3$，则第三次行走的方向为 $\boldsymbol{x}_1,\boldsymbol{x}_2,\boldsymbol{x}_3$ 这三个原子的角平分线，然后再次迭代残差找到与新的残差平行的原子。按照上述方法逐次迭代，直到残差小于预先设定的阈值，迭代终止。最小角度回归与前面介绍的方法的不同之处是：之前介绍的方法是一次投影，且残差与投影之间是一个正交的关系，两者垂直；而 LARS 则不是一次投影，它先找到原子 $\boldsymbol{x}_j$，然后按原子的方向做投影，直到找到与 u_2 平行的原子为止，然后将该原子纳入入选原子中，下次行走路线则为入选原子的角平分线，依次迭代往下进行。

LARS 算法的具体操作流程如下。

(1) 将预测变量标准化，使其具有零均值和单位范数。

设定初始残差：$\boldsymbol{r}^{(0)}=\boldsymbol{b}-\bar{\boldsymbol{b}}$，令 $x_1=x_2=\cdots=x_m=0$ 。

(2) 寻找残差 $\boldsymbol{r}$ 对应的入选原子 $\boldsymbol{A}(j)$ 。

(3)将 $\boldsymbol{x}_j$ 从原点逐步向其平方最小的方向移动，系数等于 $\left\langle \boldsymbol{A}(j), \boldsymbol{r}^{(0)} \right\rangle$，直到找到另外一个与当前残差相匹配的原子 $\boldsymbol{A}(k)$，并将其纳入入选原子中。

(4)①以 $\boldsymbol{A}(j)$ 和 $\boldsymbol{A}(k)$ 角平分线的方向为此次迭代的方向出发逐步移动，直到找到其他与当前残差平行的原子，并将其纳入入选原子。

$$[\boldsymbol{A}(j), \boldsymbol{A}(k)] \begin{bmatrix} x_j \\ x_k \end{bmatrix} = \boldsymbol{r}^{(1)} \tag{4.118}$$

②$[s_j \boldsymbol{A}(j), s_k \boldsymbol{A}(k)]$ 为角平分线方向，其中，

$$s_j = \operatorname{sign}\left\langle \boldsymbol{r}^{(1)}, \boldsymbol{A}(j) \right\rangle \tag{4.119}$$

(5)继续迭代，直到 m 个原子全部入选为止。下面推导角平分线的计算公式。假设：

$$\tilde{\boldsymbol{A}}^{(J)} = [s_1 \boldsymbol{A}(\Lambda^{(1)}), s_2 \boldsymbol{A}(\Lambda^{(2)}), \cdots, s_m \boldsymbol{A}(\Lambda^{(m)})] \tag{4.120}$$

其中，$s_1 \boldsymbol{A}(\Lambda^{(1)})$ 为首次入选的原子，$\tilde{\boldsymbol{A}}^{(J)}$ 为所有入选的原子，s_1 为校正原子方向的符号，以使得原子尽量在同一方向上。假设 $\boldsymbol{u}$ 为 $\tilde{\boldsymbol{A}}^{(J)}$ 中入选原子的角平分线：

$$\boldsymbol{u} = \tilde{\boldsymbol{A}}^{(J)} \boldsymbol{\omega} \tag{4.121}$$

角平分线与所有向量内积都相等，且所有原子在角平分线上的投影长度相等，故而：

$$\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \boldsymbol{u} = \tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)} \boldsymbol{\omega} = z \cdot \mathbf{1} \tag{4.122}$$

其中，z 为投影长度，$\mathbf{1}$ 为全 1 的向量，$\tilde{\boldsymbol{A}}^{(J)}$ 为所有入选的原子。有

$$\boldsymbol{\omega} = z(\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)})^{-1} \mathbf{1} \tag{4.123}$$

将式(4.123)代入式(4.121)可得

$$\boldsymbol{u} = z\tilde{\boldsymbol{A}}^{(J)} (\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)})^{-1} \boldsymbol{1} \tag{4.124}$$

因为 $\boldsymbol{u}^{\mathrm{T}} \boldsymbol{u} = 1$，有

$$z^2 (\tilde{\boldsymbol{A}}^{(J)} (\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)})^{-1} \mathbf{1})^{\mathrm{T}} (\tilde{\boldsymbol{A}}^{(J)} (\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)})^{-1} \mathbf{1}) = 1 \tag{4.125}$$

故

$$z^2 \mathbf{1}^{\mathrm{T}} ((\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)})^{-1})^{\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)} (\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)})^{-1} \mathbf{1} = 1 \tag{4.126}$$

则

$$z = (\mathbf{1}^{\mathrm{T}} ((\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)})^{-1})^{\mathrm{T}} \mathbf{1})^{-\frac{1}{2}} \tag{4.127}$$

将求得的 z 回代，得到角平分线方向：

$$\boldsymbol{\omega} = (\mathbf{1}^{\mathrm{T}} ((\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)})^{-1})^{\mathrm{T}} \mathbf{1})^{-\frac{1}{2}} (\tilde{\boldsymbol{A}}^{(J)\mathrm{T}} \tilde{\boldsymbol{A}}^{(J)})^{-1} \mathbf{1} \tag{4.128}$$

4.2.4 迭代收缩算法

迭代收缩算法(iterative shrinkage algorithm)是一类算法，这一类算法和前文所讲算法不同之处在于，前文提到过的算法都是通过逼近向量来求解稀疏表示系数的，而迭代收缩算法则是通过函数直接进行稀疏化。

迭代收缩算法的最小化目标函数为

$$f(\boldsymbol{x})=\lambda\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x})+\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2 \tag{4.129}$$

函数 $\rho(\boldsymbol{x})$ 对向量 $\boldsymbol{x}$ 进行运算。例如，对于 $\rho(\boldsymbol{x})=|\boldsymbol{x}|^p$，可以得到 $\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x})=\|\boldsymbol{x}\|_p^p$，这使我们可以自由选择任何合适的 $\rho(\boldsymbol{x})$。这种函数的最小化模型可以用各种经典的迭代优化算法来处理，从最速下降法和共轭梯度法再到更复杂的内点算法，都可以用来求解这种最小化模型。前面介绍过的 IRLS 算法、OMP 算法和 LARS 算法的性能有待提高，因此一部分人提出使用迭代收缩算法对算法性能进行提高。

迭代收缩算法扩展了经典的 Donoho-Johnstone 收缩去噪方法。尽管这些算法的结构简单，但在上述最小化目标函数方面显示出非常好的效果。近几年的深入理论分析证明了这些方法的收敛性，保证了对于凸函数的解是全局最小的，并研究了这些算法的收敛速度。

由于思考角度不同，产生的迭代收缩算法的具体实现方法也不同如可分离代理算法、基于迭代重加权最小二乘的迭代收缩算法、平行坐标下降算法等。下面通过几个迭代收缩算法来简单介绍一下这类算法的实现方法。

1. 可分离代理算法

可分离代理(separable surrogate functionals，SSF)算法首先令矩阵 $\boldsymbol{A}=[\boldsymbol{\Psi},\boldsymbol{\Phi}]$，通过此操作将矩阵 $\boldsymbol{A}$ 写成两个矩阵的形式，假定矩阵 $\boldsymbol{\Psi}$ 和 $\boldsymbol{\Phi}$ 正交，SSF 算法的模型为

$$f(\boldsymbol{x})=\lambda\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x})+\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2=\lambda\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x})+\frac{1}{2}\left\|\boldsymbol{b}-[\boldsymbol{\Psi}\quad\boldsymbol{\Phi}]\begin{bmatrix}\boldsymbol{x}_{\Psi}\\ \boldsymbol{x}_{\Phi}\end{bmatrix}\right\|_2^2 \tag{4.130}$$

因为存在两个变量 $\boldsymbol{x}_{\Phi}$ 和 $\boldsymbol{x}_{\Psi}$，所以要交替优化，定义向量 $\tilde{\boldsymbol{b}}=\boldsymbol{b}-\boldsymbol{\Phi}\boldsymbol{x}_{\Phi}$。新函数模式为

$$f(\boldsymbol{x}_{\Psi},\boldsymbol{x}_{\Phi}^k)=\frac{1}{2}\left\|\tilde{\boldsymbol{b}}-\boldsymbol{\Psi}\boldsymbol{x}_{\Psi}\right\|_2^2+\lambda\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x}_{\Psi}) \tag{4.131}$$

首先对 $\boldsymbol{\Psi}$ 进行优化，使 $\boldsymbol{b}-\boldsymbol{\Phi}\boldsymbol{x}_{\Phi}$ 尽量逼近 $\boldsymbol{\Psi}\boldsymbol{x}_{\Psi}$，对 $f(\boldsymbol{x}_{\Psi},\boldsymbol{x}_{\Phi}^k)$ 求偏导数，且令偏导数等于零，有

$$\boldsymbol{x}_{\Psi}^{(J+1)}=\mathbf{S}_{\rho,\lambda}(\boldsymbol{\Psi}^{\mathrm{T}}\tilde{\boldsymbol{b}})=S_{\rho,\lambda}(\boldsymbol{\Psi}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{\Phi}\boldsymbol{x}_{\Phi}^{(J)})) \tag{4.132}$$

$\mathbf{S}_{\rho,\lambda}$ 为对表示系数进行稀疏化的操作，$\mathbf{S}_{\rho,\lambda}$ 的作用实际上是让稀疏表示系数中小系数置零。同样地，对 $\boldsymbol{\Phi}$ 进行优化，并用同样的方法将稀疏表示系数中的小系数置零。最后求得的稀疏表示系数就是将求得的两个稀疏表示系数结合起来，得到：

$$\begin{aligned}\boldsymbol{x}^{(J+1)}&=\begin{bmatrix}\boldsymbol{x}_{\Psi}^{(J+1)}\\ \boldsymbol{x}_{\Phi}^{(J+1)}\end{bmatrix}=\begin{bmatrix}\mathbf{S}_{\rho,\lambda}(\boldsymbol{\Psi}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{\Phi}\boldsymbol{x}_{\Phi}^{(J)}))\\ \mathbf{S}_{\rho,\lambda}(\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{\Psi}\boldsymbol{x}_{\Psi}^{(J)}))\end{bmatrix}\\&=\mathbf{S}_{\rho,\lambda}\left(\begin{bmatrix}(\boldsymbol{\Psi}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(J)}+\boldsymbol{\Psi}\boldsymbol{x}_{\Psi}^{(J)}))\\ (\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(J)}+\boldsymbol{\Phi}\boldsymbol{x}_{\Phi}^{(J)}))\end{bmatrix}\right)\\&=\mathbf{S}_{\rho,\lambda}\left(\begin{bmatrix}(\boldsymbol{\Psi}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(J)})+\boldsymbol{x}_{\Psi}^{(J)})\\ (\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(J)})+\boldsymbol{x}_{\Phi}^{(J)})\end{bmatrix}\right)\\&=\mathbf{S}_{\rho,\lambda}(\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(J)})+\boldsymbol{x}^{(J)})\end{aligned} \tag{4.133}$$

SSF 算法与其他算法的不同之处在于，前面所学的算法基本上都是运用迭代公式和推导的方法求解稀疏表示系数，SSF 算法则是在推导步骤做完之后，人为地进行筛选稀疏表示系数，将求得的稀疏表示系数中的小系数人为置零，这就是 SSF 算法的核心思想。

SSF 算法的具体操作流程如下(基于线搜索的 SSF 迭代收缩算法)。

任务：找到满足该条件的最小化的 $\boldsymbol{x}$ ： $f(\boldsymbol{x})=\lambda\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x})+\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2$。

初始化：初始化 $k=0$，并设置初始解 $\boldsymbol{x}_0=\mathbf{0}$，初始残差 $\boldsymbol{r}_0=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_k=\boldsymbol{b}$。

主迭代：将 k 增加 1，并进行以下步骤。

(1)反投影：计算 $\boldsymbol{e}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}_{k-1}=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_{k-1})$。

(2)收缩率：计算 $\boldsymbol{e}_s=\boldsymbol{S}_{\rho,\lambda}(\boldsymbol{x}_{k-1}+\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_{k-1}))=\boldsymbol{S}_{\rho,\lambda}(\boldsymbol{x}_{k-1}+\boldsymbol{e})$， $\boldsymbol{S}_{\rho,\lambda}$ 表示将小系数置零。

(3)线性搜索(可选)：选择 μ 以最小化实值函数 $f(\boldsymbol{x}_{k-1}+\mu(\boldsymbol{e}_s-\boldsymbol{x}_{k-1}))$。该步骤是为了指定搜索方式为线性搜索。

(4)更新稀疏表示系数：计算 $\boldsymbol{x}_k=\boldsymbol{x}_{k-1}+\mu(\boldsymbol{e}_s-\boldsymbol{x}_{k-1})$。

(5)更新残差：计算 $\boldsymbol{r}_k=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_k$。

(6)停止规则：如果 $\|\boldsymbol{x}_k-\boldsymbol{x}_{k-1}\|_2^2$ 比预先设定的阈值更小，停止迭代。否则，进行下一次迭代。

输出：结果是 $\boldsymbol{x}_k$。

---

2. 基于迭代重加权最小二乘的迭代收缩算法(IRLS-based shrinkage algorithm)

接下来要介绍的这个算法是在 IRLS 算法的基础上提出的一个新的算法，它将 IRLS 算法和迭代收缩算法结合起来，具体流程如下。

数学模型：

$$f(\boldsymbol{x})=\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2+\lambda\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x}) \tag{4.134}$$

令 $\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x})=\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}^{-1}(\boldsymbol{x})\boldsymbol{x}$，对上式进行转化：

$$f(\boldsymbol{x})=\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2+\frac{\lambda}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}^{-1}(\boldsymbol{x})\boldsymbol{x} \tag{4.135}$$

很明显，上式为二次型函数，这就是 IRLS 算法的特点，IRLS 算法将一个约束函数转化为一个二次型函数。接下来要做的就是对二次型函数进行求解。

根据 $\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x})=\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}^{-1}(\boldsymbol{x})\boldsymbol{x}$ 可得 $\boldsymbol{W}(\boldsymbol{x})$ 为一个对角阵：

$$\boldsymbol{W}[k,k]=0.5\boldsymbol{x}[k]^2/\rho(\boldsymbol{x}[k]) \tag{4.136}$$

则

$$\boldsymbol{W}^{-1}(\boldsymbol{x})=\begin{bmatrix}\dfrac{\rho(x[1])}{0.5x[1]^2} & & \\ & \dfrac{\rho(x[2])}{0.5x[2]^2} & \\ & & \dfrac{\rho(x[3])}{0.5x[3]^2}\end{bmatrix} \tag{4.137}$$

又 $\boldsymbol{x}=\begin{bmatrix}x[1]\\x[2]\\x[3]\end{bmatrix}$ 故而有

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}^{-1}(\boldsymbol{x})\boldsymbol{x}=[x[1]\quad x[2]\quad x[3]]\begin{bmatrix}\dfrac{\rho(x[1])}{0.5x[1]^2} & & \\ & \dfrac{\rho(x[2])}{0.5x[2]^2} & \\ & & \dfrac{\rho(x[3])}{0.5x[3]^2}\end{bmatrix}\begin{bmatrix}x[1]\\x[2]\\x[3]\end{bmatrix} \tag{4.138}$$

对 $f(\boldsymbol{x})$ 求偏导数并令偏导数等于 0：

$$\nabla f(\boldsymbol{x})=-\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x})+\lambda\boldsymbol{W}^{-1}(\boldsymbol{x})\boldsymbol{x}=0 \tag{4.139}$$

上式等价为

$$-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}+(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}-c\boldsymbol{I})\boldsymbol{x}+(\lambda\boldsymbol{W}^{-1}(\boldsymbol{x})+c\boldsymbol{I})\boldsymbol{x}=0$$

即

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}-(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}-c\boldsymbol{I})\boldsymbol{x}^{(J)}=(\lambda\boldsymbol{W}^{-1}(\boldsymbol{x}^{(J)})+c\boldsymbol{I})\boldsymbol{x}^{(J+1)} \tag{4.140}$$

对上式进行整理，最终迭代公式为

$$\boldsymbol{x}^{(J+1)}=\left(\frac{\lambda}{c}\boldsymbol{W}^{-1}(\boldsymbol{x}^{(J)})+\boldsymbol{I}\right)^{-1}\left(\frac{1}{c}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}-\frac{1}{c}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}-c\boldsymbol{I})\boldsymbol{x}^{(J)}\right) \tag{4.141}$$

令 $\boldsymbol{S}=\left(\frac{\lambda}{c}\boldsymbol{W}^{-1}(\boldsymbol{x}^{(J)})+\boldsymbol{I}\right)^{-1}$，则

$$\begin{aligned}\boldsymbol{x}^{(J+1)}&=\left(\frac{\lambda}{c}\boldsymbol{W}^{-1}(\boldsymbol{x}^{(J)})+\boldsymbol{I}\right)^{-1}\left(\frac{1}{c}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}-\frac{1}{c}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}-c\boldsymbol{I})\boldsymbol{x}^{(J)}\right)\\&=\boldsymbol{S}\cdot\left(\frac{1}{c}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}-\frac{1}{c}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}-c\boldsymbol{I})\boldsymbol{x}^{(J)}\right)\end{aligned} \tag{4.142}$$

对 $\boldsymbol{S}$ 进行整理，得

$$\boldsymbol{S}=\left(\frac{\lambda}{c}\boldsymbol{W}^{-1}(\boldsymbol{x}^{(J)})+\boldsymbol{I}\right)^{-1}=\left(\frac{\lambda}{c}\boldsymbol{I}+\boldsymbol{W}(\boldsymbol{x}^{(J)})\right)^{-1}\boldsymbol{W}(\boldsymbol{x}^{(J)}) \tag{4.143}$$

其中，$\boldsymbol{W}[k,k]=0.5\boldsymbol{x}[k]^2/\rho(x[k])$，将 $\boldsymbol{W}[k,k]=0.5\boldsymbol{x}[k]^2/\rho(\boldsymbol{x}[k])$ 代入上式中，有

$$\boldsymbol{S}=\frac{0.5\boldsymbol{x}^{(J)}[k]^2/\rho(\boldsymbol{x}^{(J)}[k])}{\frac{\lambda}{\mathrm{c}}+0.5\boldsymbol{x}^{(J)}[k]^2/\rho(\boldsymbol{x}^{(J)}[k])}=\frac{\boldsymbol{x}^{(J)}[k]^2}{\frac{2\lambda}{\mathrm{c}}\rho(\boldsymbol{x}^{(J)}[k])+\boldsymbol{x}^{(J)}[k]^2} \tag{4.144}$$

通过观察上式发现，当 $\boldsymbol{x}$ 较大时，$\boldsymbol{S}\approx 1$。当 $\boldsymbol{x}$ 较小而 ρ 比较大时，则 $\boldsymbol{S}$ 很小，相当于 $\boldsymbol{x}^{(J+1)}=\boldsymbol{S}\cdot\left(\frac{1}{c}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}-\frac{1}{c}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}-c\boldsymbol{I})\boldsymbol{x}^{(J)}\right)$ 乘了一个很小的数 $\boldsymbol{S}$。所以，参数 $\boldsymbol{S}$ 的作用是，保留稀疏表示系数中的大系数，并尽量使小系数置零。注意，这个算法不能像以前的方法那样用零初始化，因为这个向量是这个不动点迭代的稳定解。有趣的是，一旦解中的某个项变为零，就永远无法“恢复”，这意味着该算法可能会陷入局部极小值无法脱身。

基于 IRLS 的线性搜索迭代收缩算法的具体操作流程如下。

任务：找到满足该条件的最小化的 $\boldsymbol{x}$：$f(\boldsymbol{x})=\lambda\mathbf{1}^{\mathrm{T}}\rho(\boldsymbol{x})+\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2$。

初始化：初始化 $k=0$，并设置初始解 $\boldsymbol{x}_0=\boldsymbol{0}$，初始残差 $\boldsymbol{r}_0=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_k=\boldsymbol{b}$。

主迭代：将 k 增加 1，并应用以下步骤。

(1) 反投影：计算 $\boldsymbol{e}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}_{k-1}=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_{k-1})$。

(2) 收缩更新：通过 $\boldsymbol{S}[i,i]=\frac{\boldsymbol{x}_k[i]^2}{\frac{2\lambda}{c}\rho(\boldsymbol{x}_i[i])+\boldsymbol{x}_k[i]^2}$ 计算对角线矩阵 $\boldsymbol{S}$。

(3)收缩率：计算 $\boldsymbol{e}_s = \boldsymbol{S}\cdot\left(\frac{1}{c}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}-\frac{1}{c}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}-c\boldsymbol{I})\boldsymbol{x}_{k-1}\right)$

$$=\boldsymbol{S}\cdot\left(\frac{1}{c}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_{k-1})+\boldsymbol{I}\boldsymbol{x}_{k-1}\right)$$

$$=\boldsymbol{S}\cdot\left(\frac{1}{c}\boldsymbol{e}+\boldsymbol{I}\boldsymbol{x}_{k-1}\right)$$

(4)线性搜索(可选)：选择 μ 以最小化实值函数 $f(\boldsymbol{x}_{k-1}+\mu(\boldsymbol{e}_s-\boldsymbol{x}_{k-1}))$，该步骤是为了指定搜索方式为线性搜索。

(5)更新稀疏表示系数：计算 $\boldsymbol{x}_k=\boldsymbol{x}_{k-1}+\mu(\boldsymbol{e}_s-\boldsymbol{x}_{k-1})$。

(6)更新残差：计算 $\boldsymbol{r}_k=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_k$。

(7)停止规则：如果 $\|\boldsymbol{x}_k-\boldsymbol{x}_{k\text{-}1}\|_2^2$ 比预先设定的阈值更小，停止迭代。否则，进行下一次迭代。

输出：结果是 $\boldsymbol{x}_k$。

---

3. 平行坐标下降算法

平行坐标下降[37](the parallel coordinate descent，PCD)算法从一个简单的坐标下降算法开始，然后将一组这样的下降步骤合并为一个更简单的联合步骤，从而导出PCD迭代收缩算法。

数学模型：

$$f(\boldsymbol{x})=\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2+\lambda\boldsymbol{1}^{\mathrm{T}}\rho(\boldsymbol{x}) \tag{4.145}$$

一次更新一个原子，而其余原子保持不变。设 $\boldsymbol{x}_0$ 为初始值，$\boldsymbol{a}_k$ 为 $\boldsymbol{A}$ 中的第 k 个原子，$\rho(z)=|z|$，希望围绕其当前值 $\boldsymbol{x}_0[k]$ 更新第 k 个原子，可以得到一个一维函数的形式：

$$g(\mathrm{z})=\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_0-\boldsymbol{a}_{\mathrm{k}}(\mathrm{z}-\boldsymbol{x}_0[k])\|_2^2+\lambda\rho(\mathrm{z}) \tag{4.146}$$

令 $\tilde{\boldsymbol{b}}=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_0+\boldsymbol{x}_0[\mathrm{k}]\boldsymbol{a}_{\mathrm{k}}$ 代入上式，可得

$$\begin{aligned}g(z)&=\frac{1}{2}\|\tilde{\boldsymbol{b}}-\boldsymbol{a}_k z\|_2^2+\lambda\rho(z)=\frac{1}{2}\|\tilde{\boldsymbol{b}}\|_2^2-\tilde{\boldsymbol{b}}^{\mathrm{T}}\boldsymbol{a}_k\mathrm{z}+\frac{1}{2}\|\boldsymbol{a}_k\|_2^2\cdot z^2+\lambda\rho(z)\\&=\|\boldsymbol{a}_k\|_2^2\left(\frac{\|\tilde{\boldsymbol{b}}\|_2^2}{2\|\boldsymbol{a}_k\|_2^2}-\frac{\boldsymbol{a}_k^{\mathrm{T}}\tilde{\boldsymbol{b}}}{\|\boldsymbol{a}_k\|_2^2}\mathrm{z}+\frac{\mathrm{z}^2}{2}+\frac{\lambda}{\|\boldsymbol{a}_k\|_2^2}\rho(z)\right)\\&=\|\boldsymbol{a}_k\|_2^2\left(\frac{1}{2}\left(z-\frac{\boldsymbol{a}_k^{\mathrm{T}}\tilde{\boldsymbol{b}}}{\|\boldsymbol{a}_k\|_2^2}\right)^2+\frac{\lambda}{\|\boldsymbol{a}_k\|_2^2}\rho(z)\right)+\mathrm{Const}\end{aligned} \tag{4.147}$$

根据上式可得，要使 $g(z)$ 达到最小值，$\left(z-\dfrac{\boldsymbol{a}_k^{\mathrm{T}}\tilde{\boldsymbol{b}}}{\|\boldsymbol{a}_k\|_2^2}\right)=0$，所以 z 的最佳值为 $\dfrac{\boldsymbol{a}_k^{\mathrm{T}}\tilde{\boldsymbol{b}}}{\|\boldsymbol{a}_k\|_2^2}$。另外要满足约束 $\dfrac{\lambda}{\|\boldsymbol{a}_k\|_2^2}\rho(z)$，因此 z 的最佳值为

$$z_{\mathrm{opt}}=S_{\rho,\lambda/\|\boldsymbol{a}_k\|_2^2}\left(\frac{\boldsymbol{a}_k^{\mathrm{T}}\tilde{\boldsymbol{b}}}{\|\boldsymbol{a}_k\|_2^2}\right)=S_{\rho,\lambda/\|\boldsymbol{a}_k\|_2^2}\left(\frac{1}{\|\boldsymbol{a}_k\|_2^2}\boldsymbol{a}_k^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_0)+\boldsymbol{x}_0[k]\right) \tag{4.148}$$

由于函数具有以下性质：当最小化一个函数时，如果有几个下降方向，那么它们的任何非负组合也是下降方向。因此，考虑对上述方法继续进行改进，因为每个步骤都处理目标向量中的一个原子，所以可以将这个和写成以下形式：

$$\begin{aligned}\boldsymbol{v}_0&=\begin{bmatrix}S_{\rho,\lambda/\|\boldsymbol{a}_1\|_2^2}\left(\dfrac{1}{\|\boldsymbol{a}_1\|_2^2}\boldsymbol{a}_1^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_0)+\boldsymbol{x}_0[1]\right)\\ S_{\rho,\lambda/\|\boldsymbol{a}_k\|_2^2}\left(\dfrac{1}{\|\boldsymbol{a}_k\|_2^2}\boldsymbol{a}_k^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_0)+\boldsymbol{x}_0[k]\right)\\ S_{\rho,\lambda/\|\boldsymbol{a}_m\|_2^2}\left(\dfrac{1}{\|\boldsymbol{a}_m\|_2^2}\boldsymbol{a}_m^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_0)+\boldsymbol{x}_0[m]\right)\end{bmatrix}\\&=S_{\rho,\mathrm{diag}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\lambda}(\mathrm{diag}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_0)+\boldsymbol{x}_0)\end{aligned} \tag{4.149}$$

在上述过程中，虽然每个坐标方向都保证下降，但如果没有适当的缩放，它们的线性组合不一定下降。因此，考虑这个方向并沿着它执行一个线性搜索 (line search，LS)。

$$\begin{aligned}\boldsymbol{x}^{(J+1)}&=\boldsymbol{x}^{(J)}+\mu(\boldsymbol{v}^{(J)}-\boldsymbol{x}^{(J)})\\&=\boldsymbol{x}^{(J)}+\mu(S_{\rho,\mathrm{diag}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\lambda}(\mathrm{diag}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(J)})+\boldsymbol{x}^{(J)})-\boldsymbol{x}^{(J)})\end{aligned} \tag{4.150}$$

在上述过程中可以看出，与 SSF 算法相比，PCD 算法在以下两个方面有所不同：

① $\boldsymbol{A}$ 中原子的范数在加权反向投影误差方面起着重要作用，而之前的算法使用常数；

②该算法需要进行搜索以获得下降效果。

PCD 算法的具体操作流程如下(基于线搜索的 PCD 迭代收缩算法)。

任务：找到满足该条件的最小化的 $\boldsymbol{x}$ ： $f(x)=\lambda\mathbf{1}^{\mathrm{T}}\rho(x)+\dfrac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2$。

初始化：初始化 $k=0$，并设置初始解 $\boldsymbol{x}_0=\boldsymbol{0}$，初始残差 $\boldsymbol{r}_0=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_k=\boldsymbol{b}$。

准备权重：$\boldsymbol{W}=\mathrm{diag}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}$。

主迭代：将 k 增加 1，并应用以下步骤。

(1) 反投影：计算 $\boldsymbol{e}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}_{k-1}=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_{k-1})$。

(2) 收缩率：计算

$$\begin{aligned}\boldsymbol{e}_s&=\boldsymbol{x}_{k-1}+\mu(S_{\rho,\mathrm{diag}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\lambda}(\mathrm{diag}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_{k-1})+\boldsymbol{x}_{k-1})-\boldsymbol{x}_{k-1})\\&=\boldsymbol{x}_{k-1}+\mu(S_{\rho,\mathrm{diag}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\lambda}(\mathrm{diag}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{e}+\boldsymbol{x}_{k-1})-\boldsymbol{x}_{k-1})\end{aligned}$$

使用给定的阈值 $\lambda\boldsymbol{W}[i,i]$。

(3) 线性搜索(可选)：选择 μ 以最小化实值函数 $f(\boldsymbol{x}_{k-1}+\mu(\boldsymbol{e}_s-\boldsymbol{x}_{k-1}))$，该步骤是为了指定搜索方式为线性搜索。

(4) 更新稀疏表示系数：计算 $\boldsymbol{x}_k=\boldsymbol{x}_{k-1}+\mu(\boldsymbol{e}_s-\boldsymbol{x}_{k-1})$。

(5) 更新残差：计算 $\boldsymbol{r}_k=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_k$。

(6) 停止规则：如果 $\|\boldsymbol{x}_k-\boldsymbol{x}_{k-1}\|_2^2$ 比预先设定的阈值更小，停止迭代。否则，进行下一次迭代。

输出：结果是 $\boldsymbol{x}_k$。

4. 阶段式正交匹配追踪算法

与先前的几种迭代收缩算法不同，阶段式正交匹配追踪(stage-wise orthogonal-matching-pursuit，StOMP)算法的创建者是从 OMP 算法出发进行改进的。StOMP 算法的主要思想是将传统的 OMP 算法中一次只选入一个原子改成了一次选入多个原子。这样减少了相似度求解的次数，降低了算法复杂度。该算法特别适合于矩阵 $\boldsymbol{A}$ 是随机的情况，如在压缩感知中。尽管 StOMP 算法和其他几种迭代收缩方法存在一定的区别，但是他们的迭代收缩过程具有一般相似性。

StOMP 算法包括以下步骤。

(1) 初始化：$\boldsymbol{r}_0=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}_0=\boldsymbol{b}$，$\boldsymbol{x}_0=\boldsymbol{0}$，$\boldsymbol{I}_0=\boldsymbol{\varphi}$。

(2) 反向投影残差：$\boldsymbol{e}^{(J)}=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(J-1)})=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{r}^{(J)}$。

(3) 将与残差相似度高的多个原子筛选出来：$\boldsymbol{J}^{(J)}=\left\{k\,|\,1\leqslant k\leqslant m,\left|\boldsymbol{e}^{(J)}[k]\right|>T\right\}$。

(4) 将筛选出来的原子加入选用原子集合：$\boldsymbol{I}^{(J)}=\boldsymbol{I}^{(J-1)}\bigcup\boldsymbol{J}^{(J)}$。

(5) 受约束的最小二乘：$\arg\min\limits_{\boldsymbol{x}}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{P}\boldsymbol{x}\|_2^2$，其中，$\boldsymbol{P}$ 是从 $\boldsymbol{x}$ 中选择的非零 $\boldsymbol{I}^{(J)}$。

(6) 更新残差：$\boldsymbol{r}^{(J)}=\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}^{(J)}$。

在上述过程中，当阈值化步骤一次选择一个原子时，这正是 OMP 方法。对于在每个步骤中选择多个原子的 StOMP 算法，上述过程应反复多次(作者建议 10 次迭代，因为每次迭代可能会在认可度集合中添加许多原子)，从而得到近似满足线性方程 $\boldsymbol{b}\approx\boldsymbol{A}\boldsymbol{x}$ 的稀疏解。

StOMP 算法可以用于去噪，$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{e}^{(J)}$ 中的值可以解释为稀疏噪声矢量，其中噪声可以可靠的假设为零均值高斯白噪声。因此，接下来的阈值化步骤只不过是一种去噪算法。

5. 顺序子空间优化算法

对于一种算法性能的评价，其中一个标准就是算法的执行速度，因此，对算法进行加速是改进一种算法的重要途径之一。以上算法都可以通过多种方式进一步加速，如 PCD 算法中，执行线性搜索的思想与其他算法具有相关性，现在计算一个临时结果 x_{temp}，然后将解定义为 $\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\mu(\boldsymbol{x}_{\text{temp}}-\boldsymbol{x}_k)$，针对标量 μ 优化 $f(\boldsymbol{x}_{k+1})$。

更有效的加速方法是顺序子空间优化(sequential subspace optimization algorithm，SESOP)算法，其主要思想是将前面多步搜索到的最优解结合起来。原始的 SESOP 算法提出通过优化函数 f 在一个仿射子空间上获得下一次迭代 $\boldsymbol{x}_{k+1}$，该仿射子空间由 q 最近的步长集合 $\{\boldsymbol{x}_{k-i}-\boldsymbol{x}_{k-j-1}\}_{i=0}^{q-1}$ 和当前的梯度所张成。这个 $q+1$ 维优化问题可以用牛顿算法来解决，因为这个问题是在一个维度很低的空间上定义的。这个过程的主要计算负担是需要将这些方向乘以 $\boldsymbol{A}$，但是这 q 个乘法可以存储在以前的迭代中，从而使得 SESOP 加速算法几乎没有任何额外的计算负担。

4.3　本 章 小 结

本章中首先介绍了稀疏表示的概念和模型，接着主要介绍了各范数约束的求解方法。l_2 范数约束越小越好，且求解二范数约束问题用最小二乘法。l_0 范数问题是为了求稀疏表示系数中不为零的数的个数，重点介绍了几种经典的解 l_0 范数约束问题的方法。

首先是 MP 算法，由于匹配追踪法不精确，为了提高它的精确性，引入 OMP 算法。OMP 算法跟 MP 算法之间的差别就在于，OMP 算法将前面选过的所有原子去做一个最小二乘，即将最小二乘法与 MP 算法结合起来。之所以称之为 OMP 算法，是因为迭代过程中的残差和所有选定的原子都是正交的，也就是说每一步迭代的残差与所有选定原子张成的空间正交，所以称为 OMP 算法。但是 OMP 算法有两个算法复杂度比较高的环节，其一是找最佳匹配原子的过程，此过程需要每次迭代的残差与所有原子求内积，找到内积最大的原子作为此次的入选原子，这是一个算法复杂度比较高的环节；其二就是在求解稀疏表示系数过程中，逆矩阵的计算过程。为了解决这两个问题，接下来介绍了 LS-OMP 算法和 Cholesky 快速正交匹配追踪算法，LS-OMP 算法主要思想是用前一次的逆矩阵来求当前的逆矩阵。而 Cholesky 快速正交匹配追踪算法则是将 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$ 转化成一个上三角矩阵和一个下三角矩阵的乘积，接下来再使用高斯消去法，求解稀疏表示系数的问题。LS-OMP 算法和

Cholesky 快速正交匹配追踪算法解决的都是第二个问题——求逆问题，而 B-OMP 算法则是解决了寻找最佳匹配原子这个问题。

由于 l_0 范数约束为非凸函数，计算复杂。为了方便求解，将其近似替换为 l_1 范数，则凸函数转化为线性规划问题，并介绍了几种求解 l_1 范数的方法。首先是基础的基追踪算法，该算法主要是将 l_1 范数问题最终变为线性规划问题，就可以用单纯形法和对偶法等这些简单方法求解。紧接着介绍了 IRLS 算法和 LARS 算法，IRLS 算法的最主要的思想就是用迭代方程的形式去求 l_1 范数或者是 l_p 范数的一个最小化问题，而 LARS 算法的思想是对目标信号进行中心化，中心化以后不断调整稀疏表示系数的大小，然后一直到它跟另外一个原子之间平行，接下来将该平行原子纳入入选原子中。

最后，又介绍了一类算法——迭代收缩算法。这一类算法和前面算法的不同之处在于，迭代收缩算法是通过函数直接进行稀疏化。首先是 SSF 算法，与其他算法的不同之处在于它是在推导步骤做完之后，人为地进行筛选稀疏表示系数，将求得的稀疏表示系数中的小系数人为置零。紧接着介绍了基于迭代重加权最小二乘的迭代收缩算法和 PCD 算法。最后介绍了 StOMP 算法，这种算法加速了 OMP 算法，一次迭代入选多个原子，而更有效的加速算法是 SESOP 算法。

课 后 习 题

1．对于问题：$\min\limits_{\boldsymbol{x}}\|\boldsymbol{x}\|_2^2$ s.t. $\boldsymbol{b}=\boldsymbol{A}\boldsymbol{x}$，其最小二乘解为（　　）

A．$\hat{\boldsymbol{x}}_{\text{opt}}=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{b}$　B．$\hat{\boldsymbol{x}}_{\text{opt}}=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{b}$　C．$\hat{\boldsymbol{x}}_{\text{opt}}=\boldsymbol{A}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{b}$

2．对于匹配追踪算法，我们需要在每个步骤中比较（　　）

A．残差与字典原子之间的相似性

B．信号和字典原子之间的相似性

C．两个字典原子之间的相似性

3．以下哪个停止规则相对更好？（　　）

A．残差的能量　　B．迭代的次数　　C．稀疏性

4．在正交匹配追踪算法中，正交的是（　　）

A．不同迭代中的残差　　B．每次迭代中的残差和被选原子

C．原始信号和残差

5．在正交匹配追踪算法中，选择原子后，如何计算稀疏表示系数？（　　）

A．$\boldsymbol{x}^{(J)}=\boldsymbol{A}^{(J)\mathrm{T}}(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{b}$　　B．$\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$

C．$\boldsymbol{x}^{(J)}=\boldsymbol{A}^{(J)}(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{b}$　　D．$\boldsymbol{x}^{(J)}=\boldsymbol{A}^{(J)}(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})\boldsymbol{b}$

6．在正交匹配追踪算法中，每次迭代中选择几个原子？（　　）

A．1　　B．2　　C．3　　D．4

7．在正交匹配追踪算法中，以下哪个步骤是复杂步骤？（　　）

A．原子匹配：$\Lambda^{(J)}\in\arg\max_{\varpi\in\Omega}\left|\left\langle \boldsymbol{r}^{(J-1)},\boldsymbol{A}(\varpi)\right\rangle\right|$

B．计算稀疏表示系数：$\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$

C．计算残差：$\boldsymbol{r}^{(J)}=\boldsymbol{b}-\tilde{\boldsymbol{b}}^{(J)}$

D．计算逼近量：$\tilde{\boldsymbol{b}}^{(J)}=\boldsymbol{A}^{(J)}\boldsymbol{x}^{(J)}$

8．Cholesky-OMP 算法是将 $\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)}$ 分解成了（　　）

A．上三角矩阵和对称矩阵

B．对角矩阵和对称矩阵

C．上三角矩阵和下三角矩阵

9．在 Cholesky-OMP 算法中，下三角矩阵 $\boldsymbol{L}^{(J)}$ 仅仅更新（　　）

A．一列　　B．两列　　C．一行　　D．两行

10．在块正交匹配追踪算法中，以下步骤中仍为复杂步骤的是（　　）

A．原子匹配：$\Lambda^{(J)}\in\arg\max_{\varpi\in\Omega}\left|\left\langle \boldsymbol{r}^{(J-1)},\boldsymbol{A}(\varpi)\right\rangle\right|$

B．计算稀疏表示系数：$\boldsymbol{x}^{(J)}=(\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{A}^{(J)})^{-1}\boldsymbol{A}^{(J)\mathrm{T}}\boldsymbol{b}$

C．计算残差：$\boldsymbol{r}^{(J)}=\boldsymbol{b}-\tilde{\boldsymbol{b}}^{(J)}$

D．以上都不是

11．一个向量 $\boldsymbol{a}\in R^m$ 点乘以 $\boldsymbol{b}\in R^m$ 的算法复杂度为（　　）

A．$m\times m$　　B．$m+m$　　C．m

12．一个矩阵 $\boldsymbol{A}\in R^{m\times n}$ 乘以一个向量 $\boldsymbol{b}\in R^n$ 的算法复杂度为（　　）

A．$m\times m$　　B．$m\times n$　　C．m

13．以下哪些函数可以代替 ℓ_0 范数？（　　）

A．$\sum_j x_j/(\alpha+x_j)$　　B．$\sum_j x_j^2/(\alpha+x_j^2)$

C．$\sum_j(1-\exp(-\alpha x_j))$　　D．$\sum_j(1-\exp(-\alpha x_j^2))$

14．基追踪算法的主要思想是（　　）

A．将非线性问题转化为线性问题

B．将线性问题转化为非线性问题

15．当 $\boldsymbol{X}=\mathrm{diag}(|\boldsymbol{x}|)$，问题 $\min_{\boldsymbol{x}}\lambda\|\boldsymbol{x}\|_1+\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2$ 可以被转换成（　　）

A．$\min_{\boldsymbol{x}}\ \lambda\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})\boldsymbol{x}^{-1}+\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2$

B．$\min_{\boldsymbol{x}}\ \lambda\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})\boldsymbol{x}+\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2$

C．$\min_{\boldsymbol{x}}\ \lambda\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{X}^{(J-1)})^{-1}\boldsymbol{x}+\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2$

16. 当 $\boldsymbol{X}^{(J-1)}=\operatorname{diag}\left(\left|\boldsymbol{x}^{(J-1)}\right|^{q}\right)$，问题 $L(\boldsymbol{x})=\|\boldsymbol{x}\|_p^p+\lambda^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})$ 可以被转换成（　　）

A. $\boldsymbol{L}(\boldsymbol{x})=\left\|(\boldsymbol{X}^{(J-1)})^{+}\boldsymbol{x}\right\|_2^2+\lambda^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x})$

B. $\boldsymbol{L}(\boldsymbol{x})=\left\|(\boldsymbol{X}^{(J-1)})\boldsymbol{x}\right\|_2^2+\lambda^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x})$

C. $\boldsymbol{L}(\boldsymbol{x})=\left\|(\boldsymbol{X}^{(J-1)})\boldsymbol{x}^{-1}\right\|_2^2+\lambda^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x})$

17. 以下算法中，哪个在提供过程中错误的可能性更低？（　　）

A. OMP　　B. IRLS　　C. BP 线性程序

18. 在 LARS 算法中，所选的接近方向是（　　）

A. 所选原子的组合方向　B. 角平分线方向　C. 残差的方向

19. 普通的多变量优化方法是（　　）

A. 同时优化所有变量

B. 优化一个变量并删除其他变量

C. 交替的优化每个变量

20. 在迭代收缩算法中，每次迭代中的收缩过程指的是（　　）

A. 字典原子的数目　B. 大的稀疏表示系数　C. 小的稀疏表示系数

21. 如果我们想通过改进算法来加速 OMP 算法的过程，我们可以（　　）

A. 在每次迭代中选择更多的原子

B. 增加计算机的内存

C. 提高计算机的 CPU 基本频率

D. 增加 CPU 核数

22. 在 PCD 算法中，什么是用来加速算法收敛的？（　　）

A. 根据另一个字典仿射子空间

B. 根据最近步骤的结果仿射子空间

C. 根据其余原子仿射子空间

第 5 章　稀疏表示字典的求解

稀疏表示理论是近年来的研究热点问题之一，因其优秀的数据特征表示能力和对数据主要特征的自动提取，在很多领域显示出卓越的应用效果，近年来吸引越来越多的学者投入到稀疏表示理论的研究和应用中。稀疏表示理论中最为重要的角色——稀疏表示字典，它的好坏直接影响稀疏编码的性能，所以稀疏字典的学习是稀疏表示理论中不可或缺的组成部分。

5.1　稀疏表示字典的生成问题

在信号分析中，通常希望以更加简明的形式表示信号，以求更加鲜明地突显信号的本质。在某些数据处理和分析领域，如图像识别等，人们希望能将数据的维度尽可能地降低，在对数据进行降维的同时，得到最能刻画数据原始本质的特征信息，如自然图像的边缘、纹理等信息。稀疏表示就是通过一个字典，将原信号表示成少数几个字典原子的线性组合的形式。

稀疏表示的模型如式(5.1)所示：

$$\boldsymbol{B} = \boldsymbol{AX} \tag{5.1}$$

其中，$\boldsymbol{A}$ 是字典，$\boldsymbol{B}$ 是样本数据集，$\boldsymbol{X}$ 是稀疏表示系数。

如果式(5.1)中方程的个数小于未知数的个数，那么它就是欠定问题。假定 $\boldsymbol{B} = \boldsymbol{AX}$ 中，$\boldsymbol{B}$ 和 $\boldsymbol{A}$ 已知，在求解欠定问题的过程中，人们就需要对欠定方程式添加约束，缩小解的范围，进而求得系数 $\boldsymbol{X}$，这就是第 4 章中学过的稀疏表示系数的求解。

在本章中，主要学习稀疏表示字典的求解方法，也就是说，在求解稀疏表示的过程中，式(5.1)中的 $\boldsymbol{A}$ 和 $\boldsymbol{X}$ 都是未知的，仅有数据 $\boldsymbol{B}$，这里的 $\boldsymbol{X}$ 是要求解的稀疏表示系数，$\boldsymbol{A}$ 即为本章中所说的稀疏表示字典。即在 $\boldsymbol{A}$ 和 $\boldsymbol{X}$ 都是未知的情况下，需要训练能适应 $\boldsymbol{B}$ 这组样本数据的稀疏表示字典 $\boldsymbol{A}$。

目前，生成稀疏表示字典主要有两种方法：一种是使用数学公式生成，另外一种就是使用机器学习的方法训练生成。通常使用的数学公式生成的字典 $\boldsymbol{A}$ 都是标准正交基底，即 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}$，字典原子需要互相正交，例如，离散余弦基底和小波基底，一般这些算法都有表示系数的快速求解方法。利用机器学习所得到的稀疏表示字典，

原子之间并不要求正交且可过完备。非正交过完备的字典求解仍然是一个开放问题，不同的训练方法获得的字典性能差别很大。

利用机器学习进行字典训练和其他机器学习方法一样，包括训练阶段和测试阶段。训练阶段是对一组训练数据进行训练进而得到字典的过程；测试阶段是利用训练阶段已经得到的字典，完成稀疏表示系数求解的过程。衡量一个机器学习算法的性能包括学习力和推广力，如果把机器学习的过程比作人学习的过程，那么学习力是指这个人对材料学习的能力，是否学得好、学得会，是否能领悟到学习材料的精髓；推广力是指这个人对于问题能够举一反三，能够利用学习领悟到的知识去解决其他问题。

5.2 字典的训练模型

从矩阵分解角度看字典学习过程：给定样本数据集 $\boldsymbol{B}$，$\boldsymbol{B}$ 的每一列表示一个样本；字典学习的目标是把 $\boldsymbol{B}$ 矩阵分解为系数 $\boldsymbol{X}$ 和字典 $\boldsymbol{A}$：

$$\boldsymbol{B} \approx \boldsymbol{A}\boldsymbol{X} \tag{5.2}$$

其中，$\boldsymbol{B}=[\boldsymbol{b}_1 \quad \boldsymbol{b}_2 \quad \cdots \quad \boldsymbol{b}_M]$ 为样本集合，$\boldsymbol{b}_1,\boldsymbol{b}_2,\cdots,\boldsymbol{b}_M$ 为样本，M 为样本个数；$\boldsymbol{A}=[\boldsymbol{a}_1 \quad \boldsymbol{a}_2 \quad \cdots \quad \boldsymbol{a}_m]$ 为字典，$\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_m$ 为字典的原子，m 为原子个数；$\boldsymbol{X}=[\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_M]$，$\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_M$ 为字典原子的系数，$\boldsymbol{x}_i$ 为 $\boldsymbol{b}_i$ 的系数。

系数 $\boldsymbol{X}$ 和字典 $\boldsymbol{A}$ 的求解有下列三种形式。

(1) 第一种形式：

$$\boldsymbol{A},\boldsymbol{X}=\arg\min_{\boldsymbol{A},\boldsymbol{X}}\frac{1}{2}\left\|\boldsymbol{B}-\boldsymbol{A}\boldsymbol{X}\right\|_{\mathrm{F}}^2+\lambda\left\|\boldsymbol{x}_i\right\|_0 \tag{5.3}$$

其中，λ 为正则化系数，用来均衡稀疏性。在这种形式中，因为 l_0 范数难以求解，所以很多时候用 l_1 正则项代替近似。

(2) 第二种形式：

$$\begin{aligned}&\boldsymbol{A},\boldsymbol{X}=\arg\min_{\boldsymbol{A},\boldsymbol{X}}\{\left\|\boldsymbol{x}_i\right\|_0\}\\&\text{s.t. }\left\|\boldsymbol{B}-\boldsymbol{A}\boldsymbol{X}\right\|^2\leqslant\varepsilon\end{aligned} \tag{5.4}$$

其中，ε 是重构误差所允许的最大值。

(3) 第三种形式：

$$\begin{aligned}&\boldsymbol{A},\boldsymbol{X}=\arg\min_{\boldsymbol{A},\boldsymbol{X}}\left\|\boldsymbol{B}-\boldsymbol{A}\boldsymbol{X}\right\|^2\\&\text{s.t. }\left\|\boldsymbol{X}\right\|_0\leqslant L\end{aligned} \tag{5.5}$$

其中，L 是一个常数，作为稀疏度约束参数。

上述三种形式的目标函数中都存在两个位置变量 $\boldsymbol{A}$ 和 $\boldsymbol{X}$，它们相互之间是等价的。

如果 $\boldsymbol{A}$ 的列数小于 $\boldsymbol{B}$ 的行数且 $\boldsymbol{A}$ 的每一列不相关，则 $\boldsymbol{A}$ 相当于欠完备字典；如果 $\boldsymbol{A}$ 的列数大于 $\boldsymbol{B}$ 的行数且 $\boldsymbol{A}$ 能张成整个空间，则称 $\boldsymbol{A}$ 为过完备字典；如果 $\boldsymbol{A}$ 的列数刚好等于 $\boldsymbol{B}$ 的行数且 $\boldsymbol{A}$ 的每一列不相关，则称 $\boldsymbol{A}$ 为完备字典。

假设现在有一个 $M \times N$ 的过完备字典 $\boldsymbol{A}$，一个待表示的样本数据 $\boldsymbol{b}_i$（即要重建的图像），求系数 $\boldsymbol{x}_i$，使得 $\boldsymbol{b}_i = \boldsymbol{A}\boldsymbol{x}_i$，这里 $N > M$，因此该方程组为无穷多解，如式(5.6)所示。

$$\begin{bmatrix} b_{1i} \\ b_{2i} \\ b_{3i} \\ b_{4i} \\ b_{5i} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} \end{bmatrix} \times \begin{bmatrix} x_{1i} \\ x_{2i} \\ x_{3i} \\ x_{4i} \\ x_{5i} \\ x_{6i} \\ x_{7i} \\ x_{8i} \end{bmatrix} \tag{5.6}$$

其中，$M=5$，$N=8$，$\boldsymbol{b}_i = \boldsymbol{A}\boldsymbol{x}_i$，$\boldsymbol{b}_i = [b_{1i} \;\; b_{2i} \;\; b_{3i} \;\; b_{4i} \;\; b_{5i}]^{\mathrm{T}}$，$\boldsymbol{A} = [\boldsymbol{a}_1 \;\; \boldsymbol{a}_2 \;\; \cdots \;\; \boldsymbol{a}_8] = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} \end{bmatrix}$，$\boldsymbol{x}_i = [x_{1i} \;\; x_{2i} \;\; x_{3i} \;\; x_{4i} \;\; x_{5i} \;\; x_{6i} \;\; x_{7i} \;\; x_{8i}]^{\mathrm{T}}$。

由式(5.6)可知，该问题是一个不适定问题，即有多个满足此条件的解，无法判断哪个更加合适，于是需要对式(5.6)增加约束条件来得到最佳解。增加限制条件，要求 $\boldsymbol{x}_i$ 尽可能稀疏，也就是说，使 $\boldsymbol{x}_i$ 中的 0 尽可能多，非零数尽可能少，即 $\max(\|\boldsymbol{x}_i\|_0, 0)$ 尽可能小。

为了更加便于读者理解，列举了以下例子，如图 5.1 所示[38]。

现在给定一个任务，在字典中找出 10 张图像，用这 10 张图像的一个线性组合去尽可能地表示测试样本。如果是你的话，你会怎么选？你会选 10 张花草图像去表示一张人脸的图像吗？不会的。你会选 10 张人脸的图像尽可能地描述测试人脸图像，这也就是稀疏表示的过程。表示就是用字典中的元素(就是字典中的样本)的线性组合尽可能地描述(还原)测试样本。稀疏表示要用尽可能少的字典中的元素去描述测试样本。为什么要稀疏呢？为什么选用的字典中的样本要尽可能少呢？你可以