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	中山大学	
参赛队号	23105580005	
队员姓名	1.	罗思远
	2.	柳文涛
	3.	欧瀚祺

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：基于修正精确分解的矩阵连乘拟合模型

摘　要：

本文基于矩阵连乘拟合的思路，将离散傅里叶变换（Discrete Fourier Transform, DFT）中使用的 DFT 矩阵通过近似分解得到稀疏有限取值的矩阵连乘的形式，以期降低目前 DFT 的计算复杂度。

问题一中，我们针对问题约束和优化目标，建立了**基于 FFT 思路的 DFT 矩阵分解模型**，在满足约束 1 的条件下实现了对N 阶 DFT 矩阵的**精确分解**。我们运用 **Cooley-Tukey 快速傅里叶算法** [1]，基于分治思想，将 N 阶 DFT 矩阵 F_N 递归地拆分为两个 $N/2$ 阶的 DFT 矩阵，最后分解为连乘的满足约束 1 的稀疏矩阵，并详细地给出分解结果关于 N 的通项形式，得出最优的矩阵缩放因子为$\beta = 1/\sqrt{N}$. 在这种分解形式下，DFT 矩阵被精确分解，因此目标函数的最优值即 $\min_{A,\beta} \mathrm{RMSE}(\mathcal{A}, \beta) = 0$. 最后，我们得出 N 阶 DFT 矩阵 F_N 的**硬件复杂度的通项公式为 $C = 16 \times [(\log_2 N - 3)N + 4]$**.

问题二中，我们先对目标函数中缩放因子的设置进行了分析，发现**原先的缩放因子设置存在对目标函数的过度奖励和过度惩罚**，因此我们对缩放因子的设置进行了修改。由于分解矩阵的约束条件发生了改变，问题一中的精确分解得到的矩阵不满足约束 2，但同时稀疏性的约束被放宽。为此，我们**基于连乘后矩阵修正**的思想，以将近似矩阵的稀疏性转化为精度为目的，提出了 N 阶 DFT 矩阵 F_N 的近似分解算法。我们将 DFT 精确分解中的前 $K-3$ 个不满足约束 2 的分解矩阵连乘为一个矩阵 A_0 进行修正，使得在满足约束 2 的条件下最小化目标函数。为此，我们通过建立 Frobenius 范数下对于约束 2 的**矩阵插值逼近算法**，并基于对缩放因子的取值分析，提出了**缩放因子邻域搜索算法**以找到最佳缩放因子 β 以及 A_0 的最佳逼近 B，最后将分解矩阵相乘计算目标函数值以及硬件复杂度。求解结果如下表0.1所示。

问题三中，模型应同时满足约束 1 和约束 2，优化空间减小。我们基于 DFT 矩阵的精确分解，在分解矩阵已经满足约束 1 的情况下，对所有分解矩阵进行**连乘前矩阵修正**，使得修正后每一个分解矩阵都满足约束 2. 然后将修正后的连乘矩阵关于总体缩放因子进行优化，**求得最优缩放因子β^* 的表达式**，最终得到 F_N 在 Frobenius 范数下的近似，以及相应的最小误差和硬件复杂度。求解结果如下表0.1所示。

表 0.1　问题二和三结果

	问题二			**问题三**		
矩阵维数 N	目标函数值	复乘次数 L	硬件复杂度 C	目标函数值	复乘次数 L	硬件复杂度 C
2	0	0	0	0	0	0
4	0	0	0	0	0	0
8	0.0479	12	36	0.0479	12	36
16	0.0380	64	192	0.0405	52	156
32	0.0251	272	816	0.0353	184	552

问题四中，我们需要对 DFT 矩阵之间的 Kronecker 积（张量积）作矩阵连乘近似。注意到问题中的 $F_4 \otimes F_8$ 并非 DFT 矩阵，因此我们基于数学推导对 $F_4 \otimes F_8$ 进行了初步的分解，并通过**证明一组矩阵之积与单位阵的张量积满足矩阵乘法和张量积的分配律**，成功将原问题转化为 F_4 和 F_8 在约束 1 和 2 控制下的近似分解问题，这实际上在问题三已经得到解答。最后我们得到 $F_4 \otimes F_8$ 的近似分解的相关结果如下：

表 0.2　问题四结果

目标函数值	缩放因子 β	复乘次数 L	硬件复杂度 C
0.0240	0.0936	48	144

问题五中，需要我们在问题三的基础上增加一个精度的约束，目标函数是硬件复杂度 C，并同时对分解矩阵 A，实值缩放因子 β 和矩阵元素的实部和虚部取值范围 $\mathcal{P}$ 进行优化。我们经过分析认为对元素取值的减少相较于降低乘法器个数对复杂度的影响更加显著。基于这种想法，我们提出了针对这种精度约束的递进寻优模型。首先我们引入**最小范围 q^* 的搜索方法**，从最极端的情况 $P = \{1, 2, \cdots 2^{q-1}, q = 16\}$ 开始寻找最优的矩阵连乘拟合，当满足精度要求时逐步收紧元素取值范围，直至无法取得更小。在此基础上，我们提出**简单复数替换的贪婪算法**，将分解矩阵中的复杂元素用 $0, \pm 1, \pm j, \pm 1 \pm j$ 进行替换，复杂元素的选取原则是保证每步替换对精度影响最小。重复这种替换直至无法再满足精度要求，最终得到的结果认为是满足约束 1、约束 2 和精度限制下硬件复杂度最优的方案。

关键词：离散傅里叶变换　快速傅里叶变换　矩阵连乘拟合　矩阵修正近似　插值逼近　邻域搜索算法　张量积　贪婪算法

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。在芯片设计中，期望通过降低 DFT 算法的复杂度和限制数据取值范围来降低其硬件复杂度。由于 DFT 矩阵的特殊结构，在实际产品中，可以用 FFT 算法快速实现 DFT，从而降低 DFT 的计算复杂度。然而，随着无线通信技术的演进，这种传统的 FFT 思路在实现时的硬件开销也越来越大。

为了节省成本，现在提出了新的思路：矩阵连乘拟合思路。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的连乘形式。因为通过这种方法分解后的矩阵均为稀疏的有限取值矩阵，降低了乘法器的个数，所以硬件复杂度得到大幅降低。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。综上，运用矩阵连乘拟合思路，研究 DFT 的低复杂度计算方案，以实现对传统的 FFT 思路进行替代，具有十分重要的价值和意义。

1.2 问题提出

本课题在运用矩阵连乘拟合思路来降低 DFT 算法的硬件复杂度时，主要任务是在不同的约束条件下对 $\mathcal{A}$ 和 β 进行优化，并计算最小误差和方案的硬件复杂度 C. 其中，对 $\mathcal{A}$ 进行优化以减少硬件的复杂度 C，实现的方法有两个：

1. 控制 DFT 矩阵 F_N 近似分解后的矩阵 A_k 为稀疏矩阵，以此来减少乘法器的个数。
2. 限制 A_k 中元素实部和虚部的取值范围。

另一方面，在对 $\mathcal{A}$ 和 β 进行优化时，要使得拟合的精度足够高，也就是要最小化 DFT 矩阵 F_N 与其连乘拟合之间的 Frobenius 范数。

问题一： 建立 DFT 矩阵分解为稀疏矩阵连乘拟合的数学模型，通过减少乘法器个数来降低硬件的复杂度 C. 通过优化 $\mathcal{A}$ 和 β 来最小化目标函数，即

$$\min_{\mathcal{A},\beta} \mathrm{RMSE}(\mathcal{A},\beta) = \frac{1}{N}\sqrt{\|F_N - \beta A_1 A_2 \cdots A_K\|_F^2}.$$

要求满足约束 1：限定 $\mathcal{A}$ 中每个矩阵 A_k 的每行至多只有 2 个非零元素。

计算硬件复杂度时可默认 $q = 16$. 建立并求解数学模型，设计稀疏矩阵连乘拟合的分解方案，给出 $\mathcal{A}$ 和 β 的优化结果，并计算最小误差和硬件的复杂度 C.

问题二： 通过限制 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 C. 通过优化 $\mathcal{A}$ 和 β 来最小化目标函数，要求满足约束 2：$A_k[l,m] \in \{x + jy | x, y \in \mathcal{P}\}, \mathcal{P} = \{0, \pm1, \pm2, \cdots, \pm 2^{q-1}\}, q = 3$.

计算硬件复杂度时固定 $q = 3$. 建立并求解数学模型，设计矩阵连乘拟合的分解方案，给出 $\mathcal{A}$ 和 β 的优化结果，并计算最小误差和硬件的复杂度 C.

问题三： 通过减少乘法器个数和限制 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 C. 通过优化 $\mathcal{A}$ 和 β 来最小化目标函数, 要求满足：

约束 1：限定 $\mathcal{A}$ 中每个矩阵 A_k 的每行至多只有 2 个非零元素。

约束 2：$A_k[l,m] \in \{x+jy | x,y \in \mathcal{P}\}, \mathcal{P} = \{0, \pm 1, \pm 2, \cdots, \pm 2^{q-1}\}, q = 3.$

计算硬件复杂度时固定 $q = 3$. 建立并求解数学模型，设计矩阵连乘拟合的分解方案，给出 $\mathcal{A}$ 和 β 的优化结果，并计算最小误差和硬件的复杂度 C.

问题四： 考虑 $F_N = F_{N_1} \otimes F_{N_2}$，其中 F_{N_1}, F_{N_2} 分别是 N_1, N_2 维的 DFT 矩阵，$\otimes$ 表示张量积。当 $N_1 = 4, N_2 = 8$ 时，在同时满足约束 1 和 2 的条件下，对 $F_N = F_{N_1} \otimes F_{N_2}$ 进行矩阵连乘拟合。

计算硬件复杂度时固定 $q = 3$. 建立并求解数学模型，设计矩阵连乘拟合的分解方案，给出 $\mathcal{A}$ 和 β 的优化结果，并计算最小误差和硬件的复杂度 C.

问题五： 在问题三的基础上，要求将精度限制在 0.1 以内，即 RMSE $\leq$ 0.1. 对于 $N = 2^t, t = 1, 2, \cdots$ 的 DFT 矩阵 F_N，在同时满足精度限制、约束 1 和 2 的条件下，对 $\mathcal{A}$ 和 β 进行优化，使方案的硬件复杂度尽可能小。

建立并求解数学模型，设计矩阵连乘拟合的分解方案，给出 $\mathcal{A}$ 和 β 的优化结果，并计算方案的硬件的复杂度 C.

2 模型的假设

本文依据题意进行如下假设：

1. 硬件复杂度仅考虑乘法器的复杂度，硬件复杂度与乘法器个数和每个乘法器的复杂度成正比。
2. 单个乘法器的复杂度简化为仅与复数中实部和虚部的取值范围相关。
3. 乘法器个数定义为复数乘法的次数，且与 $0, \pm 1, \pm j \ \pm 1 \pm j$ 相乘时不计入乘法次数。

3 符号说明

表 3.1　模型参数

参数	含义
F_N	N 阶 DFT 矩阵
ω_N^k	$\exp\left(-\frac{k \cdot j2\pi}{N}\right)$
β	实值矩阵缩放因子
A_k	第 k 个分解矩阵，$k = 1, 2, \cdots, K$
$\mathcal{A}$	分解矩阵组成的集合，即 $A_k \in \mathcal{A}$
q	分解矩阵中元素的取值范围
L	矩阵连乘中复数乘法的次数
C	乘法器的硬件复杂度，$C = q \times L$
$\mathcal{P}$	分解矩阵中元素实部和虚部的取值范围
$M(\mathcal{P}^2)$	满足元素实部和虚部在 P 上的矩阵集合
M_{sparse}	满足每行非零元不超过两个的矩阵集合

4 问题一的模型建立与求解

4.1 问题一分析

问题一通过减少乘法器个数来降低硬件复杂度，优化目标为矩阵 F_N 和 $\beta A_1 A_2 \cdots A_K$ 在 Frobenius 范数意义下尽可能接近，同时还要满足“约束 1：限定 $\mathcal{A}$ 中每个矩阵 A_k 的

每行至多只有 2 个非零元素”。注意到问题一并没有对 A_k 中元素实部和虚部的大小做任何约束，因此对给定已知的 N 维 DFT 矩阵 F_N，总可以设计 K 个满足约束 1 的稀疏矩阵 $\mathcal{A}=\{A_1,A_2,\cdots,A_K\}$，使得它们在矩阵连乘和缩放因子的作用下精确等于 F_N，即

$$\exists\mathcal{A}_0=\{A_1,A_2,\cdots,A_K\},\forall A_k\in\mathcal{A}_0,A_k[l,m]\in\mathbb{C},\quad s.t.\ F_n=\beta A_1A_2\cdots A_K,$$

在精确相等的情形下，矩阵 F_N 和 $\beta A_1A_2\cdots A_K$ 在 Frobenius 范数意义下等于 0，即

$$\text{RMSE}(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\|F_N-\beta A_1A_2\cdots A_K\|_F^2}=0,$$

显然达到最小误差。

因此，在问题一的模型中，需要构造一种对 F_N 的精确分解算法，使得分解矩阵 A_k 在每行至多只有两个非零元素。在此基础上，分析并计算方案的硬件复杂度 C.

4.2 问题一数学模型

4.2.1 Cooley-Tukey 快速傅里叶变换算法

在问题一中需要建立如下的数学规划模型

$$\begin{aligned}&\min_{\mathcal{A},\beta}\ \text{RMSE}(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\|F_N-\beta A_1A_2\cdots A_K\|_F^2}\\&\text{s.t.}\sum_{m=1}(A_k[l,m]\neq0)\leqslant2,k=1,2,\cdots,K,l=1,2,\cdots,N.\end{aligned}$$

其中，

$$(A_k[l,m]\neq0)=\begin{cases}1, & A_k[l,m]\neq0\\0, & A_k[l,m]=0\end{cases}$$

由于问题一没有对分解矩阵的元素进行限制，可以考虑定义在复数域上：$A_k[l,m]\in\mathbb{C}$.

为了表达上的方便，我们考虑矩阵 $\sqrt{N}F_N$ 的分解，由于缩放因子 β 的存在，两个矩阵在优化过程中可以看作是等价的：

$$\tilde{F}_N=\sqrt{N}F_N=\begin{bmatrix}1&1&1&\cdots&1\\1&\omega&\omega^2&\cdots&\omega^{N-1}\\1&\omega^2&\omega^4&\cdots&\omega^{2(N-1)}\\\vdots&\vdots&\vdots&\ddots&\vdots\\1&\omega^{N-1}&\omega^{2(N-1)}&\cdots&\omega^{(N-1)(N-1)}\end{bmatrix}.$$

其中 $\omega=\omega_N=\exp(-2\pi j/N)$，因此 $\sqrt{N}F_N[l,m]=\omega_N^{(l-1)(m-1)}$.

受到 Cooley-Tukey FFT 算法 [X] 的启发，我们将 N 阶的 DFT 矩阵递归的拆分成 $N/2$ 阶的 DFT 矩阵。这种方法又称为 2 基底 (radix-2) 的 FFT 算法。

下面我们以 $\tilde{F}_4$ 为例，简要阐述这种分解方法的原理，并给出对任意 $\tilde{F}_N$ 成立的分解公式。

为了更好的理解，我们首先介绍 $\sqrt{N}F_N$ 的一些性质:

1. $\omega_{2N}^{2k} = \omega_N^k$.
2. $\omega_N^{k+\frac{N}{2}} = -\omega_N^k$.
3. $\omega_N^{Nk} = 1$.

考虑 4 阶 DFT 变换如下

$$\begin{pmatrix} X[0] \\ X[1] \\ X[2] \\ X[3] \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4 & \omega_4^2 & \omega_4^3 \\ 1 & \omega_4^2 & \omega_4^4 & \omega_4^6 \\ 1 & \omega_4^3 & \omega_4^6 & \omega_4^9 \end{bmatrix} \begin{pmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{pmatrix}$$

我们对于序列 $x[n]$ 按奇偶分组如下，同时考虑到上述性质:

$$\begin{aligned} X[k] &= \sum_{n=0}^{\frac{N}{2}-1} x[2n]\omega_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x[2n+1]\omega_N^{(2n+1)k} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[2n]\omega_{\frac{N}{2}}^{nk} + \omega_N^k \sum_{n=0}^{\frac{N}{2}-1} x[2n+1]\omega_{\frac{N}{2}}^{nk} \quad k = 0, 1, \ldots, N-1 \end{aligned}$$

容易观察到，第一项 $\sum_{n=0}^{\frac{N}{2}-1} x[2n]\omega_{\frac{N}{2}}^{nk}$ 和第二项 $\sum_{n=0}^{\frac{N}{2}-1} x[2n+1]\omega_{\frac{N}{2}}^{nk}$ 分别是在对 $[x[0], x[2]]^{\mathrm{T}}$ 和 $[x[1], x[3]]^{\mathrm{T}}$ 作 2 阶 DFT。

重新对矩阵排列我们发现，

$$\begin{aligned} \tilde{F}_4 \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} &= \begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix} \begin{bmatrix} \tilde{F}_2 \begin{bmatrix} x[0] \\ x[2] \end{bmatrix} & \\ & \tilde{F}_2 \begin{bmatrix} x[1] \\ x[3] \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix} \begin{bmatrix} \tilde{F}_2 & \\ & \tilde{F}_2 \end{bmatrix} \begin{bmatrix} x[0] \\ x[2] \\ x[1] \\ x[3] \end{bmatrix} \end{aligned}$$

对于一般的 $n = 2, 4, \cdots, 2^k, \cdots$, 不难证明有如下的递推公式:

$$\tilde{F}_{2n} = \begin{bmatrix} I_n & D_n \\ I_n & -D_n \end{bmatrix} \begin{bmatrix} \tilde{F}_n & 0 \\ 0 & \tilde{F}_n \end{bmatrix} P_{2n}.$$

这些矩阵的具体定义如下:

1. $D_n = \text{diag}\{1, \omega_{2n}, \cdots, \omega_{2n}^{n-1}\}$.
2. $\tilde{F}_n = \frac{1}{n}\,\text{DFT}(n)$.
3. $P_{2n} = [e_1, e_3, ..., e_{2n-1}, e_2, e_4, ..., e_{2n}]^{\mathrm{T}}$，其中 e_i 是第 i 个元素为 1 的单位列向量.

观察到 $\begin{bmatrix} I_n & D_n \\ I_n & -D_n \end{bmatrix}$ 和 P_{2n} 已经满足了每行的非零元不超过两个，下一步就是按照同样的方法对 $\tilde{F}_N$ 进行分解。

由分块矩阵的性质，我们有

$$\begin{bmatrix} \tilde{F}_n & 0 \\ 0 & \tilde{F}_n \end{bmatrix} = \begin{bmatrix} I_{n/2} & D_{n/2} & & \\ I_{n/2} & -D_{n/2} & & \\ & & I_{n/2} & D_{n/2} \\ & & I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} \tilde{F}_{n/2} & & & \\ & \tilde{F}_{n/2} & & \\ & & \tilde{F}_{n/2} & \\ & & & \tilde{F}_{n/2} \end{bmatrix} \begin{bmatrix} P_n & 0 \\ 0 & P_n \end{bmatrix}.$$

同理只需要继续分解中间的矩阵。以此类推，当 $n = 4$ 时，我们有

$$\tilde{F}_{n/2} = \tilde{F}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

此时可以满足矩阵每行的非零元不超过两个。

结合上述分析，对于 N 阶 DFT 矩阵（$N = 2^t, t = 1, 2, \cdots$），我们总能分解成如下形式

$$\tilde{F}_N = \text{diag}_N\left(\begin{bmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix}\right) \cdot \text{diag}_N\left(\begin{bmatrix} I_{N/4} & D_{N/4} \\ I_{N/4} & -D_{N/4} \end{bmatrix}\right) \cdots \text{diag}_N\left(\begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix}\right) \cdot \text{diag}_N(\tilde{F}_2) \cdot \text{diag}_N(P_4)\,\text{diag}_N(P_8) \cdots \text{diag}_N(P_N). \quad (4.1)$$

其中 $\text{diag}_N(\cdot)$ 的含义是用括号内的矩阵构成 N 阶的分块对角矩阵。

不难证明排列矩阵 P_k 所构成 N 阶的分块对角矩阵 $\text{diag}_N(P_k)$ 也是排列阵，若干个排列阵的乘积也是排列阵，因此可将分解结果里后面的排列矩阵的乘积作为一个矩阵

$$A_K = \text{diag}_N(P_4)\,\text{diag}_N(P_8) \cdots \text{diag}_N(P_N).$$

因此我们有最终的分解结果如下：

$$A_1 = \text{diag}_N(\begin{bmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix})$$

$$A_2 = \text{diag}_N(\begin{bmatrix} I_{N/4} & D_{N/4} \\ I_{N/4} & -D_{N/4} \end{bmatrix})$$

$$\cdots$$

$$A_k = \text{diag}_N(\begin{bmatrix} I_{N/2^k} & D_{N/2^k} \\ I_{N/2^k} & -D_{N/2^k} \end{bmatrix})$$

$$\cdots$$

$$A_{K-2} = \text{diag}_N(\begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix})$$

$$A_{K-1} = \text{diag}_N(\tilde{F}_2)$$

$$A_K = \prod_{t=2}^{\log_2 N} \text{diag}_N(P_{2^t})$$

其中 K 的值由 N 确定：$K = 2 + (\log_2 N - 1) = 1 + \log_2 N$。那么，最后我们可以取 $\beta = 1/\sqrt{N}$，

$$F_N = \frac{1}{\sqrt{N}} F_N = \beta A_1 A_2 \cdots A_K.$$

4.2.2 方案硬件复杂度的分析与计算

在本题中，考虑乘法器的硬件复杂度的计算公式为 $C = q \times L$.

问题一中默认单个乘法器的复杂的 $q = 16$，因此只需考虑乘法器个数，即复数乘法的次数 L，并且由于问题一要求对 $N = 2^t, t = 1, 2, 3, \cdots$ 的 N 阶 DFT 矩阵进行分解，因此我们需要找到一种通用的复杂度计算公式，它对于所有基于 radix-2 FFT 算法得到的 N 阶 DFT 矩阵分解结果均成立。

由于题目对于复杂度计算只提供了两个二阶矩阵相乘的例子，但通过对更一般或者实际中的情况进行分析，我们发现由于矩阵乘法满足结合律，在一组矩阵的乘法中可以有不同的计算顺序，并且不同计算顺序所对应的乘法器个数，即计入的复数乘法次数，可能相异。

以下列矩阵乘法为例，如果从左到右进行乘法，按照题目的对复数乘法次数的计算规则会得到 $L = 2$，但如果从右到左进行乘法会得到 $L = 4$。

$$\begin{bmatrix} \text{X}_0 \\ \text{X}_1 \\ \text{X}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -4 & -1 \\ 1 & -4 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2j \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}.$$

为了避免这种情况，我们希望从 DFT 的背景出发，在符合题目对复数乘法的假设下从另一角度考虑更稳定且符合实际要求的乘法器个数计算方法。

在实际应用中，DFT 矩阵是作用在复数列向量 $[x_0, x_1, \cdots, x_{N-1}]^{\mathrm{T}}$ 上来实现对一维时域信号的傅里叶变换。

$$\begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix} = F_N \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}.$$

因此我们在计算复杂度时也依托这样的实际背景，通过一组矩阵的乘积右乘一个复数列向量并计算总共的乘法次数得到该组矩阵乘法的乘法器个数，并且由于这个列向量是任意的，因此我们可以直接通过计算矩阵中满足计入复数乘法次数条件的元素个数来得到乘法器个数。

简单来说我们只考虑分解结果 A_k 中，满足 $A_k[l,m] \neq 0, \pm 1, \pm j \ \pm 1 \pm j$ 的元素个数为乘法器个数。由于 A_K 是排列矩阵，$A_{K-1} = \mathrm{diag}_N(F_2)$ 是一个元素全为 $1, -1$ 的矩阵，实际上我们只需要考虑前面 $K-2$ 个矩阵中复数乘法的次数。

这些矩阵是形如 $\mathrm{diag}_N(\begin{bmatrix} I_{N/2^k} & D_{N/2^k} \\ I_{N/2^k} & -D_{N/2^k} \end{bmatrix})$ 的分块对角阵，共有 $log_2 N - 1$ 个，对于每个 D_k 有一个 1 和一个 $-j$，因此 $\mathrm{diag}_N(\begin{bmatrix} I_{N/2^k} & D_{N/2^k} \\ I_{N/2^k} & -D_{N/2^k} \end{bmatrix})$ 上有 $N - 4 \times 2^{k-1}$ 个乘法器。

故对于 N 阶的 DFT 分解，其硬件复杂度如下：

$$C = 16 \times \sum_{k=1}^{\log_2 N - 1} (N - 2^{k+1}) = 16 \times [(\log_2 N - 3)N + 4].$$

4.3 问题一求解结果

4.3.1 以 $N = 8$ 为例展示分解结果

我们基于上述 Cooley-Tukey 快速傅里叶变换算法，对 F_N 直接进行精确分解，得到的分解矩阵满足约束 1（每行的非零元至多两个），并且我们的优化目标 $\mathrm{RMSE}(\mathcal{A}, \beta) = 0$，这种分解对 $N = 2^1, 2^2, \cdots$ 均成立，分解表达式已在 4.2.1 中给出。最后我们得到了计算复杂度 C 关于 N 的通项公式。

本节我们以 $N = 8$ 为例，对分解结果进行展示：

$$F_n = \frac{1}{\sqrt{8}} A_1 A_2 A_3 A_4.$$

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \omega_8 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \omega_8^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \omega_8^3 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -\omega_8 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -\omega_8^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -\omega_8^3 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \omega_8^2 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\omega_8^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \omega_8^2 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -\omega_8^2 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

$$A_4 = [e_0, e_4, e_2\ e_6, e_1, e_5, e_3, e_7]^T$$

$$\omega_8 = \frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}j$$

硬件复杂度为

$$C = 16 \times \sum_{k=1}^{\log_2 8-1} (8 - 2^{k+1}) = 64.$$

其中，复乘次数 L 为 4.

4.3.2 DFT 矩阵与矩阵分解连乘的复杂度对比分析

在 4.2.2 中，我们已计算出对 N 阶的 DFT 分解，其硬件复杂度为（固定 $q = 16$）

$$C = 16 \times [(\log_2 N - 3)N + 4].$$

因此可以认为 N 阶 DFT 分解的硬件复杂度为 $\mathcal{O}(N\log_2 N)$，N 阶 DFT 矩阵直接作用于向量的硬件复杂度是 $\mathcal{O}(N^2)$. 可以看出二者具有显著差别。数值对比如图 4-1 所示：

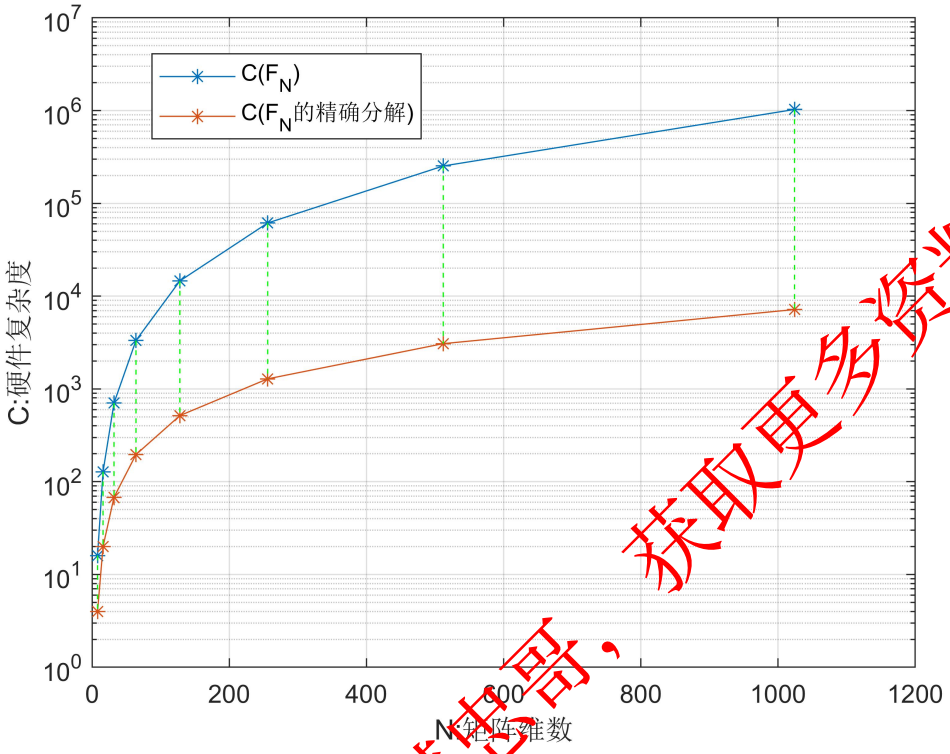

图 4.1　DFT 矩阵与矩阵分解连乘的复杂度对比

5 问题二的模型建立与求解

5.1 问题二分析

5.1.1 目标函数的确定

问题二通过减少限制 A_k 中元素实部和虚部取值范围的方式来降低硬件复杂度，优化目标为矩阵 F_N 和 $\beta A_1A_2\cdots A_K$ 在 Frobenius 范数意义下尽可能接近，同时还要满足约束 2：限定 $\mathcal{A}$ 中每个矩阵 A_k 的元素满足以下要求：

$$A_k[l,m]\in\{x+jy|x,y\in\mathcal{P}\},\mathcal{P}=\{0,\pm1,\pm2,\cdots,\pm2^{q-1}\},$$

$$k=1,2,\cdots,K;\quad l,m=1,2\cdots N.$$

注意到问题二中，A_k 中的元素的实部和虚部被限制在 $\mathcal{P}$ 中，由整数域的封闭性，当 $t>2$ 时，在约束 2 下不存在矩阵 βF_N 的精确分解。在不存在精确分解的情况下，出于最

小化目标函数的目的，我们对 βF_N 和 $A_1A_2\cdots A_K$ 之间的 Frobenius 范数进行分析，有

$$\text{RMSE}(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\|\beta F_N-A_1A_2\cdots A_K\|_F^2}=\frac{\beta}{N}\sqrt{\|F_N-\frac{1}{\beta}A_1A_2\cdots A_K\|_F^2}.$$

可以发现，该目标函数在缩放因子 β 的设置上存在两个问题：

1. 减小 β 带来的好处过于显著：在 $\mathcal{A}$ 中加入零矩阵，使得 $A_1A_2\cdots A_K=0$，此时矩阵分解同时满足约束 1 和约束 2. 取 β 为一个接近 0 的极小正实值，此时有

$$\text{RMSE}(\mathcal{A},\beta)=\frac{\beta}{N}\sqrt{\|F_N-\frac{1}{\beta}A_1A_2\cdots A_K\|_F^2}=\frac{\beta}{N}\|F_N\|_F\to 0\quad(\beta\to 0^+).$$

显然，若要最小化目标函数，取 A_k 为零矩阵，β 充分接近 0 即可。这与本题用矩阵连乘逼近 DFT 矩阵的初衷矛盾。

2. 增大 β 带来的坏处较大：可以发现，当我们试图通过增大 β 以使 $\frac{1}{\beta}A_1A_2\cdots A_K$ 与 F_N 更相似时，范数外部系数 β 的存在会导致范数增大，而本题使用 Frobenius 范数的目的仅是描述 F_N 和 $\frac{1}{\beta}A_1A_2\cdots A_K$ 之间的相似程度，两者之间存在矛盾。

另一方面，在问题二中，我们在尝试简单的优化方案时发现，目标函数 $\text{RMSE}(\mathcal{A},\beta)$ 的值总是大于 0.5，且改进优化方案后，目标函数值下降的幅度不足。这与问题 5 中约束条件更多，但却要求目标函数小于 0.1 发生矛盾。

综合上述考虑，我们将缩放因子 β 的设置进行修改，将目标函数确定为以下形式：

$$\text{RMSE}(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\|F_N-\beta A_1A_2\cdots A_K\|_F^2},$$

从而避免了上述的两处矛盾。

5.1.2 矩阵修正思想及其模型简化

考虑问题一中我们提出的 DFT 矩阵 F_N 的精确分解：

$$F_N=\frac{1}{\sqrt{N}}\tilde{F}_N=\frac{1}{\sqrt{N}}A_1A_2\cdots A_K,$$

其中，$A_k(k=1,2,\cdots,K)$ 是每行至多两个非零元的稀疏矩阵，满足约束 1，但不一定满足约束 2.

由问题一中的分解结果，我们有

$$A_k = \text{diag}_N(\begin{bmatrix} I_{N/2^k} & D_{N/2^k} \\ I_{N/2^k} & -D_{N/2^k} \end{bmatrix})$$

$$\cdots$$

$$A_{K-2} = \text{diag}_N(\begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix})$$

$$A_{K-1} = \text{diag}_N(F_2)$$

$$A_K = \prod_{t=2}^{\log_2 N} \text{diag}_N(P_{2^t})$$

其中，

A_K 是置换矩阵，仅含有元素 1 和 0，即 $A_K[l,m] \in \mathcal{P}^2$；

A_{K-1} 由 F_2 和 0 组成，而 F_2 中仅含有元素 1 和 -1，即 $A_{K-1}[l,m] \in \mathcal{P}^2$；

A_{K-2} 由 I_2 和 $\pm D_2$ 组成，而 $\pm D_2$ 中仅含有元素 $-j, -1, j, 1$，即 $A_{K-2}[l,m] \in \mathcal{P}^2$；

A_{K-3} 由 I_4 和 $\pm D_4$ 组成，但 D_4 中包含元素 $\sqrt{2}/2 + \sqrt{2}/2j$，因此 A_{K-3} 不满足约束 2；

$A_{K-4}, A_{K-5}, \cdots$ 均与 A_{K-3} 类似。

当 $N = 2^1 = 2$ 或 $N = 2^2 = 4$ 时，有

$$\sqrt{2}F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad 2F_4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \text{diag}_4(F_2) \begin{bmatrix} {e_1}^T \\ {e_3}^T \\ {e_2}^T \\ {e_4}^T \end{bmatrix},$$

可以看出，当 $t = 1, 2$ 时，$\sqrt{N}F_N = F_{2^t}$ 均存在符合约束 2 的精确分解，此时目标函数的值与硬件复杂度均为 0.

当 $N = 2^t, t \geq 3$ 时，由上述分解结果的分析可知，F_N 的精确分解中总存在 3 个满足约束 2 的矩阵，以及 $t-2$ 个不满足约束 2 的矩阵。在此基础上，我们提出一种“矩阵修正”的思想，即对不满足约束 2 的 $t-2$ 个矩阵进行修正，使得它们在满足约束 2 的同时，尽可能地使连乘的结果逼近 F_N.

另一方面，若将 F_N 的精确分解中满足约束 2 的 3 个矩阵纳入逼近优化模型的搜索空间，模型优化的效果可能更佳，但同时会导致硬件复杂度上升与求解难度的大幅增高。因此，出于简化模型和控制矩阵稀疏性的目的，我们在进行矩阵修正逼近时，仅对不满足约束 2 的 $t-2$ 个矩阵进行修正，而不考虑满足约束 2 的 3 个矩阵。

5.2 问题二数学模型：矩阵连乘后的修正逼近

在问题二中需要建立如下的数学规划模型

$$\min_{\mathcal{A},\beta} \text{RMSE}(\mathcal{A},\beta) = \frac{1}{N}\sqrt{\|F_N - \beta A_1 A_2 \cdots A_K\|_F^2}$$
$$\text{s.t.} A_k[l,m] \in \mathcal{P}^2, k = 1,2,\cdots,K, l,m = 1,2,\cdots,N.$$

其中，$\mathcal{P}^2 = \{a+bi | a,b = 0,\pm1,\pm2,\cdots,\pm2^{q-1}\}, q = 3.$

在问题一的基础上，我们知道 DFT 矩阵 F_N 可以精确分解成如下形式：

$$F_N = \frac{1}{\sqrt{N}} A_1 A_2 \cdots A_K,$$

由 5.1.2 中的叙述，A_K, A_{K-1}, A_{K-2} 均满足元素在空间 $\mathcal{P}^2$ 的约束条件，而当 $1 \leqslant k \leqslant K-3$ 时，A_k 的元素并不是都在 $\mathcal{P}^2$ 中。

考虑到问题二中没有对分解矩阵的稀疏性做出约束，我们可以通过减弱分解矩阵稀疏性的方式，来提高矩阵连乘逼近的精度，即减小目标函数的值。为此我们将前 $K-3$ 个矩阵连乘成一个矩阵，减弱稀疏性后再进行修正逼近，即设

$$A_0 = A_1 A_2 \cdots A_{K-3}.$$

因为 A_0 的元素并不全在 $\mathcal{P}^2$ 中，我们试图通过最邻近插值的方法去“改善” A_0. 具体来说，我们先限制 A_{K-2}, A_{K-1}, A_K 不变，找到一个符合约束 2 的矩阵 X，用 $\beta^* X$ 去近似 A_0 来达到一定程度上的优化（$\beta^* = \sqrt{N}\beta$ 是根据问题背景选取的若干个离散实值，X 与 β^* 的选值相关），下面的推导说明目标函数 RMSE$(\mathcal{A},\beta)$ 的值与 A_{K-2}, A_{K-1}, A_K 无关，体现了我们限制 A_{K-2}, A_{K-1}, A_K 不变的想法是可行的：

$$\begin{aligned}
\min_{X,\beta^*} \text{RMSE}(\mathcal{A},\beta^*) &= \frac{1}{N}\|F_N - \frac{1}{\sqrt{N}} \cdot \beta^* X \cdot A_{K-2}A_{K-1}A_K\|_F \\
&= \frac{2}{N^{3/2}}\|(A_0 - \beta^* X)\frac{A_{K-2}}{\sqrt{2}}\frac{A_{K-1}}{\sqrt{2}}A_K\|_F \qquad (5.2) \\
&= \frac{2}{N^{3/2}}\|A_0 - \beta^* X\|_F \qquad (5.3)
\end{aligned}$$

其中(5.2)到(5.3)的推导使用了 Frobenius 范数在酉变换下的不变性，即对于 n 阶方阵 A 和 n 阶酉矩阵 U，有 $\|A\|_F = \|AU\|_F$. 由问题一中的定义，$\frac{A_{K-2}}{\sqrt{2}}, \frac{A_{K-1}}{\sqrt{2}}, A_K$ 均是酉矩阵，证明如下：

1. $\sqrt{2}^{-1} A_{K-2} = \sqrt{2}^{-1}\,\text{diag}_N(\begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix})$，因此根据分块对角矩阵的性质，只需考虑

$$\frac{1}{2}\begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix}^H \begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} I_2 + I_2 & D_2 - D_2 \\ \bar{D}_2 - \bar{D}_2 & \bar{D}_2 D_2 + \bar{D}_2 D_2 \end{bmatrix} = I.$$

2. $\sqrt{2}^{-1}A_{K-1} = \sqrt{2}^{-1}\,\mathrm{diag}_N(F_2)$，同理考虑

$$\frac{1}{2}F_2^H F_2 = \frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}^H\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} = I.$$

3.$A_K = P$，由置换矩阵的性质有 $P^H P = I$. □

因此问题转化为优化如下的目标函数：

$$B = \arg\min_{X\in M(\mathcal{P}^2)} \|\beta^* X - A_0\|_F.$$

由此，对于某个特定的 β^*，我们能够找到 F_N 在 Frobenius 范数下的最佳近似 $\hat{F}_N$：

$$F_N \approx \hat{F}_N = \frac{\beta^*}{\sqrt{N}} B \cdot A_{K-2}A_{K-1}A_K.$$

此时 B, A_{K-2}, A_{K-1}, A_K 均是满足约束 2 的矩阵，令 $\beta = \frac{1}{\sqrt{N}}\beta^*$，我们就能找到一种关于 β 的最优近似。

5.3 问题二模型求解

5.3.1 矩阵插值逼近算法

在 5.2 中提到，我们试图通过最邻近插值的方法去“改善” A_0，即解如下的目标函数：

$$B = \arg\min_{X\in M(\mathcal{P}^2)} \|\beta^* X - A_0\|_F.$$

本节主要介绍的是求解该目标函数的矩阵插值逼近算法。

对于 $A_0[l,m] = a + b\mathrm{j}$，要求它的最佳近似，就是要在集合 $\beta^*\mathcal{P}^2$ 上找到距离它最近的点（以复数的模来描述距离）。由于 $\beta^*\mathcal{P}^2$ 中实部和虚部的选值没有关联性的限制，所以最近的点满足实部和虚部都最近，因此我们只需要讨论实部或虚部其中一个在集合 $\beta^*\mathcal{P}$ 上最近的点，不妨以实部为例。

$\mathcal{P} = \{0, \pm1, \pm2, \cdots, \pm2^{q-1}\}$，令 $a = (\mathrm{Sgn}\,a)\cdot|a|$，用函数 round$(a)$ 表示 a 在整数下的四舍五入. 若 $0 \le |a| < 2.5$，显然 $\mathcal{P}$ 中与 a 距离最近的点为 $(\mathrm{Sgn}\,a)\cdot\mathrm{round}(|a|)$.

若 $|a| \ge 2.5$，设 $|a| \in [2^t, 2^{t+1}](t = 1, 2, \cdots)$，则有 $\log_2|a| \in [t, t+1]$，由对数函数的单调性可知，此时 $\mathcal{P}$ 中与 a 距离最近的点为 $(\mathrm{Sgn}\,a)\cdot 2\text{^}[\mathrm{round}(\log_2|a|)]$.

以上分析和讨论均在集合 $\mathcal{P}$ 上进行，而实际上的插值集合为 $\beta^*\mathcal{P}$. 因此，我们将目标函数调整为

$$B = \arg\min_{X\in M(\mathcal{P}^2)} \beta^*\|X - \frac{1}{\beta^*}A_0\|_F,$$

从而将插值空间调整到 $\mathcal{P}$.

综合上述分析和讨论，我们提出如下的矩阵插值逼近算法：

算法 1 矩阵插值逼近

输入: 待逼近矩阵 A_0，缩放因子 β^*

输出: A_0 的最佳逼近 B

/* 检索最近插值 */
for A_0 中每个元素 $A_0[l,m]$ **do**
    $a+bj \leftarrow A_0[l,m]$
    $a^*+b^*j \leftarrow (a+bj)/\beta^*$
    if $0 \le a^* < 2.5$ **then**
        $a^* \leftarrow (\mathrm{Sgn}\, a^*)\cdot \mathrm{round}(|a^*|)$
    else
        $a^* \leftarrow (\mathrm{Sgn}\, a^*)\cdot 2\text{^}[\mathrm{round}(\log_2 |a^*|)]$
    if $0 \le b^* < 2.5$ **then**
        $b^* \leftarrow (\mathrm{Sgn}\, b^*)\cdot \mathrm{round}(|b^*|)$
    else
        $b^* \leftarrow (\mathrm{Sgn}\, b^*)\cdot 2\text{^}[\mathrm{round}(\log_2 |b^*|)]$
    /* 为 B 赋值 */
    $B[l,m] \leftarrow a^*+b^*j$
return B

5.3.2 β^* 的分析与选取

对分解矩阵 A_k 进行观察分析

$$A_k = \mathrm{diag}_N\left(\begin{bmatrix} I_{N/2^k} & D_{N/2^k} \\ I_{N/2^k} & -D_{N/2^k} \end{bmatrix}\right).$$

可以发现，A_k 中存在大量元素 1，其数量与 ω_N^α 的数量相等。

5.3.1 中提出的矩阵插值逼近算法本质上是用 $\mathcal{P}^2$ 中的点去近似 ω_N^α，通过选取恰当的 β^*，可以找到 $\mathcal{P}^2$ 中距离 ω_N^α/β^* 最近的点，从而使得目标函数中 $\beta^*\|X-\frac{1}{\beta^*}A_0\|_F$ 的值更小。

另一方面，由于 A_k 中存在大量元素 1，经过缩放因子 β^* 的变换后，元素 1 变为元素 $1/\beta^*$，若 $1/\beta^*$ 距离 $\mathcal{P}$ 中的点过远，会使得 $\beta^*\|X-\frac{1}{\beta^*}A_0\|_F$ 的值显著增大，因此 β^* 的选值应接近 $2^{-t}(t=0,1,\cdots)$. 考虑到问题二中固定 $q=3$，即 $\mathcal{P}=\{0,\pm1,\pm2,\pm4\}$，因此 β^* 的选值应充分接近 $1,\frac{1}{2},\frac{1}{4}$. 我们分别在 $N=8,N=16,N=32$ 时，对 β^* 在 $1,\frac{1}{2},\frac{1}{4}$ 这三个点的某邻域进行 F 范数逼近的数值实验，得到结果如图5.1所示。

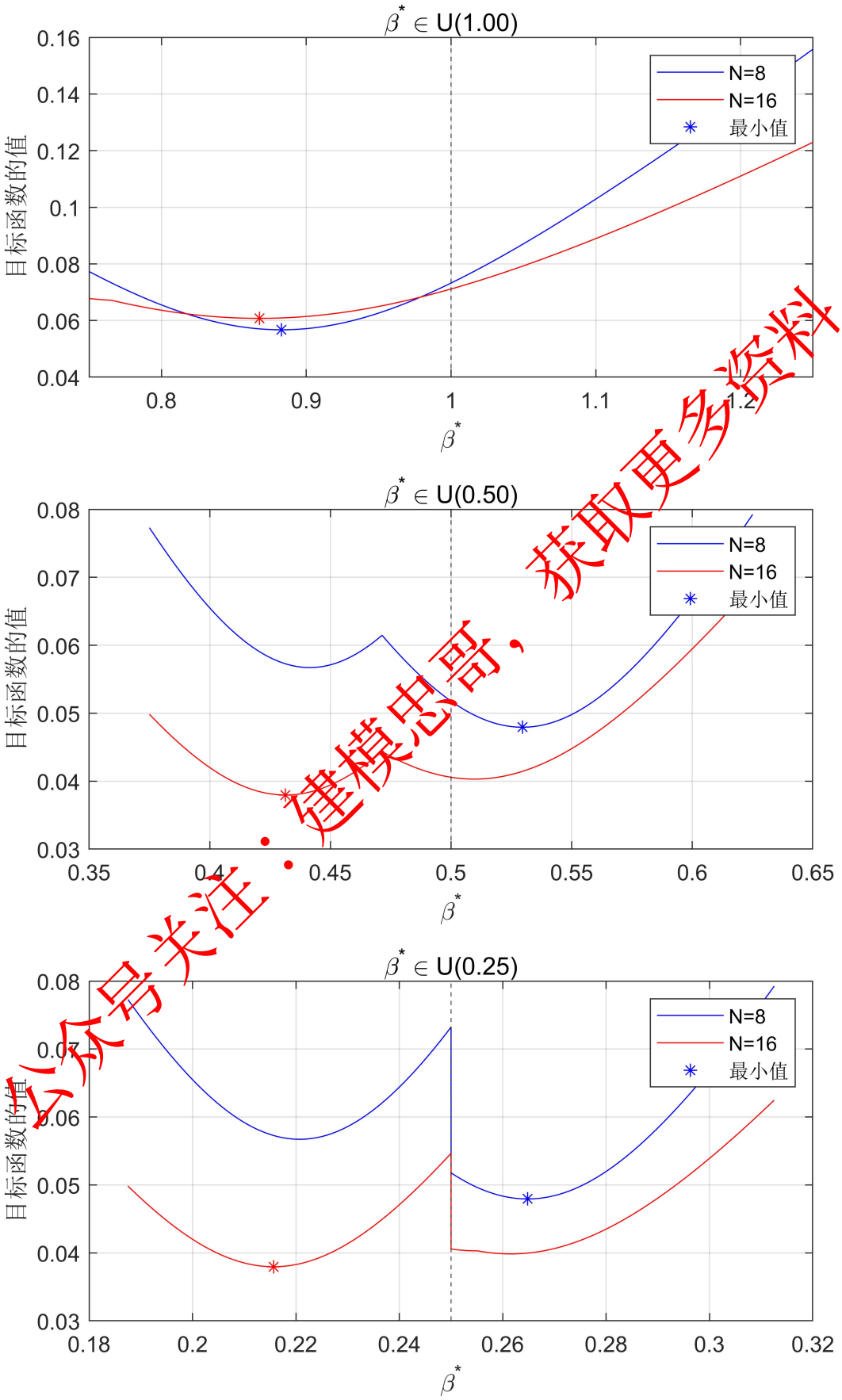

图 5.1　β^* 的选值问题

可以看出，在 $1, \frac{1}{2}, \frac{1}{4}$ 这三个点的邻域内，总存在一个极小值点 β^*，使得目标函数的值在领域内取得最小值。我们尚未求得该极小值点的解析表达式，但可以通过邻域搜索的方法找到它的一个数值解。因此，我们提出如下的缩放因子邻域搜索算法：

算法 2 缩放因子邻域搜索

输入: 待逼近矩阵 A_0，缩放因子中心点集合 $C_\beta = \{2^{-t} | t = 0, 1, \cdots, q-1\}$，邻域范围 α

输出: 最佳缩放因子 β^*，A_0 的最佳逼近 B

/* 算法介绍：在某中心点的邻域 $[2^{-t} - \alpha, 2^{-t} + \alpha]$ 均匀离散地选取若干个点（一般取 1000），组成邻域离散点集 $U^\alpha(t)$，该邻域中局部最佳的缩放因子记作 $V[t]$，最后在 V 中选取全局最佳的缩放因子 β^**/

for C_β 中每个元素 2^{-t} **do**

  for $U^\alpha(t)$ 中每个元素 β_u **do**

    $B^* \leftarrow$ 对 A_0 使用 β_u 进行矩阵插值逼近

    $K[u] \leftarrow \|\beta_u B^* - A_0\|_F$

  $Q[t] \leftarrow K$ 中的最小值 $K[u^*]$

  $u^* \leftarrow K$ 中最小值 $K[u^*]$ 的次序

  $V[t] \leftarrow \beta_{u^*}$

$t^* \leftarrow Q$ 中最小值 $Q[t^*]$ 的次序

$\beta^* \leftarrow V[t^*]$

$B \leftarrow$ 对 A_0 使用 β^* 进行矩阵插值逼近

return B, β^*

5.4 问题二求解结果

5.4.1 矩阵连乘拟合结果

根据上述矩阵连乘后的修正逼近模型，以及对应的矩阵插值逼近算法和缩放因子邻域搜索算法，我们对 $N = 2^t, t = 1, 2, 3, 4, 5$ 的 DFT 矩阵 F_N 进行约束 2 下的矩阵连乘修正拟合。

1. 当 $t = 1, N = 2^t = 2$ 时，有

$$F_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

显然，F_2 自身同时满足约束 1 与约束 2，且硬件复杂度为 0，无需做任何分解.

2. 当$t=2, N=2^t=4$时，有

$$F_4=\frac{1}{4}\begin{bmatrix}1&0&1&0\\0&1&0&-j\\1&0&-1&0\\0&1&0&j\end{bmatrix}\operatorname{diag}_4(F_2)\begin{bmatrix}{e_1}^T\\{e_3}^T\\{e_2}^T\\{e_4}^T\end{bmatrix},\tag{5.4}$$

可以看出，由 F_4 精确分解得到的三个矩阵均满足约束 1 和约束 2，此时目标函数的值为 0，且硬件复杂度为 0.

3. 当 $t=3, N=2^t=8$ 时，我们求解得到如下的最佳连乘分解矩阵（置换矩阵连乘后仍是置换矩阵，因此视作一个矩阵）：

$$F_8\approx\frac{0.5296}{\sqrt{8}}\begin{bmatrix}S_1&S_2\\S_1&-S_2\end{bmatrix}\operatorname{diag}_8(\begin{bmatrix}I_2&D_2\\I_2&-D_2\end{bmatrix})\operatorname{diag}_8(F_2)\prod_{t=2}^{3}\operatorname{diag}_8(P_{2^t}),\tag{5.5}$$

其中，

$$S_1=\begin{bmatrix}2&0&0&0\\0&2&0&0\\0&0&2&0\\0&0&0&2\end{bmatrix},\quad S_2=\begin{bmatrix}2&0&0&0\\0&1-j&0&0\\0&0&-2j&0\\0&0&0&-1-j\end{bmatrix}$$

4. 当$t=4, N=2^t=16$时，我们求解得到如下的最佳连乘分解矩阵：

$$F_{16}\approx\frac{0.2157}{\sqrt{16}}\begin{bmatrix}K_1&K_2\\K_1&-K_2\end{bmatrix}\operatorname{diag}_{16}(\begin{bmatrix}I_2&D_2\\I_2&-D_2\end{bmatrix})\operatorname{diag}_{16}(F_2)\prod_{t=2}^{4}\operatorname{diag}_{16}(P_{2^t})$$

其中

$$K_1=\begin{bmatrix}4&0&0&0&4&0&0&0\\0&4&0&0&0&4-4j&0&0\\0&0&4&0&0&0&-4j&0\\0&0&0&4&0&0&0&-4-4j\\4&0&0&0&-4&0&0&0\\0&4&0&0&0&-4+4j&0&0\\0&0&4&0&0&0&4j&0\\0&0&0&4&0&0&0&4+4j\end{bmatrix}$$

$$K_2 = \begin{bmatrix} 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 4-2j & 0 & 0 & 0 & 2-4j & 0 & 0 \\ 0 & 0 & 4-4j & 0 & 0 & 0 & -4-4j & 0 \\ 0 & 0 & 0 & 2-4j & 0 & 0 & 0 & -4+2j \\ -4j & 0 & 0 & 0 & 4j & 0 & 0 & 0 \\ 0 & -2-4j & 0 & 0 & 0 & 4+2j & 0 & 0 \\ 0 & 0 & -4-4j & 0 & 0 & 0 & 4-4j & 0 \\ 0 & 0 & 0 & -4-2j & 0 & 0 & 0 & -4-4j \end{bmatrix}$$

可以看出该矩阵分解满足约束 2.

5. 当$t=5, N=2^t=32$时，我们求解得到如下的最佳连乘分解矩阵：

$$F_{32} \approx \frac{0.2179}{\sqrt{32}} \begin{bmatrix} J_1 & J_2 \\ J_1 & -J_2 \end{bmatrix} \mathrm{diag}_{32}\left(\begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix}\right) \mathrm{diag}_{32}(P_2) \prod_{t=2}^{5} \mathrm{diag}_{32}(P_{2^t})$$

其中

$$J_1 = \begin{bmatrix} M_1 & M_2 \\ M_1 & -M_2 \end{bmatrix}, \quad J_2 = \begin{bmatrix} Q_1 & Q_2 \\ -jQ_1 & jQ_2 \end{bmatrix}, \quad M_1 = K_1, \quad M_2 = K_2,$$

$$Q_1 = \begin{bmatrix} 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 4-j & 0 & 0 & 0 & 2-4j & 0 & 0 \\ 0 & 0 & 4-2j & 0 & 0 & 0 & -2-4j & 0 \\ 0 & 0 & 0 & 4-2j & 0 & 0 & 0 & -4-j \\ 4-4j & 0 & 0 & 0 & -4+4j & 0 & 0 & 0 \\ 0 & 2-4j & 0 & 0 & 0 & 1+4j & 0 & 0 \\ 0 & 0 & 2-4j & 0 & 0 & 0 & 4+2j & 0 \\ 0 & 0 & 0 & 1-4j & 0 & 0 & 0 & 4-2j \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 4-2j & 0 & 0 & 0 & 1-4j & 0 & 0 \\ 0 & 0 & 2-4j & 0 & 0 & 0 & -4-2j & 0 \\ 0 & 0 & 0 & -1-4j & 0 & 0 & 0 & -2+4j \\ -4-4j & 0 & 0 & 0 & 4+4j & 0 & 0 & 0 \\ 0 & -4-j & 0 & 0 & 0 & 4-2j & 0 & 0 \\ 0 & 0 & -4+2j & 0 & 0 & 0 & -2-4j & 0 \\ 0 & 0 & 0 & -2+4j & 0 & 0 & 0 & -4+j \end{bmatrix}$$

5.4.2 最小误差与硬件复杂度

对 $N = 2^t, t = 1, 2, 3, 4, 5$ 的 DFT 矩阵 F_N 进行约束 2 下的矩阵连乘修正拟合，我们得到了 5.4.1 中的连乘分解结果。对于每个维度下的连乘修正拟合，我们计算出它的最小误差、硬件复杂度以及其他相关指标如表5.1所示（其中 $C = q \times L = 3L$）

表 5.1　问题二结果

矩阵维数 N	目标函数值	$\beta^*(=\sqrt{N}\beta)$	缩放因子 β	复乘次数 L	硬件复杂度 C
2	0	1	$\sqrt{2}/2$	0	0
4	0	1	$1/2$	0	0
8	0.0479	0.5296	0.1872	12	36
16	0.0380	0.2157	0.0539	64	192
32	0.0251	0.2179	0.0385	272	816

另一方面，若我们未按照 5.1.1 中的方法对目标函数中缩放因子 β 的设置进行修改，则目标函数应为

$$\mathrm{RMSE}(\mathcal{A}, \beta) = \frac{1}{N}\sqrt{\|F_N - \frac{1}{\beta}A_1 A_2 \cdots A_K\|_F^2},$$

不难看出，修改前目标函数的值为修改后的 $\frac{1}{\beta}$ 倍（注意：上式中的 β 为修改前的，即 β 是 F_N 的系数，与本句中的 β 二者互为倒数关系）。我们计算得出修改前目标函数的值如表5.2所示。

表 5.2　修改目标函数前的结果

矩阵维数 N	（修改前）目标函数值	缩放因子 β
2	0	$\sqrt{2}/2$
4	0	$1/2$
8	0.5120	0.0936
16	0.7041	0.0539
32	0.6528	0.0385

可以看出，修改前目标函数值显著大于 0.1，将导致我们无法对问题五进行求解。

6 问题三的模型建立与求解

6.1 问题三分析

问题三考虑 N 阶 DFT 的近似分解，$N = 2^t (t = 1, 2, 3, 4, 5)$.

$$F_N \approx \beta A_1 \cdots A_K.$$

其中 $A_k (k = 1, \cdots, K)$ 要求满足

约束 1：A_k 每行的非零元至多两个，满足这种条件的矩阵组成的集合记为 M_{Sparse}

约束 2：$A_k[l, m] \in \mathcal{P}^2 (\mathcal{P}^2 = \{a + bi | a, b = 0, \pm 1, \pm 2, \cdots, \pm 2^{q-1}\})$，满足这种条件的矩阵组成的集合记为 $M(\mathcal{P}^2)$.

在此约束下对矩阵连乘与 F_N 在 Frobenius 范数意义下的近似程度进行优化，即

$$\min_{\mathcal{A},\beta} \text{RMSE}(\mathcal{A}, \beta) = \frac{1}{N} \| F_N - \beta A_1 A_2 \cdots A_K \|_F.$$

基于前两问的数学模型，对问题三的模型构建出现了两种思路。一是直接对 F_N 的元素在集合 $\mathcal{P}^2$ 中选取插值近似得到 $\hat{F}_N$，再对 $\hat{F}_N$ 实现在缩放因子作用下的连乘分解 [2]，即

$$\hat{F}_N = \beta A_1 \cdots A_K, \quad A_k \in M(\mathcal{P}^2) \cap M_{Sparse}.$$

这种思路的好处在于直接对 F_N 进行近似，此时目标函数是可控的，但随之而来的问题是 $\hat{F}_N$ 应该在一个什么样的矩阵空间上才能存在一个这样的矩阵分解。

参考文献 [2, 3] 中提到了 $N = 8, 16$ 两种情况的近似方法和整数分解，他们选取 $\{x + iy | x, y = 0, \pm\frac{1}{2}, \pm 1, \pm 2\}$ 作为 $\hat{F}_N$ 的元素取值集合，并对 $\hat{F}_N$ 进行的分解矩阵也属于 $M(\mathcal{P}^2) \bigcap M_{Sparse}$，满足题目条件。但当 $N = 32, 64, \cdots$ 时，原文并没有论述这种 $\hat{F}_N$ 的元素取值集合选取的原则、什么样的原则能存在这种稀疏整数矩阵分解，以及设置缩放因子。总体来说，这种思路在本题的背景限制下存在一定的困难。

因此我们考虑第二种思路。结合问题一的结论，我们首先可以对 $\tilde{F}_N = \frac{1}{\sqrt{N}} F_N$ 作精确分解，分解得到的矩阵均满足约束 1，即

$$\tilde{F}_N = \bar{D}_1 \bar{D}_2 \cdots \bar{D}_{\log_2 N - 1} \, \text{diag}_N(F_2) P, \quad \bar{D}_k = \text{diag}_N\left(\begin{bmatrix} I_{N/2^k} & D_{N/2^k} \\ I_{N/2^k} & -D_{N/2^k} \end{bmatrix}\right),$$

以此为基础，我们对其中不满足约束 2 的矩阵分别进行 “修正”，使得所有分解矩阵都属于 $M(\mathcal{P}^2) \bigcap M_{Sparse}$.

这种思路的关键在于构建修正模型，使得修正后矩阵连乘的结果能够在 Frobenius 范数意义下与 F_N 比较相似。下节我们将进行具体的论述。

6.2 问题三数学模型：矩阵连乘前的修正逼近

在问题三中需要建立如下的数学规划模型:

$$\min_{\mathcal{A},\beta} \text{RMSE}(\mathcal{A},\beta) = \frac{1}{N}\sqrt{\|F_N - \beta A_1 A_2 \cdots A_K\|_F^2}$$
$$\text{s.t. } A_k \in M(\mathcal{P}^2) \cap M_{Sparse}, k = 1, 2, \cdots, K.$$

由问题一的结果，我们知道 DFT 矩阵有精确分解如下：

$$F_N = \beta A_1 A_2 \cdots A_K.$$

根据 5.1.2 的分析，A_{K-2}, A_{K-1}, A_K 是满足约束 1 和 2 的矩阵，$A_k (1 \leqslant k \leqslant K-3)$ 满足约束 1，但不满足约束 2.

我们的思路是以 DFT 矩阵的精确分解为基础，固定已经满足约束 1，2 的分解矩阵，对 $\{A_k | 1 \leqslant k \leqslant K-3\}$ 分别作适当的近似 $\{\hat{A}_k | 1 \leqslant k \leqslant K-3\}$，使得它们同时满足约束 1 和约束 2。**直观上当这种近似比较好的时候，它们作矩阵乘法的乘积与 DFT 矩阵的差距也应当比较接近**。事实上实验结果也说明了这种思路是合理的。

自然地，在近似原则上，我们考虑让 A_k 与其近似矩阵 X_k 在相差一个缩放系数的情况下 Frobenius 距离最接近，而 $\hat{A}_k$ 需要满足约束 2：$\hat{A}_k[l,m] \in \mathcal{P}^2$. 也即是考虑如下形式的优化问题：

$$\begin{aligned} &\min_{X \in M(\mathcal{P}^2)} \|A_k - \beta^* X\|_F \\ &\text{s.t. } X[l,m] \in \mathcal{P}^2, \ k = 1, 2, \cdots, K-3, \\ &\qquad l, m = 0, 1, \cdots, N-1. \end{aligned} \tag{6.6}$$

注意到这个优化问题有两个决策变量，因此我们希望先将其中一个变量固定为离散值来求解优化问题，然后再对这些解进行比较。由于 X 的决策空间会随着维数的变大而指数增大，对 β^* 进行离散取值来搜索是比较适合的。

考虑到 X 的元素只能为 $0, \pm 1, \pm 2, \pm 4$，而 A_k 的元素是 ω_p^q。因此 β^* 的取值范围考虑在 $[\frac{1}{4}, 1]$ 比较合理。事实上，一旦 $\beta > 1$，则 βX 的所有元素的模都将大于 1，而 $A_k[l,m]$ 的模都为 1，显然不是最优的。同理，对于 $\beta < \frac{1}{4}$，βX 的所有元素的模都将小于 1，也不可能是最优。

在此基础上，我们对 $\beta \in [\frac{1}{4}, 1]$ 进行离散取值，使用 5.3.2 中提出的缩放因子邻域搜索算法，代入优化问题进行求解，得到选取最优的一组解 (X, β).

对(6.6)中 $K-3$ 个待修正矩阵进行优化近似，即对 $K-3$ 个优化问题按上述方法进行求解，我们能得到 F_N 的一个近似如下：

$$F_N \approx \frac{1}{N} \prod_{k=1}^{K-3} (\beta_k X_k) A_{K-2} A_{K-1} A_K.$$

至此，虽然我们以及对每个分解矩阵进行了在 Frobenius 意义下的最优近似，但它们的乘积与 F_N 的 Frobenius 距离仍有可能在实值放缩因子的作用下得到缩小。因此，我们最后再对缩放因子进行优化，我们可以统一的将这步优化写成如下形式：

$$\beta^* = \arg\min_{\beta\in\mathbb{R}} \|F_N - \beta \prod_{k=1}^{K-3} X_k \cdot A_{K-2}A_{K-1}A_K\|_F. \tag{6.7}$$

不难发现这个优化问题实际是关于 β 的二次函数。事实上，考虑

$$\|F_N - \beta X\|_F^2 = \sum_{i=1}^{N}\sum_{j=1}^{N} |F_N[i,j] - \beta X[i,j]|^2.$$

由于 $|z_1 - \beta z_2|^2 = [\mathrm{Re}(z_1) - \beta\,\mathrm{Re}(z_2)]^2 + [\mathrm{Im}(z_1) - \beta\,\mathrm{Im}(z_2)]^2$，因此目标函数是关于 β 的二次函数，可以直接计算解析解。

至此我们得到了 F_N 的最优近似分解：

$$F_N \approx \beta^* X_1X_2\cdots X_{K-3}\,\mathrm{diag}_N(\begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix})\,\mathrm{diag}_N(F_2)P.$$

6.3 问题三模型求解与求解结果

对 $N = 2^t$，考虑 F_N 的精确分解

$$X = \beta A_1A_2\cdots A_K$$

其中，A_{K-2}, A_{K-1}, A_K, 同时满足约束 1 和约束 2.

对 $A_k(k = 1, 2, \cdots, K-3)$，使用 5.3 中提出的矩阵插值逼近算法和缩放因子邻域搜索算法，进行 6.2 中式 (6.6) 的优化拟合，得到单个矩阵 A_k 的最优拟合矩阵 $\beta_k X_k$.

为跳出缩放因子集合 $\{\beta_1, \beta_2, \cdots, \beta_{K-3}\}$ 带来的局部最优，实现全局最优，我们进行 6.2 中式 (6.7) 的求解。已推知目标函数是关于 β 的二次函数，计算得到如下的最优解 β^*：

$$\begin{cases} a &= \sum_{i=1}^{N}\sum_{j=1}^{N}(\mathrm{Re}^2(X[i,j]) + \mathrm{Im}^2(X[i,j])) \\ b &= -2\sum_{i=1}^{N}\sum_{j=1}^{N}(\mathrm{Re}(F_N[i,j])\,\mathrm{Re}(X[i,j]) + \mathrm{Im}(F_N[i,j])\,\mathrm{Im}(X[i,j])) \\ \beta^* &= -\frac{b}{2a} \end{cases}$$

至此我们得到了 F_N 的近似分解：

$$F_N \approx \beta^* X_1X_2\cdots X_{K-3}\,\mathrm{diag}_N(\begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix})\,\mathrm{diag}_N(F_2)P.$$

对 F_N 的最优近似分解计算最小误差和硬件复杂度，得到结果和其他相关指标如表6.1所示（其中 $C = q \times L = 3L$）：

表 6.1　问题三结果

矩阵维数 N	目标函数值	$\beta^*(=\sqrt{N}\beta)$	缩放因子 β	复乘次数 L	硬件复杂度 C
2	0	1	$\sqrt{2}/2$	0	0
4	0	1	$1/2$	0	0
8	0.0479	0.5296	0.1872	12	36
16	0.0405	0.2661	0.0665	52	156
32	0.0353	0.0635	0.0112	184	552

结合问题二的结果进行分析，可以看出：

1. 由于约束条件的增多，连乘矩阵拟合的精度（目标函数值）不可避免地减少了；

2. 随着 N 的增大，连乘次数增加，缩放因子的值随之减小；

3. 由于分解矩阵均满足约束 1，连乘矩阵的稀疏性增强了，硬件复杂度较问题二的结果有所下降。

由于连乘矩阵的次数过多、维数过大，我们不在本节展示具体的连乘过程。

7 问题四的模型建立与求解

7.1 问题四分析与数学模型

问题四考虑对如下矩阵的近似分解，其中 F_n 是 n 阶 DFT 矩阵，$\otimes$ 表示 Kronecker 积。

$$F = F_4 \otimes F_8.$$

近似分解的要求和第三问相同，要求分解矩阵满足约束 1 和约束 2，并以 F 分解矩阵乘积和 F 在 Frobenius 范数意义下的近似程度为目标进行优化。

$$\begin{aligned} &\min \mathrm{RMSE}(\mathcal{A}, \beta) = \frac{1}{N}\|F - \beta A_1 A_2 \cdots A_K\|_F \\ &\text{s.t. } A_k \in M(\mathcal{P}^2) \cap M_{Sparse}, k = 1, 2, \cdots, K. \end{aligned} \tag{7.8}$$

本题的关键在于 F 是两个 DFT 矩阵的 Kronecker 积，但本身并非是 DFT 矩阵，因此无法使用问题三的结论。但是结合 Kronecker 积的性质，通过对 F 作适当的变换，最终这个问题依然能转化为问题三的情形。

通过观察 F 的特点，发现 $F_4 \otimes F_8$ 可以转化成如下形式。

$$
\begin{aligned}
F_4 \otimes F_8 &= \begin{bmatrix} F_8 & F_8 & F_8 & F_8 \\ F_8 & -jF_8 & jF_8 & jF_8 \\ F_8 & -F_8 & F_8 & -F_8 \\ F_8 & jF_8 & -F_8 & -jF_8 \end{bmatrix} \\
&= \begin{bmatrix} F_8 & 0 & 0 & 0 \\ 0 & F_8 & 0 & 0 \\ 0 & 0 & F_8 & 0 \\ 0 & 0 & 0 & F_8 \end{bmatrix} \begin{bmatrix} I_8 & I_8 & I_8 & I_8 \\ I_8 & -jI_8 & jI_8 & jI_8 \\ I_8 & -I_8 & I_8 & -I_8 \\ I_8 & jI_8 & -I_8 & -jI_8 \end{bmatrix} \\
&= (I_4 \otimes F_8)(F_4 \otimes I_8).
\end{aligned}
$$

为了方便下一步的说明，我们首先证明如下定理：

定理 1 $(A_1 A_2 \cdots A_K) \otimes I_n = \prod_{k=1}^{K} A_k \otimes I_n$.

证明 1 若有 $(AB) \otimes I_n = (A \otimes I_n)(B \otimes I_n)$ 成立，定理也显然成立。注意到

$$
\begin{aligned}
(A \otimes I_n)(B \otimes I_n) &= \begin{pmatrix} a_{11}I_n & \cdots & a_{1n}I_n \\ \vdots & \ddots & \vdots \\ a_{m1}I_n & \cdots & a_{mn}I_n \end{pmatrix} \begin{pmatrix} b_{11}I_n & \cdots & b_{1p}I_n \\ \vdots & \ddots & \vdots \\ b_{n1}I_n & \cdots & b_{np}I_n \end{pmatrix} \\
&= \begin{pmatrix} \sum_{k=1}^{n} a_{1k}b_{k1}I_n & \cdots & \sum_{k=1}^{n} a_{1k}b_{kp}I_n \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} a_{mk}b_{k1}I_n & \cdots & \sum_{k=1}^{n} a_{mk}b_{kp}I_n \end{pmatrix} \\
&= (AB) \otimes (I_n).
\end{aligned}
$$

因此定理成立。

类似的，我们还能证明下列定理，证明方法类似。

定理 2 $I_n \otimes (A_1 A_2 \cdots A_K) = \prod_{k=1}^{K} I_n \otimes A_k$.

接下来，我们考虑如果矩阵 A 可以作如下的稀疏矩阵分解

$$
A = A_1 A_2 \cdots A_K, \quad A_k \in M(\mathcal{P}^2) \cap M_{sparse},
$$

那么结合定理 1 和 2，再通过简单的推导，不难验证 $A \otimes I_n, I_n \otimes A$ 也能进行这种分解：

$$
A \otimes I_n = \prod_{k=1}^{K} A_k \otimes I_n, \quad A_k \otimes I_n \in M(\mathcal{P}^2) \cap M_{sparse}.
$$

$$
I_n \otimes A = \prod_{k=1}^{K} I_n \otimes A_k, \quad I_n \otimes A_k \in M(\mathcal{P}^2) \cap M_{sparse}.
$$

这就说明了如果 F_4, F_8 存在满足约束 1 和约束 2 的矩阵近似分解，结合上述的推导，我们就能证明 $F_4 \otimes F_8$ 也存在这种矩阵近似分解，即 $I_4 \otimes A_i$ $I_8 \otimes B_k$ 均是满足约束条件的矩阵：

$$\begin{aligned} F_4 \otimes F_8 &= (I_4 \otimes F_8)(F_4 \otimes I_8) \\ &= \Big(\prod_{i=1}^{M} I_4 \otimes A_i\Big)\Big(\prod_{k=1}^{N} I_8 \otimes B_k\Big). \end{aligned}$$

由问题一的结果，对于 F_4，满足约束 1 和约束 2 的这种稀疏矩阵近似分解已经找到了。对于 F_8，我们知道它的精确分解矩阵并不满足约束 2，因此我们只能退而求其次，尝试得到 F_8 在 Frobenius 范数意义下的满足约束 1 和约束 2 的近似分解，而这个分解我们在问题三中也已经解决了。

7.2 问题四模型求解与求解结果

通过 7.1 的推导，我们知道 $F = F_4 \otimes F_8$ 可以转化为如下形式

$$F_4 \otimes F_8 = (I_4 \otimes F_8)(F_4 \otimes I_8).$$

由 5.4.1 的结果(5.4)和(5.5)，不妨记

$$\begin{aligned} F_4 &= \beta_1 A_1 A_2 A_3 \\ F_8 \approx \hat{F}_8 &= \beta_2 B_1 B_2 B_3 B_4. \end{aligned}$$

我们用可分解的 $\hat{F}_8$ 代替 F_8，那么 F 就能近似分解成如下形式

$$\begin{aligned} F_4 \otimes F_8 &= (I_4 \otimes F_8)(F_4 \otimes I_8) \\ &\approx \beta_1\beta_2 \prod_{i=1}^{4}(I_4 \otimes B_i) \prod_{k=1}^{3}(A_k \otimes I_8). \end{aligned}$$

结合 7.1 中的推导，不难看出 $I_4 \otimes B_i (i = 1, 2, 3, 4)$ 和 $A_k \otimes I_8 (k = 1, 2, 3)$ 均同时满足约束 1 和约束 2. 因此该近似分解是一个同时满足约束 1 和约束 2 的分解，且具有较高的精度。考虑到 F_8 与 F_4 的分解已经在 5.4.1 的结果(5.4)和(5.5)中展示，且所占篇幅较大，因此不在本节重复展示。

最小误差和硬件复杂度及其他相关指标如表 7.1 所示（$C = q \times L = 3L$）

表 7.1　问题四结果

目标函数值	缩放因子 β	复乘次数 L	硬件复杂度 C
0.0240	0.0936	48	144

分析上表可以得出，我们提出的分解具有较高的精度，且硬件复杂度远小于问题三中32 阶 DFT 矩阵近似连乘分解后的硬件复杂度（$C = 552$）. 因此我们求得的分解结果同时具有较高的精度和较好的硬件复杂度。

8 问题五的模型建立与求解

8.1 问题五分析与数学模型

由问题一、问题二到问题三的递进中，我们发现，随着约束条件的增多，矩阵连乘拟合的最小误差增大，但方案的硬件复杂度降低。因此，我们应使得目标函数的值尽可能地接近 0.1，并在此基础上，不断地对模型的约束条件进行强化，使得方案的硬件复杂度尽可能地降低。

在问题五中需要建立如下的数学规划模型：

$$\min\ C(q, L) = q \times L$$
$$\text{s.t.} \begin{cases} \text{RMSE}(\mathcal{A}, \beta) = \frac{1}{N}\|F - \beta A_1 A_2 \cdots A_K\|_F \leq 0.1 \\ A_k \in M(\mathcal{P}^2) \cap M_{Sparse}, k = 1, 2, \cdots, K \end{cases}$$

硬件复杂度 $C = q \times L$，即硬件复杂度与分解后的矩阵 A_k 中元素的取值范围，以及矩阵的复乘次数有关。注意到，在大多数情况下，q 总是小于 L 的（尤其是 N 较大时），因此降低 q 对硬件复杂度 C 的优化程度较降低复乘次数 L 更显著。在此分析基础上，我们提出了以下的递进模型：

1. 在问题三的基础上，以 $q^* = 16$ 为约束条件，寻找最优的矩阵连乘拟合。若该矩阵连乘拟合的精度满足 RMSE < 0.1 的限制，则以 $q = q^* - 1$ 为约束条件再寻找最优的矩阵连乘拟合。重复这个过程，直到 $q = 1$，此时矩阵元素的实部和虚部均为 0 或 ± 1，硬件复杂度为 0，或在约束条件 $q = q^*$ 下最优的矩阵连乘拟合无法满足精度限制，则在约束条件 $q = q^* + 1$ 上寻找最优的矩阵连乘拟合，若满足精度限制，则停在该约束条件上；若不满足精度限制，则重复此过程。

2. 经过第一步，我们得到了约束条件 $q = q^*$，使得 $q^* = 1$ 或在 $q = q^* - 1$ 上最优的矩阵连乘拟合无法满足精度限制。设简单复数集合 $G = \{0, \pm 1, \pm j, \pm 1 \pm j\}$. 若 $q^* = 1$，硬件复杂度为 0，得到最优解；若 $q^* > 1$，将矩阵 A_k 中某个不属于 G 的元素 a_{ij} 替换成 G 中距离 a_{ij} 的元素最近的元素。重复此过程，直到 A_k 中所有元素都被替换成 G 中的元素，或矩阵连乘拟合的精度无法满足精度限制。可以看出，替换的元素越多，硬件复杂度下降的越大，因此需要构建恰当的算法以选取出最多的替换元素。

3. 经过前两步，可以认为该矩阵连乘拟合在精度限制下得到了最优的硬件复杂度。

8.2 问题五模型求解

8.2.1 最小取值范围 q^* 搜索算法

基于 8.1 的模型构建，优化矩阵连乘拟合的硬件复杂度的第一步，就是尽可能降低矩阵中元素实部和虚部的取值范围。因此，我们提出如下的最小取值范围 q^* 搜索算法：

算法 3 最小取值范围 q^* 搜索

输入: 矩阵的维数 N

输出: 最小取值范围 q^*

/* 初始时刻取 $q = 16$*/
$\beta B \leftarrow$ 对 F_N 在范围 q 下使用问题三中的方法进行连乘近似
if RMSE$(\mathcal{B}, \beta)$<=0.1 **then**
    while RMSE$(\mathcal{B}, \beta) \leq 0.1$ and $q \geq 1$ **do**
        $q \leftarrow q - 1$
        $\beta B \leftarrow$ 对 F_N 在范围 q 下使用问题三中的方法进行连乘近似
    if RMSE$(\mathcal{B}, \beta) > 0.1$ **then**
        $q \leftarrow q + 1$
else
    while RMSE$(\mathcal{B}, \beta) > 0.1$ **do**
        $q \leftarrow q + 1$
        $\beta B \leftarrow$ 对 F_N 在范围 q 下使用问题三中的方法进行连乘近似
$q^* \leftarrow q$
return q^*

8.2.2 简单复数替换的贪婪算法

在确定最小取值范围 q^* 后，需要将连乘矩阵 A_k 中的非简单复数替换为 G 中的简单复数，以达到降低复乘次数 L 的目的。对连乘矩阵中的元素 $A_k[l, m]$，找到 G 中距离它最近的点，记作 $R_k[l, m]$，并将它们两个之间的距离记为 $S_k[l, m]$.

若要复乘次数 L 被降低地尽可能大，就需要选取 A_k 中尽可能多的点，将它们替换为 G 中的简单复数。考虑到替换后矩阵连乘拟合的精度会下降，而选取 $S_k[l, m]$ 中较小值对应的点 $A_k[l, m]$ 可以使单步替换后下降的精度较小。

基于上述思路，构建简单复数替换的贪婪算法，即每一步都选取 $S_k[l, m]$ 中最小值对应的点 $A_k[l, m]$，将它替换为 G 中距离最近的简单复数，并将 $S_k[l, m]$ 删去。重复该过程，直到 $A_k[l, m]$ 中的所有元素均被替换为简单复数后停止，或某次替换后矩阵连乘拟合的精度不满足限制，保留上一步替换后的矩阵连乘拟合后停止。此时可以认为该矩阵连乘拟合

在精度限制下得到了最优的硬件复杂度。

8.3 问题五求解结果

令人意外的是，也许是因为对目标函数中缩放因子设置的修改，导致矩阵连乘拟合的结果与 F_N 在 Frobenius 范数下的距离随着维度 N 的增大而减小（已在 5.4.2 中说明，若不进行修改，无法进行问题五的求解），且当 $N=8$ 时，$F_N=F_8$ 在取值范围 $q=1$ 下的最优矩阵连乘拟合的精度满足 RMSE≤ 0.1 的限制。数值实验表明，$N=2^t(t=3,4,\cdots,10)$ 时，F_N 在取值范围 $q=1$ 下的最优矩阵连乘拟合的精度均满足 RMSE≤ 0.1 的限制，以致复乘次数和硬件复杂度均为 0，简单复数替换的贪婪算法未生效。

为完整展示我们在问题五中的模型，我们对 $N=8$, 固定 F_N 在取值范围 $q=2$ 下最优矩阵连乘拟合的缩放因子，并对其进行简单复数替换的贪婪算法，以展示算法的效果。结果如表 8.1 所示

表 8.1　问题五实验结果

	目标函数值	缩放因子 β	复乘次数 L	硬件复杂度 C
实验前	0.0479	0.1872	12	24
实验后	0.0954	0.1872	8	16

可以看出，随着简单复数替换的贪婪算法的进行，目标函数上升至接近限制 0.1，同时复乘次数由 12 下降到了 8，使得硬件复杂度下降，体现了该算法的有效性。（注：该实验仅为体现算法的有效性，并不代表本文模型的真实性能）

9 模型评价

本文对离散傅里叶变换中 DFT 矩阵以及其他相似矩阵的矩阵连乘拟合进行了深入研究，在深入理解题意的基础上，紧扣题目要求，严格满足了每一个问题中模型的约束条件，且在问题一至问题四中尽可能地提高精度，最终得到矩阵连乘拟合的结果与相应的待拟合矩阵在 Frobenius 范数意义下的距离均小于 0.05，且问题一至问题四的硬件复杂度均远小于待拟合矩阵自身的硬件复杂度，充分满足了题目中使用矩阵连乘拟合思路降低硬件复杂度的初衷。在问题五中，

9.1 模型的优点

1. 发现目标函数中缩放因子 β 的设置存在不合理之处，并对此进行修改，使得在问题五的求解中可以满足精度限制。

2. 在问题二中提出了缩放因子邻域搜索算法，改善了 β 对目标函数的优化程度。

3. 在许多问题的结论提供了解析的表达式，而非单纯地利用程序求解，使得模型推广性更强。

9.2 模型的展望

问题二至问题五的模型建立均基于问题一中 N 阶 DFT 矩阵的精确分解算法，可能会陷入局部最优。后续可以考虑与其他分解算法进行对比分析。

参考文献

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

[2] Ariyarathna V, Madanayake A, Tang X, 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, 8(3):466-479, 2018.

[3] Suarez D, Cintra R J, Bayer F M, et al., Multi-beam RF Aperture Using Multiplierless FFT Approximation, Electronics Letters, 50(24):1788-1790, 2014.

附录 A　MATLAB 源程序

A.1　第 1 问程序

the_main_1.m

```
NNN = 8;
O_FN = zeros(1,NNN);
O_FN_decom = zeros(1,NNN);

for i = 3:NNN+2
    N = 2^i;
    FN = produce_DFT(N);
    [FN_decom,N_length] = my_CT(N);

    O_FN(i-2) = calculate_On(FN,N);
    for j = 1:N_length
        O_FN_decom(i-2) = O_FN_decom(i-2)+calculate_On(FN_decom{j},
            N);
    end
end

semilogy(2.^(3:NNN+2),O_FN,'-*')
grid on
hold on
semilogy(2.^(3:NNN+2),O_FN_decom,'-*')
for i = 1:NNN
    line([2^(i+2),2^(i+2)],[O_FN(i),O_FN_decom(i)],'LineStyle','--'
        ,'Color','g');
end
xlabel('N:矩阵维数')
ylabel('C:硬件复杂度')
legend('C(F_N)','C(F_N的精确分解)')
```


produce_DFT.m

```
function result = produce_DFT(N)
result = zeros(N,N);
w = exp(((-2*pi)/N)*1i);
result(:,1) = ones(N,1);
temp = ones(N,1);
for i = 2:N
```

```
    temp(i) = w*temp(i-1);
end
result(:,2) = temp;
for i = 3:N
    result(:,i) = result(:,i-1).*temp;
end
```

my_CT.m

```
function [result,t] = my_CT(N)
w = exp(((-2*pi)/N)*1i);
t = log2(N);
result = cell(1,t+1);
P_eye = eye(N);
for i = 1:t
    temp = zeros(N,N);
    P_temp = zeros(N,N);
    Ni = N/(2^(i-1));
    Ni_2 = Ni/2;
    re_temp = zeros(Ni,Ni);
    P = zeros(Ni,Ni);
    for j = 1:Ni/2
        re_temp(j,j) = 1;
        re_temp(j,j+Ni_2) = w^(2^(i-1)*(j-1));
        P(j,2*j-1) = 1;
    end
    for j = Ni_2+1:Ni
        re_temp(j,j-Ni_2) = 1;
        re_temp(j,j) = -w^(2^(i-1)*(j-1-Ni_2));
        P(j,2*(j-Ni_2)) = 1;
    end
    for j = 1:2^(i-1)
        temp(1+Ni*(j-1):Ni*j,1+Ni*(j-1):Ni*j) = re_temp;
        P_temp(1+Ni*(j-1):Ni*j,1+Ni*(j-1):Ni*j) = P;
    end
    result{i} = temp;
    P_eye = P_temp*P_eye;
end
result{t+1} = P_eye;
t = t+1;
```

calculate_On.m

```
function OA = calculate_On(A,N)
OA = 0;
A_real = real(A); A_imag = imag(A);
for i = 1:N
    for j = 1:N
        if abs(A_real(i,j)) <= 1e-8 && abs(A_imag(i,j)) <= 1e-8
        elseif abs(A_real(i,j)) <= 1e-8 && abs(A_imag(i,j)) == 1
        elseif abs(A_real(i,j)) == 1 && abs(A_imag(i,j)) <= 1e-8
        elseif (A_real(i,j) == 1 || A_real(i,j) == -1) && (A_imag(i
            ,j) == 1 || A_imag(i,j) == -1)
        else
            OA = OA+1;
        end
    end
end
```

A.2 第 2 问程序

the_main_2_beta.m

```
ii = 0;
for beta_i = [1,0.5,0.25]
    ii = ii+1;
    subplot(3,1,ii)
    [result_beta,beta_test] = test_for_beta_2(8,beta_i);
    p1 = plot(beta_test,result_beta,'b');
    hold on
    [~,temp] = min(result_beta);
    p3 = plot(beta_test(temp),result_beta(temp),'*','Color','b');
    [result_beta,beta_test] = test_for_beta_2(16,beta_i);
    p2 = plot(beta_test,result_beta,'r');
    [~,temp] = min(result_beta);
    plot(beta_test(temp),result_beta(temp),'*','Color','r');
    grid on
    xline(beta_i,'LineStyle','--')
    legend([p1 p2 p3],'N=8','N=16','最小值')
    title(['\beta^* \in U(',sprintf('%.2f)',beta_i)])
    xlabel('\beta^*')
    ylabel('目标函数的值')
```

```
end
```

the_main_2.m

```
the_result_2 = zeros(5,6);
Ak = cell(1,5);
Ak{1} = produce_DFT(2);
Ak{2} = my_CT(4);
for i = 1:5
    the_result_2(i,1) = 2^i;
end
the_result_2(1:2,3) = [1;1];
the_result_2(1:2,4) = the_result_2(1:2,3)./sqrt(the_result_2(1:2,1)
   );
for i = 3:5
    N = the_result_2(i,1);
    test_Fn = produce_DFT(N);
    [result,N_length] = my_CT(N);
    beta_ii = 0;
    rf = zeros(1,3);
    beta_rf = zeros(1,3);
    Ak_beta = cell(1,3);
    for beta_X = [1,0.5,0.25]
        beta_ii = beta_ii+1;
        [result_beta,beta_test,Ak_beta{beta_ii}] = test_for_beta_2(
           N,beta_X);
        [rf(beta_ii),temp] = min(result_beta);
        beta_rf(beta_ii) = beta_test(temp);
    end
    [the_result_2(i,2),temp] = min(rf);
    Ak{i} = Ak_beta{temp};
    the_result_2(i,3) = beta_rf(temp);
    the_result_2(i,4) = beta_rf(temp)/sqrt(N);
end
for i = 2:5
    target = Ak{i};
    Ak_n = length(target);
    for j = 1:Ak_n
        the_result_2(i,5) = the_result_2(i,5)+calculate_On(target{j
           },the_result_2(i,1));
```

```
    end
end
the_result_2(:,6) = the_result_2(:,5)*3;
the_result_2(:,7) = the_result_2(:,2)./the_result_2(:,4);
```

test_for_beta_2.m

```
function [result_beta,beta_test,result_qs_2] = test_for_beta_2(N,
   beta)
beta_in = 0.25;
beta_left = beta*(1-beta_in);
beta_right = beta*(1+beta_in);
beta_N = 1000/beta*beta_in;
beta_test = linspace(beta_left,beta_right,beta_N);
result_beta = zeros(1,beta_N);

test_Fn = produce_DFT(N);
[result,N_length] = my_CT(N);

result_qs_2 = cell(1,4);
qs_2_end = result{N_length-2};
result_qs_2{2} = result{N_length-2};
for i = N_length-1:N_length
    result_qs_2{i-N_length+4} = result{i};
    qs_2_end = qs_2_end*result{i};
end
qs_2 = eye(N);
for i = 1:N_length-3
    qs_2 = qs_2*result{i};
end
qs_2_0 = qs_2;

ii = 0;
for k = beta_test
    k1 = k^(-1);
    ii = ii+1;
    qs_2 = qs_2_0;
    qs_2_temp = k1*qs_2;

    qs_2_real = real(qs_2_temp);
```

```
    qs_2_imag = imag(qs_2_temp);
    qs_2_real_sign = sign(qs_2_real);
    qs_2_imag_sign = sign(qs_2_imag);
    qs_2_real = qs_2_real.*qs_2_real_sign;
    qs_2_imag = qs_2_imag.*qs_2_imag_sign;
    for j = 1:N
        for i = 1:N
            if qs_2_real(i,j) < 2.5
                qs_2_real(i,j) = round(qs_2_real(i,j),0);
            else
                qs_2_real(i,j) = 2^round(log2(qs_2_real(i,j)),0);
            end
            if qs_2_imag(i,j) < 2.5
                qs_2_imag(i,j) = round(qs_2_imag(i,j),0);
            else
                qs_2_imag(i,j) = 2^round(log2(qs_2_imag(i,j)),0);
            end
        end
    end
    qs_2 = qs_2_real_sign.*qs_2_real+qs_2_imag_sign.*qs_2_imag*1i;
    qs_2 = qs_2/k1;
    test_CT_and_Fn = qs_2*qs_2_end-test_Fn;
    result_beta(ii) = norm(test_CT_and_Fn,'fro')/N/sqrt(N);
end
[~,temp] = min(result_beta);
k1 = beta_test(temp)^(-1);
qs_2 = qs_2_0;
qs_2_temp = k1*qs_2;

qs_2_real = real(qs_2_temp);
qs_2_imag = imag(qs_2_temp);
qs_2_real_sign = sign(qs_2_real);
qs_2_imag_sign = sign(qs_2_imag);
qs_2_real = qs_2_real.*qs_2_real_sign;
qs_2_imag = qs_2_imag.*qs_2_imag_sign;
for j = 1:N
    for i = 1:N
        if qs_2_real(i,j) < 2.5
            qs_2_real(i,j) = round(qs_2_real(i,j),0);
```

```
        else
            qs_2_real(i,j) = 2^round(log2(qs_2_real(i,j)),0);
        end
        if qs_2_imag(i,j) < 2.5
            qs_2_imag(i,j) = round(qs_2_imag(i,j),0);
        else
            qs_2_imag(i,j) = 2^round(log2(qs_2_imag(i,j)),0);
        end
    end
end
qs_2 = qs_2_real_sign.*qs_2_real+qs_2_imag_sign.*qs_2_imag*1i;
result_qs_2{1} = qs_2;
```

A.3 第 3 问程序

the_main_3_beta.m

```
the_result_3 = zeros(5,6);
Ak = cell(1,5);
Ak{1} = produce_DFT(2);
Ak{2} = my_CT(4);
for i = 1:5
    the_result_3(i,1) = 2^i;
end
the_result_3(1:2,3) = [1;1];
the_result_3(1:2,4) = the_result_3(1:2,3)./sqrt(the_result_3(1:2,1)
    );
for i = 3:5
    N = the_result_2(i,1);
    test_Fn = produce_DFT(N);
    [beta_for_Ak,qs_2_test] = produce_matrix_appro_3(N);
    Ak{i} = qs_2_test;
    result_3_beta = beta_quadratic(beta_for_Ak,qs_2_test,N);
    test_CT = eye(N);
    for j = 1:length(qs_2_test)
        test_CT = test_CT*qs_2_test{j};
        the_result_3(i,5) = the_result_3(i,5) + calculate_On(
            qs_2_test{j},N);
    end
    test_CT_and_Fn = result_3_beta*test_CT-test_Fn;
```

```
    rf = norm(test_CT_and_Fn,'fro')/N/sqrt(N);
    the_result_3(i,2) = rf;
    the_result_3(i,4) = result_3_beta/sqrt(N);
    the_result_3(i,3) = result_3_beta;
    the_result_3(i,6) = the_result_3(i,5)*3;
end
```

produce_matrix_appro_3.m

```
[result,N_length] = my_CT(N);
beta = [1,2,4];
beta_for_Ak = zeros(1,N_length-3);
qs_2_test = cell(1,N_length);

for i = N_length-2:N_length
    qs_2_test{i} = result{i};
end

for qs_i = 1:N_length-3
    qs_2_0 = result{qs_i};
    ii = 0;
    rrr = zeros(1,3);
    qs_2_store = cell(1,3);
    for k = [1,2,4]
        ii = ii+1;
        qs_2 = qs_2_0;
        qs_2_temp = k*qs_2;

        qs_2_real = real(qs_2_temp);
        qs_2_imag = imag(qs_2_temp);
        qs_2_real_sign = sign(qs_2_real);
        qs_2_imag_sign = sign(qs_2_imag);
        qs_2_real = qs_2_real.*qs_2_real_sign;
        qs_2_imag = qs_2_imag.*qs_2_imag_sign;
        for j = 1:N
            for i = 1:N
                if qs_2_real(i,j) < 2.5
                    qs_2_real(i,j) = round(qs_2_real(i,j),0);
                else
                    qs_2_real(i,j) = 2^round(log2(qs_2_real(i,j))
```

```
                        ,0);
                end
                if qs_2_imag(i,j) < 2.5
                    qs_2_imag(i,j) = round(qs_2_imag(i,j),0);
                else
                    qs_2_imag(i,j) = 2^round(log2(qs_2_imag(i,j))
                        ,0);
                end
            end
        end
        qs_2 = qs_2_real_sign.*qs_2_real+qs_2_imag_sign.*qs_2_imag
            *1i;
        qs_2_store{ii} = qs_2;
        qs_2 = qs_2/k;
        test_CT_and_Fn = qs_2-result{qs_i};
        rrr(ii) = norm(test_CT_and_Fn,'fro');
    end
    [~,temp] = min(rrr);
    beta_for_Ak(qs_i) = beta(temp);
    qs_2_test{qs_i} = qs_2_store{temp};
end
beta_for_Ak = [beta_for_Ak(2:1),1];
```

beta_quadratic.m

```
function result_3_beta = beta_quadratic(beta_for_Ak,qs_2_test,N)
beta_for_Ak = (beta_for_Ak).^(-1);
test_Fn = produce_DFT(N);
N_length = length(qs_2_test);
qs_3 = eye(N);
for i = 1:N_length
    qs_3 = qs_3*qs_2_test{i}*beta_for_Ak(i);
end
Re_X = real(qs_3); Im_X = imag(qs_3);
Re_F = real(test_Fn); Im_F = imag(test_Fn);

Re_X2 = Re_X.^2; Im_X2 = Im_X.^2;
Re_XF = Re_X.*Re_F; Im_XF = Im_X.*Im_F;

a = sum(sum(Re_X2 + Im_X2)); b = sum(sum(Re_XF + Im_XF));
```

```
result_3_beta = b/a;

for i = 1:N_length
    result_3_beta = result_3_beta*beta_for_Ak(i);
end
```

A.4 第 4 问程序

the_main_4.m

```
the_result_3 = zeros(5,6);
Ak = cell(1,5);
Ak{1} = produce_DFT(2);
Ak{2} = my_CT(4);
for i = 1:5
    the_result_3(i,1) = 2^i;
end
the_result_3(1:2,3) = [1;1];
the_result_3(1:2,4) = the_result_3(1:2,3)./sqrt(the_result_3(1:2,1)
    );

i = 3;
N = the_result_2(i,1);
test_Fn = produce_DFT(N);
[beta_for_Ak,qs_2_test] = produce_matrix_appro_3(N);
Ak{i} = qs_2_test;
result_3_beta = beta_quadratic(beta_for_Ak,qs_2_test,N);
test_CT = eye(N);
for j = 1:length(qs_2_test)
    test_CT = test_CT*qs_2_test{j};
    the_result_3(i,5) = the_result_3(i,5) + calculate_On(qs_2_test{
        j},N);
end
test_CT_and_Fn = result_3_beta*test_CT-test_Fn;
rf = norm(test_CT_and_Fn,'fro')/N/sqrt(N);
the_result_3(i,2) = rf;
the_result_3(i,4) = result_3_beta/sqrt(N);
the_result_3(i,3) = result_3_beta;
the_result_3(i,6) = the_result_3(i,5)*3;
```

```
F4 = produce_DFT(4);
F8 = produce_DFT(8);
F = kron(F4,F8);
F_beta = the_result_3(2,4)*the_result_3(3,4);
M1 = F_beta*eye(32);
OM = 0;
for i = 1:4
    M1 = M1*kron(eye(4),Ak{3}{i});
    OM = OM+calculate_On(kron(eye(4),Ak{3}{i}),32);
end
for i = 1:3
    M1 = M1*kron(Ak{2}{i},eye(8));
    OM = OM+calculate_On(kron(Ak{2}{i},eye(8)),32);
end
test_CT_and_Fn = M1-F/2/sqrt(8);

the_result_4 = zeros(1,4);
the_result_4(1) = norm(test_CT_and_Fn,'fro')/32;
the_result_4(2) = F_beta;
the_result_4(3) = OM;
the_result_4(4) = 3*OM;
```

A.5 第 5 问程序

search_for_q.m

```
function q_x = search_for_q(N)
q = 4;
rf = 0.01;
test_Fn = produce_DFT(N);
while (rf <= 0.1 && q > 1)
    q = q-1;
    [beta_for_Ak,qs_2_test] = produce_matrix_appro_4(N,q);
    result_3_beta = beta_quadratic(beta_for_Ak,qs_2_test,N);
    test_CT = eye(N);
    for j = 1:length(qs_2_test)
        test_CT = test_CT*qs_2_test{j};
    end
    test_CT_and_Fn = result_3_beta*test_CT-test_Fn;
    rf = norm(test_CT_and_Fn,'fro')/N/sqrt(N);
```

```
end
q_x = q;
```

replace_Ak

```
function result_4 = replace_Ak(N,qx)
test_Fn = produce_DFT(N);
[beta_for_Ak,qs_2_test] = produce_matrix_appro_4(N,qx);
N_length = length(qs_2_test);
result_3_beta = beta_quadratic(beta_for_Ak,qs_2_test,N);

result_4 = qs_2_test;
Rk = qs_2_test;
Sk = qs_2_test;
for j = 1:N_length
    target = result_4{j};
    for j1 = 1:N
        for j2 = 1:N
            if target(j1,j2) <= 1e-8
                Rk{j}(j1,j2) = 0;
                Sk{j}(j1,j2) = 10000;
            else
                Re_t = real(target(j1,j2)); Im_t = imag(target(j1,
                    j2));
                if abs(Re_t) < 0.5
                    R_R = 0;
                else
                    R_R = sign(Re_t);
                end
                if abs(Im_t) < 0.5
                    I_R = 0;
                else
                    I_R = sign(Re_t);
                end
                Sk{j}(j1,j2) = (R_R-Re_t)^2+(I_R-Im_t)^2;
                Rk{j}(j1,j2) = R_R + I_R*1i;
                if Sk{j}(j1,j2) <= 1e-8
                    Sk{j}(j1,j2) = 10000;
                end
            end
```

```
        end
    end
end
rf = 0;
for j = 1:N_length
    while (rf <= 0.1)
        [n,im] = min(Sk{j});
        [~,im2] = min(n);
        rows = im(1,im2);
        cols = im2;
        Sk{j}(rows,cols) = 10000;
        result_4{j}(rows,cols) = Rk{j}(rows,cols);
        test_CT = eye(N);
        for jjj = 1:N_length
            test_CT = test_CT*result_4{jjj};
        end
        test_CT_and_Fn = result_3_beta*test_CT-test_Fn;
        rf = norm(test_CT_and_Fn,'fro')/N/sqrt(N);
    end
end
```