

---

# 基于 Jacobi 旋转的奇异值分解及其并行优化

---

杨扬 计算机科学与技术 1611132

## 摘要

矩阵奇异值分解 (Singular Value Decomposition, SVD) 是数值分析和线性代数领域中一种重要的矩阵分解方法，在信号处理、统计学、机器学习等诸多领域都有着广泛的应用。计算速度及计算精度是奇异值计算研究中所面临的两大基本问题，Jacobi 方法 [Jacobi, 1846] 因其在速度和精度两方面均有较强的表现，是最重要的奇异值分解方法之一。本文从奇异值分解的原理入手，讨论了通过 Jacobi 旋转方法实现奇异值计算的具体算法，基于单边 Jacobi 旋转探究可并行性，并在经典并行算法上提出改进。在实验中，本文提出的并行算法在不同矩阵规模下性能均优于串行算法和经典并行算法，为 Jacobi 方法在奇异值分解的应用中提供了有益的参考。

**关键词：**矩阵、奇异值分解、SVD、Jacobi 旋转、并行

# 目录

<b>1</b>	<b>引言</b>	<b>1</b>
<b>2</b>	<b>相关工作</b>	<b>2</b>
2.1	矩阵奇异值分解 . . . . .	2
2.2	双边 Jacobi 方法 . . . . .	2
2.3	单边 Jacobi 方法 . . . . .	3
<b>3</b>	<b>基于单边 Jacobi 方法的 SVD</b>	<b>5</b>
3.1	单边 Jacobi 方法的串行算法 . . . . .	5
3.2	单边 Jacobi 方法的直接并行算法 . . . . .	6
3.3	单边 Jacobi 方法的分块并行算法 . . . . .	6
<b>4</b>	<b>实验</b>	<b>7</b>
4.1	实验环境 . . . . .	7
4.2	参数设置 . . . . .	8
4.3	实验结果分析 . . . . .	8
<b>5</b>	<b>总结</b>	<b>10</b>
	<b>参考文献</b>	<b>10</b>

# 1 引言

矩阵奇异值分解 (Singular value decomposition, SVD) 是数值分析和线性代数领域中一种重要的矩阵分解方法, 在信号处理、统计学、机器学习等诸多领域, 如主成分分析 (Principal components analysis, PCA) [Pearson, 1901]、潜在语义分析 (Latent semantic analysis, LSA) [Deerwester et al., 1990] 等都有着广泛的应用。

近年来, 随着硬件算力的极大提升, 矩阵计算问题的规模也不断增大, 学术界与工业界对矩阵奇异值计算算法的性能要求也不断提升, 随之出现的是许多含有奇异值分解功能的数值运算软件包, 如 LAPACK [Anderson et al., 1999]、ScaLAPACK [Blackford et al., 1997] 等。这类数值运算包均基于一些经典的奇异值计算算法进行实现, 如 QR 分解、分治算法、Jacobi 方法等, 并根据实际需求应用于不同的计算场景。

Jacobi 方法是最经典的奇异值计算方法之一, 关于其的深入研究可追溯至 20 世纪 50 年代, [Hestenes, 1958] 基于经典的双边 Jacobi 旋转方法 [Jacobi, 1846] 提出求解矩阵奇异值的单边 Jacobi 旋转方法, 此方法通过对矩阵进行迭代地 Jacobi 旋转, 实现对矩阵的单边正交化, 最终通过一系列的矩阵运算求解得到矩阵的奇异值向量。相比于其他的奇异值计算方法及双边 Jacobi 旋转方法, 单边 Jacobi 旋转方法得到的奇异值不光在精度上更有优势, 且算法本身更易于并行实现。

基于单边 Jacobi 算法的深入研究同样很多, 在 1985 年, [Van Loan, 1985] 就提出了基于分块思想的 Jacobi 求解奇异值的算法, 此算法综合考虑了 CPU 内存及 Cache 的存储特性, 设计了更好的内存访问模式, 有效地提高了矩阵数据的存取效率。此外, 就目前而言, 现有的 Jacobi 旋转系列算法在奇异值计算速度上都不如矩阵三对角化后的 QR 分解算法或某些分治算法, 但经过 [Demmel and Veselic, 1992] 证明, Jacobi 旋转方法在奇异值的计算精度上高于 QR 分解算法, 因此, 为应用于某些对精度要求更高的奇异值计算问题, 关于 Jacobi 算法的深入研究仍吸引着众多研究者的目光, 同时也具有相当大的应用价值。

**课题研究动机** 本课题的研究动机首先来自于 Jacobi 方法较深的理论支撑, Jacobi 方法在数学上有严格证明, 研究脉络清晰, 易于查找相关资料; 其次, Jacobi 方法在计算精度上显著强于现有其他算法, 在速度提升方面具有较大的研究价值; 此外, 基于单边 Jacobi 旋转的 SVD 算法本身具有较强的可并行性, 是一种典型的可应用数值并行算法的实例。

**课题研究内容** 本课题将从矩阵奇异值分解的数学原理入手，简要介绍其理论支撑；然后对 Jacobi 系列方法进行回顾，对双边 Jacobi 旋转进行简单描述，着重于对单边 Jacobi 方法的研究，分析其串行算法及经典的并行算法，并在此基础上提出本课题的改进算法；在实验部分，本课题将复现单边 Jacobi 方法的串行算法及经典并行算法，并实现改进的并行算法，随机初始化不同规模的矩阵作为输入，对比单边 Jacobi 方法串行算法、经典并行算法及本文改进算法的测试效果，并得出结论。

## 2 相关工作

### 2.1 矩阵奇异值分解

对于给定大小为  $m \times n$  的一个矩阵  $M$ ， $M$  所有的元素均属于实数域，则存在一个大小为  $m \times m$  的矩阵  $U$  和一个大小为  $n \times n$  的矩阵  $V$ ，使得：

$$M = U \Sigma V^T \quad (1)$$

其中  $\Sigma$  的大小为  $m \times n$ ，其对角线上的  $\Sigma_{ii}$  即矩阵  $M$  的特征值。而形如1的分解公式即被称作矩阵  $M$  的奇异值分解（SVD）。

更直观地来说，在矩阵  $M$  的奇异值分解  $M = U \Sigma V^T$  中：

- $V$  的列（rows）组成一套对  $M$  的正交”输入”或”分析”的基向量。这些向量是  $M^T M$  的特征向量。
- $U$  的列（rows）组成一套对  $M$  的正交”输出”的基向量。这些向量是  $M M^T$  的特征向量。
- $\Sigma$  对角线上的元素是奇异值，可视为是在输入与输出间进行的标量的”膨胀控制”。这些是  $M M^T$  及  $M^T M$  的特征值的非负平方根，并与  $U$  和  $V$  的行向量相对应。

### 2.2 双边 Jacobi 方法

1846 年，[Jacobi, 1846] 提出双边 cobi 旋转方法，其算法最初应用于求解对称矩阵的特征值。核心思想在于通过一系列的 Jacobi 平面旋转，将对称矩阵  $H$  转化为对角矩阵  $D$ ：

$$D = \cdots J_3^T (J_2^T (J_1^T H J_1) J_2) J_3 \cdots = (\cdots J_3^T J_2^T J_1^T) H (J_1 J_2 J_3 \cdots) \quad (2)$$

通过对分解得到的结果做进一步整理，使其与  $H$  的特征值分解  $H = GDG^T$  相对应，从而即知  $D$  的对角线元素即  $H$  矩阵的特征值，而  $G = J_1 J_2 J_3 \cdots$ 。

在应用于 SVD 时，我们由1式可知：

$$V^T (M^T M) V = \Sigma^2 \quad (3)$$

$$M^T M = V \Sigma^2 V^T \quad (4)$$

因为  $\Sigma^2$  为对角矩阵，则如将  $M^T M$  转化为对角矩阵，则其转化矩阵即为右奇异矩阵  $V$ 。

选择 Jacobi 矩阵  $J$ ，其可将  $(i, j)$  上的元素与  $(j, i)$  上的元素进行正交变换，使得非对角线上的元素  $M_{i,j}$  为 0，将矩阵  $M$  与一系列  $J_i$  进行右联乘，将矩阵上对角元素全化为 0，同样地再进行左联乘，将矩阵下对角元素也全化为 0，则有：

$$J_k^T \cdots J_2^T J_1^T (M^T M) J_1 J_2 \cdots J_k = \Lambda = \Sigma^2 \quad (5)$$

因此右奇异矩阵  $V = J_1 J_2 \cdots J_k$ ，而奇异值  $\sigma$  即对角矩阵  $\Lambda$  的对角元素  $\Lambda_{i,i}$  的开方，根据公式  $U_i = \frac{AV_i}{\sigma}$ ，则还可求出左奇异矩阵。

在具体求解奇异值时，为了尽快地消去所有非对角线上的元素，双边 Jacobi 方法在每一次对矩阵进行旋转变换时，都会选取非对角元素中绝对值最大的元素，如果该元素小于算法预设的精度  $\epsilon$ ，则判断算法收敛，得到了矩阵  $M$  的奇异值分解，否则算法将继续进行变换。

**复杂度分析** 如前文所描述一样，双边 Jacobi 方法每一次进行旋转变换前都会在  $n(n-1)/2$  个非对角元素中搜索绝对值最大的元素，这将带来  $O(n^2)$  级别的时间开销。此外，由于执行旋转变换的时间代价仅为  $O(n)$ ，因此对于高维矩阵，搜索最大值的操作时间开销较大，导致算法的运行效率偏低。

## 2.3 单边 Jacobi 方法

单边 Jacobi 方法的核心思想在于通过一系列 Jacobi 平面旋转操作将给定大小为  $m \times n$  的矩阵  $A$  进行正交化：

$$B = A (J_1 J_2 J_3 \cdots) \quad (6)$$

使得  $B$  中任意两列向量  $b_i$  与  $b_j$  满足  $b_i^T b_j = 0$ ，然后对矩阵  $B$  进行归一化处理，得到：

$$B = U \Sigma \quad (7)$$

其中  $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_{n-1}, \sigma_n)$ ,  $\sigma_i = b_i^T b_i$ , 由于右奇异矩阵  $V = J_1 J_2 J_3 \dots$ , 则关于矩阵  $A$  的奇异值分解为如下形式:

$$A = U \Sigma V^T \quad (8)$$

相比于双边 Jacobi 方法, 不难看出单边 Jacobi 方法在求解矩阵奇异值过程中仅从矩阵  $A$  的一个方向对其进行 Jacobi 旋转变换, 这也即是其名称的由来。如上文所描述的, 对矩阵  $B$  进行正交化处理时, 仅有  $i$  和  $j$  两列元素受到影响, 其过程可描述为如下:

$$(b_i^T b_j^T) = (a_i^T a_j^T) \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \quad (9)$$

然后即可得到:

$$b_i^T = c a_i^T + s a_j^T \quad (10)$$

$$b_j^T = s a_i^T - c a_j^T \quad (11)$$

则可根据非对角元素全为 0 的性质计算上式中的参数  $c$  和  $s$ :

$$\begin{aligned} b_i^T b_j &= (c a_i^T + s a_j^T) (s a_i - c a_j) \\ &= a_i^T a_j (c^2 - s^2) + (a_j^T a_j - a_i^T a_i) c s \\ &= 0 \end{aligned} \quad (12)$$

如果我们定义

$$t = \frac{\text{sign}(\tau)}{|\tau| + \sqrt{1 + \tau^2}} \quad (13)$$

$$\tau = \frac{a_i^T a_i - a_j^T a_j}{2 a_i^T a_j} \quad (14)$$

最终即可推导得出  $c$  和  $s$ :

$$c = \frac{1}{\sqrt{1 + t^2}} \quad (15)$$

$$s = t \times c \quad (16)$$

不难看出, 单边 Jacobi 方法需要矩阵  $B$  中所有向量都两两相交后, 算法才可收敛, 因此一共需要对矩阵  $A$  进行  $n(n-1)/2$  次 Jacobi 旋转操作。此外,  $t$  和  $s$  参数的计算是影响单边 Jacobi 方法收敛速度的主要原因之一, 后续也有较多这方面的研究。

## 3 基于单边 Jacobi 方法的 SVD

### 3.1 单边 Jacobi 方法的串行算法

首先，结合于前文我们对单边 Jacobi 方法原理的数学推导，我们可概括单边 Jacobi 方法计算矩阵奇异值的算法步骤为如下形式：

---

**Algorithm 1** One-sided Jacobi's method

---

**Require:** a convergence criterion  $\epsilon$ , a matrix  $U$  that starts out as  $U = A$ , a matrix  $V$  that starts out as the identity matrix

**Ensure:** the left singular matrix  $U$ , the right singular matrix  $V$

```
1: while  $|c|/\sqrt{\alpha\beta} \leq \epsilon$  do
2:   for all pairs  $i < j$  do
3:      $\alpha = \sum_{k=1}^n U_{ki}^2$ 
4:      $\beta = \sum_{k=1}^n U_{kj}^2$ 
5:      $\gamma = \sum_{k=1}^n U_{ki}U_{kj}$ 
6:      $\zeta = (\beta - \alpha)/(2\gamma)$ 
7:      $t = \text{signum}(\zeta)/(|\zeta| + \sqrt{1 + \zeta^2})$ 
8:      $c = 1/\sqrt{1 + t^2}$ 
9:      $s = ct$ 
10:    for  $k = 1$  to  $n$  do
11:       $t = U_{ki}$ 
12:       $U_{ki} = ct - sU_{kj}$ 
13:       $U_{kj} = st + cU_{kj}$ 
14:    end for
15:    for  $k = 1$  to  $n$  do
16:       $t = V_{ki}$ 
17:       $V_{ki} = ct - sV_{kj}$ 
18:       $V_{kj} = st + cV_{kj}$ 
19:    end for
20:  end for
21: end while
```

---

**复杂度分析** 不难看出，串行的单边 Jacobi 算法的外层循环需要遍历矩阵  $U$  所有的两两向量对，因此是一个复杂度为  $O(n^2)$  的操作，内层循环包含对矩阵  $U$  和矩阵  $V$  的旋转操作，复杂度为  $O(n)$ 。因此，对于输入大小为  $n \times n$  的矩阵  $A$ ，单边 Jacobo 方法的串行算法的复杂度为  $O(n^3)$ 。

由于输入矩阵的读取操作复杂度为  $O(n^2)$ ，因此对于单边 Jacobi 方法的并行优化是确实可行的，可预见并行算法的复杂度将会与处理器的核心数有关。

### 3.2 单边 Jacobi 方法的直接并行算法

从执行步骤上看，算法1的内层循环主要由第 3-5 行、第 10-19 行的 3 个复杂度为  $O(n)$  的子矩阵计算和矩阵旋转组成，因此最直接的并行思路即是直接在内层循环中开启多个线程，每个线程执行各自的子矩阵计算和矩阵旋转。

### 3.3 单边 Jacobi 方法的分块并行算法

相比于直接从串行算法的执行步骤入手，另一种并行思路是把处理器的存储特性加入到考虑之中。不难看出，算法1内部循环三个矩阵计算及旋转操作均是矩阵大小  $n$  直接相关的，这样的操作导致 Cache 中的数据不断被换进换出，命中率大大降低。

基于此，我们可以依据分块的思想，提出优化的单边 Jacobi 方法并行算法。一般来说，对于输入矩阵  $A$ ，我们将其按列划分呈块  $A = [A_1, A_2, \dots, A_r]$ ，其中  $A_i$  包含  $n_i$  列数据。相应的，我们也需要对右奇异矩阵  $V$  做分块处理，矩阵  $V$  的初始值为单位矩阵，则我们将其按照与矩阵  $A$  的做法划分为  $I = [I_1, I_2, \dots, I_r]$ 。划分结束后，我们将  $A_i$  和  $I_i$  分配给各个线程，每一个线程获得两个列块  $A_i$  和  $A_j$ ，在更新其的同时更新  $V_i$  和  $V_j$ 。

单边 Jacobi 方法的分块并行算法执行步骤具体如下所示：



---

**Algorithm 2** Block-Parallel One-sided Jacobi's method

---

**Require:** a convergence criterion  $\epsilon$ , a matrix  $U$  that starts out as  $U = A$ , a matrix  $V$  that starts out as the identity matrix

**Ensure:** the left singular matrix  $U$ , the right singular matrix  $V$

```
1: Initialize  $M$  blocks with block size is  $sz$ 
2: Initialize NUM_THREADS threads
3: for  $block = 1$  to  $M$  do
4:   for  $i = block * sz$  to  $(block + 1) * sz - 1$  do
5:     for  $j = i + 1$  to  $i + 1 + sz$  do
6:       Rotate each internal block as in Algorithm1 in each thread to be orthogonal
7:     end for
8:   end for
9: end for
10: Store indexes of block in  $I$  and  $J$ 
11: for each pair  $(i, j)$  in  $(I, J)$  do
12:   Rotate pair as in Algorithm1 to be mutually orthogonal
13: end for
```

---

**复杂度分析** 相比于直接在串行算法上进行循环的并行划分，基于分块思想设计的并行算法从循环结构上对串行算法进行了重构，使得算法的两个外层循环均是分块数  $M$  相关，易于使用若干个线程并行工作，在开启  $p$  个线程的情况下，理论上此算法的复杂度能达到  $O(\frac{n^3}{p})$  级别。

## 4 实验

### 4.1 实验环境

本次实验的设备配置信息如下：

- 实验设备：HP EliteBook 848 G3
- 操作系统：Ubuntu 18.04 LTS
- CPU：Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40GHz
- L1 Cache：128KB

- L2 Cache: 512KB
- L3 Cache: 3.0MB
- 内存: 8.0GB
- IDE: CLion 2019.1.4

## 4.2 参数设置

实验中我使用 C++ 中的 `rand()` 函数随机初始化输入矩阵，为方便计算，将矩阵元素值大小放缩至 100 以内。实验中我使用 OpenMP 作为多线程的开发接口，实验中设置线程数为 4，囿于实验设备性能限制，实验中测试的矩阵规模限制在  $400 \times 400$  以内。

## 4.3 实验结果分析

依照前文所描述的参数设置，我测试各种串行算法及并行算法在不同矩阵规模下的运行情况如表 1 所示：

	串行	直接并行	分块并行
N = 50	9.639ms	5.641ms	9.516ms
N = 100	75.911ms	44.041ms	48.202ms
N = 150	255.06ms	141.088ms	129.709ms
N = 200	663.396ms	358.199ms	308.866ms
N = 250	1308.07ms	661.524ms	607.666ms
N = 300	2293.39ms	1254.43ms	1059.99ms
N = 350	3437.7ms	1787.97ms	1608.7ms
N = 400	5363.14ms	2796.52ms	2593.66ms


表 1: 单边 Jacobi 旋转各算法运行时间代价

实验结果是符合我的预期的，首先来说，并行算法相对于串行算法的性能提升是毫无疑问的，不管是直接并行算法还是我们所改进的分块并行算法，随着矩阵规模的提升，线程间通信开销所占全部时间开销的比例也在逐步下降，因此并行的加速比也是在不断提升的。

其次，实验结果表明，本文所改进的分块并行算法在矩阵规模较大时明显表现出了优于直接并行算法的效果，并且随着矩阵规模增大加速效果愈加明显。但在矩阵规模较小时，分块并行算法的效果并不如直接并行算法，这种问题出现的原因很简单，我们之前的实验也做过相似的探讨。简单来说，在矩阵规模较小时，整个矩阵甚至都可以存入 CPU 的 Cache，在这种情况下，分块算法所带来的 Cache 存储模式的优越性便不再存在，将输入矩阵进行分块的操作反而造成了一些额外的开销，使得分块并行算法表现劣于直接并行算法。

那么，如何确定使得分块并行算法能够优于直接并行算法的矩阵规模大小值，也就成了我们需要探讨的另一个问题，这里我们再引入一个测试实验。

首先回顾一下实验开发设备的配置信息，如图1所示



Cache Organization <a href="#">[Edit/Modify Cache Info]</a>				
L1\$	128 KiB	L1I\$ 64 KiB	2x32 KiB	8-way set associative
		L1D\$ 64 KiB	2x32 KiB	8-way set associative
				write-back
L2\$	512 KiB		2x256 KiB	4-way set associative
				write-back
L3\$	3 MiB		2x1.5 MiB	write-back

图 1: 实验设备 Cache 配置信息

可以看到，实验设备的 L1 数据 Cache 大小仅为 64KiB，实验中矩阵元素的数据类型为 double，因此理论 L1 Cache 能够存下大小为  $\sqrt{8 * 1024} \times \sqrt{8 * 1024}$  的矩阵，在这种情况下可以认为分块并行算法和直接并行算法的效果是一样的（实际由于分块带来的开销直接并行优于分块并行），随着矩阵规模再增大，分块并行算法因其更高的 Cache 命中率，则逐渐优于直接并行算法。

基于此，我们测试大小为  $\sqrt{8 * 1024} \times \sqrt{8 * 1024}$ （取下界为  $90 \times 90$ ）的输入矩阵：

	串行	直接并行	分块并行
N = 90	55.955ms	30.116ms	39.902ms

表 2: 大小为  $90 \times 90$  的输入矩阵运行效果

对比矩阵规模在  $180 \times 180$  时的运行情况：

	串行	直接并行	分块并行
N = 180	463.118ms	248.848ms	206.538ms

表 3: 大小  $180 \times 180$  的输入矩阵运行效果

可以看到，算法的运行情况是基本符合我们的理论推测的，也即是本文所改进的分块并行算法相比于直接并行算法性能提升的原因很大一部分即来自于对 Cache 命中率的改善。

## 5 总结

本次实验是并行计算课程的结课作业，总的来说这学期的课程学到了很多东西，在调研本课题的时候也对所学的内容也做了系统的回顾，收获还是蛮大的，但由于时间比较仓促，虽然总体上达到了预想的优化效果，但有些想法还没有能落实到代码上，希望能在以后对课题继续展开一些深入的研究。

## 参考文献

- [Anderson et al., 1999] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition.
- [Blackford et al., 1997] Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. (1997). *ScaLAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.
- [Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407.
- [Demmel and Veselic, 1992] Demmel, J. and Veselic, K. (1992). Jacobi's method is more accurate than qr. *SIAM J. Matrix Anal. Appl.*, 13:1204–1245.

- [Hestenes, 1958] Hestenes, M. (1958). Inversion of matrices by biorthogonalization and related results. *Journal of the Society for Industrial and Applied Mathematics*, 6(1):51–90.
- [Jacobi, 1846] Jacobi, C. (1846). Über ein leichtes verfahren die in der theorie der säcularstörungen vorkommenden gleichungen numerisch aufzulösen. *Journal für die reine und angewandte Mathematik*, 30:51–94.
- [Pearson, 1901] Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572.
- [Van Loan, 1985] Van Loan, C. (1985). The block jacobi method for computing the singular value decomposition. Technical report, Ithaca, NY, USA.