摘　要

矩阵重建是信号处理、人工智能和优化领域最近研究的热点。基于凸优化的矩阵重建问题衍生于近几年非常流行的压缩感知技术，主要分为矩阵填充和矩阵恢复问题，是一种重要的数据分析工具，在图像处理、计算机视觉、文本分析、推荐系统等方面已经找到重要的应用。本文经过仔细调研国内外的研究现状，对矩阵重建技术现有的算法和应用进行了全面的总结和分析，指出了现有技术的不足之处。现有的矩阵重建算法存在计算量大、速度慢、能够处理的矩阵规模小等问题，使得在很多场合这个工具不能充分发挥出其作用和优势；另外，在很多应用中，实际数据也往往不完全符合预定的模型。本文主要对矩阵重建中的一些计算问题进行了研究，针对其算法和实现提出了一整套解决办法。本文的贡献有以下几个方面：

（1）针对计算中的关键问题，设计和实现了一种收敛速度极快的算法，称为非精确的增广拉格朗日乘子法，它是拉格朗日乘子法的一个变种。此算法较以往主流算法提速明显，并且实现方便，占用内存少，适合于大规模矩阵的处理。同时，它具有极好的扩展性，可以应用于矩阵重建的一系列变种，对于实际应用有重要意义。

（2）对于矩阵重建中涉及到的子问题，即消耗大部分计算时间的奇异值分解，本文亦给出了一种改进的计算部分奇异值分解的算法，使其能够在矩阵重建算法中很好地工作。同时给出一套雅可比奇异值分解算法的改进的预处理和优化方案，该方案可以提高算法的收敛速度，并且适合在具有并行计算能力的硬件上实现。

（3）设计和实现了奇异值分解和矩阵恢复算法的GPU并行版本，这个实现可以进一步提升实际应用中对矩阵的处理速度。针对处理大规模矩阵的问题，参与设计和实现了矩阵恢复的集群版本，这个分布式实现使得能处理的矩阵规模有非常大的提高，不再受到单机内存的限制。

关键词：矩阵重建　矩阵填充　矩阵恢复　奇异值分解　压缩感知

Algorithms and Implementations of Matrix Reconstruction

Minming Chen (Computer Application)
Directed by Zhouchen Lin and Heung-Yeung Shum

Matrix reconstruction is a hot topic in signal processing, artificial intelligence and optimization. Convex optimization based matrix reconstruction problem comes from the compressive sensing technology, which is very popular these years. There are mainly two kinds of problems in matrix reconstruction: matrix completion and matrix recovery. Both of them are important data analysis tools and have found important applications in image processing, computer vision, text analysis, recommendation system, etc. The existing matrix reconstruction algorithms have problems in the convergence speed and computation load, and hence are not suitable for large scale data. Besides, in many applications, real data may not exactly match the existing models. This thesis focuses on the computing problems in matrix reconstruction and brings out solutions to the algorithms and implementations. The contributions of this thesis include:

(1) We propose a fast convergent algorithm, called the inexact augmented Lagrange multiplier, which is a variation of the Lagrange multiplier. It is significantly faster than the existing algorithms and requires less memory. It is also very easy to be adapted to the variations of the matrix reconstruction problems, which is important to the practical applications.

(2) Singular value decomposition (SVD) is the most expensive operation in algorithms of matrix reconstruction. We give an improved partial SVD algorithm. An improved preprocessing and optimization strategy for the Jacobi SVD algorithm is also given.

(3) We realize a parallel implementation of the SVD and matrix recovery algorithms on GPU, which leads to additional speed up. A cluster version is also designed and implemented to handle large scale data that cannot be fit into a single computer.

Keywords: Matrix Reconstruction, Matrix Completion, Matrix Recovery, Singular Value Decomposition, Compressive Sensing

第一章　引言

1.1　矩阵重建简介

随着计算机科学与技术在人们生活中的不断普及和发挥日益重要的作用，人工智能成为人们越来越关注的问题，它是计算机科学与技术的终极目标。总的来说，人工智能分为强人工智能与弱人工智能。强人工智能观点认为有可能制造出真正能推理和解决问题的智能机器，并且这些机器将被认为是有知觉的、有自我意识的。弱人工智能观点认为不可能制造出能真正地推理和解决问题的智能机器，这些机器只不过是智能的，但是并不真正拥有智能，也不会有自主意识。强人工智能的研究目前处于停滞不前的状态，而研究者却已大量制造出看起来像是智能的机器，取得了丰硕的理论和实践上的成果。这方面的研究先后经历了博弈时期、自然语言理解、知识工程等阶段，目前的研究热点是机器学习。

机器学习是人工智能最重要的分支之一，它主要关注于开发一些让计算机可以自动学习的技术。更具体说，机器学习是一种用于创建数据集分析程序的方法。机器学习跟统计学有着重要的关系，因为它也是研究数据分析，但是又不像统计学，机器学习关注的是计算实现的算法复杂度。很多推理问题属于无程序可循难度，所以部分的机器学习研究是在开发容易处理的近似算法。机器学习已经有了十分广泛的应用，如生物特征识别、搜索引擎、医学诊断、检测信用卡欺诈、证券市场分析、DNA序列测序、语音和手写识别、计算机视觉、战略游戏和机器人应用等。机器学习中现有的一些关键技术包括人工神经网络、决策树、线性判别分析、支持向量机等等。最近两年，基于凸优化的矩阵重建技术作为一种有效的数据分析工具，在机器学习中扮演着重要的角色。

讲到矩阵重建，不得不提到压缩感知，它们之间有着很大的关联。压缩感知已经存在了四十年，是一种利用稀疏或可压缩的信号进行信号重建的技术。它是解如下优化问题：

$$\begin{aligned} &\min \quad \|x\|_1, \\ &\text{subject to} \quad Ax = y. \end{aligned} \tag{1.1}$$

David Donoho、Emmanuel Candes、Justin Romberg和陶哲轩等人最近几年的工作（可以参见文章[46][7][13][12][10][11]）使这个领域有了长足的发展。他们的理论告诉我们：当感兴趣的信号是可压缩的或者可稀疏表示的，那么我们可以通过极少的采样精确地获得该信号。压缩感知中，信号的获取并不是直接测量信号本身，而是采样测量信号与一个感知矩阵相乘后的信号。而矩阵重建研究的则是在一个矩阵可压缩或可稀疏表示时，通过观测矩阵的某种线性或非线性运算后的元素，来精确地重建出该矩阵。通

常假定待恢复的矩阵是低秩的，通过以下优化问题求解

$$\begin{aligned} \min \quad & \operatorname{rank}(X), \\ \text{subject to} \quad & A(X) = b. \end{aligned} \tag{1.2}$$

本质上来讲，低秩和稀疏都说明信号可由更精简的形式来表达。在这种情况下，用很少的采样就可以完成信号的重建。

在很多的具体问题中，信号或者数据往往可以用矩阵来表示，使得对数据的理解、建模、处理和分析更为方便。然而这些数据经常面临缺失、损坏、受噪声污染等等问题。如何在各种情况下得到干净、准确、结构性良好的数据，就是矩阵重建所要解决的问题。大致来讲，矩阵重建分为矩阵填充（Matrix Completion）和矩阵恢复（Matrix Recovery）两大类。前者主要研究如何在数据不完整的情况下将缺失数据进行填充，后者主要研究在某些数据受到严重损坏的情况下恢复出准确的矩阵。无论是这个问题本身，还是其应用，都是最近的研究热点。最近的研究主要集中在矩阵重建在何种情况下可以准确地实现[8][9]、有没有快速的算法解决矩阵重建问题[6][42]和矩阵重建的应用[39][25][40]。

接下来我们具体介绍什么是矩阵填充，什么是矩阵恢复，以及它们各自有哪些应用。

1.2 矩阵填充

1.2.1 矩阵填充的定义

矩阵填充（Matrix Completion）考虑的是这样一个问题，对于某个矩阵，我们只能采样得到矩阵的一部分元素，其它一部分或者大部分元素由于各种原因丢失了或无法得到，如何将这些空缺的元素合理准确地填充。解决任何问题都是有一定条件的，矩阵填充也类似，不可能任何矩阵都可以填充。为解决这个问题我们往往假设这个矩阵是有信息冗余的，比如是低秩的，也就是说其数据分布在一个低维的线性子空间上。于是可以通过如下优化问题来实现矩阵填充：

$$\begin{aligned} \min \quad & \operatorname{rank}(X), \\ \text{subject to} \quad & X_{ij} = M_{ij}, (i,j) \in \Omega, \end{aligned} \tag{1.3}$$

其中Ω是已知元素下标的集合。这个模型的意义是说，将空缺的元素填充之后使得矩阵的结构尽可能好，即秩尽可能低。然而不幸的是，这是一个NP-Hard的问题，在理论和实践中，均只存在指数复杂度（相对于矩阵维数n）的算法。我们知道，一个矩阵的秩r与它的非零奇异值的个数相同。于是有一个选择是用矩阵的奇异值的和，即核范数，来近似地替代矩阵的秩：

$$\begin{aligned} \min \quad & \|X\|_*, \\ \text{subject to} \quad & X_{ij} = M_{ij}, (i,j) \in \Omega, \end{aligned} \tag{1.4}$$

其中$\|X\|_* = \sum_{k=1}^{n} \sigma_k(X)$，$\sigma_k(X)$表示矩阵的第$k$大的奇异值。核范数对应于矩阵奇异值组成的向量的1范数，而秩对应于该向量的0范数，它们之间的不同在于核范数是凸的而零范数并非数学意义上的范数。于是原来的问题就转化为一个凸优化问题，但它仍然是一个比较难计算的问题，可以考虑通过各种迭代法，尤其是梯度法进行求解。

1.2.2　矩阵填充的应用举例

矩阵填充的一个著名应用是Netflix推荐系统[4]。Netflix是世界上最大的在线影片租赁服务商，从2006年10月份开始举办Netflix大奖赛。它公开了大约一亿个1～5级的匿名电影评级，来自大约48万个客户对1.8万部电影的评价，所有个人信息都被从评级数据里面删除，所以数据集仅包含了影片名称、评价星级和评价日期，没有任何的文本评价内容。比赛要求参赛者预测Netflix客户分别喜欢什么影片，要把预测的效率相对原推荐系统Cinematch提高10%以上。这是一个典型的矩阵填充问题，即矩阵的每一行对应某个用户对电影的评级，每一列表示某电影在所有用户中的评级，但是每个用户只可能对一部分电影进行评价，所以我们可以通过矩阵填充得出用户对每部电影的喜好程度。

矩阵填充在图像和视频处理中也有重要应用，如视频去噪[25]。由于同一视频中各帧之间非常相似，同一帧中的不同图像区域之间也有很大的相似程度，我们很自然地可以假定由这些图像块排列而成的矩阵是低秩的，而根据某一像素值是否背离同一位置处所有像素的“均值”判定该点是否可靠，进而用矩阵填充来得到那些被噪声污染的像素。图1.1为[25]中的去噪效果。

图 1.1: 视频去噪
左图为噪声污染前的图片，中间为被噪声污染
后的图片，右图为通过矩阵填充去噪后的图片

1.3　矩阵恢复

1.3.1　矩阵恢复的定义

矩阵恢复，最早由John Wright等人提出[43]，又称为Robust PCA 或者稀疏与低秩

矩阵分解，是指当矩阵的某些元素被严重破坏后，自动识别出被破坏的元素，恢复出原矩阵。同样，假定原矩阵有非常良好的结构，即是低秩的；另外，假定只有很少一部分元素被严重破坏，即噪声是稀疏的但大小可以任意。于是矩阵恢复可用如下优化问题来描述：

$$\begin{aligned} \min \quad & \operatorname{rank}(A) + \lambda \|E\|_0, \\ \text{subject to} \quad & A + E = D, \end{aligned} \tag{1.5}$$

其中目标函数为矩阵A的秩以及噪声矩阵E的零范数，即E的非零元素的个数，λ表明噪声所占的权重。同样，这是一个NP-Hard问题，没有有效的算法。于是可以用矩阵的核范数近似秩，矩阵的1范数（本文定义为所有元素绝对值的和）来近似零范数，转化为如下问题：

$$\begin{aligned} \min \quad & \|A\|_* + \lambda \|E\|_1, \\ \text{subject to} \quad & A + E = D. \end{aligned} \tag{1.6}$$

这也是一个凸优化问题，但由于变量数目众多，所以如何有效地求解是一个很有意义的问题。

1.3.2 矩阵恢复的应用举例

[8]中，John Wright等人展示了矩阵恢复在背景建模、人脸图像处理等问题中的应用。其中，背景建模利用图片帧与帧之间的相似性，将每帧作为一列排列成一个矩阵，该矩阵理应具备相对较低的秩。于是利用稀疏与低秩矩阵分解技术可以将每帧中间相似的部分和特有的部分分开，即将背景与前景分离，见图1.2。同一个人脸的多幅图片排列而成的矩阵也理应具有较低的秩。利用这一性质，矩阵恢复可以将人脸中的阴影、高光或被损坏的部分去除，见图1.3。另外，[39]中Yigang Peng等人还将矩阵恢复技术应用到图片对齐。

图 1.2: **背景建模**
左图为视频中的某一帧，中图为通过矩阵恢复得到的背景，右图为通过矩阵恢复得到的前景

1.4 本文的贡献

前面简单介绍了矩阵填充和矩阵恢复的具体问题，将矩阵重建建模成凸优化问题

图 1.3: 人脸处理
左图为人脸图片，中图为通过矩阵恢复处理
后的人脸，右图为通过矩阵恢复得出的噪声

后，如何找到高效的算法是一个非常关键的问题。我们考虑的主要就是解决矩阵重建中的计算问题。本文的主要贡献体现在以下几个方面：

（1）针对计算中的关键问题，设计和实现了一种收敛速度极快的算法，称为非精确的增广拉格朗日乘子法，它是拉格朗日乘子法的一个变种。此算法较以往算法提速明显，并且具有极好的扩展性，可以应用于矩阵重建的一系列变种，对于实际应用有重要意义。

（2）对于矩阵重建中涉及到的子问题，即消耗大部分计算时间的奇异值分解，本文亦给出了一种改进的计算部分奇异值分解的算法，使其能够在矩阵重建算法中很好地工作。同时给出一套雅可比奇异值分解算法的改进的预处理和优化方案，使其速度有所提高。

（3）设计和实现了奇异值分解和矩阵恢复算法的GPU并行版本，使得处理速度得到进一步的提升。针对处理大规模矩阵的问题，参与设计和实现了矩阵恢复的集群版本，使得能处理的矩阵规模不再受到单机内存的限制。

1.5 本文的组织结构

本文总共由六章组成。

第一章对矩阵重建技术进行了大概的介绍，同时也对本文的主要内容做了简单的说明。

第二章对矩阵重建的国内外相关研究现状做了概括性的综述，指出相关的研究重点和存在的问题。

第三章对矩阵重建技术中的关键部分进行研究，设计和实现了一种基于增广拉格朗日乘子法的快速迭代算法，给出了一种改进的部分奇异值分解算法，并进行了仿真数据和实际数据上的比较。

第四章给出了奇异值分解和矩阵恢复的GPU并行算法的设计与实现。

第五章讨论和分析了矩阵重建的一些变种，给出在这些情况下的算法。

第六章对本文进行了全面的总结。

第二章　国内外研究综述

接上一章所提到的，矩阵重建主要分为两大类，即矩阵填充和矩阵恢复。矩阵填充主要解决在仅观察到一个矩阵的某一小部分数据时，填充那些未知或者缺失的数据。矩阵恢复则解决矩阵的某部分数据被严重破坏时，将数据中那些严重的噪声去除从而恢复出原矩阵。需要说明的是矩阵恢复又称之为Robust PCA或者稀疏与低秩矩阵分解[43]，其中的稀疏成分在大多数情况下可以理解为噪声，但在有些情况下也可能是一些十分有用的重要信息，它表明了每个对象的特有属性。国内外许多数学家和计算机科学家都对矩阵填充和矩阵恢复做了大量研究。目前的研究主要分为理论、算法和应用三个方面。关于理论，研究的主要是在怎样的情况下可以通过求解前面提到的凸优化问题来精确地重建矩阵，这方面的工作主要是一些基于概率模型的证明，如[9][8]。在算法方面，最近有很多一阶方法用以解决矩阵填充问题，见[6][32][42]，这些方法都是迭代地进行奇异值分解最终得到原问题的最优解。一般相似的方法也可以用于矩阵恢复，见[43]。矩阵重建的应用也越来越受到重视，如[25]中的视频去噪，[39]中的图片对齐，[40]中的推荐系统。下面，我们分别就矩阵填充和矩阵恢复在这三个方面的研究现状进行综述性的介绍。

2.1　矩阵填充的研究现状

2.1.1　矩阵填充的可行性

矩阵填充理论研究主要考虑的是矩阵填充的可行性，即究竟在什么情况下可以精确无误地把缺失数据填充完整。

首先，我们需要理解的是，一个杂乱无章的矩阵是不可能进行重建的，所以我们假定矩阵结构良好，如低秩。但要注意的是并非所有的低秩矩阵都可以精确地重建，如下图所示的矩阵，它的秩为1，但是仅在右上角处一个位置值为1，其他元素均为0，这样的矩阵在大多数情况下我们采样得到的会全是0，这样无论如何也没有办法填充出那个值为1的元素。

$$\begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

所以说我们不能指望通过采样可以重建出所有的低秩矩阵，而应当考虑有多大的概率

可以重建。如[9]中所分析的，如果一个秩为r的矩阵的奇异值分解为

$$M = \sum_{k=1}^{r} \sigma_k u_k v_k^*, \tag{2.1}$$

假定k个左奇异向量和k个右奇异向量都是均匀分布在所有由k个正交归一化向量的集合中，对于这样的矩阵，有很大的概率可以重建。

第二，并不是用所有的采样方式采样得到的矩阵均能够重建。例如，当矩阵的某一列完全没有被采样时，任何方法都不可能将这一列的元素准确地填充。于是，一般都假定矩阵的采样方式也是均匀采样，在这样的情况下来考虑矩阵精确重建的概率。

第三，采样的矩阵元素数目必须大于一定范围时，才有可能将矩阵进行精确填充。[9]中指出，在上述两个假定条件下，设n为矩阵维数，r为矩阵的秩，则存在常数C和c，使得当采样数目m满足

$$m \geq C n^{5/4} r \log n$$

时（1.4）的最优解正好为M的概率满足

$$p \geq 1 - c n^{-3}.$$

图2.1是[9]中对上述定理的数值模拟。其中m为采样的元素数目，n为矩阵维数，$dr = r * (2n - r)$表示秩为r的矩阵的自由度。白点表示在所有实验中百分之一百精确恢复，黑色表示不能恢复，灰色表示在实验中有时候可以恢复有时候不行。从中可以看出，当矩阵的秩越低，采样元素的数目越多，矩阵缺失元素越容易被精确地填充，这与我们的直觉相符。

除[9]外，Sahand Negahban等人[35]，Raghu Meka等人[33]以及[41][21][27]也在这方面做了重要的研究。

2.1.2 SVT算法

前面介绍了矩阵填充的理论上的可行性分析的研究现状，即什么情况下解凸优化问题（1.4）可以精确地重建出低秩矩阵。下一个至关重要的问题是解这个凸优化问题有没有好的算法。[6]中提出了一种简单的一阶方法，奇异值阈值（Singular Value Thresholding，简称SVT）算法，来求解矩阵填充问题。

矩阵填充问题可以写为：

$$\begin{aligned} \min \quad & \|X\|_*, \\ \text{subject to} \quad & P_\Omega(X) = P_\Omega(M), \end{aligned} \tag{2.2}$$

其中Ω表示所有的采样元素的坐标(i, j)的集合，$P_\Omega(X)$ 表示一种投影算子，它将矩阵在Ω以外的元素置0，Ω内部元素保持不变。

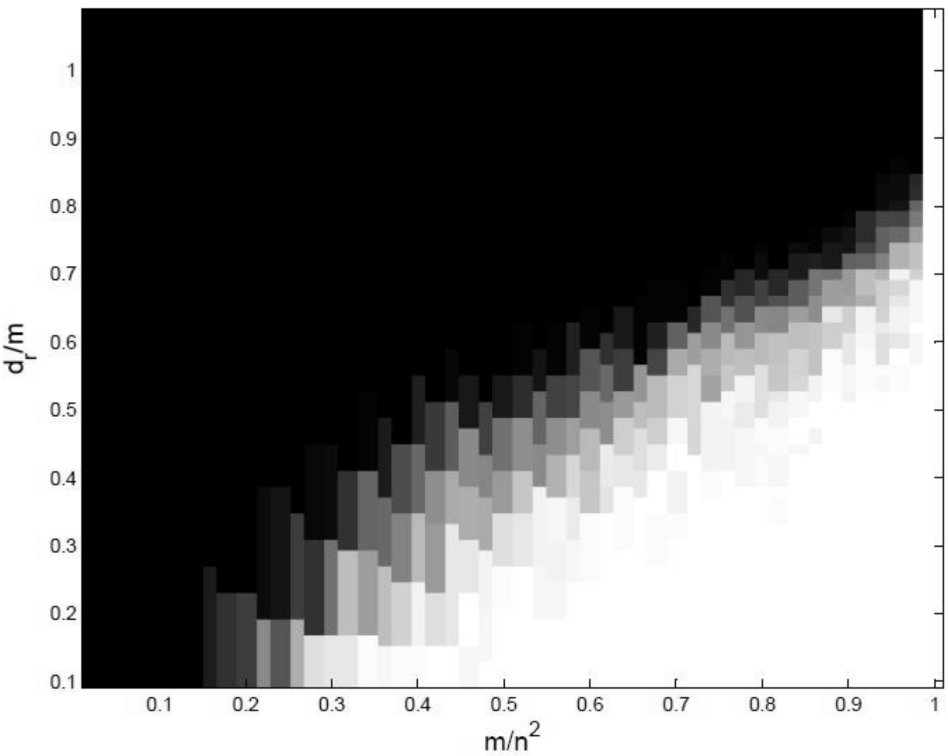

图 2.1: 矩阵填充的可行性

SVT算法可以理解为一种拉格朗日乘子法。首先，它求解的是原问题的一个近似问题：

$$\begin{aligned} \min \quad & \tau\|X\|_* + \frac{1}{2}\|X\|_F^2, \\ \text{subject to} \quad & P_\Omega(X) = P_\Omega(M), \end{aligned} \tag{2.3}$$

其拉格朗日乘子为

$$L(X,Y) = \tau\|X\|_* + \frac{1}{2}\|X\|_F^2 + \langle Y, P_\Omega(M-X)\rangle,$$

其中$\langle A,B\rangle = trace(AB^T)$。然后每一步通过最小化这个拉格朗日函数，最终收敛到最优解。迭代序列为：

$$\begin{cases} X_k = \arg\min\limits_X \ L(X, Y^{k-1}), \\ Y_k = Y_{k-1} + \delta_k P_\Omega(M - X_k). \end{cases}$$

由于

$$L(X,Y) = \tau\|X\|_* + \frac{1}{2}\|X - P_\Omega(Y)\|_F^2 + \langle P_\Omega(Y), M\rangle - \frac{1}{2}\|P_\Omega(Y)\|_F^2,$$

并且

$$\arg\min_X \tau\|X\|_* + \frac{1}{2}\|X - P_\Omega\|_F^2 = \mathcal{D}_\tau(P_\Omega(Y)),$$

其中

$$\mathcal{D}_\tau(X) := U\mathcal{S}_\tau(\Sigma)V^T, \text{如果} X = U\Sigma V^T. \tag{2.4}$$

而$\mathcal{S}_\tau$为收缩算子：

$$\mathcal{S}_\tau(x) = \begin{cases} x-\tau, & \text{如果} x > \tau, \\ x+\tau, & \text{如果} x < -\tau, \\ 0, & \text{其它}. \end{cases} \tag{2.5}$$

当Y初始化为0时，$P_\Omega(Y) = Y$始终成立，矩阵填充的SVT算法可写成算法1。

Algorithm 1 (矩阵填充的SVT算法)

1: 初始化$Y_0 = 0$,
2: **while** not converged **do**
3: $\quad X_k = D_\tau(Y^{k-1})$,
4: $\quad Y_k = Y_{k-1} + \delta_k P_\Omega(M - X_k)$.
5: **end while**

值得一提的是，序列X_k始终低秩，可以表示成两个瘦矩阵U_k与V_k^T之积，而Y_k始终是稀疏的，由于这些性质，该算法执行过程内存需求大大降低，因此适合大规模矩阵的计算。

2.1.3 APG算法

APG（Accelerated Proximal Gradient）算法[1]是一种利用Nesterov技巧的一阶算法，其收敛速度很有竞争力。[42]中将矩阵填充问题转化为一个与原问题近似的无约束优化问题：

$$\min \quad F(X) = \frac{1}{2}\|P_\Omega(X-M)\|_F^2 + \mu\|X\|_*. \tag{2.6}$$

APG算法每次最小化上述函数在某个点Y处的一个二阶近似：

$$Q(X,Y) = \frac{1}{2}\|P_\Omega(X-M)\|_F^2 + \langle P_\Omega(Y), X-Y\rangle + \frac{L_f}{2}\|X-Y\|_F^2 + \mu\|X\|_*, \tag{2.7}$$

其中L_f为$\bigtriangledown f$的Lipschitz常数：

$$\|\bigtriangledown f(X_1) - \bigtriangledown f(X_2)\| \le L_f\|X_1 - X_2\|. \tag{2.8}$$

由于

$$\arg\min_X Q(X,Y) = \arg\min_X \frac{L_f}{2}\left\|X - Y + \frac{1}{L_f}P_\Omega(Y)\right\|_F^2 + \mu\|X\|_*, \tag{2.9}$$

于是每次迭代：

$$X_{k+1} = \arg\min_X Q(X,Y_k) = \mathcal{D}_{\frac{\mu}{L_f}}\left(Y_k - \frac{1}{L_f}P_\Omega(Y_k)\right). \tag{2.10}$$

通常Y_k取以下序列时，算法收敛速度可以达到$O(k^{-2})$：

$$Y_k = X_k + \frac{t_{k-1}-1}{t_k}(X_k - X_{k-1}),\ t_{k+1}^2 - t_{k+1} \leq t_k^2. \tag{2.11}$$

由上面的推导，矩阵填充的APG算法如算法2所示。

Algorithm 2 (矩阵填充的APG算法)

1: 初始化X_0,X_{-1},
2: **while** not converged **do**
3: $\quad Y_k = Y_k = X_k + \frac{t_{k-1}-1}{t_k}(X_k - X_{k-1})$,
4: $\quad X_{k+1} = D_{\mu/L_f}(Y_k - \frac{1}{L_f}P_\Omega(Y_k))$,
5: $\quad t_{k+1} = \frac{1+\sqrt{t_k^2+1}}{2}, k = k+1.$
6: **end while**

[42]中的实验表明，APG算法采用了continuation技术和line search技术后，收敛速度比SVT等算法可以提高不少。

2.1.4　矩阵填充的应用

矩阵填充在计算机视觉和数据分析中已经找到重要的应用。矩阵填充非常适合于用来做推荐系统，可以参见J. Rennie等人的一篇文章[40]。Hui Ji等人[25]将其应用于视频去噪。

2.1.5　小结

以上讨论了矩阵填充目前的一些研究状况，包括理论上的可行性分析，当今的一些主流算法，和最新的应用。理论上，对符合一定模型的数据，可以通过求解上述优化问题对矩阵进行精确重建。但很多文章实验中的得到的结果表明可以精确重建的范围比理论上能证明的要宽的多。从矩阵填充的两种主流算法中可以看到，无论是哪种方法，都需要用到奇异值分解这一计算量较大的操作，而其收敛速度往往是次线性的，这些都说明在算法和计算方面有一定的提升空间。

2.2　矩阵恢复的研究现状

2.2.1　矩阵恢复的可行性

低秩矩阵恢复又称为Robust PCA或者稀疏与低秩矩阵分解。我们可以从PCA（Prin-cipal Component Analysis）的角度来看这个问题。传统的PCA可以理解为高维数据在低维线性子空间上的投影，它可写成：

$$\begin{aligned} &\min \quad \|E\|_F, \\ &\text{subject to} \quad \text{rank}(A) \leq r, D = A + E, \end{aligned} \tag{2.12}$$

其中D的每一列均为给定的数据，$\|\cdot\|_F$是矩阵的Frobenius范数，即所有元素的平方和再开根号。通过此约束优化问题可找到D在一个最近的r维线性子空间上的投影。当E为轻微的高斯随机噪声时，PCA可以通过一次SVD准确地找到最优的A。但当A被严重破坏，即E 很大时，A的估计往往不准确，并且PCA还有一个问题是需要预知子空间维数r。于是[43]中提出Robust PCA来解决A中数据被严重破坏的情况，见式（1.6）。与矩阵填充中类似，[43]中证明了: 对任意的$p>0$，存在常数C，s，m_0，使得当A满足随机正交模型时，如果$m>m_0$，且矩阵A的秩满足

$$r \leq C\frac{m}{\log m} \tag{2.13}$$

而E在某位置为正和为负的概率都小于s时，A能够精确恢复的概率为$1-Cm^{-p}$。图2.2是[8]中的实验结果，其横坐标为r/n，纵坐标为损坏元素所占比率。白色表示精确恢复，黑色表示不能恢复，灰色表示一定概率恢复。从图中可以看出，当矩阵的秩越低，被损坏的元素数目越少时，矩阵越容易精确恢复。

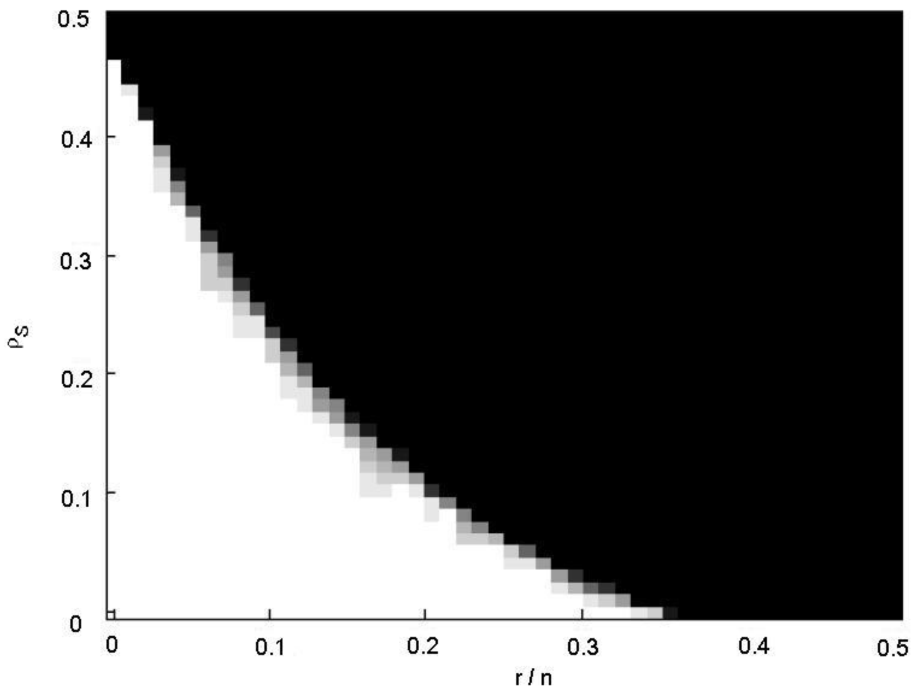

图 2.2: 矩阵恢复的可行性

2.2.2 IT算法

IT算法即迭代阈值（Iterative Thresholding）算法，与矩阵填充中的SVT算法类似，是求解矩阵恢复最早的算法[19]。它将原问题转化为如下近似问题：

$$\begin{aligned} \min_{A,E} \quad & \|A\|_* + \lambda \|E\|_1 + \frac{1}{2\tau} \|A\|_F^2 + \frac{1}{2\tau} \|E\|_F^2, \\ \text{subject to} \quad & A + E = D, \end{aligned} \tag{2.14}$$

其中τ取一个很大的正数，以使此问题的解近似于原问题。其拉格朗日函数为：

$$L(A,E,Y) = \|A\|_* + \lambda \|E\|_1 + \frac{1}{2\tau} \|A\|_F^2 + \frac{1}{2\tau} \|E\|_F^2 + \frac{1}{\tau}\langle Y, D - A - E\rangle. \tag{2.15}$$

于是

$$\arg\min_{A,E} L(A,E,Y) = \arg\min_{A,E} \|A\|_* + \frac{1}{2\tau}\|A - Y\|_F^2 + \lambda\|E\|_1 + \frac{1}{2\tau}\|E - Y\|_F^2. \tag{2.16}$$

IT算法的流程如算法3：

Algorithm 3 (矩阵恢复的迭代阈值法)

1: Input D, λ, τ.
2: **while** not converged **do**
3: $(U,S,V) = \text{svd}(Y_{k-1})$,
4: $A_k = U\mathcal{S}_\tau[S]V^T$,
5: $E_k = \mathcal{S}_{\lambda\tau}[Y_{k-1}]$,
6: $Y_k = Y_{k-1} + \delta_k(D - A_k - E_k)$.
7: **end while**
8: Output A_k,E_k

2.2.3 APG算法

类似于矩阵填充，矩阵恢复问题可用以下无约束问题来近似：

$$\min \quad F(A,E) = \frac{1}{2}\|D - A - E\|_F^2 + \mu(\|A\|_* + \lambda\|E\|_1). \tag{2.17}$$

接下来最小化上述函数在某个点（Y^A, Y^E）处的二阶近似：

$$\begin{aligned} Q(X^A, X^E, Y^A, Y^E) = & \frac{1}{2}\|D - Y^A - Y^E\|_F^2 + \langle Y^A + Y^E - D, X^A - Y^A + X^E - Y^E\rangle \\ & + \frac{L_f}{2}(\|X^A - Y^A\|_F^2 + \|X^E - Y^E\|_F^2) + \mu(\|X^A\|_* + \lambda\|X^E\|_1). \end{aligned} \tag{2.18}$$

由于

$$\begin{aligned}&\arg\min_{X^A,X^E} Q(X^A,X^E,Y^A,Y^E)=\\&\arg\min_{X^A,X^E}\frac{L_f}{2}\left\|X^A-Y^A+\frac{1}{L_f}(Y^A+Y^E-D)\right\|_F^2+\mu\|X^A\|_*\\&+\frac{L_f}{2}\left\|X^E-Y^E+\frac{1}{L_f}(Y^A+Y^E-D)\right\|_F^2+\mu\lambda\|X^E\|_1,\end{aligned}\tag{2.19}$$

于是得到如下迭代：

$$\begin{aligned}X_{k+1}^A&=\mathcal{S}_{\mu/L_f}\left(Y_k^A-\frac{1}{L_f}(Y_k^A+Y_k^E-D)\right),\\X_{k+1}^E&=\mathcal{S}_{\mu\lambda/L_f}\left(Y_k^E-\frac{1}{L_f}(Y_k^E+Y_k^E-D)\right).\end{aligned}\tag{2.20}$$

通常Y_k取以下序列时算法的收敛较快：

$$\begin{aligned}&Y_k^A=X_k^A+\frac{t_{k-1}-1}{t_k}(X_k^A-X_{k-1}^A),\\&Y_k^E=Y_k^E+\frac{t_{k-1}-1}{t_k}(X_k^E-X_{k-1}^E),\\&t_{k+1}^2-t_{k+1}\le t_k^2.\end{aligned}\tag{2.21}$$

由以上推导得到矩阵恢复的APG算法如算法4。

Algorithm 4 (矩阵恢复的APG算法)

1: Input D, λ.
2: $A_0=A_{-1}=0$; $E_0=E_{-1}=0$; $t_0=t_{-1}=1$; $\bar{\mu}>0$; $0<\eta<1$.
3: **while** not converged **do**
4: $\quad Y_k^A=A_k+\frac{t_{k-1}-1}{t_k}(A_k-A_{k-1})$, $Y_k^E=E_k+\frac{t_{k-1}-1}{t_k}(E_k-E_{k-1})$,
5: $\quad G_k^A=Y_k^A-\frac{1}{2}\left(Y_k^A+Y_k^E-D\right)$,
6: $\quad (U,S,V)=\mathrm{svd}(G_k^A)$, $A_{k+1}=U\mathcal{S}_{\frac{\mu_k}{2}}[S]V^T$,
7: $\quad G_k^E=Y_k^E-\frac{1}{2}\left(Y_k^A+Y_k^E-D\right)$,
8: $\quad E_{k+1}=\mathcal{S}_{\frac{\lambda\mu_k}{2}}[G_k^E]$,
9: $\quad t_{k+1}=\frac{1+\sqrt{4t_k^2+1}}{2}$, $\mu_{k+1}=\max(\eta\,\mu_k,\bar{\mu})$,
10: $\quad k\leftarrow k+1$.
11: **end while**
12: Output A_k, E_k.

[19]中指出，此算法通过使用continuation技术也可使收敛加快很多，但与矩阵填充不同，line search技术并不能使收敛加快。

2.2.4　矩阵恢复的应用

矩阵恢复在图像处理中应用广泛，如John Wright等人[43]展示了如何在背景建模、人脸识别等问题中利用矩阵恢复。Yigang Peng等人[39]将矩阵恢复技术应用于图片对齐中，其效果如图2.3。

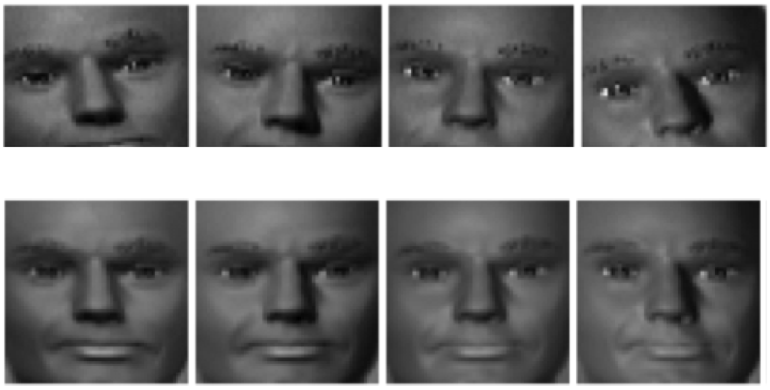

图 2.3: **图片对齐**

上行为对齐前的人脸，下行为利用矩阵恢复对齐后的人脸

2.2.5　小结

矩阵恢复与矩阵填充的理论分析和算法研究有所相似，最大的不同在于矩阵恢复中不知道噪声所在的位置，在模型中引入了1范数，变量的数目和约束多很多。在算法收敛速度方面，一般来讲现有的RPCA算法的收敛速度要慢一些，看起来是由于此问题噪声位置不明朗所致。矩阵恢复的应用前景很好，已经有很多工作将其应用到具体问题中。矩阵恢复中的稀疏矩阵E也很有意义，它表明对象之间的差异性，可能在文本分析和搜索技术等领域中会比较有用。

2.3　本章小结

本章介绍了矩阵重建的国内外研究现状。理论分析方面，[9]等文中对矩阵填充在何种情况下能够精确地重建出低秩矩阵进行了系统的分析和讨论，最终给出了概率上的证明；类似地，[8]等文对矩阵恢复在什么情况可以精确恢复也给出了重要的证明。算法方面，矩阵填充主要有SVT算法[6]、FPCA算法[32]、和APG算法[42]；矩阵恢复有迭代阈值算法、对偶算法和APG算法[19]。这些算法在收敛速度和每一步的时间代价上面有了不错的进展。矩阵填充和恢复在计算机视觉和数据分析方面的应用也开始广受关注，如背景建模、人脸识别、图片对齐、推荐系统等领域都已经开始使用到矩阵重建技术，未来在文本分析和搜索技术当中还可能会有进一步的发展。

第三章　矩阵重建算法的关键问题研究

在上一章中，我们介绍了矩阵重建，包括矩阵填充和矩阵恢复的研究现状，从中可以看出，已经有很多工作证明了矩阵重建的可行性，也有一些算法已经被提出来解决矩阵填充和矩阵恢复问题。但是，矩阵填充和矩阵恢复并不好解，无论在算法的收敛速度，还是在每一步所要计算的奇异值分解上，计算量都相当大。在算法收敛性方面，目前的方法的收敛速度在理论上都是次线性的；在每一步的运算量上，都需要一次奇异值分解，其复杂度为$O(n^3)$。因此，矩阵重建的运算量相当大，于是一个重要且有意义的问题是能否找到更快的算法。本章给出了一种增广拉格朗日乘子法来求解矩阵恢复和矩阵填充问题，实验表明此算法的收敛速度超过目前的主流算法。由于矩阵重建算法的关键运算是奇异值分解，本章亦给出了一种改进的求解部分奇异值分解的算法。

3.1　矩阵重建的增广拉格朗日乘子法

前面曾经提到过，很多算法如SVT算法可以理解为一种拉格朗日乘子法。但是还没有人在矩阵重建中应用增广拉格朗日乘子法（Augmented Lagrange Multiplier）。这里我们首先介绍一下一般的增广拉格朗日乘子法。

对于一个约束优化问题：

$$\begin{aligned} &\min \quad f(X), \\ &\text{subject to} \quad h(X) = 0, \end{aligned} \tag{3.1}$$

其中$f : \mathbb{R}^n \to \mathbb{R}$，$h : \mathbb{R}^n \to \mathbb{R}^m$。其增广拉格朗日函数为：

$$L(X, Y, \mu) = f(X) + \langle Y, h(X) \rangle + \frac{\mu}{2}\|h(X)\|_F^2, \tag{3.2}$$

其中μ为一个正数。注意增广拉格朗日函数相对普通的拉格朗日函数多出一个关于约束的惩罚项。增广拉格朗日乘子法每次迭代最小化增广拉格朗日函数，得到新的X_k，通过这个X_k和μ更新乘子Y_k，然后继续求解下一个X_k，最终X_k将收敛到原问题的最优解，见算法5。

3.1.1　矩阵恢复的增广拉格朗日乘子法

我们利用增广拉格朗日乘子法来求解矩阵恢复问题。矩阵恢复问题的增广拉格朗日函数为

$$L(A, E, Y, \mu) = \|A\|_* + \lambda\|E\|_1 + \langle Y, D - A - E \rangle + \frac{\mu}{2}\|D - A - E\|_F^2. \tag{3.3}$$

Algorithm 5 (增广拉格朗日乘子法)

1: 初始化μ_k, Y_k,
2: **while** not converged **do**
3: $\quad X_{k+1} = \arg\min_X L(X, Y_k, \mu_k)$,
4: $\quad Y_{k+1} = Y_k + \mu_k h(X_{k+1})$,
5: $\quad$更新 μ_k.
6: **end while**

每一步最小化（3.3）时可以利用交替更新的方式，即先固定E和Y求一个使L最小化的A，而后固定A和Y，求一个使L最小的E，这样迭代数次就可以收敛到这个子问题的最优解。更新A时，

$$\arg\min_A \|A\|_* + \frac{\mu}{2}\|D - A - E + \mu^{-1}Y\|_F^2 = \mathcal{D}_{\mu^{-1}}(D - E + \mu^{-1}Y), \tag{3.4}$$

更新E时，

$$\arg\min_E \lambda\|E\|_1 + \frac{\mu}{2}\|D - A - E + \mu^{-1}Y\|_F^2 = \mathcal{S}_{\lambda/\mu}(D - A + \mu^{-1}Y). \tag{3.5}$$

交替更新直到对子问题的求解收敛。这样，我们称此算法为矩阵恢复的精确增广拉格朗日乘子法(Exact Augmented Lagrange Multiplier, 简称EALM)，见算法6。

Algorithm 6 (矩阵恢复的EALM算法)

1: 初始化Y_0^*, $E_0^* = 0$, μ_0, $k = 0$,
2: **while** not converged **do**
3: $\quad E_{k+1}^0 = E_k^*$, $j = 0$,
4: $\quad$**while** not converged **do**
5: $\quad\quad (U, S, V) = \mathrm{svd}(D - E_{k+1}^j + \mu_k^{-1}Y_k^*)$,
6: $\quad\quad A_{k+1}^{j+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T$,
7: $\quad\quad E_{k+1}^{j+1} = \mathcal{S}_{\lambda\mu_k^{-1}}[D - A_{k+1}^{j+1} + \mu_k^{-1}Y_k^*]$,
8: $\quad\quad j = j + 1$.
9: $\quad$**end while**
10: $\quad Y_{k+1}^* = Y_k^* + \mu_k(D - A_{k+1}^* - E_{k+1}^*)$; $\mu_{k+1} = \rho\mu_k$,
11: $\quad k = k + 1$.
12: **end while**

之所以称算法6为精确增广拉格朗日乘子法，是因为我们发现，每一步我们并不需要求解出子问题的精确解，实际上，我们只需要更新A与E各一次得到子问题的一个近似解，已足以使算法最终收敛到原问题的最优解，这样我们可以得到一个更简洁且收敛更快的算法。我们称之为非精确增广拉格朗日乘子法（Inexact Augmented Lagrange Multiplier，简称IALM），见算法7。算法的收敛性证明请参见[30]。

Algorithm 7 (矩阵恢复的IALM算法)

初始化Y_0, $E_0 = 0$, μ_0, $k = 0$,

while not converged **do**

$(U, S, V) = \text{svd}(D - E_k + \mu_k^{-1} Y_k)$,

$A_{k+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T$,

$E_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}[D - A_{k+1} + \mu_k^{-1} Y_k]$,

$Y_{k+1} = Y_k + \mu_k(D - A_{k+1} - E_{k+1})$,

更新μ_k,

$k = k + 1$.

end while

3.1.2 矩阵恢复的实验结果

首先，在仿真数据下，我们将IALM算法与主流的APG算法进行比较，实验结果如表3.1和3.2。实验中，我们根据三元组$\{m, \text{rank}(A^*), \|E^*\|_0\}$随机生成矩阵。其中$m$为矩阵维数，假定$\text{rank}(A^*) = r$，我们首先生成每个元素为独立高斯随机变量的矩阵$L = \text{randn}(m, r)$和$R = \text{randn}(m, r)$，然后计算出$A^* = LR^T$。同时，我们生成一个非零元素数目为$\|E^*\|_0$，非零元素均匀分布，且值均匀分布在$[-500, 500]$之间的稀疏矩阵$E^*$。接着利用$D = A^* + E^*$生成目标矩阵。松弛因子$\lambda$始终设为$m^{-1/2}$[43]。利用三种矩阵恢复算法得出$\hat{A}$与$\hat{E}$，我们统计$\hat{A}$的秩以及与$A^*$的相对误差，$\hat{E}$的非零元素数目，以及算法的迭代次数和运行时间。从仿真结果中可以看出，同一矩阵上IALM算法和EALM算法都比主流的APG算法快，而IALM比EALM更快，并且在精度更高的情况下比APG快5倍以上。

我们将IALM算法应用到实际数据中对于720×576大小的视频，选取其中的102帧进行背景建模。图3.1中左、中、右三幅图分别是原图、提取出的背景和前景。从中可见，IALM算法可以很清晰地将视频中的背景，也就是低秩矩阵A中的数据分离出来。

图 3.1: **利用矩阵恢复的IALM算法进行背景建模**

左图为原图，中图为提取出的背景，右图为前景

表 3.1: **矩阵恢复算法比较1**

m	algorithm	$\frac{\\|\hat{A}-A^*\\|_F}{\\|A^*\\|_F}$	rank($\hat{A}$)	$\\|\hat{E}\\|_0$	#SVD	time (s)
rank$(A^*) = 0.05\,m$, $\\|E^*\\|_0 = 0.05\,m^2$						
500	APG	1.12e-5	25	12542	127	11.01
	EALM	3.99e-7	25	12499	28	4.08
	IALM	5.21e-7	25	12499	20	1.72
1000	APG	8.79e-6	50	50082	126	57.62
	EALM	7.85e-8	50	50000	29	33.28
	IALM	2.67e-7	50	49999	22	10.13
2000	APG	6.27e-6	100	200243	126	353.63
	EALM	4.61e-8	100	200000	30	243.64
	IALM	9.54e-8	100	200000	22	68.69
3000	APG	5.20e-6	150	450411	126	1106.22
	EALM	4.39e-8	150	449998	30	764.66
	IALM	1.49e-7	150	449993	22	212.34
rank$(A^*) = 0.05\,m$, $\\|E^*\\|_0 = 0.10\,m^2$						
500	APG	1.41e-5	25	25134	129	14.35
	EALM	8.72e-7	25	25009	34	4.75
	IALM	9.31e-7	25	25000	21	2.52
1000	APG	9.97e-6	50	100343	129	65.41
	EALM	6.07e-7	50	100002	33	30.63
	IALM	3.78e-7	50	99996	22	10.77
2000	APG	7.11e-6	100	400988	129	353.30
	EALM	1.23e-7	100	400001	34	254.77
	IALM	3.31e-7	100	399993	23	70.33
3000	APG	5.79e-6	150	901974	129	1110.76
	EALM	1.05e-7	150	899999	34	817.69
	IALM	2.27e-7	150	899980	23	217.39

我们再次将算法在视频数据中的运行结果与APG算法进行比较。IALM与APG的运行结果如表3.3。IALM比APG算法快4-5倍。可见无论在仿真中，还是实际应用中，IALM算法都体现出很大的优势。

表 3.2: **矩阵恢复算法比较2**

m	algorithm	$\frac{\Vert\hat{A}-A^*\Vert_F}{\Vert A^*\Vert_F}$	rank($\hat{A}$)	$\Vert\hat{E}\Vert_0$	#SVD	time (s)
rank(A^*) $= 0.10\,m$, $\Vert E^*\Vert_0 = 0.05\,m^2$						
500	APG	9.36e-6	50	13722	129	13.99
	EALM	5.53e-7	50	12670	41	7.35
	IALM	6.05e-7	50	12500	22	2.32
1000	APG	6.64e-6	100	54128	129	129.40
	EALM	4.20e-7	100	50207	39	50.31
	IALM	2.61e-7	100	50000	22	20.71
2000	APG	4.77e-6	200	215874	129	888.93
	EALM	1.15e-7	200	200512	41	423.83
	IALM	2.49e-7	200	199998	23	150.35
3000	APG	3.98e-6	300	484664	129	2923.90
	EALM	7.92e-8	300	451112	42	1444.74
	IALM	1.30e-7	300	450000	23	485.70
rank(A^*) $= 0.10\,m$, $\Vert E^*\Vert_0 = 0.10\,m^2$						
500	APG	9.78e-6	50	27478	133	13.90
	EALM	1.14e-6	50	26577	52	9.46
	IALM	7.64e-7	50	25000	25	2.62
1000	APG	7.75e-6	100	109632	132	130.37
	EALM	3.40e-7	100	104298	49	77.26
	IALM	3.73e-7	100	99999	25	22.95
2000	APG	5.49e-6	200	437099	132	884.86
	EALM	2.81e-7	200	410384	51	570.72
	IALM	4.27e-7	200	399999	24	154.27
3000	APG	4.50e-6	300	980933	132	2915.40
	EALM	2.02e-7	300	915877	51	1904.95
	IALM	3.39e-7	300	899990	24	503.05

表 3.3: **APG和IALM在视频数据上的比较.**

size	Algorithm	Iterations	Time(s)
414720×102	IALM	37	528.3
414720×102	APG	151	2238.4

3.1.3 矩阵填充的IALM算法

既然IALM算法可以在矩阵恢复问题中很好地工作，那么在矩阵填充中，应当也会有效。

我们首先将矩阵填充问题写成：

$$\begin{aligned} &\min \quad \|A\|_*, \\ &\text{subject to} \quad A + E = D,\ P_\Omega(E) = 0. \end{aligned} \tag{3.6}$$

它的带约束$P_\Omega(E) = 0$的部分增广拉格朗日函数（Partial Augmented Lagrange Multiplier）为：

$$L(A, E, Y, \mu) = \|A\|_* + \langle Y, D - A - E\rangle + \frac{\mu}{2}\|D - A - E\|_F^2. \tag{3.7}$$

这样每次更新A时，

$$\arg\min_A \|A\|_* + \frac{\mu}{2}\|D - A - E + \mu^{-1}Y\|_F^2 = \mathcal{D}_{\mu^{-1}}(D - E + \mu^{-1}Y), \tag{3.8}$$

更新E时，

$$\arg\min_{P_\Omega(E)=0} \|D - A - E + \mu^{-1}Y\|_F^2 = P_{\bar{\Omega}}(D - A + \mu^{-1}Y), \tag{3.9}$$

于是我们得到矩阵填充的IALM算法8。

需要说明的是上述算法中：

$$\begin{aligned} &Y_{k+1} \\ &= Y_k + \mu_k(D - A_{k+1} - P_{\bar{\Omega}}(D - A_{k+1} + \mu_k^{-1}Y_k)) \\ &= Y_k - P_{\bar{\Omega}}(Y_k) + \mu_k(D - A_{k+1} - P_{\bar{\Omega}}(D - A_{k+1}) \\ &= P_\Omega(Y_k + \mu_k(D - A_{k+1})), \end{aligned} \tag{3.10}$$

也就是说Y_{k+1}始终只在Ω中非零。而A_{k+1}可以写成两个条形矩阵的乘积

$$A_{k+1} = \left(U\sqrt{\mathcal{S}_{\mu_k^{-1}}[S]}\right)\left(\sqrt{\mathcal{S}_{\mu_k^{-1}}[S]}V^T\right) := U_{k+1}V_{k+1}^T. \tag{3.11}$$

Algorithm 8 (矩阵填充的IALM算法)

$Y_0 = 0,\ E_0 = 0,\ \mu_0 = 0,\ k = 0,$
while not converged **do**
  $(U, S, V) = \mathrm{svd}(D - E_k + \mu_k^{-1} Y_k),$
  $A_{k+1} = U \mathcal{S}_{\mu_k^{-1}}[S] V^T,$
  $E_{k+1} = P_{\bar{\Omega}}(D - A_{k+1} + \mu_k^{-1} Y_k),$
  $Y_{k+1} = Y_k + \mu_k(D - A_{k+1} - E_{k+1}),$
  更新 $\mu_k,$
  $k = k + 1.$
end while

于是E_{k+1}可以表示为：

$$E_{k+1} = P_{\Omega}(A_{k+1}) - A_{k+1} = P_{\Omega}(A_{k+1}) - U_{k+1} V_{k+1}^T. \tag{3.12}$$

这样，我们可以将运算过程中的所有矩阵都用某两个条形矩阵和一个稀疏矩阵来表示，内存消耗极少，计算矩阵向量乘积时的计算量也大为减小，因此适于大规模矩阵的处理。

3.1.4　矩阵填充的实验结果

我们将矩阵恢复的IALM算法与前面提到的SVT算法和APG算法比较，实验结果如3.4。实验过程中，我们根据三元组$\{m,\ \mathrm{rank}(A^*),\ p/d_r\}$随机生成矩阵，其中$m$为方阵维数，$p$为采样元素的数目，$r$为矩阵的秩，$d_r = r * (2m - r)$为秩为$r$的$m$乘$m$矩阵的自由度。先利用前面提到的随机正交模型生成一个秩为r的矩阵A，然后均匀采样矩阵的p个元素。将采样的数据输入到三种算法中，我们给出了三种算法针对同一矩阵的计算时间和精度。从中可以看出，IALM算法在所有情况下都比SVT算法快，在采样率大于10%时速度比APG快。的确正如我们所预期的，IALM算法在矩阵填充中表现良好。

另外，我们也在实际的搜索数据上应用了IALM算法进行填充。搜索数据的格式为

（URL，Query，Features，Rating），

表示每一个网页对于用户输入的某个查询都有一组特征，最终的Rating决定了该网页与用户查询的符合程度。这些众多的特征当中，有大量的信息缺失，因为很多情况下某个网页的某些特征值是无法得到的。由于一个网页的好坏往往是由少数几种特征所决定，所以我们可以假定特征之间线性相关，即矩阵是低秩的，于是我们利用矩阵填充推测出空缺位置的一些特征值。实验结果如表3.5。该实验中，对于大小为1,682,588×174的矩阵，所用时间不超过2个小时，说明我们的算法适合大规模数据的处理。正如前面分析的，这主要得益于矩阵填充的IALM 算法中的中间矩阵都有比较良好的低秩结构和稀疏性，可以很精简地表达和计算。

表 3.4: **SVT、APG 与IALM 矩阵填充算法比较.**

m	r	p/d_r	p/m^2	algorithm	#iter	rank($\hat{A}$)	time (s)	$\frac{\|\hat{A}-A^*\|_F}{\|A^*\|_F}$
1000	10	6	0.12	SVT	208	10	18.23	1.64e-6
				APGL	69	10	4.46	3.16e-6
				IALM	69	10	3.73	1.40e-6
1000	50	4	0.39	SVT	201	50	126.18	1.61e-6
				APGL	76	50	24.54	4.31e-6
				IALM	38	50	12.68	1.53e-6
1000	100	3	0.57	SVT	228	100	319.93	1.71e-6
				APGL	81	100	70.59	4.40e-6
				IALM	41	100	42.94	1.54e-6
5000	10	6	0.024	SVT	231	10	141.88	1.79e-6
				APGL	81	10	30.52	5.26e-6
				IALM	166	10	68.38	1.37e-6
5000	50	5	0.10	SVT	188	50	637.97	1.62e-6
				APGL	88	50	208.08	1.93e-6
				IALM	79	50	230.73	1.30e-6
5000	100	4	0.158	SVT	215	100	2287.72	1.72e-6
				APGL	98	100	606.82	4.42e-6
				IALM	64	100	457.79	1.53e-6
8000	10	6	0.015	SVT	230	10	283.94	1.86e-6
				APGL	87	10	66.45	5.27e-6
				IALM	235	10	186.73	2.08e-6
8000	50	5	0.06	SVT	191	50	1095.10	1.61e-6
				APGL	100	50	509.78	6.16e-6
				IALM	104	50	559.22	1.36e-6
10000	10	6	0.012	SVT	228	10	350.20	1.80e-6
				APGL	89	10	96.10	5.13e-6
				IALM	274	10	311.46	1.96e-6
10000	50	5	0.05	SVT	192	50	1582.95	1.62e-6
				APGL	105	50	721.96	3.82e-6
				IALM	118	50	912.61	1.32e-6

注：APGL为采用line search对APG进行加速后的算法

表 3.5: **矩阵填充的IALM算法用于搜索数据.**

size	Algorithm	Iterations	Time(s)
$1,682,588 \times 174$	IALM	24	5331

3.2 一种改进的部分奇异值分解算法

前面我们提出了一种收敛速度更快的增广拉格朗日乘子法，使矩阵重建算法的速度得到大幅提升。事实上，我们还可以考虑的是，每一轮迭代过程中的计算量是否可以减少。回顾前面提到的各种矩阵重建算法中，每一轮迭代的计算主要都是在解如下形式的问题：

$$\arg\min_X \varepsilon\|X\|_* + \frac{1}{2}\|X - W\|_F^2. \tag{3.13}$$

[6]等文中都指出了解上述问题需要对矩阵W做一次奇异值分解，而后将所有大于ε 的奇异值及其对应的奇异向量组合成X即可（详见（2.4）与（2.5））。其中对于一个矩阵进行奇异值分解的计算复杂度是$O(n^3)$，所以计算量是相当大的。但由于我们需要得到的仅仅是大于某一阈值ε 的奇异值分解，计算全部的奇异值分解是不必要的。然而遗憾的是，目前为止并没有任何直接提供这种功能的算法。[6][42]等文章中都用了Larsen等人的PROPACK包[29]，它基于Lanczos方法，可以计算奇异值分解中前k个奇异值和奇异向量。于是在矩阵重建算法中，需要在每一步预测需要多少个奇异值及奇异向量，这样有可能造成计算的浪费或者不准确。在仔细研究PROPACK中的Lanczos SVD后，本文给出了一种改进的直接计算大于某一阈值的部分奇异值分解算法。

3.2.1 部分奇异值分解

谈到部分奇异值分解，我们首先回顾一下一般的奇异值分解。对一个矩阵$A \in \mathbb{R}^{m\times n}, m \ge n$，它的奇异值分解为：

$$A = \sum_{j=1}^{n} \sigma_j u_j v_j^T, \tag{3.14}$$

其中

$$v_j^T v_k = u_j^T u_k = \begin{cases} 1, & \text{if } j = k, \\ 0, & \text{otherwise,} \end{cases}$$

$$u_j \in \mathbb{R}^m, v_j \in \mathbb{R}^n, j = 1, 2, \ldots, n, \sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_n \ge 0.$$

或者将u_j排列成矩阵U，σ_j排列成对角阵S，v_j排列成矩阵V，写成：

$$A = USV^T. \tag{3.15}$$

奇异值分解历史悠久，现在已有非常稳定的算法[5][14][15]，在解决最小二乘问题时，以及在数据分析、信息检索等领域一直有非常重要的作用。

而传统的部分奇异值分解是指求出前k个奇异值和奇异向量，即对$k < n$，

$$\bar{A}_k := \sum_{j=1}^{k} \sigma_j u_j v_j^T, \tag{3.16}$$

也就是求出矩阵

$$U_k = [u_1, u_2, \ldots, u_k],$$

$$S_k = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_k \end{pmatrix},$$

$$V_k = [v_1, v_2, \ldots, v_k],$$

满足：

$$\begin{aligned} AV_k &= U_k S_k, \\ A^T U_k &= V_k S_k. \end{aligned} \tag{3.17}$$

但我们在矩阵重建算法中间需要用到的，是要求出大于某一阈值t的所有奇异值和奇异向量。即：

$$\bar{A}_t := \sum_{\sigma_j > t} \sigma_j u_j v_j^T. \tag{3.18}$$

目前没有找到直接提供这一功能的算法，因此每次都需要估计出一个大于阈值t的奇异值的个数k，然后通过传统的方法求解出前k个奇异值看是否找到了所有大于t的奇异值。在这个过程中，如果这个k估计得过大，那么我们浪费了一些时间，如果k估计得过小，那么可能会造成计算的不精确（虽然实验表明，每一步奇异值分解的精度对IALM算法的收敛影响不大）。为了避免这个问题，我们首先研究一下现有的求解部分奇异值分解的PROPACK包是如何做到部分求解功能的。

PROPACK中的奇异值分解主要是基于Lanczos二对角化算法，结合了一种隐式重启动（Implicit Restart）方法和部分再正交化（Partial Reorthogonalization）方法来提高算法的速度。它在迭代过程中首先将目标矩阵二对角化，然后将二对角阵进行奇异值分解，最终得到原矩阵的奇异值分解。而在将矩阵而二对角化的过程中，中间生成的二对角矩阵的最大和最小的那部分奇异值总是最快地收敛到原矩阵的奇异值，这一性质使其易于计算部分奇异值分解。

对于一个$m \times n$的输入矩阵A，Lanczos二对角化的主要过程如算法9[29]。

Algorithm 9 (Lanczos二对角化)

初始化向量$p_0 \in \mathbb{R}^m$,
$\beta_1 = \|p_0\|, u_1 = p_0/\beta_1, v_0 = 0$,
for $i = 1, 2, ..., k$ **do**
　$r_i = A^T u_i - \beta_i v_{i-1}, r_i = \mathrm{reorth}(r_i)$,
　$\alpha_i = \|r_i\|, v_i = r_i/\alpha_i$,
　$p_i = A v_i - \alpha_i u_i, p_i = \mathrm{reorth}(p_i)$,
　$p_{i+1} = \|p_i\|, u_{i+1} = p_i/\beta_{i+1}$.
end for

这样迭代k步以后，A有如下分解形式：

$$\begin{aligned} AV_k &= U_{k+1} B_k, \\ A^T U_{k+1} &= V_k B_k^T + \alpha_{k+1} v_{k+1} e_{k+1}^T, \end{aligned} \tag{3.19}$$

其中V_k和U_{k+1}为算法9中的Lanczos向量组合而成，每一列都正交，且已归一化。而B_k为如下形式的二对角矩阵：

$$B_k = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{pmatrix}.$$

二对角化过程中的B_k的最大和最小的那部分奇异值都近似于矩阵A的奇异值。当$k = n$时，将B_k做奇异值分解就可以得到原矩阵的SVD。PROPACK中的隐式重启动的SVD是指计算部分奇异值分解时，在二对角化过程中迭代k步，判定是否得到足够数目的精确的奇异值，如已足够，则将B_k做SVD，并将相应的正交矩阵相乘得到左奇异矩阵和右奇异矩阵；如奇异值数目不够，则重新开始进行更高维数的二对角化。注意前面得到的信息不需要浪费，所以一般都是接着前面的结果继续进行二对角化（此即为隐式的重启动），这样直到算出足够数目的奇异值为止。求解前k个奇异值的部分奇异值分解的具体过程如算法10。

3.2.2　改进的部分奇异值分解算法

前面提到的算法对给定的k，进行隐式重启动不断增加二对角矩阵B_k的大小K，这个K的选取对算法的速度有重要的影响。我们在计算大于某一阈值的部分奇异值分解

Algorithm 10 (lansvd部分奇异值分解)

```
输入矩阵A; 奇异值个数k,初始化向量p_0,
二对角矩阵维数K,β_1 = ||p_0||,
u_1 = p_0/β_1, i = 1, neig = 0,
while neig < k do
  while i ≤ K do
    r_i = A^T u_i − β_i v_{i−1}, r_i = reorth(r_i),
    α_i = ||r_i||, v_i = r_i/α_i,
    p_i = A v_i − α_i u_i, p_i = reorth(p_i),
    p_{i+1} = ||p_i||, u_{i+1} = p_i/β_{i+1},
    i = i + 1.
  end while
  计算二对角阵B_k的奇异值分解,
  neig = 已精确求出的奇异值的数目,
  if neig > 0
    K = K + min(100, max(2, 0.5 ∗ (k − neig) ∗ K/(neig + 1))),
  else
    K = max(1.5 ∗ K, K + 10).
  end
end while
```

时，在原来算法的基础上，经验地给出一种新的隐式重启动的规则如下：

$$K = \begin{cases} K + \min(100, \max(2, (\mathrm{length}(S > svthr) - neig) * K/neig)), & \text{if } neig > 0, \\ 2 * K, & \text{otherwise.} \end{cases} \tag{3.20}$$

其中S为二对角矩阵B_k的奇异值构成的向量，$svthr$为输入的阈值，$neig$为已计算出的精确奇异值的数目。算法11即为完整的阈值奇异值算法。

3.2.3 仿真与比较

我们将改进后的部分奇异值分解算法，我们称之为阈值部分奇异值分解，用于矩阵恢复算法，并与改进前的算法作比较，见表3.6。改进后的算法精度和速度有一定优势。需要指出的是，这个工作主要的贡献在于提供了一种直接可以调用的功能，而以往我们需要进行额外的预测奇异值和奇异向量个数的工作。

3.3 本章小结

本章主要针对矩阵重建中的一些关键问题进行了研究和探讨。在收敛速度方面，

Algorithm 11 (阈值部分奇异值分解)

```
输入矩阵$A$, 阈值$svthr$, 初始化向量$p_0$,
    二对角矩阵维数$K$,$\beta_1 = \|p_0\|$,
    $u_1 = p_0/\beta_1$, $i = 1$, $minsv = svthr + 1$,
while $minsv \geq svthr$ do
    while $i \leq K$ do
        $r_i = A^T u_i - \beta_i v_{i-1}, r_i = \text{reorth}(r_i)$,
        $\alpha_i = \|r_i\|, v_i = r_i/\alpha_i$,
        $p_i = Av_i - \alpha_i u_i, p_i = \text{reorth}(p_i)$,
        $p_{i+1} = \|p_i\|, u_{i+1} = p_i/\beta_{i+1}$,
        $i = i + 1$.
    end while
    计算二对角矩阵$B_k$的奇异值分解, 得到奇异值向量S,
    $neig$ = 已精确求出的奇异值的数目,
    $minsv$ = 已求得的精确的奇异值中的最小值,
    利用规则(3.20)更新$K$.
end while
```

表 3.6: **IALM和IALMthr矩阵恢复算法比较.**

m	algorithm	$\frac{\|\hat{A}-A^*\|_F}{\|A^*\|_F}$	rank$(\hat{A})$	$\|\hat{E}\|_0$	#SVD	time (s)
rank$(A^*) = 0.05\,m$, $\|E^*\|_0 = 0.10\,m^2$						
500	IALM	1.05e-6	25	25000	21	3.62
	IALMthr	1.01e-6	25	25000	21	2.28
1000	IALM	3.41e-7	50	99999	24	19.79
	IALMthr	3.11e-7	50	99999	24	9.26
2000	IALM	1.03e-6	100	399996	23	75.49
	IALMthr	1.03e-6	100	399996	23	65.68
3000	IALM	1.71e-6	150	899989	23	184.97
	IALMthr	1.69e-6	150	899989	23	181.38

给出了一种增广拉格朗日乘子法提高算法的收敛速度，实验表明我们的算法无论在仿真数据上，还是在真实数据上速度相比APG算法都提高4倍以上。在奇异值分解方面，给出了一种改进的求解部分奇异值分解的算法，该算法可以直接求解大于某一阈值的

奇异值分解。这个改进的部分奇异值分解算法的价值在于提供了一种可以直接使用的功能，而不再需要预测所需矩阵奇异值和奇异向量的个数；并且实验表明，该算法在速度上比用改进前的算法有所提升。

第四章　并行算法的设计与实现

这是一个资讯呈爆炸性增长的时代，工程实践和科学研究中数据量膨胀，导致对计算的需求无论在速度还是在规模上都空前迫切。除了多核处理器，一些显卡厂商如Nvidia和ATI等，将图形处理器（Graphics Processing Unit，GPU）引入到通用计算中，其强大的并行计算能力在科学计算中显示出日益重要的作用，其发展速度十分迅猛甚至超过了摩尔定律。除此之外，大规模集群的分布式计算显得尤为重要，它破除了单机计算的内存制约，使得一些重要的应用不再有数据规模的限制。本章首先简单介绍GPU并行计算（可参见[37]），然后依次描述雅可比奇异值分解算法，矩阵恢复的并行算法的设计与实现。另外，我们还将对集群上的矩阵恢复算法进行简单的介绍。

4.1　GPU并行计算

图形处理器（GPU）十分适合处理并行的任务，因为在对图像进行处理或者图形进行渲染时，往往对每个像素做相同的操作，所以GPU的设计从一开始就从并行处理出发，设计成多核架构，易于多线程同时执行。相比于CPU，GPU利用更多资源来进行数据处理，而非控制，如图4.1所示（参见[37]）。

图 4.1: **CPU与GPU**

由于GPU强大的计算能力，人们在图形图像处理以外，开始将其用于通用计算。Nvidia公司推出CUDA(Compute Unified Device Architecture)计算环境，提供类似于C语言的接口，使用和学习代价较低。ATI推出CTM(Close-To-Metal) 提供一组API以实现并行编程。Nvidia GTX 280理论上对单精度浮点数运算能够达到933GFLOPS，双精度浮点数运算达到78GFLOPS。因此，我们利用CUDA来实现并行算法。CUDA中，通过编写一种特殊的函数Kernel来实现并行，它为每个线程分配一个id，不同的线程针对各自id进行运算，同一条指令，会对应到多个线程同步执行。线程组织成一个个Block，同一个Block内的线程通过共享内存进行交互。[37]中GPU多线程执行模型如

图4.2所示，GPU的并行执行是通过CPU(Host)来调用，调用完之后回到Host执行串行代码。

[37]中，GPU上的内存分为普通全局内存、共享内存、寄存器、常量内存和纹理内存，如图4.3。常用的是前三者，其中全局内存即为普通的显存，容量较大，一般预先将Host上的数据导入到此，以便GPU对其进行处理。共享内存和寄存器的传输速度极快，但是容量有限，在每个多核处理器上分布有一小部分。

写程序时可以调用线性运算库CUBLAS[36]中的一些函数，也可以自己编写Kernel。并行程序的编写应选择适合并行的算法，实现时注意内存的聚合访问，控制内存读写上的开销，使用尽可能大吞吐率的指令，根据硬件设置好相应的参数使同时活跃的线程尽可能多。

4.2 并行雅可比奇异值分解算法的设计与实现

前面已经提到过，矩阵重建问题中奇异值分解消耗了大部分时间。虽然部分奇异值分解可以很大程度上提升算法的性能，但是实验表明，当需要计算的奇异值的数目超过矩阵维数（行数和列数的较小值）的20%时，部分奇异值分解的速度甚至会比做全部奇异值分解更慢。而且，在Lanczos部分奇异值分解中，对双对角矩阵也需要做全部奇异值分解。所以，我们首先考虑的是做一个通用的奇异值分解的GPU并行实现。

4.2.1 基本的雅可比奇异值分解算法

奇异值分解主要有三种典型方法：QR迭代法[5]、Jacobi法[14]、分而治之法[22]。其中Jacobi法的并行性[3][2]最好，并且它是精度最高的奇异值分解。对于一个矩阵A，它通过一系列平面旋转将A^TA的非对角元素不停地消0。也就是说，不停地将矩阵的每两列正交化，迭代使矩阵所有列都两两正交。将正交化后的矩阵的每一列归一化后，每一列的范数即为奇异值，归一化后的矩阵即为左奇异矩阵。而右奇异矩阵在正交化过程中累积得到。算法12即为最基本的雅可比奇异值分解算法[24]。

4.2.2 并行雅可比奇异值分解算法

算法12中，在对一个n阶方阵的每两列做正交化时，共有$n(n-1)/2$种组合。而n列可以分成$n/2$对二元组，每个二元组中的两列可以互不干扰地进行正交化。于是在每次内循环中，又可以分为$n-1$次进行，每次$n/2$个列对同步进行正交化，这就是此算法的并行性所在。由于在对矩阵的所有列进行正交化的过程中，前面已经正交化的列的正交性将丢失，所以需要迭代多轮最终使所有列两两正交。所以一个非常重要的问题是，如何排列这$n(n-1)/2$种组合，使得所有列之间两两之间都做过正交化，并使整体算法尽可能快地收敛。事实上，在串行算法中，一般采用行循环策略或者列循环策略（即每次都先让某一行或者一列与其它所有行或列进行正交化）可以使收敛最快，

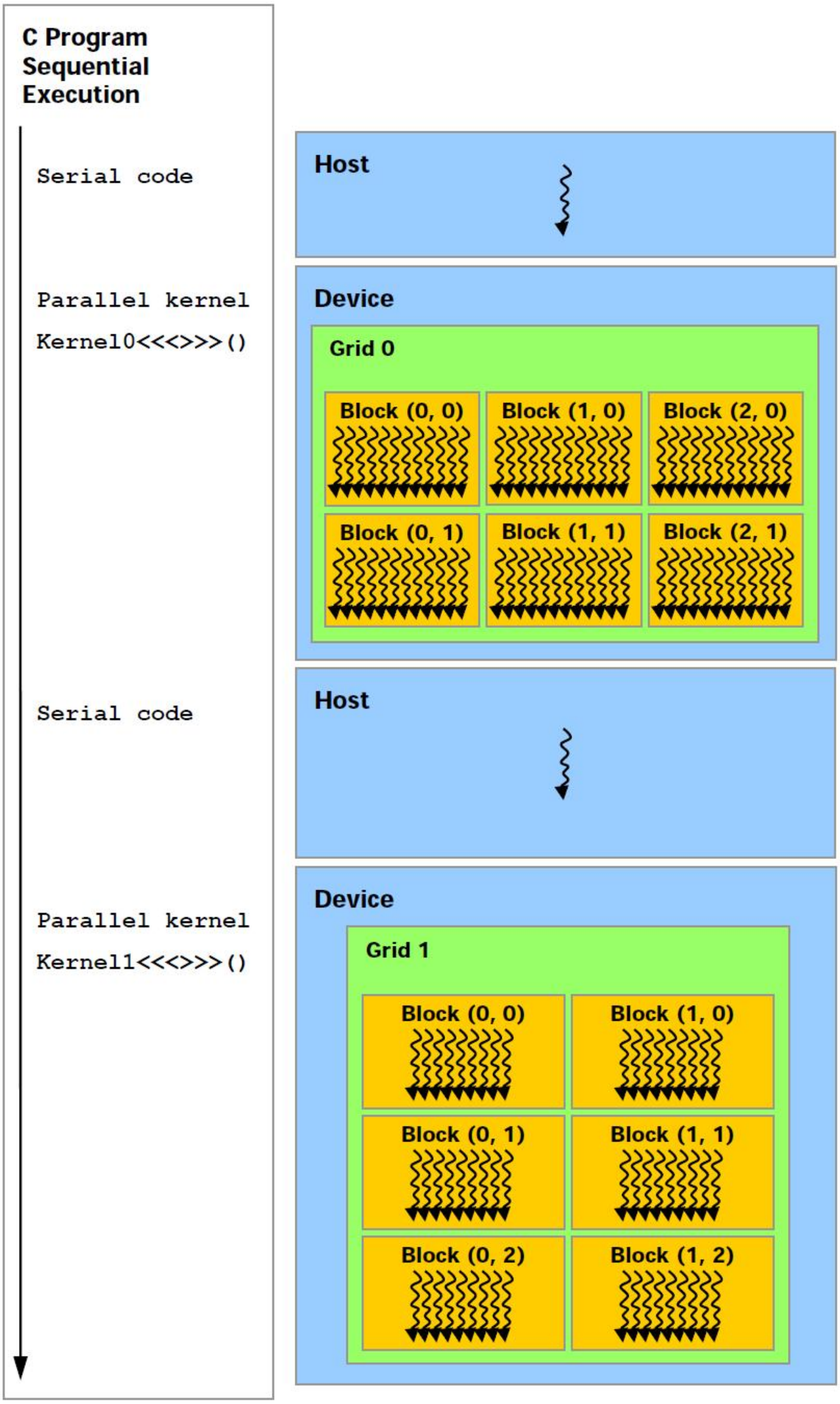

图 4.2: **GPU多线程执行模型**

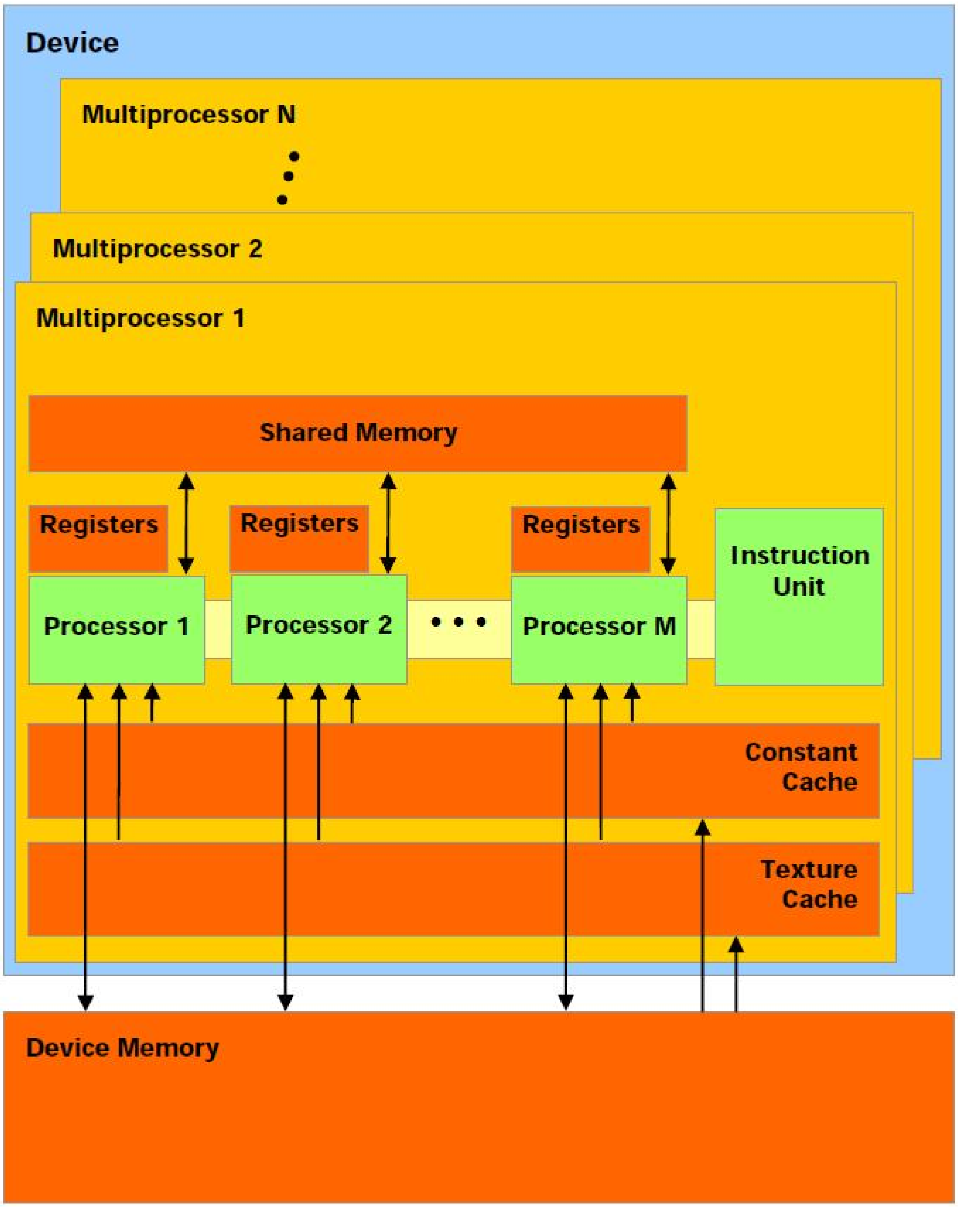

图 4.3: **GPU内存模型**

10～12次就可结束。但是在并行排序时，算法的收敛速度往往会变得比较慢，如果采用常规的Round Robin排序，如图4.4，一般都要迭代14次以上算法才能收敛。数学家们对此又做了大量工作，最终[45]中的一种称为Ring Jacobi Ordering的方法可以使收敛达到最快，其策略如图4.5。

Round Robin 策略可以说是一种万能策略，如图4.4，每一步的每一列为矩阵的某两列坐标的配对，每一步有$n/2$个配对，固定住右上角的列标，其它$n-1$个列标逆时针旋转$n-1$次，就必定得到$n-1$种完全不同的两两配对方式。而Ring Jacobi Ordering 稍微复杂一些，这种策略先按照(a)中的方法将列标配对进行正交化，而后再按(b)中方式

Algorithm 12 (Jacobi SVD算法)

输入n乘n的矩阵A，初始化V为单位阵
while not converged **do**
  for all $i < j$ **do**
    /* 计算A^TA在(i,j)位置的子矩阵$\begin{bmatrix} a & c \\ c & b \end{bmatrix}$*/
    $a = \sum_{k=1}^{n} A_{ki}^2,$
    $b = \sum_{k=1}^{n} A_{kj}^2,$
    $c = \sum_{k=1}^{n} A_{ki} * A_{kj},$
    /* 计算Jacobi旋转的参数从而对角化$\begin{bmatrix} a & c \\ c & b \end{bmatrix}$*/
    $\zeta = (b-a)/(2c),$
    $t = \mathrm{sign}(\zeta)/(|\zeta| + \sqrt{1+\zeta^2}),$
    $cs = 1/\sqrt{1+t^2},$
    $sn = cs * t,$
    /*更新A的第i列和第j列*/
    for $k = 1 : n$ **do**
      $tmp = G_{ki},$
      $G_{ki} = cs * tmp - sn * G_{kj},$
      $G_{kj} = sn * tmp + cs * G_{kj},$
    end for
    /*更新V的第i列和第j列*/
    for k=1:n **do**
      $tmp = V_{ki},$
      $V_{ki} = cs * tmp - sn * V_{kj},$
      $V_{kj} = sn * tmp + cs * V_{kj},$
    end for
  end for
end while
A的列向量的范数即为奇异值，A归一化以后的矩阵为左奇异矩阵，V为右奇异矩阵

进行正交化，[45]中说明这样的方法其实等同于某种Round Robin策略，并且启发式地说明了它可以使雅可比奇异值分解收敛非常快。我们在GPU上利用此策略进行并行正

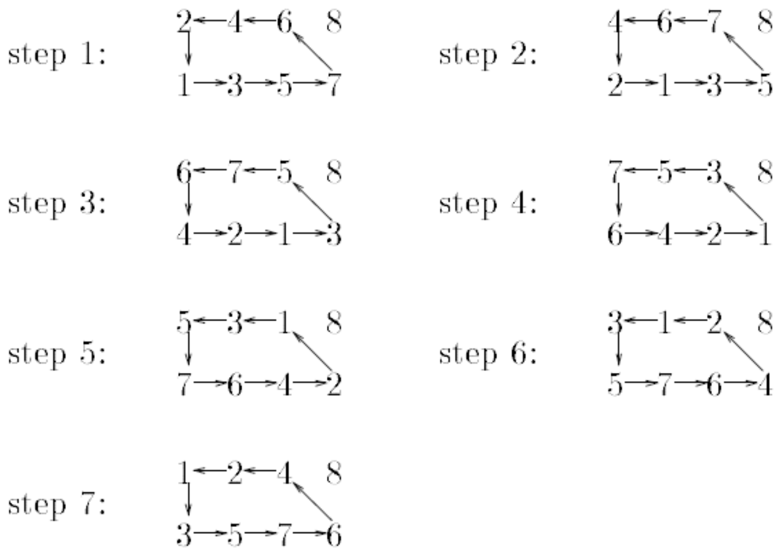

图 4.4: **Round Robin Ordering**
列数为8的矩阵的列标组合策略，每一步同一列中的两个坐标形成一个配对，每步有4组配对，循环7次穷尽所有28种两两配对。

交化，一般迭代11～13次算法即可达到收敛。

4.2.3 预处理和优化

前面描述了最基本的雅可比奇异值分解，很多文章中指出，对输入的矩阵进行QR预处理[17][18][16]，算法的收敛可以变快。我们采用如下形式的QR预处理，使算法的收敛相对变快，一般迭代9～11次即可收敛：

$$\begin{aligned} &[Q, R] = \mathrm{qr}(A), \\ &[U_r, S_r, V_r] = \mathrm{svd}(R^T), \\ &V = U_r; S = S_r; U = AVS^{-1}. \end{aligned} \tag{4.1}$$

其中，在对一个$m\times n$的矩阵A做QR分解时，我们通过在GPU上实现Householder变换[23][20][26]进行，见算法13。

另外在正交化过程中，A^TA对角线附近的0总是最难消除，即A比较相邻的两列间正交化进程最慢。我们利用一种新的优化方案，即对比较相邻的列进行额外的正交化。实验表明，我们对列标号相距$n/32$范围内的两列作额外的正交化性价比最高，一般迭代7～9次算法即可收敛。

需要说明的是雅可比奇异值分解的最后一轮迭代其实进行的正交化操作很少，大部分时间都是用于计算两列的内积，而后根据正交化程度确定是否继续做正交。所以，

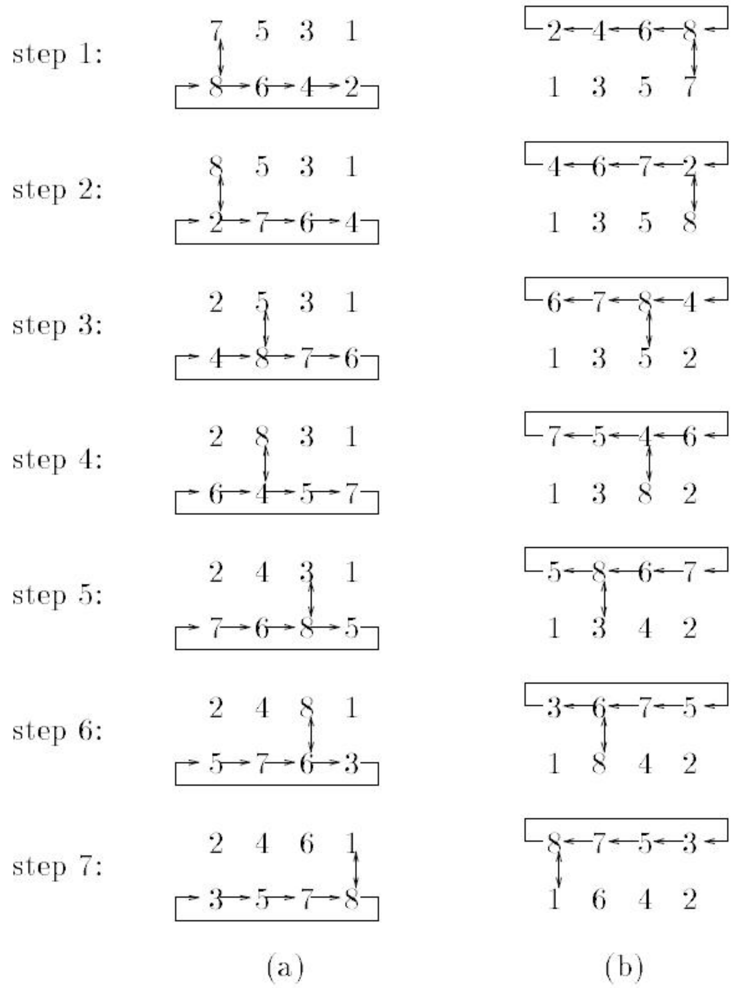

图 4.5: **Ring Jacobi Ordering**

列数为8的排序策略，先按左侧顺序进行两两配对正交化，

再按右侧顺序进行两两配对正交化，可达到快速收敛。

Algorithm 13 (用Householder变换实现QR分解)

$Q = I$;
for $k = 1$ *to* n **do**
  $[v, \beta] = \text{house}(A(k:m, k))$,
  $A(k:m, k:n) = A(k:m, k:n) - \beta v v^H A(k:m, k:n)$,
  $Q(1:m, k:m) = Q(1:m, k:m) - \beta Q(1:m, k:m) v v^H$,
end for

对于有些对奇异值分解的精度要求不是很高的应用，最后一轮迭代可以不做。利用我们前面的预处理和优化方案，一般固定地迭代6次算法也能够得到足够精确的解。

4.2.4 GPU实现

利用前面几小节讨论的并行策略以及预处理和优化方法，Jacobi SVD的GPU实现的主函数代码如图4.6所示。

4.2.5 仿真与比较

首先，由于雅可比SVD不需要进行二对角化，其精度相对较高，我们在GPU实现中利用双精度浮点数计算，可以达到比较高的精度。对于生成的随机矩阵，在Nvidia GTX 280上我们统计出对于不同维数的方阵我们并行算法的迭代次数、计算时间、以及$\frac{\|USV^T-A\|_F}{\|A\|_F}$、$\frac{\|U^TU-I\|_F}{\|I\|_F}$、$\frac{\|V^TV-I\|_F}{\|I\|_F}$ 三组后向误差，如表4.1。从中可以看出，我们的Jacobi SVD的GPU实现的确达到了很高的精度，并且利用我们策略以后，算法在随机矩阵上7次就可收敛。

表 4.1: **GPU上Jacobi SVD的运算时间和精度.**

m	#Iter	Time(s)	$\frac{\|USV^T-A\|_F}{\|A\|_F}$	$\frac{\|U^TU-I\|_F}{\|I\|_F}$	$\frac{\|V^TV-I\|_F}{\|I\|_F}$
256	7	0.17	2.7e-12	6.8e-11	2.8e-12
512	7	0.47	3.8e-12	3.2e-10	3.9e-12
1024	7	2.35	5.9e-12	6.7e-10	5.7e-12
2048	7	14.97	8.7e-12	6.6e-10	8.2e-12

在GPU上进行奇异值分解的并行实现的工作的主要有[28]和[44]。其中[44]也是一种基于单边雅可比的并行实现，但是它并没有用最好的并行排序策略，并且没有做QR预处理，所以速度并不快。[28] 是一种基于QR迭代的奇异值分解，它先用Householder变换[20]将矩阵进行二对角化，再将二对角矩阵利用隐式的带位移的QR方法[20]进行对角化从而得到原矩阵的奇异值分解。在文章[28]中，GTX 280与双核CPU的速度比较中提速了60倍，我们将我们的GPU实现与其进行一个比较。

[28]中给出了对于10次随机生成矩阵的QR迭代奇异值分解的运算时间的平均值，我们也用同样的方式随机生成矩阵10次，得出进行并行雅可比奇异值分解的运算时间的平均值[1]。图4.7和图4.8中我们将我们的并行实现与[28]中基于QR迭代的SVD算法的并行版本相比，从图中可以看出，对于方阵，我们的雅可比并行实现可以快10%。对于行数为8K的条形矩阵，我们的并行实现可以快5倍以上。对于这种比较窄的矩阵，我们的算法之所以快更多的原因是我们只需对原矩阵进行QR分解，并且只需要更新R即可，Q不需要得到因为我们的左奇异矩阵是后验得到的，而[28]中需要将矩阵进行二对角化，所以它的计算量会大很多。

[1]由于无法得到基于QR迭代的SVD 的GPU实现的源代码，我们在同样的硬件和实验环境下与[28]中的实验数据进行对比。

```
int JacobiSvd(double *d_G, double *d_S, double *d_R, double *d_Y, int M, int N,
                        int *dp, int *h, int *h_mark, int *d_mark, bool swap)
{
    int i = 0;
    int iter = 0;
    int maxIter = 20;
    int halfN = (N + 1) / 2;
    int fullN = halfN * 2;
    bool mark = 1;
    // qr preprocessing
    qr_householder( d_G, d_R, d_S, M, N);
    cudaThreadSynchronize();
    // implicitly compute ai' * ai
    product<<< N, BLOCK_SIZE>>> (d_R, d_S, N, N);
    cudaThreadSynchronize();
    while(mark && iter < maxIter) {
        // initial mark
        mark = 0;
        cublasSetVector( halfN, sizeof(h[0]), h, 1, d_mark, 1);
        // orthogonalize every column pair
        for( i = 0; i < fullN - 1; i ++) {
            ParallelOrthogonalize<<< halfN, BLOCK_SIZE>>> (d_R, dp + i * fullN, d_mark, N, fullN-1, d_S );
        }
        for( int k = N / 32; k >= 1 ; k-- )  {
            quasi_cyclic<<<N/2, BLOCK_SIZE>>>(d_R, d_mark, N, N, d_S, k);
            quasi_cyclic2<<<N/2, BLOCK_SIZE>>>(d_R, d_mark, N, N, d_S, k);
        }
        // get mark
        cublasGetVector( halfN, sizeof(h_mark[0]), d_mark, 1, h_mark, 1);
        for( i = 0; i < halfN; i++)     {
            if( h_mark[i] == 1) {
                mark = 1;
                break;
            }
        }
        iter++;
    }
    // normalize
    Normalize<<< N, BLOCK_SIZE>>>(d_R, d_S, N, N);
    cudaThreadSynchronize();
    // posterior compute U .ie. d_G
    cublasDgemm( 'n', 'n', M, N, N, 1.0, d_Y, M, d_R, N, 0.0, d_G, M);
    Posterior<<< N, BLOCK_SIZE>>>( d_G, d_S, M, N);
    cudaThreadSynchronize();
    return iter;
}
```

图 4.6: **Jacobi SVD的GPU实现主函数代码**

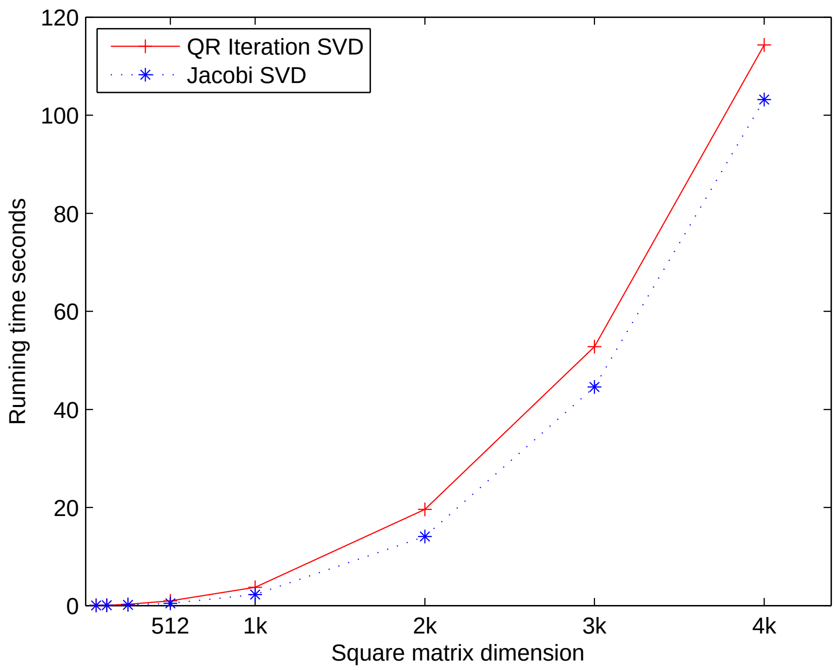

图 4.7: **Jacobi方法与QR迭代法的并行实现在方阵上的比较**

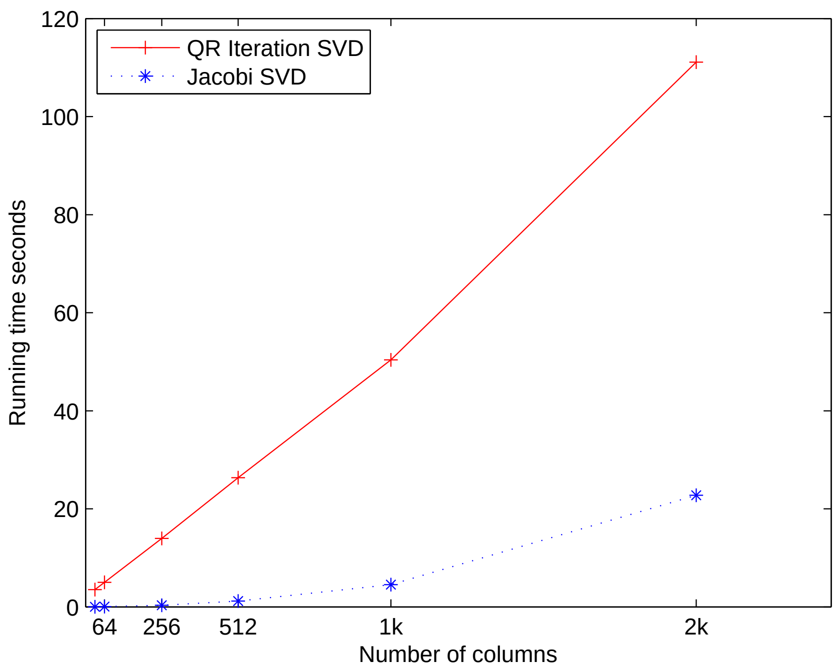

图 4.8: **Jacobi方法与QR迭代法在行数为8K的条形矩阵上的比较**

4.3　矩阵恢复并行算法的设计与实现

前面我们已经给出了矩阵恢复的一种快速收敛的增广拉格朗日乘子法，针对每一步迭代的关键操作（部分）奇异值分解，给出了一种改进的算法，同时在GPU上实现了雅可比奇异值分解的并行算法。在并行实现部分奇异值分解时，由于其大量计算矩阵和向量的乘积，所以我们直接调用了CUBLAS中的库函数来解决这些矩阵运算。实现矩阵恢复算法时，主要将算法中的收缩算子等操作写成kernel，其它矩阵乘法和矩阵向量积等运算同样调用CUBLAS中的函数。奇异值分解则调用上一小节已经实现的雅可比并行算法。GPU实现的流程如图4.9。

最终我们的并行实现算法与CPU上的串行算法的比较如表4.2所示。进行比较的GPU和CPU分别是Nvidia GTX 280（拥有1GB显存，安装在Intel Pentium4 3.0GHz的主机上）和Intel Xeon E5540 2.53GHz（四核，64位处理器，安装64位操作系统，24GB内存）。GPU上算法是利用雅可比奇异值分解，CPU上则为MATLAB实现(Matlab版本R2009b)。从中可以发现，我们的并行实现在维数越大时加速比越高，在4K的方阵上可达超过8倍的提速。

表 4.2: **GPU和CPU矩阵恢复算法的仿真数据比较.**

m	algorithm	$\frac{\\|\hat{A}-A^*\\|_F}{\\|A^*\\|_F}$	rank($\hat{A}$)	$\\|\hat{E}\\|_0$	#SVD	time (s)
rank$(A^*) = 0.25\,m$, $\\|E^*\\|_0 = 0.01\,m^2$						
512	CPU	2.52e-6	128	2621	23	21.99
	GPU	2.13e-6	128	2621	23	10.53
1024	CPU	1.95e-6	256	10486	23	172.41
	GPU	2.02e-6	256	10486	23	53.08
2048	CPU	1.42e-6	512	41950	24	2142.6
	GPU	1.02e-6	512	41942	24	354.43
4096	CPU	1.36e-6	1024	167780	24	22223.0
	GPU	9.19e-7	1024	167816	24	2654.0

在实际数据上，我们继续以背景建模为例，102帧720×576大小的视频，选取其中的60帧进行背景建模。表4.3中为GPU和CPU的运算时间、迭代次数和最终结果，我们可以看到结果几乎完全一样，GPU并行实现相对CPU的加速比达到10倍。

4.4　集群上的矩阵恢复算法的实现简介

实际应用的矩阵往往规模巨大，特别是一些文本数据和搜索数据矩阵，动辄几百

表 4.3: **GPU与CPU矩阵恢复算法在视频数据上的比较.**

size	Hardware	#Iter	Time(s)	rank($\hat{A}$)
414720×60	*Intel Xeon E5540*	37	248.23	23
414720×60	*Nvidia GTX 280*	37	24.65	23

万的数量级。这种情况下，就需要利用集群来进行处理。在设计集群上的矩阵恢复算法时，需尽可能减少对磁盘读写数据，使通讯开销最少。如图4.10所示，我们重新设计矩阵恢复算法，右侧框中的操作写成一个函数，部分奇异值分解写成另一个函数同时尽可能减小每步操作的磁盘读写，使算法从一开始的26次读、18次写减小为4次读、2次写操作。

表4.4为集群上的矩阵恢复算法的运行情况，表明了不同的矩阵上所用的节点数以及其运行时间。注意这两组数据运行在不同的集群上，所以性能会有所不同。一般而言，单机上由于内存限制，可以处理的矩阵规模不超过10000×10000，而集群可以使能处理的规模得到很大提高。并且对于大规模数据，集群上的实现总体上表现得很稳定。

表 4.4: **集群上的矩阵恢复算法的运行情况.**

size	#Nodes	Run(hours)	Queue(hours)	rank($\hat{A}$)
50000×50000	100	47.7	28.8	501
42186×18320	40	90.3	33.2	355

4.5 本章小结

本章主要介绍了奇异值分解和矩阵恢复算法的并行实现。其中奇异值分解利用雅可比并行算法实现，因为这个算法非常适合并行，同时我们对其进行了一些预处理和优化，使得收敛速度提高很多。而矩阵恢复算法则在奇异值分解的基础上加入一些矩阵向量运算的并行实现，最终使算法整体上得到提速。矩阵恢复在集群上的实现则使得能处理的矩阵的规模不再受到单机内存的限制。

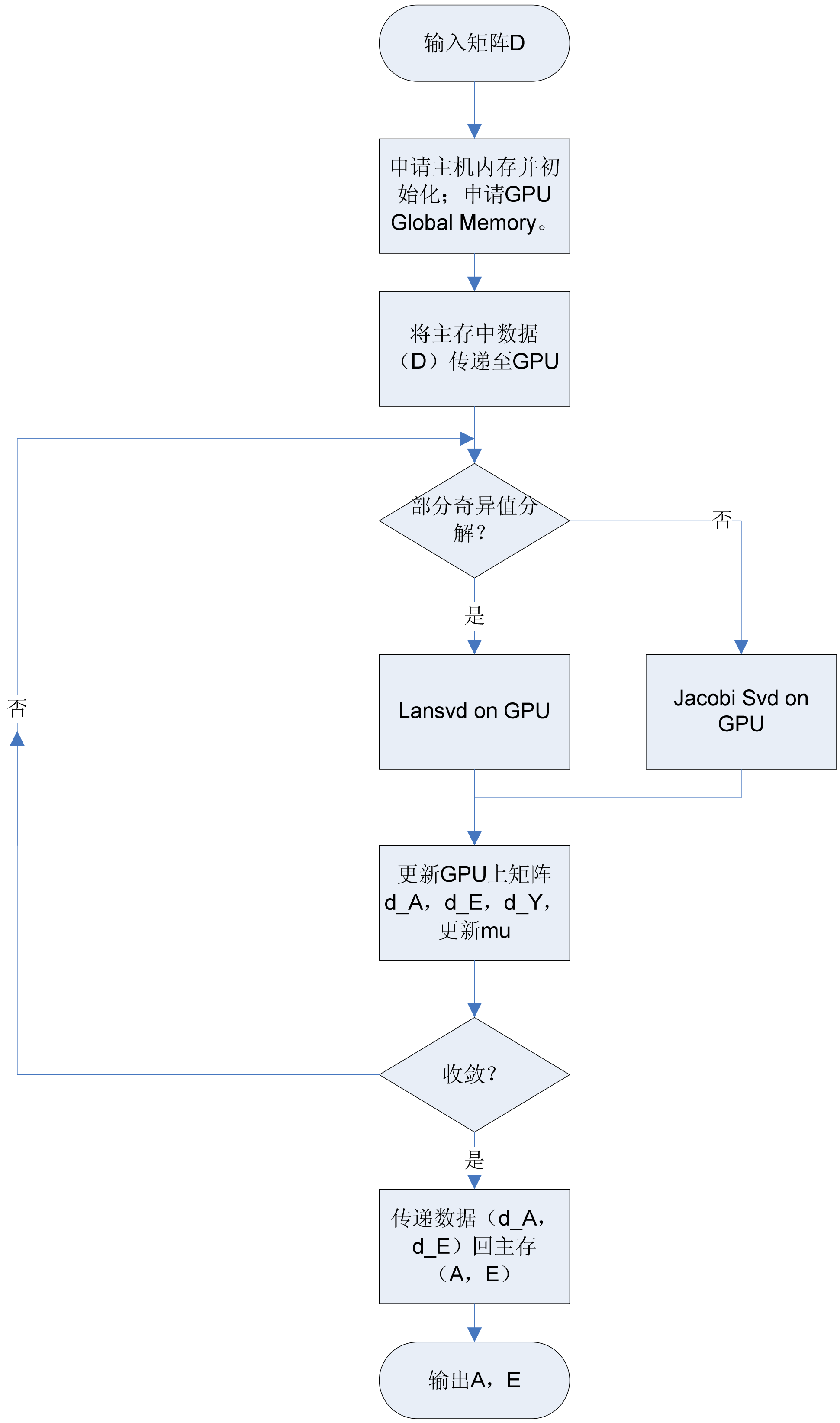

图 4.9: **矩阵恢复的GPU实现**

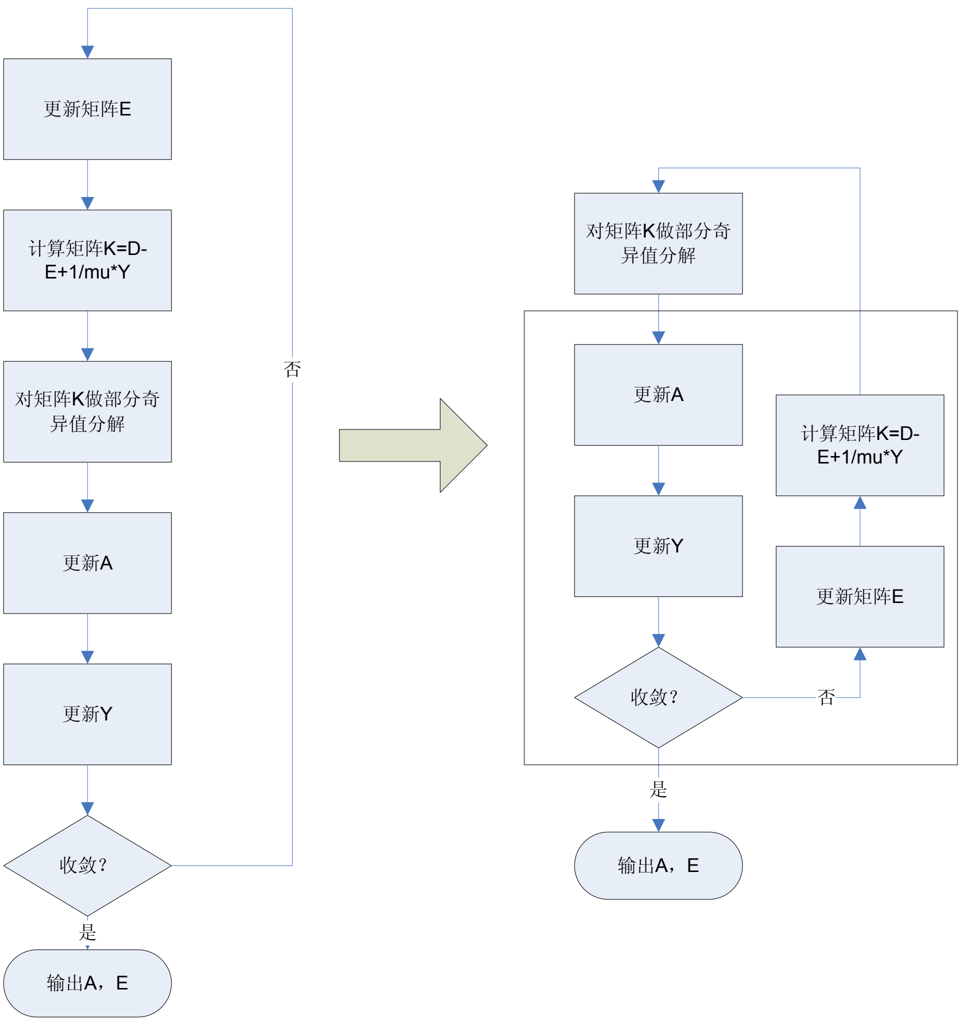

图 4.10: 矩阵恢复算法在集群上的重新设计
左图为改进前的矩阵恢复算法，右图为改进后的适合于集群实现的矩阵恢复算法。

第五章　矩阵重建问题的一些变种

在很多实际应用问题中，经常需要对原始矩阵填充和恢复问题加一些约束，我们在这里讨论和分析其中比较典型的一些变种，给出相应的算法。

5.1　矩阵填充并恢复

有一类典型的问题是矩阵元素丢失，而同时又有一些已知的元素被严重破坏了，如在搜索领域，往往很多特征缺失，同时又有数据严重受损，这类问题是矩阵填充与矩阵恢复的结合。我们可以写成如下形式：

$$\begin{aligned} \min \quad & \|A\|_* + \lambda\|P_\Omega(E)\|_1, \\ \text{subject to} \quad & A + E = D. \end{aligned} \tag{5.1}$$

其增广拉格朗日函数为：

$$L(A, E, Y, \mu) = \|A\|_* + \lambda\|P_\Omega(E)\|_1 + \langle Y, D - A - E\rangle + \frac{\mu}{2}\|D - A - E\|_F^2. \tag{5.2}$$

更新A时，

$$\arg\min_A \|A\|_* + \frac{\mu}{2}\|D - A - E + \mu^{-1}Y\|_F^2 = \mathcal{D}_{\mu^{-1}}(D - E + \mu^{-1}Y); \tag{5.3}$$

更新E时，

$$\begin{aligned} &\arg\min_E \lambda\|P_\Omega(E)\|_1 + \frac{\mu}{2}\|D - A - E + \mu^{-1}Y\|_F^2 = \\ &P_\Omega(\mathcal{S}_{\frac{\lambda}{\mu}}(D - A + \mu^{-1}Y)) + P_{\bar{\Omega}}(D - A + \mu^{-1}Y). \end{aligned} \tag{5.4}$$

于是我们得到算法14以解决有缺失数据的矩阵恢复问题。

Algorithm 14 (有数据缺失的矩阵恢复算法)

1: 初始化Y_0, $E_0 = 0$, μ_0, $k = 0$,
2: **while** not converged **do**
3: $(U, S, V) = \text{svd}(D - E_k + \mu_k^{-1}Y_k)$,
4: $A_{k+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T$,
5: $E_{k+1} = P_\Omega(\mathcal{S}_{\frac{\lambda}{\mu_k}}(D - A_{k+1} + \mu_k^{-1}Y_k)) + P_{\bar{\Omega}}(D - A_{k+1} + \mu_k^{-1}Y_k)$,
6: $Y_{k+1} = Y_k + \mu_k(D - A_{k+1} - E_{k+1})$,
7: 更新μ_k,
8: $k = k + 1$.
9: **end while**

5.2 有噪声的矩阵恢复

在实际应用中噪声总是不可避免，这里我们讨论一下一种更为稳定的矩阵恢复，即在有些元素被严重损坏时，每个元素收到一些轻量级的噪声污染，这种噪声一般用F范数来刻画。问题可以写成如下形式：

$$\begin{aligned} \min \quad & \|A\|_* + \lambda\|E\|_1 + \gamma\|Z\|_F^2, \\ \text{subject to} \quad & A + E + Z = D. \end{aligned} \tag{5.5}$$

其增广拉格朗日函数为：

$$\begin{aligned} & L(A,E,Z,Y,\mu) = \\ & \|A\|_* + \lambda\|E\|_1 + \gamma\|Z\|_F^2 + \langle Y, D-A-E-Z\rangle + \frac{\mu}{2}\|D-A-E-Z\|_F^2. \end{aligned} \tag{5.6}$$

更新A时，

$$\arg\min_A \|A\|_* + \frac{\mu}{2}\|D-A-E-Z+\mu^{-1}Y\|_F^2 = \mathcal{D}_{\mu^{-1}}(D-E-Z+\mu^{-1}Y); \tag{5.7}$$

更新E时，

$$\arg\min_E \lambda\|E\|_1 + \frac{\mu}{2}\|D-A-E-Z+\mu^{-1}Y\|_F^2 = \mathcal{S}_{\frac{\lambda}{\mu}}(D-A-Z+\mu^{-1}Y); \tag{5.8}$$

更新Z时，

$$\arg\min_Z \gamma\|Z\|_F^2 + \frac{\mu}{2}\|D-A-E-Z+\mu^{-1}Y\|_F^2 = \frac{\mu}{\mu+2\gamma}(D-A-E+\mu^{-1}Y). \tag{5.9}$$

于是我们得到算法15以解决有缺失数据的矩阵恢复问题。

Algorithm 15 (带噪声的矩阵恢复算法)

1: 初始化Y_0, $E_0 = 0$, Z_0, μ_0, $k = 0$,
2: **while** not converged **do**
3: $(U,S,V) = \text{svd}(D - E_k + \mu_k^{-1}Y_k)$,
4: $A_{k+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T$,
5: $E_{k+1} = \mathcal{S}_{\frac{\lambda}{\mu_k}}(D - A_{k+1} - Z_k + \mu_k^{-1}Y_k)$,
6: $Z_{k+1} = \frac{\mu_k}{\mu_k+2\gamma}(D - A_{k+1} - E_{k+1} + \mu_k^{-1}Y_k)$,
7: $Y_{k+1} = Y_k + \mu_k(D - A_{k+1} - E_{k+1} - Z_{k+1})$,
8: 更新μ_k;
9: $k = k+1$.
10: **end while**

5.3　有噪声的矩阵填充

同样，矩阵填充中，那些已知的元素也有可能受到噪声的污染。我们可以写出如下形式：

$$\begin{aligned} \min \quad & \|A\|_* + \gamma \|P_\Omega(Z)\|_F^2, \\ \text{subject to} \quad & A + Z = D. \end{aligned} \tag{5.10}$$

其增广拉格朗日函数为：

$$L(A, E, Z, Y, \mu) = \|A\|_* + \gamma \|P_\Omega(Z)\|_F^2 + \langle Y, D - A - Z \rangle + \frac{\mu}{2} \|D - A - Z\|_F^2. \tag{5.11}$$

更新A时，

$$\arg\min_A \|A\|_* + \frac{\mu}{2} \|D - A - Z + \mu^{-1} Y\|_F^2 = \mathcal{D}_{\mu^{-1}}(D - Z + \mu^{-1} Y); \tag{5.12}$$

更新Z时，

$$\begin{aligned} & \arg\min_Z \gamma \|P_\Omega(Z)\|_F^2 + \frac{\mu}{2} \|D - A - Z + \mu^{-1} Y\|_F^2 = \\ & \frac{\mu}{\mu + 2\gamma} P_\Omega(D - A + \mu^{-1} Y) + P_{\bar{\Omega}}(D - A + \mu^{-1} Y). \end{aligned} \tag{5.13}$$

注意无论Y初始值多少，上述过程中$P_{\bar{\Omega}}(Y)$将始终为0。于是我们得到算法16以解决带噪声的矩阵填充问题。

Algorithm 16 (带噪声的矩阵填充算法)

1: 初始化Y_0, Z_0, μ_0, $k = 0$,
2: **while** not converged **do**
3: $(U, S, V) = \text{svd}(D - E_k + \mu_k^{-1} Y_k)$,
4: $A_{k+1} = U \mathcal{S}_{\mu_k^{-1}}[S] V^T$,
5: $Z_{k+1} = \frac{\mu_k}{\mu_k + 2\gamma}(D - A_{k+1} + \mu_k^{-1} Y_k) + P_{\bar{\Omega}}(D - A_{k+1})$,
6: 更新μ_k,
7: $k = k + 1$.
8: **end while**

注意这个算法实现时可以利用矩阵低秩和稀疏性质，有精简的表达和计算方式。

5.4　有噪声的矩阵填充并恢复

现在我们可以考虑一种更通用的形式，矩阵中既有数据缺失，而在已知的数据中既有元素被严重损坏，每个元素又收到一些噪声的影响。我们可以写出如下形式：

$$\begin{aligned} \min \quad & \|A\|_* + \lambda \|P_\Omega(E)\|_1 + \gamma \|P_\Omega(Z)\|_F^2, \\ \text{subject to} \quad & A + E + Z = D. \end{aligned} \tag{5.14}$$

其增广拉格朗日函数为：

$$
\begin{aligned}
&L(A,E,Z,Y,\mu)= \\
&\|A\|_* + \lambda\|P_\Omega(E)\|_1 + \gamma\|P_\Omega(Z)\|_F^2 + \langle Y, D-A-E-Z\rangle + \frac{\mu}{2}\|D-A-E-Z\|_F^2.
\end{aligned}
\tag{5.15}
$$

更新A时，

$$
\arg\min_A \|A\|_* + \frac{\mu}{2}\|D-A-E-Z+\mu^{-1}Y\|_F^2 = \mathcal{D}_{\mu^{-1}}(D-E-Z+\mu^{-1}Y); \tag{5.16}
$$

更新E时，

$$
\arg\min_E \lambda\|E\|_1 + \frac{\mu}{2}\|D-A-E-Z+\mu^{-1}Y\|_F^2 = \mathcal{S}_{\frac{\lambda}{\mu}}(D-A-Z+\mu^{-1}Y); \tag{5.17}
$$

更新Z时，

$$
\begin{aligned}
&\arg\min_Z \gamma\|P_\Omega(Z)\|_F^2 + \frac{\mu}{2}\|D-A-E-Z+\mu^{-1}Y\|_F^2 = \\
&\frac{\mu}{\mu+2\gamma}P_\Omega(D-A-E+\mu^{-1}Y) + P_{\bar\Omega}(D-A-E+\mu^{-1}Y).
\end{aligned}
\tag{5.18}
$$

于是我们得到算法17解决带噪声的矩阵填充同时恢复的问题。

Algorithm 17 (带噪声的矩阵填充并恢复)

初始化Y_0, $E_0=0$, Z_0, μ_0, $k=0$,
while not converged **do**
  $(U,S,V)=\text{svd}(D-E_k+\mu_k^{-1}Y_k)$,
  $A_{k+1}=U\mathcal{S}_{\mu_k^{-1}}[S]V^T$,
  $E_{k+1}=\mathcal{S}_{\frac{\lambda}{\mu_k}}(D-A_{k+1}-Z_k+\mu_k^{-1}Y_k)$,
  $Z_{k+1}=\frac{\mu_k}{\mu_k+2\gamma}(D-A_{k+1}-E_{k+1}+\mu_k^{-1}Y_k)+P_{\bar\Omega}(D-A_{k+1}-E_{k+1}+\mu_k^{-1}Y_k)$,
  $Y_{k+1}=Y_k+\mu_k(D-A_{k+1}-E_{k+1}-Z_{k+1})$,
  更新μ_k,
  $k=k+1$.
end while

5.5 A非负时的矩阵恢复

在很多情况下，经常要求恢复出来的矩阵满足一定的约束，如图像处理中会要求像素值大于0。于是我们得到以下变形：

$$
\begin{aligned}
\min &\quad \|A\|_* + \lambda\|E\|_1, \\
\text{subject to} &\quad A+E=D, A\geq 0.
\end{aligned}
\tag{5.19}
$$

由于求解E比求解A容易得多，我们将原问题转换为如下形式：

$$\begin{aligned} &\min\ \|A\|_* + \lambda\|E\|_1, \\ &\text{subject to } A + E = D, E \leq D. \end{aligned} \tag{5.20}$$

其带约束$E \leq D$的部分增广拉格朗日函数为：

$$L(A, E, Y, \mu) = \|A\|_* + \langle Y, D - A - E\rangle + \frac{\mu}{2}\|D - A - E\|_F^2. \tag{5.21}$$

每次更新A时，

$$\arg\min_A \|A\|_* + \frac{\mu}{2}\|D - A - E + \mu^{-1}Y\|_F^2 = \mathcal{D}_{\mu^{-1}}(D - E + \mu^{-1}Y); \tag{5.22}$$

更新E时，

$$\arg\min_{E \leq D} \|D - A - E + \mu^{-1}Y\|_F^2 = P_{\leq D}(\mathcal{S}_{\lambda/\mu}(D - A + \mu^{-1}Y)). \tag{5.23}$$

其中$P_{\leq D}$表示将一个矩阵投影到每个元素都比D中相应元素小的区域。于是得到解决A非负的矩阵恢复问题的算法18。

Algorithm 18 (A非负时的矩阵恢复算法)

1: $Y_0 = 0,\ E_0 = 0,\ \mu_0 = 0,\ k = 0,$
2: **while** not converged **do**
3: $\quad (U, S, V) = \mathrm{svd}(D - E_k + \mu_k^{-1}Y_k),$
4: $\quad A_{k+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T,$
5: $\quad E_{k+1} = P_{\leq D}(\mathcal{S}_{\lambda/\mu}(D - A_{k+1} + \mu_k^{-1}Y_k)),$
6: $\quad Y_{k+1} = Y_k + \mu_k(D - A_{k+1} - E_{k+1}),$
7: $\quad$ 更新 $\mu_k,$
8: $\quad k = k + 1.$
9: **end while**

5.6 E非负时的矩阵恢复

在有些场合，E往往被视为有用信息，它表现出对象之间的不同之处，可以将各个对象之间的独特属性提取出来。于是我们会遇到要求E非负的矩阵恢复，它可以写成如下形式：

$$\begin{aligned} &\min \quad \|A\|_* + \lambda\|E\|_1, \\ &\text{subject to} \quad A + E = D, E \geq 0. \end{aligned} \tag{5.24}$$

其带约束$E \geq 0$的部分增广拉格朗日函数为：

$$L(A, E, Y, \mu) = \|A\|_* + \langle Y, D - A - E\rangle + \frac{\mu}{2}\|D - A - E\|_F^2. \tag{5.25}$$

每次更新A时，

$$\arg\min_{A} \|A\|_* + \frac{\mu}{2}\|D - A - E + \mu^{-1}Y\|_F^2 = \mathcal{D}_{\mu^{-1}}(D - E + \mu^{-1}Y), \tag{5.26}$$

更新E时，

$$\arg\min_{E\geq 0} \|D - A - E + \mu^{-1}Y\|_F^2 = P_+(\mathcal{S}_{\lambda/\mu}(D - A + \mu^{-1}Y)), \tag{5.27}$$

其中P_+表示将一个矩阵所有小于0的元素都置0。于是得到解决E非负时的矩阵恢复问题的算法19。

Algorithm 19 (E**非负时的矩阵恢复算法**)

1: $Y_0 = 0,\ E_0 = 0,\ \mu_0 = 0,\ k = 0,$
2: **while** not converged **do**
3: $\quad (U, S, V) = \mathrm{svd}(D - E_k + \mu_k^{-1}Y_k),$
4: $\quad A_{k+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T,$
5: $\quad E_{k+1} = P_+(\mathcal{S}_{\lambda/\mu}(D - A_{k+1} + \mu_k^{-1}Y_k)),$
6: $\quad Y_{k+1} = Y_k + \mu_k(D - A_{k+1} - E_{k+1}),$
7: $\quad$更新 $\mu_k,$
8: $\quad k = k + 1.$
9: **end while**

5.7 A非负时的矩阵填充

同样，会有一些矩阵填充问题要求填进的元素值非负，我们将其写成如下形式：

$$\begin{aligned} &\min \quad \|A\|_*, \\ &\text{subject to } A + E = D,\ P_\Omega(E) = 0,\ A \geq 0, \end{aligned} \tag{5.28}$$

由于求解E比求解A容易得多，我们将上式转换成：

$$\begin{aligned} &\min \quad \|A\|_*, \\ &\text{subject to } A + E = D,\ P_\Omega(E) = 0,\ P_{\bar{\Omega}}(E) \leq P_{\bar{\Omega}}(D). \end{aligned} \tag{5.29}$$

其带约束$P_\Omega(E) = 0, P_{\bar{\Omega}}(E) \leq P_{\bar{\Omega}}(D)$的部分增广拉格朗日函数为：

$$L(A, E, Y, \mu) = \|A\|_* + \langle Y, D - A - E\rangle + \frac{\mu}{2}\|D - A - E\|_F^2. \tag{5.30}$$

每次更新A时，

$$\arg\min_{A} \|A\|_* + \frac{\mu}{2}\|D - A - E + \mu^{-1}Y\|_F^2 = \mathcal{D}_{\mu^{-1}}(D - E + \mu^{-1}Y); \tag{5.31}$$

更新E时，

$$\arg\min_{P_\Omega(E)=0, P_{\bar{\Omega}}(E)\leq P_{\bar{\Omega}}(D)} \|D - A - E + \mu^{-1}Y\|_F^2 = P_{\bar{\Omega}}(P_{\leq D}(D - A + \mu^{-1}Y)). \tag{5.32}$$

于是得到矩阵恢复的增广拉格朗日乘子法如算法20。

Algorithm 20 (**A非负时的矩阵填充算法**)

$Y_0 = 0,\ E_0 = 0,\ \mu_0 = 0,\ k = 0,$
while not converged **do**
　$(U, S, V) = \mathrm{svd}(D - E_k + \mu_k^{-1} Y_k),$
　$A_{k+1} = U \mathcal{S}_{\mu_k^{-1}}[S] V^T,$
　$E_{k+1} = P_{\bar{\Omega}}(P_{\le D}(D - A_{k+1} + \mu_k^{-1} Y_k)),$
　$Y_{k+1} = Y_k + \mu_k (D - A_{k+1} - E_{k+1}),$
　更新 μ_k,
　$k = k + 1.$
end while

5.8　仿真实验

由于矩阵重建的变种问题众多，此处我们举矩阵填充并恢复这一典型情况（5.1节）进行仿真实验。对于既有元素缺失，又有元素被严重破坏的情况，我们首先利用前面的模型生成秩为r，大小为m的低秩方阵。而后根据给定的采样率sr对矩阵元素进行均匀采样，再按照比率ρ_s对采样到的元素施加大小在$[-500, 500]$ 之间的均匀随机噪声。对此矩阵进行矩阵的填充并且恢复，实验结果如表5.1。从中可以看出，当采样元素越少，迭代次数会随之增加，一般迭代三十次可以收敛到非常精确的解。

5.9　本章小结

有些应用中遇到的问题往往不完全符合预定的模型，所以本章讨论和分析了矩阵填充和矩阵恢复问题的一系列变种。我们前面提出的增广拉格朗日乘子法是一种扩展性非常好的算法，我们基于此方法给出了各个变种相应的算法，这些变种在实际问题中经常会碰到。

表 5.1: **矩阵填充并恢复仿真实验.**

m	method	sr	$\frac{\Vert\hat{A}-A_0\Vert_F}{\Vert A_0\Vert_F}$	rank($\hat{A}$)	iter	time(s)
$\rho_r = \mathrm{rank}(A_0)/m = 0.05,\ \rho_s = 0.05$						
200	IALM	0.9	3.42e-5	10	19	1.56
		0.8	3.11e-5	10	23	1.90
		0.7	3.37e-5	10	26	1.81
		0.6	3.44e-5	10	29	1.96
400	IALM	0.9	2.28e-5	20	23	4.96
		0.8	2.27e-5	20	22	5.42
		0.7	2.58e-5	20	25	5.68
		0.6	2.58e-5	20	29	5.85
800	IALM	0.9	1.53e-5	40	20	26.66
		0.8	1.74e-5	40	22	26.59
		0.7	1.94e-5	40	25	27.12
		0.6	1.93e-5	40	30	29.75

第六章　总结与展望

6.1　总结

矩阵重建是信号处理、人工智能和优化领域人们最近研究的热点，并已找到了不少应用。本文分析了国内外的研究现状，从中发现存在的一些问题：现有方法计算量大、速度慢、所能处理矩阵规模小；实际数据并不完全符合预定的模型。本文的工作主要集中在解决矩阵重建中的一些计算问题，针对其算法和实现提出了一整套解决办法，主要内容包括：

（1）给出了一种增广拉格朗日乘子法更快地求解矩阵恢复问题。此算法较以往主流算法速度提升明显，并且实现方便，占用内存少，有极好的扩展性，可以应用于矩阵重建的各个变种。

（2）给出了矩阵填充的增广拉格朗日乘子法。这个算法较以往算法提速明显，消耗内存极少，适合于大规模数据的处理和分析。

（3）给出了一种改进的部分奇异值分解算法，它能够直接计算大于某一阈值的奇异值分解，可以直接应用于矩阵重建问题中并且很好地工作。同时，它也可以应用于所有需要使用此类型部分奇异值分解的算法中。

（4）通过仔细研究Jacobi SVD的现有工作并进行大量实验，给出了一套新的预处理和优化方案。实验表明，这套方案可以提高算法的收敛速度，并且适合在具有并行计算能力的硬件上实现。

（5）设计和实现了奇异值分解和矩阵恢复的GPU并行算法。这个实现可以进一步提升实际应用中对矩阵的处理速度。

（6）参与设计和实现集群上的矩阵恢复算法，以突破单机内存的制约，这使得大规模矩阵处理成为现实。

（7）讨论和分析了矩阵重建的一系列变种，并给出了相应算法，这对于实际应用有重要意义。

6.2　展望

矩阵重建是一种重要的数据分析工具，已经在图像处理、计算机视觉、推荐系统等领域找到了不少应用[39][25][34][31][40]。随着其理论上不断完善，算法上不断优化，以及并行和分布式计算的不断普及，矩阵重建将会进一步在未来的科学研究和工程实践中找到更多应用。下一步的工作主要包括：

（1）进一步完善大规模数据上的矩阵重建算法，使得它能够处理动态变化和更新的矩阵。

（2）与人工智能中现有的方法相结合，利用矩阵重建进行大规模数据的挖掘和预测。

参 考 文 献

[1] A. Beck and M. Teboulle. *A fast iterative shrinkage-thresholding algorithm for linear inverse problems.* SIAM Journal on Imaging Sciences, pp. 183-202, 2008.

[2] M. Becka and G. Oksa. *On variable blocking factor in a parallel dynamic block-Jacobi SVD algorithm.* Parallel Computing, pp. 1153-1174, 2003.

[3] M. Becka, G. Oksa, and M. Vajtersic. *Dynamic ordering for a parallel block-Jacobi SVD algorithm.* Parallel Computing, pp. 243-262, 2002.

[4] J. Bennett and S. Lanning. *The netflix prize.* In Proceedings of KDD Cup and Workshop, 2007.

[5] P. A. Businger and G. H. Golub. *Algorithm 358: singular value decomposition of a complex matrix.* Communications of the ACM, pp. 564-565, 1969.

[6] J. F. Cai, E. J. Candès, and Z. Shen. *A singular value thresholding algorithm for matrix completion.* Preprint, 2008.

[7] E. J. Candès. *Compressive sampling.* In Proceedings of the International Congress of Mathematicians, 2006.

[8] E. J. Candès, X. Li, Y. Ma, and J. Wright. *Robust principal component analysis?* Preprint, 2009.

[9] E. J. Candès and B. Recht. *Exact matrix completion via convex optimization.* Preprint, 2009.

[10] E. J. Candès and J. Romberg. *Sparsity and incoherence in compressive sampling.* Inverse Problems, pp. 969-985, 2006.

[11] E. J. Candès, M. Rudelson, T. Tao, and R. Vershynin. *Error correction via linear programming.* In Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, 2005.

[12] E. J. Candès and T. Tao. *Reflections on compressed sensing.* IEEE Information Theory Society Newsletter, pp. 14-17, 2008.

[13] E. J. Candès and M. Wakin. *An introduction to compressive sampling.* IEEE Signal Processing Magazine, pp. 21-30, 2008.

[14] J. W. Demmel and W. Kahan. *Accurate singular values of bidiagoanl matrices.* SIAM: SIAM Journal on Scientific Computing, pp. 873-912, 1990.

[15] I. Dhillon, J. Demmel, and M. Gu. *Efficient computation of the singular value decomposition with applications to least squares problems.* LBL Report #36201, 1994.

[16] Z. DRMAC. *A posteriori computation of the singular vectors in a preconditioned Jacobi SVD algorithm.* IMA Journal of Numerical Analysis, pp. 191-213, 1999.

[17] Z. Drmac and K. Veselic. *New fast and accurate Jacobi SVD algorithm. I.* SIAM Journal on Matrix Analysis and Applications, pp. 1322-1342, 2007.

[18] Z. Drmac and K. Veselic. *New fast and accurate Jacobi SVD algorithm: II.* SIAM J. Matrix Anal. Appl., pp. 1343-1362, 2008.

[19] A. Ganesh, Z. Lin, J. Wright, L. Wu, M. Chen, and Y. Ma. *Fast algorithms for recovering a corrupted low-rank matrix.* International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, 2009.

[20] G. H. Golub and C. F. Van Loan. *Matrix computations (3rd ed.).* Johns Hopkins University Press, 1996.

[21] D. Gross. *Recovering low-rank matrices from few coefficients in any basis.* Preprint, 2009.

[22] M. Gu and S. C. Eisenstat. *A divide-and-conquer algorithm for the bidiagonal SVD.* SIAM Journal on Matrix Analysis and Applications Volume 16, Number 1, pp. 79-92, 1995.

[23] A. S. Householder. *Unitary triangularization of a nonsymmetric matrix.* Journal of the ACM, pp. 339-342, 1958.

[24] D. James and V. Kresimir. *Jacobi's method is more accurate than QR.* SIAM Journal on Matrix Analysis and Applications, pp. 1204-1245, 1992.

[25] H. Ji, C. Liu, Z. Shen, and Y. Xu. *Robust video denoising using low rank matrix completion.* In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010.

[26] A. Kerr, D. Campbell, and M. Richards. *QR decomposition on GPUs.* In Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, 2009.

[27] R. H. Keshavan, A. Montanari, and S. Oh. *Matrix completion from noisy entries.* Priprint, 2009.

[28] S. Lahabar and P. J. Narayanan. *Singular value decomposition on GPU using CUDA.* IEEE International Symposium on Parallel&Distributed Processing, 2009.

[29] R. M. Larsen. *http://soi.stanford.edu/ rmunk/PROPACK/.* 2004.

[30] Z. Lin, M. Chen, L. Wu, and Y. Ma. *The augmented lagrange multiplier method for exact recovery of a corrupted low-rank matrix.* Preprint, 2009.

[31] Z. Liu and L. Vandenberghe. *Semidefinite programming methods for system realization and identification.* In Proceedings of IEEE Conference on Decision and Control, 2009.

[32] S. Ma, D. Goldfarb, and L. Chen. *Fixed point and bregman iterative methods for matrix rank minimization.* Preprint, 2009.

[33] R. Meka, P. Jain, and I. S. Dhillon. *Matrix completion from power-law distributed samples.* In Proceedings of the Conference on Neural Information Processing Systems (NIPS), 2009.

[34] J. Meng, W. Yin, E. Houssain, and Z. Han. *Collaborative spectrum sensing from sparse observations using matrix completion for cognitive radio networks.* In Proceedings of International Conference on Acoustics, Speech, and Signal Processing, 2010.

[35] S. Negahban, P. Ravikumar, M. Wainwright, and B. Yu. *A unified framework for high-dimensional analysis of m-estimators with decomposable regularizers.* In Proceedings of the Conference on Neural Information Processing Systems (NIPS), 2009.

[36] NVIDIA. *CUDA CUBLAS library.* NVIDIA Corporation, 2009.

[37] NVIDIA. *NVIDIA CUDA programming guide.* 2009.

[38] C. C. Paige and M. A. Saunders. *LSQR: an algorithm for sparse linear equations and sparse least squares.* ACM Transactions on Mathematical Software, pp. 43-71, 1982.

[39] Y. Peng, A. Ganesh, J. Wright, W. Xu, and Y. Ma. *RASL: robust alignment by sparse and low-rank decomposition for linearly correlated images.* In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010.

[40] J. Rennie and N. Srebro. *Fast maximum margin matrix factorization for collaborative prediction.* International Conference on Machine Learning (ICML), 2005.

[41] A. Singer and M. Cucuringu. *Uniqueness of low-rank matrix completion by rigidity theory.* SIAM Journal on Matrix Analysis and Applications, pp. 1621-1641, 2010.

[42] K.C. Toh and S.W. Yun. *An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems.* Accepted by Pacific J. Optimization, 2009.

[43] J. Wright, A. Ganesh, S. Rao, Y. Peng, and Y. Ma. *Robust principal component analysis: exact recovery of corrupted low-rank matrices via convex optimization.* In Proceedings of the Conference on Neural Information Processing Systems (NIPS), 2009.

[44] S. Zhang. *One sided jacobi method on CUDA for SVD.* Application Research of computers.

[45] B. B. Zhou and R. P. Brent. *A parallel ring ordering algorithm for efficient one-sided Jacobi SVD computations.* Journal of Parallel and Distributed Computing, pp. 1-10, 1997.

[46] A. Zymnis, S. Boyd, and E. J. Candès. *Compressed sensing with quantized measurements.* Signal Processing Letters 17(3), pp. 149-152, 2010.

致　谢

首先我要特别感谢我的导师林宙辰老师和沈向洋老师。林老师和沈老师都是特别尊重学生想法的老师，同时更是为学生着想的老师。感谢林老师在学习和科研的各个方面对我细心和耐心的指导，这对我的帮助十分巨大。感谢沈老师总是在关键时刻给我一些非常好的建议，为我的学习和研究指明方向。

接着，我要感谢马毅老师、John Wright、Arvind Ganesh和杨旭老师在科研过程中给我的建议和帮助，这对我的研究进展十分重要。

然后我要感谢师兄魏嗣明和刘光灿。他们扎实的专业基础和过硬的动手技能令我佩服不已，他们总能耐心的解答我的问题，是学生中的榜样。

我还要感谢师兄师姐师弟师妹们，吴中、苏昊、陶李天、张晓、何因、陶阳宇、刘江宇、梁霄、张宏波、苗又山，同你们的交流和讨论，时常让我豁然开朗。

我需要感谢吴乐秦、彭义刚、Mergen Nachin、张铮东、梁文轩、闵可锐、罗毅、蒋芸，与你们一起研究和工作，让我学到很多很多。

实验室的兄弟姐妹都亲如一家人，我们之间的情谊不是一句感谢可以表达的。王宝元、罗威、冯明、施柏鑫、李扬曦、何鹏、李文耀、吴城磊，与你们的讨论丰富了我的知识，陶冶了我的情操。

我要感谢陈雯老师和马歆老师，他们在学习和生活中给了我很多帮助。

我需要感谢山世光老师、王晓彪老师，他们在研究和工作中给予我很多帮助。同样感谢路亮、翟艺涛、任昊宇、杨涛和蒋长征同学，他们耐心地解答我研究和工作中的疑问。

我还要感谢我的同学钱一峰、郜韦、曹磊、桑波、齐欣、于利前和谭忝升，我们经常一起讨论编程、算法方面的问题，对我编程能力的提高起到了很大的帮助。

我要感谢我现在的室友李东、林辉，还有曾经的室友刘洋、李勇、陈实富、曹光明，与他们一起生活和学习非常愉快。

感谢研究生部的宋守礼老师、李林老师、周世佳老师、李丹老师、冯刚老师、李广鹏老师和郭晓康老师，他们和蔼可亲、平易近人、对学生无微不至的关怀，都给我留下了深刻的印象。

我要谢谢我的父母和姐姐，没有你们的支持和帮助，就没有今天的我。在学习和工作最艰难的时刻，是你们一直陪伴在我左右。你们的爱是我好好学习，努力工作的动力。

最后，我诚挚地感谢在百忙之中抽出时间审阅本文的专家、教授。

作 者 简 历

基本情况

姓名：陈敏铭　　**性别：**男　　**出生日期：**1986年3月　　**籍贯：**江苏

教育经历

2007年9月—至今，中国科学院计算技术研究所，工学硕士

2003年9月—2007年7月，南京大学计算机科学与技术系，理学学士

【攻读硕士学位期间参与的论文】

[1] Zhouchen Lin, Minming Chen, Leqin Wu, and Yi Ma, The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices, Submitted to Mathematical Programming(审稿中)

[2] Arvind Ganesh, Zhouchen Lin, John Wright, Leqin Wu, Minming Chen, and Yi Ma, Fast Algorithms for Recovering a Corrupted Low-Rank Matrix, International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, December 2009

【攻读硕士学位期间参加的科研项目】

[1] 判别分析和张量秩的研究　2008.09-2008.10

[2] 矩阵重建问题的研究，奇异值分解与矩阵恢复并行算法的设计与实现，参与设计集群上的矩阵恢复算法　2009.3-2009.12

[3] 矩阵重建在搜索技术中的应用　2010.01-2010.02