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	电子科技大学
参赛队号	23106140164
队员姓名	1.李紫荆
	2.何诗宇
	3.龙子璇

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问题研究

摘　要

随着芯片的发展，DFT 计算的硬件复杂度与其算法复杂度和数据元素取值范围相关，如何降低硬件复杂度至关重要。本文针对 DFT 矩阵分解中的问题，基于**稀疏矩阵连乘拟合**的思想，以**最小化误差和硬件复杂度**为目标，建立了 DFT 类矩阵的整数分解逼近模型，并使用**蝶形分解算法、遗传算法、Feig-Winograd 算法和优化搜索算法**对模型进行求解，同时我们提出了**两种新的“混合积分解—降维寻优”算法**来拟合 Kronecker 积矩阵。

问题一的目标是限制乘法器的个数从而降低硬件复杂度。对此，本文采用 **Cooley-Tukey 算法**，根据**蝶形运算**的对称性，把 N 阶 DFT 矩阵分解为 $\log_2(N)+1$ 个稀疏子矩阵的乘积。该方法分解效率高，且能达到 10^{-17} **的高精度**，复杂度如 3.4 表 2 所示。

问题二的目标是限制分解矩阵中的元素实虚部的取值，对此采用了基于 **Feig-Winograd 矩阵映射的算法、遗传算法、序列二次规划算法**，评估了三种不同算法的误差和复杂度，三种算法的**复杂度都为 0**，遗传算法的精度最高，**RMSE 最小可达 0.15**。

问题三的目标是保证矩阵稀疏性的同时限制元素范围，除了采用遗传算法，还提出了一种**新的优化搜索算法**保证分解结果的精度和复杂度。两种算法的**硬件复杂度都为 0，**遗传算法分解的矩阵 **RMSE 可达 0.08，**优化搜索算法 **RMSE 最小可达 0.03**。因为优化搜索算法优秀的性能和较短的训练时间，问题五选择采用此方法进行求解。

问题四的目标是分解 DFT 矩阵的 Kronecker 积，并且要同时限制分解矩阵的稀疏性和元素范围。对此，结合 Cooley-Tukey 算法、Kronecker 积的运算性质、矩阵初等行变换、遗传算法这四种基本原理，本文提出**两种新的“混合积分解—降维寻优法”**。这两种方法都能达到 **RMSE 最小可达 0.05**。其中，方法 II 的**复杂度为 0,** 方法 I 的复杂度如 6.4 表 6 所示。

问题五的目标是在 $RMSE \le 0.1$ 的前提下同时满足问题三的约束条件，**改进的优化搜索算法**在把元素搜索范围提升到 $P=[0,\pm1,\pm2,\pm4,\pm8]$ 的情况下，**硬件复杂度都为 0**，8 阶 DFT 矩阵分解精度 **RMSE 可达 0.095**。

最后，我们通过衡量**最小误差和硬件复杂度，**对提出的模型进行全面的评价：本文的模型贴合实际，能合理解决提出的问题，计算得出的 RMSE 较小，能很好地拟合 DFT 矩阵，部分模型复杂度较低，该模型在大规模 DFT 类矩阵的整数分解方面也能使用。

关键词： **稀疏矩阵，蝶形算法，遗传算法，优化搜索，混合积分解—降维寻优**

目录

1 问题重述

1.1 问题背景

离散傅里叶变换（Discrete Fourier Transform，DFT）作为一种基本工具广泛应用于工程、科学以及数学领域。例如，通信信号处理中，常用 DFT 实现信号的正交频分复用（Orthogonal Frequency Division Multiplexing，OFDM）系统的时频域变换。另外，在信道估计中，也需要用到逆 DFT（IDFT）和 DFT 以便对信道估计结果进行时域降噪。

在芯片设计中，DFT 计算的硬件复杂度与其算法复杂度和数据元素取值范围相关。算法复杂度越高、数据取值范围越大，则硬件复杂度就越大。目前在实际产品中，一般采用快速傅里叶变换（Fast Fourier Transform，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。为进一步降低芯片资源开销，一种可行的思路是将 DFT 矩阵分解成整数矩阵连乘的形式。

给定N点的时域一维复数信号$x_0, x_1, \dots, x_{N-1}$，DFT 后得到的复数信号X_k（$k = 0,1,\dots,N-1$）由下式给出（其中j为虚数单位，下同）：

$$\mathrm{X}_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{j2\pi nk}{N}}, k = 0,1,2,\dots,N-1$$

写成矩阵形式为：

$$\mathbf{X} = \mathbf{F}_N \mathbf{x}$$

其中 $\mathbf{x}= [x_0 \ x_1 \ \cdots \ x_{N-1}]^{\mathrm{T}}$为时域信号向量，$\mathbf{X} = [\mathrm{X}_0 \ \mathrm{X}_1 \ \cdots \ \mathrm{X}_{N-1}]^{\mathrm{T}}$为变换后的频域信号向量，$\mathbf{F}_N$为 DFT 矩阵，形式如下：

$$\mathbf{F}_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^{N-1} \\ 1 & w^2 & w^4 & \cdots & w^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{N-1} & w^{2(N-1)} & \dots & w^{(N-1)(N-1)} \end{bmatrix}, w = \mathrm{e}^{-\frac{j2\pi}{N}}$$

由于 DFT 矩阵的特殊结构，存在很多方法加速傅里叶变换的计算，其中，分治的策略以及蝶形计算单元的优化是 DFT 性能的关键。下面分别给出用 FFT 和矩阵连乘拟合近似计算 DFT 的具体思路。

FFT 思路：FFT 采用蝶形运算的思想，以 radix-3 蝶形计算为例，其计算过程可以表示为：

$$\begin{bmatrix} \mathrm{X}_0 \\ \mathrm{X}_1 \\ \mathrm{X}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -3 & -1 \\ 1 & -3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & \sqrt{3}j/2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

可以看到蝶形的设计相对于直接 DFT 矩阵乘积的形式大幅降低了复数乘法运算的次数。

矩阵连乘拟合思路：DFT 可以用传统的蝶形计算方法精确实现，也可以用一种矩阵乘法拟合近似获得，其核心思想是将 DFT 矩阵近似表达为一连串稀疏的、元素取值有限的矩阵连乘形式。

可以看到在该方案中，分解后的矩阵元素均为整数，从而降低了每个乘法器的复杂度；另外$\mathbf{A}_1$~$\mathbf{A}_4$的稀疏特性可以减少乘法运算数量。可以看出，这其实是一种精度与硬件复杂度的折中方案，即损失了一定的计算精度，但是大幅降低了硬件复杂度。在对输出信噪比要求不高的情况下可以优先考虑此类方案。

1.2 问题提出

问题一：首先通过减少乘法器个数来降低硬件复杂度。由于仅在非零元素相乘时需要使用乘法器，若$\mathbf{A}_k$矩阵中大部分元素均为 0，则可减少乘法器的个数，因此希望$\mathbf{A}_k$为稀疏矩阵。对于$N=2^t, t=1,2,3,...$的 DFT 矩阵$\mathbf{F}_N$，请在满足**约束 1** 的条件下，对最优化问题(6)中的变量$\mathcal{A}$和β进行优化，并计算最小误差（ 即(6)的目标函数，下同）和方案的硬件复杂度C（由于本题中没有限定$\mathbf{A}_k$元素的取值范围，因此在计算硬件复杂度时可默认$q=16$）。

问题二：讨论通过限制$\mathbf{A}_k$中元素实部和虚部取值范围的方式来减少硬件复杂度的方案。对于$N=2^t, t=1,2,3,4,5$的 DFT 矩阵$\mathbf{F}_N$，请在满足**约束 2** 的条件下，对$\mathcal{A}$和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：同时限制$\mathbf{A}_k$的稀疏性和取值范围。对于$N=2^t, t=1,2,3,4,5$的 DFT 矩阵$\mathbf{F}_N$，请在**同时满足约束 1 和 2** 的条件下，对$\mathcal{A}$和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：进一步研究对其它矩阵的分解方案。考虑矩阵$\mathbf{F}_N=\mathbf{F}_{N1}\otimes\mathbf{F}_{N2}$，其中$\mathbf{F}_{N1}$和$\mathbf{F}_{N2}$分别是$N_1$和$N_2$维的 DFT 矩阵，$\otimes$表示 Kronecker 积（注意$\mathbf{F}_N$非 DFT 矩阵）。当$N_1=4, N_2=8$时，请在**同时满足约束 1 和 2** 的条件下，对$\mathcal{A}$和β进行优化，并计算最小误差和方案的硬件复杂

问题五：在问题 3 的基础上加上精度的限制来研究矩阵分解方案。要求将精度限制在 0.1 以内，即 RMSE≤0.1。对于$N=2^t, t=1,2,3,4,5$的 DFT 矩阵$\mathbf{F}_N$，请在**同时满足约束 1 和 2** 的条件下，对$\mathcal{A}$和β，$\mathcal{P}$进行优化，并计算方案的硬件复杂度C。

2 符号说明与整体思路

2.1 符号说明

符号说明如**表 1** 所示。

表1 符号说明

符号	含义
$\mathcal{A}$	$\{\mathbf{A}_1,\cdots,\mathbf{A}_K\}$为分解的系数矩阵
K	分解的矩阵个数
β	实值矩阵缩放因子
$\mathbf{F}_N$	N阶 DFT 矩阵
$\mathbf{I}_N$	N阶单位矩阵
$\mathbf{P}$	排列矩阵
$\mathbf{D}$	对角矩阵
C	乘法器的硬件复杂度
$\mathcal{P}$	$\mathbf{A}_k$中元素实部和虚部的取值范围
L	复数乘法的次数
B	β的量化复杂度
$\mathbf{D}_k$	标记$\mathbf{A}_k$中非简单元素位置的二进制矩阵
$\mathbf{P}_k$	标记$\mathbf{A}_k$中非零元素位置的二进制矩阵

2.2 整体思路

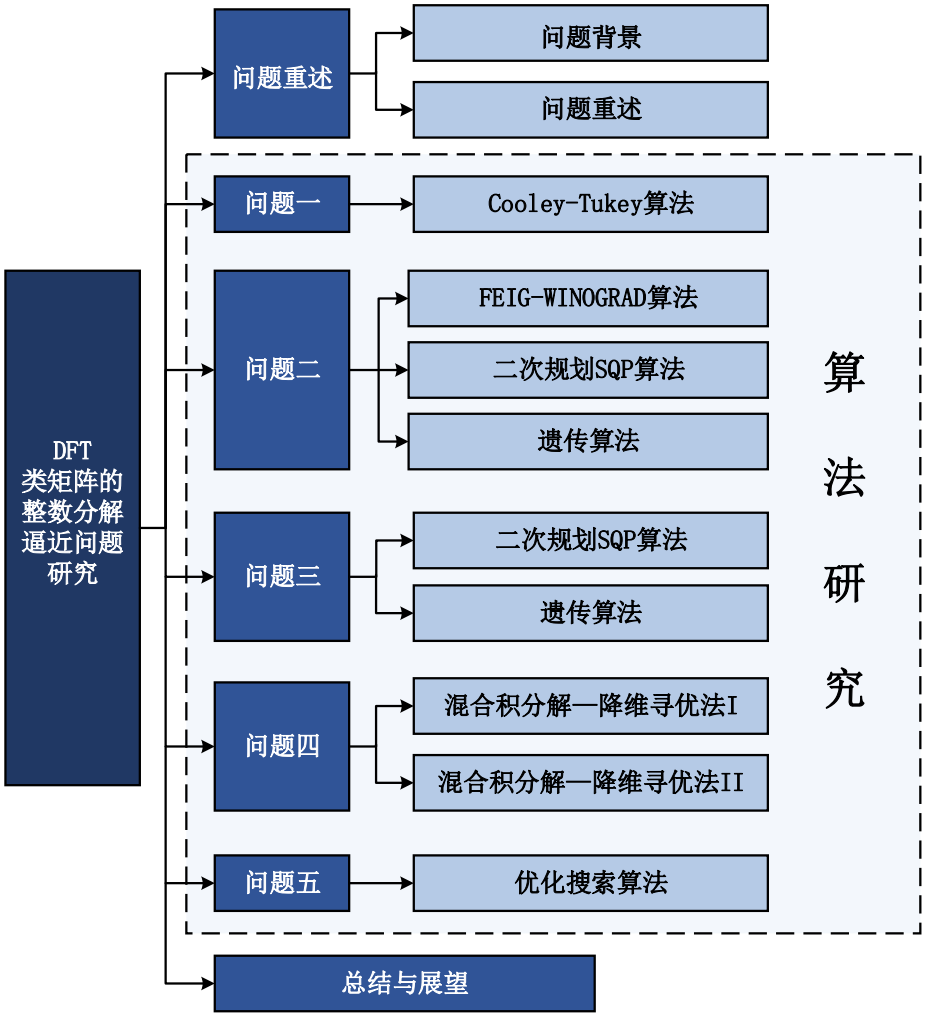

图1 本文技术路线

3 问题一的模型建立与求解

3.1 问题分析

问题一是通过减少乘法器的个数来降低硬件复杂度。也就是要求 $\mathbf{A}_k$ 为稀疏矩阵。由于问题一只在约束 1 下完成目标函数的最优求解，不限定 $\mathbf{A}_k$ 元素的取值范围。对此，本文采用 Cooley-Tukey 算法[1] [2] [3] 把 DFT 矩阵 $\mathbf{F}_N$ 分解为若干稀疏子矩阵的乘积，如式(1)所示

$$\mathbf{F}_N = \beta \mathbf{A}_K \cdots \mathbf{A}_2 \mathbf{A}_1 \mathbf{P} \tag{1}$$

其中 β 是实矩阵缩放因子，并且 $\mathbf{A}_1, \cdots, \mathbf{A}_K$ 和 $\mathbf{P}$ 都是稀疏矩阵。

需要注意的是，$\mathbf{P}$ 是一种排列矩阵，$\mathbf{P}$ 是若干个初等列变换矩阵的乘积。$\mathbf{A}_1 \cdots \mathbf{A}_K$ 右乘以 $\mathbf{P}$ 可以理解为矩阵 $\mathbf{P}$ 仅仅调换 $\mathbf{A}_1 \cdots \mathbf{A}_K$ 列的顺序，这种列顺序变换操作并不会增加乘法复杂度。附录所示的 MATLAB 程序也可以验证 $\mathbf{A}_K \cdots \mathbf{A}_2 \mathbf{A}_1 \mathbf{P}$ 与 $\mathbf{A}_K \cdots \mathbf{A}_2 \mathbf{A}_1$ 的乘法复杂度是相同的。

$\mathbf{A}_1, \cdots, \mathbf{A}_K$ 和 $\mathbf{P}$ 的分解过程具体在 3.3 小节中描述。

3.2 模型建立

分解之后的矩阵与误差及硬件复杂度之间的对应关系可以建立模型来表达，前文中已模型中相关符号进行定义，根据问题描述和表 1 内给出的参数变量符号，分解之后的矩阵与误差及硬件复杂度之间的关系可以采用以下数学模型来表示：

目标函数：

问题一的优化目标有三个，分别是变量$\mathcal{A}$，β 和乘法器个数 L，这是典型的多目标问题，我们对每个目标进行加权，使其变成单目标问题，具体如下：

1) 目标函数最小化

$$F=\min\left\{RMSE(\mathcal{A},\beta)q_1+Bq_2+Lq_3\right\} \tag{2}$$

其中：q 为权重，$q_1=0.6$，$q_2=0.2$，$q_3=0.2$。

2) 优化目标一

使计算误差最小，即：

$$RMSE(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\left\|\mathbf{F}_N-\beta\mathbf{A}_K\cdots\mathbf{A}_2\mathbf{A}_1\right\|_F^2} \tag{3}$$

3) 优化目标二

要优化 β，使 β 的复杂度也最小，定义 B 的定义与 P 相似，使 β 能用更小的位宽表示出来

$$B=\left|\log_2\beta\right| \tag{4}$$

4) 优化目标三

用 $\mathbf{D}_1,\mathbf{D}_2,\cdots,\mathbf{D}_K$ 矩阵分别表示对应 $\mathbf{A}_1,\mathbf{A}_2,\cdots,\mathbf{A}_K$ 矩阵非简单元素的位置（0、±1、$\pm j$ 不需要计算复杂度，所以把它们记为简单元素），即非简单元素标记为 1，简单元素标记为 0，得到 $\mathbf{D}_1,\mathbf{D}_2,\cdots,\mathbf{D}_K$ 矩阵为二进制矩阵。

把 $\mathbf{A}_1,\mathbf{A}_2,\cdots,\mathbf{A}_K$ 矩阵映射到 $\mathbf{D}_1,\mathbf{D}_2,\cdots,\mathbf{D}_K$ 矩阵的操作记为 dif，$\mathbf{D}_k=dif(\mathbf{A}_k)$，得到乘法器的个数 L 为

$$L=\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbf{D}_K dif(\mathbf{A}_{K-1}\cdots\mathbf{A}_2\mathbf{A}_1)+\cdots+\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbf{D}_3 dif(\mathbf{A}_2\mathbf{A}_1)+\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbf{D}_2\mathbf{D}_1 \tag{5}$$

约束条件：

用 $\mathbf{P}_1,\mathbf{P}_2,\cdots,\mathbf{P}_K$ 矩阵分别表示对应 $\mathbf{A}_1,\mathbf{A}_2,\cdots,\mathbf{A}_K$ 矩阵非零元素的位置，即非零元素标记为 1，零元素标记为 0，得到 $\mathbf{P}_1,\mathbf{P}_2,\cdots,\mathbf{P}_K$ 矩阵为二进制矩阵。

每个矩阵的每行至多只有 2 个非零元素

$$\sum_{j=1}^{N}\mathbf{P}_k(i,j)\le 2 \tag{6}$$

其中 $k=1,2,\ldots,K$，$i=1,2,\ldots,N$，$\mathbf{P}_k(i,j)$ 表示 $\mathbf{P}_k$ 矩阵第 i 行第 j 列的元素。

3.3 基于 Cooley-Tukey 算法的分解方法

Cooley-Tukey 的基本思想是把一个合数点数的 $N=N_1N_2$ 点 DFT 拆分成 N_1 个 N_2 点 DFT。下面 $\mathbf{F}_8$ 矩阵为例，给出 DFT 矩阵的 Cooley-Tukey 分解过程，即为蝶形运算，示意图如图 2 所示。

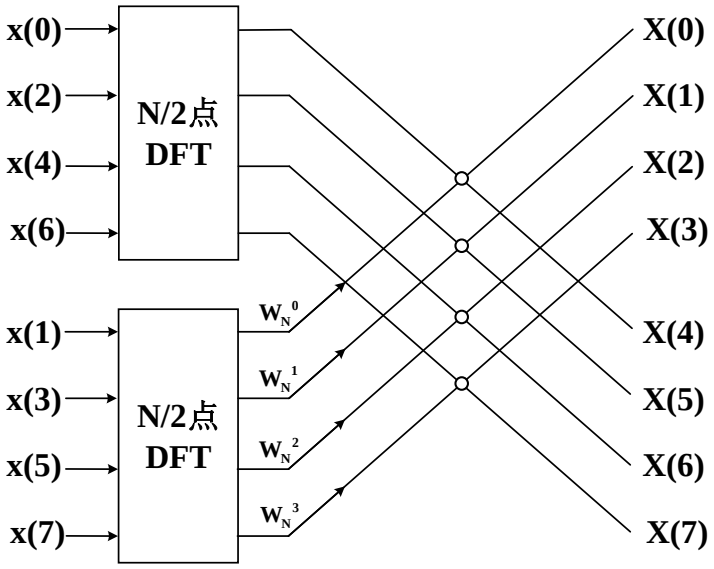

图2 蝶形算法示意图

已知 8 点的 DFT 可以写成下形式：

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & W_8 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ 1 & W_8^2 & W_8^4 & W_8^6 & W_8^8 & W_8^{10} & W_8^{12} & W_8^{14} \\ 1 & W_8^3 & W_8^6 & W_8^9 & W_8^{12} & W_8^{15} & W_8^{18} & W_8^{21} \\ 1 & W_8^4 & W_8^8 & W_8^{12} & W_8^{16} & W_8^{20} & W_8^{24} & W_8^{28} \\ 1 & W_8^5 & W_8^{10} & W_8^{15} & W_8^{20} & W_8^{25} & W_8^{30} & W_8^{35} \\ 1 & W_8^6 & W_8^{12} & W_8^{18} & W_8^{24} & W_8^{30} & W_8^{36} & W_8^{42} \\ 1 & W_8^7 & W_8^{14} & W_8^{21} & W_8^{28} & W_8^{35} & W_8^{42} & W_8^{49} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \tag{7}$$

其中$\begin{bmatrix} x(0) & \cdots & x(7) \end{bmatrix}^T$是时域信号，$\begin{bmatrix} X(0) & \cdots & X(7) \end{bmatrix}^T$是频域信号。

第一步，对$\begin{bmatrix} x(0) & \cdots & x(7) \end{bmatrix}^T$做时域抽取重新排列顺序可得

$$\begin{bmatrix} x(0) \\ x(4) \\ x(2) \\ x(6) \\ x(1) \\ x(5) \\ x(3) \\ x(7) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \tag{8}$$

第二步，把重排后的序列分成 4 组，每组做 2-DFT 变换，例如，对$\begin{bmatrix} x(0) & x(4) \end{bmatrix}^T$做 2-DFT 变换，可得

$$\begin{bmatrix} G_1(0) \\ G_1(1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(4) \end{bmatrix} \tag{9}$$

那么，做完 4 组 2-DFT 变换可得

$$\begin{bmatrix} G_1(0) \\ G_1(1) \\ G_2(0) \\ G_2(1) \\ G_3(0) \\ G_3(1) \\ G_4(0) \\ G_4(1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(4) \\ x(2) \\ x(6) \\ x(1) \\ x(5) \\ x(3) \\ x(7) \end{bmatrix} \tag{10}$$

第三步，由 2-DFT 样点合成 4-DFT 样点，例如，由(10)中的 4 个 2-DFT 样点 $G_1(0), G_1(1), G_2(0), G_2(1)$ 合成 4-DFT，如式(11)所示

$$\begin{bmatrix} F_1(0) \\ F_1(1) \\ F_2(2) \\ F_2(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & W_8^2 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -W_8^2 \end{bmatrix} \begin{bmatrix} G_1(0) \\ G_1(1) \\ G_2(0) \\ G_2(0) \end{bmatrix} \tag{11}$$

那么，合成后的 4-DFT 样点可以表示为 $\begin{bmatrix} F_1(0) & \cdots & F_2(1) \end{bmatrix}^T$，如式(12)所示

$$\begin{bmatrix} F_1(0) \\ F_1(1) \\ F_1(2) \\ F_1(3) \\ F_2(0) \\ F_2(1) \\ F_2(2) \\ F_2(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & W_8^2 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -W_8^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & W_8^2 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -W_8^2 \end{bmatrix} \begin{bmatrix} G_1(0) \\ G_1(1) \\ G_2(0) \\ G_2(1) \\ G_3(0) \\ G_3(1) \\ G_4(0) \\ G_4(1) \end{bmatrix} \tag{12}$$

第四步，由 4-DFT 样点 $\begin{bmatrix} F_1(0) & \cdots & F_2(1) \end{bmatrix}^T$ 即可合成所需的 8-DFT 频域信号 $\begin{bmatrix} X(0) & \cdots & X(7) \end{bmatrix}^T$，如式(13)所示

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & W_s & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & W_s^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & W_s^3 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -W_s & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -W_s^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -W_s^3 \end{bmatrix} \begin{bmatrix} F_1(0) \\ F_1(1) \\ F_1(2) \\ F_1(3) \\ F_2(0) \\ F_2(1) \\ F_2(2) \\ F_2(3) \end{bmatrix} \tag{13}$$

综上所述，$\mathbf{F}_8 = \mathbf{A}_3\mathbf{A}_2\mathbf{A}_1\mathbf{P}$，其中，$\mathbf{P}, \mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$ 分别如下所示，可见 $\mathbf{P}, \mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$ 均是满足约束 1 的稀疏矩阵。

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{15}$$

$$\mathbf{A}_2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & W_8^2 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -W_8^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & W_8^2 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -W_8^2 \end{bmatrix} \tag{16}$$

$$\mathbf{A}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & W_s & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & W_s^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & W_s^3 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -W_s & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -W_s^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -W_s^3 \end{bmatrix} \tag{17}$$

3.4 结果与分析

本文采用 MATLAB 实现上述模型，可以完成对 $N=2^t, t=1,2,3,...$ 阶 DFT 矩阵的分解，当 N=2,4,8,16,32 时，DFT 矩阵 $\mathbf{F}_N$ 分解后的计算复杂度和 RMSE 的值如 3.4 表 2 所示

表2 问题一的硬件复杂度

阶数 N	2	4	8	16	32
硬件复杂度 C	0	0	0	512	5120

绘制得到图像如下

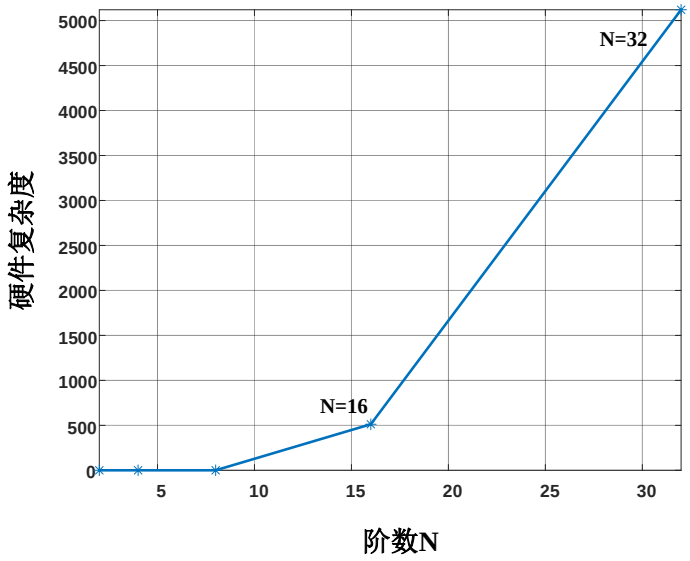

图3 问题一的硬件复杂度

如图 3 所示，阶数 N 为 2，4，8 时硬件复杂度为 0，N 为 16 时硬件复杂度为 512，N 为 32 时硬件复杂度为 5120，由图像可以看出复杂度随指数型增长，所以在后续问题中需要元素实部和虚部取值范围来减小复杂度。

受文档纸张大小限制，以下展示 $\mathbf{F}_2,\mathbf{F}_4,\mathbf{F}_8,\mathbf{F}_{16}$ 的分解结果：

（1）当 N=2 时，$\mathbf{F}_2=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$，$\mathbf{F}_2$ 的所有元素都是简单元素，满足约束 1；

（2）当 N=4 时，$\mathbf{F}_4=\beta\mathbf{A}_2\mathbf{A}_1\mathbf{P}$，取 $\beta=\frac{1}{2}$，分解后的稀疏矩阵为

$$\mathbf{P}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\quad \mathbf{A}_1=\begin{bmatrix}1 & 1 & 0 & 0\\ 1 & -1 & 0 & 0\\ 0 & 0 & 1 & 1\\ 0 & 0 & 1 & -1\end{bmatrix}\quad \mathbf{A}_2=\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & W_4^1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -W_4^1\end{bmatrix}$$

（3）当 N=8 时，$\mathbf{F}_8=\beta\mathbf{A}_3\mathbf{A}_2\mathbf{A}_1\mathbf{P}$，取 $\beta=\frac{1}{\sqrt{8}}$，分解后的稀疏矩阵为

$$\mathbf{P}=\begin{bmatrix}1&0&0&0&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&0&0&0&1&0\\0&1&0&0&0&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&0&0&0&1\end{bmatrix}\quad \mathbf{A}_1=\begin{bmatrix}1&1&0&0&0&0&0&0\\1&-1&0&0&0&0&0&0\\0&0&1&1&0&0&0&0\\0&0&1&-1&0&0&0&0\\0&0&0&0&1&1&0&0\\0&0&0&0&1&-1&0&0\\0&0&0&0&0&0&1&1\\0&0&0&0&0&0&1&-1\end{bmatrix}$$

$$\mathbf{A}_2=\begin{bmatrix}1&0&1&0&\cdots&&\mathbf{O}\\0&1&0&W_8^2&&&\vdots\\1&0&-1&0&\ddots&&\vdots\\0&1&0&-W_8^2&&&\\&&&&1&0&1&0\\\vdots&&\ddots&&0&1&0&W_8^2\\\vdots&&&&1&0&-1&0\\\mathbf{O}&&\cdots&&0&1&0&-W_8^2\end{bmatrix}\quad \mathbf{A}_3=\begin{bmatrix}1&0&0&0&1&0&0&0\\0&1&0&0&0&W_8&0&0\\0&0&1&0&0&0&W_8^2&0\\0&0&0&1&0&0&0&W_8^3\\1&0&0&0&-1&0&0&0\\0&1&0&0&0&-W_8&0&0\\0&0&1&0&0&0&-W_8^2&0\\0&0&0&1&0&0&0&-W_8^3\end{bmatrix}$$

（4）当 N=16 时，$\mathbf{F}_{16}=\mathbf{A}_4\mathbf{A}_3\mathbf{A}_2\mathbf{A}_1\mathbf{P}$，取 $\beta=\dfrac{1}{4}$，分解后的稀疏矩阵为

$$\mathbf{P}=\left[e_0e_8e_4e_{12}e_2e_{10}e_6e_{14}e_1e_9e_5e_{13}e_3e_{11}e_7e_{15}\right]$$

$$\mathbf{A}_1=\begin{bmatrix}1&1&&&&&&&\cdots&&\cdots&&&&&\mathbf{O}\\1&-1&&&&&&&&&&&&&&\\&&1&1&&&&&&&&&&&&\\&&1&-1&&&&&&&&&&&&\\&&&&1&1&&&&&&&&&&\\&&&&1&-1&&&&&&&&&&\\&&&&&&1&1&&&&&&&&\vdots\\&&&&&&1&-1&&&&&&&&\\&&&&&&&&1&1&&&&&&\vdots\\&&&&&&&&1&-1&&&&&&\\&&&&&&&&&&1&1&&&&&\\&&&&&&&&&&1&-1&&&&&\\&&&&&&&&&&&&1&1&&&\\&&&&&&&&&&&&1&-1&&&\\\mathbf{O}&&&&&&&&\cdots&&\cdots&&&&1&1\\&&&&&&&&&&&&&&1&-1\end{bmatrix}$$

$$\mathbf{A}_2 = \begin{bmatrix}
1 & 0 & 1 & 0 & & & & & & & & & & & & \\
0 & 1 & 0 & W_{16}^4 & & & & & & & & & & & & \\
1 & 0 & -1 & 0 & & & \cdots & & & & \cdots & & & & \mathbf{O} & \\
0 & 1 & 0 & -W_{16}^4 & & & & & & & & & & & & \\
 & & & & 1 & 0 & 1 & 0 & & & & & & & & \\
 & & & & 0 & 1 & 0 & W_{16}^4 & & & & & & & & \\
 & \vdots & & & 1 & 0 & -1 & 0 & & & \ddots & & & & \vdots & \\
 & & & & 0 & 1 & 0 & -W_{16}^4 & & & & & & & & \\
 & & & & & & & & 1 & 0 & 1 & 0 & & & & \\
 & & & & & & & & 0 & 1 & 0 & W_{16}^4 & & & & \\
 & \vdots & & & & & \ddots & & 1 & 0 & -1 & 0 & & & \vdots & \\
 & & & & & & & & 0 & 1 & 0 & -W_{16}^4 & & & & \\
 & & & & & & & & & & & & 1 & 0 & 1 & 0 \\
 & & & & & & & & & & & & 0 & 1 & 0 & W_{16}^4 \\
 & & \mathbf{O} & & & & \cdots & & & & \cdots & & 1 & 0 & -1 & 0 \\
 & & & & & & & & & & & & 0 & 1 & 0 & -W_{16}^4
\end{bmatrix}$$

$$\mathbf{A}_3 = \begin{bmatrix}
1 & & & & 1 & & & & & & & & & & & \\
 & 1 & & & & W_{16}^2 & & & & & & & & & & \\
 & & 1 & & & & W_{16}^4 & & & & & & & & & \\
 & & & 1 & & & & W_{16}^6 & & & & & & & & \\
1 & & & & -1 & & & & & & & & & & & \\
 & 1 & & & & -W_{16}^2 & & & & & & & & & & \\
 & & 1 & & & & -W_{16}^4 & & & & & & & & & \\
 & & & 1 & & & & -W_{16}^6 & & & & & & & & \\
 & & & & & & & & 1 & & & 1 & & & & \\
 & & & & & & & & & 1 & & & W_{16}^2 & & & \\
 & & & & & & & & & & 1 & & & W_{16}^4 & & \\
 & & & & & & & & & & & 1 & & & W_{16}^6 & \\
 & & & & \mathbf{O} & & & & 1 & & & & -1 & & & \\
 & & & & & & & & & 1 & & & & -W_{16}^2 & & \\
 & & & & & & & & & & 1 & & & & -W_{16}^4 & \\
 & & & & & & & & & & & 1 & & & & -W_{16}^6
\end{bmatrix}$$

$$\mathbf{A}_4=\begin{bmatrix}
1 & & & & 1 & & & & & & & & & & & \\
& 1 & & & & W_{16}^{1} & & & & & & & & & & \\
& & 1 & & & & W_{16}^{2} & & & & & & & & & \\
& & & 1 & & & & W_{16}^{3} & & & & & & & & \\
& & & & 1 & & & & W_{16}^{4} & & & & & & & \\
& & & & & 1 & & & & W_{16}^{5} & & & & & & \\
& & & & & & 1 & & & & W_{16}^{6} & & & & & \\
& & & & & & & 1 & & & & W_{16}^{7} & & & & \\
1 & & & & -1 & & & & & & & & & & & \\
& 1 & & & & -W_{16}^{1} & & & & & & & & & & \\
& & 1 & & & & -W_{16}^{2} & & & & & & & & & \\
& & & 1 & & & & -W_{16}^{3} & & & & & & & & \\
& & & & 1 & & & & -W_{16}^{4} & & & & & & & \\
& & & & & 1 & & & & -W_{16}^{5} & & & & & & \\
& & & & & & 1 & & & & -W_{16}^{6} & & & & & \\
& & & & & & & 1 & & & & -W_{16}^{7} & & & &
\end{bmatrix}$$

3.5 总结与评价

3.5.1 模型优点

（1）精确拟合。本文采用 Cooley-Tukey 算法把 DFT 矩阵$\mathbf{F}_N$分解为若干稀疏子矩阵的乘积，即$\mathbf{F}_N=\beta\mathbf{A}_K\cdots\mathbf{A}_2\mathbf{A}_1\mathbf{P}$，RMSE 值是$10^{-17}$量级。

（2）分解后的矩阵满足稀疏特性。由 4.4 小节的模型求解结果可见，分解后的每个子矩阵每行的非零元素不超过 2 个，满足约束 1

（3）分解效率高。已有文献只给出了 8 阶或 16 阶 DFT 矩阵的分解方法，由于阶数小，大多采用遍历轮询的方法求解出各个子矩阵。但随着阶数增大，遍历空间会呈指数级增长，大大增加了分解的难度。本文仅通过$\log_2(N)+1$次分解就能分别得到$\log_2(N)+1$个稀疏矩阵$\mathbf{P},\mathbf{A}_1,\cdots,\mathbf{A}_{\log_2(N)}$。只要 N 是 2 的整数次幂，采用本文的算法就能快速分解 DFT 矩阵。对应的实现代码详见附录。

3.5.2 模型缺点

尽管得到了稀疏矩阵$\mathbf{P},\mathbf{A}_1,\cdots,\mathbf{A}_{\log_2(N)}$，但由于该算法只满足约束 1，分解的矩阵中存在非简单元素，因此复杂度还需进一步降低，这将在问题三中进一步优化。

4 问题二的模型建立与求解

4.1 问题分析

问题二的主要难点在于需要限制稀疏矩阵元素的实虚部取值范围，如果采用传统的遍历搜索，需要考虑每一个元素可能的取值，这大大增加了分解难度。我们考虑先拟合出一个近似 DFT 的矩阵，然后再拆分成几个稀疏矩阵或对角阵相乘。基于这个想法，我

们简单推导了 Feig-Winograd 算法、遗传算法、序列二次规划 SQP（Sequential Quadratic Programming）等方法，进行精度、复杂度估计，最后展示了估计结果。

4.2 模型建立

问题二的数学模型与问题一类似，只是目标函数和约束条件都略作修改，具体如下：

目标函数：

问题二的优化目标有两个，分别是变量$\mathcal{A}$和β，这是典型的多目标问题，我们对每个目标进行加权，使其变成单目标问题，具体如下：

1) 目标函数最小化

$$F=\min\left\{RMSE(\mathcal{A},\beta)q_1+Bq_2\right\} \tag{18}$$

其中：q 为权重，$q_1=0.7$，$q_2=0.3$。

2) 优化目标一

使计算误差最小，即：

$$RMSE(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\left\|\mathbf{F}_N-\beta\mathbf{A}_K\cdots\mathbf{A}_2\mathbf{A}_1\right\|_F^2} \tag{19}$$

3) 优化目标二

要优化β，使β的复杂度也最小，即：

$$B=\left|\log_2\beta\right| \tag{20}$$

约束条件：

限定$\mathcal{A}$每个矩阵$\mathbf{A}_k$满足以下要求：

$$\begin{gathered}\mathbf{A}_k[l,m]\in\{x+jy\mid x,y\in\mathcal{P}\},\mathcal{P}=\{0,\pm1,\pm2,\ldots,+2^{q-1}\},\\ k=1,2,\ldots,K;l,m=1,2,\ldots,N\end{gathered} \tag{21}$$

其中，$\mathbf{A}_k[l,m]$表示矩阵$\mathbf{A}_k$第l行第m列的元素。

4.3 问题求解

在建立模型的基础上，本文给出了三种分解方法，将其应用到 DFT 类矩阵的整数分解方法中。问题一中，$N=2,4$的分解结果依然满足问题二的限制条件，所以我们只计算$N=8,16,32$的分解方法。

4.3.1 基于 FEIG-WINOGRAD 矩阵映射的分解方法

Feig-Winograd 算法是由一个指定元素的向量映射出一个逼近$\mathbf{F}_N$的矩阵，进而有利于分解成多个稀疏矩阵相乘[1]，具体流程如图 4 所示。

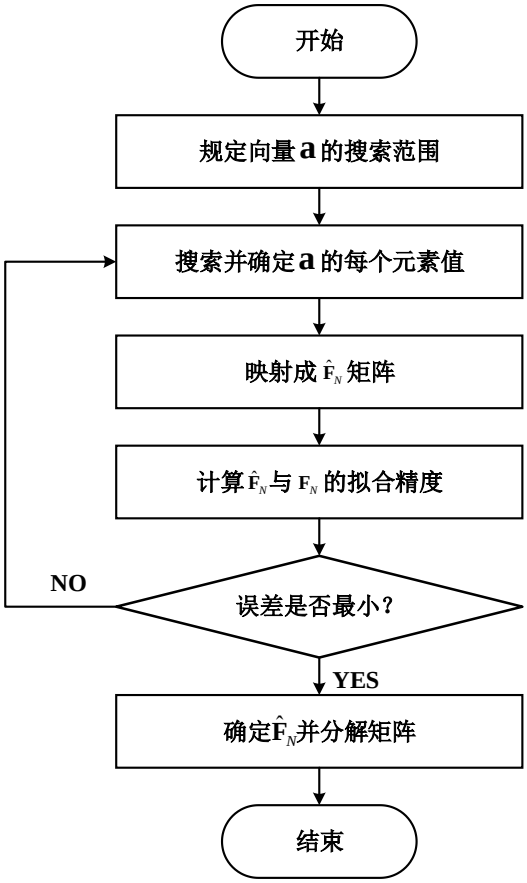

图4 FEIG-WINOGRAD 算法流程图

N 点 DFT 矩阵 $\mathbf{F}_N$ 中每一个元素可表示为

$$[\mathbf{F}_N]_{m,n} = \omega_N^{nk} \tag{22}$$

其中 $\omega_N = \exp(-2\pi j / N)$ 是旋转因子。如果 N 是偶数，则有如下关系

$$[\mathbf{F}_N]_{m+\frac{N}{2},n} = (-1)^n [\mathbf{F}_N]_{m,n} \tag{23}$$

其中 $m = 0,1,...,N/2-1$， $n = 0,1,...,N$，并且

$$[\mathbf{F}_N]_{m,n+\frac{N}{2}} = (-1)^m [\mathrm{F}_N]_{m,n} \tag{24}$$

其中 $m = 0,1,...,N$， $n = 0,1,...,N/2-1$。因此， $\mathbf{F}_N$ 可以分为四个子矩阵

$$\mathbf{F}_N = \begin{pmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} \end{pmatrix} \tag{25}$$

每个子矩阵都为 $N/2 \times N/2$ 阶，并且 $\mathbf{A}_{0,1}$， $\mathbf{A}_{1,0}$， $\mathbf{A}_{1,1}$ 中的元素和 $\mathbf{A}_{0,0}$ 是同样的，只是部分元素符号需要根据**错误!未找到引用源。**和**错误!未找到引用源。**取反[3] 。如果 N 是 4 的整数倍，则 $\mathbf{A}_{0,0}$ 内部还存在对称性，满足如下关系

$$[\mathbf{A}_{0,0}]_{m+\frac{N}{4},n} = (-j)^n [\mathbf{A}_{0,0}]_{m,n} \tag{26}$$

其中 $m = 0,1,...,N/4-1$， $n = 0,1,...,N$，并且

$$[\mathbf{A}_{0,0}]_{m,n+\frac{N}{4}} = (-j)^m [\mathbf{A}_{0,0}]_{m,n} \tag{27}$$

其中 $m = 0,1,...,N$， $n = 0,1,...,N/4-1$。因此，子矩阵 $\mathbf{A}_{0,0}$ 内部同样可以和**错误!未找到引用源。**一样再次分为 4 个子矩阵。

根据上面的分析，$\mathbf{F}_N$ 内部子矩阵的对称性意味着分解 $\mathbf{F}_N$ 仅需要计算 N/4 个不同的复数。根据 Feig-Winograd 因子分解，向量和矩阵映射关系可表示为：

$$\begin{aligned} f:\mathbb{C}^{N/4-1} &\to \mathbb{C}^N \times \mathbb{C}^N \\ \mathbf{a} &\mapsto \hat{\mathbf{F}}_N \end{aligned} \tag{28}$$

其中 $\mathbf{a}=[a_0,a_1,a_2,...,a_{N/4-1}]^T$ 是一个 $N/4\times 1$ 维的向量，$a_0=1$，因此 $\hat{\mathbf{F}}_N$ 的每个元素由 $\mathbf{a}$ 映射得出

$$[\hat{\mathbf{F}}_N]_{m,n}=(-1)^p(-j)^t a_{mn \bmod N/4} \tag{29}$$

其中 $p=m \bmod N/2+n \bmod N/2$，$t=m \bmod N/4+n \bmod N/4$。

可以看出估计矩阵 $\hat{\mathbf{F}}_N$ 是由 $\mathbf{F}_N$ 的对称性归纳得出的，向量 $\mathbf{a}$ 中每个元素的实虚部取值范围可以由约束 2 限制，即 $\boldsymbol{a}\in\{x+jy|x,y\in\mathcal{P}\}$，$\mathcal{P}=\{0,\pm1,\pm2,\dots,\pm4\}$。$\mathbf{a}$ 中每个元素的值决定了估计矩阵 $\hat{\mathbf{F}}_N$ 逼近 $\mathbf{F}_N$ 的程度，即决定了目标函数的精度。因此我们首先采用遍历搜索来确定 $\mathbf{a}$ 的每个元素取值，得出 $\hat{\mathbf{F}}_N$ 后，再进行快速分解。以 8 阶 DCT 矩阵为例，通过遍历得出的 $\mathbf{a}=[0,1-j]$, 其映射出的

$$\hat{\mathbf{F}}_8=\frac{1}{2}\begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 1-j & -2j & -1-j & -2 & -1+j & 2j & 1+j \\ 2 & -2j & -2 & 2j & 2 & -2j & -2 & 2j \\ 2 & -1-j & 2j & 1-j & -2 & 1+j & -2j & -1+j \\ 2 & -2 & 2 & -2 & 2 & -2 & 2 & -2 \\ 2 & -1+j & -2j & 1+j & -2 & 1-j & 2j & -1-j \\ 2 & 2j & -2 & -2j & 2 & 2j & -2 & -2j \\ 2 & 1+j & 2j & -1+j & -2 & -1-j & -2j & 1-j \end{pmatrix} \tag{30}$$

其分解因子由以下矩阵构成：

$$\hat{\mathbf{F}}_8=\mathbf{P}\cdot\mathbf{A}_4\cdot\mathbf{D}\cdot\mathbf{A}_3\cdot\mathbf{A}_2\cdot\mathbf{A}_1 \tag{31}$$

其中

$$\mathbf{A}_1=\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix},\mathbf{A}_2=\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix}$$

$$\mathbf{A}_3=\begin{pmatrix}1&1&0&0&0&0&0&0\\1&-1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&1\\0&0&0&0&0&1&1&0\\0&0&0&0&0&1&-1&0\\0&0&0&0&1&0&0&-1\end{pmatrix},\mathbf{A}_4=\begin{pmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&-1&0&0&0&0\\0&0&1&1&0&0&0&0\\0&0&0&0&1&-1&0&0\\0&0&0&0&0&0&-1&1\\0&0&0&0&1&1&0&0\\0&0&0&0&0&0&1&1\end{pmatrix}$$

$\mathbf{D}=diag([1,1,1,j,1,j,j,1])$ ， $\mathbf{P}$ 为初等变换矩阵 $\mathbf{P}=[\mathbf{e}_0\,|\,\mathbf{e}_4\,|\,\mathbf{e}_2\,|\,\mathbf{e}_5\,|\,\mathbf{e}_1\,|\,\mathbf{e}_7\,|\,\mathbf{e}_3\,|\,\mathbf{e}_6\,|]^T$ ，$\mathbf{e}_i(i=0,1,...,7)$ 为单位向量[7] 。同样地，16 阶 DCT 矩阵映射后得到

$$\hat{\mathbf{F}}_{16}=\frac{1}{2}\begin{pmatrix}\mathbf{A}_{0,0}&\mathbf{A}_{0,1}\\\mathbf{A}_{1,0}&\mathbf{A}_{1,1}\end{pmatrix}\tag{32}$$

其中

$$\mathbf{A}_{0,0}=\begin{pmatrix}2&2&2&2&2&2&2&2\\2&2-j&1-j&1-2j&-2j&-1-2j&-1-j&-2-j\\2&1-j&-2j&-1-j&-2&-1+j&2j&1+j\\2&1-2j&1-j&-2+j&2j&2+j&1-j&-1-2j\\2&-2j&-2&2j&2&-2j&-2&2j\\2&-1-2j&-1+j&2+j&-2j&-2+j&1+j&1-2j\\2&-1-j&2j&1-j&-2&1+j&-2j&-1+j\\2&-2-j&1+j&-1-2j&2j&1-2j&-1+j&2-j\end{pmatrix},$$

$$\mathbf{A}_{0,1}=\begin{pmatrix}2&2&2&2&2&2&2&2\\-2&-2+j&-1+j&-1+2j&2j&1+2j&1+j&2+j\\2&1-j&-2j&-1-j&-2&-1+j&2j&1+j\\-2&-1+2j&1+j&2-j&-2j&-2-j&-1+j&1+2j\\2&-2j&-2&2j&2&-2j&-2&2j\\-2&1+2j&1-j&-2-j&2j&2-j&-1-j&-1+2j\\2&-1-j&2j&1-j&-2&1+j&-2j&-1+j\\-2&2+j&-1-j&1+2j&-2j&-1+2j&1-j&-2+j\end{pmatrix},$$

$$\mathbf{A}_{1,0}=\begin{pmatrix}2&-2&2&-2&2&-2&2&-2\\2&-2+j&1-j&-1+2j&-2j&1+2j&-1-j&2+j\\2&-1+j&-2j&1+j&-2&1-j&2j&-1-j\\2&-1+2j&1-j&2-j&2j&-2-j&1-j&1+2j\\2&2j&-2&-2j&2&2j&-2&-2j\\2&1+2j&-1+j&-2-j&-2j&2-j&1+j&-1+2j\\2&1+j&2j&-1+j&-2&-1-j&-2j&1-j\\2&2+j&1+j&1+2j&2j&-1+2j&-1+j&-2+j\end{pmatrix},$$

$$\mathbf{A}_{1,1}=\begin{pmatrix} 2 & -2 & 2 & -2 & 2 & -2 & 2 & -2 \\ -2 & -2+j & -1+j & -1+2j & 2j & 1+2j & 1+j & 2+j \\ 2 & -1+j & -2j & 1+j & -2 & 1-j & 2j & -1-j \\ -2 & 1-2j & 1+j & -2+j & -2j & 2+j & -1+j & -1-2j \\ 2 & 2j & -2 & -2j & 2 & 2j & -2 & -2j \\ -2 & -1-2j & 1-j & 2+j & 2j & -2+j & -1-j & 1-2j \\ 2 & 1+j & 2j & -1+j & -2 & -1-j & -2j & 1-j \\ -2 & -2-j & -1-j & -1-2j & 2j & 1-2j & 1-j & 2-j \end{pmatrix},$$

此时 $\hat{\mathbf{F}}_{16}$ 可以被分解为

$$\hat{\mathbf{F}}_{16}=\mathbf{B}_1\cdot\mathbf{D}\cdot\mathbf{B}_2\cdot\mathbf{B}_3\cdot\mathbf{B}_4\cdot\mathbf{B}_5 \tag{33}$$

其中

$$\mathbf{B}_5=\begin{pmatrix} 1 & & & & & & & & & & & & & & & 1 \\ & 1 & & & & & & & & & & & & & 1 & \\ & & 1 & & & & & & & & & & & 1 & & \\ & & & 1 & & & & & & & & & 1 & & & \\ & & & & 1 & & & & & & & 1 & & & & \\ & & & & & 1 & & & & & 1 & & & & & \\ & & & & & & 1 & & & 1 & & & & & & \\ & & & & & & & 1 & 1 & & & & & & & \\ -1 & & & & & & & & 1 & & & & & & & \\ & & & & & & & -1 & 1 & & & & & & & \\ & & & & & & -1 & & & 1 & & & & & & \\ & & & & & -1 & & & & & 1 & & & & & \\ & & & & -1 & & & & & & & 1 & & & & \\ & & & -1 & & & & & & & & & 1 & & & \\ & & -1 & & & & & & & & & & & 1 & & \\ & -1 & & & & & & & & & & & & & 1 & \end{pmatrix},$$

$$
\mathbf{B}_3 = \begin{pmatrix}
1 & & 1 & & & & & & & & & & & & & \\
& 1 & & 1 & & & & & & & & & & & & \\
-1 & & 1 & & & & & & & & & & & & & \\
& -1 & & 1 & & & & & & & & & & & & \\
& & & & 1 & & & & & & & & & & & \\
& & & & & 1 & & 2 & & & & & & & & \\
& & & & & & 1 & & 2 & & & & & & & \\
& & & & & 2 & & -1 & & & & & & & & \\
& & & & & & 1 & & -2 & & & & & & & \\
& & & & & & & & & 1 & & 2 & & & & \\
& & & & & & & & & & 1 & & 2 & & & \\
& & & & & & & & & 2 & & -1 & & & & \\
& & & & & & & & & & 1 & & -2 & & & \\
& & & & & & & & & & & & & 1 & & 1 \\
& & & & & & & & & & & & & & 1 & \\
& & & & & & & & & & & & & 1 & & -1
\end{pmatrix},
$$

$$
\mathbf{B}_2 = \begin{pmatrix}
1 & 1 & & & & & & & & & & & & & & \\
1 & -1 & & & & & & & & & & & & & & \\
& & 1 & & & & & & & & & & & & & \\
& & & 1 & 2 & & & & & & & & & & & \\
& & & 1 & -2 & & & & & & & & & & & \\
& & & & & 1 & 1 & & & & & & & & & \\
& & & & & 1 & -1 & & & & & & & & & \\
& & & & & & & 1 & 1 & & & & & & & \\
& & & & & & & 1 & -1 & & & & & & & \\
& & & & & & & & & 1 & 1 & & & & & \\
& & & & & & & & & 1 & -1 & & & & & \\
& & & & & & & & & & & 1 & 1 & & & \\
& & & & & & & & & & & 1 & -1 & & & \\
& & & & & & & & & & & & & 1 & 2 & \\
& & & & & & & & & & & & & 1 & -2 & \\
& & & & & & & & & & & & & & & 1
\end{pmatrix},
$$

$$\mathbf{B}_1=\begin{pmatrix}
2 & & & & & & & & & & & & & & & \\
 & & & -1 & & & & 1 & & & & & & & & \\
 & & -1 & & & & & & & & & 1 & & & & \\
 & & & & & 1 & & & & 1 & & & & & & \\
 & -2 & & & & & & & & & & & & & -2 & \\
 & & & & & & -1 & & & & 1 & & & & & \\
 & & & 1 & & & & & & & & & 1 & & & \\
 & & & & & 1 & & & & 1 & & & & & & \\
2 & & & & & & & & & & & & & & & \\
 & & & & & 1 & & & & -1 & & & & & & \\
 & & & 1 & & & & & & & & & -1 & & & \\
 & & & & & & -1 & & & & -1 & & & & & \\
 & -2 & & & & & & & & & & & & & 2 & \\
 & & & & & 1 & & & & -1 & & & & & & \\
 & & -1 & & & & & & & & & -1 & & & & \\
 & & & & -1 & & & & -1 & & & & & & & \\
 & & & & & & & & & & & & & & & 2
\end{pmatrix},$$

$\mathbf{D}=1/2diag(1,1,1,1,1,1,1,1,j,j,j,j,j,j,j,j)$。

由于 Feig-Winograd 算法映射得到的矩阵$\mathcal{A}$元素**全部为实数或纯虚数，硬件复杂度较为 0**，而代价是最小误差相对较大，如**表 3** 列出了 $N=2^t,t=1,2,3,4,5$时计算 $\mathbf{F}_N$ 的最小误差和硬件复杂度 C，具体性能如 4.3.1 表 3 所示。

表3 Feig-Winograd 算法性能表

阶数 N	2	4	8	16	32
RMSE	0	0	0.07	0.25	0.21
硬件复杂度 C	0	0	0	0	0

4.3.2 基于序列二次规划 SQP 的分解方法

二次规划(QP)是一种特殊的非线性规划，即优化最小化或最大化多个变量的二次函数，并服从于这些变量的线性约束。序列二次规划 SQP 算法是将复杂的非线性优化问题转换为简单的二次规划问题来求解的算法，在每一步迭代中，先使用牛顿法计算一个二次近似函数，然后将其转化为一个二次规划问题。问题二中的约束二是一种非线性约束，故可以采用 SQP 迭代得出目标矩阵。给定一个非线性约束的最优问题：

$$\begin{aligned}&\min f(x)\\ &s.t.g_u(x)\le 0(u=1,2,,...,p)\\ &h_v(x)\le 0(v=1,2,...,m)\end{aligned}\tag{34}$$

利用泰勒展开把上式子的非线性约束问题的目标函数在迭代点 x^k 简化成二次函数，把非线性约束函数简化成线性函数后得到如下二次规划问题：

$$\begin{aligned}\min f(\mathbf{X}) &= \frac{1}{2}[\mathbf{X}-\mathbf{X}^k]^T\nabla^2 f(\mathbf{X}^k)[\mathbf{X}-\mathbf{X}^k]+\nabla f(\mathbf{X}^k)^T[\mathbf{X}-\mathbf{X}^k]\\ s.t.&\nabla g_u(\mathbf{X}^k)^T[\mathbf{X}-\mathbf{X}^k]+g_u(\mathbf{X}^k)\le 0(u=1,2,,...,p)\\ &\nabla h_v(\mathbf{X}^k)^T[\mathbf{X}-\mathbf{X}^k]+h_v(\mathbf{X}^k)=0(v=1,2,...,m)\end{aligned} \tag{35}$$

此问题为原来约束最优问题的近似问题，令：

$$\mathbf{S}=\mathbf{X}-\mathbf{X}^k \tag{36}$$

将上述二次规划问题变成关于变量$\mathbf{S}$的问题，即：

$$\begin{aligned}\min f(\mathbf{X}) &= \frac{1}{2}\mathbf{S}^T\nabla^2 f(\mathbf{X}^k)\mathbf{S}+\nabla f(\mathbf{X}^k)^T\mathbf{S}\\ s.t.&\nabla g_u(\mathbf{X}^k)^T\mathbf{S}+g_u(\mathbf{X}^k)\le 0(u=1,2,,...,p)\\ &\nabla h_v(\mathbf{X}^k)^T\mathbf{S}+h_v(\mathbf{X}^k)=0(v=1,2,...,m)\end{aligned} \tag{37}$$

令：

$$\begin{aligned}\mathbf{H}&=\nabla^2 f(\mathbf{X}^k)\\ \mathbf{C}&=\nabla f(\mathbf{X}^k)\\ \mathbf{A}_{eq}&=[\nabla h_1(\mathbf{X}^k),\nabla h_2(\mathbf{X}^k),...,\nabla h_m(\mathbf{X}^k)]^T\\ \mathbf{A}&=[\nabla g_1(\mathbf{X}^k),\nabla g_2(\mathbf{X}^k),...,\nabla g_p(\mathbf{X}^k)]^T\\ \mathbf{B}_{eq}&=[h_1(\mathbf{X}^k),h_2(\mathbf{X}^k),...,h_m(\mathbf{X}^k)]^T\\ \mathbf{B}&=[g_1(\mathbf{X}^k),g_2(\mathbf{X}^k),...,g_p(\mathbf{X}^k)]^T\end{aligned} \tag{38}$$

写成一般形式为：

$$\begin{aligned}&\min\frac{1}{2}\mathbf{S}^T\mathbf{H}\mathbf{S}+\mathbf{C}^T\mathbf{S}\\ &s.t.\mathbf{AS}=-\mathbf{B}\\ &\mathbf{A}_{\text{eq}}\mathbf{S}=-\mathbf{B}_{eq}\end{aligned} \tag{39}$$

求解此二次规划问题，将其最优解$\mathbf{S}^*$作为原问题的下一个搜索方向$\mathbf{S}^k$，并在该方向上进行原约束问题目标函数的约束一维搜索，这样就可以得到原约束问题的一个近似解$\mathbf{X}^{k+1}$。反复这一过程，就可以得到原问题的最优解。采用 SQP 方法，取$\beta=\sqrt{N}$优化 RMSE 目标函数,例如以 4 阶 DFT 矩阵为参考，迭代出的矩阵$\hat{\mathbf{F}}_4$最小误差为 0.3953，以 8 阶 DFT 矩阵为参考，迭代出的$\hat{\mathbf{F}}_8$最小误差为 0.4336，以下展示了用 SQP 估计的$\hat{\mathbf{F}}_4$和$\hat{\mathbf{F}}_8$。

$$\hat{\mathbf{F}}_4=\begin{pmatrix}1&0&1&1\\1&1&0&0\\1&0&1&0\\1&0&0&0\end{pmatrix},$$

$$\hat{\mathbf{F}}_8=\begin{pmatrix}0&0&0&0&1&1&1&1\\1&0&0&1&1&1&1&1\\0&1&1&1&0&0&0&0\\0&1&1&0&1&1&0&1\\0&0&1&0&0&1&1&1\\0&0&1&0&1&0&0&0\\1&1&1&1&1&0&0&0\\1&1&0&1&0&1&1&0\end{pmatrix}$$

SQP 能找出的结果与其他算法相比误差最大，优点是硬件复杂度 C 基本为 0。下表列出了 $N=2^t, t=1,2,3,4,5$ 时计算 $\mathbf{F}_N$ 的最小误差和硬件复杂度 C，具体性能如 4.3.2 表 4 表 3 所示。

表4 二次规划 SQP 算法性能表

N	2	4	8	16	32
RMSE	-	0.39	0.43	0.30	0.20
硬件复杂度 C	-	0	0	0	0

使用 SQP 算法存在的缺点很明显，对初始点的依赖性强，并且求解时需要较长的计算时间。如果初始点不同，计算的最小误差也会有区别，可能是因为陷入了局部最优解，这就是为什么目标函数的最小误差相比其他算法最大，不过 SQP 算法能够处理非线性约束和非光滑函数，在求解器的选择上，具有灵活性，因为可以根据具体问题的特点选择适合的求解器。

4.3.3 基于遗传算法的分解方法

遗传算法(GA)是数学中最常用来解决最优化问题的算法，Grenfenstette 提出的遗传算 法从代表问题可能潜在解集的一个种群开始，对种群反复进行复制、交叉已经变异操作，直接对结构对象进行操作，估计各个体的适应值，采用概率化的方式进行全局最优解的搜寻，根据“适者生存、优胜劣汰”的进化规则，使得群体越来越向最优解的方向进化[8] [9] [10] 。

算法的基本运算流程包括染色体编码、初始化种群设置、建立适应度函数对个体进行 评价、执行遗传操作、算法的终止规则、染色体解码，具体的算法流程图如图 5 所示。得到的分解矩阵 $\mathcal{A}=\{\mathbf{A}_1,\mathbf{A}_2,\ldots,\mathbf{A}_k\}$ 可以被简便地编码为染色体，只要给出适应度函数的定义，遗传算法就能直接对不同的分解序列进行筛选，每轮迭代都在原本的种群上更新得到表现更好的种群，基本遗传算法的时间复杂度为 $O(n^2)$，能够在较短的运算时间内给出较为优质的可行解。

在运行步骤当中，合理的种群规模大小很关键，会对遗传算法的性能产生影响[11] ，较大数量的种群可以遍历所有可行解，较小数量的种群计算量小，算法可以以很快的速度收敛，但可能结果是局部最优解，所以一个合适的种群规模变得尤为重要，最优的种群规模应当是决策变量数量的 4 倍到 6 倍，一般在 20~100 之间[12] ；其次是交叉概率，来衡量种群个体是否进行交叉操作的可能性大小，从种群中随机选出部分个体做交叉操作产生子代新个体，交叉概率一般的取值范围为 0.4~0.99；变异概率是个体是否进行变异操作的评价标准，变异概率太大可能会产生不可行解，太小则不易跳出局部最优而重复迭代，变异概率的取值一般为 0.0001~0.5；最大迭代次数是遗传算法预先设定的

一种运行终止规则[13] ，若当前已达到最大迭代次数，则退出循环终止操作，得到最优解。

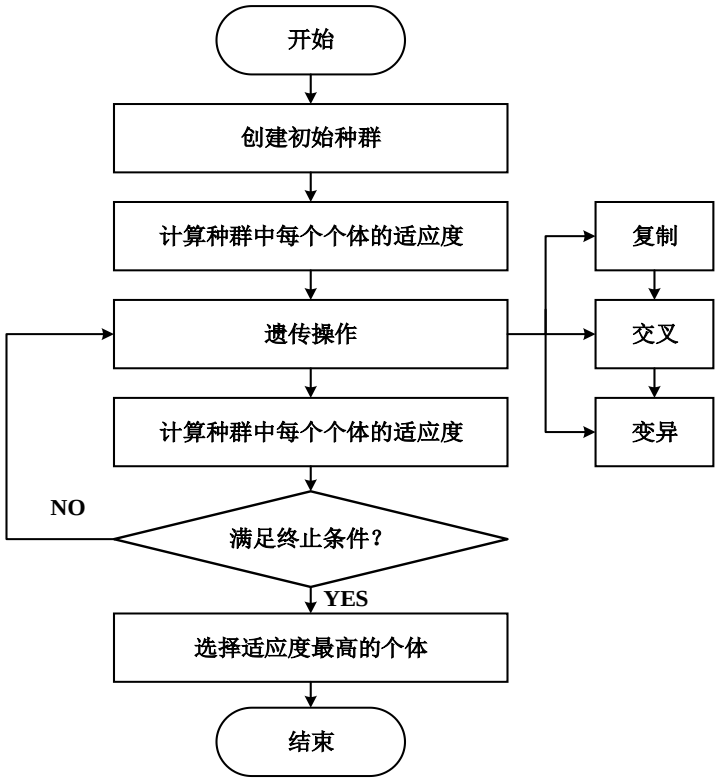

图5 遗传算法流程图

Step1：最优分解方法的获取

通过遗传算法解决 DFT 类矩阵的整数分解的过程如下：

1) 令迭代次数 i=0，随机生成初始种群，内含有 N 条染色体，每条染色体代表问题的一个解。
2) 将随机产生的符合约束条件的矩阵作为初始染色体，产生初始种群，种群中的染色体是由初始染色体中的基因点随机交换生成，计算种群内每条染色体的适应度值 Fit，判断算法是否满足设定的终止条件，如果不满足则更新迭代次数 i，令 i=i+1，生成复制种群 Selext(i+1)。
3) 进行复制操作，首先将当前代中适应度最高的两个个体保留到下一代种群中，再按照轮盘赌选择方式生成新一代种群。
4) 进行交叉操作，采用双点交叉，随机选取两个染色体，随机选定交叉点，两层基因链同时进行交叉操作，生成交叉种群 Pc(i+1)。
5) 进行变异操作，在一条染色体上随机选择两个基因点，交换两个基因点的值，两层基因链同时进行变异，生成变异种群 Pm(i+1)。
6) 在新一代种群的基础上，返回步骤 3。
7) 输出适应度最高的染色体，解码得到最优出车序列，算法结束。

Step2：β 调整方案及得分

在获取最优分解方法的同时，不断调制 β 的取值，最终找到使之最小的误差值 $RMSE(\mathcal{A},\beta)$ 。

采用 Feig-Winograd 算法得到的 8 阶 DFT 分解矩阵的误差较小，我们接着来采用**遗传算法**对 β 的取值进行优化，继续降低 16 阶和 32 阶 DFT 矩阵的误差。通过使用 MATLAB 编程求解，求得结果如下

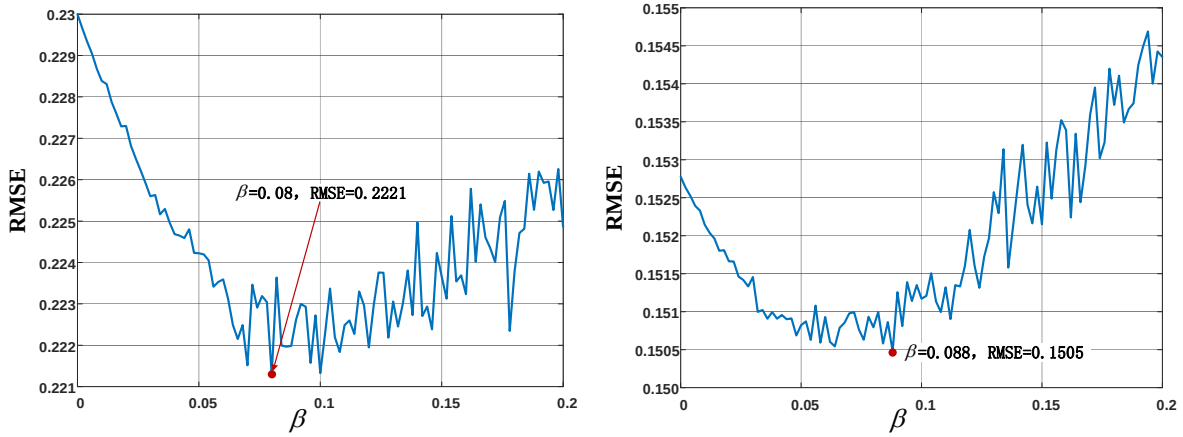

图6 16 阶和 32 阶 DFT 分解矩阵的误差

如图 6 所示，分别是 16 阶和 32 阶 DFT 分解矩阵的误差，

1） N=16 时，在 β=0.08 达到最小误差 0.2221；

2） N=32 时，在 β=0.088 达到最小误差 0.1505；

性能是要好于 Feig-Winograd 算法的。在基于遗传算法时，本文中我们并未对矩阵进行分解，而是直接进行拟合，所以**复杂度都为 0**，得到拟合的部分 16 阶 DFT 矩阵（拟合的完整 16 阶 DFT 矩阵和 32 阶 DFT 矩阵见附录）为：

$$\hat{\mathbf{F}}_{16}=\begin{bmatrix} -0.5+0.5j & -0.5-j & -1+4j & \cdots & 0.5-j & -0.25-0.25j & -1-4j \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -0.5+2j & -2-j & 1-0.5j & \cdots & 0.5+j & 0.25+j & 1+j \end{bmatrix}$$

然后我们将找到的最优的 β 取值带入遗传算法，进行 200 次迭代，得到适应度进化曲线。

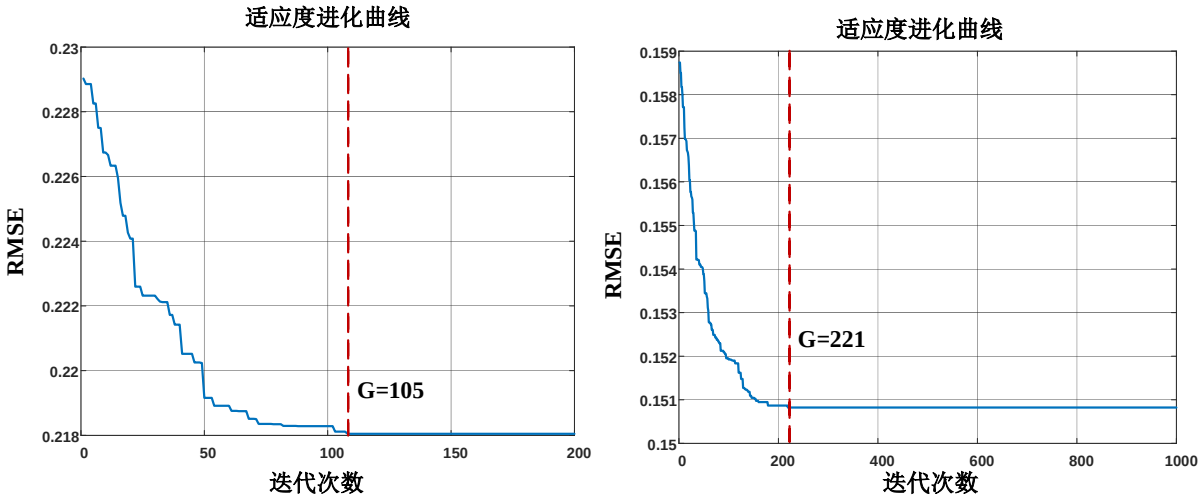

图7 16 阶和 32 阶拟合矩阵适应度进化曲线

如图 7 所示，分别是 16 阶和 32 阶拟合矩阵适应度进化曲线，分别第 105 代和第 221 代完成了收敛，且性能由于需要分解的 Feig-Winograd 算法。

4.4 结果与分析

对比三种方法的性能如下图所示：

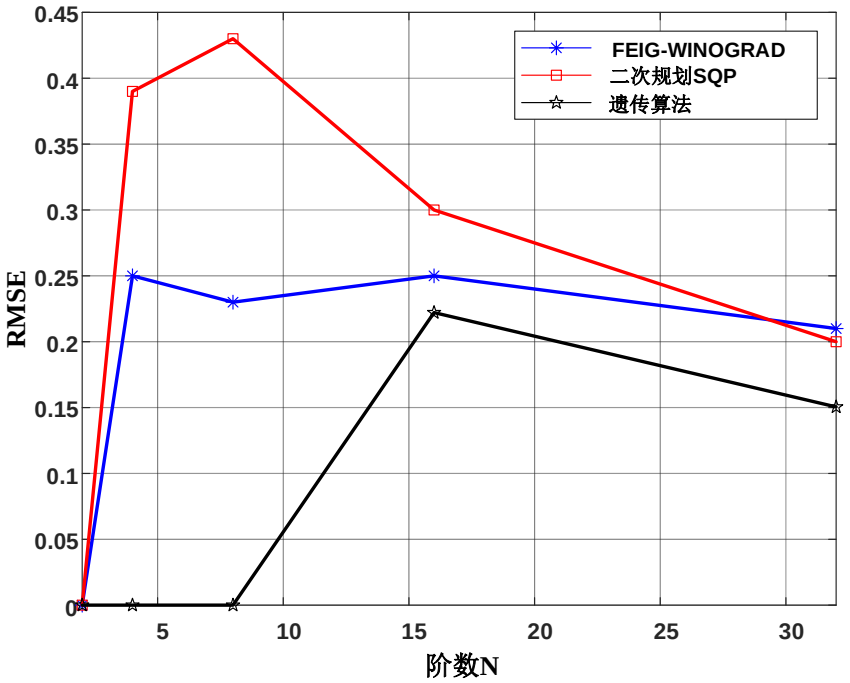

图8 三种算法的 RMSE 对比图

如图 8 所示，对比三种算法的 RMSE 可知，由于**三种算法的复杂度都为 0**，遗传算法的 RMSE 最小，可知遗传算法的性能最优。

4.5 总结与评价

问题二共使用了三种算法，各有优缺点。其中基于序列二次规划 SQP 的分解算法精度最差，遗传算法精度最高，FEIG-WINOGRAD 矩阵映射算法精度介于两者之间。但是序列二次规划 SQP 代码实现简单，遗传算法的编程实现则比较复杂，训练时间也比其他两种算法长。FW 矩阵映射算法训练时间最短，但是在矩阵分解部分依赖特定公式，适用性不高

5 问题三的模型建立与求解

5.1 问题分析

本节中，我们承接问题二中算法选型相关分析，增加限制稀疏矩阵的每行元素个数这个限制条件，对遗传算法进行改善。增加优化搜索算法，即使用对角阵分解，并使每个对角阵的非零元素取值为整数集 P 中的值。最后对比几类算法的估计结果。

5.2 模型建立

问题三的数学模型页与问题二类似，只是多加了 $\mathbf{A}_k$ 必须为稀疏矩阵的约束条件，修改后的目标函数和约束条件具体如下：

目标函数：

1) 目标函数最小化

$$F = \min\left\{RMSE(\mathcal{A},\beta)q_1 + Bq_2\right\} \tag{40}$$

其中：q 为权重，$q_1=0.7$，$q_2=0.3$。

2) 优化目标一

使计算误差最小，即：

$$RMSE(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\left\|\mathbf{F}_N-\beta\mathbf{A}_K\cdots\mathbf{A}_2\mathbf{A}_1\right\|_F^2} \tag{41}$$

3) 优化目标二

要优化 β，使 β 的复杂度也最小，即：

$$B=\left|\log_2\beta\right| \tag{42}$$

约束条件：

限定 $\mathcal{A}$ 每个矩阵 $\mathbf{A}_k$ 满足以下要求：

$$\begin{gathered}\mathbf{A}_k[l,m]\in\{x+jy\mid x,y\in\mathcal{P}\},\mathcal{P}=\{0,\pm1,\pm2,\ldots,+2^{q-1}\},\\ k=1,2,\ldots,K;l,m=1,2,\ldots,N\end{gathered} \tag{43}$$

其中，$\mathbf{A}_k[l,m]$表示矩阵 $\mathbf{A}_k$ 第 l 行第 m 列的元素。

约束条件：

1) 每个矩阵的每行至多只有 2 个非零元素

用 $\mathbf{P}_1,\mathbf{P}_2,\cdots,\mathbf{P}_K$ 矩阵分别表示对应 $\mathbf{A}_1,\mathbf{A}_2,\cdots,\mathbf{A}_K$ 矩阵非零元素的位置

$$\sum_{j=1}^{N}\mathbf{P}_k(i,j)\le 2 \tag{44}$$

其中 $k=1,2,\ldots,K$，$i=1,2,\ldots,N$，$\mathbf{P}_k(i,j)$ 表示 $\mathbf{P}_k$ 矩阵第 i 行第 j 列的元素。

2) 限定 $\mathcal{A}$ 每个矩阵 $\mathbf{A}_k$ 满足以下要求：

$$\begin{gathered}\mathbf{A}_k[l,m]\in\{x+jy\mid x,y\in\mathcal{P}\},\mathcal{P}=\{0,\pm1,\pm2,\ldots,+2^{q-1}\},\\ k=1,2,\ldots,K;l,m=1,2,\ldots,N\end{gathered} \tag{45}$$

其中，$\mathbf{A}_k[l,m]$表示矩阵 $\mathbf{A}_k$ 第 l 行第 m 列的元素。

5.3 问题求解

问题一中，$N=2,4$ 的分解结果依然满足问题三的限制条件，所以我们只计算 $N=8,16,32$ 的分解方法。

5.3.1 基于优化搜索的分解方法

由于稀疏矩阵的元素个数有限制，在一定程度上削减了遍历规模，可以考虑用一种优化搜索方式找出目标矩阵。该方法基于问题一中的 Cooley-Tukey 算法分解，得到初步分解结果后再对每一个稀疏矩阵进行元素范围约束，遍历所有可能的复数，寻找逼近 DFT 矩阵误差最小的最优方案，具体流程如图 9 所示。

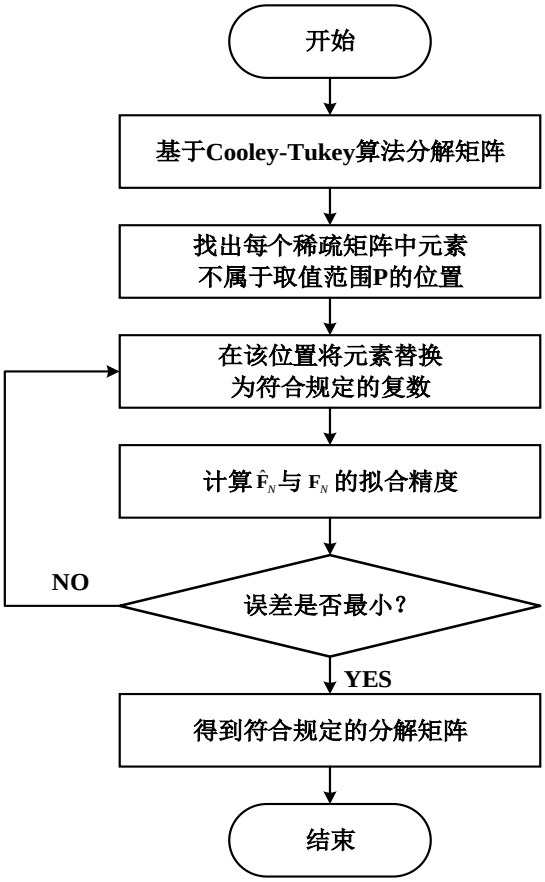

图9 优化搜索算法流程图

在搜索过程中，固定 $\beta=\sqrt{N}$，以 8 阶 DFT 矩阵为例，分解后的 RMS 误差为 0.125，复杂度 C 为 0，分解矩阵 $\hat{\mathbf{F}}_8=\mathbf{A}_3\mathbf{A}_2\mathbf{A}_1\mathbf{P}$ 如下：

$$\hat{\mathbf{F}}_8=\begin{pmatrix}1&1&1&1&1&1&1&1\\1&0&-j&0&-1&0&j&0\\1&-j&-1&j&1&-j&-1&j\\1&0&j&0&-1&0&-j&0\\1&-1&1&-1&1&-1&1&-1\\1&0&-j&0&-1&0&j&0\\1&j&-1&-j&1&j&-1&-j\\1&0&j&0&-1&0&-j&0\end{pmatrix},\ \mathbf{A}_1=\begin{pmatrix}1&1&&&&&&\\1&-1&&&&&&\\&&1&1&&&&\\&&1&-1&&&&\\&&&&1&1&&\\&&&&1&-1&&\\&&&&&&1&1\\&&&&&&1&-1\end{pmatrix},$$

$$\mathbf{A}_2=\begin{pmatrix}1&&1&&&&&\\&1&&-j&&&&\\1&&-1&&&&&\\&1&&j&&&&\\&&&&1&&1&\\&&&&&1&&-j\\&&&&1&&-1&\\&&&&&1&&j\end{pmatrix},\ \mathbf{A}_3=\begin{pmatrix}1&&&&1&&&\\&1&&&&&&\\&&1&&&&&-j\\&&&1&&&&\\1&&&&-1&&&\\&1&&&&&&\\&&1&&&&&j\\&&&1&&&&\end{pmatrix},$$

$\mathbf{P}=(\mathbf{e}_1|\mathbf{e}_5|\mathbf{e}_3|\mathbf{e}_7|\mathbf{e}_2|\mathbf{e}_6|\mathbf{e}_4|\mathbf{e}_8)$.

因为基于 Cooley-Tukey 算法分解的 4 阶矩阵已经满足问题三的要求，故该算法只优化了 8 阶、16 阶、32 阶的稀疏矩阵，具体性能如表 4 4.3.1 表 3 所示。

表5 优化搜索算法性能表

N	2	4	8	16	32
RMSE	-	-	0.125	0.062	0.031
硬件复杂度 C	-	-	0	0	0

5.3.2 基于遗传算法的分解方法

1) 阶数 N=8 时

基于问题一求得的分解结果，N=8 时，得到的分解结果只有 $\mathbf{A}_3$ 的第 6 列和第 8 列不满足本题的约束条件，引用 3.4 节中的 $\mathbf{A}_3$ 如下

$$\mathbf{A}_3=\begin{bmatrix}1&0&0&0&1&0&0&0\\0&1&0&0&0&W_8&0&0\\0&0&1&0&0&0&W_8^2&0\\0&0&0&1&0&0&0&W_8^3\\1&0&0&0&-1&0&0&0\\0&1&0&0&0&-W_8&0&0\\0&0&1&0&0&0&-W_8^2&0\\0&0&0&1&0&0&0&-W_8^3\end{bmatrix}$$

我们采用遗传算法对 $\mathbf{A}_3$ 进行优化，为了简化优化思路，提高搜索效率，我们上式圈出的不满足约束条件的元素进行优化，得到拟合出的结果 $\hat{\mathbf{A}}_3$ 为

$$\hat{\mathbf{A}}_3=\begin{bmatrix}1&0&0&0&1&0&0&0\\0&1&0&0&0&-0.25-j&0&0\\0&0&1&0&0&0&W_8^2&0\\0&0&0&1&0&0&0&1+0.25j\\1&0&0&0&-1&0&0&0\\0&1&0&0&0&1-j&0&0\\0&0&1&0&0&0&-W_8^2&0\\0&0&0&1&0&0&0&-0.5-j\end{bmatrix}$$

同时得到 β**=0.9 达到最小误差 0.0645**，并等得到了适应度进化曲线如下。

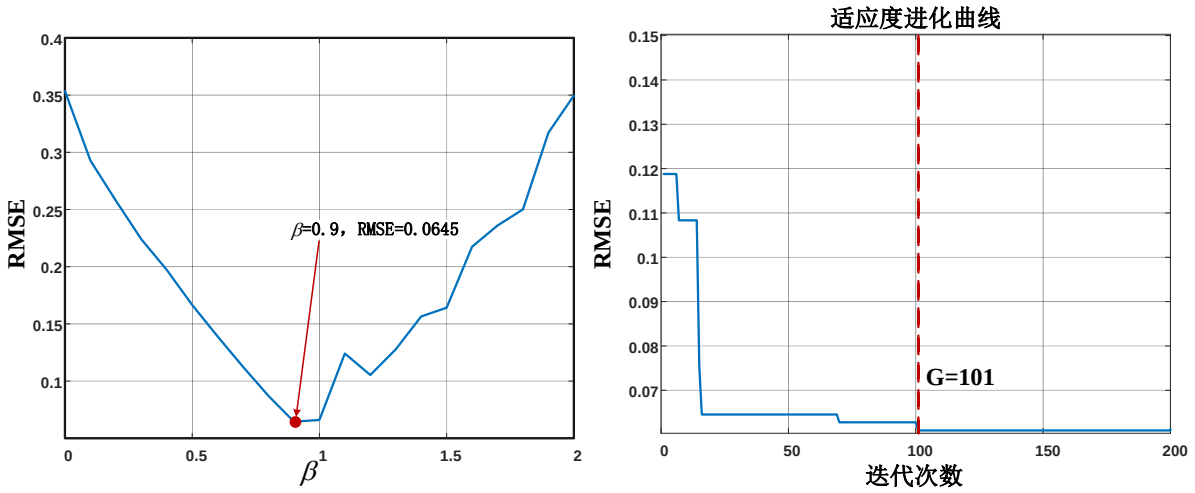

图10 N=8 时的搜索结果

如图 10 所示，**当 β=0.9 时，最终的 RMSE=0.064**，并在第 101 代完成了收敛。

2) 阶数 N=16 时

基于问题一求得的分解结果，N=16 时，$\mathbf{A}_3$和$\mathbf{A}_4$都需要进行优化，我们现在$\mathbf{A}_3$不满足约束条件的情况下，用遗传算法先优化$\mathbf{A}_4$，得到最优的$\hat{\mathbf{A}}_4$和β_1，然后再将$\hat{\mathbf{A}}_4$带入，求得最优的$\hat{\mathbf{A}}_3$和β_2。$\hat{\mathbf{A}}_3$和$\hat{\mathbf{A}}_4$即为拟合出的最优解，而β的最优拟合值即为β_2。

Step1：优化 $\hat{\mathbf{A}}_4$

采用遗传算法，得到拟合出来的$\hat{\mathbf{A}}_4$为，此时的 RMSE=0.0751。

$$\hat{\mathbf{A}}_4=\begin{bmatrix}
1 & & & & 1 & & & & & & & & & & & \\
 & 1 & & & & -0.25-2j & & & & & & & & & & \\
 & & 1 & & & & 1+2j & & & & & & & & & \\
 & & & 1 & & & & 1+1j & & & & & & & & \\
 & & & & 1 & & & & W_{16}^{4} & & & & & & & \\
 & & & & & 1 & & & & 0.5-4j & & & & & & \\
 & & & & & & 1 & & & & -0.5-2j & & & & & \\
 & & & & & & & 1 & & & & -1-0.5j & & & & \\
1 & & & & -1 & & & & & & & & & & & \\
 & 1 & & & & -2+j & & & & & & & & & & \\
 & & 1 & & & & 1+2j & & & & & & & & & \\
 & & & 1 & & & & -1-2j & & & & & & & & \\
 & & & & 1 & & & & -W_{16}^{4} & & & & & & & \\
 & & & & & 1 & & & & 0.25-0.5j & & & & & & \\
 & & & & & & 1 & & & & -2+0.5j & & & & & \\
 & & & & & & & 1 & & & & -2+j & & & &
\end{bmatrix}$$

Step2：优化 $\hat{\mathbf{A}}_3$和 β

将拟合得出的最优$\hat{\mathbf{A}}_4$代替$\mathbf{A}_4$，继续采用遗传算法求得$\hat{\mathbf{A}}_3$为

$$\hat{\mathbf{A}}_3 = \begin{bmatrix} 1 & & 1 & & & & & & & & & & & & & \\ & 1 & & 0.25-4j & & & & & & & & & & & & \\ & & 1 & & W_{16}^4 & & & & & & & & & & & \\ & & & 1 & & -2+j & & & & & & & & & & \\ 1 & & & & -1 & & & & & & & & & & & \\ & 1 & & & & -2-0.5j & & & & & & & & & & \\ & & 1 & & & & -W_{16}^4 & & & & & & & & & \\ & & & 1 & & & & -0.25+2j & & & & & & & & \\ & & & & & & & & 1 & & 1 & & & & & \\ & & & & & & & & & 1 & & 2-2j & & & & \\ & & & & & & & & & & 1 & & W_{16}^4 & & & \\ & & & & & & & & & & & 1 & & -2-j & & \\ & & & & & & & & 1 & & -1 & & & & & \\ & & & & & & & & & 1 & & 2-4j & & & & \\ & & & & & & & & & & 1 & & -W_{16}^4 & & & \\ & & & & & & & & & & & 1 & & -1+2j & & \end{bmatrix}$$

并绘制两次迭代的适应度曲线如下：

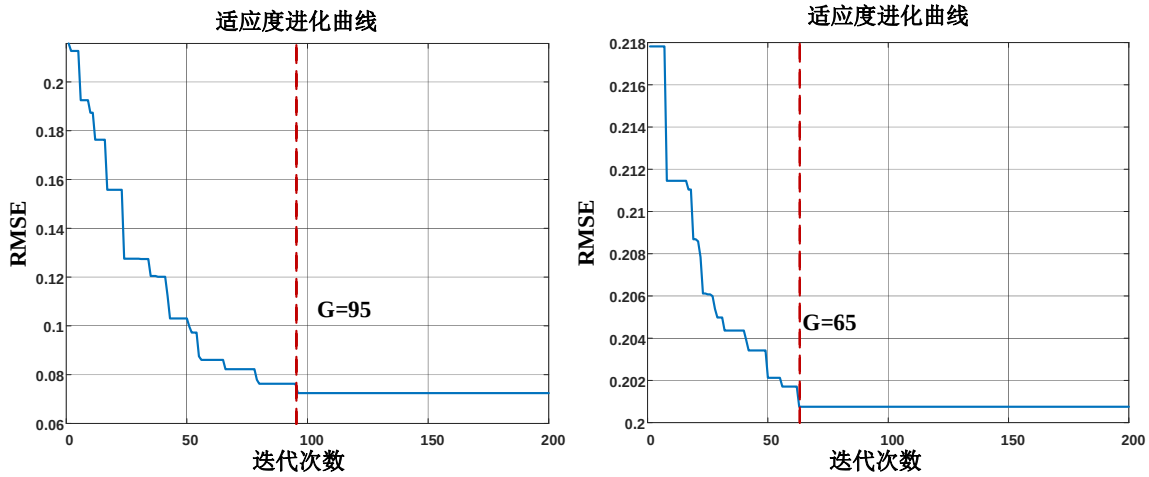

图11 N=16 两次迭代结果

如图 11 所示，在对在矩阵 $\mathbf{A}_4$ 和 $\mathbf{A}_3$ 进行优化时，分别在第 95 代和第 65 代完成了收敛，且每次优化 RMSE 都有所增加，当 β **=0.4 时，最终的 RMSE=0.2008**。

3) 阶数 N=32 时

方法同 N=16 时，当 β **=0.26 时，最终的 RMSE=0.3465**。

5.4 结果与分析

对比两种方法的性能如下图所示：

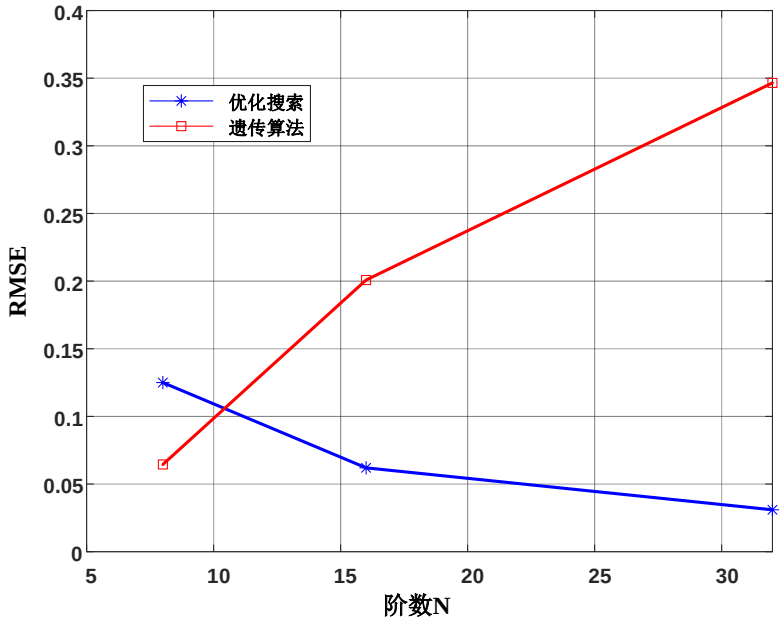

图12 两种算法的 RMSE 对比图

如图 12 所示，对比两种算法的 RMSE 可知，**优化算法的 RMSE 在 N=16 和 N=32 时更低，在 N=8 时略高，且优化算法的复杂度都为 0，而遗传算法的复杂度与问题一相同，在 N=32 为 5120，故优化算法更好。**

5.5 总结与评价

问题三共使用两种算法对矩阵进行约束，可以看出优化搜索算法性能优秀，在误差较小的情况下能使计算复杂度降为 0。遗传算法同时对稀疏矩阵和 β 进行了优化，而优化搜索算法固定了 $\beta=\sqrt{N}$，故遗传算法精度更高，但是复杂度同时也提升了

6 问题四的模型建立与求解

6.1 问题分析

问题四要求把矩阵 $\mathbf{F}=\mathbf{F}_4\otimes\mathbf{F}_8$ 分解为若干个稀疏矩阵相乘，并且分解后的稀疏矩阵同时满足约束 1 和约束 2。求解问题四的基本思路是“由大化小，分而治之”。对此，本文提出一种“混合积分解—降维寻优法”，其中的混合积分解方法是结合 Cooley-Tukey 算法、Kronecker 积的运算性质、矩阵初等行变换这三种基本原理[4] [5] [6] 推导出的。

具体求解过程将在 6.3 小节中详细描述。

6.2 模型建立

分解之后的矩阵与误差及硬件复杂度之间的对应关系可以建立模型来表达，前文中已模型中相关符号进行定义，根据问题描述和表 1 内给出的参数变量符号，分解之后的矩阵与误差及硬件复杂度之间的关系可以采用以下数学模型来表示：

目标函数：

问题三的优化目标有三个，分别是变量$\boldsymbol{\mathcal{A}}$，β 和乘法器个数 L，这是典型的多目标问题，我们对每个目标进行加权，使其变成单目标问题，具体如下：

1) 目标函数最小化

$$F=\min\left\{RMSE(\mathcal{A},\beta)q_1+Bq_2+Lq_3\right\} \tag{46}$$

其中：q 为权重，$q_1=0.6$，$q_2=0.2$，$q_3=0.2$。

2) 优化目标一

使计算误差最小，即：

$$RMSE(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\left\|\mathbf{F}_N-\beta\mathbf{A}_K\cdots\mathbf{A}_2\mathbf{A}_1\right\|_F^2} \tag{47}$$

需要注意的是，这里的$\mathbf{F}_N$不是 DFT 矩阵，而是两个 DFT 矩阵的混积

3) 优化目标二

要优化β，使β的复杂度也最小，定义 B 的定义与 P 相似，使β能用更小的位宽表示出来

$$B=\left|\log_2\beta\right| \tag{48}$$

4) 优化目标三

用$\mathbf{D}_1,\mathbf{D}_2,\cdots,\mathbf{D}_K$矩阵分别表示对应$\mathbf{A}_1,\mathbf{A}_2,\cdots,\mathbf{A}_K$矩阵非简单元素的位置（0、±1、$\pm j$不需要计算复杂度，所以把它们记为简单元素），即非简单元素标记为 1，简单元素标记为 0，得到$\mathbf{D}_1,\mathbf{D}_2,\cdots,\mathbf{D}_K$矩阵为二进制矩阵。

把$\mathbf{A}_1,\mathbf{A}_2,\cdots,\mathbf{A}_K$矩阵映射到$\mathbf{D}_1,\mathbf{D}_2,\cdots,\mathbf{D}_K$矩阵的操作记为$dif$，$\mathbf{D}_k=dif(\mathbf{A}_k)$，得到乘法器的个数$L$为

$$L=\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbf{D}_K dif(\mathbf{A}_{K-1}\cdots\mathbf{A}_2\mathbf{A}_1)+\cdots+\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbf{D}_3 dif(\mathbf{A}_2\mathbf{A}_1)+\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbf{D}_2\mathbf{D}_1 \tag{49}$$

约束条件：

用$\mathbf{P}_1,\mathbf{P}_2,\cdots,\mathbf{P}_K$矩阵分别表示对应$\mathbf{A}_1,\mathbf{A}_2,\cdots,\mathbf{A}_K$矩阵非零元素的位置，即非零元素标记为 1，零元素标记为 0，得到$\mathbf{P}_1,\mathbf{P}_2,\cdots,\mathbf{P}_K$矩阵为二进制矩阵。

每个矩阵的每行至多只有 2 个非零元素

$$\sum_{j=1}^{N}\mathbf{P}_k(i,j)\le 2 \tag{50}$$

其中$k=1,2,\ldots,K$，$i=1,2,\ldots,N$，$\mathbf{P}_k(i,j)$表示$\mathbf{P}_k$矩阵第i行第j列的元素。

6.3 问题求解

6.3.1 基于“混合积分解—降维寻优”算法 I 的分解方法

根据 6.1 小节的问题分析，“混合积分解—降维寻优法”简单描述为以下四步：

第一步，分别对$\mathbf{F}_4$和$\mathbf{F}_8$做高精度分解。采用问题一的 Cooley-Tukey 算法，易得

$$\mathbf{F}_4\otimes\mathbf{F}_8=\beta\left(\mathbf{B}_2\mathbf{B}_1\mathbf{P}_{\mathbf{N4}}\right)\otimes\left(\mathbf{C}_3\mathbf{C}_2\mathbf{C}_1\mathbf{P}_{\mathbf{N8}}\right) \tag{51}$$

第二步，分离乘积项。把$\mathbf{F}_4\otimes\mathbf{F}_8$分解为若干个乘积项。根据混积性质，可得式(52)所示的表达式

$$\mathbf{F}_4\otimes\mathbf{F}_8=\beta\left(\mathbf{I}_4\otimes\mathbf{C}_3\right)\left(\mathbf{B}_2\otimes\mathbf{C}_2\right)\left(\mathbf{B}_1\otimes\mathbf{C}_1\right)\left(\mathbf{P}_{\mathbf{N4}}\otimes\mathbf{P}_{\mathbf{N8}}\right) \tag{52}$$

第三步，分解成稀疏矩阵，使其满足约束 1。把式(52)等式右端的每一个乘积项都分解为稀疏矩阵。分析问题一中$\mathbf{F}_4$和$\mathbf{F}_8$稀疏子矩阵的特点，可得

$$\mathbf{F}_4\otimes\mathbf{F}_8=\beta\left(\mathbf{C}_3\right)\left(\mathbf{P}_2\Lambda_2\right)\left(\mathbf{P}_1\Lambda_1\right)\left(\mathbf{P}_0\right) \tag{53}$$

第四步，对$\mathbf{C}_3$做寻优，使其满足约束 2。由于$\mathbf{P}_3,\Lambda_3,\mathbf{P}_2,\Lambda_2,\mathbf{P}_1$这 5 个子矩阵都是同时满足约束 1 和约束 2 的稀疏矩阵，并且$\left(\mathbf{P}_2\Lambda_2\right),\left(\mathbf{P}_1\Lambda_1\right),\left(\mathbf{P}_0\right)$三项分别是

$(\mathbf{B}_2 \otimes \mathbf{C}_2),(\mathbf{B}_1 \otimes \mathbf{C}_1),(\mathbf{P}_{N4} \otimes \mathbf{P}_{N8})$这三项的精确拟合，因此只需$\mathbf{C}_3$这个 8 阶矩阵进行寻优即可得到$\mathbf{F} = \mathbf{F}_4 \otimes \mathbf{F}_8$分解的最优解。

相应的流程图如图 13 所示。

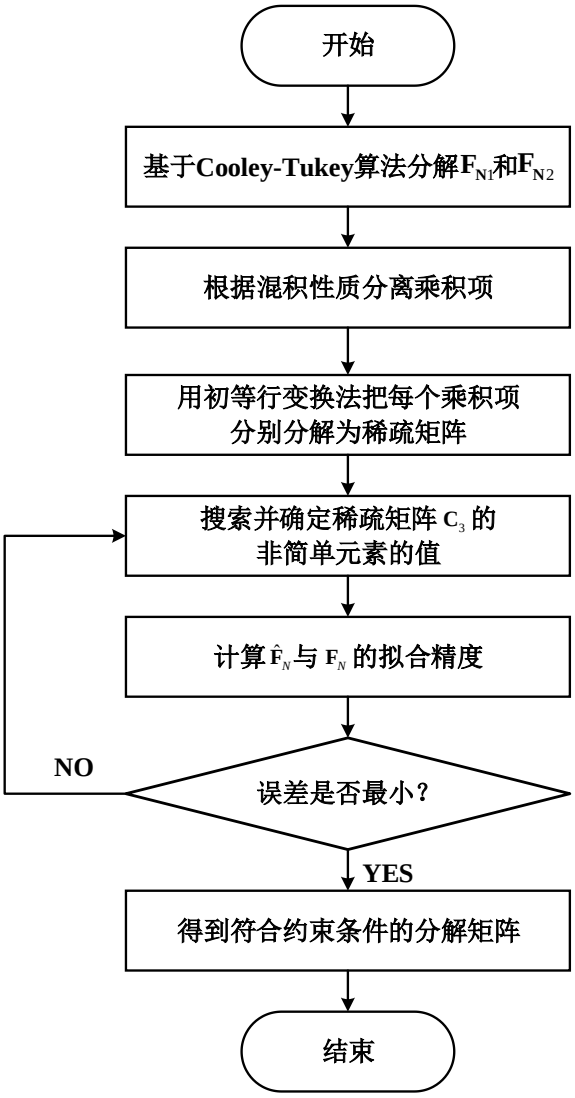

图13 混合积分解-降维寻优算法 I 流程图

下面详细描述第三步和第四步的公式推导过程。

第三步，把式(52)再次书写如下：

$$\mathbf{F}_4 \otimes \mathbf{F}_8 = \beta(\mathbf{I}_4 \otimes \mathbf{C}_3)(\mathbf{B}_2 \otimes \mathbf{C}_2)(\mathbf{B}_1 \otimes \mathbf{C}_1)(\mathbf{P}_{N4} \otimes \mathbf{P}_{N8}) \tag{54}$$

我们发现分解后的稀疏矩阵$\mathbf{B}_2,\mathbf{B}_1,\mathbf{C}_3,\mathbf{C}_2,\mathbf{C}_1$具有某种特定的规律，这种规律可以把$(\mathbf{B}_2 \otimes \mathbf{C}_2)$和$(\mathbf{B}_1 \otimes \mathbf{C}_1)$这两项分别分解为一个稀疏矩阵与一个对角矩阵（或满足约束 1 的分块对角阵）相乘的形式。下面对该式中的每一个乘积项单独分析。

（1）计算第一项$(\mathbf{P}_{N4} \otimes \mathbf{P}_{N8})$

$\mathbf{P}_{N4}$和$\mathbf{P}_{N8}$分别是$\mathbf{F}_4$和$\mathbf{F}_8$的列变换矩阵，每行仅有一个非零元，且所有非零元素均为 1，因此$(\mathbf{P}_{N4} \otimes \mathbf{P}_{N8})$是满足约束 1 和约束 2 的稀疏矩阵，记

$$\mathbf{A}_1 = (\mathbf{P}_{N4} \otimes \mathbf{P}_{N8}) \tag{55}$$

（2）计算第二项$\left(\mathbf{B}_1 \otimes \mathbf{C}_1\right)$

根据 4.4 小节，$\mathbf{B}_1$和$\mathbf{C}_1$的值分别如下，

$$\mathbf{B}_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}, \mathbf{C}_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{56}$$

观察发现$\mathbf{B}_1$和$\mathbf{C}_1$是分块对角矩阵，且每个子块都相同，为了简化表达，记这个子块为

$$\mathbf{R} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{57}$$

则，$\mathbf{B}_1$和$\mathbf{C}_1$可记为

$$\mathbf{B}_1 = \mathbf{I}_2 \otimes \mathbf{R}, \mathbf{C}_1 = \mathbf{I}_4 \otimes \mathbf{R} \tag{58}$$

写成分块矩阵的形式更直观

$$\mathbf{B}_1 = \begin{bmatrix} \mathbf{R} & \\ & \mathbf{R} \end{bmatrix}, \mathbf{C}_1 = \begin{bmatrix} \mathbf{R} & & & \\ & \mathbf{R} & & \\ & & \mathbf{R} & \\ & & & \mathbf{R} \end{bmatrix} \tag{59}$$

由于初等变换矩阵和对角矩阵必然是稀疏矩阵，所以把$\mathbf{R}$可分解为 2 个初等行变换矩阵和 1 个对角矩阵的乘积

$$\mathbf{R} = \mathbf{P}_{\mathbf{R}2}\mathbf{P}_{\mathbf{R}1}\mathbf{R}_{diag} \tag{60}$$

由于上式均是二阶矩阵，因此很容易求解出等式右端的矩阵$\mathbf{P}_{\mathbf{R}2}, \mathbf{P}_{\mathbf{R}1}, \mathbf{R}_{diag}$，如下式所示，

$$\mathbf{P}_{\mathbf{R}2} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \mathbf{P}_{\mathbf{R}1} = \begin{bmatrix} 1 & -1/2 \\ 0 & 1 \end{bmatrix}, \mathbf{R}_{diag} = \begin{bmatrix} 1 & 0 \\ 0 & -2 \end{bmatrix} \tag{61}$$

把 2 个初等行变换矩阵合成 1 个变换矩阵，记为

$$\mathbf{P}_{\mathbf{R}} = \mathbf{P}_{\mathbf{R}2}\mathbf{P}_{\mathbf{R}1} \tag{62}$$

那么$\mathbf{B}_1$可进一步表示为

$$\begin{aligned} \mathbf{B}_1 &= \left(\mathbf{I}_2\right) \otimes \left(\mathbf{P}_{\mathbf{R}}\mathbf{R}_{diag}\right) = \left(\mathbf{I}_2\mathbf{I}_2\right) \otimes \left(\mathbf{P}_{\mathbf{R}}\mathbf{R}_{diag}\right) \\ &= \left(\mathbf{I}_2 \otimes \mathbf{P}_{\mathbf{R}}\right)\left(\mathbf{I}_2 \otimes \mathbf{R}_{diag}\right) \end{aligned} \tag{63}$$

则$\left(\mathbf{B}_1 \otimes \mathbf{C}_1\right)$可一进步表示为

$$\begin{aligned} \left(\mathbf{B}_1 \otimes \mathbf{C}_1\right) &= \left(\mathbf{I}_2 \otimes \mathbf{P}_{\mathbf{R}}\right)\left(\mathbf{I}_2 \otimes \mathbf{R}_{diag}\right) \otimes \mathbf{C}_1 \\ &= \left[\left(\mathbf{I}_2 \otimes \mathbf{P}_{\mathbf{R}}\right)\left(\mathbf{I}_2 \otimes \mathbf{R}_{diag}\right)\right] \otimes \left[\mathbf{C}_1\mathbf{I}_8\right] \\ &= \left[\left(\mathbf{I}_2 \otimes \mathbf{P}_{\mathbf{R}}\right) \otimes \left(\mathbf{C}_1\right)\right] \bullet \left[\left(\mathbf{I}_2 \otimes \mathbf{R}_{diag}\right) \otimes \mathbf{I}_8\right] \end{aligned} \tag{64}$$

根据$\mathbf{P}_{\mathbf{R}}$和$\mathbf{R}_{diag}$本身的特性，可得$\left(\mathbf{I}_2\otimes\mathbf{P}_{\mathbf{R}}\right)$和$\left(\mathbf{I}_2\otimes\mathbf{R}_{diag}\right)$每行仅有 1 个非零元素，而$\mathbf{C}_1$每行有 2 个非零元素，所以$\left(\mathbf{I}_2\otimes\mathbf{P}_{\mathbf{R}}\right)\otimes\mathbf{C}_1$和每行仅有 2 个非零元素，$\left(\mathbf{I}_2\otimes\mathbf{R}_{diag}\right)\otimes\mathbf{I}_8$为对角矩阵，记式(64)中等号右端第三行的 2 个乘积项分别为

$$\mathbf{P}_2=\left(\mathbf{I}_2\otimes\mathbf{P}_{\mathbf{R}}\right)\otimes\left(\mathbf{C}_1\right)\qquad\Lambda_2=\left(\mathbf{I}_2\otimes\mathbf{R}_{diag}\right)\otimes\mathbf{I}_8 \tag{65}$$

则第二项$\left(\mathbf{B}_1\otimes\mathbf{C}_1\right)$的分解结果如下

$$\left(\mathbf{B}_1\otimes\mathbf{C}_1\right)=\mathbf{P}_1\Lambda_1 \tag{66}$$

（3）计算第三项$\left(\mathbf{B}_2\otimes\mathbf{C}_2\right)$

与（2）同理，可得分解结果

$$\left(\mathbf{B}_2\otimes\mathbf{C}_2\right)=\mathbf{P}_2\Lambda_2 \tag{67}$$

需要注意的是，第三项$\left(\mathbf{B}_2\otimes\mathbf{C}_2\right)$无法写成类似于式(59)的分块对角矩阵形式，但可以把整个$\left(\mathbf{B}_2\otimes\mathbf{C}_2\right)$写成如下的分块矩阵形式

$$\left(\mathbf{B}_2\otimes\mathbf{C}_2\right)=\begin{bmatrix}\mathbf{F}_{8\times8} & & \mathbf{F}_{8\times8} & \\ & \mathbf{F}_{8\times8} & & -j\bullet\mathbf{F}_{8\times8}\\ \mathbf{F}_{8\times8} & & -\mathbf{F}_{8\times8} & \\ & \mathbf{F}_{8\times8} & & j\bullet\mathbf{F}_{8\times8}\end{bmatrix}_{32x32} \tag{68}$$

基于式(68)，可以先对矩阵进行分块行变换，可分解出分块对角矩阵

$$\Lambda_2=\begin{bmatrix}\mathbf{F}_{8\times8} & & & \\ & \mathbf{F}_{8\times8} & & \\ & & -2\mathbf{F}_{8\times8} & \\ & & & -2\,j\bullet\mathbf{F}_{8\times8}\end{bmatrix} \tag{69}$$

由于子块$\mathbf{F}_{8\times8}$每行仅有 2 个非零元素，满足约束 1。且经过计算得到分块行变换$\mathbf{P}_2$也是满足约束 1 的稀疏矩阵

（4）计算第四项$\left(\mathbf{I}_4\otimes\mathbf{C}_3\right)$

由问题一的求解结果知$\mathbf{C}_3$是满足约束 1 的稀疏矩阵，则$\left(\mathbf{I}_4\otimes\mathbf{C}_3\right)$必然也是满足约束 1 的稀疏矩阵。

第四步，经过上述分解过程容易得到，式(53)中的稀疏子矩阵里，只有$\mathbf{C}_3$不满足约束 2，因此可以采用问题三种的寻优算法求解$\mathbf{C}_3$的最优值。

本文采用 MATLAB 实现上述算法，由于$\mathbf{F}=\mathbf{F}_4\otimes\mathbf{F}_8$分解后的子矩阵都是 32 阶的，收纸张大小限制无法显示，因此$\mathbf{F}=\beta\left(\mathbf{C}_3\right)\left(\mathbf{P}_2\Lambda_2\right)\left(\mathbf{P}_1\Lambda_1\right)\left(\mathbf{P}_0\right)$的分解结果详见附件，得到 RMSE 的值为 0.04。

6.3.2 基于“混合积分解—降维寻优”算法 II 的分解方法

该方法根据 Kronecker 积的结合律和混合积性质把$\mathbf{F}_{N1}$和$\mathbf{F}_{N2}$拆分成稀疏矩阵相乘，再进行变换最终得到稀疏矩阵的积。具体过程如下，首先将$\mathbf{F}_4$和$\mathbf{F}_8$根据问题一中的 Cooley-Tukey 算法拆分为

$$\begin{aligned}\mathbf{F}_4&=\mathbf{A}_{42}\mathbf{A}_{41}\mathbf{P}_4\\ \mathbf{F}_8&=\mathbf{A}_{83}\mathbf{A}_{82}\mathbf{A}_{81}\mathbf{P}_8\end{aligned} \tag{70}$$

可以看出 $\mathbf{A}_{41}$ 和 $\mathbf{A}_{81}$ 可进一步准确的拆解为

$$\begin{aligned}\mathbf{A}_{41} &= \mathbf{B}\otimes\mathbf{I}_2\\ \mathbf{A}_{81} &= \mathbf{B}\otimes\mathbf{I}_4\end{aligned} \tag{71}$$

其中

$$\mathbf{B}=\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix},$$

$\mathbf{I}_2$、$\mathbf{I}_4$ 分别为 2 阶、4 阶单位阵。

而 $\mathbf{A}_{42}$、$\mathbf{A}_{82}$、$\mathbf{A}_{83}$ 无法精准的分为两个简单元素矩阵的 Kronecker 积，故采用近似的矩阵 $\hat{\mathbf{A}}_{42}$、$\hat{\mathbf{A}}_{82}$、$\hat{\mathbf{A}}_{83}$ 来代替：

$$\hat{\mathbf{A}}_{42} = \mathbf{B}\otimes\mathbf{R}_{42} \tag{72}$$

$$\hat{\mathbf{A}}_{82} = \mathbf{B}\otimes\mathbf{R}_{42}\otimes\mathbf{I}_2 \tag{73}$$

$$\hat{\mathbf{A}}_{83} = \mathbf{B}\otimes\mathbf{R}_{83} \tag{74}$$

其中

$$\mathbf{R}_{42}=\begin{pmatrix}1 & 0\\ 0 & -j\end{pmatrix},$$

$$\mathbf{R}_{83}=\begin{pmatrix}1 & & & \\ & 0.707-0.707j & & \\ & & -j & \\ & & & -0.707-0.707j\end{pmatrix}.$$

故 $\mathbf{F}_{32}=\mathbf{F}_4\otimes\mathbf{F}_8$ 可近似地表达为

$$\hat{\mathbf{F}} = \hat{\mathbf{F}}_4\otimes\hat{\mathbf{F}}_8 \tag{75}$$

接下来对 Kronecker 积进行组合与分解

$$\begin{aligned}\hat{\mathbf{F}} &= \hat{\mathbf{A}}_{42}\mathbf{A}_{41}\mathbf{P}_4\otimes\hat{\mathbf{A}}_{83}\hat{\mathbf{A}}_{82}\mathbf{A}_{81}\mathbf{P}_8\\ &= \mathbf{I}_4\hat{\mathbf{A}}_{42}\mathbf{A}_{41}\mathbf{P}_4\otimes\hat{\mathbf{A}}_{83}\hat{\mathbf{A}}_{82}\mathbf{A}_{81}\mathbf{P}_8\\ &= (\mathbf{I}_4\otimes\hat{\mathbf{A}}_{83})(\hat{\mathbf{A}}_{42}\otimes\hat{\mathbf{A}}_{82})(\mathbf{A}_{41}\otimes\mathbf{A}_{81})(\mathbf{P}_4\otimes\mathbf{P}_8)\\ &= (\mathbf{I}_4\otimes\hat{\mathbf{A}}_{83})(\mathbf{B}\otimes\mathbf{R}_{42}\otimes\mathbf{B}\otimes\mathbf{R}_{42}\otimes\mathbf{I}_2)(\mathbf{B}\otimes\mathbf{I}_2\otimes\mathbf{B}\otimes\mathbf{I}_4)(\mathbf{P}_4\otimes\mathbf{P}_8)\\ &= (\mathbf{I}_4\otimes\hat{\mathbf{A}}_{83})((\mathbf{BB})\otimes(\mathbf{R}_{42}\otimes\mathbf{B}\otimes\mathbf{R}_{42}\otimes\mathbf{I}_2)(\mathbf{I}_2\otimes\mathbf{B}\otimes\mathbf{I}_4))(\mathbf{P}_4\otimes\mathbf{P}_8)\\ &= (\mathbf{I}_4\otimes\hat{\mathbf{A}}_{83})((\mathbf{BB})\otimes(\mathbf{R}_{42}\mathbf{I}_2)\otimes(\mathbf{B}\otimes\mathbf{R}_{42}\otimes\mathbf{I}_2)(\mathbf{B}\otimes\mathbf{I}_4))(\mathbf{P}_4\otimes\mathbf{P}_8)\\ &= (\mathbf{I}_4\otimes\hat{\mathbf{A}}_{83})((\mathbf{BB})\otimes\mathbf{R}_{42}\otimes(\mathbf{BB})\otimes(\mathbf{R}_{42}\otimes\mathbf{I}_2)\mathbf{I}_4)(\mathbf{P}_4\otimes\mathbf{P}_8)\\ &= (\mathbf{I}_4\otimes\hat{\mathbf{A}}_{83})((\mathbf{BB})\otimes\mathbf{R}_{42}\otimes(\mathbf{BB})\otimes\mathbf{R}_{42}\otimes\mathbf{I}_2)(\mathbf{P}_4\otimes\mathbf{P}_8)\\ &= \mathbf{C}_1\mathbf{C}_2\mathbf{C}_3\end{aligned} \tag{76}$$

其中

$$\begin{aligned}\mathbf{C}_1 &= \mathbf{I}_4\otimes\hat{\mathbf{A}}_{83},\\ \mathbf{C}_2 &= (\mathbf{BB})\otimes\mathbf{R}_{42}\otimes(\mathbf{BB})\otimes\mathbf{R}_{42}\otimes\mathbf{I}_2,\\ \mathbf{C}_3 &= \mathbf{P}_4\otimes\mathbf{P}_8.\end{aligned}$$

在分解过程中利用了稀疏矩阵与对角阵的 Kronecker 积还是稀疏矩阵这一性质，以及

$$\mathbf{B}\cdot\mathbf{B}=\begin{pmatrix}2 & 0\\ 0 & 2\end{pmatrix}$$

这一特点，最终化解成立三个稀疏矩阵$\mathbf{C}_1$、$\mathbf{C}_2$、$\mathbf{C}_3$相乘，为确保满足约束 2，还需要对它们进行元素控制，基于问题三中的优化搜索算法，将$\mathbf{C}_1$、$\mathbf{C}_2$、$\mathbf{C}_3$的元素全部限制在取值范围 P 内，此时计算 RMSE 只有 0.0589，硬件复杂度为 0。最终分解结果见附件中的变量 Opti_C1、Opti_C2、Opti_C3。

6.4 结果与分析

得到两种方法的性能对比如 6.4 表 6 所示：

表6 混合积分解—降维寻优法性能对比

	方法 1	方法 2
RMSE	0.0323	0.0589
硬件复杂度 C	640	0

如上表所示，两种方法都可以得到低于 0.1 的 RMSE 值。相对而言，方法 1 具有更高的拟合精度，但复杂度也随之增大。方法 2 牺牲了一定的拟合精度，但大大降低了复杂度。两种方法应根据实际工程的指标要求进行选择。

6.5 总结与评价

6.5.1 模型优点

（1）矩阵降维处理。利用 Kronecker 积的性质，把高维矩阵分解转化为低维矩阵分解，同时在寻优的时候不必对 32 阶矩阵寻优，只需对 8 阶矩阵寻优，和问题三相比，问题四只是目标函数有所变化，寻优空间并未增大；
（2）稀疏度高。由于初等变换矩阵和对角矩阵一定是稀疏矩阵，所以从初等变换的角度出发，把矩阵逐一分解，在满足约束 1 的条件下适当地把一些初等矩阵合并相乘从而减少子矩阵的个数；
（3）较理想的寻优起点。先采用精确分解法得到满足约束 1 的子矩阵，再使用寻优算法求出同时满足约束 1 和约束 2 的最优解，这样会容易得到更理想的最优解；
（4）较完整的数学理论支撑。本文提出的“混积分解法”是根据 DFT 矩阵的特征和混积的性质进行有限步骤的矩阵分解。该分解过程有很强的规律性，论文简要总结了这些规律，并给出较详细的公式推导。该理论依据对于更高阶的 DFT 混积矩阵分解有重要的参考价值。

6.5.2 模型缺点

随着矩阵阶数的增大，DFT 矩阵经过 Cooley-Tukey 算法分解后的稀疏子矩阵的规律性可能会越复杂，DFT 混积矩阵分解后的数学表达式可能也会越复杂。因此，DFT 混积矩阵分解的数学表达式还有优化的空间。

7 问题五的模型建立与求解

7.1 问题分析

问题五在问题三的基础上增加了 RMSE 限制，主要实现难点是如何调整稀疏矩阵元素的取值范围来满足 $RMSE \le 0.1$，并权衡硬件复杂度 C。该问题的解决思路基于问题三的实现算法，对取值范围 P 进行递增搜索，使 RMSE 最小。

7.2 模型建立

问题五的数学模型页与问题三类似，只是加上了精度的限制，将精度从目标函数变为了限制条件，然后对 $\mathcal{P}$ 也进行优化，修改后的目标函数和约束条件具体如下：

目标函数：

1) 目标函数得分最小化

$$F=\min\left\{RMSE(\mathcal{A},\beta)q_1+Pq_2+\mathcal{B}q_3\right\} \tag{77}$$

其中：q 为权重，$q_1=0.6$，$q_2=0.2$，$q_3=0.2$。

2) 优化目标一得分

$$RMSE(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\left\|\mathbf{F}_N-\beta\mathbf{A}_K\cdots\mathbf{A}_2\mathbf{A}_1\right\|_F^2} \tag{78}$$

3) 优化目标二得分

限定每个元素的取值范围来减小复杂度，使 $\mathcal{P}$ 集合中的元素对 2 取对数，然后取绝对值最大的数，使其最小

$$P=\max\{\left|\log_2\mathcal{P}\right|\} \tag{79}$$

4) 优化目标三得分

要优化 β，使 β 的复杂度也最小

$$B=\left|\log_2\beta\right| \tag{80}$$

约束条件：

1) 每个矩阵的每行至多只有 2 个非零元素

用 $\mathbf{P}_1,\mathbf{P}_2,\cdots,\mathbf{P}_K$ 矩阵分别表示对应 $\mathbf{A}_1,\mathbf{A}_2,\cdots,\mathbf{A}_K$ 矩阵非零元素的位置，即非零元素标记为 1，零元素标记为 0，得到 $\mathbf{P}_1,\mathbf{P}_2,\cdots,\mathbf{P}_K$ 矩阵为二进制矩阵。

$$\sum_{j=1}^{N}\mathbf{P}_k(i,j)\le 2 \tag{81}$$

其中 $k=1,2,\ldots,K$，$i=1,2,\ldots,N$，$\mathbf{P}_k(i,j)$ 表示 $\mathbf{P}_k$ 矩阵第 i 行第 j 列的元素。

2) 限定 $\mathcal{A}$ 每个矩阵 $\mathbf{A}_k$ 满足以下要求：

$$\begin{gathered}\mathbf{A}_k[l,m]\in\{x+jy\mid x,y\in\mathcal{P}\},\mathcal{P}=\{0,\pm1,\pm2,\ldots,+2^{q-1}\},\\ k=1,2,\ldots,K;l,m=1,2,\ldots,N\end{gathered} \tag{82}$$

其中，$\mathbf{A}_k[l,m]$ 表示矩阵 $\mathbf{A}_k$ 第 l 行第 m 列的元素。

3) 限定研究矩阵的精度

$$RMSE(\mathcal{A},\beta)\le 0.1 \tag{83}$$

7.3 基于优化搜索的分解方法

不难看出问题三的分解方法在 16 阶和 32 阶的矩阵分解下已经达到了要求，故重点在于 8 阶矩阵的优化，通过调整取值范围使 $P=[0,\pm1,\pm2,\pm4,\pm8]$，该优化搜索算法能使 RMSE 达到 0.095，复杂度 C 为 0，$\hat{\mathbf{F}}_8=\mathbf{A}_3\mathbf{A}_2\mathbf{A}_1\mathbf{P}$ 具体分解情况如下：

$$\hat{\mathbf{F}}_8=\begin{pmatrix}1&1&1&1&1&1&1&1\\1&-j&-j&-1&-1&j&j&1\\1&-j&-1&j&1&-j&-1&j\\1&-j&j&1&-1&j&-j&-1\\1&-1&1&-1&1&-1&1&-1\\1&j&-j&1&-1&-j&j&-1\\1&j&-1&-j&1&j&-1&-j\\1&j&j&-1&-1&-j&-j&1\end{pmatrix},\ \mathbf{A}_1=\begin{pmatrix}1&1&&&&&&\\1&-1&&&&&&\\&&1&1&&&&\\&&1&-1&&&&\\&&&&1&1&&\\&&&&1&-1&&\\&&&&&&1&1\\&&&&&&1&-1\end{pmatrix},$$

$$\mathbf{A}_2=\begin{pmatrix}1&&1&&&&&\\&1&&-j&&&&\\1&&-1&&&&&\\&1&&j&&&&\\&&&&1&&1&\\&&&&&1&&-j\\&&&&1&&-1&\\&&&&&1&&j\end{pmatrix},\ \mathbf{A}_3=\begin{pmatrix}1&&&&1&&&\\&1&&&&8j&&\\&&1&&&&-j&\\&&&1&&&&8j\\1&&&&-1&&&\\&1&&&&-2j&&\\&&1&&&&j&\\&&&1&&&&-2j\end{pmatrix},$$

$\mathbf{P}=(\mathbf{e}_1\,|\,\mathbf{e}_5\,|\,\mathbf{e}_3\,|\,\mathbf{e}_7\,|\,\mathbf{e}_2\,|\,\mathbf{e}_6\,|\,\mathbf{e}_4\,|\,\mathbf{e}_8)$。

7.4 结果与分析

该算法依旧固定 $\beta=\frac{1}{\sqrt{N}}$，以下是精度、复杂度和取值范围 P 的对比，如表 6 所示：

表7 优化搜索算法仿真结果

N	8	16	32
RMSE	0.095	0.062	0.031
硬件复杂度 C	0	0	0
取值范围 P	$P=[0,\pm1,\pm2,\pm4,\pm8]$	$P=[0,\pm1,\pm2]$	$P=[0,\pm1,\pm2]$

基于优化搜索算法的性能如下图所示：

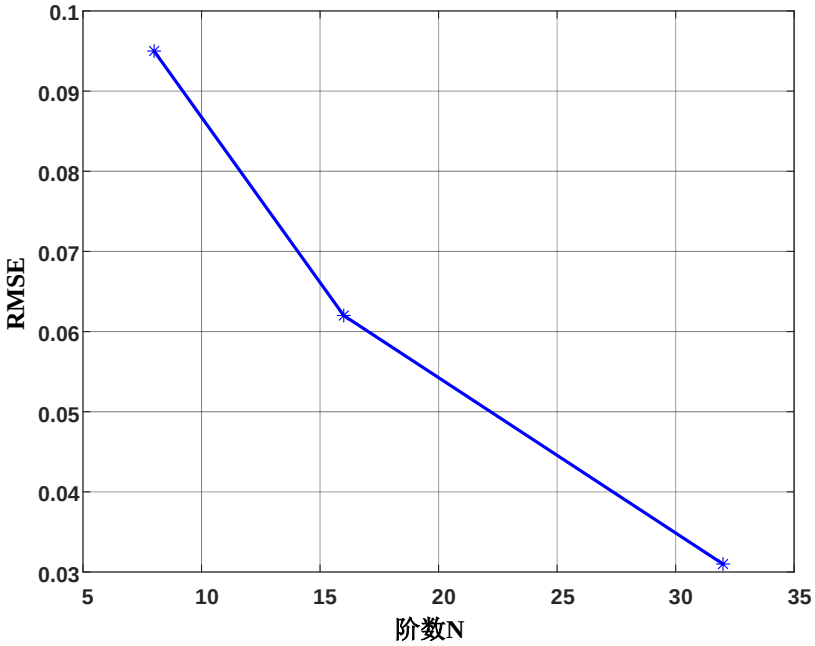

图14 基于优化搜索算法的 RMSE 对比图

如图 14 所示，基于优化搜索算法得到的 RMSE 符合问题五给出的限制条件，且复杂度非常低，**为 0**。同时，**5.3.2 中 N=8 的情况也可以作为本题的解，RMSE=0.064，且复杂度也为 0。**

7.5 总结与评价

从测试性能表来看优化搜索算法在保证 $RMSE \le 1$的同时也使硬件复杂度降到最低，并且优化了稀疏矩阵元素的取值范围 P。优化搜索代码的缺点是需要遍历可能的复数取值，取值范围 P 越大需要迭代的次数就越多，训练时间相应增加很多。不过综合来看优化搜索方法是一个不错的选择。

8 总结与展望

经过建模与分析，本文较好地解决了 DFT 矩阵分解的问题，有效降低了硬件计算复杂度，对信号的波束成形具有重大意义。针对各种约束提出了不同的算法，不过拟合精度还有提高的空间，可以优化代码继续改进。本文针对 DFT 矩阵分解作出的贡献主要包括以下几方面：

1.针对问题一给出了基于 Cooley-Tukey 算法的分解方法，具有极高的准确性。

2.针对问题二采用了基于 Feig-Winograd 矩阵映射的算法、遗传算法、序列二次规划算法进行分解，分析了各自的优劣势。

3.针对问题三采用了遗传算法，并提出了一种新的优化搜索法，在极低的复杂度下也能分解出满足题目约束的稀疏矩阵。

4.针对问题四提出了两种新的基于混合积分解-降维寻优的算法，分解出的矩阵完全符合题目条件，RMSE 都达到了 0.05。

5.针对问题五继续改进了优化搜索算法，在 N=2、4、8、16、32 的情况下全部满足了 $RMSE \le 0.1$的要求，求出的稀疏矩阵复杂度都为 0，极大地降低了硬件计算难度。

DFT 矩阵分解应用广泛，难点在于保证低复杂度的情况下还要求较高的拟合精度，目前的分解方法仍比较有限，需要进一步对分解方法进行研究，改进搜索算法。

参考文献

[1] Viduneth A,Arjuna M,Xinyao T, et al. Analog Approximate-FFT 8/16-Beam Algorithms, Architectures and CMOS Circuits for 5G Beamforming MIMO Transceivers[J]. IEEE Journal on Emerging and Selected Topics in Circuits and Systems,2018,8(3).

[2] Blahut R E. Fast algorithms for signal processing[M]. Cambridge University Press, 2010.

[3] Du J, Chen K, Yin P, et al. Design of an approximate FFT processor based on approximate complex multipliers[C]//2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2021: 308-313.

[4] Bouguezel S, Ahmad M O, Swamy M N S. A note on" Split vector-radix-2/8 2-D Fast Fourier Transform"[J]. IEEE signal processing letters, 2005, 12(3): 185.

[5] Suzuki T, Ikehara M. Integer DCT based on direct-lifting of DCT-IDCT for lossless-to-lossy image coding[J]. IEEE Transactions on image processing, 2010, 19(11): 2958-2965.

[6] Alexey P, Olga P. Methods and Algorithms for Vertical Sliding Spatial-Frequency Fourier Processing of Two-Dimensional Discrete Finite Signals[C]//2023 25th International Conference on Digital Signal Processing and its Applications (DSPA). IEEE, 2023: 1-6.

[7] Du J, Chen K, Yin P, et al. Design of an approximate FFT processor based on approximate complex multipliers[C]//2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2021: 308-313.

[8] Goldberg D E. Genetic Algorithm in search[J]. Optimization and Machine Learning, 1989 (11): 3-26.

[9] Holand J.H. Adaptation in Natural and Artificial Systems [M].Massachusetts: MIT-Press.1975.

[10] 马永杰,云文霞.遗传算法研究进展[J].计算机应用研究，2012,29(04):1201-1206+1210.

[11] 孙宝凤,申琇秀,龙书玲,卢昭宇.混流装配线的双目标投产排序决策模型[J].计算机集成制造系统，2017,23(07):1481-1491.DOI:10.13196/j.cims.2017.07.013.

[12] 刘晓霞. 种群规模对遗传算法性能影响的研究[D].华北电力大学（河北）,2010

[13] 刘佳楠 . 基于遗传算法的混流装配线平衡与排产问题研究 [D]. 青岛理工大学，2020.DOI:10.27263/d.cnki.gqudc.2020.000160.

附录

附录 1

基于 Cooley-Tukey 算法的第一问求解代码

```
clc;clear;

%参数设置
N=32;%N 维 DFT 矩阵
M=log2(N);%“级”的数量
FN = dftmtx(N)/sqrt(N);
E = cell(1,N);
A = cell(1,M);

%%%%%步骤一：求变换矩阵 P
for i=1:N
    e=zeros(N,1);
    e(i)=1;
    E{i}=e;
end
%蝶形倒序
sel_0=[1:N];

sel=permute(reshape(sel_0,2*ones(1,M)),M:-1:1);
sel=sel(:);
%产生蝶形变换矩阵 P
P=[];
for i =1:N
    P=[P,E{sel(i)}];
end


%%test
% a=[1,2,3,4,5,6,7,8].';
% c=P*a;

%%%%步骤二：求 FFT 蝶形因子，即待分解的矩阵 A1~Ak

for i=1:M%N 阶 DFT 矩阵需要分解成 M 个矩阵相乘
    n=2^i;%第 i 个矩阵是基 n-fft 运算
    d1=eye(n/2);
    d2=[];
    for j=1:n/2
        d2=blkdiag(d2,exp(-1j*2*pi*(j-1)/n));
    end
    I=[d1,d2;
    d1,-d2];
    Atemp=[];
    for k=1:N/n
        Atemp=blkdiag(Atemp,I);
    end
    A{i}=Atemp;
end


%%%%步骤三：得到估计矩阵 F_esti 并计算误差
```

```
F_esti=P;
for i =1:M
    F_esti=A{i}*F_esti;
end
beta=1/sqrt(N);
F_esti=F_esti*beta;
err = norm(F_esti-FN,'fro')/N;
%%%%步骤四：计算复杂度
[complexnum] = complexity_cal(A,N,M)
q=16;
com_qxL=complexnum*q;
```

附录 2

基于 FEIG-WINOGRAD 矩阵映射的第二问求解代码

```
%Initial
N = 8;
q = 3;
W = DFT_matrix(N);
P = [-4,-2,-1,0,1,2,4];
iter = 1000;
disp(['    N=' num2str(N) '  ']);
[esti_W,min_err] = mtx_FWdecom1(W,P,iter);

%calculate complexity
simple_num=[0,1,-1,1j,-1j];
A1=ones(N,N);
A_round=roundn(esti_W,-6);
A_pos_dir=A1-ismember(A_round,simple_num);
    C = q * sum(sum(A_pos_dir));

function [esti_W,min_err] = mtx_FWdecom1(W,P,iter)
N = size(W,1);
min_err = inf;
alpha = zeros(1,N/4);
alpha(1) = 1;
A00 = zeros(N/2,N/2);

for i = 1:iter

    for q = 2:(length(alpha)-1)
        numSelect = 2;
        randomIndex = randperm(length(P),numSelect);
        randomElements = P(randomIndex);
        a = randomElements(1) + 1j*randomElements(2);
        alpha(q) = a;
    end
    A00(1,:) = 1;
    A00(:,1) = 1;
    for m = 2:N/4
        for n = m:N/4
            p = mod(m,N/2) + mod(n,N/2);
            t = mod(m,N/4) + mod(n,N/4);
            a = (-1)^t;
            b = (-1j)^p;
            c = alpha(mod(n*m,N/4)+1);
            A00(m,n) = a*b*c;
            A00(n,m) = A00(m,n);
```

```
            A00(m+N/4,n) = ((-1j)^n)*A00(m,n);
            A00(m,n+N/4) = ((-1j)^m)*A00(m,n);
            A00(m+N/4,n+N/4) = ((-1j)^(m+n))*A00(m,n);
        end
    end
    A01 = Change_even_row(A00);
    A10 = Change_even_col(A00);
    A11 = Change_even_row(Change_even_col(A00));
    F = [A00,A01;A10,A11];

    err = norm(W-F/sqrt(N),'fro')/N;
    if err<min_err
        min_err = err;
        esti_W = F;
        min_alpha = alpha;
    end

end

disp(min_err);
disp(min_alpha);

end
```

附录 3

基于二次规划 SQP 的第二问求解代码

```
%Initial
N = 16;
W = DFT_matrix(N);
W0 = randi(N,N);

%Optimization
A = [];
b = [];
Aeq = [];
beq = [];
lb = [];
ub = [];
options = optimoptions('fmincon','Display','iter','Algorithm','sqp');
nonlcon = @constraints;
[esti_W, rmse] = fmincon(@objective_function,W0,A,b,Aeq,beq,lb,ub,nonlcon,op-
tions);
disp('  esti_W:');
disp(esti_W);
disp('  rmse=');
disp(rmse);

function rmse = objective_function(X)
N = size(X,1);
W = DFT_matrix(N);
rmse = norm(W-X/sqrt(N),'fro')/N;
end

function [c,ceq] = constraints(X)

real_part = real(X);
imag_part = imag(X);
```

```
real_min = -4;
real_max = 4;
imag_min = -4;
imag_max = 4;
c = [
    real_part-real_min;
    real_max-real_part;
    imag_part-imag_min;
    imag_max-imag_part];

ceq = [real_part - round(real_part);
       imag_part - round(imag_part)];
end
```

附录 4

基于遗传算法的第二问求解代码

```
%主程序
clc;clear;
error = [];
k = 0:0.01:1;
for k_beta = 0:0.01:1
    error = [error GA(k_beta)];
end
plot(k,error);
grid on

function error=GA(k_beta)
%% GA 主程序
% clear
% close all
k_beta = 0.088;
popsize = 500; % 群体大小
pc = 0.6; %交叉概率
pm = 0.1; %变异概率
G = 1000 ; %迭代次数

%%参数设置
N=32;     %N 维 DFT 矩阵
F_esti = cell(1,popsize);   %随机产生初始群体
x = cell(1,popsize);
for i = 1:popsize
    real_pow = round( rand(N,N)*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand(N,N)*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    for k=1:N
        for h=1:N
            if rand>0.5
                b(k,h) = conj(b(k,h));
            end
            if rand>0.5
                b(k,h) = -b(k,h);
            end
        end
```

```
    end
    F_esti{i} = b;
end
fx = calobjvalue( F_esti ,N ,popsize, k_beta) ; % 计算初代目标函数值
[y(1),l] = min(fx);
x{1} = F_esti{1};

for i=2 : G
    fx = calobjvalue( F_esti ,N ,popsize, k_beta) ; % 计算上一代目标函数值
    fitvalue = calfitvalue(fx) ;  % 适应度映射
    newF_esti = copyx(F_esti,fitvalue,popsize); %复制
    newF_esti = crossover(newF_esti, pc, popsize ,N); %交叉
    newF_esti = mutation(newF_esti,pm, popsize, N); %变异
    % 这时的 newpop 是经过复制交叉变异产生的新一代群体
    new_fx = calobjvalue(newF_esti ,N ,popsize, k_beta) ; %计算新解目标函数
    new_fitvalue = calfitvalue(new_fx); %计算新群体中每个个体的适应度
    index = find(new_fitvalue > fitvalue) ;

    for j = 1:length(index)
        F_esti{index(j)} = newF_esti{index(j)}; % 更新得到最新解
    end
    fx = calobjvalue( F_esti ,N ,popsize, k_beta)  ; %计算结果
     % 找出更新后的个体最优函数值
    [bestindividual,bestindex] = min( fx ) ;
    y(i) = bestindividual; % 记录每一代的最优函数值
    x{i} = F_esti(bestindex) ; %十进制解
    plot(y);
    title('适应度进化曲线');
    grid on
    i = i + 1 ;
end
error = y(1,end);

%% 复制操作
function newx = copyx(F_esti, fitvalue,popsize )
% 按照 PPT 的轮盘赌策略对个体复制
    newx = F_esti; %只是起到申请一个 size 为 pop 大小空间的作用，newx 之后要更新的
    i = 1;  j = 1;
    p = fitvalue / sum(fitvalue) ;
    Cs = cumsum(p) ;
    R = sort(rand(popsize,1)) ; %每个个体的复制概率
    while j <= popsize
        if R(j) < Cs(i)
            newx{j} = F_esti{i} ;
            j = j + 1;
        else
            i = i + 1;
        end
    end
end

%% 交叉操作
function newx = crossover(F_esti, pc, popsize ,N)
% 12 34 56 交叉方式，随机选择交叉位点
% 注意个体数为奇数偶数的区别
i = 2 ;
```

```
newx = F_esti ; %申请空间
while i + 2 <= popsize
    %将第 i 与 第 i -1 进行随机位点交叉
    if rand < pc
        x1 = F_esti{i-1};
        x2 = F_esti{i} ;
        r = randperm( N , 2 ) ; %返回范围内两个整数
        r1 = min(r); r2 =max(r) ; % 交叉复制的位点
        newx{i-1} = [x1(:,1:r1-1),x2(:,r1:r2) , x1(:,r2+1: end)];
        newx{i} = [x2(:, 1 : r1-1),x1(:,r1:r2) , x2(:,r2+1: end)];
%         newx(i-1,:) = [x1( 1 : r1-1),x2(r1:r2) , x1(r2+1: end)];
%         newx(i , : ) = [x2( 1 : r1-1),x1(r1:r2) , x2(r2+1: end)];
    end
    i = i + 2 ; %更新 i
end
end

%% 变异
function newx = mutation(F_esti,pm, popsize,N)
i = 1 ;
while i <= popsize
    if rand < pm
        r = randperm( N , 1 ) ;
%         p = round( rand(1,N)*8 ) - 4  ;
%         temp = 2.^p;
        F_esti{i}(:,r) = 1./F_esti{i}(:,r);
    end
    i = i + 1;
end

newx = F_esti; %将变异后的结果返回。

end

%% 计算适应度
function fitvalue = calfitvalue(fx)
    fitvalue = -fx ; %找适应度的最大值，及误差函数的最小值
end

%% 目标函数
function fx = calobjvalue( F_esti ,N ,popsize, k_beta)
    FN = dftmtx(N)/sqrt(N); %生成标准的 FFT 矩阵
%     F = k_beta.*F_esti;
    err = [];
    for i = 1:popsize   %计算每个群体的误差
        err = [err norm(k_beta.*F_esti{i}/sqrt(N) - FN,'fro')/N];
    end
    fx = err;
end
    end
```

附录 5

基于优化搜索的第三问求解代码

```
N = 32;
q = 3;
```

```
min_err = inf;
P = [0,-1,1,-2,2,-4,4];
P = [P,0,0,0,0];
F = DFT_matrix(N);
[AA,D] = mtx_CTdecom(F);
m = log2(N);
index_A = cell(1,m);
round_A = cell(1,m);
for i = 1:m
    Index = find_nonesimple(AA{i},N);
    index_A{i} = logical(Index);
end

for i = 1:1000

    for j = 1:m
        numSelect = 2;
        randomIndex = randperm(length(P),numSelect);
        randomElements = P(randomIndex);
        value = randomElements(1) + 1j*randomElements(2);
        AA{j}(index_A{j}) = value;
    end

    esti_F = D;
    for k = 1:m
        esti_F = AA{k}*esti_F;
    end
    err = compute_err(F,esti_F);

    if err<min_err
        min_err = err;
        round_A = AA;
        est_W = esti_F;
    end
end

complexity = q * complexity_cal(round_A,N);
fprintf('N = %d, error = %f, complexity = %d\n',N,min_err,complexity);



function err = compute_err(F,esti_F)
N = size(F,1);
err = norm(F-esti_F/sqrt(N))/N;
end
```

附录 6

基于遗传算法的第三问求解代码

```
%主程序
clc;
% clear;
error = [];
t = 0:0.1:2;
for k_beta = 0:0.1:2
    error = [error GA(k_beta,A,P)];
end
plot(t,error);
```

```
grid on

function error=GA(k_beta,A,P)
%% GA 主程序
% clear
% close all

k_beta = 0.9;
popsize = 500; % 群体大小
pc = 0.6; %交叉概率
pm = 0.1; %变异概率
G = 100 ; %迭代次数

%%参数设置
N=8;     %N 维 DFT 矩阵
F_esti = cell(1,popsize);   %随机产生初始群体
x = cell(1,popsize);
for i = 1:popsize
    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(2,6)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(6,6)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(4,8)=b;

    real_pow = round( rand*4 ) - 2  ;
```

```
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(8,8)=b;
    F_esti{i} = A{3};
end
fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P) ; % 计算初代目标函数值
[y(1),l] = min(fx);
x{1} = F_esti{1};

% for k_beta = 0:0.1:1
for i=2 : G
    fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P) ; % 计算上一代目标函数值
    fitvalue = calfitvalue(fx) ;  % 适应度映射
    newF_esti = copyx(F_esti,fitvalue,popsize); %复制
    newF_esti = crossover(newF_esti, pc, popsize ,N); %交叉
    newF_esti = mutation(newF_esti,pm, popsize, N); %变异
    % 这时的 newpop 是经过复制交叉变异产生的新一代群体
    new_fx = calobjvalue(newF_esti ,N ,popsize, k_beta,A,P) ; %计算新解目标函数
    new_fitvalue = calfitvalue(new_fx); %计算新群体中每个个体的适应度
    index = find(new_fitvalue > fitvalue) ;

    for j = 1:length(index)
        F_esti{index(j)} = newF_esti{index(j)}; % 更新得到最新解
    end
    fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P)  ; %计算结果
     % 找出更新后的个体最优函数值
    [bestindividual,bestindex] = min( fx ) ;
    y(i) = bestindividual; % 记录每一代的最优函数值
    x{i} = F_esti(bestindex) ; %十进制解
    plot(y);
    title('适应度进化曲线');
    axis([0 200 0.06 0.15])
    grid on
    i = i + 1 ;
end
error = y(1,end);

%% 复制操作

function newx = copyx(F_esti, fitvalue,popsize )
% 按照 PPT 的轮盘赌策略对个体复制
    newx = F_esti; %只是起到申请一个 size 为 pop 大小空间的作用，newx 之后要更新的
    i = 1;  j = 1;
    p = fitvalue / sum(fitvalue) ;
    Cs = cumsum(p) ;
    R = sort(rand(popsize,1)) ; %每个个体的复制概率
    while j <= popsize
        if R(j) < Cs(i)
            newx{j} = F_esti{i} ;
```

```
            j = j + 1;
        else
            i = i + 1;
        end
    end
end

%% 交叉操作
function newx = crossover(F_esti, pc, popsize ,N)
% 12 34 56 交叉方式，随机选择交叉位点
% 注意个体数为奇数偶数的区别
i = 2 ;
newx = F_esti ; %申请空间
while i + 2 <= popsize
    %将第 i 与 第 i -1 进行随机位点交叉
    if rand < pc
        x1 = F_esti{i-1};
        x2 = F_esti{i} ;
        r = randperm( N , 2 ) ; %返回范围内两个整数
        r1 = min(r); r2 =max(r) ; % 交叉复制的位点
        newx{i-1} = [x1(:,1:r1-1),x2(:,r1:r2) , x1(:,r2+1: end)];
        newx{i} = [x2(:, 1 : r1-1),x1(:,r1:r2) , x2(:,r2+1: end)];
%         newx(i-1,:) = [x1( 1 : r1-1),x2(r1:r2) , x1(r2+1: end)];
%         newx(i , : ) = [x2( 1 : r1-1),x1(r1:r2) , x2(r2+1: end)];
    end
    i = i + 2 ; %更新 i
end
end

%% 变异
function newx = mutation(F_esti,pm, popsize,N)
i = 1 ;
while i <= popsize
    if rand < pm
        r = randperm( N , 1 ) ;
%         p = round( rand(1,N)*8 ) - 4  ;
%         temp = 2.^p;
        F_esti{i}(:,r) = 1./F_esti{i}(:,r);
    end
    i = i + 1;
end

newx = F_esti; %将变异后的结果返回。

end

%% 计算适应度
function fitvalue = calfitvalue(fx)
    fitvalue = -fx ; %找适应度的最大值，及误差函数的最小值
end

%% 目标函数
function fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P)
    FN = dftmtx(N)/sqrt(N); %生成标准的 FFT 矩阵
%     F = k_beta.*F_esti;
    err = [];
    for i = 1:popsize   %计算每个群体的误差
```

```
        err = [err norm(k_beta.*F_esti{i}*A{2}*A{1}*P/sqrt(N) - FN,'fro')/N];
    end
    fx = err;
end
end


%主程序
clc;
% clear;
error = [];
t = 0:0.1:2;
for k_beta = 0:0.1:2
    error = [error GA(k_beta,A,P)];
end
plot(t,error);
grid on


function error=GA(k_beta,A,P)
%% GA 主程序
% clear
% close all

k_beta = 1.1;
popsize = 1000; % 群体大小
pc = 0.6; %交叉概率
pm = 0.1; %变异概率
G = 200 ; %迭代次数

%%参数设置
N=16;     %N 维 DFT 矩阵
F_esti = cell(1,popsize);   %随机产生初始群体
x = cell(1,popsize);
for i = 1:popsize
    %
    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(2,10)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
```

```
        b = -b;
    end
    A{4}(10,10)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(3,11)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(11,11)=b;
    %
    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(4,12)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(12,12)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
```

```
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(6,14)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(14,14)=b;
    %
    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(7,15)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(15,15)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
```

```
    end
    A{4}(8,16)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{4}(16,16)=b;
    F_esti{i} = A{4};
end
fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P) ; % 计算初代目标函数值
[y(1),l] = min(fx);
x{1} = F_esti{1};

% for k_beta = 0:0.1:1
for i=2 : G
    fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P) ; % 计算上一代目标函数值
    fitvalue = calfitvalue(fx) ;  % 适应度映射
    newF_esti = copyx(F_esti,fitvalue,popsize); %复制
    newF_esti = crossover(newF_esti, pc, popsize ,N); %交叉
    newF_esti = mutation(newF_esti,pm, popsize, N); %变异
    % 这时的 newpop 是经过复制交叉变异产生的新一代群体
    new_fx = calobjvalue(newF_esti ,N ,popsize, k_beta,A,P) ; %计算新解目标函数
    new_fitvalue = calfitvalue(new_fx); %计算新群体中每个个体的适应度
    index = find(new_fitvalue > fitvalue) ;

    for j = 1:length(index)
        F_esti{index(j)} = newF_esti{index(j)}; % 更新得到最新解
    end
    fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P)  ; %计算结果
     % 找出更新后的个体最优函数值
    [bestindividual,bestindex] = min( fx ) ;
    y(i) = bestindividual; % 记录每一代的最优函数值
    x{i} = F_esti(bestindex) ; %十进制解
    plot(y);
    title('适应度进化曲线');
    axis([0 200 0.06 0.15])
    grid on
    i = i + 1 ;
end
error = y(1,end);

%% 复制操作

function newx = copyx(F_esti, fitvalue,popsize )
% 按照 PPT 的轮盘赌策略对个体复制
    newx = F_esti; %只是起到申请一个 size 为 pop 大小空间的作用，newx 之后要更新的
    i = 1;  j = 1;
    p = fitvalue / sum(fitvalue) ;
    Cs = cumsum(p) ;
```

```
    R = sort(rand(popsize,1)) ; %每个个体的复制概率
    while j <= popsize
        if R(j) < Cs(i)
            newx{j} = F_esti{i} ;
            j = j + 1;
        else
            i = i + 1;
        end
    end
end

%% 交叉操作
function newx = crossover(F_esti, pc, popsize ,N)
% 12 34 56 交叉方式，随机选择交叉位点
% 注意个体数为奇数偶数的区别
i = 2 ;
newx = F_esti ; %申请空间
while i + 2 <= popsize
    %将第 i 与 第 i -1 进行随机位点交叉
    if rand < pc
        x1 = F_esti{i-1};
        x2 = F_esti{i} ;
        r = randperm( N , 2 ) ; %返回范围内两个整数
        r1 = min(r); r2 =max(r) ; % 交叉复制的位点
        newx{i-1} = [x1(:,1:r1-1),x2(:,r1:r2) , x1(:,r2+1: end)];
        newx{i} = [x2(:, 1 : r1-1),x1(:,r1:r2) , x2(:,r2+1: end)];
%         newx(i-1,:) = [x1( 1 : r1-1),x2(r1:r2) , x1(r2+1: end)];
%         newx(i , : ) = [x2( 1 : r1-1),x1(r1:r2) , x2(r2+1: end)];
    end
    i = i + 2 ; %更新 i
end
end

%% 变异
function newx = mutation(F_esti,pm, popsize,N)
i = 1 ;
while i <= popsize
    if rand < pm
        r = randperm( N , 1 ) ;
%         p = round( rand(1,N)*8 ) - 4  ;
%         temp = 2.^p;
        F_esti{i}(:,r) = 1./F_esti{i}(:,r);
    end
    i = i + 1;
end

newx = F_esti; %将变异后的结果返回。

end

%% 计算适应度
function fitvalue = calfitvalue(fx)
    fitvalue = -fx ; %找适应度的最大值，及误差函数的最小值
end

%% 目标函数
function fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P)
```

```
    FN = dftmtx(N)/sqrt(N); %生成标准的 FFT 矩阵
%     F = k_beta.*F_esti;
    err = [];
    for i = 1:popsize   %计算每个群体的误差
        err = [err norm(k_beta.*F_esti{i}*A{3}*A{2}*A{1}*P/sqrt(N) -
FN,'fro')/N];
    end
    fx = err;
end

end

%主程序
% clc;
% % clear;
% error = [];
% t = 0:0.1:5;
% for k_beta = 0:0.1:5
%     error = [error GA(k_beta,A,P,est_A4)];
% end
% plot(t,error);
% grid on
%
%
% function error=GA(k_beta,A,P,est_A4)
%% GA 主程序
% clear
% close all

k_beta = 0.4;
popsize = 1000; % 群体大小
pc = 0.6; %交叉概率
pm = 0.1; %变异概率
G = 500 ; %迭代次数

%%参数设置
N=16;     %N 维 DFT 矩阵
F_esti = cell(1,popsize);   %随机产生初始群体
x = cell(1,popsize);
for i = 1:popsize
    %
    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(2,6)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
```

```
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(6,6)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(4,8)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(8,8)=b;
    %
    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(10,14)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
```

```
    A{3}(14,14)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(12,16)=b;

    real_pow = round( rand*4 ) - 2  ;
    real = 2.^real_pow;
    img_pow = round( rand*4 ) - 2  ;
    img = 2.^img_pow;
    b = complex(real , img);
    if rand>0.5
        b = conj(b);
    end
    if rand>0.5
        b = -b;
    end
    A{3}(16,16)=b;

    F_esti{i} = A{3};
end
fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P,est_A4) ; % 计算初代目标函数值
[y(1),l] = min(fx);
x{1} = F_esti{1};

% for k_beta = 0:0.1:1
for i=2 : G
    fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P,est_A4) ; % 计算上一代目标函
数值
    fitvalue = calfitvalue(fx) ;  % 适应度映射
    newF_esti = copyx(F_esti,fitvalue,popsize); %复制
    newF_esti = crossover(newF_esti, pc, popsize ,N); %交叉
    newF_esti = mutation(newF_esti,pm, popsize, N); %变异
    % 这时的 newpop 是经过复制交叉变异产生的新一代群体
    new_fx = calobjvalue(newF_esti ,N ,popsize, k_beta,A,P,est_A4) ; %计算新解目
标函数
    new_fitvalue = calfitvalue(new_fx); %计算新群体中每个个体的适应度
    index = find(new_fitvalue > fitvalue) ;

    for j = 1:length(index)
        F_esti{index(j)} = newF_esti{index(j)}; % 更新得到最新解
    end
    fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P,est_A4)  ; %计算结果
     % 找出更新后的个体最优函数值
    [bestindividual,bestindex] = min( fx ) ;
    y(i) = bestindividual; % 记录每一代的最优函数值
    x{i} = F_esti(bestindex) ; %十进制解
    plot(y);
```

```
    title('适应度进化曲线');
%     axis([0 200 0.06 0.15])
    grid on
    i = i + 1 ;
end
error = y(1,end);

%% 复制操作

function newx = copyx(F_esti, fitvalue,popsize )
% 按照 PPT 的轮盘赌策略对个体复制
    newx = F_esti; %只是起到申请一个 size 为 pop 大小空间的作用，newx 之后要更新的
    i = 1;  j = 1;
    p = fitvalue / sum(fitvalue) ;
    Cs = cumsum(p) ;
    R = sort(rand(popsize,1)) ; %每个个体的复制概率
    while j <= popsize
        if R(j) < Cs(i)
            newx{j} = F_esti{i} ;
            j = j + 1;
        else
            i = i + 1;
        end
    end
end

%% 交叉操作
function newx = crossover(F_esti, pc, popsize ,N)
% 12 34 56 交叉方式，随机选择交叉位点
% 注意个体数为奇数偶数的区别
i = 2 ;
newx = F_esti ; %申请空间
while i + 2 <= popsize
    %将第 i 与 第 i -1 进行随机位点交叉
    if rand < pc
        x1 = F_esti{i-1};
        x2 = F_esti{i} ;
        r = randperm( N , 2 ) ; %返回范围内两个整数
        r1 = min(r); r2 =max(r) ; % 交叉复制的位点
        newx{i-1} = [x1(:,1:r1-1),x2(:,r1:r2) , x1(:,r2+1: end)];
        newx{i} = [x2(:, 1 : r1-1),x1(:,r1:r2) , x2(:,r2+1: end)];
%         newx(i-1,:) = [x1( 1 : r1-1),x2(r1:r2) , x1(r2+1: end)];
%         newx(i , : ) = [x2( 1 : r1-1),x1(r1:r2) , x2(r2+1: end)];
    end
    i = i + 2 ; %更新 i
end
end

%% 变异
function newx = mutation(F_esti,pm, popsize,N)
i = 1 ;
while i <= popsize
    if rand < pm
        r = randperm( N , 1 ) ;
%         p = round( rand(1,N)*8 ) - 4  ;
%         temp = 2.^p;
        F_esti{i}(:,r) = 1./F_esti{i}(:,r);
```

```
    end
    i = i + 1;
end

newx = F_esti; %将变异后的结果返回。

end

%% 计算适应度
function fitvalue = calfitvalue(fx)
    fitvalue = -fx ; %找适应度的最大值，及误差函数的最小值
end

%% 目标函数
function fx = calobjvalue( F_esti ,N ,popsize, k_beta,A,P,est_A4)
    FN = dftmtx(N)/sqrt(N); %生成标准的 FFT 矩阵
%     F = k_beta.*F_esti;
    err = [];
    for i = 1:popsize   %计算每个群体的误差
        err = [err norm(k_beta.*est_A4*F_esti{i}*A{2}*A{1}*P/sqrt(N) -
FN,'fro')/N];
    end
    fx = err;
end

end
```

附录 7

基于“混合积分解—降维寻优”算法 1 的第四问求解代码

```
clc;clear;

%% 生成 Kronecker 积矩阵 FN
FN1 = dftmtx(4)/sqrt(4);
FN2 = dftmtx(8)/sqrt(8);
FN=kron(FN1,FN2);%FN1 和 FN2 的顺序不能换

%% 步骤 1：CT 算法分解 FN1 和 FN2
% FN1=AN1{2}*AN1{1}*P1/sqrt(4)
[P1,AN1,Comp1,Rms1] = Q1_CT_N(4);
% FN2=AN2{3}*AN2{2}*AN2{1}*P2/sqrt(8)
[P2,AN2,Comp2,Rms2] = Q1_CT_N(8);
%% 最终分解的矩阵存储在数组 A 中
A=cell(1,6);
A{1}=kron(P1,P2);
%% 待分解的两个乘积项
CN_1=kron(AN1{1},AN2{1});%对应论文中的 kron(B1,C1)
CN_2=kron(AN1{2},AN2{2});%对应论文中的 kron(B12,C2)

%% CN1 分成 2 个对角的 C16 块求解
pick_C16=CN_1(1:16,1:16);%这是 CN1 的两个对角块，优化了 C16 就能优化 CN1
pick_C8=CN_1(1:8,1:8);%C16 是由 4 个 C8 拼起来的

%求出 pick_C16 对角矩阵 C16_diag
%第一次逆变换
```

```
P_blktrans_inv1=eye(16);
for i=1:8
P_blktrans_inv1(i+8,i)=-1;%第 i 行乘以-1 加到第 i+8 行上
end
%第二次逆变换，这个时候需要 beta 的帮助
P_blktrans_inv2=eye(16);
for i=1:8
P_blktrans_inv2(i,i+8)=1/2; %第 i+8 行乘以 1/2 加到第 i 行上
end
%对角阵
C16_diag=P_blktrans_inv2*P_blktrans_inv1*pick_C16

%求出对角矩阵 C16_diag 的变换矩阵
%第一次行变换
P_blktrans_1=eye(16);
for i=1:8
P_blktrans_1(i+8,i)=1;
end
%第二次行变换
P_blktrans_2=eye(16);
for i=1:8
P_blktrans_2(i,i+8)=-1/2;
end
%这行代码用于解释稀疏矩阵的相乘顺序
C16_combine=P_blktrans_1*P_blktrans_2*C16_diag;

%% 把 C16 拼回 CN1
CN1_diag=blkdiag(C16_diag,C16_diag);
P_trans_1=blkdiag(P_blktrans_1,P_blktrans_1);
P_trans_2=blkdiag(P_blktrans_2,P_blktrans_2);

%% 得到 CN1 行变换矩阵
P1_trans=P_trans_1*P_trans_2;

%% CN2 直接对 32 阶的大块做行变换
%第一次行变换，第 1 行乘以-1 加在第 17 行,第 2 行乘以-1 加在第 18 行
P2_trans_1=eye(32);
for i=1:8
    P2_trans_1(i+16,i)=-1;
end
for i=9:16
    P2_trans_1(i,i+16)=-1;
end
%第二次行变换，
P2_trans_2=eye(32);
for i=1:8
    P2_trans_2(i,i+16)=1/2;
end
for i=9:16
    P2_trans_2(i+16,i)=1/2;
end
CN2_diag=P2_trans_2*P2_trans_1*CN_2;
P2_trans=inv(P2_trans_2*P2_trans_1);

%% 综上，分解后的矩阵分别为
A{5}=P2_trans;
A{4}=CN2_diag;
```

```
A{3}=P1_trans;
A{2}=CN1_diag;

%%寻优示例：A3_opti 是从问题三中导出的一个最优解
A3_11=eye(4);
A3_12=diag([1,-0.25-1j,-1j,1+0.25*1j]);
A3_21=A3_11;
A3_22=diag([-1,1-j,1j,-0.5-1j]);
A3_opti=[A3_11,A3_12;A3_21,A3_22];
A{6}=kron(eye(4),A3_opti);

%%test
product1=kron(eye(4),A3_opti);
product2=P2_trans*CN2_diag;
product3=P1_trans*CN1_diag;
product4=kron(P1,P2);

F_esti=1;
for i =1:6
    F_esti=A{i}*F_esti/sqrt(32);
    err1 = norm(F_esti-FN,'fro')/32
end
```

附录 8

基于“混合积分解—降维寻优”算法 2 的第四问求解代码

```
F4 = DFT_matrix(4);
F8 = DFT_matrix(8);
[AA4,D4] = mtx_CTdecom(F4);
[AA8,D8] = mtx_CTdecom(F8);
F = kron(F4,F8);
B = [1,1;1,-1];
I2 = eye(2);
I4 = eye(4);
%accurate
A41 = kron(B,I2);
A81 = kron(B,I4);
%esti
R42 = [1,0;0,-1j];
A42 = kron(B,R42);
R82 = kron(B,R42);
A82 = kron(R82,I2);
R83 = [1,0,0,0;0,0.707-0.707j,0,0;
       0,0,-1j,0;0,0,0,-0.707-0.707j];
A83 = kron(B,R83);

%decom

C1 = kron(I4,A83);
inter1 = kron(B*B,R42);
inter2 = kron(inter1,B*B);
inter3 = kron(inter2,R42);
C2 = kron(inter3,I2);
C3 = kron(D4,D8);

esti_F = C1*C2*C3;
err1 = norm(F-esti_F/sqrt(32))/32;
```

```
%round

round_C = cell(1,3);
CC = cell(1,3);
CC{1} = C1;CC{2} = C2;CC{3} = C3;
index_C = logical(find_nonesimple(C1,32));

min_err = inf;
P = [0,-1,1,-2,2,-4,4];
P = [P,0,0,0,0];
q = 3;
for i = 1:1000

    numSelect = 2;
    randomIndex = randperm(length(P),numSelect);
    randomElements = P(randomIndex);
    value = randomElements(1) + 1j*randomElements(2);
    CC{1}(index_C) = value;

    esti_F = 1;
    for k = 1:m
        esti_F = CC{k}*esti_F;
    end
    err = compute_err(F,esti_F);

    if err<min_err
        min_err = err;
        round_C = CC;
        est_W = esti_F;
    end
end

Opti_C1 = round_C{1};
Opti_C2 = round_C{2};
Opti_C3 = round_C{3};
```

附录 9

基于优化搜索的第五问求解代码

```
N = 8;
q = 4;
min_err = inf;
P = [0,-1,1,-2,2,-4,4,-8,8];
P = [P,0,0,0,0];
F = DFT_matrix(N);
[AA,D] = mtx_CTdecom(F);
m = log2(N);
index_A = cell(1,m);
round_A = cell(1,m);
for i = 1:m
    Index = find_nonesimple(AA{i},N);
    index_A{i} = Index;
end

for i = 1:100000

    for j = 1:m
        numSelect = 4;
```

```
        randomIndex = randperm(length(P),numSelect);
        randomElements = P(randomIndex);
        value1 = randomElements(1) + 1j*randomElements(2);
        value2 = randomElements(3) + 1j*randomElements(4);
        pos1 = zeros(N,N);
        pos1(1:N/2,:) = index_A{j}(1:N/2,:);
        pos1 = logical(pos1);
        pos2 = zeros(N,N);
        pos2(N/2+1:end,:) = index_A{j}(N/2+1:end,:);
        pos2 = logical(pos2);
        AA{j}(pos1) = value1;
        AA{j}(pos2) = value2;
    end

    esti_F = D;
    for k = 1:m
        esti_F = AA{k}*esti_F;
    end
    err = compute_err(F,esti_F);

    if err<min_err
        min_err = err;
        round_A = AA;
        est_W = esti_F;
    end
end

complexity = q * complexity_cal(round_A,N);
fprintf('N = %d, error = %f, complexity = %d\n',N,min_err,complexity);
```