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	**21102860248**
队员姓名	**1.** 赵银霜
	2. 徐子昂
	3. 李哲

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　A21102860248

摘要

本文研究了相关矩阵组的低复杂度计算与存储问题，主要创新点在于充分挖掘矩阵组的关联性，针对相关矩阵组的低复杂度存储问题提出了基于记忆多项式模型的自适应低复杂度存储模型，在满足精度要求的条件下，实现了矩阵组的充分压缩，节约了宝贵的存储资源。

针对问题一的第（1）小问，本文采用了**传统奇异值分解（SVD）与随机奇异值分解（RSVD）**两种算法，在满足精度要求的条件下求解了右奇异矩阵 $\boldsymbol{V}$，并分析对比了两种算法的计算复杂度。针对问题一的第（2）小问，本文采用了**高斯消元法和基于 Strassen’s 算法**的矩阵求逆算法，在满足精度要求的条件下求解了最终的输出矩阵 $\boldsymbol{W}$，并分析对比了两种算法的计算复杂度。

针对问题二中相关矩阵组 $\boldsymbol{H}$ 的压缩与解压缩建模问题，本文创新性地提出了**基于记忆多项式模型的自适应低复杂度存储模型**，充分挖掘了矩阵组的关联性，仅用一组系数实现了一整组数据的标定。在满足压缩、解压缩精度要求的同时，压缩率达到了 **50%—65%**，且由于记忆多项式算法**计算简便、抽取线性**的特性，大大减少了压缩与解压缩过程计算复杂度。

针对问题二中输出信号相关矩阵组 $\boldsymbol{W}$, 由于其同一行块或同一列块内的相关性的规律并不明显，因此基于记忆多项式模型的自适应低复杂度存储模型不再适用，故采用更具有普遍适用性的**主成分分析法**对矩阵进行压缩。采用该算法对矩阵组进行压缩时，在某些数据集上压缩比相对较小，存储复杂度降低，但同时也带来了计算复杂度的急剧增加。

针对第三问，本文摈弃了传统的“压缩解压缩-处理-压缩解压缩”的端到端流程, 采用先处理后存储的方式，减小存储复杂度；尽可能减小矩阵求解 SVD 时

的阶数，有利于减小计算复杂度。基于上述原则，本文设计了一种新的端到端整体方案，并讨论了该方案的计算复杂度和存储复杂度。

关键字： 奇异值分解　记忆多项式　主成分分析　计算复杂度　存储复杂度

目录

1. 问题重述

1.1 引言

矩阵作为高等代数学中的常用工具，广泛应用于计算机视觉、相控阵雷达、声呐、射电天文、无线通信等领域的信号研究中。这一系列信号组成的矩阵间通常在某些维度上存在一定的关联性，因此在进行数学表示时，通常采用相关矩阵组的形式。例如，视频信号中的单帧图像可作为一个矩阵，连续的多帧图像则组成了相关矩阵组。由于视频信号的连续性，相邻帧图像之间存在一定的关联性，这部分关联性在数学上就表现为矩阵间的相关性。随着成像传感器数量、雷达阵列和通信阵列等的持续扩大，待处理信号呈爆炸式增长，常规处理算法对计算和存储的需求也成倍增长，从而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，充分挖掘和利用矩阵间关联性，降低信息冗余度，实现矩阵的低复杂度计算和存储，具有十分重要的价值和意义。

1.2 问题的提出

1.2.1 问题一——相关矩阵组的低复杂度计算

本文考虑一个基于矩阵间相关性和矩阵内部元素相关性的矩阵低复杂度计算模型。给定一相关复数矩阵组 $\boldsymbol{H}=\{\boldsymbol{H}_{j,k}\}$, $\boldsymbol{H}_{j,k}\in\mathbb{C}^{M\times N}, j=1,\ldots,J, k=1,\ldots,K$ 。其中，不同矩阵之间以及同一矩阵的不同元素之间存在一定的相关性，包括：

- 处于同一行、不同列的矩阵间的关联性，即 $\{\boldsymbol{H}_{j,1},\boldsymbol{H}_{j,2},\boldsymbol{H}_{j,3},\cdots,\boldsymbol{H}_{j,K}\}$ 间的关联性。
- 矩阵内部各元素间关联性，即矩阵

$$\boldsymbol{H}_{j,k}=\begin{bmatrix} h_{1,1}^{(j,k)} & h_{1,2}^{(j,k)} & h_{1,3}^{(j,k)} & \cdots & h_{1,N}^{(j,k)} \\ h_{2,1}^{(j,k)} & h_{2,2}^{(j,k)} & h_{2,3}^{(j,k)} & \cdots & h_{2,N}^{(j,k)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{M,1}^{(j,k)} & h_{M,2}^{(j,k)} & h_{M,3}^{(j,k)} & \cdots & h_{M,N}^{(j,k)} \end{bmatrix}$$

的各个元素 $\left\{h_{m,n}^{(j,k)}\right\}, m=1,\ldots,M, n=1,\ldots,N$ 也存在关联。

定义矩阵组 $\boldsymbol{H}=\{\boldsymbol{H}_{j,k}\}$ 上的一组数学运算，其中间结果 $\boldsymbol{V}=\{\boldsymbol{V}_{j,k}\}$ 由下式给出：

$$\begin{gathered}\boldsymbol{V}_{j,k}=\operatorname{svd}\left(\boldsymbol{H}_{j,k}\right), \text{or } \boldsymbol{H}_{j,k}=\boldsymbol{U}_{j,k}\boldsymbol{S}_{j,k}\widetilde{\boldsymbol{V}}_{j,k}^{\mathrm{H}}, \boldsymbol{V}_{j,k}=\widetilde{\boldsymbol{V}}_{j,k}^{H}(:,1:L) \\ j=1,\ldots,J; k=1,\ldots,K\end{gathered} \tag{1}$$

其中，svd(·) 为矩阵的奇异值分解（即，SVD 分解）中求解右奇异向量的过程，待求解的 $\boldsymbol{V}_{j,k}$ 是由 $\boldsymbol{H}_{j,k}$ 的前 L 个右奇异向量构成的矩阵，维度为 $N \times L$。

进一步，定义矩阵 $\boldsymbol{W} = \{\boldsymbol{W}_{j,k}\}$ 为最终输出结果，$\boldsymbol{W}_k$ 的计算公式为：

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

其中，$\boldsymbol{V}_k$ 为将不同 j 下标、相同 k 下标的 $\boldsymbol{V}_{j,k}$ 横向拼接所得，即 $\boldsymbol{V}_k$，维度为 $N \times LJ$，$\boldsymbol{W}_k$ 的维数同 $\boldsymbol{V}_k$。σ^2 为固定常数；$\boldsymbol{I}$ 为单位矩阵，维度为 $LJ \times LJ$。$\boldsymbol{W}_{j,k}$ 为将各 $\boldsymbol{W}_k$ 按如下公式拆解所得顺序排列的、维度为 $N \times L$ 的子矩阵:

$$\boldsymbol{W}_k = [\boldsymbol{W}_{1,k}, \ldots, \boldsymbol{W}_{j,k}, \ldots, \boldsymbol{W}_{J,k}] \tag{3}$$

传统的数学方法可以给出 $\boldsymbol{V}$ 和 $\boldsymbol{W}$ 的精确求解结果。但随着矩阵维度的增加，采用传统方法求解 $\boldsymbol{V}_k$ 和 $\boldsymbol{W}_k$ 将会带来计算上的高复杂度，相关处理器件和算法的实现成本和功耗都将显著增加，给整个系统带来巨大的负担。因此，需分析和利用相关矩阵组的关联性，设计数学模型对输出结果 W 进行估计，在得到近似结果的同时显著降低计算复杂度。建模过程可表示为：

$$\widehat{\boldsymbol{W}} = f(\boldsymbol{H})$$

其中 $\widehat{W}$ 即为对输出结果 W 的建模估计。这一建模过程又可拆分为 2 个步骤：

$$\widehat{\boldsymbol{V}} = f_1(\boldsymbol{H}), \quad \widehat{\boldsymbol{W}} = f_2(\widehat{\boldsymbol{V}})$$

其中 $f_1(\cdot)$ 表示从输出矩阵组 H 到中间结果 V 的建模过程，$\widehat{W}$ 表示最终结果 W 的建模估计。

建模过程中，采用 $\rho_{l,j,k}(\boldsymbol{W})$ 评估模型的估计精度：

$$\rho_{l,j,k}(\boldsymbol{W}) = \frac{\left\| \widehat{W}_{l,j,k}^H \boldsymbol{W}_{l,j,k} \right\|_2}{\left\| \widehat{\boldsymbol{W}}_{l,j,k} \right\|_2 \left\| \boldsymbol{W}_{l,j,k} \right\|_2}, l = 1, \ldots, L$$

其中 $\|\cdot\|_2$ 表示欧几里得范数，$\boldsymbol{W}_{l,j,k}$ 表示 $\boldsymbol{W}_{j,k}$ 的第 l 列。为了表述方便，额外定义模型的最低估计精度 $\rho_{\min}(\boldsymbol{W})$ 为：

$$\rho_{\min}(\boldsymbol{W}) \triangleq \min \rho_{l,j,k}(\boldsymbol{W}), l \in \{1, 2, \ldots, L\}, j \in \{1, 2, \ldots, J\}, k \in \{1, 2, \ldots, K\}$$

其中，$\min_{l,j,k}(\cdot)$ 表示在 l, j, k 三个维度上取最小值。中间结果 V 的建模精度 $\rho_{l,j,k}(\boldsymbol{V})$ 与此相同。

此外，模型的计算复杂度定义为由矩阵组 H 计算得到结果矩阵组 W 所需要的总计算复杂度。复数矩阵的运算需拆解为基本复数运算，而基本复数运算又要进一步拆解为基本实数运算。基本实数运算的复杂度依据下表计算。

表 **1**　实数基本运算的计算复杂度

符号	意义
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

对于相关矩阵组的低复杂度计算，本文提出以下问题：

（1）给定输入矩阵组 H, 分析其数据间的关联性，在最小估计精度满足 $\rho_{\min}(\boldsymbol{V}) \geq \rho_{th} = 0.99$ 的条件下，设计出计算复杂度最小的近似分析模型 $\widehat{\boldsymbol{V}} = f_1(\boldsymbol{H})$。

（2）基于上述矩阵组 H，分析其数据间的关联性，在最小估计精度满足 $\rho_{\min}(\boldsymbol{W}) \geq \rho_{th} = 0.99$ 的条件下，综合设计出计算复杂度最小的近似分析模型 $\widehat{\boldsymbol{W}} = f_1(\boldsymbol{H})$。该模型可在问题（1）的基础上设计 $\widehat{\boldsymbol{W}} = f_2(\widehat{\boldsymbol{V}})$，从而依据综合模型 $\widehat{\boldsymbol{W}} = f(\boldsymbol{H}) = f_2(f_1(\boldsymbol{H}))$ 估计 $\boldsymbol{W}$, 也可不考虑中间结果 $\boldsymbol{V}$, 直接设计从 $\boldsymbol{H}$ 到 $\boldsymbol{W}$ 的估计模型，此时问题（1）的求解可跳过。

综上，问题一需要解决以下问题：

（1）分析相关矩阵组数据关联性

（2）构建矩阵 $\boldsymbol{H}$ 与 $\boldsymbol{W}$ 间的近似估计模型

（3）分析模型的计算复杂度

1.2.2 问题二——相关矩阵组的低复杂度存储

存储复杂度定义为存储矩阵组所需的内存空间大小，以 bit 为单位计算。上述复数矩阵组 $\boldsymbol{H}$、$\boldsymbol{W}$ 中，单个复数元素的实部和虚部均采用 32 比特单精度浮点数表示，因此整个矩阵组 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 所占用的存储空间分别为 64MNJKbit 和 64NLJKbit。

为了节省存储资源，需要对 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 分别进行压缩处理，再对压缩后的数据进行存储。压缩处理函数定义为 $P_1(\cdot)$ 和 $P_2(\cdot)$，压缩后的数据表示为 $P_1(\boldsymbol{H})$

和 $P_1(\boldsymbol{W})$。由于此时 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 以压缩的形式存储在存储器中，因此再次处理前，需要在读取出 $P_1(\boldsymbol{H})$ 和 $P_1(\boldsymbol{W})$ 后，对其进行解压缩操作，对 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 的解压缩函数分别定义为：$G_1(\cdot)$，$G_2(\cdot)$，解压缩处理后得到的数据定义为：$\widehat{\boldsymbol{H}}$ 和 $\widehat{\boldsymbol{W}}$，其函数表达式分别为：

$$\widehat{\boldsymbol{H}} = G_1\left(P_1(\boldsymbol{H})\right), \widetilde{\boldsymbol{W}} = G_2\left(P_2(\boldsymbol{W})\right)$$

经过压缩和解压缩操作得到的 $\widehat{\boldsymbol{H}}$ 和 $\widehat{\boldsymbol{W}}$ 和原始的 $\boldsymbol{H}$ 和 $\boldsymbol{W}$ 之间存在一定的误差，这部分误差定义为

$$\text{err}_{\boldsymbol{H}} = 10 * \log_{10} \frac{E\left\{\left\|\widehat{\boldsymbol{H}}_{j,k} - \boldsymbol{H}_{j,k}\right\|_F^2\right\}}{E\left\{\|\boldsymbol{H}_{j,k}\|_F^2\right\}}, \text{err}_W = 10 * \log_{10} \frac{E\left\{\left\|\widehat{\boldsymbol{W}}_{j,k} - \boldsymbol{W}_{j,k}\right\|_F^2\right\}}{E\left\{\|\boldsymbol{W}_{j,k}\|_F^2\right\}}$$

其中，$\|\cdot\|_F$ 表示 Frobenius 范数；$E\{\cdot\}$ 表示期望。

对于相关矩阵组的低复杂度存储，本文提出以下问题：基于给定的所有矩阵数据 $\boldsymbol{H}$ 和 $\boldsymbol{W}$, 分析各自数据间的关联性，分别设计出相应的压缩模型 $P_1(\cdot)$、$P_2(\cdot)$ 和解压缩模型 $G_1(\cdot)$、$G_2(\cdot)$，在满足误差 $\text{err}_{\boldsymbol{H}} \leq E_{th1} = -30$ dB、$\text{err}_W \leq E_{th2} = -30$ dB 的情况下，使得存储复杂度和压缩、解压缩的计算复杂度最低，复杂度计算方法同第一题。

综上，问题二需要解决以下问题：

（1）分析矩阵组数据间关联性

（2）设计压缩和解压缩函数

（3）分析存储复杂度和压缩、解压缩过程的计算复杂度

2. 模型的假设

- 假设对于相关矩阵组 $\boldsymbol{H} = \{\boldsymbol{H}_{j,k}\}$ 仅在其同一行块内部的 K 个矩阵间存在相关性，矩阵组 $\boldsymbol{H}$ 中属于不同行块的矩阵间不存在相关性。
- 假设在 SVD 分解右奇异矩阵时，计算复杂度的分析中不考虑代入特征值 λ 求解特征向量的过程。

3. 符号说明

符号	意义
$\boldsymbol{H}$	相关矩阵组
$\boldsymbol{H}_{j,k}$	相关矩阵组内部矩阵
M	矩阵组 $\boldsymbol{H}$ 内矩阵 $\boldsymbol{H}_{j,k}$ 的行数
N	矩阵组 $\boldsymbol{H}$ 内矩阵 $\boldsymbol{H}_{j,k}$ 的列数
J	相关矩阵组 $\boldsymbol{H}$ 每一列块的矩阵数
K	相关矩阵组 $\boldsymbol{H}$ 每一行块的矩阵数
$\mathrm{svd}(\cdot)$	矩阵奇异值分解求右奇异向量过程
$\boldsymbol{V}_{j,k}$	矩阵 $\boldsymbol{H}_{j,k}$ 的前 L 个右奇异向量构成的矩阵
$\boldsymbol{U}_{j,k}$	矩阵 $\boldsymbol{H}_{j,k}$ 的左奇异向量构成的矩阵
$\boldsymbol{S}_{j,k}$	矩阵 $\boldsymbol{H}_{j,k}$ 的奇异值矩阵
$\boldsymbol{V}$	由 $\boldsymbol{V}_{j,k}$ 横向拼接成的矩阵组
$\widehat{\boldsymbol{V}}$	V 的建模估计
$\boldsymbol{W}$	最终输出矩阵
$\widehat{\boldsymbol{W}}$	$\boldsymbol{W}$ 的建模估计
$\rho_{l,j,k}(\boldsymbol{V})$	$\boldsymbol{V}$ 的建模精度
$\rho_{\min}(\boldsymbol{V})$	$\boldsymbol{V}$ 的最低建模精度
$\rho_{l,j,k}(\boldsymbol{W})$	$\boldsymbol{W}$ 的建模精度
$\rho_{\min}(\boldsymbol{W})$	$\boldsymbol{W}$ 的最低建模精度
$\|\cdot\|_2$	矢量的欧几里得范数
σ^2	固定常数
$\boldsymbol{I}$	单位矩阵
$\boldsymbol{H}_{j1,kp}$	$\boldsymbol{H}$ 中某一 j1 行 kp 列矩阵单元
$\boldsymbol{H}_{j1,kq}$	$\boldsymbol{H}$ 中某一 j1 行 kq 列矩阵单元
$P_1(\cdot)$	$\boldsymbol{H}$ 的压缩函数
$P_2(\cdot)$	$\boldsymbol{W}$ 的压缩函数
$G_1(\cdot)$	$\boldsymbol{H}$ 的解压缩函数
$G_2(\cdot)$	$\boldsymbol{W}$ 的解压缩函数
$\|\cdot\|_F$	矩阵的 Frobenius 范数
$E\{\cdot\}$	求期望运算

4. 问题一：低复杂度计算

4.1 矩阵组关联性分析

相关矩阵组 $\boldsymbol{H}$ 的关联性包括两个方面：矩阵间关联性和矩阵单元内部各元素间的关联性，利用这些关联性可以从以下两个角度减少求解矩阵 $\boldsymbol{W}$ 的计算复杂度：

（1）利用矩阵单元 $\boldsymbol{H}_{j,k}$ 内部元素间的关联性，降低具体奇异值求解和矩阵求逆过程中的计算复杂度。

（2）利用矩阵组 $\boldsymbol{H}$ 内部各个矩阵单元间的关联性，降低一组数学运算的整体计算复杂度。具体思路为：通过数学运算求解出部分矩阵单元的 $\widehat{\boldsymbol{W}}_{j,k}$ 后，利用矩阵间关系求解出剩余矩阵单元对应的 $\widehat{\boldsymbol{W}}_{j,k}$，从而得到最终输出矩阵的估计 $\widehat{\boldsymbol{W}}$。

因此，在构建矩阵 $\boldsymbol{H}$ 与 $\boldsymbol{W}$ 间的近似估计模型前，需先对矩阵 $\boldsymbol{H}$ 的关联性进行分析。其中，矩阵单元 $\boldsymbol{V}_{j,k}$ 内部元素间的关联性可直接体现在奇异值求解算法中，因此这里对此不作具体分析，主要分析矩阵组 $\boldsymbol{H}$ 中处于同一行块的矩阵单元之间的关联性。

本文采用 Pearson 相关系数来度量矩阵间的相关性，矩阵组 $\boldsymbol{H}$ 中同一行块的任意两个矩阵单元 $\boldsymbol{H}_{j1,kp}$ 与 $\boldsymbol{H}_{j1,kq}$ 间的 Pearson 相关系数的定义为：

$$\rho(\boldsymbol{H}_{j_1,k_p},\boldsymbol{H}_{j_1,k_q})=\frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{\boldsymbol{H}_{j_1,k_{p_i}}-\mu_{\boldsymbol{H}_{j_1,k_p}}}{\sigma_{\boldsymbol{H}_{j_1,k_p}}}\right)\left(\frac{\boldsymbol{H}_{j_1,k_{q_i}}-\mu_{\boldsymbol{H}_{j_1,k_q}}}{\sigma_{\boldsymbol{H}_{j_1,k_q}}}\right)$$

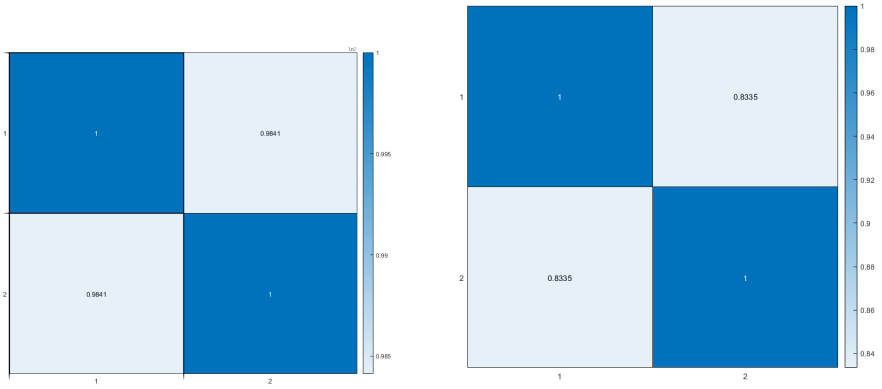
其中 $\mu_{\boldsymbol{H}_{j_1,k_p}}$ 和 $\sigma_{\boldsymbol{H}_{j_1,k_p}}$ 分别是 $\boldsymbol{H}_{j_1,k_p}$ 的均值和标准差，$\mu_{\boldsymbol{H}_{j_1,k_q}}$ 和 $\sigma_{\boldsymbol{H}_{j_1,k_q}}$ 分别是 $\boldsymbol{H}_{j_1,k_q}$ 的均值和标准差。$\boldsymbol{H}_{j1,kp}$ 与 $\boldsymbol{H}_{j1,kq}$ 间的相关系数矩阵定义为：

$$R=\begin{pmatrix}\rho(\boldsymbol{H}_{j1,kp},\boldsymbol{H}_{j1,kp}) & \rho(\boldsymbol{H}_{j1,kp},\boldsymbol{H}_{j1,kq})\\ \rho(\boldsymbol{H}_{j1,kq},\boldsymbol{H}_{j1,kp}) & \rho(\boldsymbol{H}_{j1,kq},\boldsymbol{H}_{j1,kq})\end{pmatrix}$$

依据上述定义求解得复数矩阵组 $\boldsymbol{H}$ 中同一行块的各个矩阵单元间的相关系数为复数。由于相关系数为复数没有实际意义，因此，本文将复数矩阵单元间的相关性分析转换为复数矩阵单元的实部相关性、虚部相关性以及幅值相关性、相位相关性的分析。以第一组数据的 $H_{m,n}^{1,1}$ 和 $H_{m,n}^{1,2}$ 为例，所得实部、虚部、幅值和相位的相关性图分别如图所示。

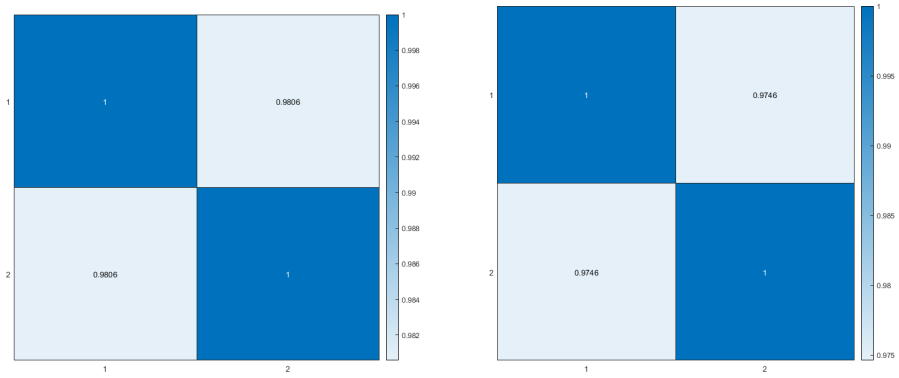
为了深入研究相关性，又对相邻矩阵的列和行之间的进行了相关系数的计算，如下图所示。

分析上图4，可知相邻的矩阵之间的同样位置的元素有很强的相关性，距离较远的关联性则比较弱，单个矩阵内部的关联性也比较弱。



(a) 幅度　　(b) 相位

图 **1**　幅度相位表示下的矩阵间相关性



(a) 实部　　(b) 虚部

图 **2**　实部虚部表示下的矩阵间相关性

4.2 模型的建立与求解

$\boldsymbol{W}$ 矩阵的求解可分为两个步骤进行：

Step1: 构建中间矩阵组 $\boldsymbol{V}$ 近似分析模型 $\widehat{\boldsymbol{V}} = f_1(\boldsymbol{H})$

Step2: 构建最终输出矩阵组 $\boldsymbol{W}$ 近似分析模型 $\widehat{\boldsymbol{W}} = f_2(\boldsymbol{V})$

针对 Step1，传统奇异值分解方法即可求解出 $\widehat{\boldsymbol{V}}$。但采用此方法计算时，计算复杂度非常高，因此，本文又采用随机 SVD 算法，利用矩阵内部元素间相关性显著降低了求解 $\widehat{\boldsymbol{V}}$ 过程的计算复杂度。

针对 Step2，由矩阵 $\boldsymbol{V}_k$ 求解 $\boldsymbol{W}_k$ 的计算公式为：$\boldsymbol{W}_k = \boldsymbol{V}_k \left(\boldsymbol{V}_k^H \boldsymbol{V}_k + \sigma^2 \boldsymbol{I}\right)^{-1}$。在获取逆矩阵的过程中，若采用高斯消元法，计算复杂度很高，采用文献中的 Strassen’s 方法时，可以将复杂度降低。

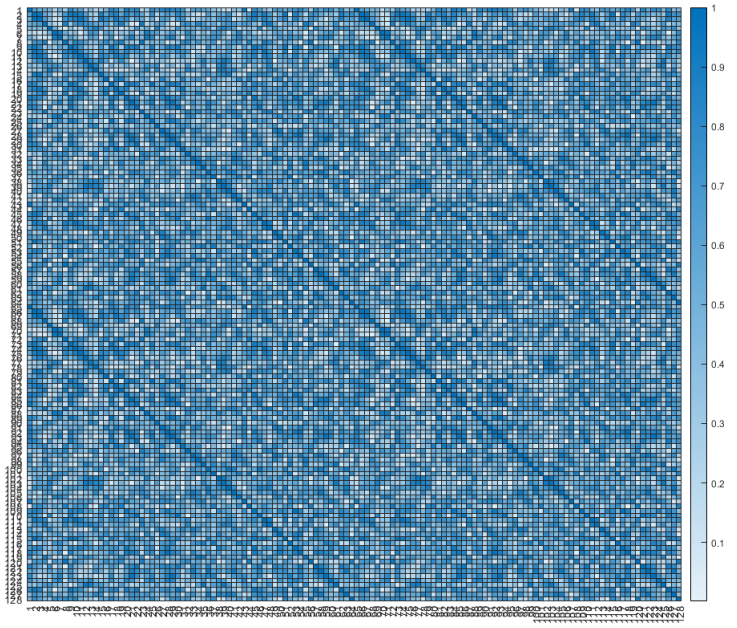


图 **3**　**2** 个矩阵列与列之间的相关性

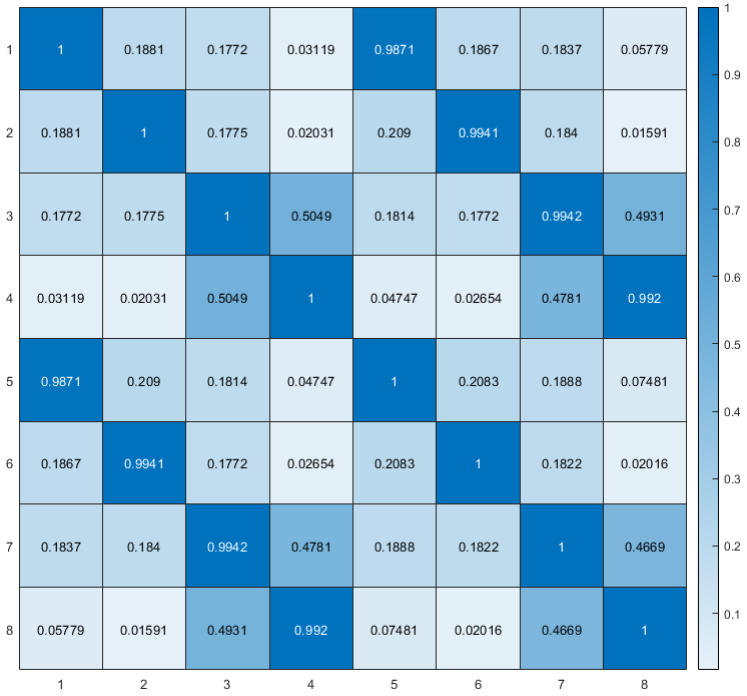


图 **4**　**2** 个矩阵行与行之间的相关性

4.2.1 子问题一求解

（1）传统 SVD 求解

奇异值分解（SVD）是特征值分解在任意维数矩阵上的推广，对于维数为 $m \times n$ 的任意复数矩阵 $\boldsymbol{A}$，其 SVD 表示为：$\mathbf{P\Sigma Q}^*$，其中 $\mathbf{\Sigma}$ 是 $m \times n$ 阶的对角线上有非负实数的矩形对角矩阵，$\mathbf{P}$、$\mathbf{Q}$ 分别为 $m \times m$ 阶和 $n \times n$ 的酉矩阵。传统方法对 $\boldsymbol{H}_{j,k}$ 进行 SVD 分解求解右奇异向量组 $\boldsymbol{V}_{j,k}$ 的具体步骤如下，matlab 代码见附录：

传统 SVD 求解步骤
Step1：构造 $\boldsymbol{M}_{j,k} = \boldsymbol{H}_{j,k}^T \boldsymbol{H}_{j,k}$
Step2: 对 $\boldsymbol{M}_{j,k}$ 进行特征分解，得到特征向量 v_i
Step3: 取最大的两个特征值对应的特征向量横向拼接成向量 $\widehat{\boldsymbol{V}}$

传统 SVD 算法求解所得 $\widehat{\boldsymbol{V}}$ 结果存于附件中，对此估计结果进行精度检验，其最低建模精度 $\rho_{\min}(V)$ 为 1，满足题目要求的 $\rho_{\min}(\boldsymbol{V}) \geq \rho_{th} = 0.99$。

（2）随机 SVD 求解

由 Halko 等提出的随机 SVD 算法，通过将概率论与线性代数相结合提出了一种构造矩阵近似分解的概率算法，即随机 SVD。该算法利用矩阵内部元素间的相关性显著降低了计算复杂度。随机 SVD 求解 $\boldsymbol{H}_{j,k}$ 的右奇异向量组 $\boldsymbol{V}_{j,k}$ 的具体步骤如下，matlab 代码见附录：

随机 SVD 求解步骤
Step1：生成一个 $b \times c$ 的高斯随机矩阵 Ω
Step2: 生成一个 $a \times c$ 的矩阵 $\mathbf{Y}_{j,k} = \boldsymbol{H}_{j,k}\mathbf{\Omega}$
Step3: 对 $\mathbf{Y}_{j,k}$ 进行 QR 分解得到正交矩阵 Q
Step4: 构建 $\mathbf{B}_{j,k} = \mathbf{Q}_{j,k}^* \boldsymbol{H}_{j,k}$
Step5: 对 $\mathbf{B}_{j,k}$ 进行 SVD 分解，$\mathbf{B}_{j,k} = \tilde{\mathbf{P}_{j,k}}\mathbf{\Sigma Q}_{j,k}^*$
Step6: 取 $\mathbf{Q}_{j,k}$ 的前两列得到 $\widehat{\boldsymbol{V}_{j,k}}$

其中 b 等于矩阵 $\boldsymbol{H}_{j,k}$ 的列数 N，r 为 $\boldsymbol{H}_{j,k}$ 的秩，$c \geq r$，$s = c - r$ 为过采样参数。

随机 SVD 算法求解所得 $\widehat{\boldsymbol{V}}$ 结果存于附件中，对此估计结果进行精度检验，其最低建模精度 $\rho_{\min}(\boldsymbol{V})$ 为 1，满足题目要求的 $\rho_{\min}(\boldsymbol{V}) \geq \rho_{th} = 0.99$。

4.2.2 子问题二求解

计算 $\boldsymbol{W}_k$ 的公式为

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

。

在矩阵求逆的过程中，采用了高斯消元法和基于 Strassen's 方法的矩阵求逆算法这两种算法，后者相较于前者有较低的计算复杂度。

（1）高斯消元法

高斯消元法的基本步骤如下：

Step1. 将 $\boldsymbol{A}$ 填充成增广矩阵 $(\boldsymbol{A} \mid \boldsymbol{I})$，$\boldsymbol{I}$ 为 $n \times n$ 阶单位矩阵

$$(A \mid I) = \left(\begin{array}{cccc|cccc} a_{11} & a_{12} & \cdots & a_{nn} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & 0 & 0 & \cdots & 1 \end{array} \right) \tag{4}$$

Step2. 第一行分别乘某个数加到下面的行，使得第一个主元 a_{11} 下的数字全为 0

Step3. 第二行分别乘某个数加到下面的行，使得第二个主元 a_{22} 下的数字全为 0

Step4. 重复上述步骤，直至将 $\boldsymbol{A}$ 转换为上三角矩阵

Step5. 将每一行除以该行主元的值，使得左边矩阵主元的值为 1

Step6. 最后一行分别乘某个数加到上面的行，使得 $a_{i,n}, i = 1, \ldots, n-1$ 下的数字全为 0

Step7. 倒数第二行分别乘某个数加到上面的行，使得 $a_{i,n-1}, i = 1, \ldots, n-2$ 下的数字全为 0

Step8. 重复上述步骤，直到将 A 转换为单位矩阵。

（2）基于 Strassen's 方法的矩阵求逆算法

基于 Strassen's 方法的矩阵求逆算法的基本步骤如下：

如果是一个 $m2^{k+1}$ 阶的矩阵 $\boldsymbol{A}$ 进行求逆，则记为：

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad A^{-1} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \tag{5}$$

其中，A_{ik}, C_{ik} 是 $m2^k$ 的矩阵。然后求解：

$$\begin{aligned}
&\mathrm{I} = A_{11}^{-1},\\
&\mathrm{II} = A_{21}\mathrm{I},\\
&\mathrm{III} = \mathrm{I}A_{12},\\
&\mathrm{IV} = A_{21}\mathrm{III},\\
&\mathrm{V} = \mathrm{IV} - A_{22},\\
&\mathrm{VI} = \mathrm{V}^{-1},\\
&C_{12} = \mathrm{III}\cdot\mathrm{VI},\\
&C_{21} = \mathrm{VI}\cdot\mathrm{II},\\
&\mathrm{VII} = \mathrm{III}\cdot C_{21},\\
&C_{\mathbf{11}} = \mathrm{I} - \mathrm{VII},\\
&C_{22} = -\mathrm{VI}
\end{aligned} \tag{6}$$

本文采用上述两种算法对最终输出矩阵 $\boldsymbol{W}$ 进行了求解，MATLAB 代码见附录，估计结果 $\widehat{\boldsymbol{W}}$ 存于附件中。对此估计结果进行精度检验，其最低建模精度 $\rho_{\min}(W)$ 为 1，满足题目要求的 $\rho_{\min}(\boldsymbol{W}) \geq \rho_{th} = 0.99$

4.3 计算复杂度分析

4.3.1 复数基本运算的计算复杂度

根据表 1 中所给实数基本运算的计算复杂度计算复数基本运算的计算复杂度。在矩阵变换中仅仅需要加减、乘法和除法这几种基本运算。

（1）复数加法过程 $a+bj+c+dj = (a+b)+(c+d)j$：使用了 2 次实数加 (减) 法。

（2）复数乘法过程 $(a+bj)(c+dj) = (ac-bd)+(ad+bc)j$：使用了 4 次实数乘法和 2 次实数加 (减) 法。

如果令 $m_1 = (a+b)(c-d) = (ac-bd)+bc-ad$，使用 2 次实数加 (减) 法和 1 次实数乘法；

$m_2 = bc$，使用 1 次实数乘法；

$m_3 = ad$，使用 1 次实数乘法。

则有 $(a+bj)(c+dj) = (ac-bd)+(ad+bc)j = (ac-bd)+(ad+bc)j = (m_1-m_2+m_3)+(m_2+m_3)j$。经统计可知，该复数乘法过程共使用 3 次实数乘法和 5 次实数加 (减) 法。根据表格 1，上述计算方法具有计算复杂度 14。

（3）复数除法过程

$$\frac{a+bj}{c+dj} = \frac{(a+bj)(c-dj)}{(c+dj)(c-dj)} = \frac{(ac+bd)+(bc-ad)j}{c^2+d^2} \tag{7}$$

其中，$1/c^2+d^2$ 使用 1 次实数加 (减) 法、2 次实数乘法和一次实数倒数；

$(ac-bd)+(ad+bc)j$ 根据对复数乘法过程的简化可知使用 3 次实数乘法和 5 次实数加 (减) 法；

$((ac+bd)+(bc-ad)j)/(c^2+d^2)$ 表示为实数 $1/c^2+d^2$ 与复数 $(ac-bd)+(ad+bc)j$ 相乘，使用了 2 次实数乘法；

根据表格 1，上述计算方法具有计算复杂度为 52。

（4）倒数求解过程

$$\frac{1}{a+bj}=\frac{a-bj}{a^2+b^2} \tag{8}$$

其中，$1/(a^2+b^2)$ 使用 1 次实数加 (减) 法、2 次实数乘法和一次实数倒数；$(a-bj)/(a^2+b^2)$ 表示为实数 $1/(a^2+b^2)$ 与复数 $a-bj$ 相乘，使用了 2 次实数乘法;

根据表格 1，上述计算方法具有计算复杂度为 38。

综上所述，可得复数基本运算的计算复杂度如表 2 所示

4.3.2 传统 SVD 方法的计算复杂度

传统 SVD 分解过程包括三个步骤，以下对其三个步骤的计算复杂度分别进行分析。

（1）求矩阵乘法 $\boldsymbol{A}^\top\boldsymbol{A}$ 的计算复杂度

$\boldsymbol{A}$ 为 $M\times N$ 阶矩阵, 则 $\boldsymbol{A}^\top$ 为 $N\times M$ 阶仿真，计算过程使用 MN^2 次复数乘法和 $(M-1)N^2$ 次复数加（减）法。

（2）求矩阵 $\boldsymbol{A}^\top\boldsymbol{A}$ 的特征值和特征向量对 $\boldsymbol{A}^\top\boldsymbol{A}-\lambda\boldsymbol{I}$ 使用高斯消元法求解矩阵特征值和特征向量的过程共使用了 $2\times(\mathrm{N}^3/3+\mathrm{N}^2/2-5\,\mathrm{N}/6)$ 次复数乘法，$2\times\left(\mathrm{N}^3/3+\mathrm{N}^2/2-5\,\mathrm{N}/6\right)$ 和 $(N+1)N$ 次复数除法。

（3）将特征向量 $v_i, i=1,2\ldots,n$ 拼接在一起得到右奇异矩阵的过程不涉及复杂度。

因此，传统 SVD 计算复杂度为：$\mathrm{MN}^2+2\times\left(\mathrm{N}^3/3+\mathrm{N}^2/2-5\,\mathrm{N}/6\right)$ 次复数乘法、$(M-1)N^2+2\times(N^3/3+N^2/2-5N/6)$ 次复数加法。

4.3.3 随机 SVD 方法的计算复杂度

依据上述随机 SVD 方法求解步骤可知，随机 SVD 的计算复杂度主要来源于矩阵 **Y** 的 QR 分解，其计算复杂度近似为近似为 O(MNL)，而其中中间矩阵 **B** 的奇异值分解的复杂度仅仅为 O(NLlogL)。因此，此算法的计算复杂度近似为 $\mathrm{O(MNL+NLlogL)}$。

4.3.4 高斯消元法求逆矩阵的计算复杂度

高斯消元法中，步骤 2—4 中的计算复杂度为：$n^3/3+n^2/2-5n/6$ 次复数乘法和 $n^3/3+n^2/2-5n/6$ 复数加法；步骤 5 的计算复杂度为：$n(n-1)$ 次复数除法；步骤 6—8 的计算复杂度为：$(n^3/3+n^2/2-5n/6)$ 次复数乘法和 $(n^3/3+n^2/2-5n/6)$ 次复数加法。

所以高斯消元法总的计算复杂度为：$2\times((n^3/3+n^2/2-5n/6))$ 次复数乘法、$2\times((n^3/3+n^2/2-5n/6))$ 次复数加法和 $n(n-1)$ 次复数除法

4.3.5 基于 Strassen's 方法的矩阵求逆算法的计算复杂度

基于 Strassen's 方法矩阵求逆算法的计算复杂度为：计算 $m2^k$ 阶矩阵的逆矩阵需要的 $m2^k$ 次复数除法，不超过 $\frac{6}{5}m^37^k-m2^k$ 次的复数乘法，不超过 $\frac{6}{5}(m+5)m^27^k-7\left(m2^k\right)^2$ 次的复数加法。

4.4 总结

4.4.1 子问题一计算复杂度

1. 若采用传统 SVD 方法求解 $\boldsymbol{V}$，则共进行了：2.99×10^8 次复数乘法和 2.93×10^8 次复数加法，因此总计算复杂度为 4.77×10^9

4.4.2 子问题二计算复杂度

2. 若采用随机 SVD 方法求解 $\boldsymbol{V}$ 计算复杂度近似为 $O(MNL+NLlogL)$，远低于传统 SVD 算法的计算复杂度 $O(MN_2+N_3)$。

子问题 2 中，$\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}$ 的计算过程使用了 N(LJ)^2 次复数乘法、N(LJ)^2 次复数加（减）法。矩阵 $\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}$ 求逆的计算复杂度如前所述。最后，由于 $\left(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)^{-1}$ 为对角阵，因此 $\boldsymbol{V}_k\left(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)^{-1}$ 可以简化为 $\boldsymbol{V}_k$ 的每一列乘以 $\left(\boldsymbol{V}_k^H\boldsymbol{V}_k+\sigma^2\boldsymbol{I}\right)^{-1}$ 对角线上的对应列的元素，使用了 $N\times LJ$ 次复数乘法，没有使用加法操作。因此总的复杂度为：

1. 若采用高斯消元法进行矩阵求逆，则共进行了 1.72×10^6 次复数乘法，1.72×10^6 次复数加法，2.8×10^4 复数除法和 3072 次复数求倒数，则总的计算复杂度约为 2.90×10^7。

2. 若采用基于 Strassen's 方法的矩阵求逆算法，其总的计算复杂度的精确值无法求出，其计算复杂度最高为 3.32×10^6

5. 问题二：低复杂度存储

5.1 问题分析

此问题的求解流程图如下所示。

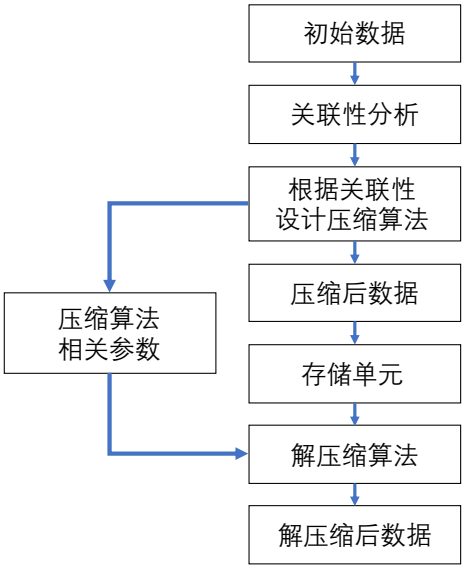


图 5　问题 2 求解流程图

5.2 矩阵组关联性分析

问题一中已对相关矩阵组 ***H*** 中同一行块的矩阵单元间的关联性进行了分析，这里就不再赘述。采用相同的方法对矩阵组 ***W*** 中同一行块的矩阵单元间的关联性进行分析，结果如下所示，左右矩阵关联表示 2 个 W 矩阵在 J 方向相同，K 方向相邻，上下矩阵关联表示 2 个 W 矩阵在 J 方向相邻，K 方向相同。由于 H 矩阵在 J 方向没有关联度，而 W 矩阵可能在 J 方向有关联性，故对 2 个维度都做了关联性分析。

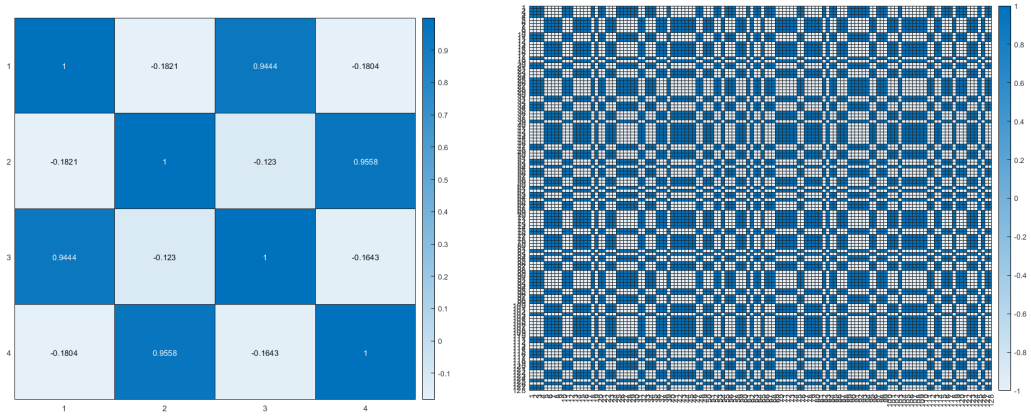
可见 W 矩阵不仅在左右维度上有相关性，在上下维度也有相关性。

5.3 压缩、解压缩模型构建

5.3.1 基于记忆多项式的自适应低复杂度存储模型

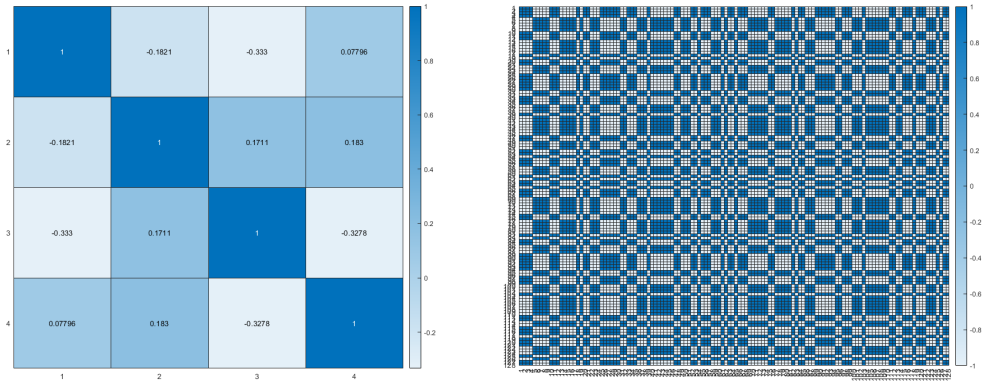
Volterra 级数是一种建模非线性和具有记忆效应的时变系统的方法。Volterra 级数完整的表达形式为：

$$\tilde{y}(n)=\sum_{k=1}^{K}\sum_{i_1=0}^{\mathcal{M}}\cdots\sum_{i_p=0}^{\mathcal{M}}h_p\left(i_1,\cdots,i_p\right)\prod_{j=1}^{k}\tilde{x}\left(n-i_j\right) \tag{9}$$



(a) 列之间关联性　　(b) 行之间关联性

图 **6**　**W** 的左右矩阵关联图



(a) 列之间关联性　　(b) 行之间关联性

图 **7**　**W** 的上下矩阵关联图

其中，$\tilde{y}(n)$ 和 $\tilde{x}(n)$ 分别表示模型的的输入和输出信号，$h_p\left(i_1,\cdots,i_p\right)$ 表示 Volterra 核的对应的系数，$\mathcal{K}$ 表示非线性阶次，$\mathcal{M}$ 表示记忆深度。由于 Volterra 级数完整的表达形式复杂度过高，因此，提出了一种简化的 Volterra 级数模型，称为记忆多项式（Memory Polynomial，MP）模型：

$$\tilde{y}(n)=\sum_{j=0}^{\mathcal{M}}\sum_{i=1}^{\mathcal{N}}c_{ji}*\tilde{x}(n-j)*\left|\tilde{x}(n-j)\right|^{i-1} \tag{10}$$

记忆多项式模型的系数可以使用最小二乘法（Least Squaress Method，LS）来计算：

$$\mathrm{C_{LS}}=\left(\mathrm{X^{H}X}\right)^{-1}\mathrm{X^{H}Y} \tag{11}$$

其中 X 是模型输入数据矩阵，Y 是模型输出数据的矩阵。为了获得准确的系数估计，输入输出数据的样本点数需要根据经验法则选取。

记忆多项式模型广泛应用于行为建模和功率放大器（PA）的线性化处理中。在本题中，由上述矩阵间关联性分析可知，相关矩阵组 $\boldsymbol{H}$ 的同一行块的矩阵单元与相邻矩阵单元间存在强相关性而与距离较远的矩阵单元之间的相关性则很弱。

为了验证记忆多项式能否应用在本场景中，假设 $H_{j,k}$ 矩阵之间在 k 维度有时间关系，提取了每个 $H_{j,k}$ 中的相同位置的元素，如 $H_{1,1}$ 的 (1,1) 元素和 $H_{1,2}$ 的 (1,1) 元素，提取到 $H_{1,383}$ 的 (1,1) 元素，总共 383 个元素，然后从 $H_{1,2}$ 的 (1,1) 元素提取到 $H_{1,384}$ 的 383 个元素，2 者之间构建映射关系。提取出多组这样的 383 个的值对，提取的过程如下图所示。

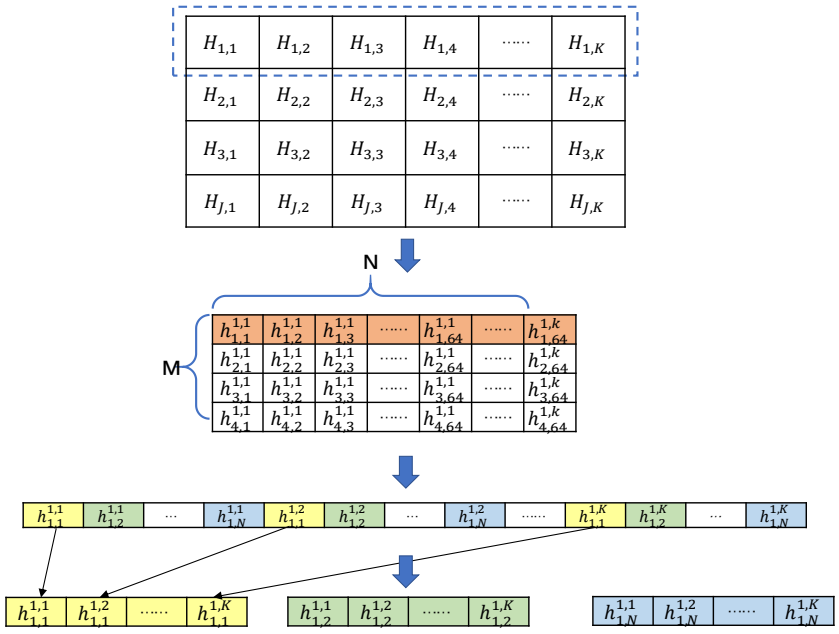
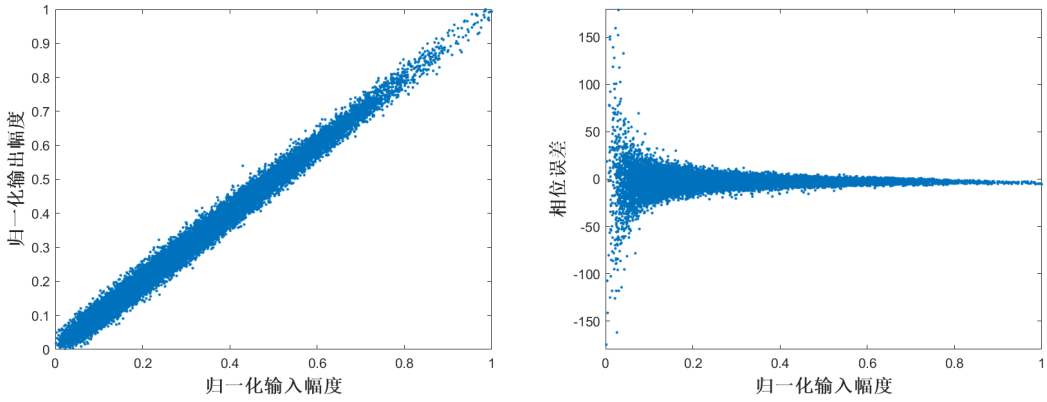


图 **8**　提取数据过程

绘制出幅度调制-幅度调制（Amplitude modulation- Amplitude modulation，AMAM）与幅度调制-相位调制（Amplitude modulation-Phase modulation，AMPM）图，如下图9所示。

因此，可以认为矩阵单元 $\boldsymbol{H}_{j,k+1}$ 可由其前几个矩阵 $\{\boldsymbol{H}_{j,k}, \boldsymbol{H}_{j,k-1}, \boldsymbol{H}_{j,k-1}, \cdots\}$ 表示。对应记忆多项式模型，即将 $\boldsymbol{H}_{j,k+1}$ 作为模型输出，将记忆项作为模型输入进行建模。

由于 $\boldsymbol{H}_{j,k}$ 为二维矩阵，无法直接代入记忆多项式模型进行求解，因此，如图所示，本文将矩阵组 $\boldsymbol{H}$ 中同一行块的全部矩阵单元 $\boldsymbol{H}_{j,k}$ 中的某一行提取出来



(a) AMAM 图　　(b) AMPM 图

图 **9**　给定数据排列方式的结果

进行记忆多项式建模。为了便于理解，以第一组矩阵的第一行块矩阵单元为例。由于矩阵间存在的相关性体现元素上时，是相邻矩阵中处于相同行列位置的元素间存在相关性，因此，如图所示，取出第一组矩阵的第一行块的全部矩阵单元的第一行的第一列的元素组成向量，即一个由 1×384 个元素组成的向量。取前 383 项定义为 $\boldsymbol{x} = \{\boldsymbol{H}_{1,1,1,1}, \boldsymbol{H}_{1,1,1,2}, ..., \boldsymbol{H}_{1,1,1,K-1}\}$，并将其作为记忆多项式建模的输入，将 $\boldsymbol{y} = \{\boldsymbol{H}_{1,1,1,2}, \boldsymbol{H}_{1,1,1,2}, ..., \boldsymbol{H}_{1,1,1,K}\}$ 作为建模的输出，取非线性阶次为 1（根据 MP 模型，实际最高有 2 次项），记忆深度为 8（经过计算，非线性阶数为 2，记忆深度为 8 时，建模效果最好），进行记忆多项式建模，求得一组系数 $\boldsymbol{c_1}$，其 matlab 代码见附录。对非线性阶次和记忆深度的确定如图10和11所示。

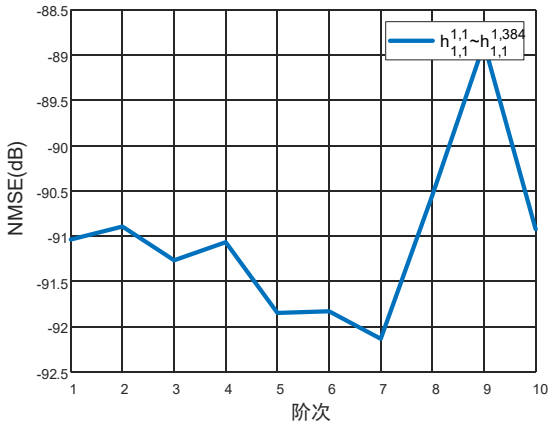


图 **10**　阶次对 **NMSE** 影响

由图可见，阶次对 NMSE 的影响不大，图9也可以看出，数据的非线性程度比较低，因此为了降低复杂度和计算资源，取 MP 模型的阶次为 1。从记忆项对

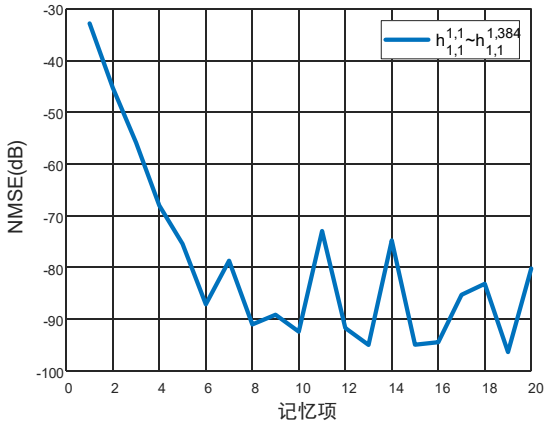


图 **11**　记忆项对 **NMSE** 影响

NMSE 影响图可见，在记忆项为 8 的时候，性能接近瓶颈，虽然取 10 项能带来些许改善，但相应会增加很大的复杂度。

得到系数 $\boldsymbol{c_1}$ 后，若以向量 $\boldsymbol{x}$ 中每个元素的前 9 个元素作为输入（根据记忆项的定义当前元素的前一个元素不是记忆项，因此记忆深度为 8 时，应选择前面 9 个元素作为输入），以 $\boldsymbol{c_1}$ 作为建模系数，对此元素进行 MP 建模估计，最终可得向量中所有元素的估计 $\hat{\boldsymbol{x}}$，$\hat{\boldsymbol{x}}$ 与 $\boldsymbol{x}$ 之间的 NMSE 可达-90dB。NMSE(dB) 定义为：

$$NMSE = 10\lg\frac{\sum_{n=1}^{N}|\widehat{\boldsymbol{x}(n)}-\boldsymbol{x}(n)|^2}{\sum_{n=1}^{N}|\boldsymbol{x}(n)|^2} \tag{12}$$

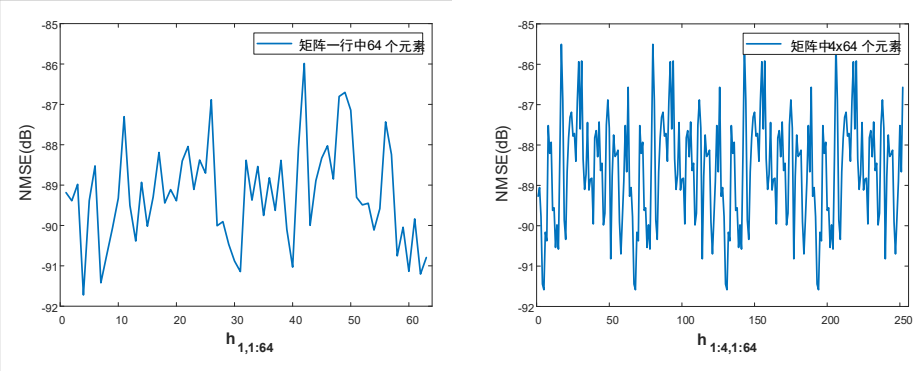
由于前述分析中，仅能确定同一行块的所有矩阵的首个元素间存在相关性。为了新一步确定矩阵间的相关性，接着分析是否能将这组系数应用于所取矩阵单元的所有元素。取出矩阵单元中相同位置的元素，例如，取出每个矩阵的第一行的第二列的元素组成向量，以每个元素的前 9 项对当前进行建模估计，可得向量中所有元素的估计值。由于矩阵单元的列维数为 64，所以重复上述操作 64 次，每组数据估计值与实际值间的 NMSE 如图12所示。

由图可知，每组数据估计值与实际值间误差很小。综合上述分析过程可得以下两个结论：

（1）若有与当前矩阵单元处于同一行块的前 9 个矩阵单元的精确值，则可通过记忆多项式模型准确估计此矩阵单元。

（2）建模时，同一行块的所有矩阵元素的估计都可采用同一组系数。

基于上述结论，即可进行压缩函数 $P_1(\cdot)$ 和解压缩函数 $G_1(\cdot)$ 的建模。以第一组矩阵的第一行块的 384 个矩阵单元为例。将压缩过程定义为：只存储 $\boldsymbol{x}$ 的前 9 项和前述参数提取过程得到的一组系数 $\boldsymbol{c_1}$，将 $\boldsymbol{x}$ 中从第 10 项开始的所有元素



(a) 单个元素标定的 MP 系数用于整行元素　(b) 单个元素标定的 MP 系数用于整个矩阵

图 **12**　单个元素标定的 **MP** 系数泛化结果

均丢弃。读取存储数据，将保留的前 9 项元素作为输入，将 $\boldsymbol{c_1}$ 作为模型系数代入 MP 模型中，对第 10 个元素进行建模估计，得到第 10 个元素的估计值后，再以第 2-9 个元素和第 10 个元素的估计值依照同样的方式建模估计第 11 个元素，依次类推，得到余下丢弃的 375 个元素的估计值，即完成解压缩。

计算解压缩后的数据 $\hat{\boldsymbol{x}}$ 与原数据 $\boldsymbol{x}$ 间的 NMSE 值，发现解压后的数据误差较大。究其原因，在于采用估计值作为输入来估计下一个元素会导致误差累积，由 9 个元素推出 375 个元素的过程中误差累积过高。因此，考虑降低压缩比，保留向量 $\boldsymbol{x}$ 中近一半的数据，即按照保留 9 项，丢弃 8 项的规律（即每 17 个矩阵，保留前 9 个，丢弃后 8 个）对该向量进行压缩处理，存储处理后的数据 $P_1(\boldsymbol{x})$ 与 $\boldsymbol{c_1}$。读取存储数据，并依据保留的数据采用上述 MP 模型建模估计出所有丢弃的数据，完成数据的解压缩处理。

经计算验证，在第一组矩阵中采取上述压缩、解压缩模型，对矩阵组 $\boldsymbol{H}$ 以 9/17 的压缩比进行压缩、解压缩处理，解压后的数据 $\hat{\boldsymbol{x}}$ 与原数据间的 NMSE 很小。将此压缩、解压缩函数以及上述提取的模型参数 $\boldsymbol{c_1}$ 应用于整个第一组矩阵，解压后的数据为 $\widehat{\boldsymbol{H}} = G_1\left(P_1(\boldsymbol{H})\right)$，解压缩后的数据和原数据间的误差为 $err_H = -31.4$，满足题目的要求。

将上述建模过程应用于其余 5 个矩阵组，实际发现并不能达到同样的-31.4 的误差，因此进行了进一步的实验。在达不到-30dB 的误差时，可以将由 MP 模型推导出来的矩阵数量减少，从 9 个矩阵推出 8 个矩阵，减少至 9 个矩阵推出 7 个矩阵，到推出 6 个矩阵，以此类推。按照记忆多项式的理论和前文的推导，由前矩阵退出来的矩阵数量越少，准确度越高，但压缩程度也会越小。经过实验，得到下图13。可见结果符合预期，对于不同的输入矩阵 H，可以使用不同的配置方案，来使误差达到-30dB 以下，实现自适应动态改变的效果。针对不同组别数

据、其使用的记忆项、使用 M 个数据估计的数据量 L、解压缩数据的误差、压缩率、计算复杂度和存储复杂度如下表所示。

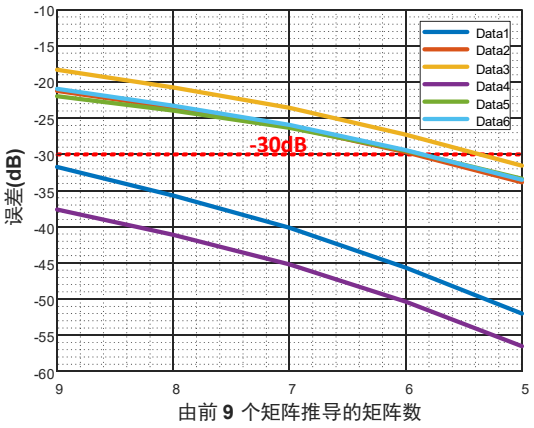


图 **13**　推导矩阵数与计算误差关系图

表 **3**　不同矩阵组建模数据

数据组	M	L	误差	压缩率 (%)	计算复杂度	存储复杂度
第一组	8	9	-31.7333	50.78	8.22×10^7	1.28×10^7
第二组	8	5	-33.8574	64.84	1.05×10^8	1.63×10^7
第三组	8	5	-31.561	64.84	1.05×10^8	1.63×10^7
第四组	8	9	-37.6122	50.78	8.22×10^7	1.28×10^7
第五组	8	5	-33.3686	64.84	1.05×10^8	1.63×10^7
第六组	8	5	-33.5098	64.84	1.05×10^8	1.63×10^7

5.3.2 主成分分析建模

主成分分析（PCA）是机器学习领域最常用的降维方法之一，主要思想是 n 维特征按照方差的降序顺序找到一组 n 维正交基，其中前 k 个特性包含绝大部分的方差，后面的特征所含的方差几乎为 0。这样通过这种方式，后 n-k 个特征包含的方差基本为 0，忽略后 n-k 个特征就实现了数据特征的降维。在对 W 的相关性分析中，可以看到相关矩阵与 H 的特征相差很大，并且不具备距离远的矩阵相关性越小的特征，因此选择使用 PCA 方法来提取特征。

PCA 算法流程

Step1：设有 m 个 n 维向量，将 m 个 n 维向量组成一个 $n \times m$ 维的矩阵 $\boldsymbol{X}$

Step2: 将 $\boldsymbol{X}$ 的每一行进行零均值化，即减去这一行的均值

Step3: 求出协方差矩阵 $\boldsymbol{C} = \frac{1}{m} X X^T$

Step4: 求出协方差矩阵 $\boldsymbol{C}$ 的特征值及对应的特征向量

Step5: 将特征向量按对应特征值大小从左到右按列排列成矩阵，取前 k 列组成矩阵 $\boldsymbol{P}$

Step6: $\boldsymbol{Y} = \boldsymbol{P}^T X$ 即为降维到 k 维后的数据

采用主成分分析对最终的输出矩阵 $\boldsymbol{W}$ 进行压缩和解压缩建模，六组数据上的建模压缩比和解压缩后数据的误差值如下表所示：

表 **4**　不同矩阵组建模数据

数据组	误差	压缩率（(%)
第一组	-30.5925	80.4
第二组	-30.0558	95.25
第三组	/	/
第四组	-30.977	43.29
第五组	-30.5568	61.85
第六组	-30.8779	68.03

5.4 存储复杂度、计算复杂度分析

下面先分析记忆多项式运算的复杂度。如果不利用矩阵间的关联性，需要对相关矩阵组 $H = H_(j,k)$ 中每一个矩阵进行 SVD 分解，总计进行 J×K 次 SVD 分解，计算量将变得十分庞大。根据 1.2 中对矩阵间的关联性分析，可知对 H 中任意 $H_(j,k)$，它总与在同一个行块内的前后几个矩阵高度相关，与相距较远的矩阵相关性很小，并且不考虑同一列块不同行块之间关联性。类比于 MIMO 天线阵列，K 表示连续的时间位点，同一列块不同行块之间的矩阵可理解为 J 个独立的天线阵列，$H_(j,k)$ 相当于第 j 个天线阵列（每个阵列含有 M×N 个阵元）在第 k 个时间点时的状态。考虑到实际使用场景下的时间连续性，$H_(j,k)$ 与前后几个时间点的有较大的关联性，可根据前几个时间点的状态推测下一个时间点的状态。

令记忆多项式的最高阶数为2阶，即 $N=1$。保留 $H_(j,(M+1)n+1),\ldots,H_(j,(M+1)n+M)$ 作为记忆项，$H_{(j,(M+1)(n+1)+1)}, n\geq 0$ 可以由记忆项推出，所以每隔 $M+1$，在解压缩的过程中再还原出来。通过这种方式可以将相关矩阵组 $H=H_{j,k}$ 中K列简化为 $[(M+1)K/(M+L+1)]+mod(K,M+L+1)$ 列，其中第一项为 $(M+1)K/(M+L+1)$ 向下取整，第二项为 K 除以 $M+L+1$ 的余数，压缩的过程不涉及计算复杂度。在解压缩过程中，被还原的每一个 $H_(j,k)$ 的每一个元素 $H_(j,k,m,n)$ 都满足上式，那么求出每一个元素需 $3(M+1)$ 次复数乘法和 $M+1$ 次复数加法，还原出完整矩阵组总计需要 $3(M+1)\times JMN\times([(M+1)K/(M+L+1)]+mod(K,M+L+1))$ 次复数乘法和 $(2M+1)\times JMN\times([(M+1)K/(M+L+1)]+mod(K,M+L+1))$ 次复数加法。这虽然带来了计算复杂度上的增加，但是仅需存储MP系数矩阵和一半的矩阵组，几乎减少了一半的存储复杂度。与计算整个相关矩阵组H的SVD分解相比，减少了一半的SVD求解过程，大大降低了计算复杂度。

6. 问题三：低复杂度计算和存储

基于给定的所有矩阵数据 $\boldsymbol{H}$，分析其数据间的关联性，在满足 $\rho_{\min}(\boldsymbol{V})\geq\rho_{th}=0.99$ 的情况下，设计低复杂度计算和存储的整体方案，完成从矩阵输入信号 $\boldsymbol{H}$ 到近似矩阵输出信号 $\boldsymbol{W}$^的端到端流程，如下图所示

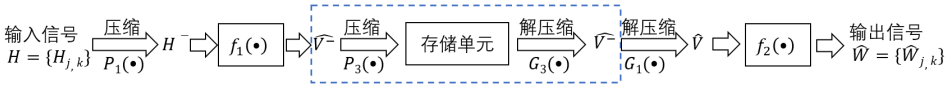


图 **14**　问题 **3** 流程图

对端到端整体流程进行简要说明：1. 对输入信号 $\boldsymbol{H}=\boldsymbol{H}_{j,k}$ 进行压缩。采用同问题二的压缩方法，将K列矩阵压缩为 $(M+1K)/(M+L+1)$ 列矩阵 $\widehat{\boldsymbol{H}}$，再通过SVD分解计算得到 $\widehat{\boldsymbol{V}}$。由于SVD分解并不会对矩阵之间的相关性产生影响，所以通过记忆多项式解压缩 $\boldsymbol{H}$ 的函数 $G1(\bullet)$ 同样适用于解压缩 $\widehat{\boldsymbol{V}}$ 这样做虽然无法减少SVD分解的计算复杂度，但是减少了需要进行SVD分解的矩阵的个数，因此大大减少了整个矩阵组的SVD分解的计算复杂度。对SVD分解计算得到 $\widehat{\boldsymbol{V}}$ 相较于传统SVD的计算复杂度，rSVD在计算复杂度上有数量级上的优势，所以采用rSVD算法将 $\boldsymbol{H}$ 分解 $\boldsymbol{V}$ 对分解后的奇异值矩阵取最大的前两个值可将每个矩阵 $\boldsymbol{V}_(j,k)$^简化为L×N阶矩阵。2. 对 $\widehat{\boldsymbol{V}}$ 进行压缩和解压缩。当信号 $H=H_(j,k)$ 输入时，往往无法不经存储地通过运算得到信号 $W=W_(j,k)$ 输出。数据通常会压缩后存放在存储单元内，等到需要使用该数据时，再通过解压

缩和一系列运算，得到输出信号。如果需要输入信号 $H = H_(j,k)$ 根据即时输出 $W = W_(j,k)$，可以跳过存储步骤。3. V 通过公式 $W_k = V_k(V_k^H V_k + {}^2I)^(-1)$ 计算得到 $W\square$ 可以结合 Coppersmith–Winograd 方法求逆矩阵，相比于采用高斯消元法的复杂度近似为 $O(N^3)$，采用该方法的复杂度为 $O(N^2.376)$, 当 N 的值较大时，Coppersmith–Winograd 方法的复杂度远小于高斯消元法。

参考文献

[1] Golub, Gene, and William Kahan，Calculating the singular values and pseudo-inverse of a matrix，Journal of the Society for Industrial and Applied Mathematics，Series B: Numerical Analysis 2.2 ： 205-224，1965。

[2] [5] Halko, Nathan, Per-Gunnar Martinsson, and Joel A. Tropp，Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions，SIAM review 53.2 ： 217-288，2011。

附录 A 我的 MATLAB 源程序

```
%%
%% 读入数据

clear
H = [];
V = [];
W = [];
for i = 1:6
eval(['H',num2str(i),' =
    load("./Data/Data',num2str(i),'_H.mat");'])
eval(['V',num2str(i),' =
    load("./Data/Data',num2str(i),'_V.mat");'])
eval(['W',num2str(i),' =
    load("./Data/Data',num2str(i),'_W.mat");'])
eval(['H = [H; H', num2str(i),'.H];'])
eval(['V = [V; V', num2str(i),'.V];'])
eval(['W = [W; W', num2str(i),'.W];'])
end
H_all = H;
V_all = V;
W_all = W;
clear H V W H1 H2 H3 H4 H5 H6 V1 V2 V3 V4 V5 V6 W1 W2 W3 W4 W5 W6
%% 选择数据
%
    为了方便处理，下面只处理Data1数据，要想改变可以通过参数data_sel来选择是Data1到6的任意一组

data_sel = 6;
H = H_all((data_sel-1)*4+1:(data_sel-1)*4+4,:,:,:);
V = V_all((data_sel-1)*4+1:(data_sel-1)*4+4,:,:,:);
W = W_all((data_sel-1)*64+1:(data_sel-1)*64+64,:,:,:);


%% 提取MP系数
% 数据量太少导致，但提取多个元素的cc再求平均不能解决问题

cc_result = [];
cc = [];
```

```
n_order = 1;
m_memory = 8; %不要轻易改记忆项，要改动的很多
for j = 1:4
H_pre = [];
H_post = [];
for i = 1:383
H_pre = cat(2,H_pre,H(1,:,j,i));
H_post = cat(2,H_post,H(1,:,j,i+1));
end

H_input = [];
H_output = [];

for i = 1:383
H_input = [H_input H_pre((i-1)*64+1)];
H_output = [H_output H_post((i-1)*64+1)];
end
[cc_o,n,yo,MM] =
   MP_e(H_input(1:383),H_output(1:383),n_order,m_memory);
cc_result = [cc_result NMSE_YU(H_output(28:383), yo(20:end))];
cc(:,j) = cc_o;
end
[min_nmse min_index] = min(cc_result);
cc1 = cc(:,1);
cc2 = cc(:,2);
cc3 = cc(:,3);
cc4 = cc(:,4);
save('cc_temp.mat',"cc","cc1","cc2","cc3","cc4");

%% 压缩H数据

reserve_index = [];
reserve_num = 9; %保留的矩阵数
mp_num = reserve_num + 5;
   %9个矩阵推广出的矩阵数，此值越小，存储占比越高，但性能越好
%需要保留的矩阵
for k = 1:40
reserve_index = [reserve_index mp_num*(k-1)+1:mp_num*(k-1)+9];
end
```

```
H_compress = [];
for i = 1:384
if ismember(i, reserve_index)
H_compress = cat(4,H_compress,H(:,:,:,i));
end
end


%压缩后的数据是是4x64x4x193的，第四维192是从每行384组矩阵里抽取了一半，再加第9列矩阵
save("H_compress.mat","H_compress");
%% 解压缩H数据
% 使用同一组（组是j维度）系数和使用不同组结果近似，故只用一组
%
% 9组推8组但只新增8组时，误差会积累，因此需要9组推9组才行，但比8推8要更差1dB
%
% 解决方案是9推8组并新增9组，压缩率变成8/17

load('cc_temp.mat');
% load('cc-28.mat');
load("H_compress.mat");

H1 = [];
H2 = [];
x_input = [];
for i = 1:reserve_num
H1 = cat(4,H1,H_compress(:,:,:,i));
end
%index是H_compress的起始值，前8个已经被加进去了
index = reserve_num+1;

%下面的循环变量与附录里的一致
for j = reserve_num+1:384
if ~ismember(j,reserve_index)
for k = 1:4
for n = 1:64
for m = 1:4
```

```
for l = 1:reserve_num
x_input = [H1(m,n,k,j-l) x_input];
end
%                 eval(['[H_temp max_input] =
   MP_v(x_input,cc',num2str(min_index),',' , 2,8);']);
   %使用不同系数不如使用效果最好的
[H_temp max_input] =
   MP_v(x_input,cc(:,min_index),n_order,m_memory);
%                 if k == 1
%                    H2(m,n,k) =
   H_temp(reserve_num)*max_input/0.9675;
%                 else
H2(m,n,k) = H_temp(reserve_num)*max_input;
%                 end
x_input = [];
end
end
end
else
H2 = squeeze(H_compress(:,:,:,index));
index = index + 1;
end
H1 = cat(4,H1,H2);
end
H_decompress = H1;

save('H_decompress.mat',"H_decompress");
%% H解压缩后的error

load("H_decompress.mat");
num_of_complex = numel(H_compress);
compress_ratio = num_of_complex/numel(H)
error = err_cal(H_decompress, H)

%% PCA法验证H的误差



H_pca = reshape(H,32,[]);
mu = mean(H_pca);
```

```
[vectors, scores] = pca(H_pca);

%64维度 pca后 总共有63个分量，取了61个才达到-30dB，压缩比为61/64 = 95%
%128维度 pca后 总共127个分量，取了100个达到-30dB，压缩比为78%
%512维度 pca后 总共511个分量，取了100个达到-30dB，压缩比为20%
nComp = 120;
H_compress = vectors(:,1:nComp);
Xhat = scores(:,1:nComp) * vectors(:,1:nComp)';
%实际的存储度是scores加上vectors的复数数量
num_of_complex = numel(scores) + numel(vectors);
compress_ratio = num_of_complex/numel(H)
Hhat = bsxfun(@plus, Xhat, mu);

Hhat = reshape(Hhat,4,64,4,384);

error = err_cal(Hhat,H)
%% 压缩解压缩W数据（PCA）以及误差计算

W_pca = reshape(W,1024,[]);
mu = mean(W_pca);
[vectors, scores] = pca(W_pca);

%pca后 总共有63个分量，取了59个才达到-30dB，压缩比为59/64
nComp = 110;
W_compress = vectors(:,1:nComp);
Xhat = scores(:,1:nComp) * vectors(:,1:nComp)';
What = bsxfun(@plus, Xhat, mu);

What = reshape(What,64,2,4,384);

num_of_complex = numel(W_compress) + numel(scores(:,1:nComp));
compress_ratio = num_of_complex/numel(W)
error = err_cal(What,W)
%%

function [rou, rou_min] = rou_4_384(W_hat,W)
%计算 任意行x任意列x4x384的矩阵的W的rou
```

```
rou = [];
rou_min = [];
for j = 1:4
for k = 1:384
[rou_temp rou_min_temp] = rou_cal(W_hat(:,:,j,k),W(:,:,j,k));
rou = [rou rou_temp];
rou_min = [rou_min rou_min_temp];
end
end

end




function [rou,rou_min] = rou_cal(W_hat, W);
% W_hat需要是64x2的矩阵才行


[row col] = size(W_hat);
[row1 col1] = size(W);
if row ~=row1 || col~=col1
error('2个矩阵维度不一致！');
end

if row < col
W_hat = W_hat.';
W = W.';
tmp=col;
col=row;
row=tmp;
end


rou = zeros(1,col);
for i = 1:col
W_tmp = W_hat(:,i)' * W(:,i);
rou(i) =
```

```
    abs(W_tmp)/(sqrt(W_hat(:,i)'*W_hat(:,i))*sqrt(W(:,i)'*W(:,i)));
end

%
    注意这只是在L层面的最小值，在J和K层面的需要调用时用循环再从所有rou_min里找个最小值
rou_min = min(rou);




\end{rou_cal}



clear;clc
A=[1 3;7 5];
B=[6 8;4 2];
C1=A*B
C1=strassen(A,B)

A=[45 12 32 12 ;52 32 10 54;20 43 20 1;15 96 03 4];
B=[96 25 34 62 ;38 10 5  41;13 62 2 17;10 62 19 27];
C2=A*B
C2=strassen(A,B)

function [C] = strassen(A,B)

n=length(A);
if n==1
C=A*B;
else

S1=B(1:n/2,n/2+1:n)-B(n/2+1:n,n/2+1:n);
S2=A(1:n/2,1:n/2)+A(1:n/2,n/2+1:n);
S3=A(n/2+1:n,1:n/2)+A(n/2+1:n,n/2+1:n);
S4=B(n/2+1:n,1:n/2)-B(1:n/2,1:n/2);
S5=A(1:n/2,1:n/2)+A(n/2+1:n,n/2+1:n);
S6=B(1:n/2,1:n/2)+B(n/2+1:n,n/2+1:n);
S7=A(1:n/2,n/2+1:n)-A(n/2+1:n,n/2+1:n);
```

```
S8=B(n/2+1:n,1:n/2)+B(n/2+1:n,n/2+1:n);
S9=A(1:n/2,1:n/2)-A(n/2+1:n,1:n/2);
S10=B(1:n/2,1:n/2)+B(1:n/2,n/2+1:n);

P1=strassen(A(1:n/2,1:n/2),S1);
P2=strassen(S2,B(n/2+1:n,n/2+1:n));
P3=strassen(S3,B(1:n/2,1:n/2));
P4=strassen(A(n/2+1:n,n/2+1:n),S4);
P5=strassen(S5,S6);
P6=strassen(S7,S8);
P7=strassen(S9,S10);

C(1:n/2,1:n/2)=P5+P4-P2+P6;
C(1:n/2,n/2+1:n)=P1+P2;
C(n/2+1:n,1:n/2)=P3+P4;
C(n/2+1:n,n/2+1:n)=P5+P1-P3-P7;
end
end

\end{strassen}



function [U,S,V] = rsvd(A,K)

[M,N] = size(A);
P = min(2*K,N);
X = randn(N,P);
Y = A*X;
W1 = orth(Y);
B = W1'*A;
[W2,S,V] = svd(B,'econ');
U = W1*W2;
K=min(K,size(U,2));
U = U(:,1:K);
S = S(1:K,1:K);
V=V(:,1:K);

\end{rsvd}
```

```
function [error] = err_cal(W_hat,W)
%求error函数，输入需要是 任意行x任意列x4x384 的矩阵

fenzi = 0;
fenmu = 0;
for j = 1:4
for k = 1:384
fenzi = fenzi + (norm(W_hat(:,:,j,k)-W(:,:,j,k),'fro'))^2;
fenmu = fenmu + (norm(W(:,:,j,k),'fro'))^2;
end
end

fenzi = fenzi/(384*4);
fenmu = fenmu/(384*4);
error = 10*log10(fenzi/fenmu);

end
```