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	**21102860059**
队员姓名	**1.** 曹苇杭
	2. 丁明远
	3. 田宇

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

相关矩阵组在无线通信、雷达、图像/视频处理等领域都有广泛的应用，其特点在于数据之间具有较强的相关性。数据维数的不断增长，使得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。

对于问题一，本文从每个子矩阵的自相关性出发，基于随机 SVD 分解方法对目标矩阵降维，并使用基于双对角化和 QR 分解的 SVD 分解计算 **V**。之后本文基于 AOR 迭代算法，从子矩阵自身的相关性出发，训练出合适的迭代因子，使得计算 **W** 时矩阵求逆的迭代次数大大减少。基于 Strassen 算法，对算法中涉及的矩阵运算进行分治计算，从而进一步降低计算复杂度。最后，本文从矩阵行块间的相关性出发，构建相邻子矩阵的互相关函数模型，建立互相关函数与插值算法的关系，给出最优插值系数。在矩阵 **V** 和矩阵 **W** 的计算中，理论上可以减少约 40% 的计算量。

对于问题二，本文提出一种基于 SVD 分解的相关矩阵组压缩算法，通过对矩阵重新排列、合并、SVD 分解并提取最大奇异值对应的列向量，可以实现对矩阵组数据的有效压缩。本文基于 SVD 分解对矩阵间相关性进行分析，验证了该压缩/解压缩算法的可行性。本文进一步推导对这种方法的存储复杂度、压缩复杂度以及解压复杂度，并建立多目标优化模型，求解最优的压缩参数。最优解的压缩率可达 0.3867，且具有较低的压缩、解压复杂度。

对于问题三，本文提出了一种联合优化策略，将问题一的中间变量存储复杂度和插值存储复杂度纳入到优化方案当中进行讨论。同时我们将迭代算法的结构调整为逐元素运算的形式，避免了部分矩阵存储的复杂度。

关键字：相关矩阵组　SVD 分解　AOR 迭代　QR 分解

目录

1. 问题重述

1.1 引言

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通常呈现为矩阵的形式，这一系列的矩阵间通常在某些维度存在一定的关联性，因此数学上可用相关矩阵组表示。例如，视频信号中的单帧图像可视为一个矩阵，连续的多帧图像组成了相关矩阵组，而相邻图像帧或图像帧内像素间的关联性则反映在矩阵间的相关性上。随着成像传感器数量/雷达阵列/通信阵列的持续扩大，常规处理算法对计算和存储的需求成倍增长，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 问题的提出

1.2.1 问题一：相关矩阵组的低复杂度计算

本问题明确提到利用矩阵相关性在满足建模精度的前提下，尽可能减少计算复杂度。因此在建模 $\hat{\mathbf{V}} = f_1(\mathbf{H})$ 和 $\widehat{\mathbf{W}} = f_2(\widehat{\mathbf{V}})$ 时首先应当考虑的时建模精度的问题，也就是 SVD 分解的精度和矩阵求逆运算的精度。目前有许多数学上的方法提供了迭代求解 SVD 算法和矩阵求逆的可能性，这些算法往往采用迭代的形式，可以根据精度要求调节迭代次数。题目要求的建模精度为 0.99，并不是很高，迭代算法的应用能够大大减少运算的复杂度。

此外，可以从矩阵相关性的角度考虑减少运算，尽量避免遍历计算每个矩阵。因为同一个行块的矩阵具有较高的相关性，可以通过模式识别或者插值的算法根据已经计算的矩阵估计出剩余部分矩阵，但是这样的估计显然是误差较大的，因为矩阵间除了有相关性还包括了随机性，我们只能估计出目标子矩阵和其他子矩阵相关的成分，但是无法估计出目标子矩阵的随机特性，这样的随机特性满足一个特定的分布，如果矩阵间的相关性较高，仍然可以在满足精度的前提下进行插值，而判断哪些矩阵满足插值的条件，以及选择合适的插值方式成为本小题的难点。

本题还给出了一种思路：从模式识别的角度，出发拟合 $\hat{\mathbf{V}} = f_1(\mathbf{H})$ 和 $\widehat{\mathbf{W}} = f_2(\widehat{\mathbf{V}})$ 的关系，但是对 6 组数据进行分析，我们发现行块与行块之间是相互独立的，并且由于数据量不足，我们很难建立一个合适的模型来拟合矩阵的之间的相关性。

从矩阵相关性的角度，我们发现相邻子矩阵之间的相关性最强，因此我们考虑从建立子矩阵互相关函数的角度出发，找到 $\mathbf{V}_{j,k}$ 和 $\mathbf{W}_{j,k}$ 与 $\mathbf{H}_{j,k}$ 互相关矩阵

$\mathbf{R}_{j,k}$ 之间的关系，从而减少 $\mathbf{V}_{j,k}$ 和 $\mathbf{W}_{j,k}$ 的计算数量。

1.2.2 问题二：相关矩阵组的低复杂度存储

本题要求基于给定的所有矩阵数据 **H** 和 **W**，分析各自数据间的关联性，分别设计相应的压缩模型 $P_1(\cdot)$、$P_2(\cdot)$ 和解压缩模型 $G_1(\cdot)$、$G_2(\cdot)$，在满足误差条件 $Err_H \leq -30$dB、$Err_W \leq -30$dB 的情况下，使得存储复杂度、压缩与解压缩的计算复杂度最低。

题目中所给的矩阵阵列可以与视频流进行类比。每一个矩阵都相当于视频中的一帧图像，每帧内部存在一定的相关性，而相同 j 下标、不同 k 下标的矩阵之间的相关性则可以理解为帧之间的相关性。因此，可以从已有图像、视频压缩算法中得到一定的启发。但在考察各种变换域算法后，我们发现传统的 FFT、DCT 变换等方法对处理本题的数据并不具有优越性。因此，如何利用相关特性，选择合适的方式去除数据中的冗余信息是本题的难点。此外，很容易想到，问题一中分析得到的相关性，也可以应用于问题二中，即低复杂度计算的方法同样可以应用于低复杂度存储。通过将行块中的矩阵合并并进行 SVD 分解，我们发现构造的新矩阵具有较高的条件数，即矩阵的信息主要集中在最前面的几个奇异值中。通过舍弃对结果恢复影响不大的奇异值，我们可以实现对相关矩阵组的有效压缩。

在完成压缩方法的框架搭建后，本题转化为一个多目标优化问题。由于题目要求两个建模优化目标（存储复杂度，压缩与解压缩的计算复杂度）的优先级相同，这就是希望我们使用对多目标加权的方法，将多目标优化转化为单目标优化。

1.2.3 问题三：相关矩阵组的低复杂度计算和存储

利用问题一中的方案计算 V 矩阵时，很多内部变量需要进行存储，比如 SVD 分解中的 QR 迭代过程需要为迭代中的中间变量开辟存储空间，SVD 分解中求 Q 矩阵也需要存储复杂度，但是由于每次迭代的结果会覆盖上次迭代的结果，所以消耗的存储复杂度不会随着迭代次数的增加而增加，因此很多中间变量存储简单的传统算法将重新进行考虑，这意味着我们需要考虑算法的存储复杂度，结合计算复杂度进行联合优化。

W 矩阵之前也需要对 J 个 V 矩阵进行合并求逆，需要的存储复杂度为 64NLJ，AOR 迭代中，每次迭代的结果也要进行存储。但是由于 AOR 的中间变量的维度只有 64NL，因此将 W 矩阵的计算拆分成维度 $N \times L$ 的子矩阵运算具有较少的存储复杂度。

问题三最终可以转换成存储复杂度和计算复杂度联合优化的问题，甚至需

要重新考虑一些存储复杂度低的算法，也可以从算法结构的角度进行优化，将迭代算法写成逐个元素优化的算法结构，尽可能在计算复杂度不变的而基础上减少存储复杂度。

1.3 思维导图

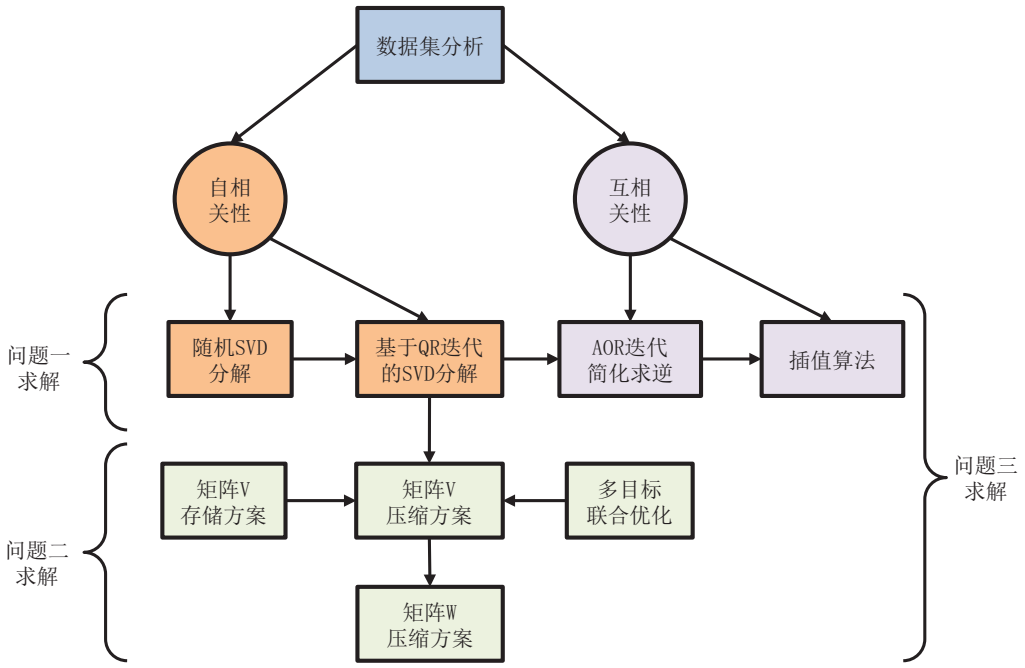

图 1　相关矩阵组的低复杂度计算和存储建模的问题解决思维导图

2. 模型假设

- 对于同一数据集的同一行块中的矩阵满足相同分布，具有相同的统计特性；
- 不同数据集之间、同一数据集不同行块之间的分布彼此独立；
- 同一个行块内的矩阵之间具有相关性，且矩阵间的距离越近，其相关性越强。

3. 符号说明

符号	意义
$\mathbf{H}$	相关矩阵组
$\mathbf{H_{j,k}}$	相关矩阵组中第 j 行 k 列的矩阵
M	相关矩阵组中每个矩阵的行数
N	相关矩阵组中每个矩阵的列数
J	相关矩阵组的行数
K	相关矩阵组的列数
ρ_{min}	**W** 的最低建模精度
err_H	**H** 的压缩误差
err_W	**W** 的压缩误差
$\mathbf{V}$	右奇异矩阵
SVD	奇异值分解
SOR	超松弛
AOR	加速超松弛

4. 问题一求解

4.1 子问题 1——矩阵 V 的近似低复杂度计算

4.1.1 基于随机 SVD 算法的目标矩阵降维

我们首先考虑对某个一般的复数矩阵 $\mathbf{A} \in \mathbb{C}^{n\times m}$ 进行 SVD 分解，其中 $n \geq m$，目前采用较多的算法有基于 Household 变换的 SVD 分解、基于 Givens 变换和 Jacobi 旋转的 SVD 算法以及基于 Golub-Kahan 双对角化的 SVD 算法。通过考察这一系列算法，我们得出以下结论：

- Household 变换的矩阵维度取决于 n，但是本题数据集中的 $n = 64$ 较大，乘法运算次数过多。
- Givens 变换每次相乘的是一个近似对角阵，乘法复杂度较小，但是 Givens 变换的角度 θ 求解需要消耗大量计算复杂度，特别地，对于复数矩阵的 Givens 变换，需要在实数变换的 Givens 矩阵基础上补偿辅角 α，复杂度也较高。
- 如果精度要求过高或者矩阵维度过大，Golub-Kahan 算法则无法满足要求，无法达到题目要求的 $\rho_{min} = \mathbf{min}\left\{\rho_{l,j,k}(\mathbf{V})\right\} \geq 0.99$，故无法采用。

$$\rho_{l,j,k}(\mathbf{V}) = \frac{\left\|\mathbf{V}_{l,j,k}^{H}\mathbf{V}_{l,j,k}\right\|_2}{\left\|\mathbf{V}_{l,j,k}\right\|_2\left\|\mathbf{V}_{l,j,k}\right\|_2}, l = 1,\ldots,L \tag{1}$$

因此我们最终考虑先通过降维的算法，在不改变奇异值的前提下，将复数矩阵 $\mathbf{A} \in \mathbb{C}^{n\times m}$ 降维得到 $\mathbf{B} \in \mathbb{C}^{m\times m}$。之后，采用基于双对角化和 QR 分解的迭代算法求解 **B** 的 SVD 分解，通过控制迭代结束的误差条件，可以有效控制计算复杂度。由于求解 $\mathbf{W}_k$ 的过程中会引入一定的误差，因此该问题可以转化为多目标规划问题。具体的优化方案，我们会在问题一求解的最后进行说明。

B 矩阵的构建主要通过基于随机 SVD 的 SVD 分解算法实现，对于本题的任一矩阵 $\mathbf{H_{j,k}} \in \mathbb{C}^{M\times N}$，令 $\mathbf{A} = \mathbf{H_{j,k}}^{H} \in \mathbb{C}^{N\times M}$。该算法主要分为两步：第一步构造 m 个标准正交列向量矩阵 $\mathbf{Q} \in \mathbb{C}^{N\times M}$；第二步计算维度为 $m \times m$ 的矩阵 $\mathbf{B} = \mathbf{Q}^{H}\mathbf{A} = \mathbf{Q}^{H}\mathbf{H_{j,k}}^{H}$ 的 SVD 分解：

$$\mathbf{B} = \widetilde{\mathbf{U}}\mathbf{S}\widetilde{\mathbf{V}}^{H} \tag{2}$$

对 **B** 求共轭转置，并右乘 $\mathbf{Q}^{H}$ 可得：

$$\mathbf{B}^{H}\mathbf{Q}^{H} = \mathbf{H_{j,k}}\mathbf{Q}\mathbf{Q}^{H} = \widetilde{\mathbf{V}}\mathbf{S}\widetilde{\mathbf{U}}^{H}\mathbf{Q}^{H} = \widetilde{\mathbf{V}}\mathbf{S}\left(\mathbf{Q}\widetilde{\mathbf{U}}\right)^{H} \approx \mathbf{H_{j,k}} \tag{3}$$

因此当 $\|\mathbf{H_{j,k}} - \mathbf{H_{j,k}}\mathbf{Q}\mathbf{Q}^{H}\| < \epsilon$ 时，其中 ϵ 为满足条件的某一小量，可以认为 $\mathbf{Q}\widetilde{\mathbf{U}} \approx \mathbf{V}$，其中 **V** 表示矩阵 **H** 的右奇异向量。如果对每一个矩阵 $\mathbf{H_{j,k}}, j = 1,\ldots,J, k = 1,\ldots,K$ 取右奇异向量的前 L 列，可得 $\mathbf{V}_{j,k}$。

上述正交矩阵 **Q** 可以使用以下迭代算法构造。对于实际数据，由于 $M = 4$，实际计算得到迭代 4 次时得到的 **Q** 最能满足需求。

Algorithm 1 基于随机 SVD 的矩阵降维 $\mathbf{Q} = randSVD(\mathbf{A}, \epsilon)$

输入: $\mathbf{A} \in \mathbb{C}^{m\times n}(m > n)$,$\epsilon$.
初始化:
$\mathbf{Q^0} = [\quad]$
$i = 0$
迭代过程：
while $\|\mathbf{A} - \mathbf{AQQ}^H\| > \epsilon$ **do**
  $i = i + 1$
  抽取 1 个维度为 n 的高斯随机矢量 ω^i
  $y^i = A\omega^i$
  $\hat{q}^i = (I - Q^{i-1}(Q^{i-1})^H)y^i$
  $q^i = \hat{q}^i / \left|\hat{q}^i\right|_2$
  $Q^i = [Q^{i-1}q^i]$
end while
输出:$\mathbf{Q}$ $s.t.$ $\mathbf{A} \approx \mathbf{QQ}^H\mathbf{A}$

4.1.2 基于双对角化和 QR 分解的 SVD 分解算法

对于公式2中的奇异值分解，采用基于双对角化和 QR 分解的 SVD 分解来实现。通过控制误差 ϵ，可以尽可能地减小迭代次数。SVD 分解算法如下。

Algorithm 2 SVD 分解 $(\mathbf{U}, \mathbf{S}, \mathbf{V}) = svd(\mathbf{B}, \epsilon)$

输入: $\mathbf{B} \in \mathbb{C}^{M\times N}(M > N)$,$\epsilon$.
初始化:
$\mathbf{S} = \mathbf{B}^H$
$\mathbf{U} = \mathbf{I}_{M\times M}$
$\mathbf{V} = \mathbf{I}_{N\times N}$
迭代过程：
while $err < \epsilon$ **do**
  $(\mathbf{Q}, \mathbf{S}) = qr(\mathbf{S}^H)$，$\mathbf{U} = \mathbf{U} \cdot \mathbf{Q}$
  $(\mathbf{Q}, \mathbf{S}) = qr(\mathbf{S}^H)$，$\mathbf{V} = \mathbf{V} \cdot \mathbf{Q}$
  取 **S** 对角线上方的所有元素：$\mathbf{e} = triu(\mathbf{S}, 1)$
  $err = \frac{\|\mathbf{e}\|_2}{\|diag(\mathbf{s})\|_2}$
end while
修复 S 的符号:
for $n = 1 : N$ **do**
  $snn = \mathbf{S}(n, n)$, $\mathbf{S}(n, n) = abs(snn)$
  if $snn < 0$
    $\mathbf{U}(:, n) = -\mathbf{U}(:, n)$
  endif
endfor
输出:$\mathbf{U}, \mathbf{S}, \mathbf{V}$ $s.t.$ $\mathbf{B} = \mathbf{USV}^H$

4.1.3 QR 分解

QR 分解可将矩阵 **A** 分解为一个正交矩阵 **Q** 和一个上三角矩阵 **R** 的乘积。经典的 QR 矩阵分解方法主要有 Gram–Schmidt 方法、Householder 反射方法和 Givens 旋转方法三种。考虑到需要计算的矩阵维数并不大，且后两种方法都需要计算反三角函数，会导致计算量较大。故这里采用第一种方法。基于 Gram–Schmidt 正交化方法的 QR 分解算法如下。

Algorithm 3 QR 分解 $(\mathbf{Q}, \mathbf{R}) = qr(A)$

输入: $\mathbf{A} \in \mathbb{C}^{M \times N}$,$\epsilon$
for $i = 1 : n$ **do**
  $v_i = A(:, i)$
endfor
for $i = 1 : n$ **do**
  $R_{ii} = \|v_i\|_2$
  $q_i = v_i / R_{ii}$
  for $j = i + 1 : n$ **do**
    $R_{ij} = q_i^T v_j$
    $v_j = v_j - R_{ij} q_i$
  endfor
endfor
输出: $\mathbf{Q} = [q_1, q_2, ...q_n]$, $\mathbf{R}$ $s.t.$ $\mathbf{A} = \mathbf{QR}$, 其中 **Q 为正交矩阵，R 为上三角矩阵**

由于经典的 Gram–Schmidt 正交化方法对舍入误差很敏感，容易导致生成的基 q_j 的正交性随着迭代越来越弱，因此这里使用改进的 Gram–Schmidt 正交化方法, 其核心思想是，在每个 q_j 生成后，直接把 A 剩下的列都去掉 q_j 的成分。它与传统方法的区别仅仅是把计算的顺序变了，但是改进之后稳定性会好很多。

4.1.4 基于分治思想的矩阵低复杂度计算

相关矩阵组的计算单元 $\widehat{\mathbf{W}} = f(\mathbf{H})$ 中需要使用如下的矩阵乘法公式获取 $\mathbf{W}_k$:

$$\mathbf{W}_k = \mathbf{V}_k \left(\mathbf{V}_k^H \mathbf{V}_k + \sigma^2 \mathbf{I} \right)^{-1} \tag{4}$$

其中 $\mathbf{V}_k$ 的维度为 $N \times LJ$，$\left(\mathbf{V}_k^H \mathbf{V}_k + \sigma^2 \mathbf{I} \right)^{-1}$ 的维度为 $LJ \times LJ$，针对式 (4) 中的矩阵乘法和矩阵求逆，可以使用 Strassen 方法 [8] 对矩阵乘法和求逆运算进行低复杂度计算。下面分析 Strassen 算法下矩阵乘法和求逆运输的计算复杂度。

对于阶数均为 $n = 2^k$ 矩阵乘法 $\mathbf{AB} = \mathbf{C}$，可以使用分治法的思想对矩阵进

行分块，得到

$$\mathbf{A}=\begin{bmatrix}\mathbf{A}_{11} & \mathbf{A}_{12}\\ \mathbf{A}_{21} & \mathbf{A}_{22}\end{bmatrix},\quad \mathbf{B}=\begin{bmatrix}\mathbf{B}_{11} & \mathbf{B}_{12}\\ \mathbf{B}_{21} & \mathbf{B}_{22}\end{bmatrix},\quad \mathbf{C}=\begin{bmatrix}\mathbf{C}_{11} & \mathbf{C}_{12}\\ \mathbf{C}_{21} & \mathbf{C}_{22}\end{bmatrix} \tag{5}$$

其中 $\mathbf{A}_{ij},\mathbf{B}_{ij},\mathbf{C}_{ij}$ 的阶数均为 2^{n-1}，此时有

$$\begin{bmatrix}\mathbf{C}_{11} & \mathbf{C}_{12}\\ \mathbf{C}_{21} & \mathbf{C}_{22}\end{bmatrix}=\begin{bmatrix}\mathbf{A}_{11}\mathbf{B}_{11}+\mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12}+\mathbf{A}_{12}\mathbf{B}_{22}\\ \mathbf{A}_{21}\mathbf{B}_{11}+\mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12}+\mathbf{A}_{22}\mathbf{B}_{22}\end{bmatrix} \tag{6}$$

需要使用 8 次乘法，与直接相乘需要的乘法次数没有区别。Strassen 算法通过构造使乘法次数减少到 7 次，此时使用 $\mathbf{M}_k$ 表示 $\mathbf{C}_{ij}$，有：

$$\begin{bmatrix}\mathbf{C}_{11} & \mathbf{C}_{12}\\ \mathbf{C}_{21} & \mathbf{C}_{22}\end{bmatrix}=\begin{bmatrix}\mathbf{M}_1+\mathbf{M}_4-\mathbf{M}_5+\mathbf{M}_7 & \mathbf{M}_3+\mathbf{M}_5\\ \mathbf{M}_2+\mathbf{M}_4 & \mathbf{M}_1-\mathbf{M}_2+\mathbf{M}_3+\mathbf{M}_6\end{bmatrix} \tag{7}$$

其中

$$\begin{aligned}
\mathbf{M}_1&=(\mathbf{A}_{11}+\mathbf{A}_{22})(\mathbf{B}_{11}+\mathbf{B}_{22})\\
\mathbf{M}_2&=(\mathbf{A}_{21}+\mathbf{A}_{22})\mathbf{B}_{11}\\
\mathbf{M}_3&=\mathbf{A}_{11}(\mathbf{B}_{12}-\mathbf{B}_{22})\\
\mathbf{M}_4&=\mathbf{A}_{22}(\mathbf{B}_{21}-\mathbf{B}_{11})\\
\mathbf{M}_5&=(\mathbf{A}_{11}+\mathbf{A}_{12})\mathbf{B}_{22}\\
\mathbf{M}_6&=(\mathbf{A}_{21}-\mathbf{A}_{11})(\mathbf{B}_{11}+\mathbf{B}_{12})\\
\mathbf{M}_7&=(\mathbf{A}_{12}-\mathbf{A}_{22})(\mathbf{B}_{21}+\mathbf{B}_{22})
\end{aligned} \tag{8}$$

Strassen 算法下矩阵相乘将使用 $n^{\log_2 7}$ 次复数乘法，$\frac{n^2\log_2 n}{4}\times 18$ 次复数加法，用 $M(n)$ 表示矩阵乘法的计算复杂度，则有

$$M(n)=n^{\log_2 7}\times 14+\frac{n^2\log_2 n}{4}\times 18\times 2 \tag{9}$$

对于阶数为 $n=2^k$ 矩阵求逆，同样可以使用分治法的思想进行复杂度分析。考虑分块后的 $2n\times 2n$ 矩阵求逆可以表示为：

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1}=\begin{bmatrix}\mathbf{A}^{-1}+\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\\ -\left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{C}\mathbf{A}^{-1} & \left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\end{bmatrix} \tag{10}$$

其中 $\mathbf{A}$ 和 $\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}$ 可逆。通过构造可以转换为 6 次乘法和 4 次加法以及对两个子块 $\mathbf{A},\mathbf{X}=\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}$ 的求逆：

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1}=\begin{bmatrix}\mathbf{A}^{-1}+\mathbf{M}_6 & -\mathbf{M}_5\\ -\mathbf{M}_3 & \mathbf{X}^{-1}\end{bmatrix} \tag{11}$$

其中

$$\begin{aligned}
\mathbf{M}_1 &= \mathbf{C}\mathbf{A}^{-1} \\
\mathbf{M}_2 &= (\mathbf{C}\mathbf{A}^{-1})\mathbf{B} \\
\mathbf{M}_3 &= \mathbf{X}^{-1}(\mathbf{C}\mathbf{A}^{-1}) \\
\mathbf{M}_4 &= \mathbf{A}^{-1}\mathbf{B} \\
\mathbf{M}_5 &= (\mathbf{A}^{-1}\mathbf{B})\mathbf{X}^{-1} \\
\mathbf{M}_6 &= ((\mathbf{A}^{-1}\mathbf{B})\mathbf{X}^{-1})(\mathbf{C}\mathbf{A}^{-1})
\end{aligned} \tag{12}$$

注意到 **X** 中包含一次加法运算，所以需要 4 次加法运算。

使用 $I(n), M(n)$ 和 $A(n)$ 分别表示 $n \times n$ 矩阵求逆，乘法和加法需要的次数。由式 (11) 可以得到

$$I(2n) = 2I(n) + 6M(n) + 4A(n) \tag{13}$$

$n = 2^k$ 时，上式可以转化为

$$\begin{aligned}
I\left(2^k\right) &= 2I\left(2^{k-1}\right) + 6M\left(2^{k-1}\right) + 4A\left(2^{k-1}\right) \\
&= 2^2 I\left(2^{k-2}\right) + 6\left(M\left(2^{k-1}\right) + 2M\left(2^{k-2}\right)\right) + 4\left(A\left(2^{k-1}\right) + 2A\left(2^{k-2}\right)\right) \\
&\vdots
\end{aligned} \tag{14}$$

通过对矩阵乘法的复杂度分析可知，Strassen 算法下 $M(n) = n^{\log_2 7}$，则需要 $6(\frac{7^k-2^k}{7-2})$ 次复数乘法，$k2^{k+1}$ 次复数加法，此时矩阵求逆的计算复杂度可以表示为

$$\begin{aligned}
I\left(2^k\right) &= 2^k I\left(1\right) + 6\frac{7^k - 2^k}{7-2} \times 14 + k2^{k+1} \times 2 \\
&= 2^k \times 33 + 6\frac{7^k - 2^k}{7-2} \times 14 + k2^{k+1} \times 2 \\
&= 33n + \frac{84}{5}(n^{\log_2 7} - n) + 4n\log_2 n
\end{aligned} \tag{15}$$

特别的，对于 $n = 4$ 和 $n = 8$ 时的矩阵乘法和矩阵求逆，使用 Strassen 算法和传统算法下的矩阵运算计算复杂度比较如表 1 所示。

事实上，当阶数 n 较小时，Strassen 算法的计算复杂度已经与传统算法相差不大，而 Strassen 算法由于使用了迭代，所以时间复杂度很高，因此可以设置迭代停止条件提前终止迭代从而得到更小的时间复杂度。到 2020 年 12 月为止，拥有最低逼近计算复杂度 $\mathcal{O}(n^{2.3728596})$ 的矩阵乘法算法由 Josh Alman 和 Virginia Vassilevska Williams 提出 [9]，然而这种方法以及其他基于 Strassen 的相似优化仅在极大规模数据下具有优势，并没有被实际应用。对于本题中的数据集大小，使用 Strassen 算法已经完全足够。

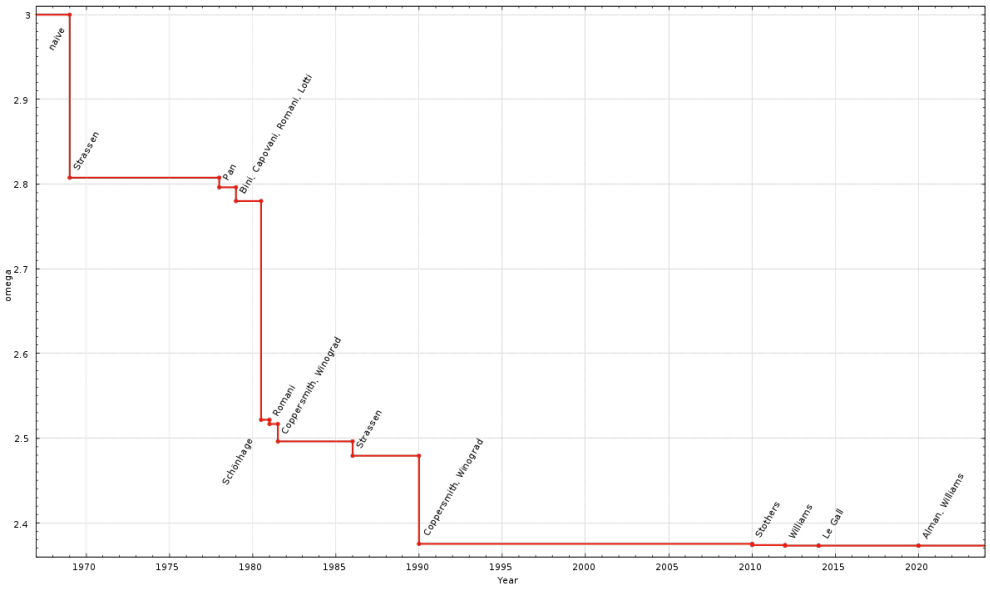

图 2　矩阵乘法计算复杂度 $\mathcal{O}(n^{\omega})$ 的优化

表 1　$n=4,8$，Strassen 算法与传统算法下矩阵乘法和求逆的计算复杂度比较

矩阵的阶数 n		4		8	
矩阵乘法	算法	Strassen	Simple	Strassen	Simple
	复数加法	144	48	864	448
	复数乘法	49	64	343	512
	计算复杂度	974	992	6530	8064
矩阵求逆	算法	Strassen			
	复数加法	16		48	
	复数乘法	54		402	
	求逆	4		8	
	计算复杂度	920		5988	

4.1.5 右奇异向量相关性分析

通过对数据集中的子矩阵进行简单分析，我们可以发现，这些子矩阵的奇异值随着行块数周期性波动，因此我们推测，行子矩阵之间的相关性可以用奇异值或者特征值之间的函数关系加以刻画，矩阵的 SVD 分解将矩阵的相关信息集中到对角线上的奇异值上，右奇异向量中继承了矩阵 **H** 的部分相关性和随

机性。事实上，根据我们对子矩阵 $\mathbf{H}_{j,k}$ 自相关性的分析，所有的自相关子矩阵 $\mathbf{R}_{j,k}=\mathbf{H}_{j,k}\mathbf{H}_{j,k}^{H}$ 相同位置的元素具有相同的均值和方差，并且距离对角线越远的元素越小，说明 $\mathbf{H}_{j,k}$ 行与行之间的自相关性随着行与行距离的增大而减小，而且自相关性衰减的速度近似为指数衰减。$\mathbf{H}_{j,k}$ 的性质比较像如下的相关信道矩阵的性质，因此我们采用该相关信道模型对子矩阵 $\mathbf{H}_{j,k}$ 进行建模。

相关信道模型中，相关矩阵中的元素表示为：

$$\begin{aligned}\mathbf{R}_r(i,k)&=\begin{cases}\left(\zeta_r e^{j\theta}\right)^{k-i}, & i\le k,\\ \mathbf{R}_r^{*}(k,i), & i>k,\end{cases}\\ \mathbf{R}_t(i,k)&=\begin{cases}\left(\zeta_t e^{j\theta}\right)^{k-i}, & i\le k,\\ \mathbf{R}_t^{*}(k,i), & i>k\end{cases}\end{aligned}\tag{16}$$

其中 $\mathbf{R}(i,k)$ 是矩阵 $\mathbf{R}$ 中的第 i 行，第 k 列的元素，ζ 表示天线间的相关程度。θ 是相位常数，仿真时设定为 $\frac{\pi}{2}$。

相关信道矩阵可以表示为：

$$\mathbf{H}_{relative}=\mathbf{R}_r^{1/2}\mathbf{H}_{rayleigh}\mathbf{R}_t^{1/2}\tag{17}$$

假设瑞利信道矩阵元素用 $h_{i,j}$ 表示，相关信道的 Gram 矩阵用 $\mathbf{G}_{relative}$ 表示，其中每个元素 $G_{i.j}$ 可以表示为：

$$G_{i,j}=\sum_{k=1}^{N_R}\sum_{a=1}^{N_R}\sum_{b=1}^{N_T}\mathbf{R}_r^{\frac{1}{2}}(i,a)h_{a,b}\mathbf{R}_t^{\frac{1}{2}}(b,j)\sum_{c=1}^{N_R}\sum_{d=1}^{N_T}\mathbf{R}_r^{\frac{1}{2}*}(i,c)h_{c,d}\mathbf{R}_t^{\frac{1}{2}*}(b,j)\tag{18}$$

当且仅当 $a=c$ 且 $b=d$ 时，瑞利信道元素才满足相关性，因此：

$$\begin{aligned}\mathbb{E}(G_{i,j})&=\sum_{k=1}^{N_R}\sum_{a=1}^{N_R}\sum_{b=1}^{N_T}\mathbf{R}_r^{\frac{1}{2}}(i,a)^2h_{a,b}^2\mathbf{R}_t^{\frac{1}{2}}(b,j)\mathbf{R}_t^{\frac{1}{2}}(b,j)\\ &=\sum_{k=1}^{N_R}\sum_{b=1}^{N_T}h_{a,b}^2\mathbf{R}_t^{\frac{1}{2}}(b,j)\mathbf{R}_t^{\frac{1}{2}}(b,j)\\ &=N_R\sigma^2\zeta_t^{|i-j|}\end{aligned}\tag{19}$$

观察到相关信道矩阵元素的期望于接收端相关系数无关，只与发射端相关系数有关，并且，随着元素的位置远离对角线，元素的均值会随着指数衰减，符合数据集中子矩阵 $\mathbf{H}_{j,k}$ 的性质，因此可以用相关信道模型描述。

下面考虑左奇异向量和右奇异向量之间的相关性，但是从我们对数据集的测试结果看，右奇异向量包含不可预测的随机性，即使可以通过插值实现估计，插值的算法如下：

$$\mathbf{V}_{l,j,k}=\lambda_{l,j,k}\mathbf{V}_{l,j,k-1}+(1-\lambda_{l,j,k})\mathbf{V}_{l,j,k+1}\tag{20}$$

一般来说 $\lambda_{l,j,k}$ 的最优值收敛到 0.5，但是即使可以调节线性插值的系数，还是会存在部分预测结果与标准数据集之间的差距小于 0.995，因此，我们提出如下方法，用于识别不同的 **H** 矩阵数据集中可插值的右奇异向量，其他的右奇异向量只能通过直接求解的方式计算。

假设子矩阵 $\mathbf{H}_{j,k}$ 的 SVD 分解对应的右奇异向量的前两列为 $\mathbf{V}^{svd}_{1:L,j,k}$，相邻子矩阵间的互相关矩阵可以写成 $\mathbf{R}_{j,(k,k+1)} = \mathbf{V}_{1:L,j,k}\mathbf{V}^{H}_{1:L,j,k+1}$，

$$\begin{aligned}\mathbf{R}_{j,k} =&\mathbf{U}_{j,k}\cdot[\mathbf{S}_{j,k}\ \mathbf{0}]\cdot[\mathbf{v}^{H}_{1,j,k}\ \mathbf{v}^{H}_{2,j,k}\cdots]\cdot[\mathbf{v}_{1,j,k+1}\ \mathbf{v}_{2,j,k+1}\cdots]^{H}\cdot[\mathbf{S}^{H}_{j,k+1}\ \mathbf{0}]^{H}\cdot\mathbf{U}^{H}_{j,k+1}\\ =&\mathbf{U}\cdot\begin{bmatrix}\lambda_1\lambda_2\mathbf{v}^{H}_{1,j,k}\mathbf{v}_{1,j,k+1} & \cdots & \cdots\\ \vdots & \lambda_1\lambda_2\mathbf{v}^{H}_{2,j,k}\mathbf{v}_{2,j,k+1} & \cdots\\ \vdots & \vdots & \ddots\end{bmatrix}\cdot\mathbf{U}^{H}\end{aligned} \tag{21}$$

其中 λ 代表奇异值。

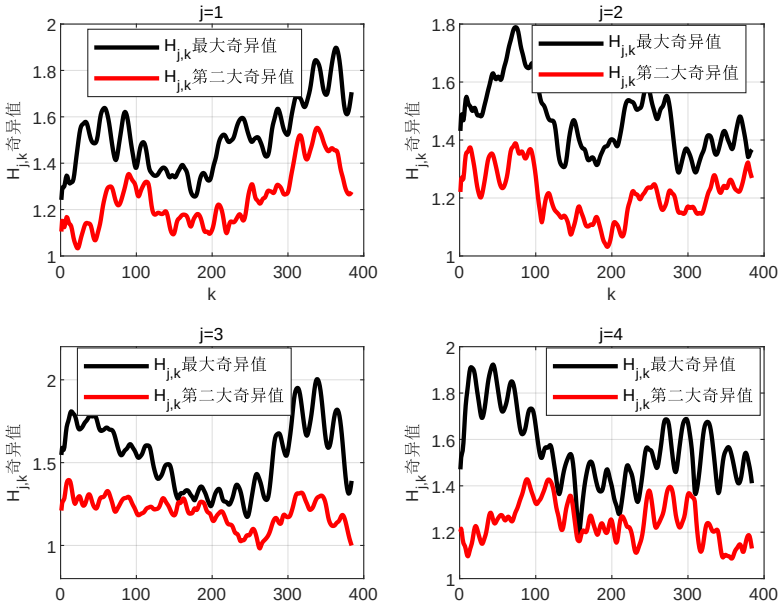

图 3　Data1_H 奇异值随行块 k 的变化

可以看到公式21 描述右奇异向量相关性的变量集中分布在对角线上，因此我们提取对角线上的元素分析相邻右奇异向量之间的相关性，由于只需要前两列的右奇异向量，因此我们只截取相邻互相关矩阵对角线上的前两个元素。由于子矩阵的奇异值可以通过拟合得到，因此很容易可以得到相邻右奇异向量的相关系数 $\mathbf{v}^{H}_{1,j,k}\mathbf{v}_{1,j,k+1}$，$\mathbf{v}^{H}_{2,j,k}\mathbf{v}_{2,j,k+1}$，也就是后文子矩阵的自相关性的定量化数值。得到估计的相邻右奇异向量的相关系数后，我们绘制数据集 $\mathbf{V}_{l,j,k}$ 的互相关系数与估计值进行对比，拟合出两者的关系，这样我们就得到了推断右奇异向量互相关性的依据，借助右奇异向量互相关性我们就可以判断出可以进行插值的向量

以及相关性较差的向量，调整我们之前的算法。

通过对数据集的观察，我们发现不同的数据集呈现不同的特征，其中数据互相关性最好的是 Data3_H 数据集和 Data4_H，基本上所有的子矩阵互相关性都达到了 0.995 以上，这说明用相邻相邻插值的方式完全可以保证估计精度在 0.995 以上，这样的数据集理论上最大可以减少一般的计算量。而数据相关性最差的数据集是 Data3_H 数据集，子矩阵的随机性较大，计算量减少的比例较少。

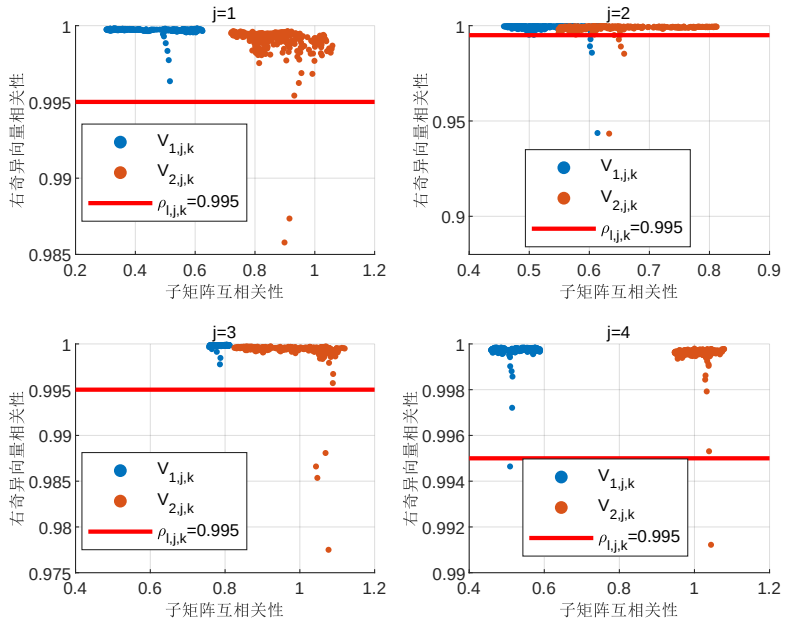

图 4　Data4_H 子矩阵互相关性和右奇异向量的相关性

由于数据集的数目不足，想要通过模型训练实现精确地找到所有弱相关性的矩阵几乎是不可能的。因此我们采用的策略是划定一个子矩阵互相关性变化的区间，在这个区间内，右奇异向量必然可以用相邻位置的右奇异向量插值得到。子矩阵互相关性在这个区间外的向量则有一定的概率是相关性弱的矩阵，为了让最小建模估计精度大于 0.99，这些矩阵只能全部计算。如图4，Data4_H 数据集中相关性较小的点基本都分布在子矩阵互相关性为 0.5 和 1 附近，可以通过计算子矩阵互相关性，推测右奇异向量的相关性。

因此每个数据集计算右奇异向量的次数取决于数据集本身的互相关性。假设一个数据集“好的”子矩阵数目为 P，“坏的”子矩阵数目为 $1-P$，由于好的矩阵可以通过插值的方式减少一半的计算量，那么一共需要计算的 SVD 分解的次数为 $1-\frac{P}{2}$，下面给出 6 个数据集计算 SVD 分解的次数：

不同数据集	SVD 分解次数	计算量减少比例
Data1_H	795	48.29%
Data2_H	824	46.37%
Data3_H	896	41.72%
Data4_H	769	49.93%
Data5_H	771	49.82%
Data6_H	781	49.19%
不考虑插值	1536	0%

表 2　不同算法的迭代式半径

由于右奇异向量引入的误差也会影响到之后矩阵求逆部分的误差，因此是否将误差阈值设置为 0.995 还有待研究，但是可以肯定的是插值的方式可以一定程度上减少 SVD 的计算次数。

4.1.6 计算复杂度分析

本节基于以上算法和分析，逐层反推矩阵 **V** 计算的复杂度。具体的分析如下。

- 基于随机 SVD 的矩阵降维：根据算法 1, 取迭代次数为 4，则步骤 9 的计算复杂度为 15872；步骤 10 计算复杂度为 $76N^2 + 140N - 8 = 320248$；步骤 11 计算复杂度为 $4(18N + 48) = 4800$，故总计算复杂度为 340920。
- 经过矩阵降维，我们还需要对 $M \times M$ 的矩阵 $\mathbf{B} = \mathbf{Q}^H\mathbf{A}$ 进行 SVD 分解，其过程设计到迭代 QR 分解的运算。对 $m \times n$ 矩阵进行一次 QR 分解的计算复杂度如表 3 所示。对于本题，$m = n = M = 4$，故一次 QR 迭代的计算复杂度为 948。
- 假设计算 SVD 过程中进行迭代部分循环次数为 $iter$，则一共进行了 $2 \times iter$ 次 QR 分解；再假设迭代结束后 S 中奇异值均为正实数，即无需修复 S 的符号。则一次 SVD 分解的总复杂度为 $1896 \times iter$。
- 对于上述 SVD 分解，ϵ 的取值，平均迭代次数 $iter$ 与 ρ_{min} 之间的关系如下表所示。由数据可知，取 $\epsilon = 0.002$ 时得到的最小迭代次数约为 20 次，则一次 SVD 分解的计算复杂度为 37920。因此，对一个子矩阵求 **V** 的计算复杂度为 378840。

表 3　ϵ 的取值，平均迭代次数 $iter$ 与 ρ_{min} 之间的关系

	SVD 分解误差	平均迭代次数	ρ_{min}
1	0.001	22.04	0.9981
2	0.002	19.17	0.9927
3	0.003	17.51	0.9834

表 4　QR 分解计算复杂度

	2 范数计算	归一化除法	向量乘法	正交化	**总计**
实数乘法	4m	2m	4m	4	6mn+2(m+1)n(n-1)
实数加法	4m-2	0	4m-2	4	(3m-1)n+(2m+1)n(n-1)
开方	1	-	-	-	n
实数除法	-	1	-	-	n
次数	n	n	n(n-1)/2	n(n-1)/2	-
总计	n(16m+23)	n(2m+25)	n(n-1)(8m-1)	8n(n-1)	n(8mn+10m+7n+41)

- 利用矩阵之间的相关性，可以减少约 40% 的计算复杂度。综上，对 4×384 的相关矩阵组计算 **V** 的近似低复杂度计算的总计算复杂度约为 $3.491e+8$。

4.2 子问题 2——矩阵 W 的近似低复杂度计算

4.2.1 基于 AOR 迭代的矩阵近似求逆算法

根据题目的定义，矩阵 **W** 捕捉方式是先将不同下标 j，相同下标 k 的 $\mathbf{V}_{j,k}$ 进行拼接，得到维度为 $N \times LJ$ 的 $\mathbf{V}_k = \left[\begin{array}{cccc} \mathbf{V}_{1,k} & \cdots & \mathbf{V}_{j,k} & \cdots \end{array} \mathbf{V}_{J,k} \right]$，然后根据如下公式获取 $\mathbf{W}_k$：

$$\mathbf{W_k} = \mathbf{V_k} \left(\mathbf{V_k^H V}_k + \sigma^2 \mathbf{I} \right)^{-1} \tag{22}$$

其中，σ^2 为固定常数；**I** 为单位矩阵，维度为 $LJ \times LJ$。

将 $\mathbf{W}_k$ 拆分成列向量的形式 $\mathbf{W}_{l,j,k} \in \mathbb{C}^{N \times 1}$，以上的求逆问题可以转换为 $L*J$ 次求解线性方程组：

$$\left(\mathbf{V}_k^H \mathbf{V}_k + \sigma^2 \mathbf{I} \right) \mathbf{W}_{l,j,k}^H = \mathbf{V}_{l,j,k}^H, l = 1, \cdots, L, j = 1 \cdots, J \tag{23}$$

为避免矩阵求逆，形如超松弛（successive overrelaxation，SOR）迭代算法、纽曼展开近似、共轭梯度法、牛顿迭代法、最速下降法、Barzilai-Borwein 算法、

QR 分解等被应用于求解线性方程组中，从数学矩阵变换的角度实现复杂度和性能上的一个均衡。通过以上的方法可以实现将原本 $\mathcal{O}(n^3)$ 的矩阵求逆复杂度，降低到 $\mathcal{O}(n^2)$，但是综合考虑算法的复杂度和性能，我们最终选择适应性较强，迭代次数较少的加速超松弛（accelerated overrelaxation，AOR）迭代算法。AOR 迭代算法具有两个收敛因子，可以根据特定的求解问题进行调整，提高算法的迭代性能，同时减小迭代次数。

AOR 算法对目标矩阵 $\mathbf{V}_k^H\mathbf{V}_k+\sigma^2\mathbf{I}$ 进行分解 $\mathbf{V}_k^H\mathbf{V}_k+\sigma^2\mathbf{I}=\mathbf{D}+\mathbf{E}+\mathbf{F}$，$\mathbf{E}$ 是下半角矩阵，$\mathbf{F}$ 是上半角矩阵，定义矩阵 $\omega\mathbf{A}$ 分解如下：

$$\omega(\mathbf{V}_k^H\mathbf{V}_k+\sigma^2\mathbf{I})=(\mathbf{D}+r\mathbf{E})-((1-\omega)\mathbf{D}-(\omega-r)\mathbf{E}-\omega\mathbf{F}) \tag{24}$$

将 $(\mathbf{D}+r\mathbf{E})$ 项乘以 $\mathbf{x}_{k+1}$，将 $((1-\omega)\mathbf{D}-(\omega-r)\mathbf{E}-\omega\mathbf{F})$ 项乘以 $\mathbf{x}_k$，得到差分表达式：

$$(\mathbf{D}+r\mathbf{E})\mathbf{x}_{k+1}=[(1-\omega)\mathbf{D}-(\omega-r)\mathbf{E}-\omega\mathbf{F}]\mathbf{x}_k+\omega\mathbf{V}_{l,j,k}. \tag{25}$$

移项可得 AOR 迭代表达式：

$$\mathbf{x}_{k+1}=(\mathbf{D}+r\mathbf{E})^{-1}[(1-\omega)\mathbf{D}-(\omega-r)\mathbf{E}-\omega\mathbf{F}]\mathbf{x}_k+\omega(\mathbf{D}+r\mathbf{E})^{-1}\mathbf{V}_{l,j,k}. \tag{26}$$

当迭代式26收敛时，$\mathbf{x}_k$ 就可以表示为 $\mathbf{W}_{l,j,k}$ 的近似结果。也可以观察到当 $\omega=r$ 时，AOR 算法退变成 SOR 算法。AOR 迭代也是收敛的。

结合以上对 AOR 系线性迭代算法的分析，我们对不同线性迭代算法的谱半径做一个总结，假设 Jacobi 迭代式的谱半径为 $\rho(\mathbf{M})$：

不同算法	迭代式谱半径（$\rho(\mathbf{M})$）
Chebyshev 多项式 +Jacobi	$1-\sqrt{1-\rho(\mathbf{M})^2}$
Gauss-Seidel	$\rho(\mathbf{M})^2$
AOR(ω_{opt})	$\frac{1-\sqrt{1-\rho(\mathbf{M})^2}}{1+\sqrt{1-\rho(\mathbf{M})^2}}$

表 5　不同算法的迭代式半径

从表中可以观察到 AOR 算法的谱半径比其他线性迭代算法都要小，也就是 AOR 迭代达到收敛所需要的迭代次数较少。因此我们采用 AOR 迭代算法计算公式23中的 $\mathbf{W}_{l,j,k}$。

通过以上分析，基于 AOR 迭代的线性方程组求解算法可以写成：

Algorithm 4 基于 AOR 迭代的线性方程组求解算法

输入: $\mathbf{V}_k$，σ^2
初始化:
$\mathbf{G} = \mathbf{V}_k^H \mathbf{V}_k + \sigma^2 \mathbf{I}$
$\mathbf{G} = \mathbf{D} + \mathbf{E} + \mathbf{F}$
$\mathbf{w} = (\mathbf{D})^{-1} \mathbf{V}_k$
$\mathbf{M} = (\mathbf{D} + r\mathbf{E})^{-1}[(1-\omega)\mathbf{D} - (\omega - r)\mathbf{E} - \omega\mathbf{F}]$
$\mathbf{N} = \omega(\mathbf{D} + r\mathbf{E})^{-1}\mathbf{w}$
迭代过程:
for $m = 1 : L * J$ **do**
  $\mathbf{x}_1 = \mathbf{w}_m$
  for $k = 1 : iternum$ **do**
    $\mathbf{x}_{k+1} = \mathbf{M}\mathbf{x}_k + \mathbf{N}_m$
  endfor
  $\mathbf{w}_m = \mathbf{x}^{iternum}$
endfor
输出 $\mathbf{W}_k = \mathbf{w}^H$

4.2.2 矩阵 W 的相关性分析

与矩阵 $\mathbf{V}_{j,k}$ 的处理方法一致，矩阵 $\mathbf{W}_k$ 也可以通过插值的方式得到，每一个子矩阵 $\mathbf{W}_{j,k}$ 也是具有较强的相关性的，以数据集 Data4_V 与 Data4_W 为例，分别计算 $\mathbf{V}_{j,k}$ 的互相关性 $\mathbf{Rv}_{j,(k,k+1)} = \|\mathbf{V}_{j,k}^H \mathbf{V}_{j,k+1}\|_2$，$\mathbf{W}_{j,k}$ 的互相关性 $\mathbf{Rw}_{j,(k,k+1)} = \|\mathbf{W}_{j,k}^H \mathbf{W}_{j,k+1}\|_2$，绘制散点图：

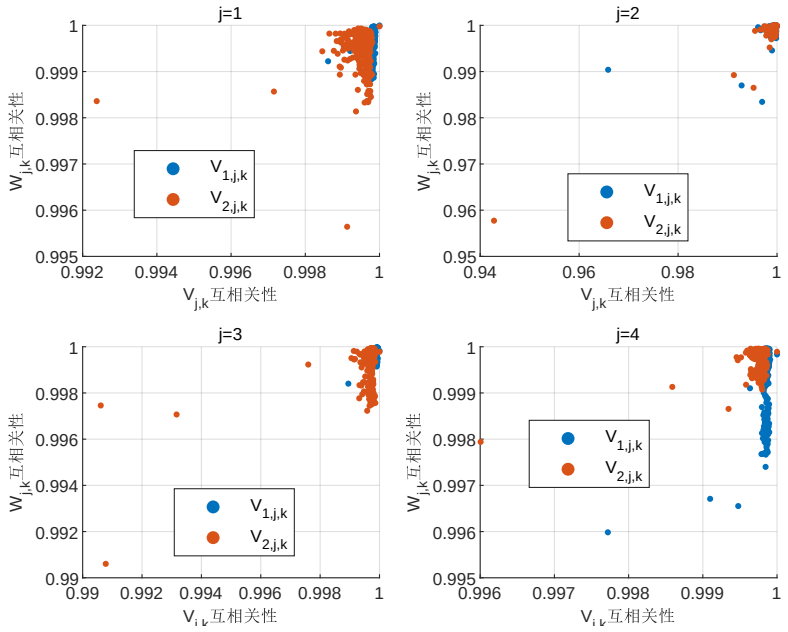

图 5　Data4_H W 矩阵和 V 矩阵相关性

可以观察到 $\mathbf{V}_{j,k}$ 的互相关性与 $\mathbf{W}_{j,k}$ 的互相关性几乎满足线性关系。因此我

们仍然可以采用第一小问的思路，划分置信空间，如果计算出的 $\mathbf{V}_{j,k}$ 的互相关性落入在置信区间内，我们就可以肯定这个 $\mathbf{V}_{j,k}$ 对应的 $\mathbf{W}_{j,k}$ 是可以通过相邻 $\mathbf{W}_{j,k}$ 插值得到的。否则，$\mathbf{W}_{j,k}$ 有一定概率得到插值错误的结果。$\mathbf{W}_{j,k}$ 的插值公式如下：

$$\mathbf{W}_{l,j,k} = \mu_{l,j,k}\mathbf{W}_{l,j,k-1} + (1-\mu_{l,j,k})\mathbf{W}_{l,j,k+1} \tag{27}$$

下面我们综合考虑 $\mathbf{V}_{j,k}$ 和 $\mathbf{W}_{j,k}$ 插值的误差，计算 $\rho_{l,j,k}(\mathbf{W})$ 中建模精度小于 0.99 的点的个数。我们设置 $\mathbf{V}_{j,k}$ 的插值系数为 $\lambda = 0.5$，$\mathbf{W}_{j,k}$ 矩阵的插值系数为 $\mu = 0.5$，以数据集 Data4 为例，最终计算出的 $\rho_{l,j,k}(\mathbf{W})$ 如下图所示：

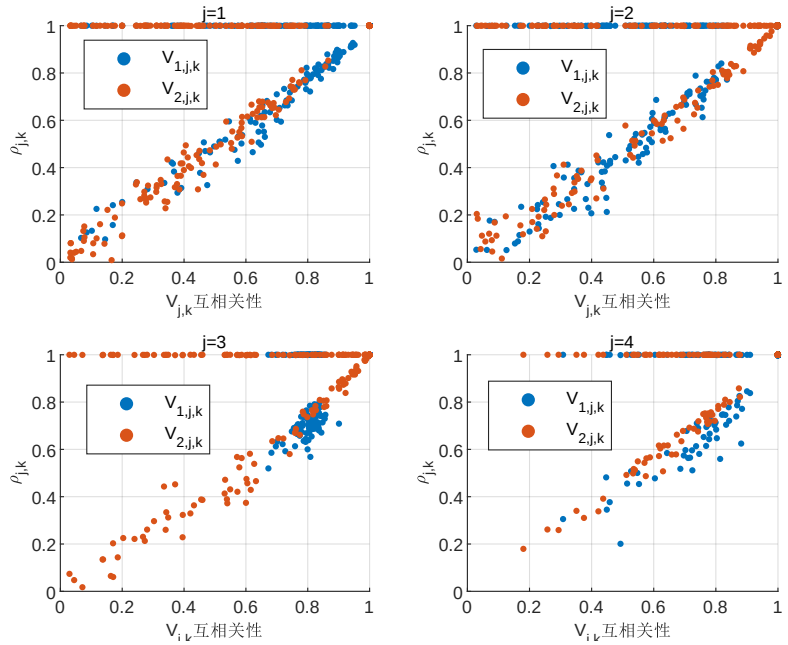

图 6　Data4_H V 矩阵相关性与建模精度 $\rho_{l,j,k}(\mathbf{W})$ 的关系

可以观察到，部分情况下 $\mathbf{V}_{j,k}$ 的互相关性与建模精度 $\boldsymbol{\rho}_{l,j,k}(\mathbf{W})$ 满足线性关系，因此可以通过评估 $\mathbf{V}_{j,k}$ 的互相关性判断是否可以对 $\mathbf{W}_{j,k}$ 行插值，如果 $\mathbf{V}_{j,k}$ 的互相关性较弱，只能通过直接计算求解 $\mathbf{W}_{j,k}$。不满足精度的点需要重新进行 $\mathbf{V}_{j,k}$ 和 $\mathbf{W}_{j,k}$ 的计算，满足精度的点可以通过插值得到。我们设置 $\mathbf{V}_{j,k}$ 的插值系数为 $\lambda = 0.5$，$\mathbf{W}_{j,k}$ 矩阵的插值系数为 $\mu = 0.5$，得到各数据集的完全插值的计算结果如下：

不同数据集	精度满足 0.99 的点数	计算量减少比例
Data1	964	37.22%
Data2	966	37.16%
Data3	1001	34.86%
Data4	941	38.74%
Data5	950	38.15%
Data6	946	38.44%
不考虑插值	1536	0%

表 6　不同算法的迭代式半径

综合考虑 $\mathbf{W}_{j,k}$ 和 $\mathbf{V}_{j,k}$ 的插值结果，我们发现对于计算精度不足的点采用直接计算的方式，对于计算精度满足要求的点采用插值的方式能够减少大约 30% 左右的计算量。

4.2.3 AOR 迭代算法复杂度分析

本小节给出了基于 AOR 迭代算法求解 $\mathbf{W}_k$ 详细计算复杂度分析。首先需要计算 $\mathbf{G} = \boldsymbol{V}_k^H \boldsymbol{V}_k + \sigma^2 \boldsymbol{I}$。这相当于一个 8×64 的矩阵和一个 64×8 的矩阵相乘，得到一个 8×8 矩阵，再在对角线元素上加 1。前者可以拆分成 8 个 8×8 矩阵的乘法再求和，后者相当于 8 次实数加法。因此，计算 $\mathbf{G}$ 的复杂度为

$$6530 \times 8 + 2 \times 7 \times 8 \times 8 + 8 = 53144. \tag{28}$$

对于 $m, k = 1, 2, 3, \cdots, LJ$ 来说，我们将 AOR 算法的迭代式转换成元素的形式：

$$\begin{aligned} \hat{x}_m^{(i+1)} =& \frac{1}{G_{m,m}}((1-\omega) G_{m,m} \hat{x}_m^{(i+1)} - (\omega - r) \sum_{k<m} G_{m,k} \hat{x}_k^{(i)} - \omega \sum_{k>m} G_{m,k} \hat{x}_k^{(i)} \\ & - r \sum_{k<m} G_{m,k} \hat{x}_k^{(i+1)} + V_{m,l,j,k}) \end{aligned} \tag{29}$$

其中 $\hat{x}_m^{(i)}$，$\hat{x}_m^{(i+1)}$ 分别是对应向量的第 m 个元素。$V_{m,l,j,k}$ 对应向量 $\mathbf{V}_{l,j,k}$ 的第 m 个元素。首先计算该式的复数乘法数量，括号外侧的存在一个复数除法，但是根据数据集测试的结果由于 $\mathbf{V}_k$ 是标准正交向量组，因此 $\mathbf{W}_k$ 对角线上的元素均为确定实数（1+σ^2），相当于运算复杂度仅为一个实数乘法，可以忽略，括号内侧的 5 个多项式的复数乘法数目分别为：2，$m+1$，$LJ-m+1$，m 和 0。因

表 7　不同线性迭代算法硬件复杂度对比

不同算法		乘法复杂度	加法复杂度
AOR	$i=2$	$3(LJ)^2+9LJ$	$3(LJ)^2+3LJ$
	$i=3$	$\frac{9}{2}(LJ)^2+\frac{27}{2}LJ$	$\frac{9}{2}(LJ)^2+\frac{9}{2}LJ$
Richardson 算法	$i=2$	$2(LJ)^2-2LJ$	$2(LJ)^2+2LJ$
	$i=3$	$3(LJ)^2-3LJ$	$3(LJ)^2+3LJ$
Jacobi 算法	$i=2$	$2(LJ)^2-2LJ$	$2(LJ)^2-2LJ$
	$i=3$	$3(LJ)^2-3LJ$	$3(LJ)^2-3LJ$
Gauss-Seidel 算法	$i=2$	$2(LJ)^2$	$2(LJ)^2$
	$i=3$	$3(LJ)^2$	$3(LJ)^2$

此对于 $\hat{x}_m^{(i+1)}$ 每次迭代消耗的复数乘法次数为 $LJ+m+4$，一次 AOR 迭代的复数次数为 $\frac{3}{2}(LJ)^2+\frac{9}{2}LJ$。可以看到 AOR 迭代算法的复杂度数量级是 $\mathcal{O}\left((LJ)^2\right)$，相比于 MMSE 检测复杂度 $\mathcal{O}\left((LJ)^3\right)$ 降低了一个数量级。

对于复数加法来说,中间五项每一项的加法复杂度分布为:0,$m-1$,$LJ-m-1$,$m-1$ 和 0，加上每一项之间彼此的加法运算，因此对于 $\hat{x}_m^{(i+1)}$ 每次迭代消耗的复数加法次数为 $LJ+m+1$，一次 AOR 迭代的加法次数为 $\frac{3}{2}(LJ)^2+\frac{3}{2}M$。

其余线性迭代算法的算法复杂度依此类推可得，在这里不详细展开。

一般来说本问题的矩阵 **G** 的病态值较高，线性迭代算法的收敛速度很快，AOR 算法只需要迭代 2 次就可以达到收敛，如果求解 $\mathbf{W}_k$ 阶段的误差允许，可以将 AOR 算法退化为 Gauss-Seidel 算法，复杂度降低较多，但是相应的性能有所下降。根据题目已知复数乘法需要 4 次实数乘法和 2 次实数加法，需要消耗 14 的复杂度，复数加法需要 2 次实数加法，需要消耗 2 的复杂度。如果采用 Gauss-Seidel 算法迭代 2 次，每计算一个 $\mathbf{W}_{l,j,k}$ 的总计算复杂度为：

$$\begin{aligned}\mathcal{O}(\mathbf{W}_{l,j,k}) =& 14*2(LJ)^2+2*(LJ)^2\\ =& 30(LJ)^2\end{aligned} \tag{30}$$

最后，我们可以计算对一个数据集由 **V** 生成 **W** 的运算复杂度：

$$(53144\times 384+30\times 8^2\times 8\times 384)\times 0.6=1.578e+7, \tag{31}$$

其中 0.6 表示利用相关性插值获得的计算复杂度减免。再加上由 **U** 生成 **V** 的计算复杂度，可得 $\widehat{\mathbf{W}} = f(\mathbf{H})$ 总复杂度为 $3.648e+8$。

5. 问题二求解

5.1 子问题 1——H 矩阵的压缩

题目要求利用矩阵数据之间的相关性，对 H 和 W 分别建立压缩/解压缩模型，并综合考虑存储复杂度（记为 C_{sav}）、压缩/解压缩复杂度（分别记为 C_{enc} 和 C_{dec}）对模型选取的影响。换言之，问题可以建模为一个多目标优化问题。类比相关矩阵与视频流，我们考察了各种帧内压缩和帧间压缩算法，并得出以下结论：

- DCT 变换对处理本题中矩阵的压缩不具有优越性，但变换域压缩方法具有启发性。JPEG 中使用变换域压缩算法，将图像划分成 8×8 的子块，并对每个子块进行 DCT 变换。但是，图像处理中一般处理的是实矩阵，实矩阵经过 DCT 变换之后仍为实数矩阵，且能量会集中于低频分量。而本题中需要处理的矩阵为复数矩阵，经过 DCT 变换之后仍为复数，且其模值不具备前述提到的主要集中于低频的特点，故不具有优越性。
- 另一方面，DCT 变换本身是无损的，实现压缩的是变换之后使用不同的值对不同分量进行量化，以达到保持低频分量，压缩高频分量的目的，而本题中要求压缩后的每个元素仍然以 32bit 单精度浮点数存储，不考虑位宽的压缩，故不能通过量化的方式实现压缩。
- 对于帧间编码，如 H.264 中使用运动估计和运动补偿的方式对中间帧内容进行预测，可以大大提高压缩效率。但是，各个 $\mathbf{H}_{j,k}$ 的数据之间并不像图像那样具有非常直观的相关性，故这种方法难以用于本题中的压缩。

5.1.1 基于 SVD 分解的矩阵压缩算法

虽然 DCT、FFT 等变换域压缩算法对于本题的矩阵并没有很好的压缩性能，但如果我们能够找到一个比较好的变换方法，那么问题将迎刃而解。而事实上，我们在问题 1 中使用的 SVD 分解就是一个非常好的切入点。

对于第 j 行的矩阵序列，我们将同一行的每个矩阵 $\mathbf{H_{j,k}}$ 都展开成 $M \times N = 256$ 维的列向量，并以每 s 个向量为一组横向拼接在一起，构成一个 $MN \times s$ 维的新矩阵 $\mathbf{X_{j,p}}$，$p = 1, 2, ... \left\lceil \frac{K}{s} \right\rceil$。如果划分到最后剩余向量数不足 s，则在其右侧补充零向量使其达到 $MN \times s$ 维。将 $\mathbf{X_{j,p}}$ 进行 SVD 分解 $\mathbf{X_{j,p}} = \mathbf{USV}^H$ 并取其奇异值。我们发现，这个新的 $MN \times s$ 矩阵的前几个奇异值都较大，而最后的几个奇异值小到几乎可以忽略，即条件数较大。这说明，这个新矩阵前面较大的奇异

值包含了矩阵绝大多数的信息，即使舍弃最后那些较小的奇异值，也不会使得原始矩阵的的值产生什么改变。因此，利用 SVD 分解可以将矩阵包含的信息集中的前几个奇异值构成的线性空间中，从而达到数据压缩的效果。我们假设选取前 c（$c < s$）个奇异值并将其余奇异值置为 0，恢复出原始的矩阵。计算可以得到这种压缩方式带来的误差 err_H 是相对较小的。

基于 SVD 分解的矩阵压缩算法 $P_1(\cdot)$ 如下。对任意 $\mathbf{X_{j,p}}$, 只需要知道仅需要存储前 c 个奇异值和 **U**、**V** 的前 c 列即可近似地会付出 $\mathbf{X_{j,p}}$。此外，可以将各奇异值分别乘以 **V** 中的对应的列向量（因为 **V** 的维数远小于 **U**），这样连 c 个奇异值也不需要存储了，且这种方法只是将解压时候的计算复杂度移动到压缩时候处理，故不会增加总的压缩/解压复杂度。

Algorithm 5 基于 SVD 分解的矩阵压缩算法 $\mathbf{C} = P_1(\mathbf{H}, s, c, \epsilon)$

输入: $\mathbf{H} = \{\mathbf{H_{j,k}}\}$,$\mathbf{H_{j,k}} \in \mathbb{C}^{M \times N}$,$j = 1, 2, ..., J$,$k = 1, 2, ..., K$,$s$,$c$.
for $j = 1 : J$ **do**
  for $p = 1 : \lceil \frac{K}{s} \rceil$ **do**
    for $kk = 1 : s$ **do**
      $k = s(p-1) + kk$
      if $k > K$
        $X(:, kk) = \theta$
      else
        $X(:, kk) = reshape(\mathbf{H_{j,k}}, [MN \times 1])$
      endif
    endfor
    $(\mathbf{U}, \mathbf{S}, \mathbf{V}) = svd(X, \epsilon)$
    $\widetilde{\mathbf{U}} = \mathbf{U}(:, 1 : c)$,$\widetilde{\mathbf{V}} = \mathbf{S}(1 : c, 1 : c)\mathbf{V}(:, 1 : c)$
    $\mathbf{C_{j,p}} = \left\{\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}}\right\}$
  endfor
endfor
输出: $\mathbf{C} = \{\mathbf{C_{j,p}}\}$,$\mathbf{C_{j,p}} = \left\{\widetilde{\mathbf{U_{j,p}}}, \widetilde{\mathbf{V_{j,p}}}\right\}$, $j = 1, 2, ..., J$,$p = 1, 2, ..., \lceil \frac{K}{s} \rceil$

对应的解压缩算法 $G_1(\cdot)$ 如下。它只需要仅需要进行 SVD 分解的逆运算 $\widehat{\mathbf{X}} = \sum_{i=1}^{c} \mathbf{u_i}(\sigma_i \mathbf{v_i}^T)$ 来恢复矩阵。因此，每次为了恢复 **H** 中的 s 个矩阵，计算 $\mathbf{u_i}(\sigma_i \mathbf{v_i}^T)$ 再求和即可恢复原始矩阵的信息。显然，基于 SVD 分解的方法是一种非对称压缩，其压缩复杂度远高于解压复杂度。

Algorithm 6 基于 SVD 分解的矩阵压缩算法 $\mathbf{H} = G_1(C)$

输入: $\mathbf{C} = \{\mathbf{C_{j,p}}\}$,$\mathbf{C_{j,p}} = \left\{\widetilde{\mathbf{U_{j,p}}}, \widetilde{\mathbf{V_{j,p}}}\right\}, j = 1, 2, ..., J$,$p = 1, 2, ..., \lceil \frac{K}{s} \rceil$
for $j = 1 : J$ **do**
  for $p = 1 : \lceil \frac{K}{s} \rceil$ **do**
    $\mathbf{X} = \widetilde{\mathbf{U_{j,p}}}\widetilde{\mathbf{V_{j,p}}}^T$
    for $kk = 1 : s$ **do**
      $k = s(p-1) + kk$
      if $k \leq K$
        $\mathbf{H_{j,k}} = reshape(X(:, p), [M \times N])$
      endif
    endfor
endfor
输出: $\mathbf{H} = \{\mathbf{H_{j,k}}\}$,$\mathbf{H_{j,k}} \in \mathbb{C}^{M \times N}$,$j = 1, 2, ..., J$,$k = 1, 2, ..., K$

针对这一压缩/解压算法，我们作如下讨论。

- 使用奇异值分解进行压缩的好处：一方面，通过对第一题的分析，我们已经有了现成的低复杂度 SVD 分解算法，则对 H 和 W 的压缩与解压无需另外设计算法；另一方面，SVD 分解无需像傅里叶变换、DCT 变换那样使用复杂的三角函数和幂函数计算，从而具有较低的计算复杂度。
- 本题中，s 一般取为 2 的幂次，如 8，16，32，64 等，这样既可以方便矩阵乘法计算，也可以使 K 能够整除 s 从而避免了补 0 的情况。
- 由于我们只需要前 c（$c < s$）个奇异值以及 **U**、**V** 的前 c 列，因此理论上可以用更加快速的方法进行计算。这里由于问题一的求解中分析了快速 SVD 分解的计算方法，故仍使用前面的算法。
- 本题中使用的 c 是通过实际计算估计出的最优结果。在实际使用中，如果不同矩阵的条件数相差较大，可以根据奇异值的实际分布，对每个矩阵独立地选取 c 的取值。本题中由于假设了各个子矩阵具有相同的统计特性，故不需要这么做。

5.1.2 基于 SVD 的矩阵相关度分析

本节基于上述数据压缩方法，分析各矩阵之间的相关性。并为存储复杂度与压缩/解压缩复杂度的联合优化分析提供参考。

图 7 给出了矩阵合并数目 $s = 8, 16, 32$ 时，不同奇异值截取数目 c 对应的 **H** 压缩解压缩误差 err_H 变化。随着奇异值截取数目 c 的增加，误差 err_H 减小，可以达到的最小误差受 SVD 允许误差 ϵ 限制。这充分说明了压缩算法的可行性，并进一步表明，在相同的 ϵ 取值下，矩阵合并数目 s 越大，则可达到的压缩效率就越高。同时，对于 $\epsilon \leq 1e-2$ 的情形，无论 s 取 $8, 16, 32$ 中的何值，都能找到相同

的 c 值满足 err_H 的约束。具体而言，分别取 $c=3,4,6$ 即可保证 $err_H<-30\text{dB}$。这说明 SVD 分解误差的上限 ϵ 不需要取得非常小即可满足题目要求，而较小的 ϵ 可以降低迭代次数，进而减小压缩复杂度。

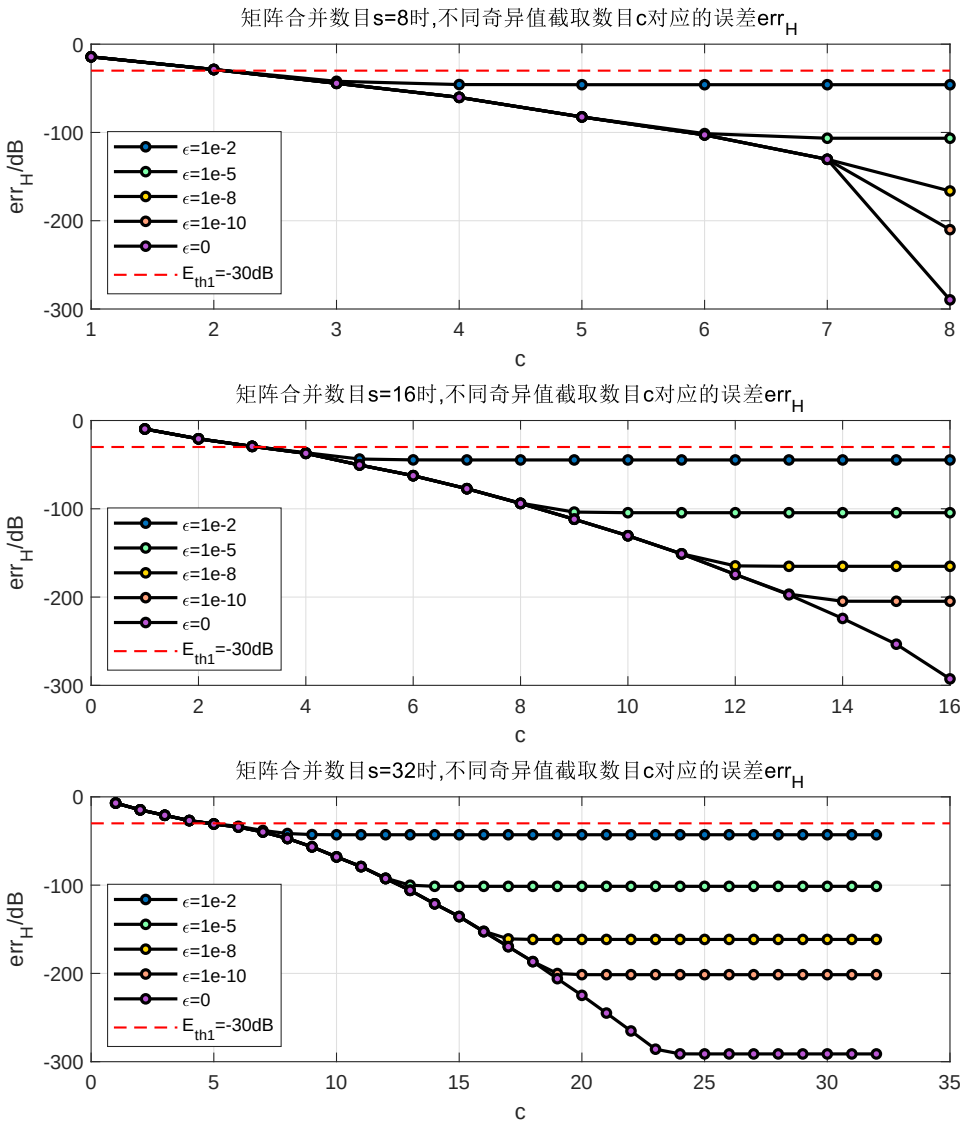

图 7　矩阵合并数目 $s=8,16,32$ 下，不同奇异值截取数目 c 对应的误差 err_H

图 8 给出了矩阵合并数目 $s=8,16,32$ 时，不同 SVD 分解允许误差 ϵ 对应的平均迭代次数 $iter$ 变化。随着 ϵ 的减小，平均迭代次数 $iter$ 增大，且矩阵合并数目 s 越大，这种增大越明显。这说明，我们需要寻找合适的 ϵ 和 s 使得 SVD 分解的计算量在可接受的范围内，并选取合适的奇异值截取数目 c 使得 $\mathbf{H}$ 的压缩解压缩误差 err_H 较为理想。注意到每当 s 增大到原来的 2 倍时，需要进行 SVD

分解的总次数会减少一半，因此虽然 $s=16$ 时的平均迭代次数是 $s=8$ 时候的 2 倍，但实际上的总迭代次数是差不多的，而 $s=32$ 显然迭代次数远大于前两者。但是，随着 s 的增大，每次 QR 分解需要的计算复杂度也会快速增加，这也是在选取 s 和 c 时需要考虑的。

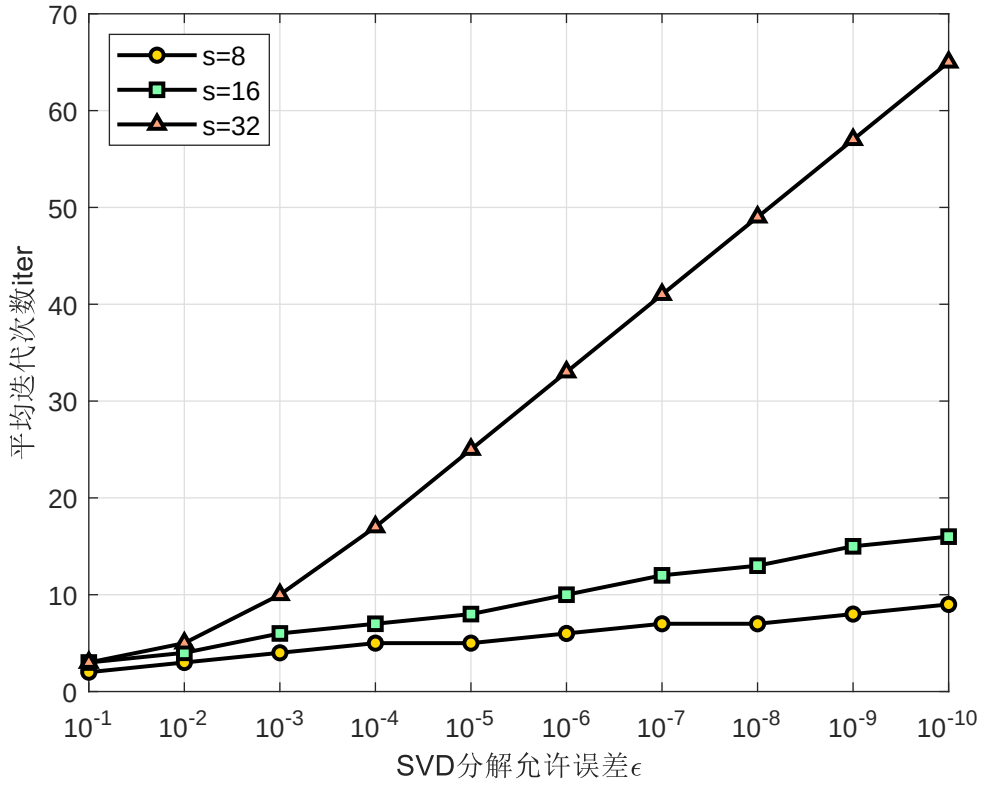

图 8　矩阵合并数目 $s=8,16,32$ 下，不同 SVD 分解允许误差 ϵ 对应的平均迭代次数 $iter$

5.1.3 复杂度分析

本节计算使用算法的存储复杂度以及压缩/解压缩复杂度。原始的矩阵 **H** 是维数为 $M\times N\times J\times K$ 的复数矩阵，其存储复杂度为

$$C_{sav}^{0}=64MNJK \tag{32}$$

经过 SVD 变换压缩后，只需存储 $J\times\left\lceil\frac{K}{s}\right\rceil$ 个存储单元，每个单元包含维数为 $MN\times c$ 的复数矩阵 $\widetilde{\mathbf{U}_{\mathbf{j,k}}}$ 和维数为 $s\times c$ 的复数矩阵 $\widetilde{\mathbf{V}_{\mathbf{j,k}}}$，其存储复杂度为

$$C_{sav}^{svd}=64cJ\left\lceil\frac{K}{s}\right\rceil(MN+s)\,. \tag{33}$$

实际使用时一般使得 s 可以整除 K，故有

$$C_{sav}^{svd}=64JK\,(MN+s)\,\frac{c}{s}, \tag{34}$$

其压缩率为

$$Cr = \frac{C_{sav}^{svd}}{C_{sav}^{0}} = (1 + \frac{s}{MN})\frac{c}{s}. \tag{35}$$

SVD 分解的压缩复杂度可以参考问题一中有关 SVD 分解的复杂度分析。本题中 $m = MN$,$n = s$，假设 QR 分解的迭代次数为 $iter$，则每恢复 s 个矩阵，SVD 分解部分的复杂度为 $s(8MNs + 10MN + 7s + 41)iter$。与一般 SVD 分解相比，本题中我们使用的压缩算法多了一步计算 $\sigma_i \mathbf{v_i}^T$，这一步的复杂度为 $3MNc$，故总压缩复杂度为

$$C_{enc}^{svd} = (s(8MNs + 10MN + 7s + 41)iter + 3MNc)J\left\lceil \frac{K}{s} \right\rceil. \tag{36}$$

对于解压算法，仅需要计算 $\widehat{\mathbf{X}} = \sum_{i=1}^{c} \mathbf{u_i}(\sigma_i \mathbf{v_i}^T)$ 来恢复矩阵。因此，每次为了恢复 $\mathbf{H}$ 中的 s 个矩阵，先计算 $\mathbf{u_i}(\sigma_i \mathbf{v_i}^T)$，需要 $MNsc$ 次复数乘法；最后计算求和，$MNs(c-1)$ 次复数加法，则解压复杂度为

$$C_{dec}^{svd} = (14MNsc + 2MNs(c-1))J\left\lceil \frac{K}{s} \right\rceil = MNJK(16c - 2). \tag{37}$$

5.1.4 存储与压缩/解压复杂度联合优化

基于矩阵的压缩、解压与存储复杂度，可以写出对复杂度的多目标优化如下：

$$\begin{aligned}
\underset{s,c\in N^+}{\text{minimize}} \quad & C_{sav}^{svd} + \frac{1}{2}\left(C_{enc}^{svd} + C_{dec}^{svd}\right) \\
\text{subject to} \quad & \text{err}_H \leq \text{-30dB}, \\
& \text{err}_H = 10\log_{10}\frac{E\left\{\left\|\widehat{\mathbf{H}}_{j,k} - \mathbf{H}_{j,k}\right\|_F^2\right\}}{E\left\{\left\|\mathbf{H}_{j,k}\right\|_F^2\right\}}, \\
& C_{sav}^{svd} = 64cJ\left\lceil \frac{K}{s} \right\rceil (MN + s), \\
& C_{enc}^{svd} = \left(s\left(8MNs + 10MN + 7s + 41\right)iter + 3MNc\right)J\left\lceil \frac{K}{s} \right\rceil, \\
& C_{dec}^{svd} = MNsJ\left\lceil \frac{K}{s} \right\rceil (16c - 2).
\end{aligned} \tag{38}$$

题目要求矩阵的存储复杂度与压缩/解压复杂度具有相同的优先级，故取存储复杂度权重为 1，压缩/解压复杂度各为 1/2。为了让 s 能整除 K，取 $s = 8, 16, 32, ...$。由于不清楚 ϵ 与 err_H 之间的显式关系，我们在不同的 s 和 ϵ 下，选取合适的 c 作为候选，计算得到的各复杂度如表 8 所示。由表可知，SVD 分解允

许误差 ϵ 只会影响压缩复杂度，对且对平均迭代次数 $iter$ 影响较小。s 的取值越大，则在约束内可达到的最小存储复杂度就越小，但压缩复杂度同时也会相应地快速增长，解压复杂度也有小幅增长。此外，压缩复杂度通常比解压复杂度要大 1-2 个数量级，这与前文的理论分析相一致。

表 8　H 矩阵不同 SVD 分解允许误差 ϵ, 矩阵合并数目 s, 奇异值截取数目 c 备选方案的性能比较 (粗体表示与 SVD 分解允许误差 ϵ 无关的性能)

SVD 分解允许误差 ϵ	1e-2					
矩阵合并数目 s	8		16		32	
奇异值截取数目 c	3	4	4	5	6	7
平均迭代次数 $iter$	3	3	4	4	5	5
误差 err_H	-42.06	-45.75	-36.48	-43.64	-33.37	-38.07
压缩复杂度 C_{enc}^{svd}	8.81e+7	8.83e+7	2.18e+8	2.18e+8	5.25e+8	5.25e+8
SVD 分解允许误差 ϵ	1e-3					
矩阵合并数目 s	8		16		32	
奇异值截取数目 c	3	4	4	5	6	7
平均迭代次数 $iter$	4	4	6	6	10	10
误差 err_H	-44.35	-59.72	-37.20	-50.25	-34.15	-39.78
压缩复杂度 C_{enc}^{svd}	1.17e+8	1.18e+8	3.27e+8	3.27e+8	1.05e+9	1.05e+9
解压复杂度 C_{dec}^{svd}	1.81e+7	2.44e+7	2.44e+7	2.18e+8	3.70e+7	4.33e+7
存储复杂度 C_{sav}^{svd}	9.73e+6	1.30e+7	6.68e+6	8.36e+6	5.31e+6	6.19e+6
压缩率 C_r	0.3867	0.5156	0.2656	0.332	0.2109	0.2461

根据表格计算可得，当取 $s = 8$，$c = 3$，$\epsilon = 1e - 2$ 时，目标函数取得最小值，故此为最优存储方案的参数设置。此时存储复杂度为 9.73e6，压缩/解压复杂度分别为 8.81e+7/1.81e+7，此时压缩率 $C_r = 0.3867$，压缩比为 2.59。

5.2 子问题 2——W 矩阵的压缩

W 矩阵的压缩和 H 矩阵压缩的思想类似，将相关性强的向量合并在一起，用较少的奇异值以及该奇异值对应的奇异向量表示。通过问题一的分析，我们知道 $\mathbf{W}_{j,k}$ 由两个几乎正交的列向量组成，可以认为这两个列向量是相互独立的，因此需要将 $\mathbf{W}_{j,k}$ 中的两个列向量分开存储。将同一行块的不同子矩阵的第一个列

向量组合成 $N \times K$ 维的矩阵 $\mathbf{W}_{1,j,K}$，同理，第二个列向量组成的 $N \times K$ 维的矩阵为 $\mathbf{W}_{2,j,K}$，对这两个矩阵采用与类似的基于 SVD 分解相同的矩阵压缩算法完成压缩。

根据 V 矩阵压缩的方案，我们仍然将 W 矩阵的压缩问题优化成压缩复杂度、解压复杂度、存储复杂度的多目标优化问题。采用和矩阵 V 相同的分析方式，我们得到如下的压缩性能表：

表 9　W 矩阵不同 SVD 分解允许误差 ϵ, 矩阵合并数目 s, 奇异值截取数目 c 备选方案的性能比较

SVD 分解允许误差 ϵ	1e-2					
矩阵合并数目 s	8		16		32	
奇异值截取数目 c	3	4	5	6	9	10
W1						
误差 err_W	-32.29	-41.62	-32.53	-37.72	-33.26	-35.95
平均迭代次数 $iter$	4		5		7	
压缩复杂度 C_{enc}^{svd}	2.98e+7		6.91e+7		1.86e+8	
W2						
误差 err_W	-30.82	-40.06	-30.97	-36.40	-31.91	-34.68
平均迭代次数 $iter$	4		5		8	
压缩复杂度 C_{enc}^{svd}	2.98e+7		6.91e+7		2.13e+8	
解压复杂度 C_{dec}^{svd}	4.52e+6	6.09e+6	7.67e+6	9.24e+6	1.40e+7	1.55e+7
存储复杂度 C_{sav}^{svd}	2.65e+6	3.54e+6	2.46e+6	2.95e+6	2.65e+6	2.95e+6
压缩率 C_r	0.4219	0.5625	0.3906	0.4688	0.4219	0.4688

针对 W 矩阵的压缩/解压算法，我们作如下分析。

- W 矩阵的压缩中，列向量的合并数目 S 一般取为 2 的幂次，如 8，16，32，64 等，这样既可以方便矩阵乘法计算，也可以使 K 能够整除 S 从而避免了补 0 的情况。
- 由于我们只需要前 C（$C < S$）个奇异值以及 **U**、**V** 的前 C 列，因此理论上可以用更加快速的方法进行计算。这里由于问题一的求解中分析了快速 SVD 分解的计算方法，故仍使用前面的算法。
- 本题中使用的 C 是通过实际计算估计出的最优结果。由于矩阵之间的列相关

性很高，所以截取位数比 V 矩阵的压缩截取位数要小很多。

- 根据 V 矩阵压缩的算法分析和优化分析，当矩阵合并数目为 8，截取数目为 4 时压缩率为 0.5625，压缩率最高。一共需要压缩 48 次。

6. 问题三的求解

问题三是在问题一的基础上增加了存储复杂度的计算，问题一当中的插值计算只要考虑简单的乘法和加法运算，基本没有运算代价，而本问题中需要综合考虑插值过程中的存储复杂度，相当于减少的运算复杂度转换成存储复杂度。如果采用问题一的线性插值，意味着 V 矩阵的需要存储 2 个相邻的 V 矩阵，需要花费的开销为

$$C_{sav}^{j,k} = 128\text{NL} \tag{39}$$

相比于计算每个 V 矩阵的子矩阵，这个存储复杂度明显小的多，因此插值带来的存储复杂度增加的成本小于插值的收益。而且每次线性插值需要的存储器维度是一样的，如果采用流水线的结构，只需要开销 128NL 的存储复杂度，对每个 (j,k) 都适用，相当于每次插值都是在对这个寄存器组内的数据进行更新。因此在 V 矩阵的计算中仍然采取问题一第一小问的思路，将能够插值计算的子矩阵进行插值处理。

同时计算 V 矩阵时，很多内部变量需要进行存储，比如 SVD 分解中的 QR 迭代过程需要为迭代中的中间变量开辟存储空间，SVD 分解中求 Q 矩阵也需要存储复杂度，但是由于每次迭代的结果会覆盖上次迭代的结果，所以消耗的存储复杂度不会随着迭代次数的增加而增加，因此很多中间变量存储简单的传统算法将重新进行考虑，这意味着我们需要考虑算法的存储复杂度，结合计算复杂度进行联合优化。

在计算 W 矩阵之前需要对 J 个 V 矩阵进行合并求逆，需要的存储复杂度为 64NLJ，AOR 迭代中，每次迭代的结果也要进行存储。但是由于 AOR 的中间变量的维度只有 64NL，因此将 W 矩阵的计算拆分成维度 $N \times L$ 的子矩阵运算具有较少的存储复杂度。

问题三最终可以转换成存储复杂度和计算复杂度联合优化的问题，甚至需要重新考虑一些存储复杂度低的算法，也可以从算法结构的角度进行优化，将迭代算法写成逐个元素优化的算法结构，尽可能在计算复杂度不变的而基础上减少存储复杂度。

参考文献

[1] R. De Leone, R. Capparuccia and E. Merelli, ”A successive overrelaxation back-propagation algorithm for neural-network training,” in IEEE Transactions on Neural Networks, vol. 9, no. 3, pp. 381-388, May 1998.

[2] A. Garcia-Ariza and L. Rubio, ”Computing the Closest Positive Definite Correlation Matrix for Experimental MIMO Channel Analysis,” in IEEE Communications Letters, vol. 15, no. 10, pp. 1038-1040, October 2011.

[3] Hestenes, Magnus R. “Inversion of Matrices by Biorthogonalization and Related Results.” Journal of the Society for Industrial and Applied Mathematics, vol. 6, no. 1, Society for Industrial and Applied Mathematics, 1958, pp. 51–90.

[4] M. Aharon, M. Elad and A. Bruckstein, ”K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation,” in IEEE Transactions on Signal Processing, vol. 54, no. 11, pp. 4311-4322, Nov. 2006.

[5] Tony F. Chan. 1982. An Improved Algorithm for Computing the Singular Value Decomposition. ACM Trans. Math. Softw. 8, 1 (March 1982), 72–83.

[6] 胡乐宇, 蔡邢菊. 低计算精度下实对称矩阵的特征值分解 [J]. 高等学校计算数学学报,2021,43(02):117-133.

[7] 韩露, 史贤俊, 秦玉峰, 林云, 李荣新. 基于相关性矩阵合并算法的系统级测试性建模方法研究 [J]. 兵工自动化,2021,40(08):80-87.

[8] Volker Strassen (Aug 1969). ”Gaussian elimination is not optimal”. Numerische Mathematik. 13 (4): 354–356.

[9] Alman, Josh; Williams, Virginia Vassilevska (2020), ”A Refined Laser Method and Faster Matrix Multiplication”, 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2021).

[10] 网址：https://www.zhihu.com/question/23008550

[11] Volker Strassen (Aug 1969). ”Gaussian elimination is not optimal”. Numerische Mathematik. 13 (4): 354–356.

附录 A MATLAB 源程序

1.1 基于随机 SVD 分解的矩阵降维

```
function [V,rho_V_err,Q,V_real,B,loopcount] = random_svd(A,iter)

%% calculate Q
[m,n]=size(A);
Q=[];
L=2;
for t=1:iter
w=(randn(n,1)+1j*randn(n,1))*sqrt(0.5)*0.01;
y=A*w;
if t==1
q=eye(m)*y;
else
q=(eye(m)-Q*Q')*y;
end
q_norm=norm(q,2);
qn=q./q_norm;
Q=[Q qn];
end

%% B=QA SVD
B=Q'*A;
[U,~,~,loopcount] = svdsim(B,0.001);
[~,~,V_real]=svds(A',2);
QU=Q*U;
V=QU(:,1:2);
rho_V_err=zeros(L,1);
for l=1:L
Vl=V(:,l);
V_reall=V_real(:,l);
rho_V_err(l)= norm(Vl'*V_reall)/norm(Vl)/norm(V_reall);
end
end
```

1.2 基于 QR 分解和双对角化的 SVD 分解

```
function [u,s,v,loopcount] = svdsim(a,tol)
if ~exist('tol','var')
tol=eps*1024;
end

%reserve space in advance
sizea=size(a);
loopmax=100*max(sizea);
loopcount=0;

% or use Bidiag(A) to initialize U, S, and V
u=eye(sizea(1));
s=a';
v=eye(sizea(2));

Err=realmax;
while Err>tol && loopcount<loopmax
%  log10([Err tol loopcount loopmax]); pause
[q,s]=qr(s'); u=u*q;
[q,s]=qr(s'); v=v*q;

% exit when we get "close"
e=triu(s,1);
E=norm(e(:));
F=norm(diag(s));
if F==0, F=1;end
Err=E/F;
loopcount=loopcount+1;
end
% [Err/tol loopcount/loopmax]

%fix the signs in S
ss=diag(s);
s=zeros(sizea);
for n=1:length(ss)
ssn=ss(n);
s(n,n)=abs(ssn);
if ssn<0
```

```
u(:,n)=-u(:,n);
end
end

if nargout<=1
u=diag(s);
end

return
```

1.3 AOR 迭代算法

```
function x = AOR(A,y,k,w,r)

E=-tril(A,-1);
F=-triu(A,1);
D=diag(diag(A));
U=inv(D-r*E);
N2=(1-w)*D+(w-r)*E+w*F;



x=1/1.01*y;

for i=1:size(y,2)
for j=1:k
x(:,i)=U*N2*x(:,i)+w*U*y(:,i);
end
end
end
```

1.4 基于 SVD 分解的压缩和解压缩

```
function [comp,loopcount,err_H] = HC(H,M,N,J,K,snap,cut,thre)
%compression
comp = cell(J,ceil(K/snap));
loopcount = zeros(J,ceil(K/snap));
for j = 1:J
X = zeros(M*N,snap);
```

```
count = 0;kk = 0;
for k = 1:K
HH = H(:,:,j,k);
kk = kk + 1;
X(:,kk) = reshape(HH,[M*N 1]);
if kk==snap || k == K
count = count + 1;
[U,S,V,lc] = svdsim(X,thre);
%[U,S,V] = svd(X);
comp{j,count}.U = U(:,1:cut);
comp{j,count}.S = diag(S(1:cut,1:cut));
comp{j,count}.V = V(:,1:cut);
loopcount(j,count) = lc;%□□□涓□vd杩□唬浜□□灏□□
kk = 0;
end
end
end
%decompression
H_hat = zeros(size(H));
for j = 1:J
for count = 1:size(comp,2)
X_re = comp{j,count}.U * diag(comp{j,count}.S) *
   comp{j,count}.V';
for kk = 1:snap
H_hat(:,:,j,(count-1)*snap+kk) = reshape(X_re(:,kk),[M N]);
end
end
end
%entropy calculation
E = zeros(J,K);
F = zeros(J,K);
for j = 1:J
for k = 1:K
HH = H(:,:,j,k);
H_re = H_hat(:,:,j,k);
E(j,k) = norm(H_re-HH,'fro')^2;
F(j,k) = norm(HH,'fro')^2;
end
end
```

```
err_H = 10*log10(mean(E,'all')/mean(F,'all'));
end
```