中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生

数学建模竞赛

学　　校	东南大学
参赛队号	21102860073
队员姓名	1.凌泰炀
	2.张逸洋
	3.钱缪峰

中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生

数学建模竞赛

题 目 相关矩阵组的低复杂度计算和存储建模

摘 要：

随着通信电子技术快速发展，以大规模阵列为代表的传感器、雷达、天线阵列凭借高性能、高速率、高增益的功能实现，得到了广泛的关注和研究。然而随着阵列的持续扩大，常规处理算法对计算和存储的需求成倍增长，对处理器件或算法的实现成本和功耗提出了巨大的挑战。因此根据相关矩阵组的特征分布，充分挖掘矩阵间关联性，以实现低复杂度的计算和存储，具有十分重要的价值和意义。

对于矩阵关联性，我们在**相关性**和**稀疏性**两个方面深入分析并获得关键的结论。在相关性分析中，我们发现输入矩阵 $\boldsymbol{H}$ 中的 K 个矩阵的同一行之间具有较强的相关性，而矩阵内的相关性较弱，输入矩阵 $\boldsymbol{H}$ 、中间矩阵 $\boldsymbol{V}$ 和输出矩阵 $\boldsymbol{W}$ 的矩阵间相关系数存在明显关联；在稀疏性分析中，我们发现输入矩阵 $\boldsymbol{H}$ 存在明显的稀疏性，可以通过建立稀疏基进行低开销的矩阵压缩和重建，此外我们利用矩阵的稀疏性解释了矩阵相关性的来源。

对于问题一，其主要目标是相关矩阵组的计算优化问题，需要提出模型在 $\rho_{\min}(\boldsymbol{W})>0.99$ 的目标下，尽可能利用矩阵内或矩阵间的相关性来简化算法流程，或者是利用分析计算的流程来实现 SVD 或者逆矩阵运算的复杂度优化。针对模型问题，我们设立了相邻矩阵相关系数阈值 λ_{th}，并完成了强关联性矩阵之间的中间矩阵 $\boldsymbol{V}$ 的替代，分别提出了**基于矩阵间关联性的 SVD 计算优化和逆矩阵计算优化**，运算复杂度可分别降低 **46.0%**和降低 **85.9%**。针对数学问题，我们引入了**基于随机正交基的 SVD 算法**，通过提前截断奇异值序列减少计算，相比传统的 Golub-Kahan 方法，运算复杂度降低了 **38.7%**。针对逆矩阵计算，引入了**基于逐次超松弛迭代法的逆矩阵求解算法**，通过迭代的方式求取逆矩阵，可以降低复杂度至传统解法的 **24.7%**。将上述四种优化算法合并，得到了基于相关系数的矩阵组计算复杂度优化模型，可在保持 99.99%最小精度达标的条件下完成从输入矩阵 $\boldsymbol{H}$ 到

输出W 的计算过程，可以降低 **72.8%**的总运算复杂度。此外，我们建立了**标准运算复杂度表**，将所有计算过程以**单位计算次数**的形式表达。

对于问题二：其主要目标是相关矩阵组压缩优化问题，需要提出的模型在 $err\_max=-30dB$ 的目标下，尽可能利用矩阵内或矩阵间的关联性，以较低的运算复杂度实现较高的数据压缩率。我们讨论了主流的矩阵压缩方法，并提出了基于奇异值分解的矩阵压缩模型和基于空间正交基的矩阵压缩模型。其中**基于奇异值分解的矩阵压缩模型**压缩效率高，压缩后存储复杂度可以低至原本的 20%左右，适合用于对压缩率要求高且运算资源充裕的场景；**基于空间正交基的压缩感知模型，**其运算复杂度比前者要低 1 个数量级以上，适合用于对压缩率要求一般但是对计算要求高的场景。最后，我们通过 MATLAB 进行了多次测试，证明了两个模型各自适用的场景的高性能和鲁棒性。

对于问题三：其主要目标是实现问题一、二中的模型进行有效的整合，从而实现更低复杂度和更高性能。我们根据分析，可以从三个方向进行优化，包括利用空间正交基辅助相关系数阈值的判断、整合压缩和计算 SVD 分解的过程、利用中间矩阵V_k 的相关性提高输出矩阵W_k 的压缩效率。最后，我们搭建了整体 MATLAB 模型并完成了测试。、

关键词：相关性 稀疏性 随机 SVD 逐次超松弛迭代 低复杂度

目 录

1. 问题重述

1.1 问题背景

随着通信电子技术快速发展，以大规模阵列为代表的传感器、雷达、天线阵列凭借高性能、高速率、高增益的功能实现，得到了广泛的关注和研究。然而随着阵列的持续扩大，常规处理算法对计算和存储的需求成倍增长，对处理器件或算法的实现成本和功耗提出了巨大的挑战。以第五代（5G）移动通信技术的大规模 MIMO 技术为例[1]，通过基站配备大型天线阵列，可以有效实现空间分集和复用，从而大幅度提高空间分辨率、提高吞吐量。但是随着天线规模的不断增加，信号运算开销和信道训练开销也成倍增长，严重限制了大规模 MIMO 技术的发展与广泛应用。

因此，在无线通信、计算机视觉的新兴子领域中，通常采用挖掘信号矩阵之间的关联性等方法，以人工智能技术为辅助，可有效实现计算和存储开销的降低。最具代表性的是华裔科学家 T. Tao 等人在 2007 年提出的压缩感知理论[2][3]，改变了奈奎斯特采样定律的思维惯性，利用矩阵间的稀疏特性有效地降低信号传输、测量、保存成本，目前已广泛应用于图像处理、信号处理等多个领域。

综上，根据矩阵组的特征分布，充分挖掘矩阵间关联性，以实现低复杂度的计算和存储，具有十分重要的价值和意义。

1.2 问题描述

在本次题目的研究中，我们主要考虑的是已知相关矩阵组的计算与存储问题。其中，矩阵运算的主要工作为将给定的输入矩阵 $\boldsymbol{H}$ 通过奇异值分解（SVD）求解的得到特征向量 $\boldsymbol{V}$，并通过计算得到输出矩阵 $\boldsymbol{W}$，计算总计算的复杂度，并确认运算结果的准确性。矩阵存储的主要工作是将给定的输入矩阵 $\boldsymbol{H}$ 和输出矩阵 $\boldsymbol{W}$ 进行压缩和解压缩，以低运算复杂度，实现高压缩率。在提供的矩阵数据组中，我们需要通过分析目前未知的矩阵组关联特性，减少重复运算和存储的过程，从而实现整个系统的高性能、低复杂度与高压缩率。

1.3 问题提出

针对问题一：我们作以下问题归纳

总体任务：

以低复杂度和高准确率完成矩阵计算

算法指标：

最小化总计算复杂度 $\min C_C$、最低建模精度 $\rho_{\min}(\boldsymbol{W}) > 0.99$

计算步骤：

① 输入矩阵 $\boldsymbol{H}$，通过奇异值分解等方法，求解的得到特征向量 $\boldsymbol{V}$

② 特征向量 $\boldsymbol{V}$，通过 $\boldsymbol{W}_k = \boldsymbol{V}_k \left(\boldsymbol{V}_k^H \boldsymbol{V}_k + \sigma^2 \boldsymbol{I} \right)^{-1}$ 得到结果矩阵 $\boldsymbol{W}$

优化目标：通过讨论，我们推测可以通过以下环节，实现计算优化：

- 利用矩阵间关联性减少 SVD 的计算频率；
- 利用矩阵间关联性简化 SVD 的计算过程；
- 利用矩阵内关联性简化 SVD 的计算过程；
- 利用矩阵间关联性减少 W 的计算过程；
- 利用迭代等方法　减少或者替代 W 的逆矩阵计算。

工作安排：针对问题一，我们完成以下具体工作：

- 建立计算复杂度标准表格；

- 完成常规处理算法下的计算与验证，并完成总计算复杂度的计算；
- 依据上述五个优化目标进行计算优化，在尽可能满足建模精度 $\rho(\boldsymbol{W})>0.99$ 的条件下，简化运算规则，减少重复计算；
- 提出基于迭代法的计算，减少计算复杂度。

针对问题二：我们作以下问题归纳

总体任务：

以低复杂度和高压缩率完成矩阵压缩与解压缩

算法指标：

最小化总计算复杂度 $\min C_S$ 、最大压缩误差 $\mathrm{err}_i \le -30dB, i=\boldsymbol{H},\boldsymbol{W}$

计算步骤：

① 对已知输入矩阵 $\boldsymbol{H}$ 完成压缩与解压缩

② 对已知结果矩阵 $\boldsymbol{W}$ 完成压缩与解压缩

优化目标：通过讨论，我们推测可以通过以下环节，实现压缩优化：

- 利用矩阵间关联性 减少 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 的实际维度；
- 利用矩阵间关联性 减少 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 的实际数据总量（如用稀疏基进行表示）；
- 利用矩阵内关联性 减少 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 的实际数据总量（如用稀疏基进行表示）；

工作安排：针对问题二，我们完成以下具体工作：

- 完成常规处理算法下的存储与验证，并完成总计算复杂度的计算；
- 依据上述三个优化目标进行计算优化，在尽可能满足最大压缩误差 $\mathrm{err}_i \le -30dB$ 的条件下，简化运算规则，减少重复计算；
- 提出基于字典法的计算，减少计算复杂度。

2. 基本假设与符号说明

2.1 基本假设

(1) 根据原题含义，问题 1 和 2 中假设实数的加法运算的运算复杂度为基本单位 1，并在此基础上推导出复数域$\mathbb{C}$上的全部计算复杂度。

(2) 根据原题含义，问题 1 和 2 中假设矩阵数据在读写（提取、移动、赋值）过程的运算复杂度为 0，即不占用程序的运算模块。

(3) 根据原题含义，问题 1 和 2 中假设比较大小和逻辑运算的复杂度为 0，不计算在迭代等操作的复杂度中。

(4) 对于复数矩阵中单个复数元素，其实部和虚部均采用 32 比特单精度浮点表示。压缩后的每个元素仍然以 32 比特单精度浮点数存储，不考虑位宽的压缩。

2.2 符号说明

符号	含义
$\boldsymbol{H}$	输入矩阵
$\boldsymbol{V}$	中间矩阵
$\boldsymbol{W}$	输出矩阵
λ	两矩阵之间的相关系数
D	正交基字典
$\hat{g}_i$	空间正交基投影幅度值
u_i	空间正交基索引
$\boldsymbol{H}_{res}$	剩余信道矩阵
λ_{th}	决定邻近矩阵是否可以相互替代的相关系数阈值
$\rho(\boldsymbol{V})$	中间矩阵的建模精度
$\rho(\boldsymbol{W})$	输出矩阵的建模精度
C_{colub}	Colub-Kahan 方法进行 SVD 分解的计算复杂度
$C_{general\_inv}$	一般方法求逆的计算
C_{SOR}	SOR 迭代法的计算复杂度
C_{judge}	横向相邻矩阵相关系数计算的计算复杂度
C_{svd}	RSVD 计算的计算复杂度
C_{inv}	逆矩阵计算的运算复杂度
α	所有中需要进行 RSVD 操作的矩阵的比例
t_{qua}	所有输出矩阵的列向量中达到预设估计精度的数量
$\boldsymbol{\Lambda}$	相关系数矩阵
α	估计精度达标率
ΔC	运算复杂度下降率
$\boldsymbol{U}$	左奇异矩阵
$\boldsymbol{S}$	奇异值矩阵
$\boldsymbol{V}$	右奇异矩阵

3. 问题一的模型建立与求解

3.1 问题分析

3.1.1 问题讨论

问题一的本质是相关矩阵组计算优化问题，需要提出的模型在的目标下，尽可能利用矩阵内或矩阵间的相关性来简化算法流程的模型问题，或者是利用分析计算的流程来实现SVD 或者逆矩阵运算优化的数学问题。

关于基于关联性的模型问题，我们作以下讨论：根据题目要求，每组 DATA 的原始数据为给定的且具有一定关联性的复数矩阵组 $\boldsymbol{H}=\{\boldsymbol{H}_{j,k}\}$。即矩阵组的同一行块内部 K 个矩阵间 $\{\boldsymbol{H}_{j,1},\boldsymbol{H}_{j,2},\boldsymbol{H}_{j,3},\cdots,\boldsymbol{H}_{j,K}\}$ 矩阵间存在一定关联性，同时，单个矩阵 $\boldsymbol{H}_{j,k}$ 内的各个元素间 $\{h_{m,n}^{(j,k)}\},m=1,\cdots,M,n=1,\cdots,N$ 也存在一定的关联性。

一方面，通过合理分析提取出矩阵间或矩阵内的关联性，可以减少重复运算，提高运算效率，减少计算的复杂度，但是另一方面，对于关联性的应用，不可避免地会造成估计精度 ρ 的降低，同时会引入新的计算过程。因此，在基于关联性的模型问题中，我们需要用尽可能简单的算法，提取最重要的矩阵关联性，才能够实现题目中的目标，否则适得其反会造成运算度提升或者性能下降，因此对模型提出了极高的要求。

关于基于特定目标的数学问题，我们作以下讨论：根据题目要求，$\rho_{\min}(\boldsymbol{W})>0.99$ 将输入矩阵 $\boldsymbol{H}$ 推导出输出矩阵 $\boldsymbol{W}$ 的过程中，需要先将输入矩阵通过 $\boldsymbol{H}$ 奇异值分解等方法，求解的得到特征向量 $\boldsymbol{V}$，再通过公式(1)计算得到输出矩阵 $\boldsymbol{W}$。

$$\boldsymbol{W}_k=\boldsymbol{V}_k\left(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)^{-1} \tag{1}$$

在数学计算的优化过程中，我们应该首先考虑更加切合题目思路的数学方法，从而实现优化计算过程或者减少非必要计算的内容。例如本题目中，从推导得到的特征向量 $\boldsymbol{V}$ 实际上是输入矩阵 $\boldsymbol{H}$ 经过 SVD 分解后得到的右奇异矩阵前 L 列，也就意味着实际上并不需要求出 $\boldsymbol{H}$ 的左奇异矩阵 $\boldsymbol{U}$ 和奇异值矩阵 $\boldsymbol{S}$，相对于传统的基于 QR 分解的 SVD 算法可以节约较大的计算开销。同样的思路可以用于逆矩阵的运算。

通过以上分析，我们得到问题一的解题思路内容如图 1 所示：

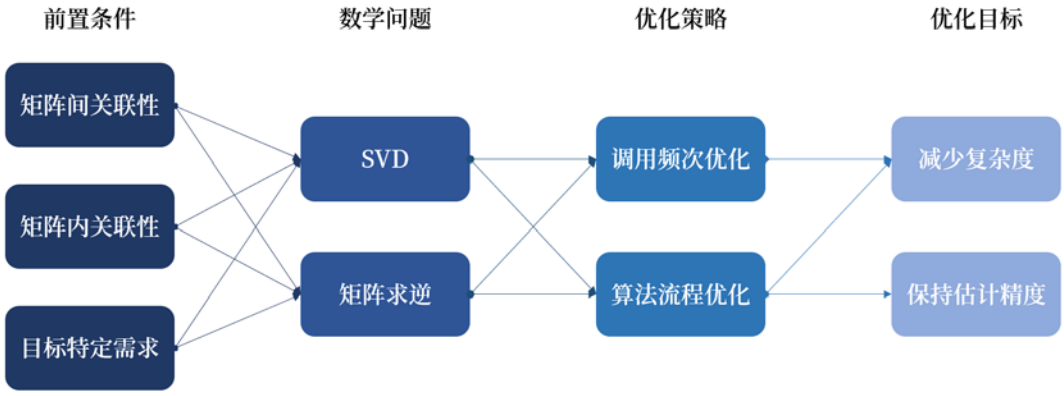

图 1 问题一的解题思路

3.1.2 矩阵相关性分析

矩阵组的相关性是本题目实现优化的重要条件，因此正确地解读出输入矩阵组{$\boldsymbol{H}$}之间的相关性至关重要。在分析过程中，我们主要采用特征值分解、奇异值分解或者相关系数计算等方法，依次分析$M\times K$矩阵间的相关性、$M\times N$矩阵内的相关性、$1\times M$行向量内的相关性、输入矩阵$\boldsymbol{H}$、中间矩阵$\boldsymbol{V}$和输出矩阵$\boldsymbol{W}$的相关性，为后续算法优化提供方向。

图 2 本节中矩阵间/内相关性分析的示意图

- **输入矩阵$\boldsymbol{H}$的$N\times K$矩阵间的相关性分析**

我们利用如下公式计算两矩阵之间的相关系数：

$$\lambda=\frac{\left\|\boldsymbol{\alpha}^{H}\boldsymbol{\beta}\right\|_2}{\left\|\boldsymbol{\alpha}\right\|_2\left\|\boldsymbol{\beta}\right\|_2} \tag{2}$$

其中，λ为两矩阵之间的相关系数，$\boldsymbol{\alpha}$和$\boldsymbol{\beta}$分别代表将两个$m\times n$矩阵的所有列（或行）首位相连组成的mn维向量。

针对第j组的K个矩阵第m行的所有矩阵，我们可以得到一个维度为$N\times K$的矩阵。合理推测，可能对于第m行向量而言，邻近的矩阵k之间可能存在较强的相关性，或者整个$N\times K$矩阵中存在一定的相关性。根据公式(2)计算行矩阵之间的相关系数，得到8个邻近矩阵和8个非邻近矩阵的相关性如下图所示。

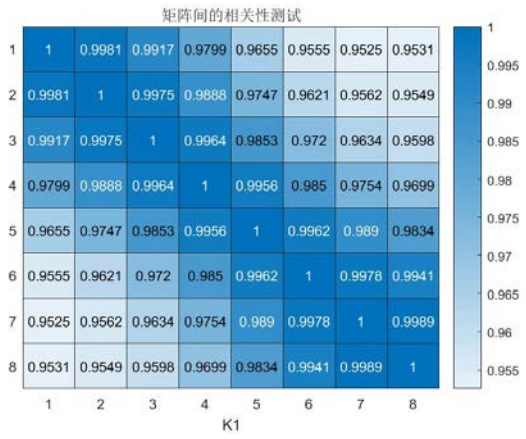

图 3 邻近矩阵间的相关系数

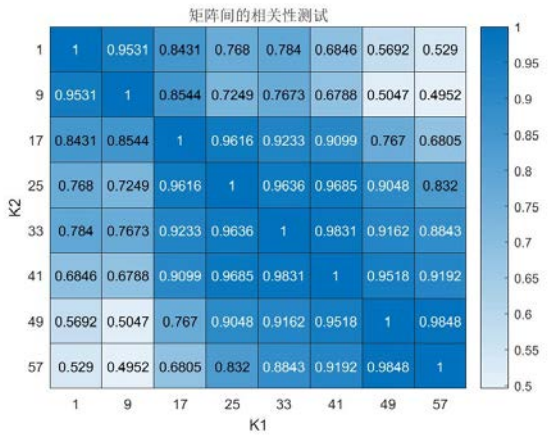

图 4 非邻近矩阵间的相关系数

图 3 中展示 8 个邻近矩阵的相关系数，可以明显看出，8 个矩阵之间最小的相关性有 0.95，且相邻的矩阵间相关系数超过 0.99，充分说明了在 K 方向上的向量具有强相关。图 4 则展示了 8 个非邻近矩阵的相关系数，每个矩阵序号相差 8 个矩阵时相关性依然很高，同时可以看出当两个矩阵在 K 方向上距离较远时，相关系数快速下降，最小值已经低于 0.50 为较弱相关。

- **输入矩阵 $\boldsymbol{H}$ 的 $M\times N$ 矩阵内的相关性分析**

已知一个矩阵组中拥有 $J\times K$ 个矩阵，每个矩阵的维度为 $M\times N$ 。因为 M 的数值较小，矩阵列向量的个数只有 $M=4$，因此可以不考虑列相关性计算。合理推测 M 行向量存在一定的线性相关性，因此对于单个矩阵 $\boldsymbol{H}_{j,k}=\left\{\boldsymbol{h}_{m,n}^{(j,k)}\right\}_{m=1,\cdots,M,n=1,\cdots,N}$ 中，根据相关性公式(2)计算行相关性，得到第 k 个矩阵邻近行之间的相关性如下图所示：

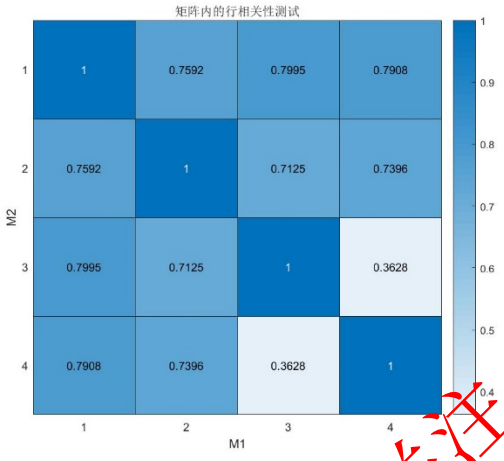

图 5 $j=1,\ k=1$ 矩阵内的行相关系数

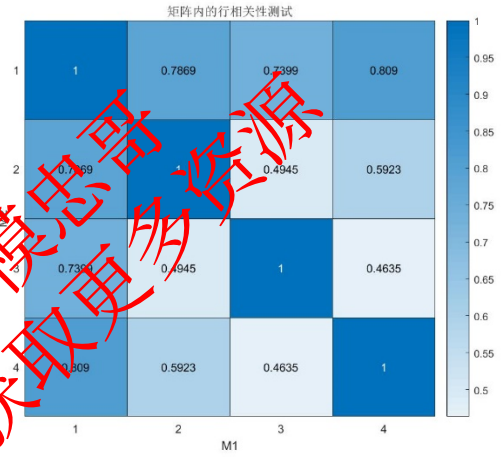

图 6 $j=1,\ k=10$ 矩阵内的行相关系数

从图 5 和图 6 中可以看出，证明 $M\times N$ 矩阵内的行相关系数低于强相关系数 0.8，并且随着 j 和 k 的不同随机变化，和矩阵间的相关系数相比，基本可以认定矩阵内只存在弱相关性。此外，我们对矩阵 $\boldsymbol{H}_{j,k}$ 进行奇异值分解，得到 M 个较大的奇异值，进一步证明 $\boldsymbol{H}_{j,k}$ 的行相关性较弱。因此，我们认为，矩阵内的相关性不适宜用于计算的化简。

- **输入矩阵 $\boldsymbol{H}$ 的 $1\times N$ 行向量的相关性分析**

行向量内的相关性分析，主要为考虑矩阵的行向量可能是通过重组得到。例如 MIMO 系统中，8×8 的天线阵列接收信号，为了方便计算和保存，通常会重新排序得到 1×64 的行向量。因此，我们将每个矩阵中的 $1\times N, N=64$ 的行向量拆分重组为 8×8 的方阵、4×16 的矩阵，通过特征值或者奇异值计算，判断其相关性。判断过程同 $M\times N$ 矩阵内的相关性分析，不再赘述。

分析结果发现，$1\times N$ 行向量内通过拆分重组后，相关性均较弱，因此不再考虑。

- **输入矩阵 $\boldsymbol{H}$、中间矩阵 $\boldsymbol{V}$ 和结果矩阵 $\boldsymbol{W}$ 的相关性分析**

分析输入矩阵 $\boldsymbol{H}$、中间矩阵 $\boldsymbol{V}$ 和结果矩阵 $\boldsymbol{W}$ 之间的关联，可以帮助我们有效地减少中间过程的重复计算。在本文中，我们主要考虑的是对于在 K 方向上邻近的输入矩阵 $\boldsymbol{H}$ 与 $\boldsymbol{V}$、$\boldsymbol{W}$ 的关系，主要目标时利用邻近矩阵的相关性[5]，从而通过替代计算结果来减少重复计算，并保持较高的性能。考虑到 $\boldsymbol{V}$ 是输入矩阵 $\boldsymbol{H}$ 通过 SVD 分解得到的，$\boldsymbol{W}$ 是通过中间矩阵 $\boldsymbol{V}$ 计算得到的，我们合理推测输入矩阵 $\boldsymbol{H}$、中间矩阵 $\boldsymbol{V}$ 和结果矩阵 $\boldsymbol{W}$ 具有一定的一致性，即当两个输入矩阵 $\boldsymbol{H}$ 相近时，他们的中间矩阵 $\boldsymbol{V}$ 和结果矩阵 $\boldsymbol{W}$ 也相近，或有着相同的变化趋势。

为了充分反映这样的相关趋势，我们进行以下试验分析：获得第 j 组在 K 方向上邻近的输入矩阵 $\boldsymbol{H}_{j,k}, k=1,2,\ldots,K$，并计算得到了他们的中间矩阵 $\boldsymbol{V}_{j,k}$ 和输出矩阵 $\boldsymbol{W}_{j,k}$。计算邻近 k 的相关系数满足公式：

$$\lambda_{X,j,k}=\frac{\left\|\boldsymbol{X}_{j,k}\cdot\boldsymbol{X}_{j,k-1}^{H}\right\|}{\left\|\boldsymbol{X}_{j,k}\right\|\left\|\boldsymbol{X}_{j,k-1}\right\|},X=\boldsymbol{H},\boldsymbol{V},\boldsymbol{W},j=1,2,\ldots,J,\quad k=1,2,\ldots,K \tag{3}$$

并绘制折线图。图中每个自变量为 K 方向的序列号 k，每个点表示为第 k 个矩阵与第 $k-1$ 个矩阵之间的相关系数，包括 $\boldsymbol{H}$、$\boldsymbol{V}$ 和 $\boldsymbol{W}$。从反映出两个重要的趋势关系，首先是证明了 $\boldsymbol{H}$、$\boldsymbol{V}$ 和 $\boldsymbol{W}$ 具有一定的一致性，反映在他们的结果值的变化上。其次是明显 $\boldsymbol{V}$ 的相关系数变化更剧烈一些，主要原因是这里的 $\boldsymbol{V}$ 是提取了 SVD 分解后右奇异矩阵的前 L 列，从而导致了 $\boldsymbol{V}$ 的性能更容易波动，符合我们的预期。

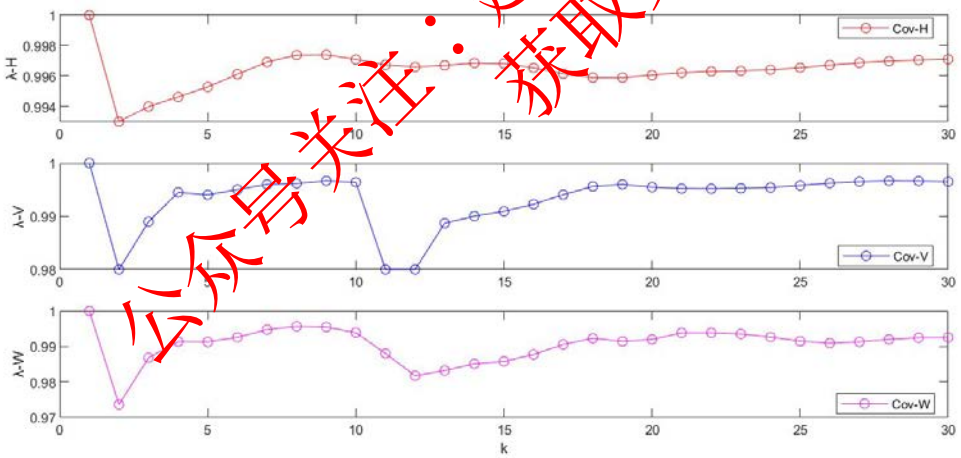

图 7　$\boldsymbol{H}$、$\boldsymbol{V}$、$\boldsymbol{W}$ 前 30 个邻近矩阵的相关系数趋势图

- **矩阵关联性分析总结**

综合上述讨论，关于矩阵相关性可以具体得到以下结论：

1. 输入矩阵 $\boldsymbol{H}$ 的 $N\times K$ 矩阵间的具有较强的相关性；
2. 输入矩阵 $\boldsymbol{H}$ 的 $N\times M$ 矩阵内的具有较弱的相关性；
3. 输入矩阵 $\boldsymbol{H}$ 的 $1\times M$ 向量内基本不具有相关性；
4. 输入矩阵 $\boldsymbol{H}$、中间矩阵 $\boldsymbol{V}$ 和输出矩阵 $\boldsymbol{W}$ 的矩阵间相关系数存在明显的一致性；
5. 根据 1 和 4，我们可以制定后续的优化方案。

3.1.3 矩阵稀疏性分析

矩阵的稀疏性在此处是指在代数空间上存在某一组正交基或者过完备基，使得矩阵的行向量或列向量可以通过较少的几个基上的表达来还原整个矩阵向量。例如信号处理领域，因为自然界中的信号低频居多[6]，高频部分基本都是噪声，使用傅立叶做基矩阵时，表达系数往往只在几个低频的基上比较大，从而可以用较少的数据还原整个矩阵或者向量[7]。

矩阵的稀疏性应当被视为矩阵关联性的一部分，在本文中，通过对矩阵的稀疏分解可以帮助理解信道的相关性的来源，从而更好地提出解决方案。在稀疏分解过程中，最重要的一点是生成一组对于矩阵而言具有稀疏性的基，因此，在测试包括基于高斯随机分布的稀疏分解、基于二值化稀疏矩阵 GSVM 的稀疏分解等方案后，**我们最终确定了以下两种具有明显稀疏性的基，将有助于对信道矩阵的理解。**

- **基于$\boldsymbol{V}_k$正交基的稀疏分解**

考虑到中间矩阵$\boldsymbol{V}_k$是通过输入矩阵$\boldsymbol{H}$进行 SVD 分解得到的，因此$\boldsymbol{H}$有大概率在$\boldsymbol{V}_k$上有投影值，因此可以通过$\boldsymbol{V}_k$来生成正交基，从而作为$\boldsymbol{H}$矩阵稀疏分解的条件。根据施密特正交化方法，假设正交基组已经有I个元素，那么新加入的$\boldsymbol{V}_k$通过正交化满足公式

$$\tilde{\boldsymbol{V}}_{I+1}=\boldsymbol{V}_k-\sum_{i=1}^{I}\frac{\left(\boldsymbol{V}_k,\tilde{\boldsymbol{V}}_i\right)}{\left(\tilde{\boldsymbol{V}}_i,\tilde{\boldsymbol{V}}_i\right)}\tilde{\boldsymbol{V}}_i \tag{4}$$

需要注意的是，原始的$\boldsymbol{V}$是$N\times N$的矩阵，因此每次提取的$\boldsymbol{V}_k$实际上是$\boldsymbol{V}$的一个向量。完成正交基生成后，我们得到一组正交基字典$\boldsymbol{D}=\{\tilde{\boldsymbol{V}}\}$，在测试中，正交基的数目大约在 300~700 个左右，与选择的停止阈值有关。

通过正交匹配追踪，根据广义似然比检验公式

$$\hat{g}=\underset{i}{\operatorname{argmax}}\frac{\boldsymbol{D}^H\boldsymbol{H}}{|\boldsymbol{D}|} \tag{5}$$

可以得到估计的投影幅度值$\hat{g}_i$和索引u_i，找到后去除该幅度的能量，得到剩余信道矩阵，重复上述操作，直到达到设定的阈值。

$$\boldsymbol{H}_{res}=\boldsymbol{H}-\hat{g}_i\boldsymbol{D}(u_i) \tag{6}$$

测试结果如下：

表 1　基于 Vk 正交基的稀疏分解试验结果

抽取次数	k=1 m=2		k=1 m=1		k=2 m=1	
	能量占比	基索引	能量占比	基索引	能量占比	基索引
1	17	4	14	3	13	3
2	8	2	9	111	10	4
3	8	15	8	4	10	111
4	6	54	5	203	5	203
5	5	256	4	34	5	7
6	4	226	4	16	4	6
7	4	6	3	6	4	16

需要说明的是，为了避免出现直接投影的情况，实际上是采用 DATA1 数据集生成的 $\boldsymbol{V}_k$ 在数据集 2 中进行测试，每次测试迭代次数为 100~200 次不等。在测试中，我们可以得到以下**四个重要结论**：

① 输入矩阵 $\boldsymbol{H}$ 存在一定的稀疏性，其第 1 次抽取的结果多数情况下会占据 10%以上的能量，前 20 次抽取的结果会占据 80%以上的能量，说明输入矩阵在稀疏基上可以进行投影。

② 对于同一个 k 矩阵里的 m 个行向量，他们之间存在相同的基索引较少，也就意味着同一个 k 矩阵里的 m 个行向量实际上组成元素不相同，这也说明了为什么矩阵内的相关性较低。

③ 对于不同 k 矩阵里的同一个行向量，他们之间存在相同的基索引较多，也就意味着同一个 k 矩阵里的 m 个行向量实际上组成元素基本相同，这也说明了为什么矩阵间的相关性较高

④ 输入矩阵 $\boldsymbol{H}$ 除了存在稀疏基外，还存在着一定能量的高斯噪声信号，导致最后 10%的能量无法被稀疏分解。

- **基于空间正交基的稀疏分解**

除了上述的利用提供的数据集生成的 $\boldsymbol{V}_k$ 基外，我们还提供了一种基于空间正交基的稀疏分解方法，该方法通过模拟建立空间信道模型，可以很好地分解输入矩阵 $\boldsymbol{H}$ 。具体如下：

设置过采样系数 β_θ 为正整数，建立角度向量，

$$\bar{\theta} \in \left[-\pi, -\pi + \frac{2\pi}{\beta_\theta N}, \cdots, \pi - \frac{2\pi}{\beta_\theta N} \right] \tag{7}$$

其中 N 为已知输入矩阵 $\boldsymbol{H}$ 的第二维度。建立角度域字典

$$\boldsymbol{D} = \left[1, e^{j2\pi \frac{d}{\lambda} \sin\bar{\theta}}, \cdots, e^{j(N_r-1)2\pi \frac{d}{\lambda} \sin\theta} \right]^T \tag{8}$$

完成字典建立后，通过正交匹配追踪，根据广义似然比检验公式同 $\boldsymbol{V}_k$ 基部分[8][9]，去除能量，迭代重复，最终可以得到输出的结果。在测试中，**空间正交基可以更快地分解出有效索引，第一条索引的能量值可以达到 30%，前 20 条索引的能量值可以超过 90%**，性能卓越。但是，同样面对剩余 10%的噪声能量，只能以较低的幅度值进行迭代，稀疏分解难度较大。

- **矩阵稀疏性分析结果**

综合上述讨论，关于矩阵稀疏性可以具体得到以下结论：

1. 输入矩阵 $\boldsymbol{H}$ 的大约 90%能量存在明显的稀疏性，可以通过 10~20 次迭代进行稀疏分解，并得到索引值，可以进行低开销的矩阵压缩和重建；
2. 输入矩阵 $\boldsymbol{H}$ 的剩余 10%能量没有明显的稀疏性，不易进行稀疏分解，这将最终限制了稀疏分解的矩阵压缩和重建能力；
3. 对于同个 k 矩阵里的 m 个行向量，分解出来的基不同，造成了矩阵内低相关性。
4. 对于不同 k 矩阵里的同一个行向量，分解出来的基相同，造成了矩阵间高相关性。
5. 矩阵稀疏性解释了矩阵相关性的来源，可以用于矩阵的压缩和解压缩。

3.2 分步优化方法

3.2.1 基于矩阵间关联性的 SVD 计算优化

通过利用矩阵之间的相关性，我们的目标是减少 SVD 的计算频率，从而达到降低运算复杂度的目标。在 3.1.2 节的图 7 中，我们展示了在 K 方向上邻近的输入矩阵 $\boldsymbol{H}$ 相关性、特征矩阵 $\boldsymbol{V}$ 相关性与输出矩阵 $\boldsymbol{W}$ 相关性，在分布上具有一致性的特征。因此，我们可以利用这一分布特点，实现邻近 $\boldsymbol{V}$ 的替换，实现降低 SVD 计算频率的目的。具体操作如下：

对于第 k 个输入矩阵 $\boldsymbol{H}_{k,k=2},\ldots,K$，计算它与前一个输入矩阵的相关系数

$$\lambda_k = \frac{\left\|\boldsymbol{H}_k^H \boldsymbol{H}_{k-1}\right\|}{\left\|\boldsymbol{H}_k\right\|\left\|\boldsymbol{H}_{k-1}\right\|} \tag{9}$$

如果该相关系数 λ_k 大于设置的阈值 λ_{th}，则认为第 k 个和第 $k-1$ 个 $\boldsymbol{W}$、$\boldsymbol{V}$、$\boldsymbol{H}$ 之间近似相等，因此可以适用 $\boldsymbol{V}_{k-1}$ 替换右奇异矩阵 $\boldsymbol{V}_k$，从而实现降低 SVD 分解计算。如果相关系数 λ_k 小于设置的阈值 λ_{th} 时，我们则视作条件不足，$\boldsymbol{V}_k$ 仍然需要通过 SVD 分解计算得到。

对于第 k 个维度为 $M \times N$ 的矩阵相关系数，其计算复杂度计算为 $24MN+114$，具体计算可参考第 3.3.2 节，因此总计算复杂度为 $(24MN+114)K \approx 24MNK = 3538944$，计算复杂度远小于 SVD 分解，只要能够完成一次 SVD 省略，就可以降低系统的复杂度。

3.2.2 基于随机正交基的 SVD 算法优化

计算矩阵的奇异值分解通常会带来较高的运算复杂度，且复杂度将随矩阵规模增加而呈指数级增长[1][10]。因此，我们引入了基于随机正交基的 SVD（*Randomized Orthogonal Basis SVD, RSVD*）算法，对每个子输入矩阵 $\boldsymbol{H}_{j,k}$ 奇异值分解的计算步骤作简化。

RSVD 即截断式奇异值分解（*Truncated SVD*），其可行性来自于大型矩阵奇异值序列迅速下降的性质。对一复数输入矩阵 $\boldsymbol{A} \in \mathbb{C}^{m \times n}$，构造 p 个输入矩阵 $\boldsymbol{A}$ 列空间或行空间（取决于矩阵行数与列数的大小关系）的随机正交基，将输入矩阵 $\boldsymbol{A}$ 变换到此 p 个正交基张成的空间中，再用传统方法对其进行奇异值分解，其中 $p \le \min\{m,n\}$[6]。变换后矩阵相当于经过了降维操作，规模大大减小。因此进行奇异值分解的运算复杂度大大降低。此时，主要的运算复杂度集中在寻找随机正交基以及对输入矩阵作空间变换上。

在本问题中，对每一个需要作奇异值分解的 $M \times N$ 维输入矩阵 $\boldsymbol{H}_{j,k}$ 均有 $M < N$，因此对 RSVD 的步骤具体描述如下。

（1）随机生成 P 个 M 维复高斯随机行向量，构成高斯随机矩阵 $\boldsymbol{\Omega} \in \mathbb{C}^{P \times M}$。

（2）令矩阵 $\boldsymbol{Y} = \boldsymbol{A\Omega}$，并将其行向量组单位正交化，得到随机正交基矩阵 $\boldsymbol{Q} \in \mathbb{C}^{P \times N}$。理论上 $\boldsymbol{Q}$ 应当是严格的酉矩阵，然而在实际运算时，正交化过程必然会出现误差，因此有 $\boldsymbol{A} \approx \boldsymbol{AQ}^H\boldsymbol{Q}$。

（3）构造辅助矩阵 $\boldsymbol{B} = \boldsymbol{AQ}^H$，则有 $\boldsymbol{B} \in \mathbb{C}^{M \times L}$。使用传统方法（如 Golub-Kahan 方法）计算 $\boldsymbol{B}$ 的奇异值分解：

$$\boldsymbol{B} = \boldsymbol{U}\boldsymbol{S}\tilde{\boldsymbol{V}}^H \tag{10}$$

由于矩阵 $\boldsymbol{B}$ 的规模比原输入矩阵 $\boldsymbol{A}$ 的小，因此作传统奇异值分解的计算量大大降低。

（4）由于 $\boldsymbol{A} \approx \boldsymbol{BQ} = \boldsymbol{U}\boldsymbol{S}\tilde{\boldsymbol{V}}^H\boldsymbol{Q} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^H$，则可得到矩阵的 $\boldsymbol{A}$ 的截断奇异值序列及其对应的左右奇异向量。其中，$\boldsymbol{V} \in \mathbb{C}^{N \times P}$，是标准右奇异向量的前 P 列。

在实际计算过程中，考虑到 P 的值难以直接确定，因此采用了增量方式构建随机正交基矩阵 $\boldsymbol{Q}$，如算法 1 所示。即每次生成 1 个 M 维复高斯随机行向量，然后不断增加 $\boldsymbol{Q}$ 的行数，直至满足给定条件，算法迭代的轮数即为 P。停止条件 *FinishCondition* 将在下面详细讨论。

算法 1

$\boldsymbol{Q}^{(0)} \leftarrow []$

for i *from* 1 *to* M :

$\boldsymbol{\omega} \leftarrow GaussRandom(1, M)$

$\boldsymbol{y} \leftarrow \boldsymbol{\omega A}$

$\tilde{\boldsymbol{q}} \leftarrow \boldsymbol{y(I - (Q^{(i-1)})^H Q^{(i-1)})}$

$$\boldsymbol{Q^{(i)}} \leftarrow \begin{bmatrix} \boldsymbol{Q(i-1)} \\ \tilde{\boldsymbol{q}} / \left\| \tilde{\boldsymbol{q}} \right\|_2 \end{bmatrix}$$

if FinishCondition is true:

break

end

end

对辅助矩阵 $\boldsymbol{B}$ 进行 SVD 分解时迭代轮数的确定

下面描述在本问题中，如何具体确定对辅助矩阵进行 SVD 分解时的 QR 迭代轮数，即选取的矩阵 $\boldsymbol{A}$ 行空间的随机正交基个数。

通常，应当将 *FinishCondition* 设为判断 $\boldsymbol{A}$ 与 $\boldsymbol{AQ^HQ}$ 近似程度的准则，如相关系数等。但基于如下考虑，我们作了一定的简化。

首先，本问题中被分解的矩阵是 $\boldsymbol{H}_{j,k}$，它是一个 $M \times N$ 矩阵，且 $M < 10N$。在所有给定数据中，对于所有 $j = 1,2,...J$，$k = 1,2,...K$ 几乎总是满足 $rank(\boldsymbol{H}_{j,k}) = M$，即为满秩矩阵。

用一般方法求 Data1 中所有行块所有 $\boldsymbol{H}_{j,k}$ 的奇异值，分别绘制每个行块每个矩阵的前 L 个奇异值与所有奇异值之和的比例图像，如图 8 所示，其中取 $L = 2$。可以发现其比值几乎都集中在 0.6 左右。这表明前 L 个奇异值并没有包含矩阵的大部分特征。

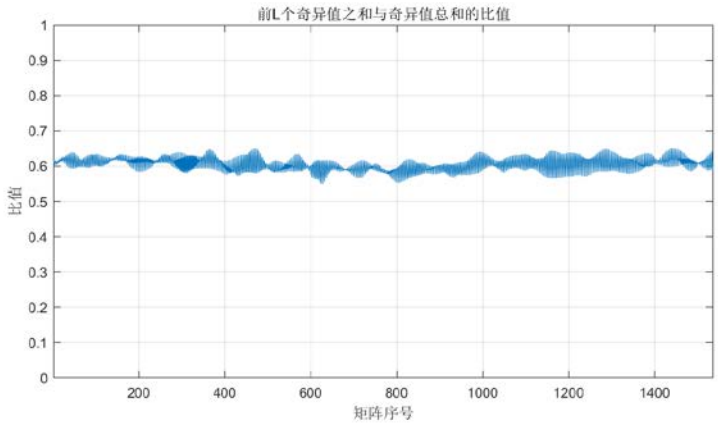

图 8 前 L 个奇异值之和与奇异值总和的比值

仍以 Data1 中所有行块所有 $\boldsymbol{H}_{j,k}$ 为例，使用 RSVD 算法优化迭代至 L 轮就停止，将所得前 L 个右奇异向量组成矩阵 $\overline{\boldsymbol{V}}_{j,k}$，计算其与标准数据中的 $\boldsymbol{V}_{j,k}$ 的误差。度量误差使用的指标为 $\rho_{j,k}(\boldsymbol{V})$，$j=1,2,...J$，$k=1,2,...,K$。

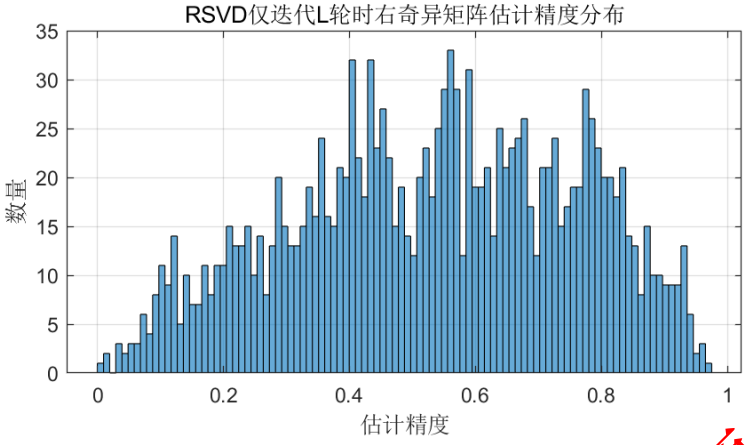

图 9 RSVD 仅迭代 L 轮时右奇异矩阵估计精度分布

所有 $\boldsymbol{H}_{j,k}$ 的右奇异向量估计精度分布如图 9 所示，从图中可以看出，仅迭代 L 轮计算出的右奇异矩阵与标准数据的估计精度大多非常低，几乎没有达到 0.99 以上的部分。这表明如果迭代至 $P=L$ 就停止，仅在奇异值分解步骤就会产生较大的估计误差，使得后续步骤无法进行。这与上述的两个条件相符合：$\boldsymbol{H}_{j,k}$ 几乎总为满秩，且前 L 个奇异值占奇异值总和的比值并不大。

由于选取的行空间随机正交基个数不会超过矩阵的行数，所以在本问题中至多进行 M 轮迭代就会停止。考虑到 M 和 $M-L$ 的值均较小，我们将迭代轮数选为定值，即 $P=M$。这样能够保证 SVD 分解的精度达到要求，同时额外的 $M-L$ 轮迭代并不会带来太大的运算复杂度增加。

此外，由于矩阵 $\boldsymbol{Q}$ 的的列数并没有减少，计算 $\boldsymbol{AQ}^{\boldsymbol{H}}\boldsymbol{Q}$ 以及 $\boldsymbol{A}$ 与 $\boldsymbol{AQ}^{\boldsymbol{H}}\boldsymbol{Q}$ 本身就会带来较大的额外复杂度。综上所述，将迭代轮数选为定值 M 是合理的。

3.2.3 基于矩阵间关联性的逆矩阵计算优化

由于逆矩阵计算的复杂度较高，且在矩阵求逆运算的一般方法中，计算复杂度随着矩阵阶数增加而大幅增加。因此我们将高阶矩阵转化为低阶矩阵求逆，尽管会增加乘法运算的次数，但是总体的计算复杂度会降低[11]。

根据前文已计算分析的矩阵关联性，可以利用矩阵的分块运算来优化高阶矩阵的求逆运算。

考虑将 $\boldsymbol{W}_k=\boldsymbol{V}_k\left(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)^{-1}$ 步骤转换为 $\left(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)\boldsymbol{W}_k^H=\boldsymbol{V}_k^H$，由 $\boldsymbol{V}_k=\left[\boldsymbol{V}_{1,k}\ \ ...\ \ \boldsymbol{V}_{j,k}\ \ ...\ \ \boldsymbol{V}_{J,k}\right]$,可以展开得

$$\begin{bmatrix}\boldsymbol{V}_{1,k}^H\\ \boldsymbol{V}_{2,k}^H\\ \boldsymbol{V}_{3,k}^H\\ \boldsymbol{V}_{4,k}^H\end{bmatrix}=\begin{bmatrix}\boldsymbol{V}_{1,k}^H\boldsymbol{V}_{1,k}+\sigma^2\boldsymbol{I} & \boldsymbol{V}_{1,k}^H\boldsymbol{V}_{2,k} & \boldsymbol{V}_{1,k}^H\boldsymbol{V}_{3,k} & \boldsymbol{V}_{1,k}^H\boldsymbol{V}_{4,k}\\ \boldsymbol{V}_{2,k}^H\boldsymbol{V}_{1,k} & \boldsymbol{V}_{2,k}^H\boldsymbol{V}_{2,k}+\sigma^2\boldsymbol{I} & \boldsymbol{V}_{2,k}^H\boldsymbol{V}_{3,k} & \boldsymbol{V}_{2,k}^H\boldsymbol{V}_{4,k}\\ \boldsymbol{V}_{3,k}^H\boldsymbol{V}_{1,k} & \boldsymbol{V}_{3,k}^H\boldsymbol{V}_{2,k} & \boldsymbol{V}_{3,k}^H\boldsymbol{V}_{3,k}+\sigma^2\boldsymbol{I} & \boldsymbol{V}_{3,k}^H\boldsymbol{V}_{4,k}\\ \boldsymbol{V}_{4,k}^H\boldsymbol{V}_{1,k} & \boldsymbol{V}_{4,k}^H\boldsymbol{V}_{2,k} & \boldsymbol{V}_{4,k}^H\boldsymbol{V}_{3,k} & \boldsymbol{V}_{4,k}^H\boldsymbol{V}_{4,k}+\sigma^2\boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{W}_{1,k}^H\\ \boldsymbol{W}_{2,k}^H\\ \boldsymbol{W}_{3,k}^H\\ \boldsymbol{W}_{4,k}^H\end{bmatrix}\tag{11}$$

根据前文可知，根据输入矩阵 H 的相关性，部分 $\boldsymbol{V}_{j,k}$ 可以由 $\boldsymbol{V}_{j,k-1}$ 替换，而无需计算 SVD，以降低计算复杂度。

这里首先假设当 $j=1,2$ 时可以替换，即 $\boldsymbol{V}_{1,k}=\boldsymbol{V}_{1,k-1},\boldsymbol{V}_{2,k}=\boldsymbol{V}_{2,k-1}$，由此可以对高阶矩阵进..行分块，有 $\begin{bmatrix}\boldsymbol{V}_{A,k-1}\\ \boldsymbol{V}_{B,k-1}\end{bmatrix}=\begin{bmatrix}\boldsymbol{A}_{k-1} & \boldsymbol{C}_{k-1}\\ \boldsymbol{B}_{k-1} & \boldsymbol{D}_{k-1}\end{bmatrix}\begin{bmatrix}\boldsymbol{W}_{A,k-1}\\ \boldsymbol{W}_{B,k-1}\end{bmatrix}$，$\begin{bmatrix}\boldsymbol{V}_{A,k}\\ \boldsymbol{V}_{B,k}\end{bmatrix}=\begin{bmatrix}\boldsymbol{A}_{k} & \boldsymbol{C}_{k}\\ \boldsymbol{B}_{k} & \boldsymbol{D}_{k}\end{bmatrix}\begin{bmatrix}\boldsymbol{W}_{A,k}\\ \boldsymbol{W}_{B,k}\end{bmatrix}$，

其中 $\boldsymbol{V}_{A,k-1}=\begin{bmatrix}\boldsymbol{V}_{1,k-1}^{H}\\ \boldsymbol{V}_{2,k-1}^{H}\end{bmatrix}$，$\boldsymbol{V}_{B,k-1}=\begin{bmatrix}\boldsymbol{V}_{3,k-1}^{H}\\ \boldsymbol{V}_{4,k-1}^{H}\end{bmatrix}$，$\boldsymbol{V}_{A,k}=\begin{bmatrix}\boldsymbol{V}_{1,k}^{H}\\ \boldsymbol{V}_{2,k}^{H}\end{bmatrix}$，$\boldsymbol{V}_{B,k}=\begin{bmatrix}\boldsymbol{V}_{3,k}^{H}\\ \boldsymbol{V}_{4,k}^{H}\end{bmatrix}$，

$$\boldsymbol{A}_{k-1}=\begin{bmatrix}\boldsymbol{V}_{1,k-1}^{H}\boldsymbol{V}_{1,k-1}+\sigma^{2}\boldsymbol{I} & \boldsymbol{V}_{1,k-1}^{H}\boldsymbol{V}_{2,k-1}\\ \boldsymbol{V}_{2,k-1}^{H}\boldsymbol{V}_{1,k-1} & \boldsymbol{V}_{2,k-1}^{H}\boldsymbol{V}_{2,k-1}+\sigma^{2}\boldsymbol{I}\end{bmatrix}=\begin{bmatrix}\boldsymbol{V}_{1,k}^{H}\boldsymbol{V}_{1,k}+\sigma^{2}\boldsymbol{I} & \boldsymbol{V}_{1,k}^{H}\boldsymbol{V}_{2,k}\\ \boldsymbol{V}_{2,k}^{H}\boldsymbol{V}_{1,k} & \boldsymbol{V}_{2,k}^{H}\boldsymbol{V}_{2,k}+\sigma^{2}\boldsymbol{I}\end{bmatrix}=\boldsymbol{A}_{k}。$$

将分块矩阵分解，有：

$$\begin{cases}\boldsymbol{V}_{A,k-1}=\boldsymbol{A}_{k-1}\boldsymbol{W}_{A,k-1}+\boldsymbol{C}_{k-1}\boldsymbol{W}_{B,k-1}\\ \boldsymbol{V}_{A,k}=\boldsymbol{V}_{A,k-1}=\boldsymbol{A}_{k}\boldsymbol{W}_{A,k}+\boldsymbol{C}_{k}\boldsymbol{W}_{B,k}=\boldsymbol{A}_{k-1}\boldsymbol{W}_{A,k}+\boldsymbol{C}_{k}\boldsymbol{W}_{B,k}\\ \boldsymbol{V}_{B,k}=\boldsymbol{B}_{k}\boldsymbol{W}_{A,k}+\boldsymbol{D}_{k}\boldsymbol{W}_{B,k}\end{cases}\tag{12}$$

求解得到 $\boldsymbol{W}_{A,k},\boldsymbol{W}_{B,k}$ 与 $\boldsymbol{W}_{A,k-1},\boldsymbol{W}_{B,k-1}$ 之间的关系如下：

$$\begin{cases}\boldsymbol{W}_{A,k}=(\boldsymbol{A}_{k-1}-\boldsymbol{C}_{k}\boldsymbol{D}_{k}^{-1}\boldsymbol{B}_{k})^{-1}(\boldsymbol{A}_{k-1}\boldsymbol{W}_{A,k-1}+\boldsymbol{C}_{k-1}\boldsymbol{W}_{B,k-1}-\boldsymbol{C}_{k}\boldsymbol{D}_{k}^{-1}\boldsymbol{V}_{B,k})\\ \boldsymbol{W}_{B,k}=\boldsymbol{D}_{k}^{-1}(\boldsymbol{V}_{B,k}-\boldsymbol{B}_{k}\boldsymbol{W}_{A,k})\end{cases}\tag{13}$$

其中 $\boldsymbol{W}_{A,k-1},\boldsymbol{W}_{B,k-1}$ 为 $k-1$ 时的输出矩阵 $\boldsymbol{W}$，其余矩阵均为 $\boldsymbol{V}_{k}^{H}\boldsymbol{V}_{k}+\sigma^{2}\boldsymbol{I}$ 的分块矩阵，在计算 $\hat{\boldsymbol{W}}=f_{2}(\hat{\boldsymbol{V}})$ 时均可视为已知矩阵，由此可以根据相关性优化 $\boldsymbol{V}_{k}^{H}\boldsymbol{V}_{k}+\sigma^{2}\boldsymbol{I}$ 这一高阶矩阵的求逆过程，而是转化为分块的低阶矩阵进行求逆运算。

同理，$\boldsymbol{V}_{j,k}=\boldsymbol{V}_{j,k-1},j=1,2,3,4$ 中有 1 个或 3 个值可以被替换时，同样可以使用分块矩阵运算来降低总计算复杂度，仅是分块矩阵的维度有所不同。

值得注意的是，如果不是从 $j=1$ 开始的连续几个 $\boldsymbol{V}_{j,k}$ 的值被替换，需要对原始矩阵方程做一些变换。

可以容易得到 $\begin{bmatrix}\boldsymbol{V}_{2,k}^{H}\\ \boldsymbol{V}_{4,k}^{H}\\ \boldsymbol{V}_{1,k}^{H}\\ \boldsymbol{V}_{3,k}^{H}\end{bmatrix}=\left[\begin{bmatrix}\boldsymbol{V}_{2,k}^{H}\\ \boldsymbol{V}_{4,k}^{H}\\ \boldsymbol{V}_{1,k}^{H}\\ \boldsymbol{V}_{3,k}^{H}\end{bmatrix}\begin{bmatrix}\boldsymbol{V}_{2,k} & \boldsymbol{V}_{4,k} & \boldsymbol{V}_{1,k} & \boldsymbol{V}_{3,k}\end{bmatrix}+\sigma^{2}\boldsymbol{I}\right]\begin{bmatrix}\boldsymbol{W}_{2,k}^{H}\\ \boldsymbol{W}_{4,k}^{H}\\ \boldsymbol{W}_{1,k}^{H}\\ \boldsymbol{W}_{3,k}^{H}\end{bmatrix}$，因此可以根据可替换 $\boldsymbol{V}_{j,k}$ 值的 j，对 $\boldsymbol{V}_{k}^{H}$ 和 $\boldsymbol{W}_{k}^{H}$ 进行行变换，由行变换后的 $\boldsymbol{V}_{k}^{H}$ 计算 $\boldsymbol{V}_{k}^{H}\boldsymbol{V}_{k}+\sigma^{2}\boldsymbol{I}$，最后代入公式进行求解。

针对 $j=1,2,3,4$ 均能够由 $\boldsymbol{V}_{j,k-1}$ 替换 $\boldsymbol{V}_{j,k}$ 的情况，即无需通过 $\boldsymbol{W}_{k}=\boldsymbol{V}_{k}\left(\boldsymbol{V}_{k}^{H}\boldsymbol{V}_{k}+\sigma^{2}\boldsymbol{I}\right)^{-1}$ 计算，可直接由 $\boldsymbol{W}_{k}=\boldsymbol{W}_{k-1}$ 得到。

针对 $j=1,2,3,4$ 均无法由 $\boldsymbol{V}_{j,k-1}$ 替换 $\boldsymbol{V}_{j,k}$ 的情况，线性矩阵方程 $\left(\boldsymbol{V}_{k}^{H}\boldsymbol{V}_{k}+\sigma^{2}\boldsymbol{I}\right)\boldsymbol{W}_{k}^{H}=\boldsymbol{V}_{k}^{H}$ 可以使用迭代法求解，具体在下一节中进行说明。

3.2.4 基于逐次超松弛迭代法的逆矩阵求解优化

针对无法由$\boldsymbol{V}_{j,k-1}$替换$\boldsymbol{V}_{j,k}$的情况，对于线性矩阵方程$\left(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)\boldsymbol{W}_k^H=\boldsymbol{V}_k^H$可以使用迭代法，利用设计好的迭代公式所产生的迭代序列逐步逼近精确解，以避免高阶一般矩阵的求逆运算，从而潜在地降低整体计算流程的计算复杂度[12]。

由于$\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}$是对称正定矩阵，因此对于求解的迭代方法始终收敛，可以得到相应的解。第一方面，可以考虑使用较为简便的 Jacobi 迭代法，是众多经典迭代法中较早且较简单的一种，计算公式简单，每迭代一次只需计算一次矩阵和向量的乘法，比较容易并行计算。

使用 Jacobi 迭代法求解$\boldsymbol{AX}=\boldsymbol{B}$时，首先将矩阵$\boldsymbol{A}$进行线性拆分，分解成一个对角阵$\boldsymbol{D}$，一个严格下三角阵$\boldsymbol{L}$以及一个严格上三角阵$\boldsymbol{U}$。

$$\boldsymbol{A}=\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,i} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & \cdots & \cdots & a_{2,n} \\ \vdots & \ddots & \ddots & \vdots & \vdots & \vdots \\ a_{i,1} & \cdots & a_{i,i-1} & a_{i,i} & \ddots & a_{i,n} \\ \vdots & & \vdots & \ddots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,i-1} & \ddots & a_{n,n-1} & a_{n,n} \end{bmatrix} \tag{14}$$

$$\boldsymbol{D}=\operatorname{diag}\left(a_{11},a_{22},\cdots,a_{nn}\right) \tag{15}$$

$$\boldsymbol{L}=\begin{bmatrix} 0 & & & & & 0 \\ a_{2,1} & 0 & & & & \\ \vdots & \ddots & \ddots & & & \\ a_{j,1} & \cdots & a_{j,j-1} & 0 & & \\ \vdots & & \vdots & \ddots & \ddots & \\ a_{n,1} & \cdots & a_{n,j-1} & \cdots & a_{n,n-1} & 0 \end{bmatrix} \tag{16}$$

$$\boldsymbol{U}=\begin{bmatrix} 0 & a_{1,2} & \cdots & a_{1,j} & \cdots & a_{1,n} \\ & 0 & \ddots & \vdots & & \vdots \\ & & \ddots & a_{j-1,j} & \cdots & a_{j-1,n} \\ & & & 0 & \ddots & \vdots \\ & & & & \ddots & a_{n-1,n} \\ 0 & & & & & 0 \end{bmatrix} \tag{17}$$

于是：

$$\begin{aligned} &\boldsymbol{A}=\boldsymbol{D}+\boldsymbol{L}+\boldsymbol{U} \\ &\boldsymbol{DX}=-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{X}+\boldsymbol{B} \\ &\boldsymbol{X}=-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{X}+\boldsymbol{D}^{-1}\boldsymbol{B} \\ &\boldsymbol{X}^{(k+1)}=-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{X}^{(k)}+\boldsymbol{D}^{-1}\boldsymbol{B},k=0,1,2,\ldots \end{aligned}$$

迭代运算直到$\lim\limits_{k\to\infty}\left\|\boldsymbol{X}^{(k+1)}-\boldsymbol{X}^{(k)}\right\|<\varepsilon$时可以视作获得精确解。

对于矩阵方程 $(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I})\boldsymbol{W}_k^H=\boldsymbol{V}_k^H$，迭代公式即：

$$\boldsymbol{W}_k^H(t+1)=-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{W}_k^H(t)+\boldsymbol{D}^{-1}\boldsymbol{V}_k^H,t=0,1,2,...,T \tag{18}$$

此外，逐次超松弛(*Successive Over Relaxation*)迭代法，简称 SOR 方法，与 Jacobi 迭代法类似，也是一种经典的迭代算法。它是为了解决大规模系统的线性等式提出来的，在 Gauss-Seidel 迭代法基础上为提高收敛速度，采用加权平均而得到的新算法。由于超松弛迭代法公式简单，编写程序容易，很多工程学、计算数学中都会应用超松弛迭代方法。使用 SOR 迭代法时引入松弛因子如下，如果松弛因子选取合适，则会大大缩短计算时间[13][14]。

$$\boldsymbol{X}^{(k+1)}=\boldsymbol{X}^{(k)}+\omega\left(\tilde{\boldsymbol{X}}^{(k+1)}-\boldsymbol{X}^{(k)}\right) \tag{19}$$

取 Gauss-Seidel 迭代格式，令 $\tilde{\boldsymbol{X}}^{(k+1)}=-(\boldsymbol{D}+\boldsymbol{L})^{-1}\boldsymbol{U}\boldsymbol{X}^{(\boldsymbol{k})}+(\boldsymbol{D}+\boldsymbol{L})^{-1}\boldsymbol{B}$，其中 $\boldsymbol{D}$、$\boldsymbol{L}$、$\boldsymbol{U}$ 与 Jacobi 迭代法中一致，分别为对角阵、严格下三角阵和严格上三角阵。

于是有

$$\boldsymbol{X}^{(k+1)}=(1-\omega)\boldsymbol{X}^{(k)}+\omega\boldsymbol{D}^{-1}\left(-\boldsymbol{L}\boldsymbol{X}^{(k+1)}-\boldsymbol{U}\boldsymbol{X}^{(k)}+\boldsymbol{B}\right) \tag{20}$$

整理得

$$\boldsymbol{X}^{(k+1)}=(\boldsymbol{D}+\omega\boldsymbol{L})^{-1}[(1-\omega)\boldsymbol{D}-\omega\boldsymbol{U}]\boldsymbol{X}^{(k)}+\omega(\boldsymbol{D}+\omega\boldsymbol{L})^{-1}\boldsymbol{B} \tag{21}$$

一个下三角矩阵的行列式计算复杂度会大大降低，且求逆与 ω 数乘仅需在迭代前计算一次。

对于矩阵方程 $(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I})\boldsymbol{W}_k^H=\boldsymbol{V}_k^H$，迭代公式即：

$$\boldsymbol{W}_k^H(t+1)=(\boldsymbol{D}+\omega\boldsymbol{L})^{-1}((1-\omega)\boldsymbol{D}-\omega\boldsymbol{U})\boldsymbol{W}_k^H(t)+\omega(\boldsymbol{D}+\omega\boldsymbol{L})^{-1}\boldsymbol{V}_k^H,t=0,1,2,...,T \tag{22}$$

对于 Jacobi 迭代法，尽管计算简单，但是收敛速度较慢，在实际应用中使用得并不多。而对于 SOR 迭代法，其应用更为广泛，因为通过选择适当的松弛因子可以大大减小迭代矩阵谱半径的值，极大提高迭代法的收敛速度。尽管，一般来说 SOR 迭代的渐进收敛速度对松弛因子较为敏感，寻找最佳松弛因子对于快速求解方程组问题是至关重要也是极其困难的，但是松弛因子已有一些常见的经验值[11][12]，可以适用于广泛的矩阵方程求解。

比较 Jacobi 迭代法以及 SOR 迭代法，可以发现前者收敛速度较慢，即迭代次数较多，但每轮迭代中的计算复杂度较低；而后者收敛速度较快，即迭代次数较少，但每轮迭代中的计算复杂度较高。总计算复杂度等于迭代次数与每轮迭代中的计算复杂度的乘积，关于 Jacobi 迭代法以及 SOR 迭代法的计算复杂度的定量分析在 3.3.4 小节中具体说明，在本问题中因 SOR 迭代法的性能优越，故选择 SOR 迭代法对逆矩阵求解进一步优化[16]。

总之，使用 SOR 迭代法等迭代法求解线性矩阵方程组时能够优化逆矩阵的求解，在本问题中矩阵组 $\boldsymbol{H}=\{\boldsymbol{H}_{j,k}\}$ 的维度，或者，矩阵组内各个矩阵 $\boldsymbol{H}_{j,k}$ 的维度继续提升时，迭代法的优势将会进一步体现[17]。

3.3 分步复杂度计算

3.3.1 复杂度标准化

实数基本运算的计算复杂度由表 3 给出[8]。

表 2 实数基本运算复杂度表

运算类型	计算复杂度
加(减)法	1
乘法	3
倒数	25
平方根	25
自然指数	25
自然对数	25
正弦	25
余弦	25
其它	100

由此，我们导出复数运算及复矩阵基本运算的复杂度，如表 3 所示。

表 3 复数基本运算复杂度表

运算	计算方法	运算数量	复杂度
复数加法	$(a+bi)+(c+di)$ $=(a+c)+(c+d)i$	2 次加法	$1\times 2=2$
复数乘法	$(a+bi)(c+di)$ • $=(ac-bd)+(bc+ad)i$	4 次乘法 2 次加法	$3\times 4+2\times 1=14$
复数求模	$\lvert a+bi\rvert=\sqrt[2]{a^2+b^2}$	2 次乘法 1 次加法 1 次平方根	$3\times 2+1\times 1+25=32$
实数乘 n 维复向量	$k\boldsymbol{a}_n$	$2n$ 次乘法	$3\times 2n=6n$
实数乘 $m\times n$ 维矩阵	$k\boldsymbol{A}_{m\times n}$	$2mn$ 次乘法	$3\times 2mn=6mn$
n 维复向量 求模	$\sqrt[2]{\boldsymbol{a}_n^H\boldsymbol{a}_n}$ $=\sqrt[2]{a_1^2+b_1^2+\ldots+a_n^2+b_n^2}$	$2n$ 次乘法 $2(n-1)$ 次加法 1 次平方根	$3\times 2n+1\times 2(n-1)+25$ $=8n+23$
两个 n 维 复向量加减	$\boldsymbol{a}_n+\boldsymbol{b}_n$	n 次复数加法	$2\times n=2n$
两个 n 维 复向量内积	$\boldsymbol{a}_n^H\boldsymbol{b}_n$	n 次复数乘法 $n-1$ 次复数加法	$14n+2(n-1)=16n-2$
$m\times n$ 维矩阵 左乘 n 维列向量	$\boldsymbol{A}_{m\times n}\boldsymbol{b}_n$	求 m 次 n 维复向量 内积	$(16n-2)\times m$
$m\times n$ 维矩阵 右乘 m 维行向量	$\boldsymbol{b}_m\boldsymbol{A}_{m\times n}$	求 n 次 m 维复向量 内积	$(16m-2)\times n$
$m\times n$ 维矩阵 左乘 $n\times k$ 维矩阵	$\boldsymbol{A}_{m\times n}\boldsymbol{B}_{n\times k}$	求 k 次 $m\times n$ 维矩阵 左乘 n 维列向量	$(16n-2)\times m\times k$

上表为本节所述的所有复杂度计算提供了依据。

3.3.2 求矩阵相关系数的复杂度

对于任意矩阵 $\boldsymbol{A},\boldsymbol{B}\in\mathbb{C}^{m\times n}$，计算使其相关系数时首先需要将其化为 mn 维列向量 $\boldsymbol{\alpha}$ 和 $\boldsymbol{\beta}$。其次，需要分别计算 $\boldsymbol{\alpha}^{H}\boldsymbol{\beta}$、$\boldsymbol{\alpha}$、$\boldsymbol{\beta}$ 的 2 范数，再做倒数和乘法。

因此，计算一次矩阵相关系数需要的复杂度包含 1 次 n 维复向量内积、1 次求虚数的模、2 次求 n 维复向量的模、1 次倒数、1 次实数乘法。因此有：

$$\begin{aligned}C_{judge}&=(8mn\text{-}2)+32+2\times(8mn+23)+25+3\\&=24mn+114\end{aligned}$$

对本问题中 $J\times K$ 维矩阵组中的 JK 个 $M\times N$ 维矩阵，需要对所有横向相邻的矩阵计算相关系数，即需要计算 $J(K-1)$ 次。因此，计算相关系数矩阵的总复杂度为：

$$C_{judge\_mat}=J(K-1)C_{judge}=J(K-1)(24MN+114)$$

3.3.3 单个矩阵 RSVD 操作的复杂度

本小节描述 RSVD 算法的总体运算复杂度。根据 3.2.2 小节所述模型，RSVD 算法的复杂度主要由两个步骤组成：第一步，计算随机正交基矩阵 $\boldsymbol{Q}$；第二步，对辅助矩阵 $\boldsymbol{B}$ 作奇异值分解。其中，每步的复杂度又由更多子步骤组成。

为表述清晰，根据 RSVD 的原理定义了 6 种基本操作，其各自的运算复杂度用表 4 中的符号。

表 4 RSVD 分步操作复杂度

操作	复杂度	迭代次数
计算 $\boldsymbol{y}=\boldsymbol{\omega A}$	C_1	P
计算 $\tilde{\boldsymbol{q}}=\boldsymbol{y(I-(Q^{(i-1)})^{H}Q^{(i-1)})}$	C_2	P
计算 $\tilde{\boldsymbol{q}}/\left\|\tilde{\boldsymbol{q}}\right\|_2$	C_3	P
计算 $\boldsymbol{B}=\boldsymbol{AQ}^{H}$	C_4	1
计算 $\boldsymbol{B}=\boldsymbol{U}\boldsymbol{S}\tilde{\boldsymbol{V}}^{H}$	C_5	1
计算 $\boldsymbol{V}^{H}=\tilde{\boldsymbol{V}}^{H}\boldsymbol{Q}$	C_6	1

对于表中的所有单步操作的复杂度计算如下：

（1）计算 $\boldsymbol{y}=\boldsymbol{\omega A}$

C_1 为 1 次 $M\times N$ 维矩阵右乘 M 维行向量的运算复杂度，其值为：

$$C_1=(16M-2)N$$

（2）计算 $\tilde{\boldsymbol{q}}=\boldsymbol{y(I-(Q^{(i-1)})^{H}Q^{(i-1)})}$

由于向量乘单位矩阵无需计算，且单位矩阵中有较多的 0 元素，因此 C_2 步骤可化为下式的计算，以节省冗余的加法：

$$\tilde{q}\leftarrow \mathrm{y}-\mathrm{y}(Q^{(i-1)})^{H}Q^{(i-1)} \tag{23}$$

共有 1 次 $N\times P$ 矩阵与 $P\times N$ 矩阵乘法、1 次 N 维行向量与 $N\times N$ 矩阵乘法、1 次 N 维复向量减法，因此其复杂度为：

$$C_2=(16P-2)N^2+(16N-2)N+2N=(16P+16-2)N^2$$

（3）计算 $\tilde{\boldsymbol{q}}/\|\tilde{\boldsymbol{q}}\|_2$

C_3 包括 1 次求 N 维复向量模，1 次取倒数，1 次实数乘 N 维复向量的复杂度，其值为：

$$C_3=(8N+23)+25+6N=14N+48$$

（4）计算 $\boldsymbol{B}=\boldsymbol{AQ}^{\boldsymbol{H}}$

C_4 步骤的复杂度为 1 个 $M\times N$ 矩阵与 $N\times P$ 矩阵的乘法，因此：

$$C_4=(16N-2)MP$$

（5）计算 $\boldsymbol{B}=\boldsymbol{U}\boldsymbol{S}\tilde{\boldsymbol{V}}^{\boldsymbol{H}}$

本步骤的复杂度计算比较繁琐。由于矩阵 $\boldsymbol{B}$ 的规模较小，因此我们采用 Golub-Kahan 双对角化与 QR 迭代法对其进行奇异值分解，其原理已在 3.2.1 小节提及。

根据 Golub-Kahan 方法的原理，对一任意矩阵 $\boldsymbol{B}\in\mathbb{C}^{M\times N}$，当 $M>N$ 和 $M\le N$ 时对其进行奇异值分解的方法是类似的。因此不妨在 $M>N$ 情况下考虑其运算复杂度。该复杂度由以下部分组成：

（i）计算矩阵 $\boldsymbol{B}$ 的 QR 分解

Golub-Kahan 采用 Householder 变换计算矩阵的 QR 分解[6]，其方法同样适用于本问题。原因在于本问题中被分解的矩阵均为稠密矩阵，且所希望求得的结果仅从右奇异向量取出，无需计算 $\boldsymbol{Q}$ 矩阵。因此，在本问题中的奇异值变换十分适合采用 Householder 变换进行分解。

根据 Householder 变换的原理，将矩阵 $\boldsymbol{B}$ 上三角化（即求矩阵 $\boldsymbol{R}$ ）只需将 $\boldsymbol{B}$ 的第 k 个列向量的第 $k\sim M$ 维作 $M-k+1$ 阶 Householder 变换即可，$k=1,2,\ldots N$。一般地，对 n 阶向量 $\boldsymbol{x}$ 作 n 阶 Householder 变换主要有四步计算：

① 计算向量 $\boldsymbol{x}$ 的模，复杂度为 $8n+23$。

② 构造辅助向量 $w=x-ke_1$，其中 $e_1=(1,0,0,0,0,\ldots)^T$。由于不计算赋值步骤的运算复杂度，因此只需作 1 次加法，即：

$$\begin{bmatrix}w_1\\w_2\\\cdots\\w_n\end{bmatrix}=\begin{bmatrix}x_1-k\\x_2\\\cdots\\x_n\end{bmatrix}\tag{24}$$

其运算复杂度为 1。

③ 计算 Householder 变换矩阵，并求结果向量 y：

$$\begin{cases}\boldsymbol{H}=\boldsymbol{I}-2(\boldsymbol{w}\boldsymbol{w}^{\boldsymbol{H}})/\|\boldsymbol{w}\|_2\\ \boldsymbol{y}=\boldsymbol{H}\boldsymbol{x}\end{cases}\tag{25}$$

根据公式(26)，实际上有：

$$\begin{aligned} \boldsymbol{y} &= (\boldsymbol{I} - 2(\boldsymbol{w}\boldsymbol{w}^{\boldsymbol{H}})/\|\boldsymbol{w}\|_2)\boldsymbol{x} \\ &= \boldsymbol{x} - \frac{2}{\|\boldsymbol{w}\|_2}\boldsymbol{w}\boldsymbol{w}^{\boldsymbol{H}}\boldsymbol{x} \end{aligned} \tag{26}$$

包括 1 次复向量求模的平方（求模运算中去掉开平方根的步骤），1 次倒数计算，1 次实数乘法，$\sum_{k=1}^{n} k$ 次复数乘法，1 次 n 阶矩阵与 n 维向量的积，及 1 次 n 维向量的减法。复杂度为：

$$(8n-2)+25+3+n(n+1)/2+(16n-2)n+2n=n(33n+17)/2+26$$

因此，对一个 n 阶矩阵作 Householder 变换的复杂度为：

$$8n+23+1+n(33n+17)/2+26=33n(n+1)/2+50$$

从而对 $M\times N$ 维矩阵作上三角变换，即计算矩阵 $\boldsymbol{B}$ 的 QR 分解的复杂度为：

$$\sum_{n=M-N+1}^{M}\frac{33}{2}n(n+1)+50$$

（ii）对矩阵 $\boldsymbol{R}$ 作双对角化

$\boldsymbol{R}$ 是 $M\times N$ 维上三角矩阵，将其双对角化相当于对其行向量作 Householder 变换。类似地，可以求其复杂度为：

$$\sum_{n=2}^{M-1}\frac{33}{2}n(n+1)+50$$

（iii）使用 QR 迭代将双对角阵 $\boldsymbol{T}$ 化为对角阵

该步骤对上一步生成的双对角矩阵 $\boldsymbol{T}$ 进行 QR 迭代，最终变换为对角矩阵，从而获得奇异值和左、右奇异向量。在每一次迭代中，需要做 $2N$ 次 Givens 变换，而 1 次对复向量的 Givens 变换包含 2 次类似复数模的运算、2 次倒数运算、2 次复数数乘运算，以及 2 次复数乘法、1 次加法。因此一轮迭代的运算复杂度为：

$$2N(32\times 2+25\times 2+2\times 3+14+1)=270N$$

由于前述步骤将矩阵进行了双对角化，因此在迭代步骤中复杂度大大降低。迭代的停止与否决定于奇异值的相对误差 ρ，其定义如下：

$$\rho(k)=\max\left(\frac{|\tilde{\sigma}_i-\sigma_i|}{\sigma_i}\right)\quad i\le n \tag{27}$$

其中，$\tilde{\sigma}_i$ 为当前计算得到的第 i 个奇异值，σ_i 为第 i 个理想奇异值，n 为矩阵维度。

使用 QR 迭代法求奇异值时，大约经过 30 次迭代后相对误差会逐渐收敛[18]。在 8 阶矩阵的情况下，经过 30 次迭代相对误差收敛至-27dB。虽然 Golub-Kahan 方法求奇异值的精度随矩阵规模增大而降低，但考虑到本模型中矩阵规模较小，因此将平均迭代次数统一选为 30，已经可以达到较高的精度。因此，使用 QR 迭代将双对角阵 $\boldsymbol{T}$ 化为对角阵的复杂度为 $270N\times 30=8100N$ 。

综合步骤(i)~(iii)，可求得用 Golub-Kahan 方法进行奇异值分解的运算复杂度。由于本模型中需要求解 SVD 的矩阵 $\boldsymbol{B}$ 为 $M\times P$ 矩阵，且 $P\le M$，因此实际上是对 $\boldsymbol{B}$ 作 LQ 分解并对下三角矩阵 $\boldsymbol{L}$ 作进一步处理，从而计算 $\boldsymbol{B}=\boldsymbol{U}\boldsymbol{S}\widetilde{\boldsymbol{V}}^{\boldsymbol{H}}$ 步骤的复杂度为：

$$C_5=\sum_{n=M-P+1}^{M}[\frac{33}{2}n(n+1)+50]+\sum_{n=2}^{M-1}[\frac{33}{2}n(n+1)+50]+8100M$$

（6）计算$\boldsymbol{V}^{\boldsymbol{H}}=\widetilde{\boldsymbol{V}}^{\boldsymbol{H}}\boldsymbol{Q}$

C_6 步骤的复杂度为 1 个 $N\times P$ 矩阵与 $P\times P$ 矩阵相乘，因此：

$$C_6=(16P-2)NP$$

综合所有单步步骤，且根据表 4 中的迭代次数，可求得对一 $M\times N$ 矩阵 $\boldsymbol{A}$ 作 RSVD 分解的运算复杂度为：

$$\begin{aligned}C_{svd}&=P(C_1+C_2+C_3)+(C_4+C_5+C_6)\\&=2P\left[(8P+7)N^2+(8M+8P+5)N+(8N-1)M+24\right]+8100N\\&+\sum_{n=M-P+1}^{M}[\frac{33}{2}n(n+1)+50]+\sum_{n=2}^{M-1}[\frac{33}{2}n(n+1)+50]\end{aligned}$$

如果直接对原矩阵使用 Colub-Kahan 方法进行 SVD 分解，则其复杂度仅由 C_5 构成，即：

$$C_{colub}=\sum_{n=N-M+1}^{N}[\frac{33}{2}n(n+1)+50]+\sum_{n=2}^{N-1}[\frac{33}{2}n(n+1)+50]$$

因原矩阵矩阵规模较大，造成直接计算的复杂度比较高。在本问题中，取 $P=M=4$，N = 4，则 $C_{svd}=1,363,181$，$C_{colub}=2,221,827$。相比使用 Colub-Kahan 方法，RSVD 方法的运算复杂度下降了 38.65%。

3.3.4 矩阵求逆优化方法的复杂度

在分析求逆优化方法之前，考虑一般矩阵求逆过程，可以通过获取伴随矩阵和计算行列式值后利用 $\boldsymbol{A}^{-1}=\frac{\boldsymbol{A}^{*}}{|\boldsymbol{A}|}$ 进行矩阵求逆运算，此处根据复数计算复杂度表格得出计算复杂度。首先，复数倒数 $\frac{1}{a+bj}=\frac{a-bj}{a^2+b^2}$ 需要做 4 次实数乘法，2 次实数加法和 1 次实数倒数，计算复杂度为 $4\times3+2+25=39$。此外，通过高斯消元法进行行变换获得伴随矩阵，对于一个 n 阶方阵，其复杂度计算为 $354n^2-276n$。然后，对于 n 阶方阵行列式的计算，总共有 $n!$ 项 n 个数的乘法，并求和，其计算复杂度为 $n!(14(n-1))+2(n!-1)$。因此，矩阵求逆运算的总计算复杂度为

$$354n^2-276n+n!(14(n-1))+2(n!-1)+39=(14n-12)n!+354n^2-276n+37$$

由 $LJ=8,N=64$，最终求解 $\boldsymbol{W}_k=\boldsymbol{V}_k\left(\boldsymbol{V}_k^{H}\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)^{-1}$ 的总计算复杂度为

$$C_{general\_inv}(14LJ-12)LJ!+354LJ^2-276LJ+37+(16LJ-2)*N*LJ=4116997$$

在基于矩阵间关联性的逆矩阵计算优化中，根据计算复杂度表格以及分块矩阵计算公式，由 $LJ=8,N=64$，可以得到当 $\boldsymbol{V}_{j,k}=\boldsymbol{V}_{j,k-1},j=1,2,3,4$ 中有 1 个值可以被替换时，对于每个 $k,k=1,2,...,K$ 的计算复杂度为 146666；当 $\boldsymbol{V}_{j,k}=\boldsymbol{V}_{j,k-1},j=1,2,3,4$ 中有 1 个值可以被替换时，对于每个 $k,k=1,2,...,K$ 的计算复杂度为 111082；当 $\boldsymbol{V}_{j,k}=\boldsymbol{V}_{j,k-1},j=1,2,3,4$ 中有 1 个值可以被替换时，对于每个 $k,k=1,2,...,K$ 的计算复杂度为 168226。

此外，为了进一步说明基于迭代法的逆矩阵求解优化，我们对 Jacobi 迭代法以及 SOR 迭代法的计算复杂度进行了定量分析，计算 $k,k=1,2,...,K$ 的计算复杂度。Jacobi 迭代法中的求逆步骤已简化为对一个 $N\times N$ 对角阵的求逆运算，即对所有对角线元素进行 1 次复数倒数运算，并且只需要再迭代前计算一次，其计算复杂度为 $39N$。对角阵 $N\times N$ 与一般矩阵 $N\times K$ 相乘,实质为 $N\times K$ 次复数数乘，计算复杂度为 $14NK$。针对 $(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I})\boldsymbol{W}_k^H=\boldsymbol{V}_k^H$，有 $\boldsymbol{W}_k^H(t+1)=-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{W}_k^H(t)+\boldsymbol{D}^{-1}\boldsymbol{V}_k^H,t=0,1,2,...,T$，其中 $\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}$ 为 $LJ\times LJ$ 方阵，$\boldsymbol{W}_k^H$、$\boldsymbol{V}_k^H$ 为 $LJ\times N$ 的矩阵，则每次迭代的计算复杂度为:

$$\begin{aligned}C_{t\_Jacobi}&=14*LJ*LJ+(16*LJ-2)*LJ*N+14*LJ*N+2*LJ*N\\&=16*LJ^2*N+14*LJ^2+14*LJ*N\end{aligned}$$

由 MATLAB 仿真可以得到 Jacobi 迭代法的迭代次数较慢，对于本题的 6 组数据集，需要 $T=17$ 次左右迭代，如图 10 所示。

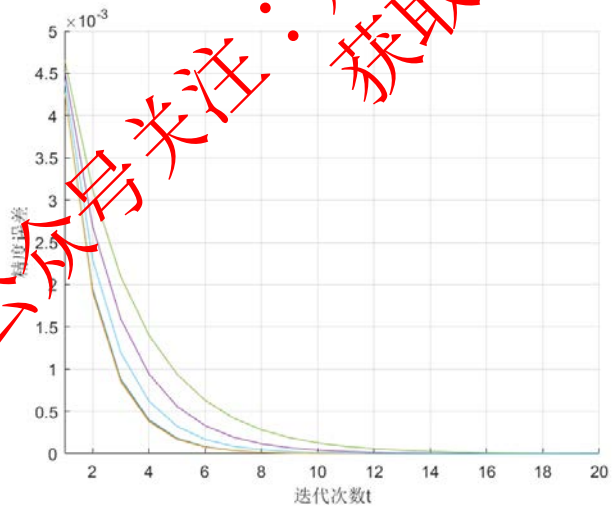

图 10 Jacobi 迭代法的收敛曲线图

代入 $LJ=8,N=64$,则总共的计算复杂度为：

$$\begin{aligned}C_{Jacobi}&=\left(16*LJ^2*N+14*LJ^2+14*LJ*N\right)*T+39*LJ\\&=1251512\end{aligned}$$

。

同样的，在 SOR 迭代法中有：

$$\boldsymbol{W}_k^H(t+1)=(\boldsymbol{D}+\omega\boldsymbol{L})^{-1}((1-\omega)\boldsymbol{D}-\omega\boldsymbol{U})\boldsymbol{W}_k^H(t)+\omega(\boldsymbol{D}+\omega\boldsymbol{L})^{-1}\boldsymbol{V}_k^H,t=0,1,2,...,T$$

其中$(\boldsymbol{D}+\omega\boldsymbol{L})^{-1}$为一个下三角矩阵的逆矩阵，其行列式计算复杂度较低。一些迭代前运算的计算复杂度为：

$$\begin{aligned}C_{bef\_iter}&=354*LJ*LJ-276*LJ+14(LJ-1)+39+2*LJ*(LJ-1)/2*3\\&\quad+2*\left(LJ+\left(LJ*LJ-LJ\right)\right)*3+6*LJ*LJ\\&=366*LJ^2-262*LJ+25\end{aligned}$$

每次迭代中的计算复杂度为：

$$\begin{aligned}C_{t\_SOR}&=\left(8*LJ*LJ+6*LJ\right)*N+2*(16*LJ-2)*LJ*N+2*LJ*N\\&=40*LJ^2*N+4*LJ*N\end{aligned}$$

取经验值$\omega=1.2$，由 MATLAB 仿真可以得到对于本题的 6 组数据集，SOR 迭代法仅需要$T=6$次左右迭代，如图 11 所示。

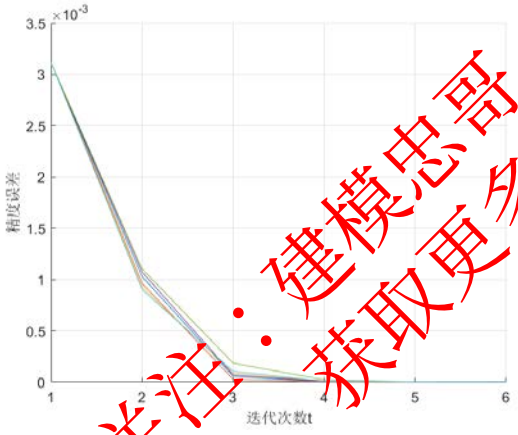

图 11 SOR 迭代法的收敛曲线图

代入，则总计算复杂度为：

$$C_{SOR}=\left(40*LJ^2*N+4*LJ*N\right)*T+366*LJ^2-262*LJ+25=1016681$$

因此，尽管 SOR 迭代法中的每轮迭代的计算复杂度相对于 Jacobi 迭代法较高，但 SOR 迭代法收敛速度快，所以其总计算复杂度低于 Jacobi 迭代法，因此选择 SOR 迭代法求解逆矩阵。

综合基于矩阵间关联性的逆矩阵计算优化以及基于逐次超松弛迭代法的逆矩阵求解优化，可以计算得到矩阵求逆部分的总体计算复杂度。首先对于$j=1,2,3,4$均能够由替换$\boldsymbol{V}_{j,k}$的情况，即无需通过$\boldsymbol{W}_k=\boldsymbol{V}_k\left(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)^{-1}$计算，可直接由$\boldsymbol{W}_k=\boldsymbol{W}_{k-1}$得到，此部分的计算复杂度为 0。其次，对于$j=1,2,3,4$中部分能够由$\boldsymbol{V}_{j,k-1}$替换$\boldsymbol{V}_{j,k}$的情况，若$k=1,2,...,K$对应的$\boldsymbol{V}_{j,k}$中出现 1 个、2 个、3 可以被替换的情况的次数分别为r_1,r_2,r_3，则此部分的计算复杂度为$C_{r_1,r_2,r_3}=146666r_1+111082r_2+168226r_3$。最后，针对$j=1,2,3,4$均无法由$\boldsymbol{V}_{j,k-1}$替换$\boldsymbol{V}_{j,k}$的情况，可以使用 SOR 迭代法求解线性矩阵方程$\left(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)\boldsymbol{W}_k^H=\boldsymbol{V}_k^H$，假设出现$r_0$次，则此部分计算复杂度为$C_{r_0}=1016681r_0$。

对于每一数据集，矩阵求逆优化方法的总计算复杂度为：

$$C_{inv}=C_{r_0}+C_{r_1,r_2,r_3}$$

3.4 基于相关系数的矩阵组计算复杂度优化模型

针对问题一的要求及对矩阵相关性的分析，我们建立了相关组矩阵计算优化模型，可以通过模型优化和算法的优化，实现在复杂度大幅度降低的同时，保持非常高的精确度。在本模型中：

第一，我们提出了基于矩阵间关联性的 SVD 计算优化，可以**低成本**地提取出同一行块的 K 个矩阵中邻近矩阵的相关特性，并使用邻近矩阵的奇异值分解结果替代当前矩阵的右奇异矩阵。该算法在**大幅度降低** SVD 分解调用频次的同时，能够保持 99.9%子结果矩阵 $\boldsymbol{W}_k$ 的估计精度达标。

第二，我们引入了基于随机正交基的 SVD 优化算法，可以通过减少非必要参数的计算和减小 SVD 输入矩阵的维度，在有效降低运算复杂度的同时，能够保证分解结果和标准 SVD 分解结果误差在10^{-12} 以内。我们通过比较多种 SVD 优化算法的原理并对其进行实际的性能测试，认为该算法通过迭代方法，可规避左奇异矩阵 $\boldsymbol{U}$ 和奇异值矩阵 $\boldsymbol{S}$ 的非必要计算，与本题只需要右奇异矩阵V 的目标契合，可以比其它优化或者迭代 SVD 算法更有效地降低复杂度。

第三，我们提出了基于矩阵间关联性的逆矩阵计算优化，利用矩阵间关联性将高阶矩阵的求逆运算转换为分块矩阵之间的运算以及低阶矩阵的逆矩阵计算，能够大幅降低矩阵求逆运算的计算复杂度。

第四，我们采用了基于逐次超松弛迭代法的逆矩阵求解优化，可以通过迭代法求解线性矩阵方程组，以避免计算高阶一般矩阵的逆矩阵。对于缺乏矩阵间关联性的矩阵进行计算时，使用逐次超松弛（SOR）迭代法能够将计算高阶一般矩阵的逆矩阵转换为计算下三角矩阵的逆矩阵，从而简化求逆运算的复杂度。通过设置足够小的精度误差限制，可以使得由 SOR 迭代法得到的估计输出矩阵 $\hat{\boldsymbol{W}}_k$ 逼近精确解 $\boldsymbol{W}_k$。

最后，我们综合以上所提模型，建立了基于相关系数的矩阵组计算复杂度优化模型，综合给定数据的性质，寻找到最优的相关系数阈值，在保证有 99.9%的结果矩阵 $\boldsymbol{W}_k$ 达到建模精度 0.99 的条件下，使得运算复杂度最低。

3.4.1 系统模型

联合本节所述所有模型，我们建立了基于相关系数的矩阵组计算复杂度优化整体模型，以求解最佳的模型参数，使得所有运算流程的总计算复杂度最低。

由于计算 SVD 的右奇异矩阵、右奇异矩阵邻近替代及求解逆矩阵过程中均会出现误差，因此必须对整个计算流程进行联合优化。优化的目标是所有运算的总复杂度，而可优化的决策变量有以下几个：

1、在基于相关系数的矩阵间计算优化算法中，决定邻近矩阵是否可以相互替代的相关系数阈值 λ_{th}。显然，阈值 λ_{th} 越小，代表将有越多的子矩阵 $\boldsymbol{V}_{j,k}$ 被 $\boldsymbol{V}_{j,k-1}$ 替代，在导致运算量下降的同时将导致建模精度降低。

2、在基于随机正交基的 SVD 优化算法中，选取正交基时的迭代轮数 P。该迭代轮数实际上与从被分解矩阵行空间中选取的正交基个数相等，并进一步等于计算出的右奇异矩阵 $\boldsymbol{V}_{j,k}$ 的列数。同样地，迭代轮数越多，则导致运算量下降和建模精度降低。

3、在基于逐次超松弛迭代法的逆矩阵求解优化算法中，SOR 迭代法的迭代过程中的精度误差限制。SOR 迭代法通过不断迭代得到矩阵方程组的近似解，当精度误差限制足够小时，近似解可以无限逼近精确解，不过会导致更多的迭代轮次，导致更高的计算复杂度，一个合适的精度误差限制既可以保证计算精度，也可以保证迭代次数较少。

上述三个决策变量组成了整体优化模型的解，其所有取值组成了本优化问题的解空间。我们对最优解的求取作了如下简化。

首先考虑到在本问题中，进行 SVD 分解的子矩阵 $\boldsymbol{H}_{j,k}$ 是 $M \times N$ 维矩阵，且均为满秩矩阵。3.2.2 小节的论述已经表明，迭代轮数太少时会带来较大的估计误差，因此总是选取迭代轮数 $P = M$ 。

其次，SOR 迭代法中的精度误差限制同样常常选取经验值 10^{-4}，作为满足计算结果正确的最大精度误差限制，以保证迭代次数尽可能少，从而降低计算复杂度。

实际上，采用 RSVD 算法求解 SVD 和采用 SOR 算法求解逆矩阵时，均在各自的数学方法上对计算复杂度作了优化。而可优化的部分仅在于利用矩阵相关性的步骤，即矩阵间优化时的相关系数 λ_{th}。

由此，决策变量的数量减少到 1，而优化目标仍然是运算复杂度，约束条件为使所有 $\boldsymbol{W}_k$ 的估计精度达标的列比例大于 99.9%。即转化为以下优化问题：

$$\begin{aligned} &\min\ z = J(K\text{-}1)\times C_{judge} + \alpha \times JK \times C_{svd} + C_{inv}\Big|_{\lambda_{th}} \\ &s.t.\ \frac{t_{qua}}{LJK} \geq 0.999 \end{aligned} \tag{28}$$

其中 z 表示按照本模型进行计算时整个流程的运算复杂度，C_{judge}，C_{svd}，C_{inv} 分别表示分别表示作 1 次横向相邻矩阵相关系数计算、1 次 RSVD 计算、1 次逆矩阵计算的运算复杂度，α 表示所有 $\boldsymbol{H}_{j,k}$ 中需要进行 RSVD 操作的矩阵的比例，λ_{th} 表示判断相邻矩阵的右奇异矩阵能否替代时相邻矩阵的相关系数阈值，t_{qua} 表示所有结果矩阵的列向量中，达到预设估计精度的数量。

我们建立了基于相关系数的矩阵组计算复杂度优化求解 λ_{th} 的最优解，其计算流程如图 12 所示。

首先，根据经验设置 λ_{th} 的初始值。对矩阵组 $\boldsymbol{H} = \{\boldsymbol{H}_{j,k}\}$ 中的所有矩阵与其左侧求相关系数，当 k=1 时相关系数设为 0。所有 $j \in 1,2,...,J$， $k \in 1,2,...,K$ 的相关系数组成相关系数矩阵 $\boldsymbol{\Lambda}$。

对于当前矩阵 $\boldsymbol{H}_{j,k}$，如果 $\boldsymbol{H}_{j,k}$ 与左侧矩阵 $\boldsymbol{H}_{j,k-1}$ 的相关系数小于设定的阈值，即：

$$\boldsymbol{\Lambda}(j,k) < \lambda_{th} \tag{29}$$

则需对 $\boldsymbol{H}_{j,k}$ 进行 RSVD 运算。否则，就用 $\boldsymbol{H}_{j,k-1}$ 已计算出的奇异向量组 $\boldsymbol{V}_{j,k-1}$ 替代 $\boldsymbol{V}_{j,k}$，即：

$$\boldsymbol{V}_{j,k} \leftarrow \boldsymbol{V}_{j,k\text{-}1} \tag{30}$$

同一行块、不同列块的 $\boldsymbol{V}_{j,k}$ 全部计算完毕后，可对其横向排成 $\boldsymbol{V}_k$。如果当前 $\boldsymbol{V}_k$ 中存在 $\boldsymbol{V}_{j,k}$ 是由其左侧的 $\boldsymbol{V}_{j,k-1}$ 直接替换得来，则使用基于矩阵间关联性的逆矩阵计算方法计算 $\boldsymbol{W}_k$，否则使用 SOR 方法计算逆矩阵，以实现较小的计算复杂度。

最后，统计该 λ_{th} 阈值下的总运算复杂度和所有 $\boldsymbol{W}_k$ 达到规定精度的列的比例。

为简明性考虑，不能出现连续替代的原则没有在流程图中明显示出，即：在作右奇异向量的邻近替代时，如果某个 $\boldsymbol{V}_{j,k}$ 自身已经是被替代的矩阵，则其不能用于替代右侧矩阵的奇异向量，即不能有连续替代的情况发生。该原则已在 3.2.1 小节详述。

图 12 基于相关系数的矩阵组计算复杂度优化模型流程图

本模型的估计精度误差主要来源于三部分：

（1）使用 RSVD 计算右奇异向量过程的估计误差；

（2）使用 SOR 方法计算逆矩阵过程的估计误差；

（3）由于右奇异向量被邻近矩阵的右奇异向量替代，导致中间矩阵与实际值产生了偏差，由此带来最终结果的估计误差。

在建模求解过程中，实际上通过在不同的阈值 λ_{th} 下计算结果矩阵所有列的估计精度误差，再统计精度达标的列数，来实现对最佳阈值的求解。

3.4.2 结果与性能分析

针对给定的 6 组数据，绘制估计精度达标率 α 和运算复杂度下降率 ΔC 随相关系数阈值 λ_{th} 的变化图像如图所示。

可以看到，随着相关系数阈值的增大，能够被邻近矩阵右奇异向量替代的 $\boldsymbol{V}_{j,k}$ 数量变少。这在导致估计精度达标率提升的同时，也将带来更多次数的 RSVD 和 SOR 求逆运算，从而导致最终的计算复杂度上升，因此运算复杂度的下降率会降低。

在估计精度达标率 $\alpha > 99.9\%$ 的情况下，6 组数据的最佳阈值 $\tilde{\lambda}_{th}$ 及其对应的运算复杂度下降比例如表 5 所示。

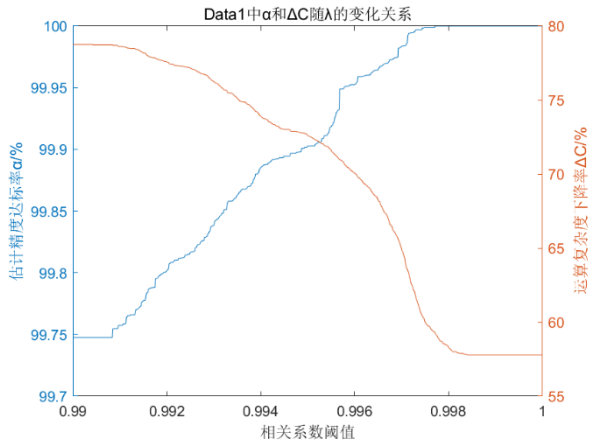

图 13 Data1 精度达标率和复杂度下降率

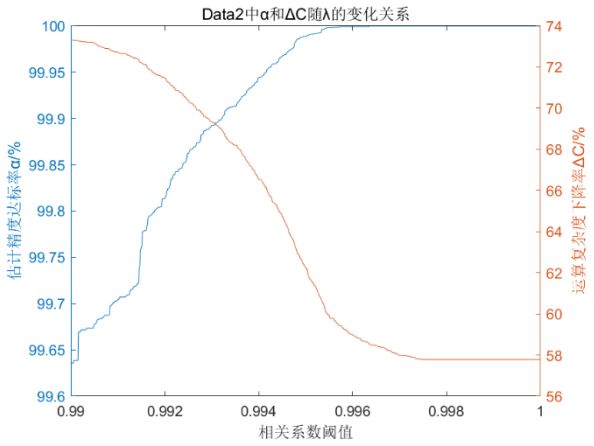

图 14 Data2 精度达标率和复杂度下降率

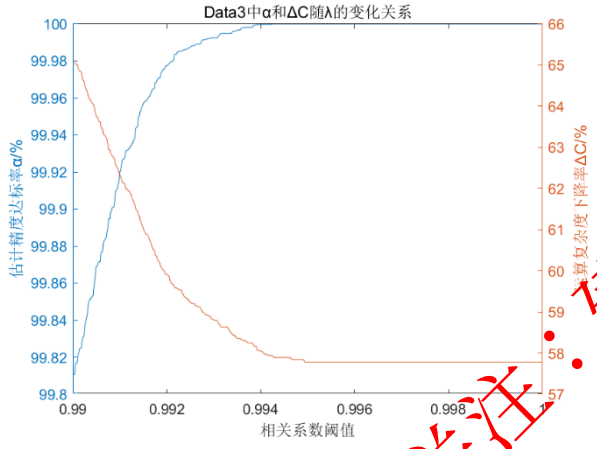

图 15 Data3 精度达标率和复杂度下降率

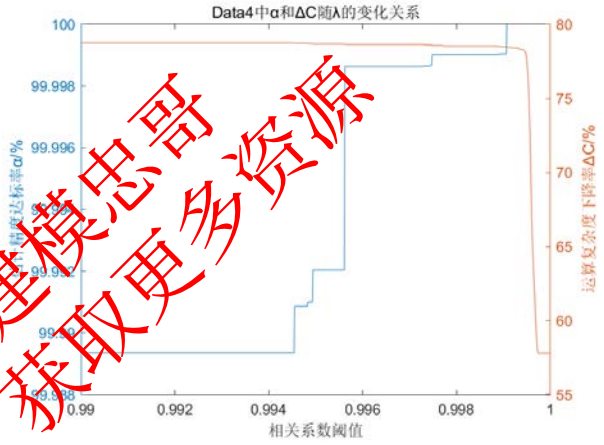

图 16 Data4 精度达标率和复杂度下降率

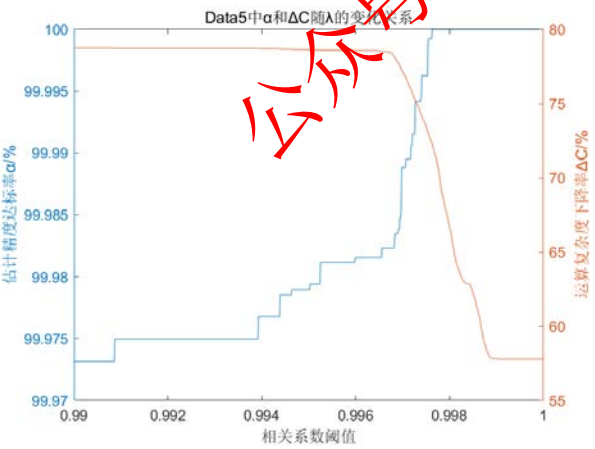

图 17 Data5 精度达标率和复杂度下降率

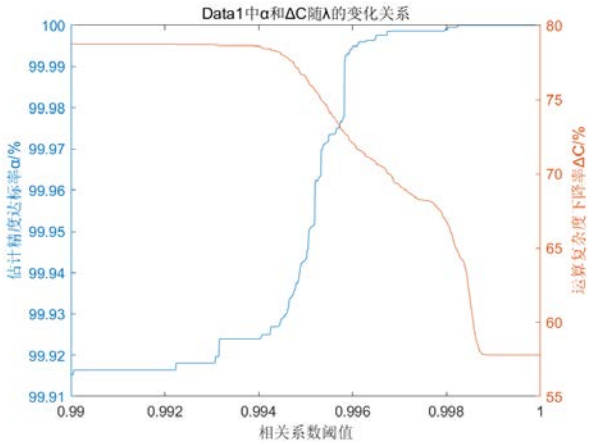

图 18 Data6 精度达标率和复杂度下降率

表 5 运算复杂度下降比例的汇总

数据集序号	1	2	3	4	5	6
最佳阈值 $\tilde{\lambda}_{th}$	0.99487	0.99321	0.99084	0.99	0.99	0.99
运算复杂度下降率/%	72.77%	68.95%	62.78%	78.75%	78.75%	78.75%

4. 问题二的模型建立与求解

4.1 问题分析

问题二的本质是相关矩阵组压缩优化问题，需要提出的模型在 $err_{\max} = -30dB$ 的目标下，尽可能利用矩阵内或矩阵间的关联性，以较低的运算复杂度实现较高的数据压缩率。

矩阵压缩的本质通过去除矩阵中的冗余信息，从而降低数据所需的存储量，方便数据的保存与传输。关于矩阵组压缩问题，我们作以下讨论：

目前主流的模型压缩方法有：

① 基于特征值分解的压缩算法
② 基于奇异值分解的压缩算法
③ 基于高斯随机正交基的压缩感知算法
④ 基于其它稀疏基的压缩感知算法
⑤ 基于帧间预测的视频压缩算法
⑥ 基于帧内特征的视频压缩算法
⑦ 基于编码方式的码率压缩算法（**题目禁用**）

其中，方法①和②的主要目标是提取矩阵最重要的特征，常用于计算机视觉和图像压缩、通常具有较高的计算复杂度和开销，但是通过关键特征的提取，可以有效地降低存储内容，实现较高的压缩率。方法③和④是通过压缩感知的方法，通过建立对已知矩阵具有稀疏性的基，找到最佳的数个投影值，通过存储投影值和索引值，来提高压缩率，通常用于信号处理和数据传输。方法⑤和⑥则是常用的视频压缩算法，利用的是视频帧率之间的连续性来降低数据存储量，当然对使用场景提出了较高的要求。对上述方法进行下图总结，

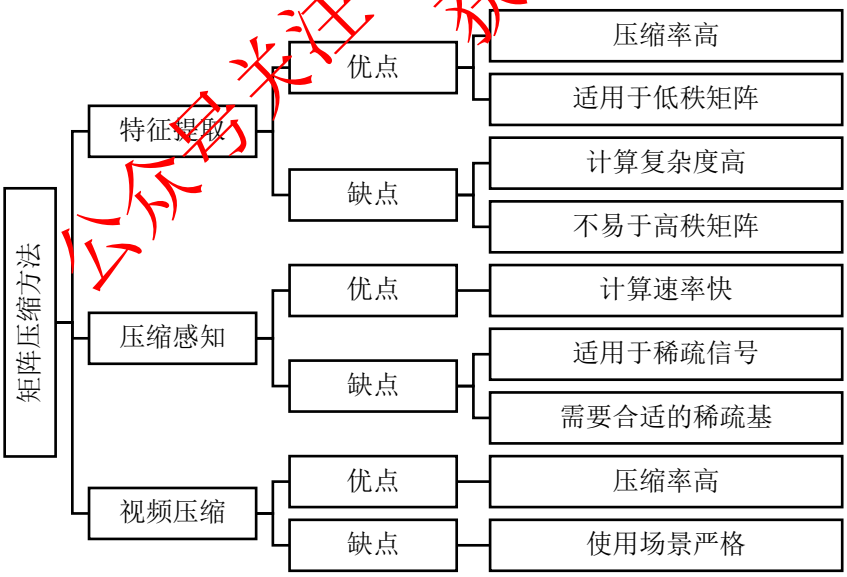

图 19 常用矩阵压缩算法汇总

最终，我们提出基于 SVD 分解的 $\boldsymbol{H}$ 矩阵和 $\boldsymbol{W}$ 矩阵压缩算法，以及基于空间正交基的 $\boldsymbol{H}$ 矩阵压缩算法。对于 $\boldsymbol{H}$ 矩阵压缩中，我们推荐根据实际场景选择使用算法，在对压缩率要求高的情况下，可以优先选择 SVD 分解，在对计算复杂度要求高的情况下，建议优先选择基于空间正交基。下一节中对两种算法进行具体的分析。

4.2 模型建立

4.2.1 高压缩率压缩算法——基于奇异值分解的矩阵压缩

在 3.1.2 节中，我们具体讨论了已知输入矩阵 $\boldsymbol{H}$ 的相关性，其中重点发现了输入矩阵 $\boldsymbol{H}$ 的 $N\times K$ 矩阵间的具有较强的相关性。因此可以通过对高维矩阵进行奇异值分解，从而大幅度降低矩阵存储开销。

在压缩过程中，我们首先将 K 个 $M\times N$ 的矩阵进行拆分重组，按行连接得到一个 $1\times MN$ 的行向量，然后对 K 个行向量按行合并，得到 $K\times MN$ 的大矩阵，对大矩阵进行奇异值分解，得到分解后的左奇异矩阵 $\boldsymbol{U}$ 、奇异值矩阵 $\boldsymbol{S}$ 和右奇异矩阵 $\boldsymbol{V}$ 。设置提取秩为 R，从而只提取 $\boldsymbol{U}$ 的 R 行， $\boldsymbol{S}$ 的 R 个以及 $\boldsymbol{V}$ 的 R 列，保存。最后再通过逆过程

$$\hat{\boldsymbol{A}}=\boldsymbol{U}_r\boldsymbol{S}_r\boldsymbol{V}_r^H$$

拆分重组 $\hat{\boldsymbol{A}}$ 得到原始的矩阵 $\hat{\boldsymbol{H}}$ 。具体的流程图如下：

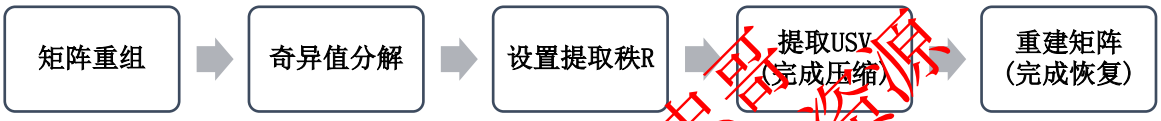

图 20 基于奇异值分解的矩阵压缩流程图

基于奇异值分解的矩阵压缩算法原理简单，压缩率高，在提供的数据测试中，在-30dB 的约束条件下，依然可以实现超过 50%的压缩率，并且恢复复杂度低，但是计算复杂度高。

计算复杂度具体参考第 3.3.3 小节 RSVD 的计算，在 $\boldsymbol{W}$ 矩阵的压缩过程中，我们也主要采用了本算法，不再赘述。

4.2.2 低复杂度压缩算法——基于空间正交基的矩阵压缩

在 3.1.2 节中，我们具体讨论了已知输入矩阵 $\boldsymbol{H}$ 的稀疏性，其中重点发现了输入矩阵 $\boldsymbol{H}$ 的主要能量分布具有较强的稀疏性，同时还发现了不同 k 矩阵里的同一个行向量存在相同的基索引较多。通过这两个要点，我们提出了基于空间正交基的矩阵压缩算法，其中算法的具体公式已经在 3.1.2 节中给出，本节主要讨论算法的流程如下：

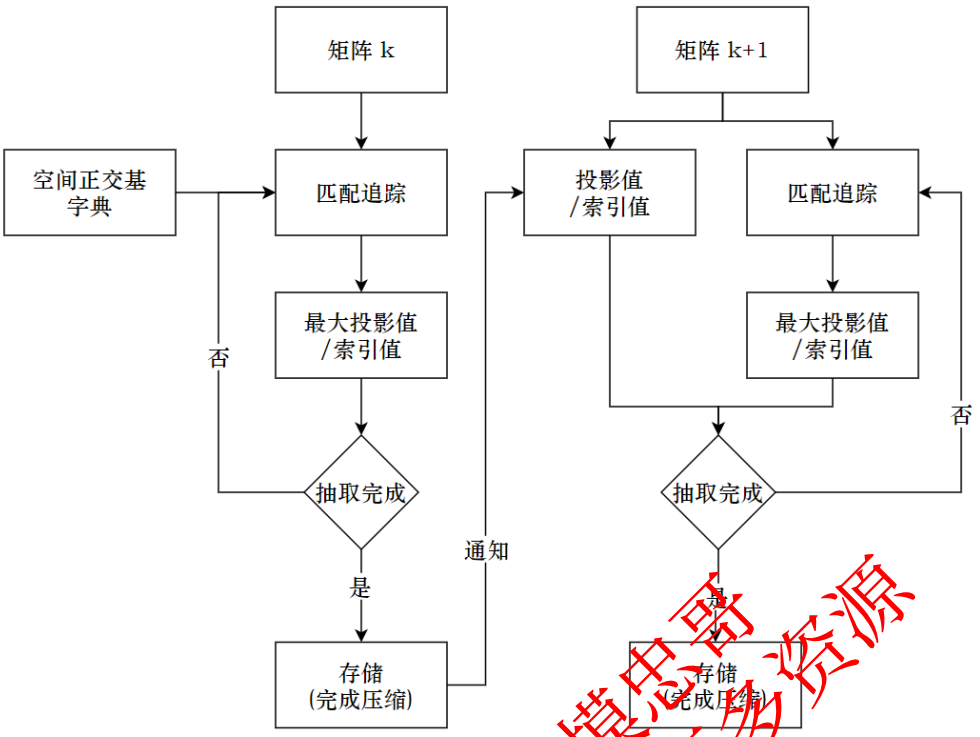

图 21 基于空间正交基的矩阵压缩流程图

在图 21 中提供了基于空间正交基的矩阵压缩的基本思路：

第一步，首先需要建立空间正交基，即字典，也称作码本。基于公式(7)、(8)，我们可以建立字典 $\boldsymbol{D}$。通过设置过采样率 β 的大小来控制整个字典的尺寸为 $N\times N\beta$。需要说明的是，因为字典可以预先存储在系统里，只需要生成一次，适用于六个数据集里 $J\times K$ 个矩阵全部计算，因此相比之下可以不考虑字典的存储复杂度和计算复杂度。

第二步，对于第 k 个矩阵，我们可以将其 $1\times N$ 的行向量与 $N\times N\beta$ 维的字典作复数乘法，得到 $1\times N\beta$ 维度的投影值向量 G，求取 G 每个元素的绝对值大小，排序得到前 20 个最大元素的索引值 i，保存 i 和投影值 $G(i)$。需要说明的是索引值 i 是一个小于 $N\beta$（计算中为 256）的正整数，因此不考虑索引值 i 的存储复杂度。

第三步，每次抽取一个投影值后，通过 3.1.2 节的公式(6)去除该投影值的能量值，如果出现重复的投影值索引，则进行迭代计算，再回到第二步，重复计算直到索引总数达到预设值 $I_{\max}$，平均每个矩阵 k 需要迭代 2 次，随着最大索引值的增加，迭代次数也会明显增加，这主要是因为剩下的能量稀疏度较低，需要重复投影。

第四步，保存投影值 $G(i)_{i=1,2,\ldots,I}$，并将投影值通知给第 $k+1$ 个矩阵，此时第 $k+1$ 个可以直接通过对上一个矩阵的索引值直接投影，可以以更高效率得到目标结果，根据观察，利用率可以达到 90%左右。其本质上是利用了邻近矩阵的索引值相同的特性，其本质上是矩阵组之间的重要关联性利用的体现。

计算复杂度：对于第 m 行的第 1 个矩阵，其复杂度包括

- 一次复数矩阵乘法 $N\beta\times N$ 与 $N\times 1$，复杂度为 $(16N-2)N\beta$
- 一次复数向量元素绝对值 $N\beta\times 1$，复杂度为 $8N\beta+23$
- 一次复数矩阵乘法 1×1 与 $1\times N$，复杂度为 $14N$
- 一次复数矩阵减法 $1\times N$，复杂度为 $2N$

因此第 1 个矩阵的复杂度计算公式为$((16N-2)N\beta+8N\beta+23+16N)\cdot M$。对于第$m$行的第$k$个矩阵，对于上一个矩阵索引的平均利用率可以达到 90%左右，其复杂度计算公式为$((16N-2)N\beta+8N\beta+23+16N)\cdot 0.1\cdot 3\cdot M$

因此$M\times N\times K$个矩阵的总共复杂度为

$((16N-2)N\beta+8N\beta+23+16N)\cdot(1+0.1(K-1))\cdot 3\cdot M \cong 4.8N^2\beta KM$

当$\beta=4$时，总复杂度为 120795955，约为 1.2 亿次计算，由此可见基于空间正交基的矩阵压缩算法在计算复杂度方面要明显低于基于 SVD 的算法，这是因为该算法不需要进行大型矩阵的特征分解，因此对运算要求较低。

存储复杂度：由于基于空间正交基的矩阵压缩算法对矩阵的稀疏性要求高，同时因为计算过程简单，因此实际测试中压缩性能只有$-15\sim-20dB$左右，且不会随着压缩效果要差于 SVD 分解。

4.3 性能分析

4.3.1 基于奇异值分解的 H 矩阵压缩恢复性能

表 6 基于奇异值分解的 *H* 矩阵压缩恢复性能汇总表

数据集序号	达到-30dB 误差的最小截断行数 *r*	压缩恢复误差	压缩率/%	运算复杂度
1	31	-30.019	20.20	2,559,025,438
2	44	-30.224	29.32	5,159,925,117
3	45	-30.090	29.32	5,159,925,117
4	32	-30.094	20.85	2,714,100,671
5	45	-30.093	29.32	5,159,925,117
6	44	-30.111	28.67	4,943,442,395

从测试结果中可以看出，基于奇异值分解的 ***H*** 矩阵压缩可以在较高性能的条件下，实现较高的矩阵压缩率，其中压缩率是指压缩后数据占用空间与压缩前数据占用空间的比值。

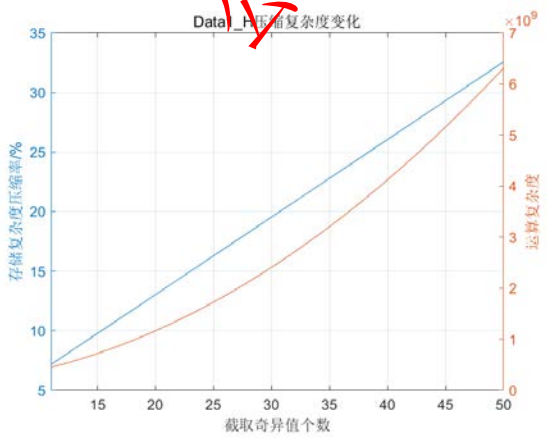

图 22 DATA1 的 *H* 矩阵压缩率与运算复杂度

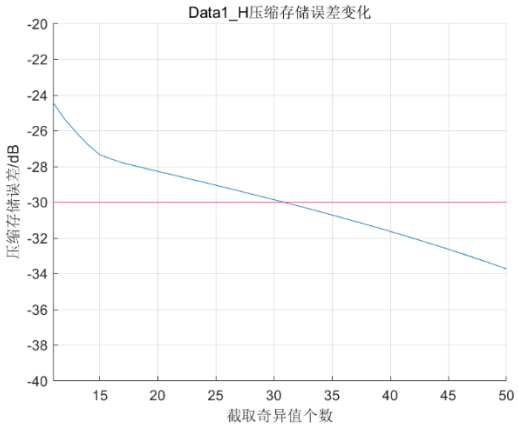

图 23 DATA1 的 *H* 矩阵压缩误差率

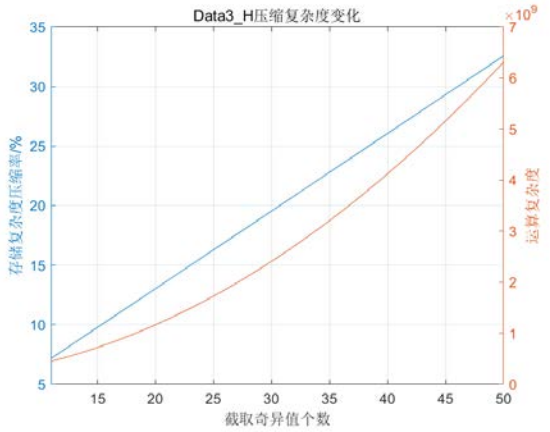

图 24 DATA3 的 H 矩阵压缩率与运算复杂度

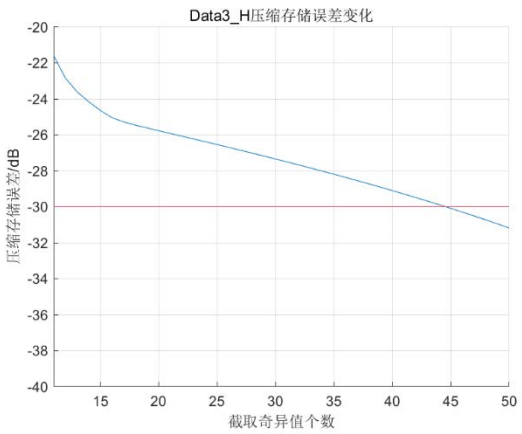

图 25 DATA3 的 H 矩阵压缩误差率

4.3.2 基于奇异值分解的 W 矩阵压缩

表 7 基于奇异值分解的 W 矩阵压缩恢复性能汇总表

数据集序号	达到-30dB 误差的最小截断行数 r	压缩恢复误差	压缩率/%	运算复杂度
1	72	-30.045	75.07	12,547,942,551
2	85	-30.296	88.63	17,423,687,009
3	91	-30.191	94.88	19,943,603,612
4	38	-30.168	39.62	3,574,281,139
5	57	-30.090	59.43	7,915,168,731
6	61	-30.213	63.60	9,046,543,397

从测试结果中可以看出，基于奇异值分解的 W 矩阵压缩的性能略差于 H 矩阵压缩，说明了 W 矩阵之间的相关性要弱于 H 矩阵。除去数据集 3 外，在大部分情况下，SVD 分解还是可以很好地完成 W 矩阵的压缩。

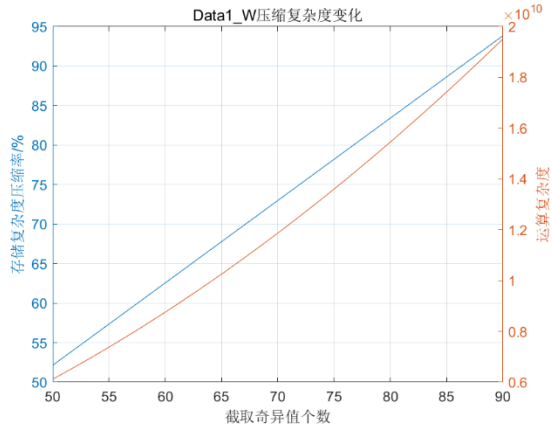

图 26 DATA1 的 W 矩阵压缩率与运算复杂度

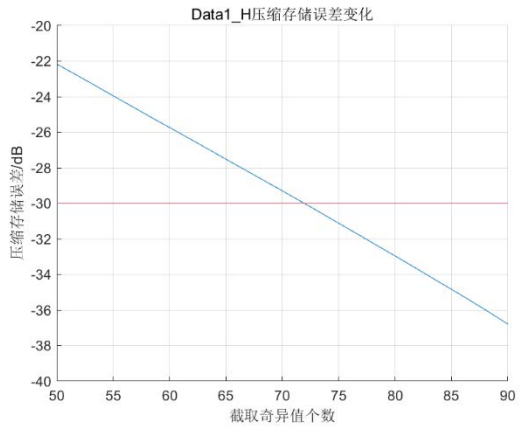

图 27 DATA1 的 W 矩阵压缩误差率

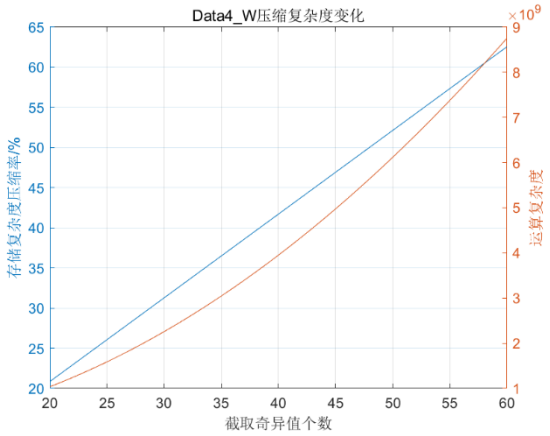

图 28 DATA4 的 W 矩阵压缩率与运算复杂度

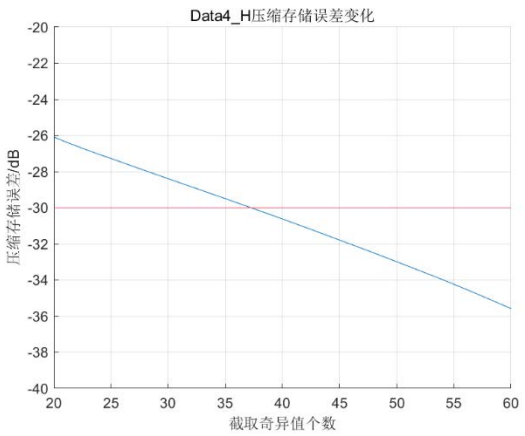

图 29 DATA4 的 W 矩阵压缩误差率

4.3.3 基于空间正交基的 H 矩阵压缩

表 8 基于奇异值分解的 H 矩阵压缩恢复性能汇总表

数据集序号	达到-20dB 误差的最小索引总数	压缩恢复误差	压缩率/%	运算复杂度
1	49	-20.341	76.56%	201,838,454
2	49	-20.204	76.56%	201,838,454
3	48	-20.078	75.00%	197,719,302
4	44	-20.138	68.75%	181,242,693
5	44	-20.159	68.75%	181,242,693
6	43	-20.119	67.19%	177,123,541

从测试结果中可以看出，基于空间正交基的 H 矩阵压缩的压缩误差只能到-20dB，压缩率也只有 70 左右，但是预算复杂度相比前两项测试要下降至少一个数量级，说明基于空间正交基的 H 矩阵压缩更适合用于低复杂度条件下的运算，符合预期设想。

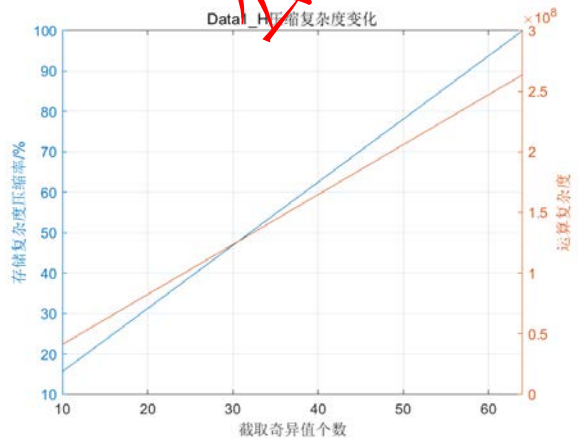

图 30 DATA1 的 H 矩阵压缩率与运算复杂度

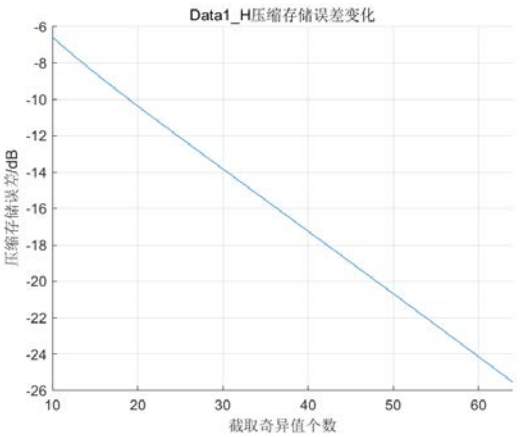

图 31 DATA1 的 H 矩阵压缩误差率

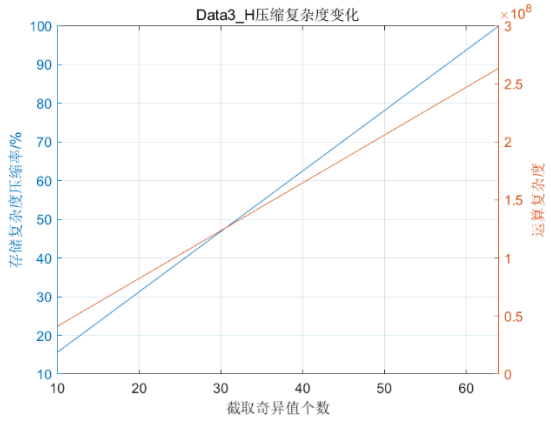

图 32 DATA3 的 H 矩阵压缩率与运算复杂度

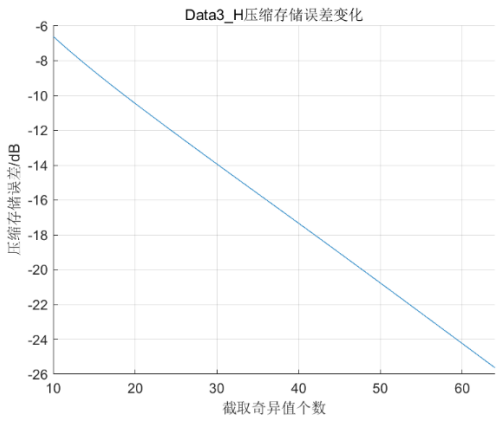

图 33 DATA3 的 H 矩阵压缩误差率

4.3.4 性能比较与分析

表 9 性能汇总

	适用压缩误差	平均压缩率	平均计算复杂度
H-SVD	-30dB	25%	42 亿次单位计算
H-CS	-20dB	70%	2 亿次单位计算
W-SVD	-30dB	70%	117 亿次单位计算

*SOB 是空间正交基（space orthogonal basis）的英文缩写

对三种算法进行对比，可以明显得到，基于奇异值分解的算法适合压缩率高、运算条件充裕的情况，基于空间正交基的算法适合压缩要求低、运算条件差的情况。我们提出了两种算法，可以灵活替换。

5. 问题三的模型建立与求解

5.1.1 模型建立

针对问题三，主要目标工作是设计低复杂度计算和存储的整体方案，完成端到端的输入和输出流程。综合我们在问题一中提出的基于矩阵间相关性的 SVD 分解算法和逆矩阵求解算法，以及问题二中的基于奇异值分解的压缩算法和基于空间正交基的压缩算法。我们可以对以下工作进行合并，可以在保持低存储度和高精确度的条件下实现低复杂度计算：

① 压缩算法的空间正交基，提取出的索引值，可以辅助于问题一中关于是否进行 SVD 分解代替的判断，从而提高估计精度；

② 问题一中关于输入矩阵 $\boldsymbol{H}$ 的 SVD 分解计算，可以加入到输入矩阵 $\boldsymbol{H}$ 的 SVD 分解存储的过程中，从而降低运算复杂度；

③ 问题一中关于矩阵 $\boldsymbol{W}_k = \boldsymbol{V}_k \left(\boldsymbol{V}_k^H \boldsymbol{V}_k + \sigma^2 \boldsymbol{I}\right)^{-1}$，可以通过存储 $\boldsymbol{V}_k$ 和 $\left(\boldsymbol{V}_k^H \boldsymbol{V}_k + \sigma^2 \boldsymbol{I}\right)^{-1}$ 及其关键特征，可以帮助实现 $\boldsymbol{W}_k$ 的存储工作，降低复杂度。

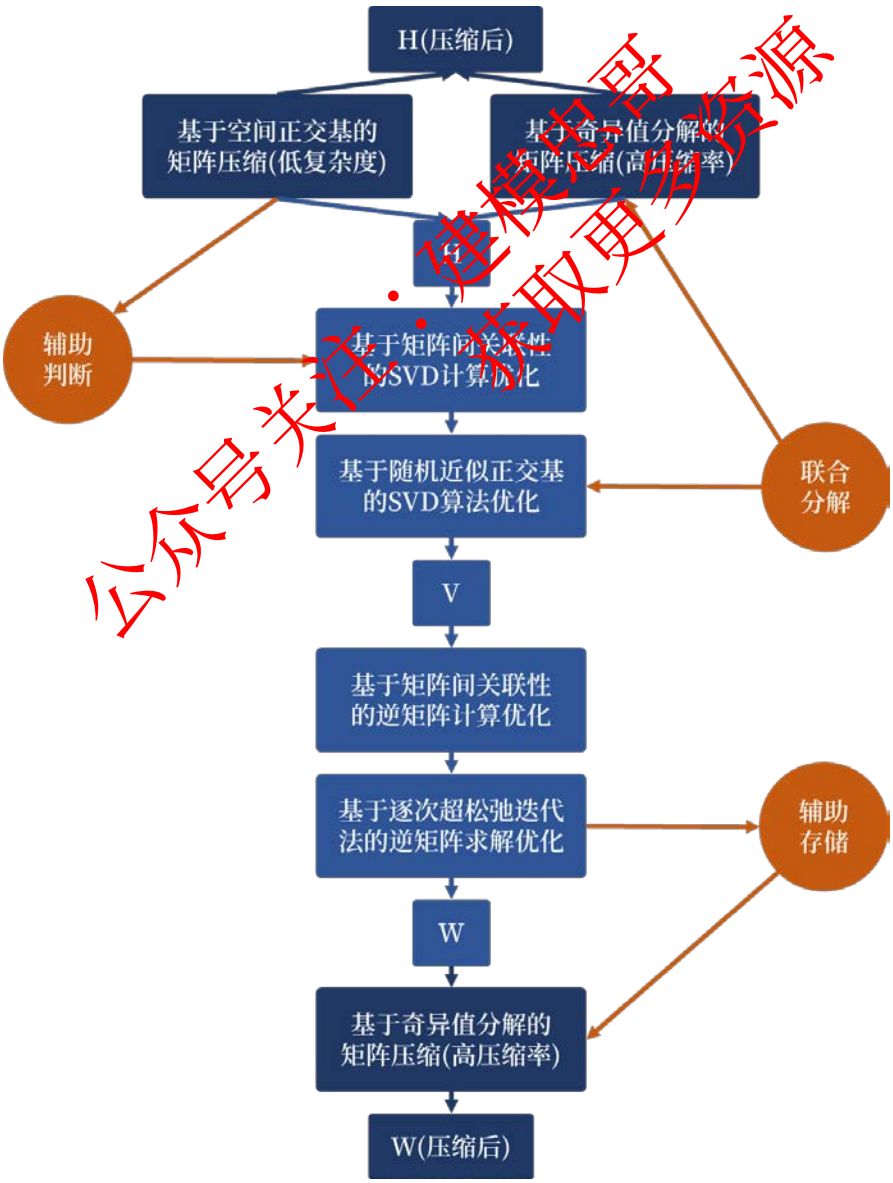

图 34 相关矩阵组的低复杂度计算和存储整体流程图

5.1.2 系统测试

我们在 MATLAB 程序中构建了完整的程序，并测试了性能，代码参考附录。

表 10 性能汇总

	DATA1	DATA3	DATA6
输入：复杂度阈值	0.995	0.995	0.995
输入：SVD-H 秩	40	40	40
输入：SVD-W 秩	70	70	70
以下为输出结果			
H-SVD 压缩误差率(dB)	**-31.62**	**-29.22**	**-29.32**
H-SVD 压缩压缩率(%)*	26.06	26.06	26.06
H-SVD 压缩复杂度	4124749687	4124749687	4124749687
SVD 省略比例(%)	**36.00**	**13.00**	**46.00**
相关系数计算的复杂度	13369344	13369344	13369344
SVD 分解的常规算法的复杂度	3412726272	3412726272	3412726272
SVD 分解的优化算法的复杂度	1345459647	1882115264	1138256135
SVD 分解的算法复杂度下降率(%)	**60.58**	**38.73**	**66.65**
矩阵求逆的常规算法的复杂度	1580926848	1580926848	1580926848
矩阵求逆的优化算法的复杂度	226144712	278064279	206817671
矩阵求逆的算法复杂度下降率(%)	**85.70**	**82.41**	**86.92**
压缩前最小精度达标率	99.98	99.97	99.93
W-SVD 压缩误差率(dB)	**-29.29**	**-29.31**	**-33.17**
W-SVD 压缩压缩率(%)	72.98	72.98	72.98
W-SVD 压缩复杂度	11868764819	11868764819	11868764819
压缩后最小精度达标率(%)	**99.88**	**99.96**	**99.93**
计算部分的常规算法总复杂度	4993653120	4993653120	4993653120
计算部分的优化算法总复杂度	1584973703	2123548887	1152642160
计算部分的总复杂度下降率(%)	**68.26**	**57.48**	**72.80**
全部系统的总复杂度	17578488209	18117063393	17351957656

*压缩率是指压缩后数据占用空间与压缩前数据占用空间的比值

以上是我们随机抽取三个数据集完成的试验输出结果，测试结果证明了我们的整套模型充分利用了矩阵间的相关性，有效降低了系统的复杂度，有较高的压缩率和恢复率，鲁棒性强，基本实现了题目的各项要求。

6. 模型评价与展望

6.1 创新点

1. 设计了基于相关系数的矩阵间关联性计算模型。
2. 提出了基于矩阵关联性的逆矩阵计算优化。
3. 在压缩与解压缩时引入了基于空间正交基的压缩感知算法。

6.2 优点

1. 利用矩阵间关联性，对 SVD 计算次数进行优化，降低了计算复杂度。
2. 采用了逐次超松弛迭代法求解矩阵方程组，避免计算高阶一般矩阵的逆矩阵。
3. 对约束进行松弛，通过达标率表征模型精度，使得模型更加简洁有效。
4. 提出了两种压缩与解压缩方法，包括低秩分解与压缩感知算法。

6.3 缺点

1. 对数据进行压缩时，$\boldsymbol{H}$ 矩阵规模较大且具有低秩性，RSVD 算法复杂度较低，而 $\boldsymbol{W}$ 规模较小，模型计算复杂度降低不显著。
2. 对于 $\boldsymbol{W}_k$，难以构建合适的正交基通过压缩感知算法进行压缩与解压缩，实际应用中计算复杂度较高。

6.4 展望

模型可以适应无线通信、计算机视觉等许多领域中的相关性矩阵。

以无线通信领域中，**基于 SVD 分解的 MIMO 系统预编码和 MMSE 均衡器**为例，其中 $\boldsymbol{H}_k$ 为信道，$\boldsymbol{V}_k$ 为信道 SVD 分解的奇异向量，$\boldsymbol{W}_k$ 为信道 MMSE 均衡矩阵。$M \times N$ 代表用户端有 M 根天线，基站端有 N 根天线；J 表示子载波数，频率之间信道不相关；K 表示时间序列数，连续时间内信道相关。

根据空间正交基，矩阵中每个行向量可以分解为多个路径，并且存在噪声信号，因此，我们的模型可以利用矩阵相关性和稀疏性，从而降低计算复杂度和存储复杂度。

参考文献

[1] Sun C, Gao X, Jin S, et al. beam division multiple access transmission for massive MIMO communications[J]. IEEE Transactions on Commumications, 2015, 63(6):2170--2184.

[2] Han Y, Hsu T H, Wen C K, et al. efficient downlink channel reconstruction for FDD multi-antenna systems[J]. IEEE Transactions on Wireless Communications, 2019, 18(6):3161--3176.

[3] Hoydis J, Brink S, Debbah M. massive MIMO in the UL/DL of cellular networks: how many antennas do we need?[J]. IEEE Joumal on Selected Areas in Communications, 2013, 31(2):160--171.

[4] Aharon M, Elad M, Bruckstein A. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation[J]. IEEE Transactions on signal processing, 2006, 54(11): 4311-4322.

[5] Knockaert L, De Backer B, De Zutter D. SVD compression, unitary transforms, and computational complexity[J]. IEEE transactions on signal processing, 1999, 47(10): 2724-2729.

[6] Halko N, Martinsson P G, Tropp J A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions[J]. SIAM review, 2011, 53(2): 217-288.

[7] Golub G, Kahan W. Calculating the singular values and pseudo-inverse of a matrix[J]. Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, 1965, 2(2): 205-224.

[8] Swartzlander E E, Saleh H H. Floating-point implementation of complex multiplication[C]//2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers. IEEE, 2009: 926-929.

[9] 陈卓. GSVD 的分布特性及其在 MIMO 预编码中的应用研究[D].中国科学技术大学,2019.

[10] 周乔,许魁,徐友云,谢天怡.联合波束域分解和 SVD 的多用户大规模 MIMO 系统信道估计[J].信号处理,2018,34(04):439-447.

[11] 杨凤. 复对称线性方程组的迭代法及预处理研究[D].温州大学,2019.

[12] 胡枫,金远平.SOR 最优松弛因子选取方法研究[J].西南师范大学学报(自然科学版),2008(05):48-50.

[13] 李兴旺. 无线通信系统中的大规模 MIMO 关键理论及技术研究[D].北京邮电大学,2015.

[14] 徐勇俊. 基于信号稀疏表示的字典设计[D].南京理工大学,2013.

[15] 张成,汪东,沈川,程鸿,陈岚,韦穗.基于奇异值分解的可分离压缩成像方法[J].计算机研究与发展,2016,53(12):2816-2823.

[16] 张涛. 视频压缩中的高效帧内编码技术研究[D].哈尔滨工业大学,2017.

[17] 练秋生,王小娜,石保顺,陈书贞.基于多重解析字典学习和观测矩阵优化的压缩感知[J].计算机学报,2015,38(06):1162-1171.

[18] 陈栩杉,张雄伟,杨吉斌,孙蒙.如何解决基不匹配问题:从原子范数到无网格压缩感知[J].自动化学报,2016,42(03):335-346.

附录

第一问 MATLAB 代码

```
Main_Model
clc
clf
clear

%% 备注
% 本文件为问题一的最终目标main文件
% 调用

%% 初始化参数
H_D = load('Data1_H');
W_D = load('Data1_W');
H_DATA1 = H_D.H;      %N M J K
W_DATA1 = W_D.W;     %M L J K
D_Size = size(H_DATA1);
W_Est = zeros(size(W_DATA1));
L = size(W_DATA1,2);
M = size(H_DATA1,1);
N = size(H_DATA1,2);
J = size(H_DATA1,3);
K = size(H_DATA1,4);
V=zeros(N,N,J,K);
V_est=zeros(N,N,J,K);
U=zeros(M,M,J,K);
A=zeros(M,N,J,K);
Q=zeros(M,M,J,K);
R=zeros(M,N,J,K);
V_L = zeros(N,L*J,K);    % 维度为N×LJ×K
W = zeros(N,L*J,K);    % 维度为N×LJ×K
W2 = zeros(L*J,L*J,K);    % 维度为N×LJ×K
H_COR= zeros(J,K-1);    % 维度为N×LJ×K
V_COR= zeros(J,K);    % 维度为N×LJ×K

%% 读取DATA数据集合
Set_num = 6;        %设置读取的数据集序号
H_D = load(['Data',num2str(Set_num),'_H']);
W_D = load(['Data',num2str(Set_num),'_W']);
H_DATA1 = H_D.H;      %N M J K
W_DATA1 = W_D.W;   %N L J K

%% 模块一(计算部分) 基于欧几里得距离的矩阵间优化
AccuReq = 0.9;
RouMin = 0.99;
```

```
RouInterval = 1e-4;
RouMax = 1;
% O_count = zeros(1,J);      %用于记录被优化的k数
% O_Mtx = zeros(J,K);          % 用于记录是否需要省略svd计算
AccurateRate = zeros(1, round((RouMax-RouMin)/RouInterval));  % 精度达标率（随阈值变化）
ComplexAll = zeros(1, round((RouMax-RouMin)/RouInterval));            % 总运算复杂度（随阈值变化）
% eta = 3;                                % 可优化的参量，用于判断是否省略svd计算
iterCount = 0;
for Rou_th = RouMin:RouInterval:RouMax      % Rou_th 是相关系数的阈值
    if (Rou_th == 0.996)
        a=1;
    end
    iterCount = iterCount + 1;
    if(mod(iterCount,10) == 0)
        fprintf("%d\n",Rou_th);
    end
    O_count = zeros(1,J);      %用于记录被优化的k数
    O_Mtx = zeros(J,K);          % 用于记录是否需要省略svd计算
    for j = 1:1:J
        for k = 1:1:K
            if mod(k,2) == 1
                O_Mtx(j,k) = 0;       %需要计算svd
            else
%                 H_delta = H_DATA1(:,:,j,k) - H_DATA1(:,:,j,k-1);
%                 O_1 = sum(abs(H_delta),'all');                    % 计算信道的欧几里得距离
%                 O_2 = sum(abs(H_DATA1(:,:,j,k-1)),'all');        % 计算信道的能量值
                A1 = reshape(H_DATA1(:,:,j,k), M*N, 1);
                A2 = reshape(H_DATA1(:,:,j,k-1), M*N, 1);
                O_1 = norm(A1'*A2) / (norm(A1) * norm(A2));
                if (O_1 > Rou_th)          % 相关系数大于预设值，视作可替代
                    O_Mtx(j,k) = 1;
                    O_count(j) = O_count(j)+1;
                else
                    O_Mtx(j,k) = 0;
                end
            end
        end
    end

%% 模块一(复杂度部分) 基于欧几里得距离的矩阵间优化
O_Count_ALL = sum(O_count);
% fprintf('模块一：省略SVD的比例为%d/%d=%.2f\n',O_Count_ALL,J*K,O_Count_ALL/(J*K));
% 本环节的主要复杂度有
% 计算J*K/2次 维度为M*N - M*N 的复矩阵减法        复杂度为 J*K/2 *M*N *2
% 计算J*K*2/2次 M*N 个复数值求绝对值                 复杂度为 J*K*2/2 *M*N *32
% 计算J*K*2/2次 M*N 个实数值求和                     复杂度为 J*K*2/2 *M*N *1
```

```
% 模块一的计算量可以表示为：
C_1 = J*K/2 *M*N *2 + J*K*2/2 *M*N *32 +J*K*2/2 *M*N *1;
% fprintf('模块一：基于欧几里得距离的矩阵间优化的复杂度为C_1 = %d\n',C_1);


%% 模块二(SVD计算部分) 基于随机性的SVD分解
for j = 1:1:J
    for k = 1:1:K
        if O_Mtx(j,k) == 0
            %                 [U(:,:,j,k),A(:,:,j,k),V2(:,:,j,k)] = svd(H_DATA1(:,:,j,k));
            V_est(:,1:4,j,k) = rand_svd(H_DATA1(:,:,j,k));
%                 [~,~, X] = svd(H_DATA1(:,:,j,k));
%                 V_est(:,1:4,j,k) = X(:,1:4);
        else
            V_est(:,:,j,k) = V_est(:,:,j,k-1);
        end
        V_L(:,1+(j-1)*L:j*L,k) = V_est(:,1:L,j,k);
    end
end
%% 模块二(复杂度部分) 基于随机性的SVD分解
% 小块的Hjk（4×64矩阵）
% 使用随机SVD的复杂度为：1363181
% 使用Colub-Kahan（常规算法）的复杂度为：：2221827
% 优化前需要调用 J*K次
% 优化后需要调用 J*K - O_Count_ALL次
C_2_NEW = 1363181 *(J*K - O_Count_ALL);
C_2_CNV = 2221827 *J*K;                                    %conventional
% fprintf('模块二：SVD分解的常规算法的复杂度为C_2_CNV = %d\n',C_2_CNV);
% fprintf('模块二：SVD分解的优化算法的复杂度为C_2_NEW = %d\n',C_2_NEW);
% fprintf('模块二：SVD分解的优化算法的复杂度优化率为ΔC_2 = %.2f%%\n',(-
C_2_CNV+C_2_NEW)/C_2_CNV*100);

%% 模块三(W计算部分) 基于SOR迭代的优化逆矩阵算法
numCal = 0;
% [W,numCal] = Optimized_Inverse_Correlation(V_L,O_Mtx);
 for k=1:1:K
 W(:,:,k) =   V_L(:,:,k)*inv(V_L(:,:,k)' * V_L(:,:,k) + 0.01* eye(L*J));    %常规计算
    for j = 1:1:J
        for l = 1:1:L
            W_Est(:,l,j,k) = W(:,l+(j-1)*L,k);
        end
    end
 end

%% 模块三(复杂度部分) 基于SOR迭代的优化逆矩阵算法
%对于每一个k，计算复杂度随j被替换次数变化；
%0个被替换，传统方法直接矩阵求逆的复杂度为4116997，使用SOR迭代法的复杂度为1016681；
```

```
%1个被替换，根据相关性优化的逆矩阵算法的复杂度为146666;
%2个被替换，根据相关性优化的逆矩阵算法的复杂度为111082;
%3个被替换，根据相关性优化的逆矩阵算法的复杂度为168226;
%4个被替换，即W(k)完全=W(k-1)，复杂度为0
replaceTimes_0 = sum(numCal(:) == 0);
replaceTimes_1 = sum(numCal(:) == 1);
replaceTimes_2 = sum(numCal(:) == 2);
replaceTimes_3 = sum(numCal(:) == 3);
C_3_NEW =
1016681*replaceTimes_0+146666*replaceTimes_1+111082*replaceTimes_2+168226*replace
Times_3;
C_3_CNV = 4116997*K;
% fprintf('模块三：矩阵求逆的常规算法的复杂度为C_3_CNV = %d\n',C_3_CNV);
% fprintf('模块三：矩阵求逆的优化算法的复杂度为C_3_NEW = %d\n',C_3_NEW);
% fprintf('模块三：矩阵求逆的优化算法的复杂度优化率为ΔC_3 = %.2f%%\n',(-
C_3_CNV+C_3_NEW)/C_3_CNV*100);

%% 模块四 算法估计精度及演示
% 第三部 估计精度计算
P = zeros(L,J,K);
for l = 1:1:L
    for j = 1:1:J
        for k = 1:1:K
            P(l,j,k) = norm(W_Est(:,l,j,k)'*W_DATA1(:,l,j,k)) / ( norm(W_Est(:,l,j,k)) *
norm(W_DATA1(:,l,j,k)) );
        end
    end
end

% 计算最小精度
P_min = min(P,[],"all");
% fprintf('最小精度为%.3f\n',P_min);

% 计算精度0.99达标率
P(P>AccuReq)=1;
P_COUNT = sum(P,'all');
if (P_COUNT/(l*j*k) < 0.9)
    1
end
% fprintf('达标比率为%.3f%%\n',100*P_COUNT/(l*j*k));


% 计算复杂度
C_CNV = C_2_CNV+C_3_CNV;
C_NEW = C_1+ C_2_NEW+C_3_NEW;
% fprintf('模块四：系统的常规算法的复杂度为C_CNV = %d\n',C_CNV);
% fprintf('模块四：系统的优化算法的复杂度为C_NEW = %d\n',C_NEW);
```

```
% fprintf('模块四：系统的优化算法的复杂度优化率为ΔC = %.2f%%\n',(-
C_CNV+C_NEW)/C_CNV*100);

%% 模块五 记录当前精度和运算复杂度
% 求该阈值下的精度达标率
AccurateRate(iterCount) = P_COUNT/(l*j*k);
ComplexAll(iterCount) = (C_CNV-C_NEW)/C_CNV;

end % 阈值变化循环结束
Threshold_num = RouMin:RouInterval:RouMax;
figure(1);
title({"Data"+num2str(Set_num)+"中α和ΔC随λ的变化关系"});
xlim([RouMin RouMax]);
xlabel('相关系数阈值');
yyaxis left;plot(Threshold_num,AccurateRate*100);
ylabel('估计精度达标率α/%')
yyaxis right;plot(Threshold_num,ComplexAll*100);
ylabel('运算复杂度优化率ΔC/%')

filename = "res"+num2str(Set_num);
save(filename, 'AccurateRate', 'ComplexAll');
```

第二问 MATLAB 代码

```
% 第二问
clc
clf
clear
% 模块初始化
H_D = load('Data1_H');
W_D = load('Data1_W');
H_DATA1 = H_D.H;      %N M J K
W_DATA1 = W_D.W;      %M L J K
D_Size = size(H_DATA1);
W_Est = zeros(size(W_DATA1));
L = size(W_DATA1,2);
M = size(H_DATA1,1);
N = size(H_DATA1,2);
J = size(H_DATA1,3);
K = size(H_DATA1,4);
V=zeros(N,N,J,K);
V2=zeros(N,N,J,K);
U=zeros(M,M,J,K);
A=zeros(M,N,J,K);
Q=zeros(M,M,J,K);
R=zeros(M,N,J,K);
V_L = zeros(N,L*J,K);     % 维度为N×LJ×K
```

```
W = zeros(N,L*J,K);    % 维度为N×LJ×K
W2 = zeros(L*J,L*J,K);    % 维度为N×LJ×K
H_COR= zeros(J,K-1);    % 维度为N×LJ×K
V_COR= zeros(J,K);    % 维度为N×LJ×K


%
% 模块一  H矩阵的压缩  基于SVD分解的压缩算法
% 基于SVD分解的压缩
r = 50;                          % 自变量
for j =1:1:J
    H = zeros(M*N,K);
    for k =1:1:K
        H(:,k) = squeeze(reshape( H_DATA1(:,:,j,k),[M*N,1]));
    end
    H_2 = reshape(H,[M*N,K]);
        [U,A,V ] =svd(H_2);
%       [U,A,V] = rand_svd_COMPLETE(H_2,r);       %将矩阵进行SVD分解

    % 截取秩为r的部分
    U_2 = U(:,1:r);
    A_2 = A(1:r,1:r);
    V_2 = V(:,1:r);

    % 还原部分
    H3 = U_2*A_2*V_2';

    %性能计算
    for k =1:1:K
        cor(j,k) = norm(H_2(:,k)-H3(:,k), 'fro')^2;
        E(j,k) = norm(H_2(:,k), 'fro')^2;
    end
end

% 模块一(性能评估部分)  H矩阵的压缩  基于SVD分解的压缩算法
% 误差值
CORV_H_SVD =10*log10( mean(cor,'all') / mean(E,'all'));
% 存储复杂度
COST_H_SVD = r*(M*N+K+1*0.5) /(M*N*K);



 % 模块二  H矩阵的压缩  基于稀疏基的压缩算法
% 基于空间正交基的压缩算法

% 基于角度域，建议空间正交基稀疏字典
Y = 64 ;
```

```
OSR_Y = 4;                                  % 过采样率
R_Y = Y * OSR_Y;
coarse_Y = (0:1:R_Y-1)/R_Y*2*pi -pi;
for n = 1:Y
    dic(n,:)=exp(1j*2*pi*(n-1)*coarse_Y);
end

% 初始化参数
Pathmax = 50;                           %最大允许路径   % 自变量
G = zeros(M,Pathmax);
% 投影查找
for j=1:1:J
    for k=1:1:K
        H_RE= zeros(N,M);
        for m =1:1:M
            H_TEMP = reshape(H_DATA1(:,:,j,k),[M,N]);
            pathcount = 0;
            while pathcount< Pathmax
                if pathcount == 0
                    H_rest = H_TEMP(m,:)';
                end
                pathcount = pathcount+1;
                g=dic'*H_rest/(N);    % g=argmax(c'y)/||c||^2
                prob=(abs(g).^2);           %找到重合的最大值的点
                [~,index_c]=max(prob);      %找到重合的最大值在c的索引位置index 并根据index
的位置，得到对应的角度和时延
                G(m,pathcount) = g(index_c);  %获取增益
                index(m,pathcount) = index_c;    % 记录索引值
                H_rest = H_rest - G(m,pathcount) * dic(:,index_c);        % 复数矩阵乘法和减法
                H_RE(:,m) = H_RE(:,m) + (G(m,pathcount) * dic(:,index_c));


            end
%                 fprintf("路径%d,剩余相关性%.5fdB
\n",pathcount,10*log10(norm(H_rest)/norm(H_TEMP(m,:))));
        end
        % 还原
        H_RE = reshape(H_RE',[M*N,1]);
        H_TE = reshape(H_TEMP,[M*N,1]);
        cor(j,k) = norm(H_RE-H_TE, 'fro')^2;
        E(j,k) = norm(H_TE, 'fro')^2;
    end
end

%% 模块二（性能估计部分）   H矩阵的压缩   基于稀疏基的压缩算法
% 因为字典可以预先存储在系统里，只需要生成一次，适用于全部情况
% 因此不考虑存储复杂度和计算复杂度
% 压缩误差
```

```
CORV_H_CS =10*log10( mean(cor,'all') / mean(E,'all'));
% 存储复杂度
% 索引值为较小的正整数，不考虑存储开销
COST_H_CS = M*Pathmax*K /(M*N*K);
% 计算复杂度
% 每迭代一次 复数矩阵乘法 N*OSR×N  与N×1，复杂度为 (16*N-2) *N*OSR
% 每迭代一次 复数向量元素绝对值  N*OSR×1，复杂度为 (8*N*OSR+23)
% 每迭代一次 复数矩阵乘法 1×1 与1×N，复杂度为 14N
% 每迭代一次 复数矩阵减法 1×N，复杂度为 2N
% 平均迭代次数为 Pathmax/20
% 矩阵间利用率为 80%
CPLX_H_CS = Pathmax/20*( (16*N-2) *N*OSR_Y  + (8*N*OSR_Y+23) + 14*N + 2*N)*M*(1+(1-0.8)*K);



%% 模块三 W矩阵的压缩 基于SVD的压缩算法（矩阵内压缩）
% 基于秩分解的压缩
r = 70;          % 自变量
for j =1:1:J
    W = zeros(N*L,K);
    for k =1:1:K
        W(:,k) = squeeze(reshape( W_DATA1(:,:,j,k),[L*N,1]));
    end
    W_2 = reshape(W,[L*N,K]);
    [U,A,V] = svd(W_2);

    % 截取
    U_2 = U(:,1:r);
    A_2 = A(1:r,1:r);
    V_2 = V(:,1:r);

    % 还原算法
    W3 = U_2*A_2*V_2';
    for k =1:1:K
            cor(j,k) = norm(W(:,k)-W3(:,k),'fro')^2;
            E(j,k) = norm(W(:,k),'fro')^2;
    end
end


%% 模块三（性能估计部分）  W矩阵的压缩 基于SVD的压缩算法
% 压缩误差
CORV_W_SVD =10*log10( mean(cor,'all') / mean(E,'all'));
% 存储复杂度
COST_W_SVD = r*(L*N+K+1*0.5) /(L*N*K);
% 计算复杂度
```

```
CPLX_W_SVD = 0;
```

Optimized_Inverse_Correlation

```
% 逆矩阵计算
function [estWk,numCal] = Optimized_Inverse_Correlation(V_L,replaceFlag)
%%
%V_L的维度是N*LJ*K,64*8*384
%replaceFlag标识哪些值可以被前一个k下的值所替代，即V(j,k+1)=V(j,k),维度为J*K，4*384
%只有k为偶数时的值可能被替换，当k为奇数时，全部使用SOR迭代法进行计算；当k为偶数时，分为三种情况：
%第一种：0个值被替换，则同样使用SOR迭代法；
%第二种：1-3个值被替换，使用优化的相关矩阵算法；
%第三种：全部被替换，则直接可以使用上一个k时的W(k-1)当做W(k)

%% 常量
N = 64;
K = 384;
L = 2;
J = 4;

estWk = zeros(N,L*J,K);
numCal = zeros(1,K);

%% 函数主体
for k = 1:1:K
%       if(mod(k,2) == 1)
%           estWk(:,:,k) = SOR_k(V_L,k);
%       else

        replaceFlag_k = replaceFlag(:,k);
        numEqu1 = sum(replaceFlag_k(:)==1);%找出等于1的个数，即当前k可以被前一个k替代的个数，0-4
        numCal(k) = numEqu1;
        if(numEqu1 == 4)%全部可以被替换，可以直接使用Wk值
            estWk(:,:,k) = estWk(:,:,k-1);
        elseif(numEqu1 == 0)%没有可以被替换的，仍使用SOR迭代法
            estWk(:,:,k) = SOR_k(V_L,k);
        else%有1-3个可以被替换
            [~,Sequence] = sort(replaceFlag(:,k),'descend');%Sequence是次序，4*1
            Seq1 = 2*Sequence-1;
            Seq2 = 2*Sequence;
            Sequence = [Seq1(1);Seq2(1);Seq1(2);Seq2(2);Seq1(3);Seq2(3);Seq1(4);Seq2(4)];
            V_k1(:,:) = V_L(:,:,k-1);%前一个k为奇数的矩阵
            Y1 = V_k1';%V^H
```

```
            Y1 = Y1(Sequence,:);%行变换，根据排序
            X1 = Y1*Y1'+0.01*eye(L*J);%V^H*V+0.01I
            Z1 = estWk(:,:,k-1)';%Y1=X1*Z1
            Z1 = Z1(Sequence,:);

            %矩阵进行分块，1*1,2*2,3*3
            V_A1 = Y1(1:2*numEqu1,:);
            V_B1 = Y1(2*numEqu1+1:L*J,:);
            A1 = X1(1:2*numEqu1,1:2*numEqu1);
            B1 = X1(2*numEqu1+1:L*J,1:2*numEqu1);
            C1 = X1(1:2*numEqu1,2*numEqu1+1:L*J);
            D1 = X1(2*numEqu1+1:L*J,2*numEqu1+1:L*J);
            W_A1 = Z1(1:2*numEqu1,:);
            W_B1 = Z1(2*numEqu1+1:L*J,:);

            V_k2(:,:) = V_L(:,:,k);%当前的k为偶数的矩阵
            Y2 = V_k2';%V^H
            Y2 = Y2(Sequence,:);%行变换，根据排序
            X2 = Y2*Y2'+0.01*eye(L*J);%V^H*V+0.01I

            V_A2 = Y2(1:2*numEqu1,:);
            V_B2 = Y2(2*numEqu1+1:L*J,:);
            A2 = X2(1:2*numEqu1,1:2*numEqu1);
            A2 = A1;
            B2 = X2(2*numEqu1+1:L*J,1:2*numEqu1);
            C2 = X2(1:2*numEqu1,2*numEqu1+1:L*J);
            D2 = X2(2*numEqu1+1:L*J,2*numEqu1+1:L*J);

            W_A2 = inv(A1-C2*inv(D2)*B2)*(A1*W_A1+C1*W_B1-C2*inv(D2)*V_B2);
            W_B2 = inv(D2)*(V_B2-B2*W_A2);
            W_k2 = [W_A2;W_B2];
            W_k2 = W_k2(Sequence,:);
            estWk(:,:,k) = W_k2';
    end
end




end
```

SOR_k

```
% SOR迭代算法

%计算第k组，即k时的一组Wk，而非所有k的Wk，输出的维度为64*8
function [estWk] = SOR_k(V_L,k)%输入为V_L，维度是64*8*384
```

```
K = 384;
L= 2;
J = 4;
N = 64;

V_k(:,:) = V_L(:,:,k);
A = V_k'*V_k+0.01*eye(L*J);%V^H*V+0.01I
B = V_k';%V^H

LowerTri = A;%严格下三角阵
D = A;%对角阵
UpperTri = A;%严格上三角阵
%处理三种矩阵
for u = 1:1:L*J
    for v = 1:1:L*J
        if(u>v)
            UpperTri(u,v) = 0;
            D(u,v) = 0;
        elseif(u<v)
            LowerTri(u,v) = 0;
            D(u,v) = 0;
        elseif(u==v)
            LowerTri(u,v) = 0;
            UpperTri(u,v) = 0;
        end
    end
end

T = 20;
q=1.2;%系数经验值
W_k_t0=zeros(size(V_k));%迭代初始值
W_k_t = zeros(L*J,N,T);
W_k_t(:,:,1) = W_k_t0;
for t = 1:T
%           W_k_t(:,:,t+1) = -inv(D)*(LowerTri+UpperTri)*W_k_t(:,:,t) + inv(D)*B;%Jacobi迭代公式
    W_k_t(:,:,t+1) = inv(D + q* LowerTri)*((1-q)*D - q*UpperTri)*W_k_t(:,:,t) + q*inv(D+q*LowerTri)*B;%SOR迭代公式
    if(norm(W_k_t(:,:,t+1) - W_k_t(:,:,t))^2 < 10^-4)%收敛精度
        break;
    end
end
estWk(:,:) = W_k_t(:,:,t)';%迭代逼近的结果Wk
```

SVD 复杂度计算

```
function [c_rand, c_tradition]=svd_complexity_cal(M_val, N_val, P_val)

if (M_val > N_val)
    c_rand = -1;
    c_tradition = -1;
else
    syms P M N n iterCount


    C_step1 = P * ((16*M-2)*N + (16*P+16-2)*N^2 + 14*N + 48);

    f1 = 33*n*(n+1)/2;
    C_step2 = symsum(f1, n, M-P+1, M) + symsum(f1, n, 2, M-1) + 8100*M ...
    + (16*N-2)*M*P + (16*P-2)*N*P;

    C_total = C_step1 + C_step2;
    C_colub = symsum(f1, n, N-M+1, N) + symsum(f1, n, 2, N-1) + 270*N*iterCount;

    P=P_val;
    M=M_val;
    N=N_val;
    iterCount=100;

    c_rand = subs(C_total);
    c_tradition = subs(C_colub);

end
```