中国研究生创新实践系列大赛

“华为杯”第二十届中国研究生数学建模竞赛

学　　校　西安电子科技大学

参赛队号　**No.23107010043**

队员姓名

1. 徐奇琛

2. 王桠鹏

3. 艾珊珊

中国研究生创新实践系列大赛

“华为杯”第二十届中国研究生数学建模竞赛

题　　目　DFT 类矩阵的整数分解逼近

摘　　要:

DFT（离散傅里叶变换）作为通信领域内一项重要的数学工具，在信号处理以及信道估计中已经得到了广泛的应用。当数据范围较大时，一般采用快速傅里叶变换（FFT）来实现 DFT。然而随着天线阵面、通道数、通信带宽的持续增长，在芯片上实现 FFT 所需的计算资源也越来越大。故而要寻求 DFT 矩阵特有的性质，使其在不影响精度的前提下，尽可能的减少计算资源的消耗。目前主要的方法有蝶形运算以及矩阵连乘拟合等。因此需要建立更加简洁高效的 DFT 实现方法，使计算复杂度明显的降低，便于工程实践运用。

本文以 DFT 矩阵的特殊性质为切入点，主要采用矩阵低秩分解逼近思想并结合**广义范德蒙矩阵在复数域上的 1-带宽 LU 分解**的相关理论、**Kronecker 积的性质**以及**改良后的遗传算法**，最终分别解决了不同约束条件以及优化空间下的矩阵乘积分解逼近优化问题。

针对**问题一**: 本题无需考虑非零元素的取值范围，只要求分解后为稀疏矩阵以及在 $RMSE$ 尽可能小的前提下，建立近似计算的**多目标优化**模型对分解矩阵以及参数 β 同时进行优化。针对此问题给出了广义范德蒙德矩阵在复数域上的 1-带宽 LU 低秩分解模型，并据此对 F_N 矩阵进行稀疏分解，将其分解成 2n 个均为 1-带宽的上、下三角矩阵，最终以$\boldsymbol{t=2}$为例可以得到$\boldsymbol{RMSE=2.13\times10^{-15}}$，硬件复杂度$\boldsymbol{C=240}$ 的结果。

针对**问题二**：本题去除了稀疏性要求但是限制了分解矩阵中各元素的取值范围。基于给定的 q 从而确定优化模型的可行域。这使得个体的编码以及遗传算法的实现成为了可能。据此建立了更改约束后的优化模型，并采用**锦标赛选择法（Tournament Selection）**策略的遗传算法进行求解，将原始 DFT 矩阵分解为了符合约束的多个矩阵，以$\boldsymbol{N=2}$ 为例得到了$\boldsymbol{RMSE=0.0059}$、硬件复杂度$\boldsymbol{C=27}$ 的结果。

针对**问题三**: 根据问题要求需要在约束条件 1 和 2 下再次求解。考虑到可行域的离散性以及稀疏性的要求，上一问的算法依然适用。但是由于稀疏性会

导致信息丢失，因此为了保证精度相应的提高了 K 的取值，并且为了满足约束条件，我们调整了其中的交叉和变异操作，具体是让交叉点在每个分解矩阵的第一个元素中随机取值，在变异中我们排除了最后一个基因，并且改变策略为**骰点、“基因敲除”**。最后在$\boldsymbol{t=2,N=4}$ 情况下得到了对应分解矩阵以及$\boldsymbol{RMSE=0.0448}$、硬件复杂度$\boldsymbol{C=114}$ 的结果。

针对**问题四**: 本题中 F_N 由 4 阶和 8 阶的 DFT 矩阵通过 Kronecker 积生成的 32 阶矩阵。我们考虑到 Kronecker 积的性质，避免了对 FN 的直接逼近分解，而是采用**分而治之**的思想。首先分别对 F_{N_1} 与 F_{N_2} 进行分解从而将高阶矩阵分解问题转化成了**低阶矩阵分解问题**。然后利用 Kronecker 积的性质将得到的分解矩阵重组，即可以得到 F_N 的一系列分解矩阵。相较于对 F_N 直接分解而言，在保证 $RMSE$ 足够小的基础上极大的降低了问题的复杂度，最终得到$\boldsymbol{RMSE=0.17678}$ 、$\boldsymbol{C=1038}$。

针对**问题五**：本题中将 q 作为优化变量，将 $RMSE$ 作为约束条件。针对于此，我们首先在定义的小可行域内进行搜索，然后逐步扩大搜索域，从而得到在不同可行域内的最佳方案，从而可以给出 q 的取值范围。由于 q 可变导致的精度浮动问题，我们将 β 作用在分解矩阵之前以尝试提高精度。最终以$\boldsymbol{t=2,q=5}$ 为例，得到$\boldsymbol{RMSE=0.065603,C=285}$。

关键字： 矩阵整数分解；多目标优化；低秩逼近；广义范德蒙矩阵；遗传算法；分治法

目录

1. 问题背景与重述

1.1 问题背景

在工程、科学以及数学等各个领域中，其采用的信号处理常用离散傅里叶变换（DFT）来实现。例如，通信信号处理中，常用 DFT 实现信号的正交频分复用（OFDM）系统的时频域变换，其过程如图1-1所示。而在这一系列过程中，算法复杂度越高、数据取值范围越大，其硬件复杂度就越大，目前在实际产品中，一般采用快速傅里叶变换[4]（FFT）算法来快速实现 DFT，其利用 DFT 变换的各种性质，可以大幅降低 DFT 的计算复杂度。然而，随着无线通信技术的演进，天线阵面越来越大，通道数越来越多，通信带宽越来越大，对 FFT 的需求也越来越大，从而导致专用芯片上实现 FFT 的硬件开销也越大。因此设计 DFT 类矩阵的分解方法[5]，从而降低 DFT 计算的硬件复杂度，对进一步降低芯片资源开销以及芯片高效率处理具有重大的价值与意义。

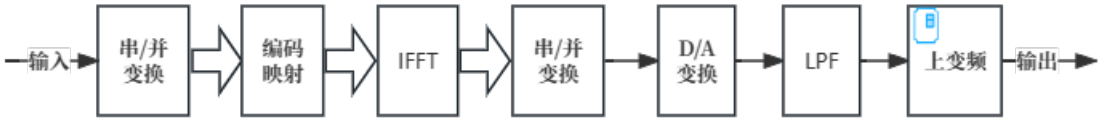

图 **1-1　DFT** 在 **OFDM** 中应用流程图

1.2 问题重述

基于上述背景，本文将在不同约束条件下，研究 DFT 的低复杂度计算方案。

- **问题一：**稀疏性约束下的矩阵分解逼近
  该问题的主要任务是，将 $N=2^t, t=1,2,3,...$ 的 DFT 矩阵 $\mathbf{F}_N$，在满足约束 1 的条件下，即限定 $\mathcal{A}$ 中每个矩阵 A_k 的每行至多只有 2 个非零元素。此时对最优化问题

$$\min_{\mathcal{A},\beta} \mathrm{RMSE}(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\|\beta\mathbf{F}_N-\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_K\|_F^2} \tag{1}$$

  中的变量 $\mathcal{A}$ 和实值矩阵缩放因子 β 进行优化，并计算最小误差和方案的硬件复杂度 C。
- **问题二：**可行域限制下的矩阵分解逼近问题。限制 $\mathbf{A}_k$ 中元素实部和虚部取值范围来减少硬件复杂度该问题的主要任务是，将 $N=2^t, t=1,2,3,...$ 的 DFT 矩阵 $\mathbf{F}_N$，在满足约束 2 的条件下，即限定 $\mathcal{A}$ 中每个矩阵 A_k 满足条件

$$\mathbf{A}_k[l,m]\in\{x+jy\mid x,y\in\mathcal{P}\},\mathcal{P}=\left\{0,\pm1,\pm2,\ldots,\pm2^{q-1}\right\} \tag{2}$$

  其中，$k=1,2,\ldots,K;\quad l,m=1,2,\ldots,N$. 对上述最优化问题 (1) 中的变量 $\mathcal{A}$ 和实值矩阵缩放因子 β 进行优化，并计算最小误差和方案的硬件复杂度 C。
- **问题三：**带有稀疏与可行域约束的矩阵分解问题
  该问题的主要任务是，在同时满足约束条件 1 和 2 的情况下，再次对上述最优化问题 (1) 中的变量 $\mathcal{A}$ 和实值矩阵缩放因子 β 进行优化，并计算最小误差和方案的硬件复杂度 C。
- **问题四：**基于矩阵 Kronecker 积的矩阵分解逼近
  该问题的主要任务是，进一步考虑非 DFT 矩阵 $\mathbf{F}_N=\mathbf{F}_{N1}\otimes\mathbf{F}_{N2}$，其中 $\mathbf{F}_{N1}$ 和 $\mathbf{F}_{N2}$ 分别为 N_1 和 N_2 维的 DFT 矩阵，$\otimes$ 表示 Kronecker 积。当 $N_1=4, N_2=8$ 且在约束条件 1 和 2 的情况下，求此时矩阵 $\mathbf{F}_N$ 分解的优化方案。

- **问题五：**精度限制下的矩阵分解逼近问题
  该问题的主要任务是,在问题三的基础上要求精度限制在 0.1 以内,即 $RMSE \leq 0.1$, 同时满足约束条件 1 和 2 的情况下，设计 DFT 矩阵 $\mathbf{F}_N$ 分解的优化方案。

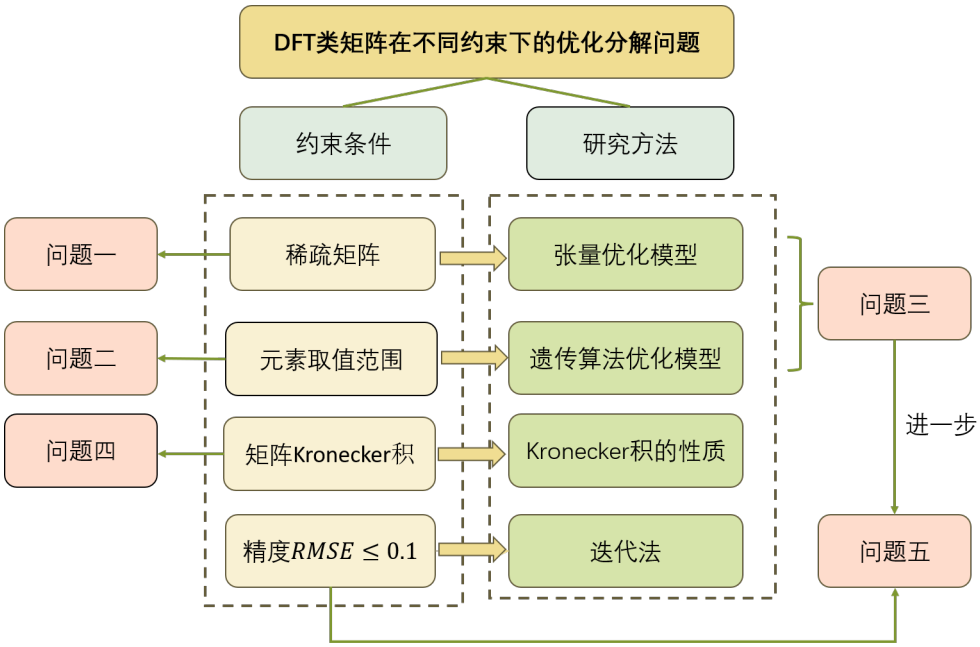

图 **1-2**　思维导图

2. 模型假设

本文提出以下合理假设：
假设 1 当矩阵中某个元素小于截断阈值，可以视其为 0；
假设 2 模型仅仅满足各个题目中要求的约束条件，不考虑其他问题的要求；
假设 3 如若没有特殊符号，矩阵与向量计算均为基础运算。

3. 符号说明

符号	意义
$RMSE$	均方根误差
α	截断阈值
$A_k(l,m)$	分解后得到的第 k 个矩阵的 (l,m) 位置的元素
$\sum_{j=1}^{N} M_k(i,j)$	第 k 个矩阵第 i 行所有非零元素的数量
N	DFT 矩阵的维度
C	乘法器的硬件复杂度
l	复数乘法的次数
K	$\mathcal{A}$ 中矩阵的个数
I	单位阵
A^T	A 的转置
$l_{i,j}^{(m)}$	第 m 个矩阵 (i,j) 位置的元素
$L^{(i)}$	LU 分解中第 i 个 L 矩阵
$U^{(i)}$	LU 分解中第 i 个 U 矩阵
$\mathcal{P}_{min}$	$\mathcal{P}$ 中的最小值
$\mathcal{P}_{max}$	$\mathcal{P}$ 中的最大值
$\mathcal{Q}^i$	第 i 组 β 和分解矩阵 A_k 的集合

4. 问题一: 稀疏性约束下的矩阵分解逼近

4.1 问题一分析

问题一要求通过建模，对给定的 N 维 DFT 矩阵 $\mathbf{F}_N$，适当的选取因子 β 以及一系列稀疏矩阵 $A_1, A_2, ..., A_k$，使得矩阵 $\beta\mathbf{F}_N$ 与 $\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_K$ 在 Frobenius 范数的意义下更接近, 即有目标函数：

$$\min_{\mathcal{A},\beta} \text{RMSE}(\mathcal{A},\beta) = \frac{1}{N}\|\beta\mathbf{F}_N - \mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_K\|_F \tag{3}$$

这里 $\|\cdot\|_F$ 表示 Frobenius 范数。此问题中不需要考虑硬件复杂度问题，即无需考虑矩阵相乘时复数的乘法次数; 且 A_k 与 β 没有取值限制，可行域为整个复数域。在模型建立的过程中，近似矩阵的个数 K 没有明显要求。

因此，问题一所需满足的要求及约束条件如下：

(1) 根据约束条件 1 我们可得，分解后得到的每个矩阵 A_k 的每行最多有两个非 0 元素。为此我们引入了一个新的变量 $M_k(i,j)$，定义如下：

$$M_k(i,j) = \begin{cases} 1, A_k(i,j) \neq 0 \\ 0, A_k(i,j) = 0 \end{cases} \tag{4}$$

故该约束条件可以表示为

$$\sum_{j=1}^{N} M_k(i,j) \leqslant 2, k = 1,2,...K, i = 1,2,...N. \tag{5}$$

(2) 在本问题中，为了得到一系列的稀疏矩阵。我们引入矩阵分解以及稀疏矩阵截断阈值的概念。当矩阵中的某个元素小于设定的截断阈值 α 时，可以视其值为 0，从而达到矩阵稀疏化的目的。例如在矩阵 $A = (a_{ij})$ 中，我们定义

$$b_{ij} = \begin{cases} a_{ij}, a_{ij} > \alpha \\ 0, a_{ij} \leqslant \alpha \end{cases} \tag{6}$$

得到稀疏化的矩阵 $B = (b_{ij})$。其过程如图4-1所示。

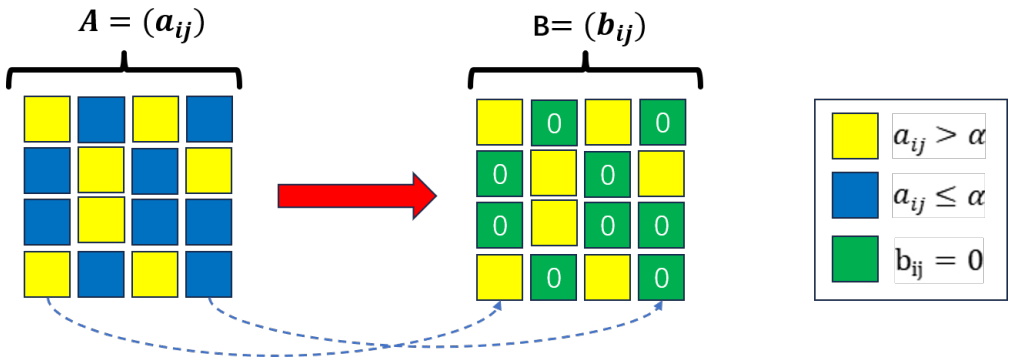

图 **4-1**　截断阈值优化流程图

在第一问不考虑硬件复杂度的前提下，尽可能的提高截断阈值从而提高分解矩阵的稀疏性, 从而得到在此约束条件下的矩阵最优分解方案，并计算出最小误差和方案的硬件复杂度 C。此过程如图4-2所示

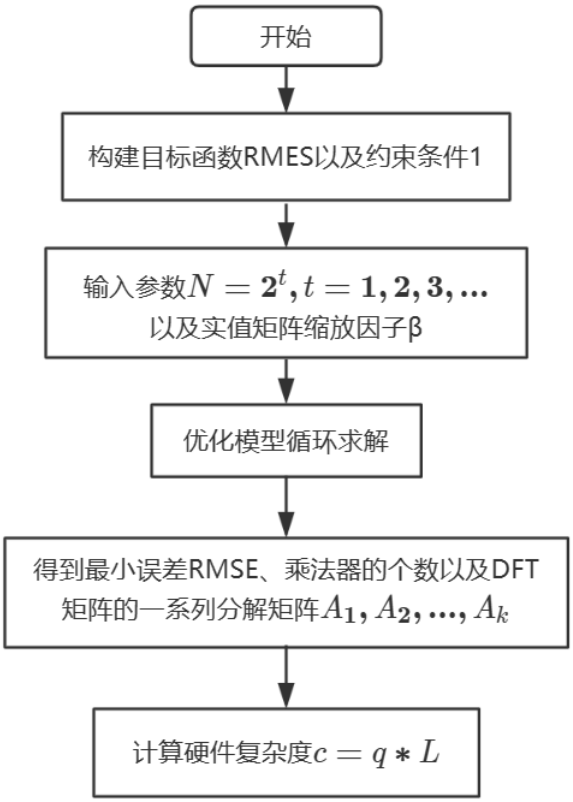

图 **4-2**　问题一流程图

4.2 基于稀疏性要求的张量优化模型

在题目给的目标函数内,其中 $\mathcal{A}$ 代表近似矩阵的集合,我们不妨令 $A_1, A_2, ..., A_k$ 为某三阶张量 $\mathcal{X}$ 前向切片，如图4-4所示, 即根据张量的定义有:$\mathcal{X}_{::k} = \mathcal{X}_k = A_k$, 其中 $\mathcal{X} \in C^{N\times N\times K}$。

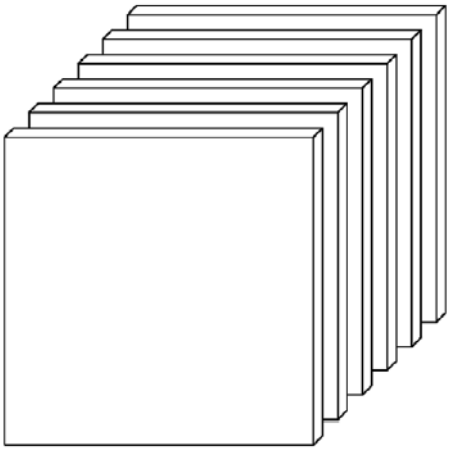

图 **4-4**　张量的前向切片

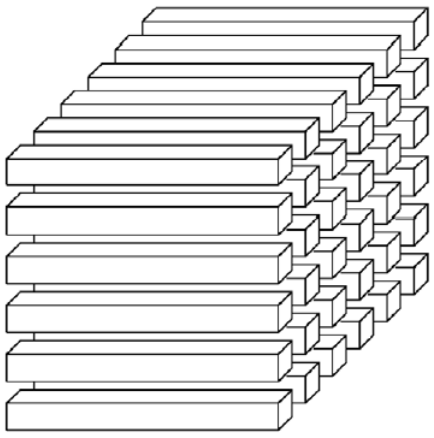

图 **4-5**　张量的模-行纤维

所以目标函数 (3) 可以优化为：

$$\min_{\mathcal{X},\beta} \mathrm{RMSE}(\mathcal{X}, \beta) = \frac{1}{N}\|\beta \mathbf{F}_N - \mathbf{X}_1\mathbf{X}_2\cdots\mathbf{X}_K\|_F \tag{7}$$

这样规定的优势在于对于约束条件

$$\sum_{j=1}^{N} M_k(i,j) \leqslant 2, k = 1,2,...K, i = 1,2,...N. \tag{8}$$

而言，可以利用张量 $\mathcal{X}$ 的行纤维 $\mathcal{X}_{i:k} \leqslant 2$ 来简化，即每个张量的行纤维如图4-5所示。故最终得到的张量优化模型为：

$$\begin{aligned} \min_{\mathcal{X},\beta} \quad & \text{RMSE}(\mathcal{X},\beta) = \frac{1}{N}\|\beta \mathbf{F}_N - \mathbf{X}_1\mathbf{X}_2\cdots\mathbf{X}_K\|_F \\ \text{s.t.} \quad & \sum_{j=1}^{N} M_k(i,j) \leqslant 2, k = 1,2,...K, i = 1,2,...N. \end{aligned} . \tag{9}$$

4.3 广义范德蒙矩阵的分解及模型求解

观察到 DFT 矩阵是以序列$1, \omega, \omega^2, ..., \omega^{N-1}$ 为基础的范德蒙德 (Vandermode) 矩阵。所以对于如此特殊的矩阵，本文优先考虑进行范德蒙德矩阵的分解[1]。不失一般性，考虑复数域上的广义范德蒙德矩阵分解，目的是将其分解成一系列尽可能稀疏的矩阵的乘积。

定义 1 对于 n 维矩阵，如果存在最小正数 m，当满足 $|i-j| \geqslant m$ 时，有 $a_{ij} = 0$，此时 $\omega = 2m-1$ 称为矩阵 A 的带宽。

由于本问题对矩阵内元素的取值以及复数的乘法运算次数无要求，最终的目的是将 F_N 化为一些列稀疏矩阵的乘积，并且使得分解矩阵中的非零元素尽可能的少。容易看出，F_N 不可能分解为一系列对角矩阵的乘积，所以考虑将 F_N 矩阵分解为一系列 1-带宽矩阵的乘积[2]，这样既能保证尽可能的逼近原矩阵，又能得到不错的稀疏性。下面主要给出基于广义范德蒙德矩阵的 LU 分解以及进一步如何将 LU 矩阵继续分解为 1-带宽矩阵的乘积。

定义 2 $\forall k \in R$，$x_0, x_2, ..., x_n$ 为 $n+1$ 个非零且互不相等数，定义如下的矩阵为广义范德蒙德矩阵

$$V_{vd}^{k} = \begin{bmatrix} x_0^k & x_0^{k+1} & \cdots & x_0^{k+n} \\ x_1^k & x_1^{k+1} & \cdots & x_1^{k+n} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^k & x_n^{k+1} & \cdots & x_n^{k+n} \end{bmatrix} \tag{10}$$

定理 1 由于 V_{vd}^{k} 矩阵的前 $n-1$ 个顺序主子式都不为 *0*，所以可以将其进行 LU 分解，其中 L 与 U 为主对角线均为 *1* 的上、下三角矩阵. 其中 L 与 U 中的元素分别如下定义*:*

$$L(i,j) = \prod_{i=0}^{j-1} \frac{x_j - x_{j-t-1}}{x_j - x_{j-t-1}}, 0 \leqslant j \leqslant i \leqslant n \tag{11}$$

$$U(i,j) = \tau_{j-i}(x_0, ..., x_i) \prod_{t=0}^{i-1} (x_i - x_t), 0 \leqslant i \leqslant j \leqslant n \tag{12}$$

其中函数 $\tau(n,r) = \tau_r(x_0, x_1, ..., x_n) = \sum_{1 \leqslant i_1 \leqslant i_2 \leqslant ... \leqslant n} x_{i_1}^{\lambda_1} ... x_{i_n}^{\lambda_n}, \lambda_1 + \lambda_2 + ... + \lambda_n = 1$,$i_1, i_2, ..., i_r$ 代表 n 中的 r 个指标集

接下来利用数学归纳法证明主定理，即

$$A = L^{(1)}L^{(2)}...L^{(n)}U^{(n)}U^{(n-1)}\dots U^{(1)} \tag{13}$$

记 $L^{(1)}L^{(2)}\dots L^{(m)} = \begin{bmatrix} I_{n-m} & 0 \\ 0 & \tilde{L}_{(m)} \end{bmatrix}$

① 当 m=1 时，等式 (13) 明显成立，下面设 m 阶时 S 定理成立，证明 $m+1$ 的情形

②有 $L^{(1)}L^{(2)}\dots L^{(m)}L^{(m+1)} = \begin{bmatrix} I_{n-m} & 0 \\ 0 & \tilde{L}_{(m)} \end{bmatrix} L^{(m+1)} = \begin{bmatrix} I_{n-m-1} & 0 \\ 0 & \tilde{L}_{(m+1)} \end{bmatrix}$

构造 n 阶矩阵 $\begin{bmatrix} I_{n-m-1} & 0^T \\ 0 & \tilde{L}_{(m)} \end{bmatrix}$, 设最后一个相乘的矩阵 $L^{(m+1)} = \begin{bmatrix} I_{n-m-1} & 0^T \\ 0 & B^{(m+1)} \end{bmatrix}$

将上述规定的矩阵代入式 (13) 中，可以得到

$$\begin{bmatrix} I_{n-m-1} & 0^T \\ 0 & \tilde{L}^{(m+1)} \end{bmatrix} = \begin{bmatrix} I_{n-m} & 0 \\ 0 & \tilde{L}_{(m)} \end{bmatrix} \begin{bmatrix} I_{n-m-1} & 0^T \\ 0 & B^{(m+1)} \end{bmatrix} \tag{14}$$

所以只需证明 $L^{(m+1)} = \tilde{L}^{(m)}B^{(m+1)}$ 即可,对比等式左右两端可得 $L^{(m+1)}(i,j) = \tilde{L}^{(m)}B^{(m+1)}(i,j)$

得到 $L^{(i)}, U^{(i)}$ 如下所示：

(1) 当 $0 \leqslant m \leqslant n$ 时，有 $l_{i,j}^{(\mathrm{m})}$ 如下所示:

$$l_{i,j}^{(\mathrm{m})} = \begin{cases} 1, & i = j; \\ \left(\frac{x_i}{x_j}\right) k \prod_{t=0}^{m-n+i-2} \frac{x_i - x_{i-1-t}}{x_{i-1} - x_{i-2-t}}, & i = j+1, i \geqslant n-m+1 \\ 0, & \text{其它}; \end{cases} \tag{15}$$

(2) 对于矩阵 $U^{(m)}$ 而言，当 $0 \leqslant m \leqslant n-1$ 时，有:

$$u_{i,j}^{(m)} = \begin{cases} 1, & i = j, i \leqslant n-m; \\ x_i - x_0, & i = j, i > n-m; \\ x_{m-n+i} \prod_{t=1}^{i} \frac{x_i - x_{i-t}}{x_{i+1} - x_{i+1-t}}, & i = j-1, i \geqslant n-m; \\ 0, & \text{其它} \end{cases} \tag{16}$$

当 $m = n$ 时，有:

$$u_{i,j}^{(n)} = \begin{cases} x_0^k, & i = j = 0; \\ x_i^k(x_i - x_0), & i = j \neq 0; \\ x_i^{k+1} \prod_{t=1}^{i} \frac{x_i - x_{i-t}}{x_{i+1} - x_{i+1-t}}, & i = j-1; \\ 0 & \text{其它}; \end{cases} \tag{17}$$

基于此, 我们定义好了这一系列稀疏矩阵各个位置上的元素。由以上结论可以清楚的看到，分解后矩阵的元素只与开始定义的序列 $x = (x_0, x_1, ..., x_n)$ 以及实数 k 有关。所以针对题目中给的 DFT 矩阵，令 $x = \left(1, \omega, \omega^2, ..., \omega^{(N-1)}, k = 0\right)$ 即可得到。

本问题的求解流程图如图4-5所示。

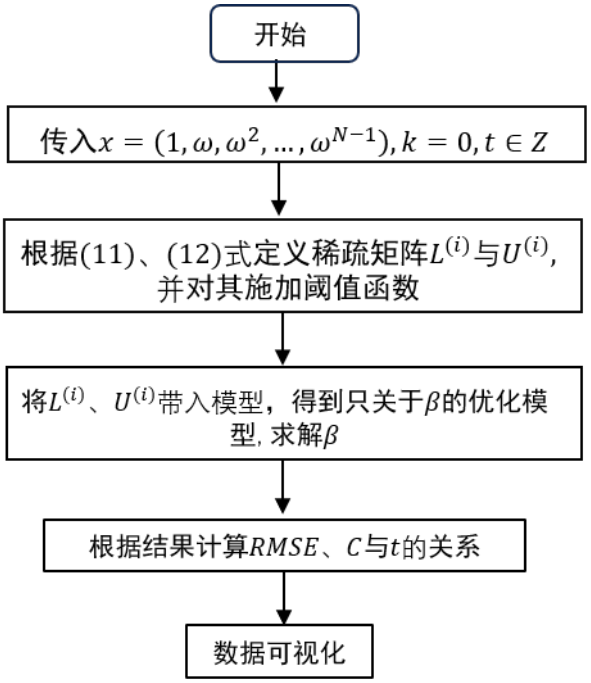

图 **4-5**　问题一求解流程图

4.4 RMSE 及硬件复杂度分析

对于上述问题求解方法利用 MATLAB 进行实现，分别得到了优化模型中复杂度 C 和均方根误差 RMSE 随 t 的变化图像，如图4-6、4-7所示。

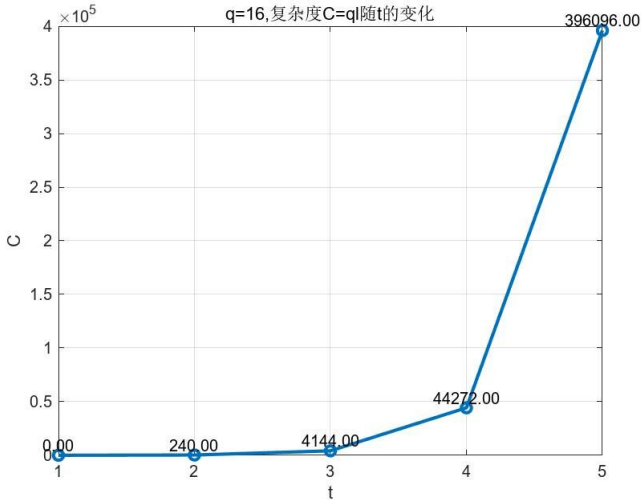

图 **4-6**　复杂度 $C=ql$ 随 t 的变化图像

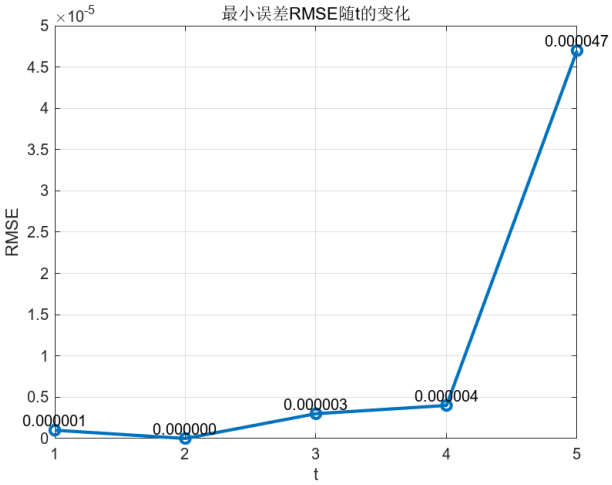

图 **4-7**　最小误差 $RMSE$ 随 t 的变化图像

通过上述图像我们可以看到，当矩阵的规模增大时，所定义的复杂度 C 也随之上升。我们所处理的是 N×N 的矩阵，当 $N=4$ 时，$N^2=16$，此时的复杂度为 240，这优于传统的 DFT 算法，并且最后得到的矩阵分解很好的满足了约束 1 的要求，并在 F-范数意义下取得了几乎可以忽略的误差。当 $t=1,2,3,4,5$ 时的结果如下表 1所示：

表 **1**　问题一最优数据表

t	1	2	3	4	5
N	2	4	8	16	32
$RMSE$	0	2.13×10^{-15}	3.58×10^{-14}	6.74×10^{-11}	4.66×10^{-5}
C	0	240	4144	44272	396096

5. 问题二: 可行域限制下的矩阵分解逼近问题

5.1 问题二分析

问题二与问题一的不同之处在于改变了约束条件，要求分解后得到的矩阵元素的实部和虚部必须必须在可行域 $\mathcal{P}$ 内，但是不再要求必须分解为稀疏矩阵，从而在此基础上优化 $RMSE$ 以及硬件复杂度 C。

由于本问题的可行域是 $\mathcal{P}=\{0,\pm1,\pm2,\pm4\}$，故该优化问题是非凸非连续的带有约束条件的优化问题，目标函数与问题一中的相同。同时应该注意到，问题一中建立的广义范德蒙德矩阵的稀疏分解不再适用，因为其中的元素不在 $\mathcal{P}$ 内，且其分解形式是唯一的 (矩阵的 LU 分解唯一)。且优化过程中应该注意，$\mathcal{A}$ 与 β 的优化优先级应该是相同的，即矩阵与矩阵缩放因子的选取优先级相同。

本章的主要工作是在可行域内设计优化模型，当分解后的矩阵元素在 $\mathcal{P}$ 内时，会极大的减少硬件复杂度。

5.2 在可行域限制下的多目标优化

根据上述问题分析，我们可以得到如下的优化模型：

$$\min_{\mathcal{X},\beta} \mathrm{RMSE}(\mathcal{X},\beta) = \frac{1}{N}\|\beta \mathbf{F}_N - \mathcal{X}_1\mathcal{X}_2\cdots\mathcal{X}_K\|_F \tag{18}$$

$s.t.$

$$\sum_{j=1}^{N} M_k(i,j) \leqslant 2, k = 1,2,...K, i = 1,2,...N. \tag{19}$$

$$\mathcal{P}_{min} \leqslant Re\left(\mathcal{X}_k\left(i,j\right)\right) \leqslant \mathcal{P}_{max} \tag{20}$$

$$\mathcal{P}_{min} \leqslant I_m\left(\mathcal{X}_k\left(i,j\right)\right) \leqslant \mathcal{P}_{max} \tag{21}$$

$$\beta \in R, \mathcal{P} = \{0, \pm 1, \pm 2, \pm 4\}$$

据此，我们定义了一个非凸非连续的约束优化问题。针对这样的问题，本文采用了“交替优化，择优选取”的遗传算法来确定参数 β 以及分解矩阵。

遗传算法[3] 也是一种启发式的搜索算法，其基本思想是模拟生物遗传规律以及达尔文进化论。首先在开始阶段规定一组可行解，然后在基因选择、交叉、变异操作中得到问题在限定条件下的最优解，输出最优方案。其算法设计的示意图如图 5-1所示：

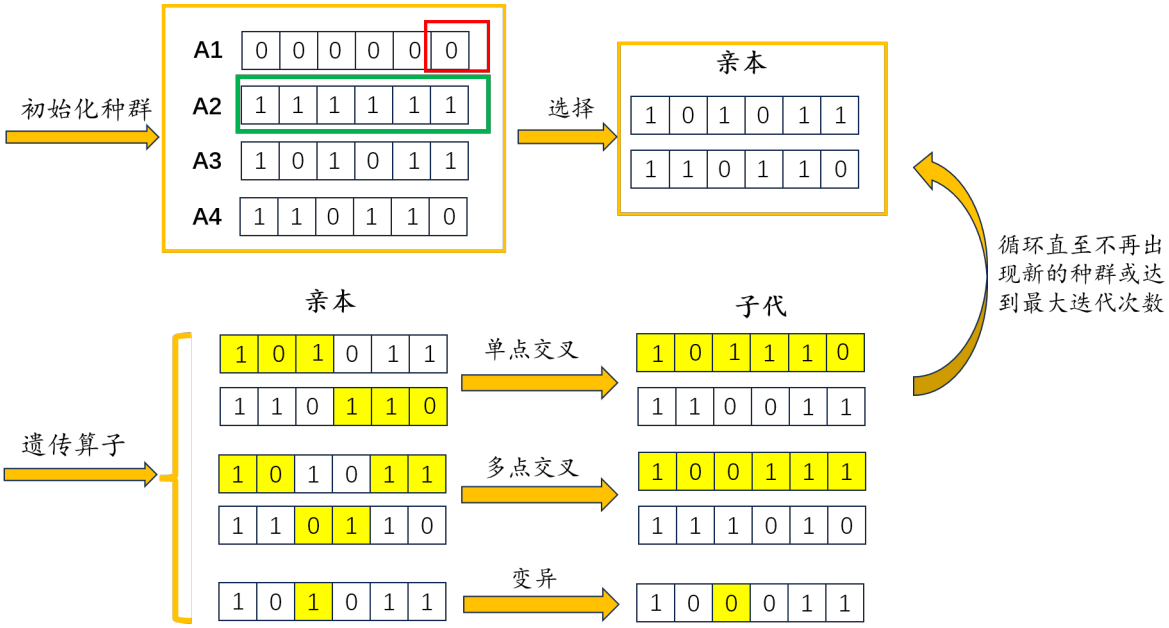

图 **5-1**　遗传算法示意图

在本问题中，由于没有要求分解矩阵的个数 K, 为了简化计算，在算法起初设计的过程中，固定 K 的取值，K 的具体取值与本章的实际情况有关。本问题基于遗传算法设计的基本流程如下：

Step1：随机生成 N 个个体:$\mathcal{Q}^i = (\beta^i, A_1^i, A_2^i, ..., A_K^i), i = 1,2,...,N$，如图5-2所示，并计算适应度，即将随机生成的元素代入目标函数 (22) 中；

Step2：将集合 $\mathcal{Q}^i$ 中的元素按照列生成 $1 \times (3 + KN^2)$ 维的行向量；

Step3：对不同的向量之间进行随机交叉操作，形成新的行向量。并记录 iter

= iter + 1；

Step4：对新的向量再转化为 $(\beta, A_1, A_2, ..., A_N)$, 计算适应度函数，记录下目前最优的种群；

Step5：判断迭代次数与最大迭代次数 max_iter, 若其值 max_iter 大于则停止循环。

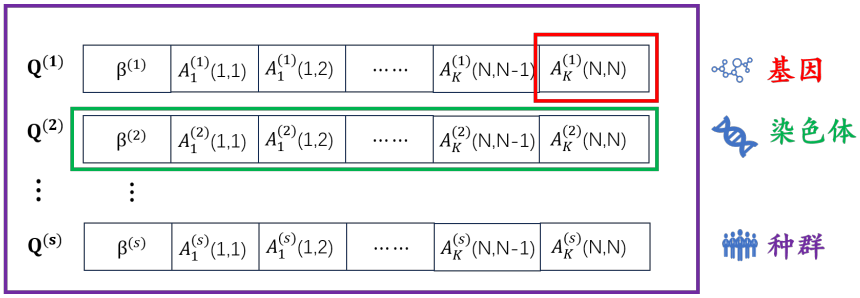

图 **5-2**　初始化种群

遗传算法 (GA) 实现本问题的伪代码如下所示：

Algorithm 1 基于遗传算法的矩阵分解

Input: 矩阵数量 K, 个体数 N,DNT 矩阵 F_N, 最大迭代次数 max_iter
Output: 最优参数 β,$A_1, A_2, ..., A_K$

Initialization：$iter = 0$
$Q = \{\beta, A_1, A_2, ..., A_K\} \leftarrow N$
while $iter \leqslant max\_iter$ **do**
  $\min_{\mathcal{A},\beta} \text{RMSE}(\mathcal{A}, \beta) = \frac{1}{N}\|\beta \mathbf{F}_N - \mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_K\|_F$
  if $f(Q^i) \leqslant f(Q^j)$ **then**
    $Q^* \leftarrow Q^i$
    $f^* \leftarrow f(Q^i)$
  end if
  $iter + 1 \leftarrow iter$
end while
Return：$f(Q^*), Q^* = \{\beta^*, A_1^*, A_2^*, ..., A_K^*\}$

5.3 RMSE 及硬件复杂度分析

针对上述遗传算法模型，我们引用了适应度函数即为本题中的目标函数 (22), 其值在不同 $t(t = 1, 2, 3, 4, 5)$ 下随着迭代次数的增加的变化曲线分别如图5-3所示。由下图我们可以看出遗传算法中前期收敛速度较快，且随着时间的增加，其优化效果越好。

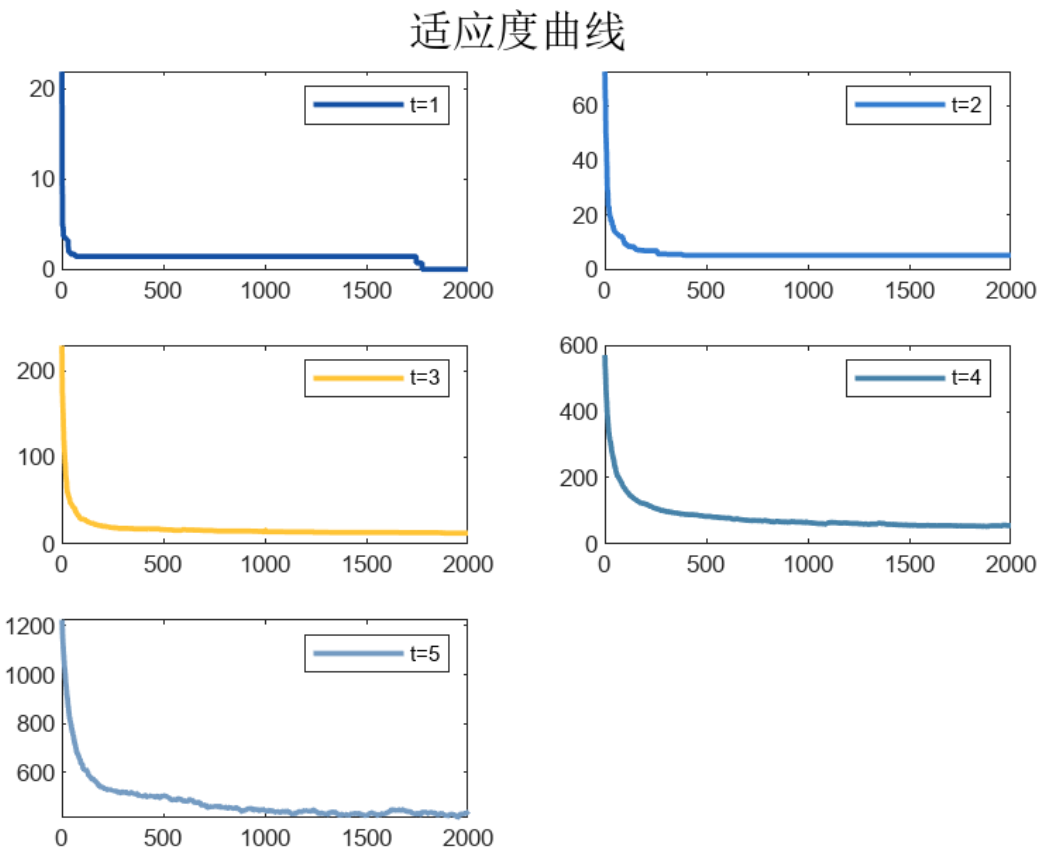

图 **5-3**　问题二求解结果图

通过 MATLAB 编程对上述优化模型进行实现，我们可以得到遗传算法模型中各参数的值，以及不同 t 值下的优化方案，并计算出对应方案下最小误差 $RMSE$ 和硬件复杂度 C，计算结果如表 2 所示。

表 **2**　问题二计算结果表

t	1	2	3	4	5
RMSE	0.0627	0.0957	0.2987	1.5203	1.6719
PopulationSize	150	150	150	150	150
mutationRate	0.004	0.004	0.004	0.004	0.003
crossoverRate	0.8	0.71	0.71	0.71	0.7
tournamentSize	5	20	20	40	40
C	27	348	2226	17724	151155
β	0.0084	0.0032	0.0025	0.0053	0.0018

其中，PopulationSize 表示遗传算法中的种群规模，mutationRate 表示变异率，crossoverRate 表示交叉率，tournamentSize 表示锦标赛抽取人数。并且我们

可以得到在不同 $t(t=1,2,3,4,5)$ 下相应 DFT 矩阵 F_N 在此模型中的优化方案。以 $t=1$ 即 $N=2$ 时为例，我们可以得到其优化的 $\beta=0.0084$ 和分解矩阵 $\mathcal{A}=\{A_1,A_2,...,A_K\}$，其中 $K=3$ 且

$$A_1=\begin{pmatrix}1-i & -i\\ -1+i & i\end{pmatrix}, A_2=\begin{pmatrix}-1-2i & 1-4i\\ i & -2+2i\end{pmatrix}, A_3=\begin{pmatrix}1+i & 1+i\\ -1 & -1\end{pmatrix}$$

在上组优化方案中，我们可以计算得到相应的最小误差 $RMSE=0.0059$ 以及硬件复杂度 $C=0$。

6. 问题三：带有稀疏与可行域约束的矩阵分解问题

6.1 问题三分析

问题三是在问题一和问题二的基础上，同时要求满足两个约束条件。分解后得到的矩阵既要保证其稀疏性也要保证在可行域内取到。在此要求下优化误差 $RMSE$ 以及硬件复杂度 C。

本题中规定了问题的可行域以及约束条件，是一个非凸非连续且带有约束条件的优化问题。目标函数与前两问相同。与问题二类似的是，此问题也需要需要同时优化矩阵缩放因子 β 以及分解矩阵，二者的优化优先级是相同的。

本章的主要工作内容是，融合问题一与问题二的约束条件，在二者的要求下对模型进行优化求解，这样既能保证误差较小，也能保证分解矩阵稀疏，从而减少硬件复杂度。此过程如图6-1所示：

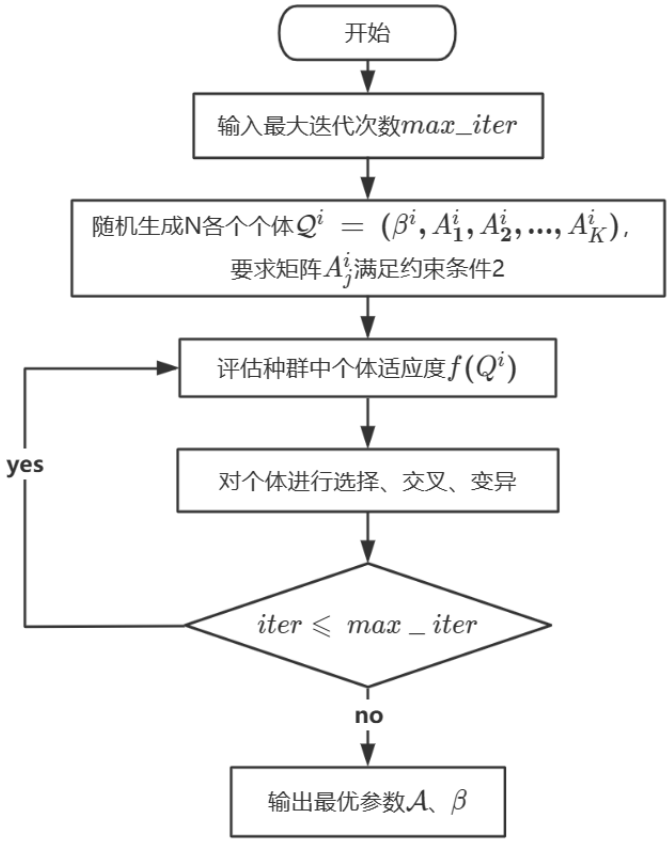

图 **6-1**　问题三流程图

6.2 融合稀疏与可行域要求的优化模型与求解

融合问题一与问题二的模型，得到如下的最优化模型:

$$\min_{\mathcal{X},\beta} \mathrm{RMSE}(\mathcal{X},\beta) = \frac{1}{N}\|\beta \mathbf{F}_N - \mathcal{X}_1\mathcal{X}_2\cdots\mathcal{X}_K\|_F \tag{22}$$

$s.t.$

$$\sum_{j=1}^{j=N} M_k(i,j) \leqslant 2, k = 1,2,...K, i = 1,2,...N. \tag{23}$$

$$\mathcal{P}_{min} \leqslant Re\left(\mathcal{X}_k\left(i,j\right)\right) \leqslant \mathcal{P}_{max} \tag{24}$$

$$\mathcal{P}_{min} \leqslant I_m\left(\mathcal{X}_k\left(i,j\right)\right) \leqslant \mathcal{P}_{max} \tag{25}$$

$$\beta \in R, \mathcal{P} = \left\{0, \pm 1, \pm 2, \ldots, \pm 2^{q-1}\right\}$$

由 (19)、(20)、(21) 式子容易发现，虽然增加了一个约束条件即规定了问题的可行域，但实际上在模型的求解过程中是简化了计算。因为我们在问题二的求解过程中使用遗传算法随机生成的个体是在整个复数域内实现的，随机规模较大。而如今规定了可行域 $\mathcal{P}$，所以沿用问题二的算法，只是在个体元素的选取上要满足稀疏性要求。

本问题求解的基本流程如下:

Step1: 首先生成 $N \times N$ 维的零矩阵 $A = [\alpha_1, \alpha_2, ..., \alpha_N]$，$(\alpha_i)$ 代表行向量；

Step2: 对于任意的 $\alpha_i, i = 1,2,...,N$，在 $\mathcal{P}$ 内随机取值并赋到 $\alpha_i^{(1)}, \alpha_i^{(2)}$；

Step3: 对于矩阵中的每一个 $\alpha_i \in 1 \times N$，随机调换不同位置的元素，生成新的随机个体，同时在实数域内随机生成缩放因子 β；

Step4: 对于由上述步骤随机生成矩阵的方法，随机生成 K 个个体。即优化目标里面要求的矩阵；

Step5: 将生成的 K 个个体代入到遗传算法中，计算适应度函数，进行选择、交叉操作，选择优异个体。

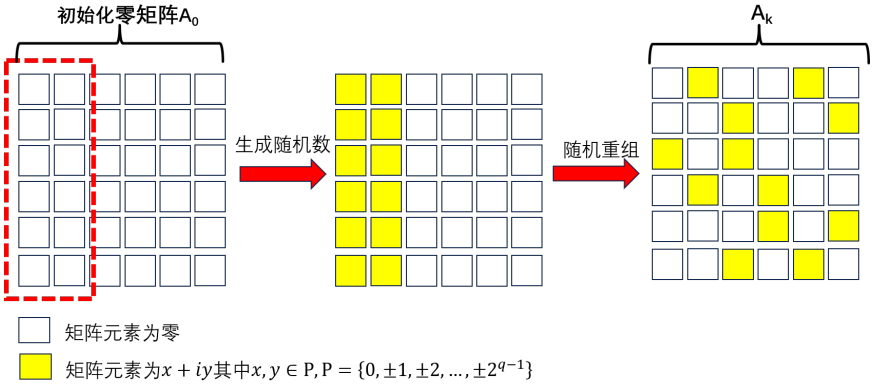

图 **6-2**　生成随机个体示意图

在上述求解过程中应该注意的是:

1.β 的选取与分解矩阵的选取无关，只在实数域内随机生成。

2. 为了防止该算法陷入局部最优解，同问题二一样，同样需要对个体进行变异操作。但与问题二不同之处在于，本题中加入了稀疏约束，所以在变异过程中必须设置符合规定的变异操作，否则会不满足约束条件。针对于此，我们设置了

两种符合约束条件的变异操作，并对这两种操作赋以权重。

(1) 对于个体中的非零元素 $\alpha_i^{i_N}$，在可行域 $\mathcal{P}$ 内随机选取某一值进行替代；

(2) 执行“敲除基因”操作，随机使得个体内的某一“基因”即元素变为 0。

3. 容易发现，在保证稀疏以及取值在可行域内的双重条件下，势必会影响精度，即 RMSE 在一定范围内会上升。所以我们会采取提高分解矩阵数量的方法来逼近 DFT 矩阵[6]，在初步计算中，我们令 K=4 进行算法实现。

6.3 分解矩阵示例及误差、复杂度分析

基于问题二中的遗传算法进行改进，并通过 MATLAB 编程对上述模型进行实现，我们同样可以得到遗传算法模型中各参数的值，以及不同 t 值下的优化方案，并计算出对应方案下最小误差 $RMSE$ 和硬件复杂度 C，计算结果如表3所示。

表 **3**　问题三计算结果表

t	1	2	3	4	5
RMSE	0.5584	0.448	0.3842	0.2528	0.2227
PopulationSize	200	200	200	200	200
mutationRate	0.1	0.1	0.1	0.1	0.1
crossoverRate	0.85	0.85	0.85	0.85	0.85
tournamentSize	15	3	3	5	3
C	33	114	270	399	1182
beta	0.0046	0.0052	0.0048	0.0024	0.0069

并且我们可以得到在不同 $t(t=1,2,3,4,5)$ 下相应 DFT 矩阵 F_N 在此模型中的优化方案。下以 $t=2$ 即 $N=4$ 时为例，我们可以得到其优化的 $\beta=0.0078$ 和分解矩阵 $\mathcal{A}=\{A_1,A_2,...,A_K\}$，其中 $K=4$ 且

$$A_1=\begin{pmatrix}-4+4i & 0 & 0 & -1-4i\\ 0 & -1+i & 0 & -2+i\\ 2-2i & 0 & -1+2i & 0\\ 2-4i & 0 & -1+2i & 0\end{pmatrix}, A_2=\begin{pmatrix}1+i & 0 & -1+2i & 0\\ 4+2i & 0 & 1-2i & 0\\ 1+4i & 0 & -2-i & 0\\ -2-4i & 0 & 0 & -1-i\end{pmatrix},$$

$$A_3=\begin{pmatrix}0 & 0 & 0 & -4\\ 0 & -2-i & 0 & 2+2i\\ 0 & 0 & 0 & -1-2i\\ 1-2i & 0 & 0 & -2+i\end{pmatrix}, \qquad A_4=\begin{pmatrix}0 & 0 & 0 & 0\\ 0 & -2 & 0 & 0\\ 0 & i & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix}.$$

在上组优化方案中，我们可以计算得到相应的最小误差 $RMSE=0.448$ 以及硬件复杂度 $C=114$。

7. 问题四: 基于矩阵 Kronecker 积的矩阵分解逼近

7.1 问题四分析

问题四的约束条件、目标函数均与问题三相同。不同的是，问题四的矩阵不再是由传入的维度 N 决定，而是由两个维度分别为 4 和 8 的 DFT 矩阵做 Kronecker

积得到的矩阵。对此，我们首先给出矩阵 Kronecker 的定义:

设 $X \in C^{m\times n}$,$Y \in C^{p\times q}$, $\otimes$ 表示两个矩阵做表示 Kronecker 积，则有如下表示:

$$\boldsymbol{X}\otimes\boldsymbol{Y}=\begin{pmatrix} x_{11}\boldsymbol{Y} & x_{12}\boldsymbol{Y} & \cdots & x_{1n}\boldsymbol{Y} \\ x_{21}\boldsymbol{Y} & x_{22}\boldsymbol{Y} & \cdots & x_{2n}\boldsymbol{Y} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1}\boldsymbol{Y} & x_{m2}\boldsymbol{Y} & \cdots & x_{mn}\boldsymbol{Y} \end{pmatrix} \in \mathbb{C}^{(mp)\times(nq)}$$

所以由 $F_N = F_{N_1} \otimes F_{N_2}$ 可知，F_N 为 32×32 阶矩阵。对该矩阵进行分析可知，有如下性质:

(1)$F_N F_N^T = I$ 即 F_N 为正交阵；

(2)$F_N = F_N^T$, 即 F_N 为复对称阵；

(3)F_N 为循环矩阵，但不是 DFT 矩阵。

7.2 分治法分解 F_N

注意到问题四引入了 Kronecker 积的概念，最后形成的 F_N 是 32 维的。所以仅仅套用问题三的求解方法不仅会使问题的求解规模过大，也无法利用矩阵 Kronecker 积的性质。所以我们考虑首先对 DFT 矩阵 F_{N_1}, F_{N_2} 进行同问题三的稀疏分解，将其分解为一系列的稀疏矩阵，然后利用 Kronecker 积的性质对分解后的矩阵进行组合。最终会将 F_N 成功分解为一系列稀疏矩阵乘积的形式，此时便可以得到 RMSE 以及复杂度。

基于上述论断，我们首先给出矩阵 Kronecker 的性质

1.$\boldsymbol{X}\otimes\boldsymbol{Y}\otimes\boldsymbol{Z}=(\boldsymbol{X}\otimes\boldsymbol{Y})\otimes\boldsymbol{Z}=\boldsymbol{X}\otimes(\boldsymbol{Y}\otimes\boldsymbol{Z})$

2.$(\boldsymbol{X}+\boldsymbol{Y})\otimes\boldsymbol{Z}=\boldsymbol{X}\otimes\boldsymbol{Z}+\boldsymbol{Y}\otimes\boldsymbol{Z}$

3. 令矩阵 $\boldsymbol{X}\in\mathbb{C}^{m\times n}$, $\boldsymbol{Y}\in\mathbb{C}^{s\times t}$, $\boldsymbol{U}\in\mathbb{C}^{n\times p}$，则有 $(\boldsymbol{X}\otimes\boldsymbol{Y})(\boldsymbol{U}\otimes\boldsymbol{V})=(\boldsymbol{XU})\otimes(\boldsymbol{YV})\in\mathbb{C}^{(ms)\times(pq)}$

本问题求解的初步计算流程如下:

Step1: 利用问题三中的算法分别对 F_{N_1} 与 F_{N_2} 进行稀疏分解，可以得到

$F_{N_1} = F_{N_1}^{(1)} F_{N_1}^{(2)}, ..., F_{N_1}^{(K_1)}$ $\quad$ $F_{N_2} = F_{N_2}^{(1)} F_{N_2}^{(2)}, ..., F_{N_2}^{(K_2)}$($K_1$ 不一定等于 K_2)

近似得到:$F_N = F_{N_1} F_{N_2} \approx (F_{N_1}^{(1)}, F_{N_1}^{(2)}, ..., F_{N_1}^{(K_1)}) \otimes (F_{N_2}^{(1)}, F_{N_2}^{(2)}, ..., F_{N_2}^{(K_2)})$

Step2: 利用矩阵 Kronecker 积的性质，可以得到:

$$F_N = (F_{N_1}^{(1)} \otimes F_{N_2}^{(1)})(F_{N_1}^{(2)} \otimes F_{N_2}^{(2)}), ..., (F_{N_1}^{(K_1)} \otimes F_{N_2}^{(K_2)})$$

Step3: 令 $A_1 = F_{N_1}^{(1)} \otimes F_{N_2}^{(1)}, A_2 = F_{N_1}^{(1)} \otimes F_{N_2}^{(2)}, ..., A_K = F_{N_1}^{(K_1)} \otimes F_{N_2}^{(K_2)}$

上述计算流程的示意图如下所示：

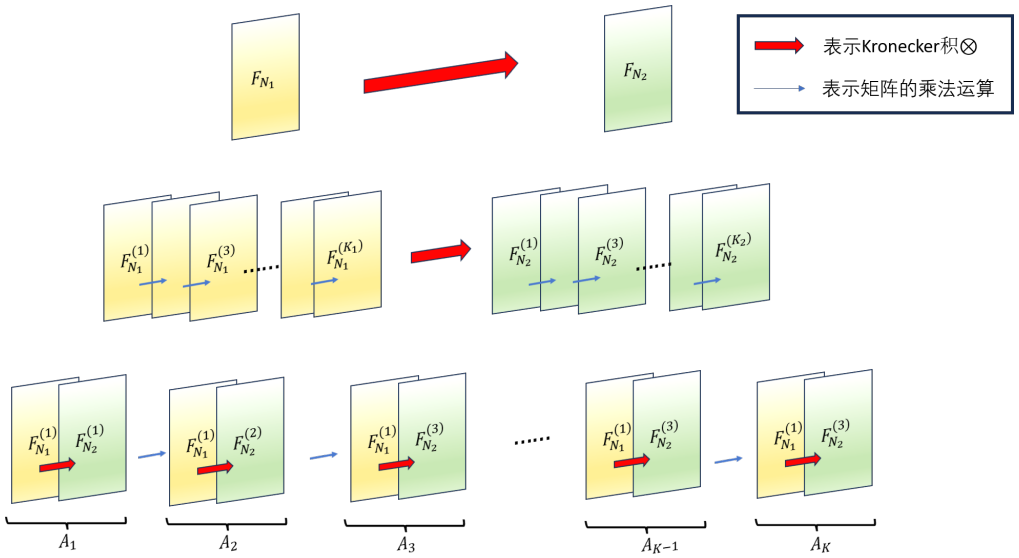

图 **7-1**　问题四示意图

7.3 分治优化模型

从而我们建立了如下的最优化模型
融合问题一与问题二的模型，得到如下的最优化模型:
分治优化模型:

$$\min_{\mathcal{X},\beta_1} \mathrm{RMSE}_1(\mathcal{X},\beta_1) = \frac{1}{N}\|\mathbf{F}_N - \beta_1 \mathcal{X}_{N_1}\mathcal{X}_{N_2}\cdots\mathcal{X}_{K_1}\|_F \tag{26}$$

$$\min_{\mathcal{X},\beta_2} \mathrm{RMSE}_2(\mathcal{X},\beta_2) = \frac{1}{N}\|\mathbf{F}_{N_2} - \beta_2 \mathcal{X}_{N_2}\mathcal{X}_{N_2}\cdots\mathcal{X}_{K_2}\|_F \tag{27}$$

主优化模型:

$$\min_{\mathcal{X},\beta} \mathrm{RMSE}(\mathcal{X},\beta) = \frac{1}{N}\|\mathbf{F}_N - \beta \mathcal{X}_1\mathcal{X}_2\cdots\mathcal{X}_K\|_F \tag{28}$$

$s.t.$

$$\sum_{j=1}^{N} M_k(i,j) \leqslant 2, k = 1,2,...K, i = 1,2,...N. \tag{29}$$

$$\mathcal{P}_{min} \leqslant Re\left(\mathcal{X}_k\left(i,j\right)\right) \leqslant \mathcal{P}_{max} \tag{30}$$

$$\mathcal{P}_{min} \leqslant I_m\left(\mathcal{X}_k\left(i,j\right)\right) \leqslant \mathcal{P}_{max} \tag{31}$$

$$\beta \in R, \mathcal{P} = \left\{0, \pm 1, \pm 2, \ldots, \pm 2^{q-1}\right\}$$

在上述的算法中，虽然初步对 F_{N_1}, F_{N_2} 分解是满足两个约束条件的。但是在 Step2 矩阵组合的过程中，并不能保证稀疏性，也无法保证取值均在可行域内。这就告诉我们直接进行矩阵的组合作为最后分解的 $A_1, A_2, ..., A_K$ 是不可行的。

针对于此，我们首先考虑将 F_{N_1}, F_{N_2} 分解完毕利用 Kronecker 积进行组合之后，对各个组合中的 $(F_{N_1}^{(i)} \otimes F_{N_2}^{(i)})$ 进行验证，观察其是否满足于约束 1 与约束 2，若不满足，则重新对 F_{N_1}, F_{N_2} 进行分解，直到满足约束即可。从而有算法流程如下:

Algorithm 2 Kronecker 积下的矩阵分解逼近问题

Input: 矩阵数量 K_1, K_2, 可行域 $q = 3$,t
Output: $F_{N_1}^{(1)}, F_{N_1}^{(2)}, \cdots, F_{N_1}^{(K_1)}$ $F_{N_2}^{(1)}, F_{N_2}^{(2)}, \cdots, F_{N_2}^{(K_2)}$,
$RMSE$, 硬件复杂度 C

Initialization：
while $flag$ **do**
  $F_{N_1} \approx F_{N_1}^{(1)} F_{N_1}^{(2)} \cdots F_{N_1}^{(K_1)}$
  $F_{N_2} \approx F_{N_2}^{(1)} F_{N_2}^{(2)} \cdots F_{N_2}^{(K_2)}$
  $A_i = F_{N_1}^{(i)} \otimes F_{N_2}^{(i)}$
  if A_i 满足约束条件 (1) 和 (2) **then**
    $Return$
  else
    $flag = 1$
  end if
end while
Return： $A_1, A_2, \cdots, A_K$,β

7.4 结果分析

根据上述模型，利用 MATLAB 编程进行实现，得到的优化方案结果如下表4所示：

表 **4** 问题四计算结果表

$RMSE$	C	$RMSE_1$	$RMSE_2$	β_1	β_2
0.17678	1038	0.018242	0.19291	0.036483	0.545632

其中 $RMSE_1$，$RMSE_2$，β_1，β_2 分别是 DFT 矩阵 F_{N_1}、F_{N_2} 在问题三的模型中进行矩阵分解，而产生的均方根误差和参数 β。

由上表的数据结果我们可以发现，此模型中硬件复杂度 $C = 1038$, 其值相对较大，这是由于模型将问题中两个 DFT 矩阵的 Kronecker 积转化为 $K_1 \times K_2$ 个矩阵相乘的形式，因此导致了硬件复杂度较高的情况；但是其均方根误差 $RMSE = 0.17678$, 其值相对较小，表明此模型在约束条件下的优化效果好，且在实际分解过程中，运行速度也比直接分解矩阵 F_N 快很多，这很直观的体现了矩阵分解优化模型的优越性。

8. 问题五: 精度限制下的矩阵分解逼近问题

8.1 问题五分析

问题五是在问题三的基础上，将问题三的目标函数 $RMSE$ 转化为约束条件 $RMSE \leqslant 0.1$。并且将可行域 $\mathcal{P}$ 作为新的约束目标，在同时满足约束条件 1，2 以及 $RMSE \leqslant 0.1$ 的情况下，适当的选取分解矩阵 A_k 以及缩放因子 β，从而计算方案的硬件复杂度 C。

根据如上分析，我们的思路是定义好分解矩阵 A_k 中的复数乘法次数。然后将求解方案的硬件复杂度的最小值作为目标函数。同时根据搜索算法的步骤，首

先将 $\mathcal{P}$ 定义为较小的范围，然后在这个小范围中寻求最优解。接着将 $\mathcal{P}$ 的范围扩大，再进行搜索，并且将根据不同可行域所得到的不同复杂度进行比较，从而确定最优硬件复杂度与可行域 P。

8.2 以复杂度为目标的优化模型建立与求解

定义 l_k 为 A_k 中复数的乘法运算次数 (不包含与 $0,-1,1,j,-j$ 相乘）。从而有硬件复杂度 $C = q \times L = q \times (\sum_{k=1}^{K} l_k)$，从而我们建立如下优化模型：

$$\min_{\mathcal{A},\beta} q \times L \tag{32}$$

$s.t.$

$$\text{RMSE}(\mathcal{A},\beta) = \frac{1}{N}\|\mathbf{F}_N - \beta\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_K\|_F \leqslant 0.1 \tag{33}$$

本问题求解的基本流程如下:

Step1: 初始化 $\mathcal{P} = \{0, \pm 1, ..., \pm 2^{q-1}\}, q = 1$, 输入 t 确定 DFT 维度；
Step2: 利用问题三算法对 F_N 进行分解，得到 $F_N \approx A_1 A_2 \cdots A_K$；
Step3: 计算并记录均方误差 RMSE 以及硬件复杂度；
Step4: $q = q + 1$, 若 $q \leqslant 5$, 则返回 Step1, 否则跳出循环。
求解此问题的伪代码如下所示：

Algorithm 3 针对不同 β 下的搜索算法

Input: 矩阵数量 K, 矩阵维度 N,q
Output: 最小误差 $RMSE_q$,β, 硬件复杂度 C

Initialization： $q = 1$
$Q = \{\beta, A_1, A_2, ..., A_K\} \leftarrow N$
while $q \leqslant 5$ **do**
  $RMSE \leftarrow \frac{1}{N}\|\beta\mathbf{F}_N - \mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_K\|_F$
  $L \leftarrow \sum_{k=1}^{K} l_k$
  $q + 1 \leftarrow q$
end while
Return： $RMSE, C$

8.3 硬件复杂度结果分析

表 **5** 均方根误差 **RMSE**

RMSE	t=1	t=2	t=3
q=1	0.047169	0.004929	0.031006
q=2	0.079024	0.030654	0.013611
q=3	0.057126	0.052283	0.037052
q=4	0.3904	0.027356	0.031549
q=5	0.4323	0.065603	0.094709

表 **6** 复杂度 **C**

C	t=1	t=2	t=3
q=1	3	10	39
q=2	12	146	272
q=3	36	249	296
q=4	48	292	416
q=5	60	285	400

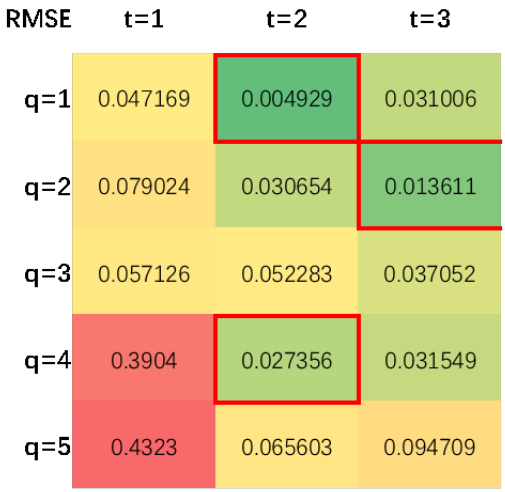

图 **8-2** **RMSE** 的权重图

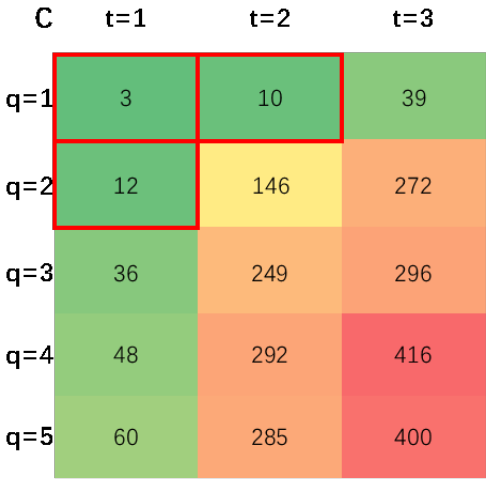

图 **8-3** 复杂度 **C** 的权重图

根据表 5 与表 6 得到的求解数据，我们绘制了图 8.2 和图 8.3 的权重图。我们将 RMSE 与 C 从小到大排列，分别对应绿色到黄色。可以从图上清晰的看到数值的分布。当 t 越大，q 越大时，RMSE 与 C 明显增大。且矩阵元素的取值范围对 RMSE 的影响明显要大于矩阵的维度。

9. 模型的评价与改进

9.1 模型的优点

1. 我们针对 DFT 矩阵的特殊形式，利用范德蒙矩阵的特殊性质，将其直接分解为一系列稀疏矩阵的乘积，节省了计算资源。最后验证 RMSE 与计算复杂度均符合预期。

2. 针对问题三要求的两个约束条件，采用遗传算法计算时。我们设置了“骰点算法”来满足变异操作，不仅能避免模型陷入局部最优解，同样可以使变异后的个体仍然满足约束。

3. 在问题四中，我们并没有对 F_N 直接进行低秩逼近。而是首先对两个维度小的矩阵进行分解然后利用 Kronecker 积的性质将其整合为 F_N 对应的分解矩阵。这样做就避免了直接对 32 阶的 DFT 矩阵分解，极大的减少了计算量。

9.2 模型的缺点

1. 在对范德蒙矩阵的分解过程中，没有采用优化求解，这样导致求得的稀疏矩阵可能不是最好的近似。

2. 在对于分解矩阵数量 K 的研究中，我们并没有将 K 加入到优化变量中，而只是根据不同的情况自行规定 K。这会导致模型可能达不到全局最优解。

3. 分解矩阵是由 $\left[1,\omega,\omega^2,\cdots,\omega^{N-1}\right]$ 直接生成的，这导致初始化此矩阵需要相对较多的时间。

9.3 模型的改进与推广

1. 问题一中采用的是矩阵的确定分解，可以尝试求解建立的张量优化模型，利用坐标下降法等数值优化方法，会得到更多的可行解，从而可能会有更好的效果。

2. 在本论文中，对于分解矩阵数量 K 的研究仅仅是对于不同的优化问题根据实际情况确定 K 的取值，而没有把 K 也作为一个优化变量。在下一步的研究中，可以尝试将 K 也作为决策变量，从而建立多目标优化模型。

3. 在第四问涉及到 Kronecker 积时，我们利用 Kronecker 积的性质对分解后的矩阵进行重组，然后为了满足约束条件多次循环。可以尝试在矩阵重组过程中，设立优化模型，从而无需循环保证约束条件。

4. 第五问中，我们的方法是逐步扩大搜索域。可以尝试将 $\mathcal{P}$ 作为决策变量，建立关于可行域 $\mathcal{P}$ 的优化模型。

10. 参考文献

[1] 梁俊平. 广义范德蒙矩阵的显式 LU 分解定理的构造法证明 [J]. 龙岩学院学报,2008(03):11-14.DOI:10.16813/j.cnki.cn35-1286/g4.2008.03.004.

[2] 张虎. 关于广义范德蒙德矩阵的显式 LU 分解探讨 [J]. 旅游纵览 (行业版),2012(12):136.

[3] 韩中庚. 数学建模方法及其应用 [M]. 高等教育出版社, 2005.

[4] James W. Cooley and John W. Tukey, An Algorithm for the Machine Calculation of Complex Fourier Series, Mathematics of Computation, vol. 19, no. 90, pp. 297-301, 1965. DOI:10.2307/2003354.

[5] K. R. Rao, D. N. Kim, and J. J. Hwang, Fast Fourier Transform: Algorithms and Applications, Springer, 2010.

[6] Viduneth Ariyarathna, Arjuna Madanayake, Xinyao Tang, Diego Coelho, et al, Analog Approximate-FFT 8/16-Beam Algorithms, Architectures and CMOS Circuits for 5G Beamforming MIMO Transceivers, IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 8, no. 3, pp. 466-479, 2018. DOI: 10.1109/JETCAS.2018.2832177.

附录 A 程序代码

1.1 初始化操作

```
%硬件复杂度计算
function C=computeComplexity(M1,M2)
n=size(M1,1);C=0;
for i=1:n
for j=1:n
for k=1:n
if isInclude(M1(i,k),M2(k,j))
C=C+1;
end
end
end
end
end

function flag=isInclude(a,b)
if any([a,b].^2==1) || any([a,b]==1+1i) || any([a,b]==1-1i) ||...
any([a,b]==-1+1i) || any([a,b]==-1-1i) || any([a,b]==0)
flag=0;
else
flag=1;
end
end


  %DFT函数定义
  function F = dftmtx(N)
  F = zeros(N, N);
  omega = exp(-2j * pi / N); % 复数单位根

  for n = 0:N-1
  for k = 0:N-1
  F(n+1, k+1) = omega^(n*k);
  end
  end

  F = 1 / sqrt(N) * F; % 归一化
  end



  %% 绘图代码
  clear;clc;close all;
  %% 复杂度比较
  % 输入序列的长度 N
  t=1:7;
  N=2.^t;

  % DFT 算法的复杂度 C
  C_DFT = N.^2;
  % FFT 算法的复杂度 C，这里假设为 O(N log N)
```

```
C_FFT = N .* log2(N);
figure1=figure;
% 绘制折线图
plot(N, C_DFT, '-o', 'LineWidth',2,'Color','#6699CC'); %
    自定义颜色为淡蓝色
hold on;
plot(N, C_FFT, '-o', 'LineWidth',2,'Color','#CC3333'); %
    自定义颜色为深蓝色
hold off;

% 添加标签和标题
xlabel('输入序列的长度 (N)');
ylabel('复杂度 (C)');
title('DFT vs FFT 算法的复杂度');

% 添加图例
legend('DFT', 'FFT');
saveas(figure1, 'complexity_plot.png');

%% 问题一复杂度
filename = 'C_results.txt'; % 文件名
fileID = fopen(filename, 'r'); % 打开文件以读取模式

% 读取文件中的数据
data = textscan(fileID, 't=%d, RMSE=%.6f, C=%f');
t_values = data{1}; % 提取t的值
RMSE_values=data{2};% 提取RMSE
C_values = data{3}; % 提取C的值
fclose(fileID);

figure2=figure;
plot(t_values,C_values,'o-','LineWidth',2)
xlabel('t');
ylabel('C');
title('q=16,复杂度C=ql随t的变化');
xticks(t_values);
grid on;
% 在图上显示每个数据点的数值
for i = 1:length(t_values)
text(t_values(i), C_values(i), sprintf('%.2f', C_values(i)),
    'HorizontalAlignment', 'center', 'VerticalAlignment',
    'bottom');
end
saveas(figure2, '复杂度C=ql随t的变化.png');

figure3=figure;
plot(t_values,RMSE_values,'o-','LineWidth',2)
xlabel('t');
ylabel('RMSE');
title('最小误差RMSE随t的变化');
xticks(t_values);
grid on
% 在图上显示每个数据点的数值
for i = 1:length(t_values)
```

```
text(t_values(i), RMSE_values(i), sprintf('%f',
    RMSE_values(i)), 'HorizontalAlignment', 'center',
    'VerticalAlignment', 'bottom');
end
saveas(figure3, '最小误差RMSE随t的变化.png');

%% 问题二 t-iter-bestFitness

iter=1:2000;
data=xlsread('results.xlsx',1,'B12:F2011');
figure4=figure;
subplot(3,2,1);
plot(iter,data(:,1),'Color','#1450A3','LineWidth',1.8);legend('t=1');
hold on
subplot(3,2,2);
plot(iter,data(:,2),'Color','#337CCF','LineWidth',1.8);legend('t=2');
subplot(3,2,3);
plot(iter,data(:,3),'Color','#FFC436','LineWidth',1.8);legend('t=3');
subplot(3,2,4);
plot(iter,data(:,4),'Color','#4682A9','LineWidth',1.8);legend('t=4');
subplot(3,2,5);
plot(iter,data(:,5),'Color','#749BC2','LineWidth',1.8);legend('t=5');
sgtitle('适应度曲线');

% set(figure4, 'Position', [100, 100, 800, 1000]);
saveas(figure4, '适应度曲线.png');
```

1.2 问题一

```
clear;clc;
%% 数据初始化
% k=2;
% n=4;
% x=[1.09,2.41,3.321,4.55,5.767];
% V=Vander_G(n,k,x);
q=16;
t=5;
t_pre=t;%由于下面将t作为了循环变量，故这里再储存一个t
k=0;
N=2^t;
n=N-1;
omega = exp(-2j * pi / N); % 复数单位根
x=zeros(n);
for i=0:n
x(i+1)=omega^i;
end
V=Vander_G(n,k,x);

l=cell(n,1);%L子分解
u=cell(n,1);%U子分解
for i=1:n
l{i}=zeros(n+1);
end
```

```
for i=1:n
u{i}=zeros(n+1);
end

%% 1-带宽 LU分解实现
L=eye(n+1);
for m=1:n
for i=1:n+1
for j=1:n+1
if i==j
l{m}(i,j)=1;
elseif i==j+1 && i>n-m+1
l{m}(i,j)=(x(i)/x(j))^k;
for t=0:m-n+i-3
l{m}(i,j)=l{m}(i,j)*(x(i)-x(i-1-t))/(x(i-1)-x(i-2-t));
end
end
end
end
l{m}=Threshold(l{m});
end

U=eye(n+1);
for m=1:n-1
for i=1:n+1
for j=1:n+1
if i==j && i<=n-m+1
u{m}(i,j)=1;
elseif i==j && i>n-m+1
u{m}(i,j)=x(i)-x(n-m+1);
elseif i==j-1 && i>=n-m+1
u{m}(i,j)=x(m-n+i);
for t=1:m-n+i-1
u{m}(i,j)=u{m}(i,j)*(x(i)-x(i-t))/(x(i+1)-x(i+1-t));
end
end

end
end
u{m}=Threshold(u{m});
end

for i=1:n+1
for j=1:n+1
if i==j && i==1
u{n}(i,j)=x(1)^k;
elseif i==j && i~=1
u{n}(i,j)=x(i)^k*(x(i)-x(1));
elseif i==j-1
u{n}(i,j)=x(i)^(k+1);
for t=1:i-1
u{n}(i,j)=u{n}(i,j)*(x(i)-x(i-t))/(x(i+1)-x(i+1-t));
end
end
```

```
end
end

%% 计算复乘次数
complexity = 0;

% 计算 l{1} * l{2} * ... * l{n} 的复杂度
L=l{1};
for i = 2:n
complexity = complexity + computeComplexity(L,l{i});
L=L*l{i};
end

% 计算 u{n} * u{n-1} * ... * u{1} 的复杂度
U=u{n};
for i = n-1:-1:1
complexity = complexity + computeComplexity(U,u{i});
U=U*u{i};
end

complexity=complexity+computeComplexity(L,U);

% 最终的复杂度为复杂度计数器乘以常数 q
C = complexity * q;
disp('L*U=');disp(L*U);
disp(['复杂度: ' num2str(complexity)]);

%% 计算最小误差
options=optimoptions('fmincon', 'Algorithm', 'sqp', 'TolCon',
   1e-6);
[best_beta,min_RMSE]=fmincon(@(x)norm_F(x,V,L*U),1,[],[],[],[],[],[],[],option
disp(['best_beta=' num2str(best_beta)]);
disp(['min_RMSE=' num2str(min_RMSE)]);
disp(['C=' num2str(C)]);

% %保存当前t的复杂度、最小误差，不用时可注释
% filename = 'C_results.txt'; % 文件名
% fileID = fopen(filename, 'a'); % 打开文件以追加写入模式
% fprintf(fileID, 't=%d, RMSE=%.6f, C=%f\n', t_pre, min_RMSE,
   C);
% fclose(fileID);

%% 函数定义
%生成范德蒙德矩阵
function V=Vander_G(n,k,x)
V=zeros(n+1);
for i=1:n+1
for j=1:n+1
V(i,j)=x(i)^(k+j-1);
end
end
end

%优化目标函数
```

```
function f = norm_F(x,F_N,LU)
beta=x;
f=norm(beta*F_N-LU,'fro');
end

%阈值函数
function f=Threshold(M)
for i = 1:numel(M)
if abs(M(i))<1e-8
M(i)=0;
end
end
f=M;
end
```

1.3 问题二

```
clear; clc;
%% 数据初始化
t = 3;
N = 2^t; % 矩阵阶数
F_N = dftmtx(N); % 生成DFT矩阵

k = 3; % 分解产生的矩阵数

q = 3; % 元素范围
% 随机种子序列
seeds = zeros(1, 2*q+1);
for i = 1:q
seeds(i) = 2^(i-1);
seeds(q+i) = -2^(i-1);
end
rand_seeds = zeros(2*q+1);
for i = 1:length(seeds)
for j = 1:length(seeds)
rand_seeds(i, j) = seeds(i) + 1i * seeds(j);
end
end
rand_seeds = reshape(rand_seeds, 1, (2*q+1)^2);

% 遗传算法参数设置
populationSize = 150;  % 种群数量
chromosomeLength = k*N*N+1; % 染色体长度
max_iter = 2000;        % 最大迭代次数
mutationRate = 0.01;  % 变异率
crossoverRate = 0.78;  % 交叉率
tournamentSize =10;    % 锦标赛选择法中每轮锦标赛的参与个体数

%% 遗传算法实现
% 初始化种群
population = zeros(populationSize, k*N*N+1);
for i = 1:populationSize
for j = 1:k*N*N
```

```
perm = rand_seeds(randperm(numel(rand_seeds)));
population(i, j) = perm(1);
end
population(i,k*N*N+1)=rand();
end

% 存储每次迭代的最优适应度值
bestFitnessHistory = zeros(max_iter, 1);

for iter = 1:max_iter
% 计算适应度
fitness = zeros(populationSize, 1);
for i = 1:populationSize
fitness(i) = norm_F(population(i, :), F_N, k, N);
end

% 记录当前迭代的最优适应度值以及最优解
[bestFitness,index] = min(fitness);
bestSolution = population(index,:);
bestFitnessHistory(iter) = bestFitness;
%fprintf('Iteration: %d, Best Fitness: %.4f\n', iter,
   bestFitness);

% 选择
% 锦标赛选择法
tournamentPopulation = zeros(populationSize, chromosomeLength);
for i = 1:populationSize
tournamentIndices = randperm(populationSize, tournamentSize);
tournament = population(tournamentIndices, :);
tournamentFitness(i) = min(fitness(tournamentIndices));
tournamentPopulation(i, :) =
   tournament(find(fitness(tournamentIndices) ==
   tournamentFitness(i), 1, 'first'), :);
end

% 交叉
for i = 1:2:populationSize-1
r = rand();
if r < crossoverRate
% 使用单点交叉法进行交叉
crossPoint = randi([1, chromosomeLength-1]);
tournamentPopulation([i, i+1], crossPoint:end-1) =
   tournamentPopulation([i+1, i], crossPoint:end-1);
end
end

% 变异
for i = 1:populationSize
for j = 1:chromosomeLength-1
r = rand();
if r < mutationRate
% 小概率进行变异
perm = rand_seeds(randperm(numel(rand_seeds)));
tournamentPopulation(i, j) = perm(1);
```

```
end
end
end

% 更新种群
population = tournamentPopulation;
end

%% 绘制适应度随迭代次数变化的图表
figure;
plot(1:max_iter, bestFitnessHistory, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Best Fitness');
title('Fitness Evolution');
grid on;

%% 解码染色体
A=cell(1,k+1);
for i=1:k
A{i}=reshape(bestSolution(N^2*(i-1)+1:N^2*i),N,N);
end
A{k+1}=bestSolution(end);

%% 计算硬件复杂度
Ak=A{1};
for i=2:k
l=computeComplexity(Ak,A{i});
Ak=Ak*A{i};
end
if k==1 C=0;
else
C= q*l;
end
disp(['硬件复杂度为: ' num2str(C)]);

%% 适应度函数和其他函数
%适应度-目标函数
function f = norm_F(x,F_N,k,N)
A=cell(1,k);
for i=1:k
A{i}=reshape(x(N^2*(i-1)+1:N^2*i),N,N);
end
Ak=eye(N);
for i=1:k
Ak=Ak*A{i};
end
f=1/N*norm(x(k*N*N+1)*F_N-Ak,'fro');
end
```

1.4 问题三

```
clear; clc;
%% 数据初始化
```

```
t = 5;
N = 2^t; % 矩阵阶数
F_N = dftmtx(N); % 生成DFT矩阵

k = 3; % 分解产生的矩阵数

q = 3; % 元素范围
% 随机种子序列
seeds = zeros(1, 2*q+1);
for i = 1:q
seeds(i) = 2^(i-1);
seeds(q+i) = -2^(i-1);
end

% 遗传算法参数设置
populationSize = 200;  % 种群数量
chromosomeLength = k*N*N+1; % 染色体长度
max_iter = 250;        % 最大迭代次数
mutationRate = 0.1;  % 变异率
crossoverRate = 0.85;  % 交叉率
tournamentSize =5;    % 锦标赛选择法中每轮锦标赛的参与个体数

%% 遗传算法实现
% 初始化种群
population = zeros(populationSize, chromosomeLength);
A=cell(1,k+1);
for i = 1:populationSize
for j=1:k
A{j}=Matrix_generate(N,seeds);
population(i,(j-1)*N*N+1:j*N*N)=reshape(A{j},1,N*N);
end
population(i,end)=rand();
end

% 存储每次迭代的最优适应度值
bestFitnessHistory = zeros(max_iter, 1);

for iter = 1:max_iter
% 计算适应度
fitness = zeros(populationSize, 1);
for i = 1:populationSize
fitness(i) = norm_F(population(i, :), F_N, k, N);
end

% 记录当前迭代的最优适应度值以及最优解
[bestFitness,index] = min(fitness);
bestSolution = population(index,:);
bestFitnessHistory(iter) = bestFitness;
%fprintf('Iteration: %d, Best Fitness: %.4f\n', iter,
   bestFitness);

% 选择
% 锦标赛选择法
tournamentPopulation = zeros(populationSize, chromosomeLength);
```

```
for i = 1:populationSize
tournamentIndices = randperm(populationSize,
   tournamentSize);%从所有个体中抽取出参与锦标赛的
tournament = population(tournamentIndices, :);%出列
tournamentFitness(i) =
   min(fitness(tournamentIndices));%让竞赛者竞争产生赢家
%保留这个赢家的数据
tournamentPopulation(i, :) =
   tournament(find(fitness(tournamentIndices) ==
   tournamentFitness(i), 1, 'first'), :);
end

% 交叉
for i = 1:2:populationSize-1
r = rand();
if r <= crossoverRate
% 使用单点交叉法进行交叉，同时不破坏约束一
if k<=2
crossPoint = randi([1,N-1])*N;
else
crossPoint =
   min(randi([2,k])*N*N+1,randi([2,k-1])*N*N+1+randi([1,N-1])*N);
end
tournamentPopulation([i, i+1], crossPoint:end) =
   tournamentPopulation([i+1, i], crossPoint:end);
end
end

% 变异
for i = 1:populationSize
for j = 1:chromosomeLength-1
r = rand();%散点
if r < mutationRate
r=rand();%散点
% 对染色体进行变异
if r <=1 && r>0.1 %策略一
% 生成一个随机值替换当前基因
if population(i,j)~=0
population(i,j)=seeds(randperm(size(seeds,2),1))+1i*seeds(randperm(size(seeds,
end
else            %策略二
% 敲除当前基因
population(i, j) = 0;
end
end
end
end

% 更新种群
population = tournamentPopulation;
end

%% 绘制适应度随迭代次数变化的图表
figure;
```

```
plot(1:max_iter, bestFitnessHistory, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Best Fitness');
title('Fitness Evolution');
grid on;

%% 解码染色体
for i=1:k
A{i}=reshape(bestSolution(N^2*(i-1)+1:N^2*i),N,N);
end
A{k+1}=bestSolution(end);

%% 计算硬件复杂度
Ak=A{1};
for i=2:k
l=computeComplexity(Ak,A{i});
Ak=Ak*A{i};
end
if k==1 C=0;
else
C= q*l;
end
disp(['硬件复杂度为: ' num2str(C)]);

%% 适应度函数和其他函数
%适应度-目标函数
function f = norm_F(x,F_N,k,N)
A=cell(1,k);
for i=1:k
A{i}=reshape(x(N^2*(i-1)+1:N^2*i),N,N);
end
Ak=eye(N);
for i=1:k
Ak=Ak*A{i};
end
f=1/N*norm(x(end)*F_N-Ak,'fro');
end

%生成随机矩阵
function M=Matrix_generate(N,x)
M=zeros(N);
for i=1:N
x1=zeros(1,N);
x1(1)=x(randperm(size(x,2),1))+1i*x(randperm(size(x,2),1));
x1(2)=x(randperm(size(x,2),1))+1i*x(randperm(size(x,2),1));
x1=x1(randperm(size(x1,2),size(x1,2)));
M(i,:)=x1;
end
end
```

1.5 问题四

```
clear;clc;
```

```
%% 数据初始化
N1=4;
N2=8;
F_N1=dftmtx(N1);
F_N2=dftmtx(N2);
F_N=kron(F_N1,F_N2);%Kronecker积

k=3;%分解产生的矩阵数
q=3;

% 随机种子序列
seeds = zeros(1, 2*q+1);
for i = 1:q
seeds(i) = 2^(i-1);
seeds(q+i) = -2^(i-1);
end

% 分别对Fn1 Fn2进行近似稀疏分解
%% 第一步：分解Fn1

% 遗传算法参数设置
populationSize = 200;  % 种群数量
chromosomeLength = k*N1*N1+1; % 染色体长度
max_iter = 1000;        % 最大迭代次数
mutationRate = 0.1;  % 变异率
crossoverRate = 0.85;  % 交叉率
tournamentSize =5;    % 锦标赛选择法中每轮锦标赛的参与个体数

% 存储每次迭代的最优适应度值
bestFitnessHistory1 = zeros(max_iter, 1);

% 初始化种群
population = zeros(populationSize, chromosomeLength);
A=cell(1,k+1);
for i = 1:populationSize
for j=1:k
A{j}=Matrix_generate(N1,seeds);
population(i,(j-1)*N1*N1+1:j*N1*N1)=reshape(A{j},1,N1*N1);
end
population(i,end)=rand();
end

for iter = 1:max_iter
% 计算适应度
fitness = zeros(populationSize, 1);
for i = 1:populationSize
fitness(i) = norm_F(population(i, :), F_N1, k, N1);
end

% 记录当前迭代的最优适应度值以及最优解
[bestFitness1,index] = min(fitness);
bestSolution1 = population(index,:);
bestFitnessHistory1(iter) = bestFitness1;
```

```
% 选择
% 锦标赛选择法
tournamentPopulation = zeros(populationSize, chromosomeLength);
for i = 1:populationSize
tournamentIndices = randperm(populationSize,
   tournamentSize);%从所有个体中抽取出参与锦标赛的
tournament = population(tournamentIndices, :);%出列
tournamentFitness(i) =
   min(fitness(tournamentIndices));%让竞赛者竞争产生赢家
%保留这个赢家的数据
tournamentPopulation(i, :) =
   tournament(find(fitness(tournamentIndices) ==
   tournamentFitness(i), 1, 'first'), :);
end

% 交叉
for i = 1:2:populationSize-1
r = rand();
if r <= crossoverRate
% 使用单点交叉法进行交叉，同时不破坏约束一
if k<=2
crossPoint = randi([1,N1-1])*N1;
else
crossPoint =
   min(randi([2,k])*N1*N1+1,randi([2,k-1])*N1*N1+1+randi([1,N1-1])*N1);
end
tournamentPopulation([i, i+1], crossPoint:end) =
   tournamentPopulation([i+1, i], crossPoint:end);
end
end

% 变异
for i = 1:populationSize
for j = 1:chromosomeLength-1
r = rand();%散点
if r < mutationRate
r=rand();%散点
% 对染色体进行变异
if r <=1 && r>0.1 %策略一
% 生成一个随机值替换当前基因
if population(i,j)~=0
population(i,j)=seeds(randperm(size(seeds,2),1))+1i*seeds(randperm(size(seeds,
end
else            %策略二
% 敲除当前基因
population(i, j) = 0;
end
end
end
end

% 更新种群
population = tournamentPopulation;
end
```

```
%解码染色体
for i=1:k
A{i}=reshape(bestSolution1(N1^2*(i-1)+1:N1^2*i),N1,N1);
end
A{k+1}=bestSolution1(end);
%% 第二步 对Fn2进行分解
% 遗传算法参数设置
populationSize = 200;  % 种群数量
chromosomeLength = k*N2*N2+1; % 染色体长度
max_iter = 250;        % 最大迭代次数
mutationRate = 0.1;  % 变异率
crossoverRate = 0.85;  % 交叉率
tournamentSize =5;    % 锦标赛选择法中每轮锦标赛的参与个体数

% 存储每次迭代的最优适应度值
bestFitnessHistory2 = zeros(max_iter, 1);

% 初始化种群
population = zeros(populationSize, chromosomeLength);
B=cell(1,k+1);
for i = 1:populationSize
for j=1:k
B{j}=Matrix_generate(N2,seeds);
population(i,(j-1)*N2*N2+1:j*N2*N2)=reshape(B{j},1,N2*N2);
end
population(i,end)=rand();
end

for iter = 1:max_iter
% 计算适应度
fitness = zeros(populationSize, 1);
for i = 1:populationSize
fitness(i) = norm_F(population(i, :), F_N2, k, N2);
end

% 记录当前迭代的最优适应度值以及最优解
[bestFitness2,index] = min(fitness);
bestSolution2 = population(index,:);
bestFitnessHistory2(iter) = bestFitness2;

% 选择
% 锦标赛选择法
tournamentPopulation = zeros(populationSize, chromosomeLength);
for i = 1:populationSize
tournamentIndices = randperm(populationSize,
    tournamentSize);%从所有个体中抽取出参与锦标赛的
tournament = population(tournamentIndices, :);%出列
tournamentFitness(i) =
    min(fitness(tournamentIndices));%让竞赛者竞争产生赢家
%保留这个赢家的数据
tournamentPopulation(i, :) =
    tournament(find(fitness(tournamentIndices) ==
    tournamentFitness(i), 1, 'first'), :);
```

```
end

% 交叉
for i = 1:2:populationSize-1
r = rand();
if r <= crossoverRate
% 使用单点交叉法进行交叉，同时不破坏约束一
if k<=2
crossPoint = randi([1,N2-1])*N2;
else
crossPoint =
   min(randi([2,k])*N2*N2+1,randi([2,k-1])*N2*N2+1+randi([1,N2-1])*N2);
end
tournamentPopulation([i, i+1], crossPoint:end) =
   tournamentPopulation([i+1, i], crossPoint:end);
end
end

% 变异
for i = 1:populationSize
for j = 1:chromosomeLength-1
r = rand();%散点
if r < mutationRate
r=rand();%散点
% 对染色体进行变异
if r <=1 && r>0.1 %策略一
% 生成一个随机值替换当前基因
if population(i,j)~=0
population(i,j)=seeds(randperm(size(seeds,2),1))+1i*seeds(randperm(size(seeds,
end
else            %策略二
% 敲除当前基因
population(i, j) = 0;
end
end
end
end

% 更新种群
population = tournamentPopulation;
end

%解码染色体
for i=1:k
B{i}=reshape(bestSolution2(N2^2*(i-1)+1:N2^2*i),N2,N2);
end
B{k+1}=bestSolution2(end);

%% 将分解后的矩阵按照Kronecker性质进行相乘，并计算硬件复杂度
l=0;FN_approx=eye(N1*N2);
for i=1:k
l=l+computeComplexity(FN_approx,kron(A{i},B{i}));
FN_approx=FN_approx*kron(A{i},B{i});
end
```

```
C=q*l;

disp(['C=' num2str(C)]);
disp(['bestFitness1: ' num2str(bestFitness1)]);
disp(['bestFitness2: ' num2str(bestFitness2)]);
disp(['RMSE: ' num2str(1/(N1*N2)*norm(F_N-FN_approx,'fro'))])

%% 函数定义
%生成随机矩阵
function M=Matrix_generate(N,x)
M=zeros(N);
for i=1:N
x1=zeros(1,N);
x1(1)=x(randperm(size(x,2),1))+1i*x(randperm(size(x,2),1));
x1(2)=x(randperm(size(x,2),1))+1i*x(randperm(size(x,2),1));
x1=x1(randperm(size(x1,2),size(x1,2)));
M(i,:)=x1;
end
end

%适应度-目标函数
function f = norm_F(x,F_N,k,N)
A=cell(1,k);
for i=1:k
A{i}=reshape(x(N^2*(i-1)+1:N^2*i),N,N);
end
Ak=eye(N);
for i=1:k
Ak=Ak*A{i};
end
f=1/N*norm(x(end)*F_N-Ak,'fro');
end
```

1.6 问题五

```
clear; clc;
%% 数据初始化
t = 1;
N = 2^t; % 矩阵阶数
F_N = dftmtx(N); % 生成DFT矩阵

k = 3; % 分解产生的矩阵数

q = 5; % 元素范围
% 随机种子序列
seeds = zeros(1, 2*q+1);
for i = 1:q
seeds(i) = 2^(i-1);
seeds(q+i) = -2^(i-1);
end

% 遗传算法参数设置
populationSize = 300;  % 种群数量
```

```
chromosomeLength = k*N*N+1; % 染色体长度
max_iter = 500;        % 最大迭代次数
mutationRate = 0.01;  % 变异率
crossoverRate = 0.70;  % 交叉率
tournamentSize =50;   % 锦标赛选择法中每轮锦标赛的参与个体数

% 遗传算法实现
bestFitness=10000;
while bestFitness>0.1
% 初始化种群
population = zeros(populationSize, chromosomeLength);
A=cell(1,k+1);
for i = 1:populationSize
for j=1:k
A{j}=Matrix_generate(N,seeds);
population(i,(j-1)*N*N+1:j*N*N)=reshape(A{j},1,N*N);
end
population(i,end)=rand();
end

for iter = 1:max_iter
% 计算适应度
fitness = zeros(populationSize, 1);
for i = 1:populationSize
fitness(i) = norm_F(population(i, :), F_N, k, N);
end

% 记录当前迭代的最优适应度值以及最优解
[bestFitness,index] = min(fitness);
bestSolution = population(index,:);
%fprintf('Iteration: %d, Best Fitness: %.4f\n', iter,
   bestFitness);

% 选择
% 锦标赛选择法
tournamentPopulation = zeros(populationSize, chromosomeLength);
for i = 1:populationSize
tournamentIndices = randperm(populationSize,
   tournamentSize);%从所有个体中抽取出参与锦标赛的
tournament = population(tournamentIndices, :);%出列
tournamentFitness(i) =
   min(fitness(tournamentIndices));%让竞赛者竞争产生赢家
%保留这个赢家的数据
tournamentPopulation(i, :) =
   tournament(find(fitness(tournamentIndices) ==
   tournamentFitness(i), 1, 'first'), :);
end

% 交叉
for i = 1:2:populationSize-1
r = rand();
if r <= crossoverRate
% 使用单点交叉法进行交叉，同时不破坏约束一
if k<=2
```

```
crossPoint = randi([1,N-1])*N;
else
crossPoint =
   min(randi([2,k])*N*N+1,randi([2,k-1])*N*N+1+randi([1,N-1])*N);
end
tournamentPopulation([i, i+1], crossPoint:end) =
   tournamentPopulation([i+1, i], crossPoint:end);
end
end

% 变异
for i = 1:populationSize
for j = 1:chromosomeLength-1
r = rand();%散点
if r < mutationRate
r=rand();%散点
% 对染色体进行变异
if r <=1 && r>0.1 %策略一
% 生成一个随机值替换当前基因
if population(i,j)~=0
population(i,j)=seeds(randperm(size(seeds,2),1))+1i*seeds(randperm(size(seeds,2)
end
else            %策略二
% 敲除当前基因
population(i, j) = 0;
end
end
end
end

% 更新种群
population = tournamentPopulation;
end
end

% 解码染色体
for i=1:k
A{i}=reshape(bestSolution(N^2*(i-1)+1:N^2*i),N,N);
end
A{k+1}=bestSolution(end);

% 计算硬件复杂度
Ak=A{1};
for i=2:k
l=computeComplexity(Ak,A{i});
Ak=Ak*A{i};
end
if k==1
C=0;
else
C= q*l;
end
disp(['硬件复杂度为: ' num2str(C)]);
disp(['RMSE: ' num2str(bestFitness) ])
```

```
%% 适应度函数和其他函数
%适应度-目标函数
function f = norm_F(x,F_N,k,N)
A=cell(1,k);
for i=1:k
A{i}=reshape(x(N^2*(i-1)+1:N^2*i),N,N);
end
Ak=eye(N);
for i=1:k
Ak=Ak*A{i};
end
f=1/N*norm(F_N-x(end)*Ak,'fro');
end

%生成随机矩阵
function M=Matrix_generate(N,x)
M=zeros(N);
for i=1:N
x1=zeros(1,N);
x1(1)=x(randperm(size(x,2),1))+1i*x(randperm(size(x,2),1));
x1(2)=x(randperm(size(x,2),1))+1i*x(randperm(size(x,2),1));
x1=x1(randperm(size(x1,2),size(x1,2)));
M(i,:)=x1;
end
end
```