


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	南京航空航天大学
参赛队号	**23102870003**
队员姓名	**1.吴日新**
	2.赵佳亮
	3.董昌龙

中国研究生创新实践系列大赛

“华为杯”第二十届中国研究生
数学建模竞赛

题　目：　　DFT 类矩阵的整数分解逼近

摘　要：

离散傅里叶变换 DFT 是一种数学变换方法，目前有关 DFT 研究中的主流方向是设计一种可以加速实现 DFT 的算法，比如快速傅里叶变换 FFT。但是随着无线通信技术的持续发展，芯片上实现 FFT 的硬件开销也随之不断增大。本文将在给定的不同约束条件以及一定精度的前提下，研究替代 FFT 的低硬件复杂度计算方法。

问题 1 是通过限制A_k的稀疏性的方式实现矩阵分解逼近。对此本文使用如下 DFT 分解式：

$$F_N=\frac{1}{\sqrt{N}}\begin{bmatrix} I & D_{\frac{N}{2}-1} \\ I & -D_{\frac{N}{2}-1} \end{bmatrix}\begin{bmatrix} F_{\frac{N}{2}} & \\ & F_{\frac{N}{2}} \end{bmatrix}[P_N]$$

当$t\leqslant 2$时，F_N矩阵已经满足约束 1（每行元素不超过两个），当$t\geqslant 3$时，本文发现可以借助 DFT 分解式及**递归**思想来获得满足约束 1 的矩阵集合$\mathcal{A}$，按照该思路计算$t=3,4,...$时的矩阵集合$\mathcal{A}$，$RMSE$与硬件复杂度C，矩阵放缩因子在不同位置的两种$RMSE$计算方式都能够得到相同的结果，即所有的$RMSE$均为 0。当$t=3,4,5$时，C分别为 64，576，3392。最后，对矩阵放缩因子进行**灵敏度分析**。

问题 2 是通过限制A_k中元素实部和虚部取值范围的方式实现矩阵整数分解逼近。对此本文依然借助在问题一的分解式及递归思想，对分解后的第一个矩阵进行替换，建立$RMSE$最小为目标函数，以替换矩阵的每个元素为决策变量，每个元素满足约束 2（属于某个固定集合）为约束条件的数学规划模型。采用启发式算法进行替换矩阵寻优，发现矩阵放缩因子乘在拟合矩阵这侧得到的结果更优，当$t=3,4,5$时，$RMSE=0.34,0.45,0.55$，$C=36,45,48$。导致拟合效果较差，即$RMSE$较高的原因是没有对元素的个数进行约束，只对矩阵元素的实部和虚部做约束时，随着t的不断增加，决策变量的规模会变得很庞大。

问题 3 是同时限制A_k的稀疏性和取值范围的方式实现矩阵整数分解逼近。对此本文需要在问题 2 的基础上同时满足约束 1，通过递归分解，只有第一个分解矩阵不满足约束，于是本文先根据A_1的矩阵特性将其拆解成UV两个矩阵相乘的形式，这时只有包含旋转因子的对角矩阵V不满足约束，首先找到一组满足约束矩阵V的**逼近矩阵**，该矩阵与V的**方向**尽量相同。接着，根据现有可用元素，找到一组放缩矩阵使得拟合的方向矩阵的**模长**尽量相同（$RMSE$尽可能小）。建立以$RMSE$最小为目标函数，分解整数个数、哪些整数进行分解为决策变量的数学规划模型。最后，提出了一种基于**双扰动**的模拟退火算法来缩短模型求解时间，结果表面当$t=3$时，整数分解的数量为 5，此时$RMSE$为 0.0038，C为 42。并且实值矩阵放缩因子与拟合矩阵相乘的目标函数方式要远高于另一种方法。

问题 4 是进一步研究对其它矩阵的分解方法。此时的新矩阵为$F_N = F_{N1} \otimes F_{N2}$。当$N_1 = 4, N_2 = 8$时，$F_N$可分解为一个形状与$F_4$非常相似的矩阵与一个主对角线上都是元素$F_8$的块对角矩阵的乘积，其中前者可直接使用 DFT 分解式拆解成三个矩阵，且这些矩阵的硬件复杂度为 0，后者中的每个子块F_8可以直接使用问题三中$t = 3$情况下的分解方法进行分解即可。通过双扰动的模拟退火算法进行求解，结果表明，当最大整数分解的数量达到 5 时，最优的$RMSE$为 0.0019、C为 168。

问题 5 是在问题 3 的基础上加上精度的**多目标**矩阵整数分解逼近模型。该问题需要先对使得$RMSE$最低的$\mathcal{A}$、β和q进行求解，将硬件复杂度最小的目标函数引入问题 3 的模型中，并增加$RMSE \leqslant 0.1$的约束。该问题中，我们的一个设计思路是，利用旋转因子的**周期性**，当考虑V_3的替换时就已经找到了一组方向相近的向量W_{16}^i替代$w_{16}^0, w_{16}^1, w_{16}^2, w_{16}^3, w_{16}^4, w_{16}^5, w_{16}^6, w_{16}^7$，这时$Y_3$中的$w_8^0, w_8^1, w_8^2, w_8^3$利用放缩性变为$w_{16}^0, w_{16}^2, w_{16}^4, w_{16}^6$，此时这些向量仍然可以使用分析$V_3$时求取的替代向量$W_{16}^0, W_{16}^2, W_{16}^4, W_{16}^6$。于是当$N = 2^t, t = 1, 2, 3, ...$时，当前$t$的优化不在需要像问题 3 一样依赖$t-1$时的最优结果，只需根据自身便可进行求解。在所有的求解方案中，选出一种在两个目标中表现都较为出色的方案，此时硬件复杂度C为 42，$RMSE$为 0.0035，此时$q = 3$，最大分解个数为 2。

关键词： DFT 矩阵分解 双扰动模拟退火算法 多目标优化模型 整数分解逼近

复数向量拟合

一、问题背景与重述

1.1 问题背景

离散傅里叶变换[5]（Discrete Fourier Transform，DFT）是一种数学变换方法，用于将一个离散时间域信号转换成其频率域表示。它是傅里叶变换的离散版本，广泛应用于信号处理、通信、图像处理、音频处理、科学计算和工程领域等多个领域。

当给定一个离散时间域信号$x_n(n=0,1,...,N-1)$，DFT 将该x_n转换成频域的复数信号X_k（$k=0,1,...,N-1$），DFT 的定义如下（其中 j 为虚数单位）：

$$X_k=\sum_{n=0}^{N-1}x_n*e^{-\frac{j2\pi nk}{N}},k=0,1,2,...,N-1 \tag{1}$$

写成矩阵形式为：

$$X=F_N x \tag{2}$$

其中x为时域信号向量$x=[x_0\ x_1\cdots x_{N-1}]^T$，$X$为经过 DFT 变换后的频域信号向量$X=[X_0\ X_1\cdots X_{N-1}]^T$，$F_N$为 DFT 矩阵，其形式如下：

$$F_N=\frac{1}{\sqrt{N}}\begin{bmatrix}1 & 1 & 1 & \cdots & 1\\ 1 & w & w^2 & \cdots & w^{N-1}\\ 1 & w^2 & w^4 & \cdots & w^{2(N-1)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & w^{N-1} & w^{2(N-1)} & \cdots & w^{(N-1)(N-1)}\end{bmatrix},w=e^{-\frac{j2\pi}{N}} \tag{3}$$

目前对于 DFT 的研究的主流方向是设计一种可以加速实现 DFT 的算法，快速傅里叶变换[6]（Fast Fourier Transform，FFT）是其中最为流行的算法，也是目前市面上的芯片中使用最多的算法。但是随着无线通信技术的持续发展，天线阵列规模、通道数以及带宽的不断增加，芯片上实现 FFT 的硬件开销也随之不断增大[1][2]。

针对这种现象，人们开始研究实现 DFT 的低复杂度计算方法。其中一种方法即通过将 DFT 矩阵分解成整数矩阵连乘的形式[3]来降低乘法器的复杂度并大幅减少乘法运算次数，与此同时，该分解方法会牺牲一定的计算精度，因此该方法可能更适合一些对精度要求不太高但需要低成本的应用领域[4]。本文将在给定的不同约束条件以及一定精度的前提下，研究替代 FFT 的低硬件复杂度计算方法。

1.2 问题重述

误差 RMSE：设计的K个矩阵$\mathcal{A}=\{A_1,A_2,...,A_K\}$与给定已知的$N$维 DFT 矩阵$F_N$之间的误差越小，即实值矩阵缩放因子$\beta$与$F_N$相乘得到的矩阵$\beta F_N$和设计的$K$个矩阵相乘的结果$A_1A_2\cdots A_K$在 Frobenius 范数意义下尽可能接近，用数学公式表示为：

$$\min_{\mathcal{A},\beta}RMSE(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\|\beta F_N-A_1A_2\cdots A_K\|_F^{\ 2}} \tag{4}$$

计算复杂度C：只考虑乘法器的硬件复杂度：

$$C=q\times L \tag{5}$$

其中，q指代A_K中元素的取值范围。我们限制A_K中元素实部和虚部的取值范围为$\mathcal{P}=\{0,\pm1,\pm2,...,\pm2^{q-1}\}$。$L$表示复数乘法的次数，其中与$0$、$\pm1$、$\pm j$或$(\pm1\pm j)$相乘时不计入复数乘法次数。

考虑以下两种约束条件：

约束 1：限定中每个矩阵的每行至多只有 2 个非零元素。

约束 2：限定中每个矩阵满足以下要求：

$$A_k[l,m] \in \{x+jy | x,y \in \mathcal{P}\}$$
$$\mathcal{P}=\{0,\pm 1,\pm 2,...,\pm 2^{q-1}\}, k=1,2,...,K;\ l,m=1,2,...,N \tag{6}$$

其中，$A_K[l,m]$表示矩阵A_k第l行第m列的元素。

问题 1：通过限制A_k的稀疏性的方式实现矩阵分解逼近。对于$N=2^t, t=1,2,3,...$的 DFT 矩阵F_N，请在满足约束 1、$q=16$的条件下，对计算$RMSE$中的变量$\mathcal{A}$和β进行优化，并计算最小误差和方法的硬件复杂度C。

问题 2：讨论通过限制A_k中元素实部和虚部取值范围的方式实现矩阵整数分解逼近。对于$N=2^t, t=1,2,3,4,5$的 DFT 矩阵F_N，请在满足约束 2、$q=3$的条件下，对$\mathcal{A}$和β进行优化，并计算最小误差和方法的硬件复杂度C。

问题 3：同时限制A_k的稀疏性和取值范围的方式实现矩阵整数分解逼近。对于$N=2^t, t=1,2,3,4,5$的 DFT 矩阵F_N，请在同时满足约束 1 和 2 以及$q=3$的条件下，对$\mathcal{A}$和β进行优化，并计算最小误差和方法的硬件复杂度C。

问题 4：进一步研究对其它矩阵的分解方法。考虑矩阵$F_N=F_{N1}\otimes F_{N2}$，其中F_{N1}和F_{N2}分别是N_1和N_2维的 DFT 矩阵，$\otimes$表示 Kronecker 积。当$N_1=4, N_2=8$时，请在同时满足约束 1 和 **2** 以及$q=3$的条件下，对$\mathcal{A}$和β进行优化，并计算最小误差和方法的硬件复杂度C。

问题 5：在问题 3 的基础上加上精度的多目标矩阵整数分解逼近模型。要求将精度限制在 0.1 以内，即$RMSE \leqslant 0.1$。对于$N=2^t, t=1,2,3,...$的 DFT 矩阵F_N，请在同时满足约束 1 和 2 的条件下，寻找满足精度要求的q，并对$\mathcal{A}$和β进行优化，最后计算方法的硬件复杂度C。

二、符号说明

数学符号	解释说明
F_N	N 维 DFT 矩阵
j	虚数单位
K	设计的矩阵数量
A_k	设计的第k个矩阵（$k=1,2,...,K$）
$\mathcal{A}$	设计的K个矩阵的集合
β	实值矩阵缩放因子
q	限制A_k中元素实部和虚部的取值范围
$\mathcal{P}$	表示A_k中元素实部和虚部的取值范围的集合
L	复数乘法的次数
C	硬件复杂度
$A_K[l,m]$	矩阵A_k第l行第m列的元素

三、模型建立前的准备

3.1、β存在可求且唯一

β是一个可根据不同约束条件来设置的实值矩阵缩放因子，但这些约束条件只是用于限制我们如何设计多个分解F_N的矩阵中的元素，该过程与β毫无关联。在我们获得一个给定的$\mathcal{A}$之后，此时的目标函数$RMSE(\mathcal{A},\beta)$只存在一个自变量β，在研究该目标函数时，当β在F_N前面时，我们发现该目标函数始终存在一个使其取得最小值的β_1（当β在A_k前面时则为β_2），且给定的$\mathcal{A}$在该β_1的条件下计算得到的误差$RMSE(\mathcal{A},\beta)$更具有比较性和说服力。于是我们希望进一步找到β_1的一般表达式，对此给出下面的定理 1 及详细的证明过程。

定理 1：给定$\mathcal{A}$，使得目标函数$RMSE(\mathcal{A},\beta)=\frac{1}{N}\sqrt{\|\beta F_N-A_1A_2\cdots A_K\|_F{}^2}$取得最小值的$\beta$是可求的，且$\beta$是唯一的。

证明：（1）当β在F_N前面时，求使得$RMSE(\mathcal{A},\beta)$取最小值时的β_1，其实就是求使$\sqrt{\|\beta F_N-A_1A_2\cdots A_K\|_F{}^2}$（矩阵$\beta F_N-A_1A_2\cdots A_K$的 Frobenius 范数）取最小值时的$\beta$，由于$F_N$和$\mathcal{A}=\{A_1,A_2,...,A_K\}$中每个位置上的元素都是已知的，不妨设：

$$F_N=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix}_{N\times N} \tag{7}$$

$$A_1A_2\cdots A_K=\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1N} \\ b_{21} & b_{22} & \cdots & b_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \cdots & b_{NN} \end{bmatrix}_{N\times N} \tag{8}$$

此时：

$$\begin{aligned}\sqrt{\|\beta F_N-A_1A_2\cdots A_K\|_F{}^2}&=\sqrt{\left\|\begin{matrix} a_{11}\beta-b_{11} & a_{12}\beta-b_{12} & \cdots & a_{1N}\beta-b_{1N} \\ a_{21}\beta-b_{21} & a_{22}\beta-b_{22} & \cdots & a_{2N}\beta-b_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1}\beta-b_{N1} & a_{N2}\beta-b_{N2} & \cdots & a_{NN}\beta-b_{NN} \end{matrix}\right\|_F{}^2} \\ &=\sqrt{\sum_{l=1}^{N}\sum_{m=1}^{N}|a_{lm}\beta-b_{lm}|^2}\end{aligned} \tag{9}$$

令：

$$f(\beta)=\sqrt{\sum_{l=1}^{N}\sum_{m=1}^{N}|a_{lm}\beta-b_{lm}|^2} \tag{10}$$

则：

$$
\begin{aligned}
f'(\beta) &= \frac{1}{2}\frac{\sum_{l=1}^{N}\sum_{m=1}^{N}2a_{lm}|a_{lm}\beta-b_{lm}|}{\sqrt{\sum_{l=1}^{N}\sum_{m=1}^{N}|a_{lm}\beta-b_{lm}|^{2}}} \\
&= \frac{\sum_{l=1}^{N}\sum_{m=1}^{N}a_{lm}|a_{lm}\beta-b_{lm}|}{\sqrt{\sum_{l=1}^{N}\sum_{m=1}^{N}|a_{lm}\beta-b_{lm}|^{2}}}
\end{aligned}
\tag{11}
$$

显然，存在唯一$\beta_1=\dfrac{\sum_{l=1}^{N}\sum_{m=1}^{N}a_{lm}b_{lm}}{\sum_{l=1}^{N}\sum_{m=1}^{N}{a_{lm}}^{2}}$，使得$f'(\beta_1)=0$，并且由于${a_{ij}}^{2}$恒大于等于 0，当$\beta<\beta_1$时，$f'(\beta_1)<0$；当$\beta>\beta_1$时，$f'(\beta_1)>0$。因此使得目标$RMSE(\mathcal{A},\beta)$取最小值时的$\beta=\beta_1$唯一，且$\beta_1=\dfrac{\sum_{l=1}^{N}\sum_{m=1}^{N}a_{lm}b_{lm}}{\sum_{l=1}^{N}\sum_{m=1}^{N}{a_{lm}}^{2}}$。

（2）当β在A_k前面时，同理可证，存在唯一$\beta_2=\dfrac{\sum_{l=1}^{N}\sum_{m=1}^{N}{a_{lm}}^{2}}{\sum_{l=1}^{N}\sum_{m=1}^{N}a_{lm}b_{lm}}$，使得$f'(\beta_2)=0$，并且由于${a_{ij}}^{2}$恒大于等于 0，当$\beta<\beta_1$时，$f'(\beta_1)<0$；当$\beta>\beta_2$时，$f'(\beta_2)>0$。因此使得目标$RMSE(\mathcal{A},\beta)$取最小值时的$\beta=\beta_2$唯一，且$\beta_2=\dfrac{\sum_{l=1}^{N}\sum_{m=1}^{N}{a_{lm}}^{2}}{\sum_{l=1}^{N}\sum_{m=1}^{N}a_{lm}b_{lm}}$。

四、问题分析与模型建立求解

4.1 问题一

4.1.1 问题一的分析

可以将F_N拆解成三个N维矩阵X_1、Y_1、Z_1相乘的形式：

$$
F_N=\frac{1}{\sqrt{N}}\underbrace{\begin{bmatrix} I & D_{\frac{N}{2}-1} \\ I & -D_{\frac{N}{2}-1} \end{bmatrix}}_{X_1}\underbrace{\begin{bmatrix} F_{\frac{N}{2}} & \\ & F_{\frac{N}{2}} \end{bmatrix}}_{Y_1}\underbrace{\begin{bmatrix} P_N \end{bmatrix}}_{Z_1}
\tag{12}
$$

其中的P_N为奇偶置换矩阵，$D_{\frac{N}{2}-1}$为：

$$D_{\frac{N}{2}-1}=\begin{bmatrix}1 & & & \\ & w & & \\ & & \ddots & \\ & & & w^{\frac{N}{2}-1}\end{bmatrix} \tag{13}$$

式（12）成立的证明过程如下：

$$\frac{1}{\sqrt{N}}\begin{bmatrix}I & D_{\frac{N}{2}-1}\\ I & -D_{\frac{N}{2}-1}\end{bmatrix}\begin{bmatrix}F_{\frac{N}{2}} & 0\\ 0 & F_{\frac{N}{2}}\end{bmatrix}[P_N]$$

$$=\frac{1}{\sqrt{N}}\left[\begin{array}{cccc|cccc}1&0&\cdots&0&1&0&\cdots&0\\0&1&\cdots&0&0&\omega_N^1&\cdots&0\\\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&\cdots&1&0&0&\cdots&\omega_N^{\frac{N}{2}-1}\\\hline1&0&\cdots&0&-1&0&0&0\\0&1&\cdots&0&0&-\omega_N^1&\cdots&0\\\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&\cdots&1&0&0&\cdots&-\omega_N^{\frac{N}{2}-1}\end{array}\right]\left[\begin{array}{cccc|cccc}1&1&\cdots&1&0&0&\cdots&0\\1&\omega_{\frac{N}{2}}^1&\cdots&\omega_{\frac{N}{2}}^{\frac{N}{2}-1}&0&0&\cdots&0\\\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&\vdots\\1&\omega_{\frac{N}{2}}^{\frac{N}{2}-1}&\cdots&\omega_{\frac{N}{2}}^{\left(\frac{N}{2}-1\right)^2}&0&0&\cdots&0\\\hline0&0&\cdots&0&1&1&\cdots&1\\0&0&\cdots&0&1&\omega_{\frac{N}{2}}^1&\cdots&\omega_{\frac{N}{2}}^{\frac{N}{2}-1}\\\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&\vdots\\0&0&\cdots&0&1&\omega_{\frac{N}{2}}^{\frac{N}{2}-1}&\cdots&\omega_{\frac{N}{2}}^{\left(\frac{N}{2}-1\right)^2}\end{array}\right]\begin{bmatrix}1&0&0&0&0&0&\cdots&0\\0&0&1&0&0&0&\cdots&0\\\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\0&0&0&0&0&\cdots&1&0\\0&1&0&0&0&0&\cdots&0\\0&0&0&1&0&0&\cdots&0\\\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\0&0&0&0&0&0&\cdots&1\end{bmatrix}$$

$$=\frac{1}{\sqrt{N}}\left[\begin{array}{cccc|cccc}1&1&\cdots&1&1&1&\cdots&1\\1&\omega_{\frac{N}{2}}^1&\cdots&\omega_{\frac{N}{2}}^{\frac{N}{2}-1}&\omega_N^1&\omega_N^1\omega_{\frac{N}{2}}^1&\cdots&\omega_N^1\omega_{\frac{N}{2}}^{\frac{N}{2}-1}\\\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&\vdots\\1&\omega_{\frac{N}{2}}^{\frac{N}{2}-1}&\cdots&\omega_{\frac{N}{2}}^{\left(\frac{N}{2}-1\right)^2}&\omega_N^{\frac{N}{2}-1}&\omega_N^{\frac{N}{2}-1}\omega_{\frac{N}{2}}^{\frac{N}{2}-1}&\cdots&\omega_N^{\frac{N}{2}-1}\omega_{\frac{N}{2}}^{\left(\frac{N}{2}-1\right)^2}\\\hline1&1&\cdots&1&-1&-1&\cdots&-1\\1&\omega_{\frac{N}{2}}^1&\cdots&\omega_{\frac{N}{2}}^{\frac{N}{2}-1}&-\omega_N^1&-\omega_N^1\omega_{\frac{N}{2}}^1&\cdots&-\omega_N^1\omega_{\frac{N}{2}}^{\frac{N}{2}-1}\\\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&\vdots\\1&\omega_{\frac{N}{2}}^{\frac{N}{2}-1}&\cdots&\omega_{\frac{N}{2}}^{\frac{\left(\frac{N}{2}-1\right)^2}{2}}&-\omega_N^{\frac{N}{2}-1}&-\omega_N^{\frac{N}{2}-1}\omega_{\frac{N}{2}}^{\frac{N}{2}-1}&\cdots&-\omega_N^{\frac{N}{2}-1}\omega_{\frac{N}{2}}^{\left(\frac{N}{2}-1\right)^2}\end{array}\right]\begin{bmatrix}1&0&0&0&0&0&\cdots&0\\0&0&1&0&0&0&\cdots&0\\\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\0&0&0&0&0&\cdots&1&0\\0&1&0&0&0&0&\cdots&0\\0&0&0&1&0&0&\cdots&0\\\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\0&0&0&0&0&0&\cdots&1\end{bmatrix}$$

$$=\frac{1}{\sqrt{N}}\left[\begin{array}{cccc|cccc}1&1&\cdots&1&1&1&\cdots&1\\1&\omega_N^2&\cdots&\omega_N^{N-2}&\omega_N^1&\omega_N^3&\cdots&\omega_N^{N-1}\\\cdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&\vdots\\1&\omega_N^{N-2}&\cdots&\omega_N^{(N-2)\left(\frac{N}{2}-1\right)}&\omega_N^{\frac{N}{2}-1}&\omega_N^{\frac{3N}{2}-3}&\cdots&\omega_N^{(N-1)\left(\frac{N}{2}-1\right)}\\\hline1&1&\cdots&1&-1&-1&\cdots&-1\\1&\omega_N^2&\cdots&\omega_N^{N-2}&-\omega_N^1&-\omega_N^3&\cdots&-\omega_N^{N-1}\\\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&\vdots\\1&\omega_N^{N-2}&\cdots&\omega_N^{(N-2)\left(\frac{N}{2}-1\right)}&-\omega_N^{\frac{N}{2}-1}&-\omega_N^{\frac{3N}{2}-3}&\cdots&-\omega_N^{(N-1)\left(\frac{N}{2}-1\right)}\end{array}\right]\begin{bmatrix}1&0&0&0&0&0&\cdots&0\\0&0&1&0&0&0&\cdots&0\\\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\0&0&0&0&0&\cdots&1&0\\0&1&0&0&0&0&\cdots&0\\0&0&0&1&0&0&\cdots&0\\\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots\\0&0&0&0&0&0&\cdots&1\end{bmatrix}$$

$$=\frac{1}{\sqrt{N}}\begin{bmatrix}1&1&1&\cdots&1\\1&\omega_N^1&\omega_N^2&\cdots&\omega_N^{N-1}\\1&\omega_N^2&\omega_N^4&\cdots&\omega_N^{2(N-1)}\\\vdots&\vdots&\vdots&\ddots&\vdots\\1&\omega_N^{N-1}&\omega_N^{2(N-1)}&\cdots&\omega_N^{(N-1)^2}\end{bmatrix}=F_N$$

将矩阵Y中的$F_{\frac{N}{2}}$继续拆解成三个$\frac{N}{2}$维矩阵X_2、Y_2、Z_2相乘的形式：

$$F_{\frac{N}{2}}=\frac{1}{\sqrt{\frac{N}{2}}}\underbrace{\begin{bmatrix} I & D_{\frac{N}{4}-1} \\ I & -D_{\frac{N}{4}-1} \end{bmatrix}}_{X_2}\underbrace{\begin{bmatrix} F_{\frac{N}{4}} & \\ & F_{\frac{N}{4}} \end{bmatrix}}_{Y_2}\underbrace{\begin{bmatrix} P_{\frac{N}{2}} \end{bmatrix}}_{Z_2} \tag{14}$$

因此F_N为：

$$F_N=\frac{\sqrt{2}}{N}[X_1]\begin{bmatrix} X_2 & \\ & X_2 \end{bmatrix}\begin{bmatrix} F_{\frac{N}{4}} & & & \\ & F_{\frac{N}{4}} & & \\ & & F_{\frac{N}{4}} & \\ & & & F_{\frac{N}{4}} \end{bmatrix}\begin{bmatrix} Z_2 & \\ & Z_2 \end{bmatrix}[Z_1] \tag{15}$$

于是经过log_2^N-1次分解后，F_N可看作块 1、块 2、块 3 这三块矩阵的乘积（为了公式的美观，于是将这三块换行表示）：

$$\begin{aligned} F_N=\frac{\sqrt{2}}{N}&\underbrace{[X_1]\begin{bmatrix} X_2 & \\ & X_2 \end{bmatrix}\cdots\begin{bmatrix} X_{log_2^N-1} & & \\ & \ddots & \\ & & X_{log_2^N-1} \end{bmatrix}}_{\text{块1}} \\ &\underbrace{\begin{bmatrix} F_2 & & \\ & \ddots & \\ & & F_2 \end{bmatrix}}_{\text{块2}} \\ &\underbrace{\begin{bmatrix} Z_{log_2^N-1} & & \\ & \ddots & \\ & & Z_{log_2^N-1} \end{bmatrix}\cdots\begin{bmatrix} Z_2 & \\ & Z_2 \end{bmatrix}[Z_1]}_{\text{块3}} \end{aligned} \tag{16}$$

首先对于块 1，其中每个矩阵可以看作包含四个块对角矩阵的分块矩阵，如下所示：

$$\left[\begin{array}{ccc:ccc} 1 & & & 1 & & \\ & 1 & & & w & \\ & & \ddots & & & \ddots \\ & & 1 & & & w^{\frac{N}{2}-1} \\ \hdashline 1 & & & 1 & & \\ & 1 & & & w & \\ & & \ddots & & & \ddots \\ & & 1 & & & w^{\frac{N}{2}-1} \end{array}\right] \tag{17}$$

故根据分块矩阵的乘法规则，块 1 中各个矩阵经过乘积后最终得到的矩阵一定也是由四个块对角矩阵组成的分块矩阵，所以块 1 显然满足约束 1。

其次对于块 2，该分块对角矩阵由$\frac{N}{2}$个$F_2=\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$组成，所以块 2 满足约束 1。

最后对于块 3，该块中的每个分块对角矩阵都是由奇偶置换矩阵组成的，所以块 3 的

每个矩阵都可以看作一个单位矩阵经过有限次初等变化生成的矩阵，故这类矩阵可拆分成有限个初等矩阵的乘积，因此块 3 中各个矩阵乘积后得到的矩阵一定是一个单位矩阵经过有限次初等变化生成的矩阵，所以块 3 显然满足约束 1。

4.1.2 问题一的求解与结果分析

算法 1：DFT 矩阵分解算法

```
Function DFT_decomposition(N):
01.  n = length(DFT)
02.  if n == 2:                                  //可分解的最小 DFT 矩阵
03.       x = [1 1; 1 -1]                        //分解的第一个矩阵最小单元
04.       X = diag(x,...,x)                      //第一个矩阵，size: N*N
05.       y = [1 1;1 -1]                         //分解的第二个矩阵最小单元
06.       Y = diag(x,...,x)                      //第二个矩阵，size: N*N
07.       z = [1 0; 0 1]                         //分解的第三个矩阵最小单元
08.       Z = diag(y,...,y)                      //第三个矩阵，size: N*N
09.       return {X, Y, Z}
10. else:
11.      w = e^(-j2π/n)
12.      D = diag(w^0,w^1,....,w^n-1)
13.      x = [I D; I -D]
14.      X = diag(x,...,x)
15.      Y = DFT_decomposition(n/2)              //递归调用
16.      z = (e1,e3,e5,....,e2,....,en)          //排列矩阵
17.      Z = diag(z,...,z)
18. end if
19. return {X, Y, Z}
```

通过算法 1 分别求取$t=1,2,3...$时的分解情况如下：

- $t=1, N=2$

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

- $t=2, N=4$

$$F_4=\frac{1}{2}\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & -j\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & j\end{bmatrix}\begin{bmatrix}1 & 0 & 1 & 0\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & 1\\ 0 & 1 & 0 & -1\end{bmatrix}$$

- $t=3, N=8$

$$F_8 \approx \frac{1}{4\sqrt{2}}\begin{bmatrix} 1 & & & 1 & & & & \\ & 1 & & & 0.7-0.7j & & & \\ & & 1 & & & & -j & \\ & & & 1 & & & & -0.7-0.7j \\ 1 & & & -1 & & & & \\ & 1 & & & -0.7+0.7j & & & \\ & & 1 & & & & j & \\ & & & 1 & & & & 0.7+0.7j \end{bmatrix} \begin{bmatrix} 1 & & 1 & & & & & \\ & 1 & & & & & & \\ 1 & & -1 & & & & & \\ & 1 & & & & & & \\ & & & & 1 & & 1 & \\ & & & & & 1 & & \\ & & & & 1 & & -1 & \\ & & & & & 1 & & \end{bmatrix}$$

$$\begin{bmatrix} 1 & & & & 1 & & & \\ 1 & & & & -1 & & & \\ & & 1 & & & & 1 & \\ & & 1 & & & & -1 & \\ & 1 & & & & 1 & & \\ & 1 & & & & -1 & & \\ & & & 1 & & & & 1 \\ & & & 1 & & & & -1 \end{bmatrix}$$

算法 2：分解矩阵硬件复杂度计算算法

Input: 分解的矩阵集合 A，矩阵的维度 N，
Output: 硬件复杂度
Initialize: 当前最大的 q 值 max_q=0，复数乘法次数 L=0

```
// 计算矩阵 A 的 max_q 值
for Ak in range(A):                                  //对 A 中所有矩阵进行遍历
    for a in range(Ak):                              //对矩阵 Ak 中所有元素进行遍历
        max_q = max(max_q, |real(a)|, |imag(a)|)     //元素 a 的实部和虚部
    end for
end for
// 计算矩阵 A 的 L 值
for Ak in range(A)-1:                                //遍历到倒数第二个矩阵
    for a in row(Ak):                                //对 Ak 所有行元素遍历
      for b in col(Ak+1):                            //对 Ak+1 所有列元素遍历
        If (a,b 的实部或虚部为整数且不超过 1):
            L = L+1;                                 //L 自增
        end if
      end for
    end for
    Ak+1 = Ak*Ak+1;                                  //更新 Ak+1 矩阵
```

并且不管按照哪种β的计算方式，总是能得到相同的F_N分解结果，下表给当 t=1 到 8，两种不同目标函数计算方式得到的β，$RMSE$以及硬件复杂度，其中硬件复杂度按照算法 2 的步骤进行计算，q固定为 16。

表 1 问题一的结果

t	1	2	3	4	5	6	7	8
β_1	1.414	2	2.828	4	5.659	8	11.314	16
$RMSE_1$	0	5.30e-17	3.56e-16	1.53e-16	2.28e-16	7.07e-16	3.85e-14	4.46e-15
β_2	0.707	0.5	0.354	0.25	0.177	0.125	0.088	0.0625
$RMSE_2$	0	2.65e-17	1.41e-16	3.82e-17	3.72e-17	8.83e-17	3.41e-15	2.91e-16
C	0	0	64	576	3392	16704	75072	320832
K	1	3	9	17	33	65	129	257

结果分析：β_1,β_2的区别只是在于β在$RMSE$式子中的位置有所不同，会作为目标函数影响F_N的分解，但是为了满足约束 1，于是对分解中不满足该约束的矩阵进行进一步的近似寻找替代矩阵，该过程会导致部分精度的丢失，但从上表中各个β所对应的$RMSE$数值可以看出，利用β_1,β_2拆分的$\mathcal{A}$都能够完美表示F_N，其中$RMSE_2$比$RMSE_1$要稍好一些。当 q 固定为 16 时，此时硬件复杂度较高，主要原因为优化目标保证了F_N的极高精度，并未对硬件复杂度进行优化。

参数β的检验与灵敏度分析，对比两个目标函数在不同参数β下，$RMSE$的变化过程，分别在$t=1,2,3,5$下进行测试，从图中可以明显的看出，最优β是正确的，而且随着β的变化，对于目标函数$RMSE$的影响是稳定线性的。

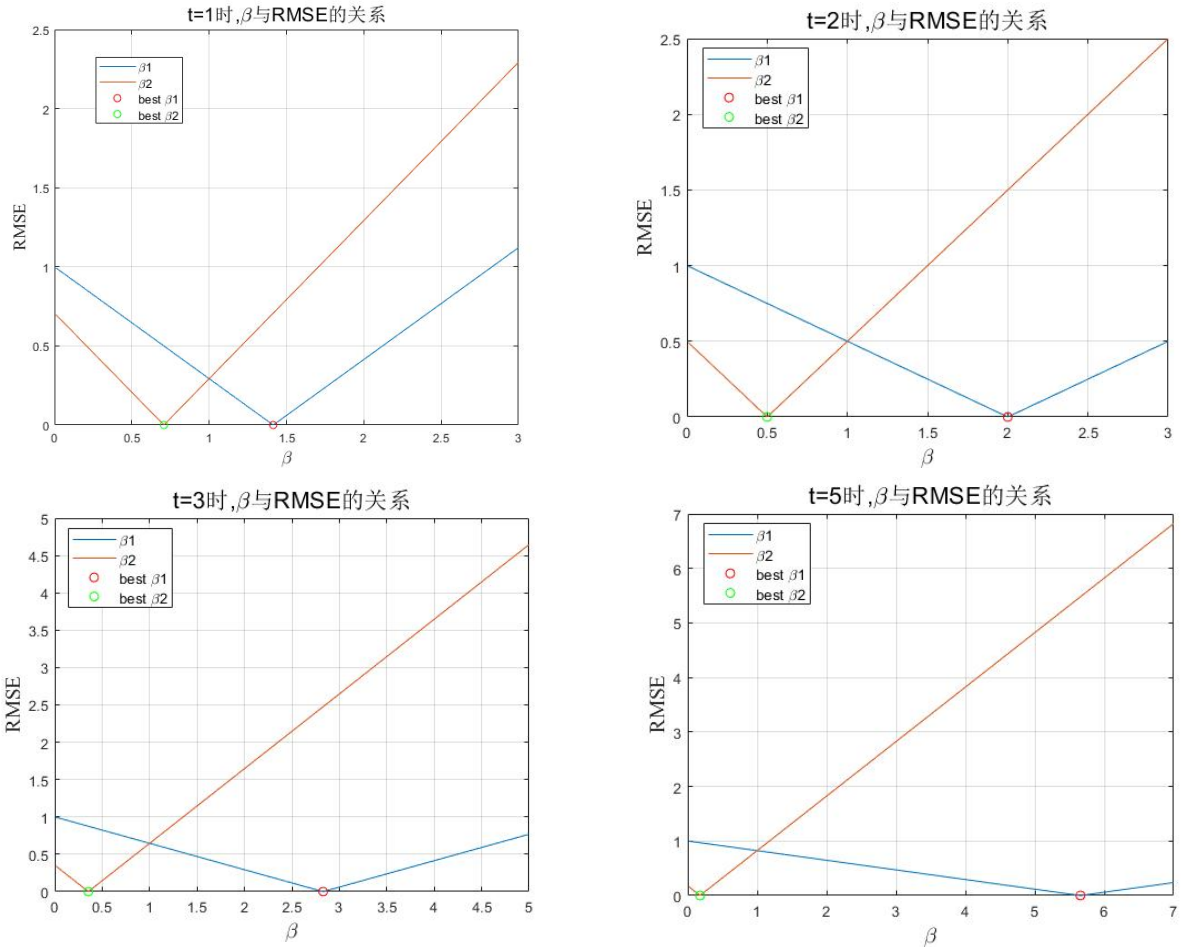

图 1 t=1,2,3,5 时，β与 $RMSE$ 的关系

4.2 问题二

4.2.1 问题二的分析

问题二是仅在约束 2 且$q=3$的条件下讨论通过限制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的方案，此时$\mathcal{P}=\{0,\pm1,\pm2,\pm4\}$，后续为详细的分类讨论过程：

- $t=1,N=2$

  此时$F_2=\frac{1}{\sqrt{2}}\begin{bmatrix}1&1\\1&-1\end{bmatrix}$，很显然满足约束 2，且$RMSE=0,C=0$。

- $t=2,N=4$

  此时$F_4=\frac{1}{2}\begin{bmatrix}1&1&1&1\\1&-j&-1&j\\1&-1&1&-1\\1&j&-1&-j\end{bmatrix}$，根据问题一的求解，可以得知，$F_4$可以被分解成$X_1$、$Y_1$、$Z_1$三个矩阵：

$$F_4=\frac{1}{2}\underbrace{\begin{bmatrix}1&0&1&0\\0&1&0&-j\\1&0&-1&0\\0&1&0&j\end{bmatrix}}_{X_1}\underbrace{\begin{bmatrix}1&1&0&0\\1&-1&0&0\\0&0&1&1\\0&0&1&-1\end{bmatrix}}_{Y_1}\underbrace{\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}}_{Z_1}$$

$$F_4=\frac{1}{2}X_1Y_1Z_1$$

可以显然看出X_1,Y_1,Z_1这三个矩阵都满足约束 2，且$RMSE=0,C=0$。

- $t=3,N=8$

对F_8采用问题一中的分解方法，可将F_8分解成X_2,Y_2,Z_2三个矩阵：

$$F_8=\frac{1}{4}\underbrace{\begin{bmatrix}I&D_3\\I&-D_3\end{bmatrix}}_{X_2}\underbrace{\begin{bmatrix}F_4&\\&F_4\end{bmatrix}}_{Y_2}\underbrace{[P_8]}_{Z_2}$$

$$F_8=\frac{1}{4}X_2Y_2Z_2$$

$$D_3=\begin{bmatrix}1&&&\\&w_8^1&&\\&&w_8^2&\\&&&w_8^3\end{bmatrix}=\begin{bmatrix}1&&&\\&0.7071-0.7071j&&\\&&-j&\\&&&-0.7071-0.7071j\end{bmatrix}$$

其中Y_2矩阵可借助问题一的分解策略及$t=2,N=4$的分解结果$F_4=X_1Y_1Z_1$进一步分解为：

$$\begin{bmatrix}F_4&\\&F_4\end{bmatrix}=\frac{1}{2}\begin{bmatrix}X_1&\\&X_1\end{bmatrix}\begin{bmatrix}Y_1&\\&Y_1\end{bmatrix}\begin{bmatrix}Z_1&\\&Z_1\end{bmatrix}$$

所以F_8又可以表示为：

$$F_8=\frac{1}{4\sqrt{2}}\underbrace{\begin{bmatrix}I & D_3\\ I & -D_3\end{bmatrix}}_{X_2}\underbrace{\begin{bmatrix}X_1 & \\ & X_1\end{bmatrix}\begin{bmatrix}Y_1 & \\ & Y_1\end{bmatrix}\begin{bmatrix}Z_1 & \\ & Z_1\end{bmatrix}}_{Y_2}\underbrace{[P_8]}_{Z_2}$$

由于X_1、Y_1、Z_1都满足约束 2，因此，Y_2,Z_2中的各个矩阵也都满足约束 2，并且Y_2中三个矩阵的C为 0，此时就只有X_2不满足约束 2。为了使X_2可以满足约束 2，需要对该矩阵进行近似（或者替换等等），即需要找到一个满足约束 2 的矩阵X_2'去对原始矩阵X_2进行近似，进而可使得F_8分解所得的所有矩阵都满足约束 2，同时在该过程中尽可能的满足精度最大（误差最小）以及复杂度最低的目标。很明显，该问题为优化问题。

4.2.2 问题二的建模

该问题就是要确定原始矩阵X_2的代替矩阵X_2'，假设X_2'中第l行第m列的元素为$a_{lm}+jb_{lm}$，其中a_{lm}、b_{lm}分别代表该元素的实部和虚部，那么X_2'可以表示为：

$$X_2'=\begin{bmatrix}a_{11}+jb_{11} & a_{12}+jb_{12} & \cdots & a_{18}+jb_{18}\\ a_{21}+jb_{21} & a_{22}+jb_{22} & \cdots & a_{28}+jb_{28}\\ \vdots & \vdots & \ddots & \vdots\\ a_{81}+jb_{81} & a_{82}+jb_{82} & \cdots & a_{88}+jb_{88}\end{bmatrix}$$

目标函数：$\min RMSE$。

决策变量：代替矩阵X_2'中每个元素的取值。

约束 1：X_2'中的每个元素实部和虚部取值范围：

$$a_{lm},b_{lm}\in\mathcal{P},\mathcal{P}=\{0,\pm1,\pm2,\pm4\}$$

- $t=4,t=5$

这两种情况的分析与$t=3$的分析十分相似，如$t=4$时的$F_{16}\approx\frac{1}{16\sqrt{2}}X_3Y_3Z_3$，其中$Z_3$作为排列矩阵一定满足约束 2，$Y_3$可以直接使用$t=2$情况下的分解结果，即$Y_3=\begin{bmatrix}X_2' & \\ & X_2'\end{bmatrix}\begin{bmatrix}Y_2 & \\ & Y_2\end{bmatrix}\begin{bmatrix}Z_2 & \\ & Z_2\end{bmatrix}$，显然$Y_3$分解的这三个矩阵都满足约束 2，所以此时只需找到一个满足约束 2 的矩阵X_3'去对矩阵X_3进行近似即可，于是问题又回归到如何寻找这个代替矩阵，对此就可以直接借助$t=3$分析中提到的算法 1、2。$t=5$情况下的分析也是如此。为了更加清晰的了解我们的设计思想，故将整体的思路整理如下：

$$t=2,N=4：F_4=\frac{1}{2}X_1\begin{bmatrix}F_2 & \\ & F_2\end{bmatrix}Z_1$$

$$=\frac{1}{2}X_1Y_1Z_1$$

$$t=3,N=8：F_8=\frac{1}{2\sqrt{2}}X_2\begin{bmatrix}F_4 & \\ & F_4\end{bmatrix}Z_2$$

$$=\frac{1}{4\sqrt{2}}X_2\begin{bmatrix}X_1 & \\ & X_1\end{bmatrix}\begin{bmatrix}Y_1 & \\ & Y_1\end{bmatrix}\begin{bmatrix}Z_1 & \\ & Z_1\end{bmatrix}Z_2$$

$$=\frac{1}{4\sqrt{2}}X_2Y_2Z_2$$

$$\approx\frac{1}{4\sqrt{2}}X_2'Y_2Z_2\text{（}X_2'\text{为}X_2\text{的代替矩阵）}$$

$$t=4,N=16：F_{16}=\frac{1}{4}X_3\begin{bmatrix}F_8 & \\ & F_8\end{bmatrix}Z_3$$

$$\approx\frac{1}{16\sqrt{2}}X_3\begin{bmatrix}X_2' & \\ & X_2'\end{bmatrix}\begin{bmatrix}Y_2 & \\ & Y_2\end{bmatrix}\begin{bmatrix}Z_2 & \\ & Z_2\end{bmatrix}Z_3$$

$$\approx\frac{1}{16\sqrt{2}}X_3Y_3Z_3$$

$$\approx\frac{1}{16\sqrt{2}}X_3'Y_3Z_3\text{（}X_3'\text{为}X_3\text{的代替矩阵）}$$

$$t=5,N=32：F_{32}=\cdots\cdots$$

4.2.3 问题二的求解与结果分析

对于问题二来说，由于并没有限制每行元素的个数，因此求解起来存在一定难度，考虑到目标函数是求矩阵的 Frobenius 范数意义最小，本质上就是要使得β_1F_N和$A_1A_2...A_K$或者F_N和$\beta_2A_1A_2...A_K$尽可能相同，这里的相同是指矩阵中每个元素都尽可能相同，对于最终得到的矩阵$A_1'A_2...A_K$来说，其第一行元素受到A_1'影响，换句话说，A_1'的第一行元素，只会对矩阵$A_1'A_2...A_K$的第一行元素产生影响，因此对于矩阵A_1'的寻优来说，每行其实都是相互独立的，即整体目标分解我们就是去找到A_1'中每行的最优元素。

当决策矩阵规模很小的时候，采用遍历等方法是行之有效的，然而当t的取值不断增加的时候，会使得决策变量的规模呈现指数级别的，因此采用模拟退火算法[7]来对矩阵进行寻优，获取在$t=3$情况下的替换矩阵寻优结果，并得到两种目标函数计算下的最优β、$RMSE$以及硬件复杂度C。

表 2 t=3 时的求解结果

	评价标准	取值
第一种目标函数计算方式	β_1	1.4284
	$RMSE_1$	0.4071
	硬件复杂度C	27
第二种目标函数计算方式	β_2	0.4785
	$RMSE_2$	0.3415
	硬件复杂度C	36

当 t=4，N=16 时，结果如下表所示：

表 3 t=4 时的求解结果

	评价标准	取值
第一种目标函数计算方式	β_1	3.3284
	$RMSE_1$	0.5571
	硬件复杂度C	42
第二种目标函数计算方式	β_2	0.2745
	$RMSE_2$	0.4452
	硬件复杂度C	45

当 t=5,N=16 时，结果如下表所示：

表 4 t=5 时的求解结果

	评价标准	取值
第一种目标函数计算方式	β_1	4.2241
	$RMSE_1$	0.6424
	硬件复杂度C	39
第二种目标函数计算方式	β_2	0.1343
	$RMSE_2$	0.5491
	硬件复杂度C	48

从以上几组的结果中我们可以发现，第二种目标函数计算方式下的$RMSE$要优于第一种目标函数计算方式，除此之外，我们也发现，当不限制每行元素个数，只对矩阵元素的实部和虚部做约束时，随着t的取值不断增加，决策变量的规模呈现指数级别，即决策变量的规模会变得很庞大，这会影响最优的替换矩阵的寻找，而且找到的替换矩阵生成的$\mathcal{A}$计算出的$RMSE$也比较高。

4.3 问题三

4.3.1 问题三的分析

问题三在问题二的基础上，还需要同时满足约束 1，即需要在一定程度上限制A_k的稀疏性。在新的约束组合下，$t=1,2$都是不需对$\mathcal{A}$进行优化的了，因为他本身就已经满足约束以及误差最小。此时$\mathcal{P}=\{0,\pm1,\pm2,\pm4\}$，后续为详细的分类讨论过程：

- $t=1,N=2$

此时$F_2=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$，很显然满足约束 1 和 2，且$RMSE=0,C=0$。

- $t=2,N=4$

此时$F_4=\frac{1}{2}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -j & -1 & j\\ 1 & -1 & 1 & -1\\ 1 & j & -1 & -j\end{bmatrix}$，根据问题一的求解可以得知，$F_4$可以被分解成$X_1,Y_1,Z_1$

三个矩阵：

$$F_4=\frac{1}{2}\underbrace{\begin{bmatrix}1&0&1&0\\0&1&0&-j\\1&0&-1&0\\0&1&0&j\end{bmatrix}}_{X_1}\underbrace{\begin{bmatrix}1&1&0&0\\1&-1&0&0\\0&0&1&1\\0&0&1&-1\end{bmatrix}}_{Y_1}\underbrace{\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}}_{Z_1}$$

$$F_4=\frac{1}{2}X_1Y_1Z_1$$

可以看出X_1,Y_1,Z_1这三个矩阵都满足约束 1 和 2，此时$RMSE=0,C=0$。

- $t=3,N=8$

对F_8采用问题一中的分解方法，可将F_8分解成X_2,Y_2,Z_2三个矩阵：

$$F_8=\frac{1}{4\sqrt{2}}\underbrace{\begin{bmatrix}I&D_3\\I&-D_3\end{bmatrix}}_{X_2}\underbrace{\begin{bmatrix}F_4&\\&F_4\end{bmatrix}}_{Y_2}\underbrace{[P_8]}_{Z_2}$$

$$F_8=\frac{1}{4\sqrt{2}}X_2Y_2Z_2$$

$$D_3=\begin{bmatrix}1&&&\\&w_8^1&&\\&&w_8^2&\\&&&w_8^3\end{bmatrix}=\begin{bmatrix}1&&&\\&0.771-0.7071j&&\\&&-j&\\&&&-0.771-0.7071j\end{bmatrix}$$

其中Y_2矩阵又可借助问题一的分解策略及$t=2,N=4$的分解结果$F_4=X_1Y_1Z_1$进一步分解为：

$$\begin{bmatrix}F_4&\\&F_4\end{bmatrix}=\frac{1}{2}\begin{bmatrix}X_1Y_1Z_1&\\&X_1Y_1Z_1\end{bmatrix}=\frac{1}{2}\begin{bmatrix}X_1&\\&X_1\end{bmatrix}\begin{bmatrix}Y_1&\\&Y_1\end{bmatrix}\begin{bmatrix}Z_1&\\&Z_1\end{bmatrix}$$

所以F_8又可以表示为：

$$F_8=\frac{1}{4\sqrt{2}}\underbrace{\begin{bmatrix}I&D_3\\I&-D_3\end{bmatrix}}_{X_2}\underbrace{\begin{bmatrix}X_1&\\&X_1\end{bmatrix}\begin{bmatrix}Y_1&\\&Y_1\end{bmatrix}\begin{bmatrix}Z_1&\\&Z_1\end{bmatrix}}_{Y_2}\underbrace{[P_8]}_{Z_2}$$

对于Y_2,Z_2中的各个矩阵都满足约束 1 和 2，并且Y_2中的三个矩阵的C为 0，此时只有X_2不满足约束 2。对于X_2，可以继续分解为：

$$\begin{bmatrix}1&&1&&&&&\\&1&&\omega_8^1&&&&\\&&1&&\omega_8^2&&&\\&&&1&&&&\omega_8^3\\1&&-1&&&&&\\&1&&-\omega_8^1&&&&\\&&1&&-\omega_8^2&&&\\&&&1&&&&-\omega_8^3\end{bmatrix}=\underbrace{\begin{bmatrix}1&&1&&&&&\\&1&&1&&&&\\&&1&&1&&&\\&&&1&&&&1\\1&&-1&&&&&\\&1&&-1&&&&\\&&1&&-1&&&\\&&&1&&&&-1\end{bmatrix}}_{U_2}\underbrace{\begin{bmatrix}1&&&&&&&\\&1&&&&&&\\&&1&&&&&\\&&&1&&&&\\&&&&1&&&\\&&&&&\omega_8^1&&\\&&&&&&\omega_8^2&\\&&&&&&&\omega_8^3\end{bmatrix}}_{V_2}$$

显然，矩阵U_2能够同时满足约束 1 和约束 2，对于矩阵V来说，虽然能够满足约束 1，但是并不满足约束 2。因此，对F_8进行分解的关键就是找到一组矩阵去代替矩阵V_2,因此分

解得到的矩阵F_8形式如下：

$$F_8 \approx \frac{1}{4\sqrt{2}} U_2 A_2 \cdots A_{K-2} Y_2 Z_2$$

- $t=4, t=5$

该情况下的分析与$t=3$的分析十分相似，当$t=3$时，$F_8 \approx \frac{1}{4\sqrt{2}} U_2 V_2 Y_2 Z_2$，我们用利用算法寻找满足约束 1 和 2 的$A_2 \cdots A_{k-2}$代替$V_2$，然后用符号$X_2'$表示$U_2 A_2 \cdots A_{k-2}$，于是得到$F_8 \approx \frac{1}{4\sqrt{2}} X_2' Y_2 Z_2$，到了这一步，我们可以联想到问题二中对于$t=4, t=5$的分析，于是整体思路如下：

$t=2, N=4$：$F_4 = \frac{1}{2} X_1 \begin{bmatrix} F_2 & \\ & F_2 \end{bmatrix} Z_1$

$$= \frac{1}{2} X_1 Y_1 Z_1$$

$t=3, N=8$：$F_8 = \frac{1}{2\sqrt{2}} X_2 \begin{bmatrix} F_4 & \\ & F_4 \end{bmatrix} Z_2$

$$= \frac{1}{4\sqrt{2}} U_2 V_2 \begin{bmatrix} X_1 & \\ & X_1 \end{bmatrix} \begin{bmatrix} Y_1 & \\ & Y_1 \end{bmatrix} \begin{bmatrix} Z_1 & \\ & Z_1 \end{bmatrix} Z_2$$

$$\approx \frac{1}{4\sqrt{2}} U_2 A_2 \cdots A_{k-2} Y_2 Z_2 \text{（}A_2 \cdots A_{k-2}\text{用于替代}V_2\text{）}$$

$$\approx \frac{1}{4\sqrt{2}} X_2' Y_2 Z_2 \text{（}X_2'\text{表示}U_2 A_2 \cdots A_{k-2}\text{）}$$

$t=4, N=16$：$F_{16} = \frac{1}{4} X_3 \begin{bmatrix} F_8 & \\ & F_8 \end{bmatrix} Z_3$

$$\approx \frac{1}{16\sqrt{2}} U_3 V_3 \begin{bmatrix} X_2' & \\ & X_2' \end{bmatrix} \begin{bmatrix} Y_2 & \\ & Y_2 \end{bmatrix} \begin{bmatrix} Z_2 & \\ & Z_2 \end{bmatrix} Z_3$$

$$\approx \frac{1}{16\sqrt{2}} U_3 A_2 \cdots A_{k-2} Y_3 Z_3 \text{（}A_2 \cdots A_{k-2}\text{用于替代}V_3\text{）}$$

$$\approx \frac{1}{16\sqrt{2}} X_3' Y_3 Z_3 \text{（}X_3'\text{表示}U_3 A_2 \cdots A_{k-2}\text{）}$$

$t=5, N=32$：$F_{32} = \cdots\cdots$

4.3.2 问题三的建模与求解

如何找到一组矩阵去拟合矩阵V的问题其实是一个优化问题，其中：

目标函数：$\min RMSE$。

决策变量：第r个矩阵中的第l行第m列实部元素和虚部元素：

$$a_{lmr}, b_{lmr}, r = 2, \cdots K-2$$

约束 1：$A_2 \cdots A_{k-2}$中每个矩阵每行至多只有 2 个非 0 元素：

$$nonzero(a_{lr}+jb_{lr})\leqslant 0, r=2,\cdots,K-2, l=1,2,\cdots,8$$

其中，$nonzero(a_{lr}+jb_{lr})$表示第r个矩阵第l行所有元素的非 0 元素个数。

约束 2： $A_2\cdots A_{k-2}$中的每个矩阵的每个元素实部和虚部取值范围为：

$$a_{lmr}, b_{lmr}\in\mathcal{P}, \mathcal{P}=\{0,\pm1,\pm2,\pm4\}, r=2,\cdots,K-2, l,m=1,2,\cdots,8$$

如何找到一组矩阵$A_2\cdots A_{K-2}$去代替矩阵V_2。具体的分析如下，观察V_2可以看出，上对角元素可以看做是单位矩阵，下对角元素其实是 DFT 的旋转因子，如下所示：

$$\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & \omega_8^1 & & \\ & & & & & & \omega_8^2 & \\ & & & & & & & \omega_8^3 \end{bmatrix}$$

下对角元素中的旋转因子即为在半圆的负方向上进行旋转的弧度，当$N=8$时，DFT 的旋转因子在复平面的一个单位圆的分布如下图所示：

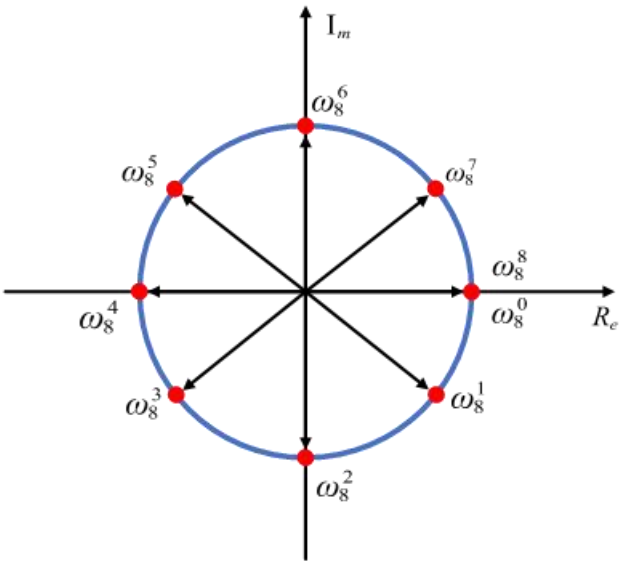

图 2 N=8 时，旋转因子在复平面上的分布

因此问题三的关键在于，如何在矩阵中元素的实部和虚部有约束的条件下，需要找到一组能够代替$\omega_8^1,\omega_8^2,\ldots,\omega_8^{\frac{N}{2}-1}$的向量，并且由于新的向量的长度可能是不一致的，需要找到一组 1、2、5、17 的公倍数，使得这组向量乘上对应的公倍数之后能够尽量分布在一个更大的圆上。

因此对于该问的求解就变为两个步骤：①在给定$\mathcal{P}$的范围中，找到一组实部和虚部都用整数表示的向量组合，能够尽量表示原始的$\omega_8^1,\omega_8^2,\ldots,\omega_8^{\frac{N}{2}-1}$向量的方向；②针对新的向量组合，继续寻找一组公倍数，能够使得新的向量组合中的每个向量乘上对应的公倍数后，尽量在同一个圆上，并且希望这一组公倍数也能够通过给定$\mathcal{P}$范围中的整数得到。

针对步骤①，在给定的整数里面找到一组向量能够表示原始$\omega_8^1,\omega_8^2,\ldots,\omega_8^{\frac{N}{2}-1}$的方向，

首先，题目给定了整数的取值范围为$\mathcal{P}=\{0,\pm1,\pm2,\pm4\}$。于是，在我们的方案中，使用新的向量$W_N^i=a_i+jb_i,i=1,2,...,\frac{N}{2}-1$替换原始向量，其中$a_i,b_i\in\mathcal{P}$，并且每个新的向量都与对应的原始向量的方向尽量相同或者逼近。因此，该问题为一个优化问题，优化的目标为新的向量$W_N^i,i=1,2,...,\frac{N}{2}-1$与对应原始向量的夹角尽可能小。

目标函数：$\min|arg(w_i)-arg(a_i+jb_i)|$

其中$arg(z)$表示向量 z 的辐角，即与实轴正方向的夹角。

决策变量：

（1）分解的向量数量$arg(i)$

（2）每个向量中的元素a_i和b_i

约束条件：$a_i,b_i\in\mathcal{P}$

对于这个算法，分解的向量数量越多肯定会使得目标函数最小，因此可以固定分解向量的数量，找到该分解数量下，每个向量的最优分解，具体的实现算法如下：

算法 3：旋转因子向量的最优分解算法

Input: 旋转因子向量 w(目标向量)，分解的向量个数 k

Output: 分解的向量实部 ai 和虚部 bi

Initialize: 最优分解向量 best_des 为单位向量，min_ang 对应的夹角

```
01. for a1,b1 in range(𝒫):                                  //遍历矩阵元素
02. .....                                                   //分解的向量个数
03.     for ak,bk in range(𝒫):
04.         if (norm([a1,b1;...;ak,bk])):                   //分解向量模长不为 0
05.             for wi in range(w):                         //遍历所有目标向量
06.                 if (|ang([a1,b1;...;ak,bk],wi)|<min_ang(wi))://更优的角度
07.                     best_des(wi) = [a1,b1;...;ak,bk]        //更新分解向量
08.                     min_ang(wi) = |ang([a1,b1;...;ak,bk],wi)|   //更新目标函数
09.                 end if
10.             end for
11.         end if
12.     end for
13.......
14.end for
```

当$t=3,N=8$时，得到满足约束 2 的最优分解复数向量，以及与原始旋转因子向量的夹角：

表 5 t=3 时求解得到的最优分解复数向量

原始旋转因子向量	分解满足约束 2 向量	分解向量模长	相对夹角/°
1+0j	1+0j	1	0
0.7071-0.7071j	1-1j	1.414	0
0-1j	0-1j	1	0
-0.7071-0.7071j	-1-1j	1.414	0

原始的矩阵V_2可以被分解如下：

$$\gamma V_2 = T\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1-j & & \\ & & & & & & -j & \\ & & & & & & & -1-j \end{bmatrix}$$

其中对角矩阵T表示左侧矩阵的缩放矩阵，在这样的考虑下，矩阵T需要使得与其右侧矩阵相乘能够为原始左侧矩阵的倍数，这样能够在参数γ的缩放下使两个矩阵尽可能相等，并且因为旋转矩阵的模为 1，因此T需要与其右侧矩阵相乘中对角元素的模长尽量一样。

除此之外，矩阵T还需要满足给定约束 2 中整数的约束，因此需要将矩阵T再次进行分解，假设拆分形式如下：

$$T = \begin{bmatrix} a_{11}+b_{11}j & & & \\ & a_{22}+b_{22}j & & \\ & & \ddots & \\ & & & a_{88}+b_{88}j \end{bmatrix} \cdots \begin{bmatrix} a_{u1}+b_{u1}j & & & \\ & a_{(u+1)2}+b_{u2}j & & \\ & & \ddots & \\ & & & a_{(u+8)8}+b_{(u+8)8}j \end{bmatrix}$$

其中u表示将矩阵T拆分的个数，第s个矩阵中任意非零元素的实部a_{lms}和虚部b_{lms}满足约束$a_{lms},b_{lms}\in\mathcal{P},\mathcal{P}=\{0,\pm1,\pm2,\pm4\},l,m=1,2\cdots8,s=1,2,\cdots,u$

矩阵T分解的矩阵的目标为使得分解向量乘以放缩矩阵T后的各向量模长尽可能一样，因此该问题是一个优化问题，其中：

目标函数：$\min RMSE$。

决策变量：第s个矩阵中的第l行第m列实部元素和虚部元素：a_{lms},b_{lms}

约束：第s个矩阵中每个非零元素的实部a_{lms}和虚部取值范围为：

$$a_{lms},b_{lms}\in\mathcal{P},\mathcal{P}=\{0,\pm1,\pm2,\pm4\},l,m=1,2\cdots8,s=1,2,\cdots,u$$

由于放缩的整数个数并不知道，并且当放缩的整数个数较大时，决策变量的取值范围也非常高，因此我们采用一种启发式规则来加速最优分解整数的择优过程。本文采用基于双扰动的模拟退火算法进行求解

模拟退火算法（Simulated Annealing）是一种优化算法，该算法通过模拟金属在退火过程中冷却时的行为，以寻找问题的全局最优解。模拟退火算法的基本思想是从一个初始解开始，通过不断迭代，通过接受较差的解以一定的概率来避免陷入局部最优解。算法在每

次迭代中会通过随机扰动当前解，并根据一定的准则来决定是否接受新的解。随着迭代的进行，算法会逐渐降低接受较差解的概率，使搜索过程逐渐收敛于全局最优解。采用双扰动的目的是相对于单点变异来说，产生两个点的变异能使得算法在一定程度上最快收敛。

算法 4：基于双交换模拟退火算法的整数分解算法

Input: 初始温度 T0，终止温度 Tend，降温速度 r，Markov 链的长度 L,目标向量 w0-wN/2-1，最大整数相乘数量 M，分解向量 w'0,w'1,...,w'N/2-1，可乘整数数组 E

Output: 放缩矩阵 T, 硬件复杂度 C

Initialize: 当前温度 T=T0, 决策变量 S1, S1 大小为 N*M/4,每个元素在 E 中
//不同模长的向量个数为 N/4

```
01. while T>Tend
02.     for k in range(L):                          //遍历所有的链长
03.         S2 = S1;                                //初始化新解
04.         S2(randi(2)) = randi(E,2)               //双扰动产生新解
05.         ΔE = obj(S2)-obj(S1)
06.         if  ΔE<0:
07.             S1 = S2
08.         else if  e^(-ΔE/T) > ξ,  ξ∈[0,1]
09.             S1 = S2
10.         end if
11.     end for
12.     T = r*T
13. end while
14.计算放缩矩阵 T
15.C =  调用算法 4
16.return T, C
```

采用模拟退火算法进行求解，分别求解不同整数分解的数量下，$RMSE$ 最小的分解方案，并计算其硬件复杂度。本文将分解的整数数量从 1 开始进行此时最优的放缩矩阵 T 的求解，得到如下表所示的结果：

表 6 t=3 时的放缩矩阵 T 求解结果

	最多分解个数	1	3	5	10	20
第一种目标函数计算方式	β_1	2.8284	2.8284	2.8284	2.8284	70.7107
	$RMSE_1$	0.2071	0.2071	0.2071	0.2071	0.6777
	硬件复杂度C	0	0	0	0	27
第二种目标函数计算方式	β_2	0.1768	0.0071	0.0012	1.058e-6	1.0837e-11
	$RMSE_2$	0.0518	0.0068	0.0038	0.0038	0.0027
	硬件复杂度C	0	36	42	84	249

从上表中可以看出，随着分解矩阵的个数的增加，β_2 方法比 β_1 方法的 $RMSE$ 更小，

但硬件复杂度C会更差。于是在综合考虑下，当$t=3$时，本文选择第二种目标函数计算且最多整数分解个数为 5 条件下得出的放缩矩阵，此时$RMSE$为 0.0038，硬件复杂度C为 42，放缩矩阵T表示为：

$$T=\begin{bmatrix}289 & & & & & & & \\ & 289 & & & & & & \\ & & 289 & & & & & \\ & & & 289 & & & & \\ & & & & 289 & & & \\ & & & & & 200 & & \\ & & & & & & 289 & \\ & & & & & & & 200\end{bmatrix}$$

$$=\begin{bmatrix}17*17 & & & & & & & \\ & 17*17 & & & & & & \\ & & 17*17 & & & & & \\ & & & 17*17 & & & & \\ & & & & 17*17 & & & \\ & & & & & 5*2*5*2*2 & & \\ & & & & & & 17*17 & \\ & & & & & & & 5*2*5*2*2\end{bmatrix}$$

其中，2 可以采用$(1-j)(1+j)$表示，5 可以采用$(1-2j)(1+2j)$表示，17 可以用$(1-4j)(1+4j)$表示，最终放缩矩阵T又可以按照上述 2、5、17 的表示分解成2^5个对角矩阵的乘积。

当$t=4$时，采用$t=3$得到的最优分解结果对其进行替代。采用模拟退火算法进行求解，分别求解不同整数分解的数量下，$RMSE$最小的分解方案，并计算其硬件复杂度。本文将分解的整数数量从 1 开始进行此时最优的放缩矩阵T的求解，得到如下表所示的结果：

表 7 t=4 时的放缩矩阵 T 求解结果

	最多分解个数	1	3	5	10	20
第一种目标函数计算方式	β_1	68	80	80	3400	1.9652e11
	$RMSE_1$	3.0772	1.4629	0.9804	23.3756	8.577e8
	硬件复杂度C	33	33	27	99	438
第二种目标函数计算方式	β_2	0.0147	0.0029	1.47e-4	1.498e-8	6.09e-13
	$RMSE_2$	0.0453	0.0122	0.0052	0.0052	0.0103
	硬件复杂度C	33	60	117	252	450

从上表中可以看出，当$t=4$时，本文选择第二种目标函数计算且最多整数分解个数为 5 条件下得出的放缩矩阵，此时$RMSE$为 0.0052，硬件复杂度C为 117，由于维度过高，这里不对放缩矩阵T进行展示。

使用同样的方法，当$t=5$时，得到如下表所示的结果：

表 8 t=5 时的放缩矩阵 T 求解结果

	最多分解个数	1	3	5	10	20
第一种目标函数计算方式	β_1	96.1665	141.42	565.69	3.27e6	1.3896e12
	$RMSE_1$	3.2117	2.0383	4.399	2.3914e4	1.3621e10
	硬件复杂度C	57	108	120	441	981
第二种目标函数计算方式	β_2	3.058e-4	4.160e-4	3.058e-4	6.117e-4	3.0584e-4
	$RMSE_2$	0.0197	0.0109	0.011	0.0223	0.018
	硬件复杂度C	180	201	192	168	174

从上表中可以看出，当$t=5$时，本文选择第二种目标函数计算且最多整数分解个数为5条件下得出的放缩矩阵，此时$RMSE$为0.011，硬件复杂度C为117，由于维度过高，这里不对放缩矩阵T进行展示。

4.4 问题四

4.4.1 问题四的分析

考虑矩阵$F_N=F_{N1}\otimes F_{N2}$，其中F_{N1}和F_{N2}分别是N_1和N_2维的DFT矩阵，$\otimes$表示Kronecker积。当$N_1=4,N_2=8$时：

$$\begin{aligned} F_N &= F_4\otimes F_8 \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \otimes F_8 \\ &= \begin{bmatrix} F_8 & F_8 & F_8 & F_8 \\ F_8 & -jF_8 & -F_8 & jF_8 \\ F_8 & -F_8 & F_8 & -F_8 \\ F_8 & jF_8 & -F_8 & -jF_8 \end{bmatrix} \end{aligned}$$

通过Kronecker积使得原始4×4和8×8的两个矩阵变为一个32×32的矩阵F_N，通过该矩阵的性质发现，F_N又可以再分解为两个矩阵相乘：

$$F_N=\begin{bmatrix} I & I & I & I \\ I & -jI & -I & jI \\ I & -I & I & -I \\ I & jI & -I & -jI \end{bmatrix} \begin{bmatrix} F_8 & & & \\ & F_8 & & \\ & & F_8 & \\ & & & F_8 \end{bmatrix}$$

对于F_N分解结果中左侧的矩阵，其形状与F_4非常相似，主要的区别就是原始的数值1都变为大小为8×8的单位矩阵I，根据问题一的分解思路，可将F_N左侧的矩阵进一步分解为：

$$\begin{bmatrix} I & I & I & I \\ I & -jI & -I & jI \\ I & -I & I & -I \\ I & jI & -I & -jI \end{bmatrix} = \begin{bmatrix} I & 0 & I & 0 \\ 0 & I & 0 & -jI \\ I & 0 & -I & 0 \\ 0 & I & 0 & jI \end{bmatrix} \begin{bmatrix} I & I & 0 & 0 \\ I & -I & 0 & 0 \\ 0 & 0 & I & I \\ 0 & 0 & I & -I \end{bmatrix} \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & 0 & I \end{bmatrix}$$

分解得到的矩阵都满足约束 1 和 2，并且其硬件复杂度同样为 0。这时只剩下F_N分解结果中右侧的矩阵，对于该矩阵中的每个子块F_8可以使用问题三中$t=3$情况下的分解方法进行分解即可。

从问题三中可知：

$$\begin{aligned}F_8 &= \frac{1}{2\sqrt{2}} X_2 \begin{bmatrix} F_4 & \\ & F_4 \end{bmatrix} Z_2 \\ &= \frac{1}{4\sqrt{2}} U_2 V_2 \begin{bmatrix} X_1 & \\ & X_1 \end{bmatrix} \begin{bmatrix} Y_1 & \\ & Y_1 \end{bmatrix} \begin{bmatrix} Z_1 & \\ & Z_1 \end{bmatrix} Z_2 \\ &\approx \frac{1}{4\sqrt{2}} U_2 A_2 \cdots A_{k-2} Y_2 Z_2 \text{（}A_2 \cdots A_{k-2}\text{用于替代}V_2\text{）} \\ &\approx \frac{1}{4\sqrt{2}} X_2' Y_2 Z_2 \text{（}X_2'\text{表示}U_2 A_2 \cdots A_{k-2}\text{）}\end{aligned}$$

对于$\mathrm{diag}(F_8,F_8,F_8,F_8)$的最优替换，F8 可以替换为$F_8 \approx \frac{1}{4\sqrt{2}} U_2 V_2 Y_2 Z_2$，其中的$U_2$、$Y_2$、$Z_2$都是仅由$1$、$-1$、$j$、$-j$组成，满足约束，且硬件复杂度为 0，因此可以不进行考虑，此时$\mathrm{diag}(F_8,F_8,F_8,F_8)$可以近似为：

$$\begin{aligned}\mathrm{diag}(F_8,F_8,F_8,F_8) \approx \frac{1}{4\sqrt{2}} &\mathrm{diag}(U_2,U_2,U_2,U_2) * \mathrm{diag}(V_2,V_2,V_2,V_2) \\ &* \mathrm{diag}(Y_2,Y_2,Y_2,Y_2) * \mathrm{diag}(Z_2,Z_2,Z_2,Z_2)\end{aligned}$$

可以看出，对于$\mathrm{diag}(F_8,F_8,F_8,F_8)$的最优替换，关键就在于找到$\mathrm{diag}(V_2,V_2,V_2,V_2)$的最优替换，也就是找到一组矩阵去代替矩阵$V_2$，其中$V_2=\mathrm{diag}(1,1,1,1,1,w_8^1,w_8^2,w_8^3)$。

因此分解得到的矩阵F_8形式如下：

$$F_8 \approx \frac{1}{4\sqrt{2}} U_2 A_2 \cdots A_{K-2} Y_2 Z_2$$

4.4.2 问题四的建模

如何找到一组矩阵去拟合矩阵V的问题其实是一个优化问题，其中：

目标函数：$\min RMSE$。

决策变量：第r个矩阵中的第l行第m列实部元素和虚部元素：

$$a_{lmr}, b_{lmr}, r=2, \cdots K-2$$

约束 1：$A_2 \cdots A_{k-2}$中每个矩阵每行至多只有 2 个非 0 元素：

$$nonzero(a_{lr}+jb_{lr}) \leqslant 0, r=2, \cdots, K-2, l=1,2,\cdots,8$$

其中，$nonzero(a_{lr}+jb_{lr})$表示第r个矩阵第l行所有元素的非 0 元素个数。

约束 2：$A_2 \cdots A_{k-2}$中的每个矩阵的每个元素实部和虚部取值范围为：

$$a_{lmr}, b_{lmr} \in \mathcal{P}, \mathcal{P}=\{0, \pm 1, \pm 2, \pm 4\}, r=2, \cdots, K-2, l, m=1,2,\cdots,8$$

关于如何找到一组矩阵$A_2 \cdots A_{K-2}$去代替矩阵V_2，则是直接使用问题三中设计的旋转因子向量的最优分解算法。

4.4.3 问题四的求解与结果分析

采用下表中第二种对于目标函数的计算方式来求解问题四，并分析不同的最大整数分解个数下，最优目标函数的取值，下表为在最大整数分解分别为 1,3,5,10,20,50 时，算法求解的最小 $RMSE$ 取值以及对应的硬件复杂度 C：

表 9 问题四的求解结果

最大分解整数个数	1	3	5	10	20	50
$RMSE$	0.0259	0.0034	0.0019	0.0019	0.0013	1.9297e-4
β_2	0.0884	0.0035	6.117e-4	6.22e-8	1.3546e-13	1.9419e-27
硬件复杂度 C	0	144	168	420	1140	2028

得到的分解矩阵和问题三的结果完全相同，即选择最大分解整数个数为 5 这组取值。在该取值下，双扰动模拟退火算法的迭代曲线如下所示：

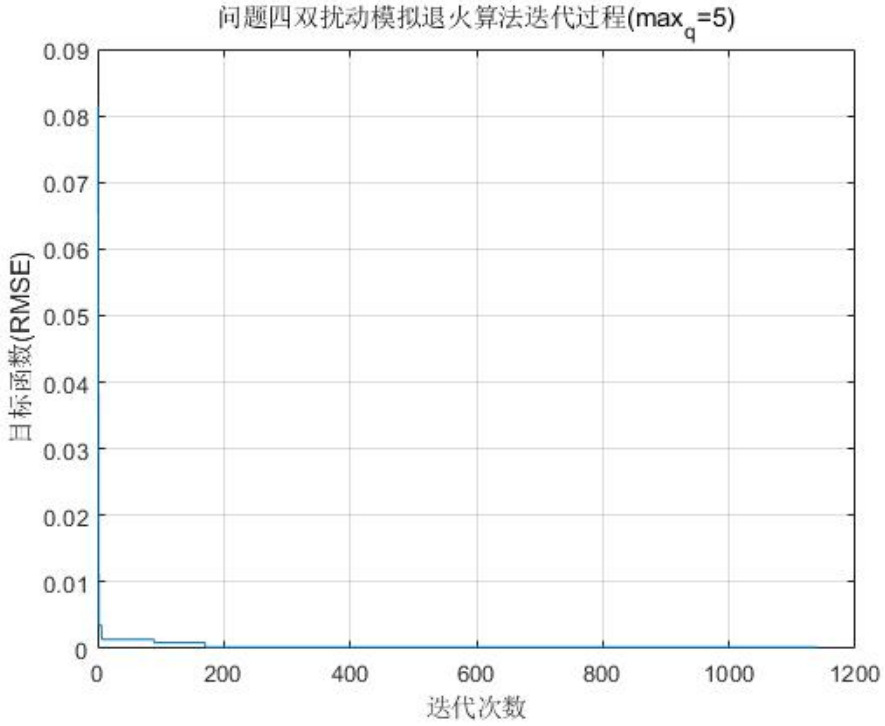

图 3 q=5 时双扰动模拟退火算法迭代过程

从上图可以得出，算法在很少的迭代次数内就能迭代完成，最终收敛到最优解 $RMSE=0.0019$。

4.5 问题五

4.5.1 问题五的分析

由于本题需要研究 $N=2^t, t=1,2,3,...$ 的 DFT 矩阵 F_N，我们希望得到一个一般的规律，而非继续沿用问题三中当前 t 的优化需要借助 $t-1$ 时的最优结果这种方式，对此我们使用 $t=4$ 情况作为样例介绍我们在本题中的详细思路。

我们发现 $t=4$ 时的 V_3 矩阵包含 8 个旋转因子 $w_{16}^0, w_{16}^1, w_{16}^2, w_{16}^3, w_{16}^4, w_{16}^5, w_{16}^6, w_{16}^7$，$t=3$

时的V_2矩阵包含 4 个旋转因子w_8^0,w_8^1,w_8^2,w_8^3，$t=2$时的V_1矩阵包含 2 个旋转因子w_4^0,w_4^1，如下图所示：

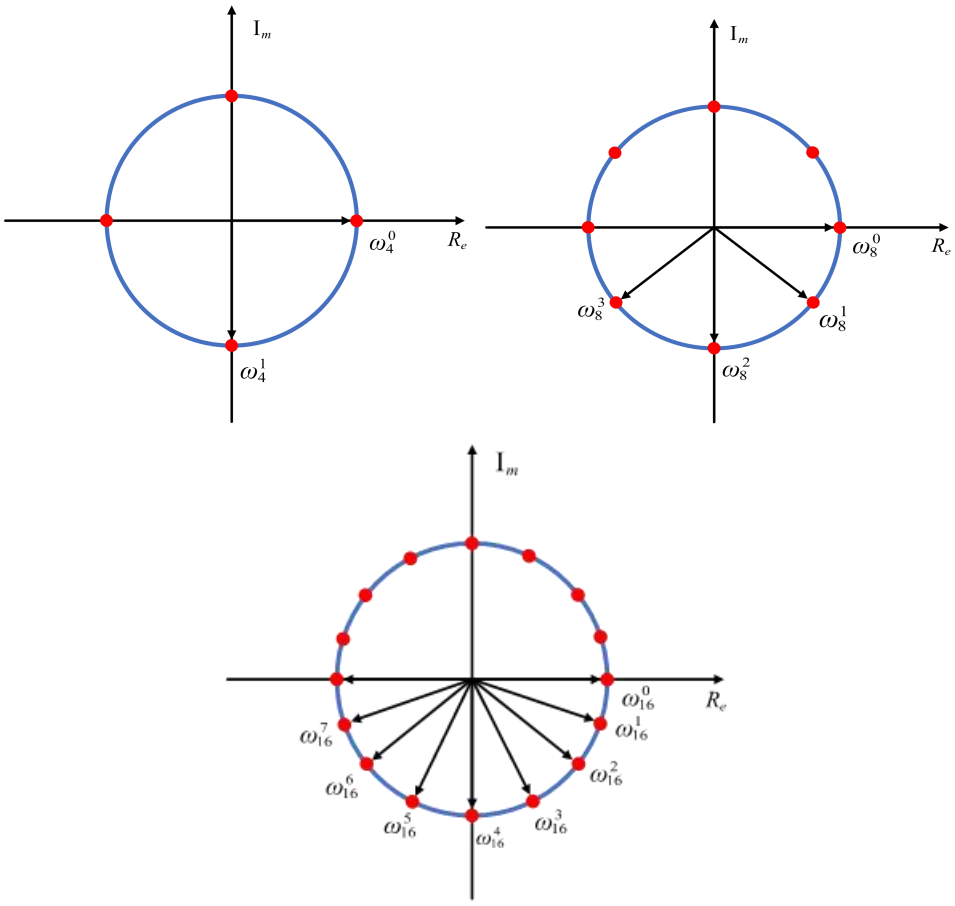

图 4 t=2,3,4 时旋转因子在复平面的分布

利用旋转因子的缩放性可以得知$w_4^0=w_8^0=w_{16}^0$，$w_4^1=w_8^2=w_{16}^4$，…。在问题三中，$t=4$时我们只考虑了V_3的替换，Y_3中使用了$t=3$的最优分解结果，下为问题三的思路：

$$t=3,N=8:\ F_8=\frac{1}{2\sqrt{2}}X_2\begin{bmatrix}F_4 & \\ & F_4\end{bmatrix}Z_2$$

$$=\frac{1}{4\sqrt{2}}U_2V_2\begin{bmatrix}X_1 & \\ & X_1\end{bmatrix}\begin{bmatrix}Y_1 & \\ & Y_1\end{bmatrix}\begin{bmatrix}Z_1 & \\ & Z_1\end{bmatrix}Z_2$$

$$\approx\frac{1}{4\sqrt{2}}U_2A_2\cdots A_{k-2}Y_2Z_2\ （A_2\cdots A_{k-2}用于替代V_2）$$

$$\approx\frac{1}{4\sqrt{2}}X_2'Y_2Z_2\ （X_2'表示U_2A_2\cdots A_{k-2}）$$

$$t=4,N=16：F_{16}=\frac{1}{4}X_3\begin{bmatrix}F_8 & \\ & F_8\end{bmatrix}Z_3$$

$$\approx\frac{1}{16\sqrt{2}}U_3V_3\begin{bmatrix}X_2' & \\ & X_2'\end{bmatrix}\begin{bmatrix}Y_2 & \\ & Y_2\end{bmatrix}\begin{bmatrix}Z_2 & \\ & Z_2\end{bmatrix}Z_3$$

$$\approx\frac{1}{16\sqrt{2}}U_3A_2\cdots A_{k-2}Y_3Z_3（A_2\cdots A_{k-2}用于替代V_3）$$

$$\approx\frac{1}{16\sqrt{2}}X_3'Y_3Z_3（X_3'表示U_3A_2\cdots A_{k-2}）$$

但是在问题四，我们的设计思路是，在考虑V_3的替换时就已经找到了一组方向相近的向量W_{16}^i替代$w_{16}^0,w_{16}^1,w_{16}^2,w_{16}^3,w_{16}^4,w_{16}^5,w_{16}^6,w_{16}^7$，这时$Y_3$中的$w_8^0,w_8^1,w_8^2,w_8^3$利用放缩性变为$w_{16}^0,w_{16}^2,w_{16}^4,w_{16}^6$，故这些向量仍然可以使用考虑$V_3$时求取的替代向量$W_{16}^0,W_{16}^2,W_{16}^4,W_{16}^6$，$Y_2$中的$w_4^0,w_4^1$本身就满足约束要求，故不做替代。做完向量替代工作后，只需要为包含替代向量的矩阵求取对应的方所矩阵T即可，该步骤就与问题三中的设计思路完全相似了。

通过上面的阐述，我们可以发现，当前t的优化不在需要依赖$t-1$时的最优结果，只需根据自身便可进行求解。

4.5.2 问题五的建模

问题五在问题三的基础上，首先需要寻找合适的q以满足精度要求，并且使得硬件复杂度C尽量低。其次增加对于精度的限制来研究矩阵的分解方案，将精度要求限制在 0.1 之内，即 *RMSE* 小于等于 0.1。而详细的分类讨论过程与问题三完全一致，此处不再赘述。唯一的区别在优化问题的目标函数与约束条件，因为问题五不仅需要对使得 *RMSE* 最低的$\mathcal{A}$、β和$\mathcal{P}$进行求解，这样 *RMSE* 小于等于 0.1 的约束就显得多余，我们考虑在模型中加入硬件复杂度最小的目标，将原始问题中的模型改进成一个综合考虑 *RMSE* 最低以及硬件复杂度最小的多目标优化问题，其中：

目标函数 1：$\min RMSE$。

目标函数 2：$\min C$。

决策变量：第r个矩阵中的第l行第m列实部元素和虚部元素：

$$a_{lmr},b_{lmr},r=2,\cdots k-2$$

约束 1：$A_2\cdots A_{k-2}$中每个矩阵每行至多只有 2 个非 0 元素：

$$nonzero(a_{lr}+jb_{lr})\leqslant 0,r=2,\cdots,k-2,l=1,2,\cdots,2^t$$

其中，$nonzero(a_{lr}+jb_{lr})$表示第r个矩阵第l行所有元素的非 0 元素个数。

约束 2：$A_2\cdots A_{k-2}$中的每个矩阵的每个元素实部和虚部取值范围为：

$$a_{lmr},b_{lmr}\in\mathcal{P},\mathcal{P}=\{0,\pm1,\pm2,\cdots,\pm2^{q-1}\},r=2,\cdots,k-2,l,m=1,2,\cdots,2^t,q=\log_2(\max\{|a_{lmr}|,|b_{lmr}|\})$$

约束 3：$RMSE\leqslant 0.1$。

此时q需要根据该优化问题得出的结果来确定，所以$q=\log_2^{(\max\{|a_{ij}|,|b_{ij}|\})}$。

4.5.3 问题五的求解与结果分析

问题五是在问题三的基础上进行改进，其中对q不再固定，并且增加$RMSE\leqslant 0.1$的约束。在问题三中，其求解思路是在每个最大分解整数个数的情况下，通过模拟退火算法

找到最小的$RMSE$。但是在问题五中，q的取值不在固定，因此可以同时在不同的q和不同的最大分解整数个数中找到在这样的参数下，$RMSE$值以及对应的硬件复杂度。

由于本问题为多目标优化模型，因此需要找到满足约束条件下，$RMSE$和硬件复杂度C的平衡点，可以采用帕累托最优的思想，产生对应的帕累托最优解。

具体的实现步骤如下：

算法 5：最优多目标矩阵分解算法

Input: 最大 q 值 max_q, 最大整数拆分个数 max_M, 分解向量个数 k

Output: 帕累托最优集合 S

Initialize:

```
for i in range(max_q):
    for j in range(max_M):
        w' =  算法 3(k)                          //调用算法 n 得到最优替换向量
        T, C =  算法 4(w', max_q,max_M)          //模拟退火算法得到最优分解
        RMSE = obj(T)                                  //计算 RMSE
        if pareto_best(RMSE,C) in S:                   //当前解是帕累托最优的
            if(RMSE<=0.1)
                S = [S RMSE,C]                         //添加帕累托最优集合
            end if
        end if
    end for
end for
return S
```

当$t=3$时，设计对问题五的求解，设计分解向量的个数从 1 遍历到 8，设计分解的整数个数从 1 遍历到 30，在遍历的所有可能的参数下，找到这些参数下的最优$RMSE$，并将两个目标函数记录下来，所有方案中两个目标函数的关系如下图所示。

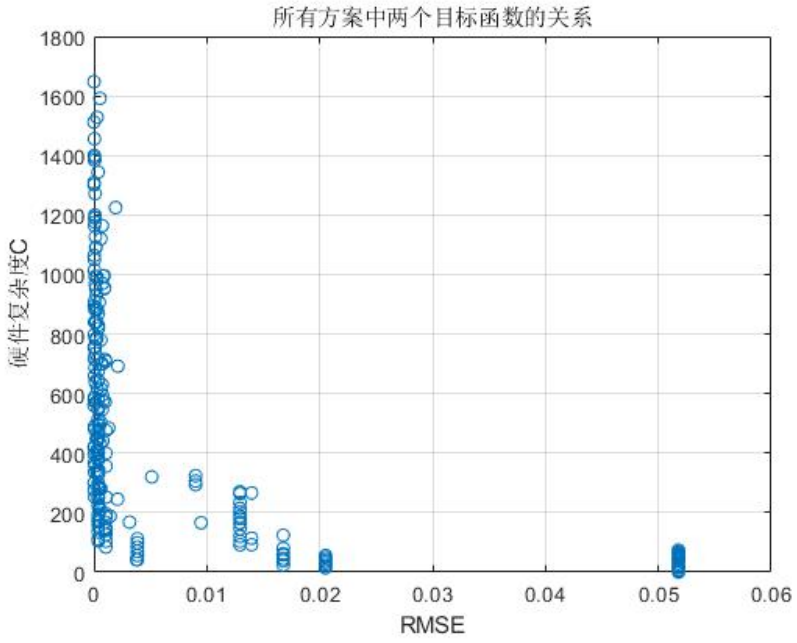

图 5 所有方案中两个目标函数的关系

在这些所有的方案中，找出一种能在目标$RMSE$和目标硬件复杂度C这两个目标中达

到权衡的方案，因此在所有的方案中，选出一种在两个目标中表现都较为出色的方案，此时硬件复杂度C为42，$RMSE$为0.0035，此时$q=3$，最大分解个数为2。虽然允许q取大，但是$t=3$时，q取值仍较小，因此也不一定表示越大的q取值会使得结果更好。

五、模型的评价与推广

5.1 模型的优点

1) 本文模型对于传统的FFT算法进行优化，在DFT分解矩阵在满足整数约束的条件下进行进一步分解，并且在*RMSE*尽可能小的情况下，也综合考虑到了硬件的复杂度，实现了*RMSE*和硬件复杂度两个目标都小的目标。
2) 本文采用了一种基于双扰动的模拟退火算法来加速优化算法的求解速度，双扰动模拟退火算法相对于原始单扰动的模拟退火算法会有更快的收敛速度。
3) 本文将抽象的矩阵运算化简为对向量方向以及模长的拟合，极大程度上简化了问题的复杂度。

5.2 模型的缺点

1) 在对第二问进行求解中，并没有对问题二的模型进行简化，因此问题二的求解具有极高的复杂度，即使使用了某些启发式算法，但是仍然很难达到较好的效果。
2) 在问题五的求解中，仅仅给出了对DFT矩阵同时进行分解的求解思路与模型，并没有给出一个能够满足所有DFT矩阵分解的通式。

5.3 模型的推广

本文提出了一种多参数影响下的多目标优化模型，该模型能够实现多个参数共同影响的模型下，多个目标函数的寻优。并且，本文提出的矩阵分解策略，不仅保证了没有过多精度的丢失，也能保证硬件复杂度尽可能小，用稀疏矩阵的形式也能使得矩阵的存储需求变得容易。

参考文献

[1] R. E. Blahut, Fast Algorithms for Digital Signal Processing. Cambridge,U.K.: Cambridge Univ. Press, 2010.

[2] D. Suarez et al., “Multi-beam RF aperture using multiplierless FFT approximation,” Electron. Lett., vol. 50, no. 24, pp. 1788–1790, 2014.

[3] C. J. Tablada et al., “A class of DCT approximations based on the FeigWinograd algorithm,” Signal Process., vol. 113, pp. 38–51, 2015.

[4] Potluri U S, Madanayake A, Cintra R J, et al. Multiplier-free DCT approximations for RF multi-beam digital aperture-array space imaging and directional sensing[J]. Measurement Science and Technology, 2012, 23(11): 114003.

[5] 余品能.离散付里叶变换(DFT)的一种高效矩阵分解算法[J].工程兵工程学院学报,1992(04):55-60+97.

[6] 曹伟丽.快速傅里叶变换的原理与方法[J].上海电力学院学报,2006(02):192-194.

[7] Wu R, Wang R, Hao J, et al. Multiobjective Hydropower Reservoir Operation Optimization with Transformer-Based Deep Reinforcement Learning[J]. arXiv preprint arXiv:2307.05643, 2023.

附录

1.compute_cpx.m

```
function [max_q,L] = compute_cpx(A,B,N)
    max_q = 0;
    L = 0;
    for i = 1:N
        for j = 1:N
            a = A(i,j);
            b = B(i,j);
            if real(a)
                max_q = max(max_q,log2(abs(real(a)))+1);
                        end
                        if imag(a)
                            max_q = max(max_q,log2(abs(imag(a)))+1);
                        end
                        if real(b)
                            max_q = max(max_q,log2(abs(real(b)))+1);
                        end
                        if imag(b)
                            max_q = max(max_q,log2(abs(imag(b)))+1);
                        end
                    end
                end

                for i = 1:N
                    for j = 1:N
                        temp = A(i,j);
                        gs1 = (temp == 0);
                        gs2 = ((abs(real(temp)) == 1) && imag(temp) == 0);
                        gs3 = (real(temp) == 0 && abs(imag(temp)) == 1);
                        gs4 = (abs(real(temp)) == 1 && abs(imag(temp)) == 1);
                        if ~(gs1|| gs2||gs3||gs4)
                            for t = 1:N
                                temp2 = B(j,t);
                                gs1 = (temp2 == 0);
                                gs2 = ((abs(real(temp2)) == 1) && imag(temp2) == 0);
                                gs3 = (real(temp2) == 0 && abs(imag(temp2)) == 1);
                                gs4 = (abs(real(temp2)) == 1 && abs(imag(temp2)) == 1);
                                if ~(gs1|| gs2||gs3||gs4)
                                    L = L+1;
                                end
                            end
```

```
                    end
                end
            end
        end
```

2.des2ang.m

```
function ang = des2ang(des)
% 分解向量的转换
ang = des(1,1) + des(1,2)*1i;
for i = 2:size(des,1)
    ang = ang*(des(i,1) + des(i,2)*1i);
end
```

3.find_best_beta.m

```
function [beta1,beta2] = find_best_beta(FN,A,N)
    top = 0;
    down = 0;
    for i = 1:N
        for j = 1:N
            top = top + FN(i,j)*A(i,j);
            down = down + FN(i,j)^2;
        end
    end
                beta1 = top/down;
                beta2 = 1/beta1;
        end
```

4. flattenCellArray.m

```
% 将 cell 数组展开
function flattenedArray = flattenCellArray(cellArray)
    flattenedArray = {};
    for i = 1:numel(cellArray)
        if iscell(cellArray{i})
            % 如果元素是 cell 数组，递归地展开并合并
            subArray = flattenCellArray(cellArray{i});
            flattenedArray = [flattenedArray, subArray];
        else
                    % 如果元素不是 cell 数组，直接合并
                    flattenedArray = [flattenedArray, cellArray{i}];
                end
            end
        end
```

5.get_fft_matrix.m

```
function A_star = get_fft_matrix(cnt,n)
cnt = cnt * 2;
if (n == 2)
    a = [1 1;
         1 -1;];
    A = zeros(n*cnt);
    for i = 1: cnt
        A(1+n*(i-1):n*i,1+n*(i-1):n*i) = a;
    end
                b = [1 0;
                     0 1;];
                B = zeros(n*cnt);
                for i = 1: cnt
                    B(1+n*(i-1):n*i,1+n*(i-1):n*i) = b;
                end
                c = [1 0;
                     0 1;];
                C = zeros(n*cnt);
                for i = 1: cnt
                    C(1+n*(i-1):n*i,1+n*(i-1):n*i) = c;
                end
                A_star = {A,B,C};
                return
            end
            w = exp(-(2*pi*1i)/n);
            a = zeros(n);
            for i = 1:n/2
                a(i,i) = 1;
                a(i+n/2,i) = 1;
                a(i,i+n/2) = w^(i-1);
                a(i+n/2,i+n/2) = -w^(i-1);
            end
            A = zeros(n*cnt);
            for i = 1: cnt
                A(1+n*(i-1):n*i,1+n*(i-1):n*i) = a;
            end
            B = get_fft_matrix(cnt,n/2);
            c = zeros(n);
            for i = 1:n/2
                c(i,i*2-1) = 1;
                c(i+n/2,2*i) = 1;
            end
```

```
43.             C = zeros(n*cnt);
44.             for i = 1: cnt
45.                 C(1+n*(i-1):n*i,1+n*(i-1):n*i) = c;
46.             end
47.             A_star = {A,B,C};
```

6.init.m

```
1. function S1= init(n,t,max_M)
2.     S1 = zeros(1,n*max_M);
3.     for i = 1:length(S1)
4.         temp = randi(length(t));
5.         S1(i) = t(temp);
6.     end
7. end
```

7. kron.m

```
1. function K = kron(A,B)
2. [ma,na] = size(A);
3. [mb,nb] = size(B);
4.
5. if ~issparse(A) && ~issparse(B)
6.
7.    [ia,ib] = meshgrid(1:ma,1:mb);
8.    [ja,jb] = meshgrid(1:na,1:nb);
9.    K = A(ia,ja).*B(ib,jb);
10.
11.             else
12.
13.               [ia,ja,sa] = find(A);
14.               [ib,jb,sb] = find(B);
15.               ia = ia(:); ja = ja(:); sa = sa(:);
16.               ib = ib(:); jb = jb(:); sb = sb(:);
17.               ka = ones(size(sa));
18.               kb = ones(size(sb));
19.               t = mb*(ia-1)';
20.               ik = t(kb,:)+ib(:,ka);
21.               t = nb*(ja-1)';
22.               jk = t(kb,:)+jb(:,ka);
23.               K = sparse(ik,jk,sb*sa.',ma*mb,na*nb);
24.
25.             end
```

8. Metropolis.m

```
function [S,R] = Metropolis(S1,S2,N,max_M,A_new,W,t)
T1 = reshape(S1,[max_M,N/4]);
T1 = T1';
T2 = reshape(S2,[max_M,N/4]);
T2 = T2';
T11 = zeros(1,N/4);
T12 = zeros(1,N/4);
for i = 1:N/4
    T11(i) = prod(T1(i,:));
        T12(i) = prod(T2(i,:));
    end
    Tr1 = zeros(N,N);
    Tr2 = zeros(N,N);
    for j = 1:N/2
        Tr1(j,j) = T11(1);
        Tr2(j,j) = T12(1);
    end
    for j = 1:N/4
        Tr1(j+N/2,j+N/2) = T11(j);
        Tr1(j+N*3/4,j+N*3/4) = T11(j);
        Tr2(j+N/2,j+N/2) = T12(j);
        Tr2(j+N*3/4,j+N*3/4) = T12(j);
    end

    A1 = A_new{1}*W*Tr1;
    A2 = A_new{1}*W*Tr2;
    for j = 3:length(A_new)
        A1 = A1*A_new{j};
        A2 = A2*A_new{j};
    end
    DFT = dftmtx(N);
    FN = (1/sqrt(N))*DFT;
    [beta11,beta21] = find_best_beta(FN,A1,N);
    [beta12,beta22] = find_best_beta(FN,A2,N);

    R1 = rmse2(FN,A1,beta21,N);
    R2 = rmse2(FN,A2,beta22,N);

    dC = R2 - R1;
    if dC<0
    S = S2;
    R = R2;
    elseif exp(-dC/t)>=rand
    S = S2;
```

```
45.             R = R2;
46.             else
47.             S = S1;
48.             R = R2;
49.             end
```

9. Metropolis2.m

```
1. function [S,R] = Metropolis2(S1,S2,N,FN,max_M,P1,P2,P3,diagA,diagC,diagD,W,t)
2. T1 = reshape(S1,[max_M,N/4]);
3. T1 = T1';
4. T2 = reshape(S2,[max_M,N/4]);
5. T2 = T2';
6. T11 = zeros(1,N/4);
7. T12 = zeros(1,N/4);
8. for i = 1:N/4
9.         T11(i) = prod(T1(i,:));
10.                 T12(i) = prod(T2(i,:));
11.             end
12.             Tr1 = zeros(N,N);
13.             Tr2 = zeros(N,N);
14.             for j = 1:N/2
15.                 Tr1(j,j) = T11(1);
16.                 Tr2(j,j) = T12(1);
17.             end
18.             for j = 1:N/4
19.                 Tr1(j+N/2,j+N/2) = T11(j);
20.                 Tr1(j+N*3/4,j+N*3/4) = T11(j);
21.                 Tr2(j+N/2,j+N/2) = T12(j);
22.                 Tr2(j+N*3/4,j+N*3/4) = T12(j);
23.             end
24.             T_1 = zeros(4*N,4*N);
25.             T_2 = zeros(4*N,4*N);
26.             for i = 1:4
27.                 T_1(N*(i-1)+1:N*i,N*(i-1)+1:N*i) = Tr1;
28.                 T_2(N*(i-1)+1:N*i,N*(i-1)+1:N*i) = Tr2;
29.             end
30.             A1 = P1*P2*P3*diagA*W*T_1*diagC*diagD;
31.             A2 = P1*P2*P3*diagA*W*T_2*diagC*diagD;
32.             [beta11,beta21] = find_best_beta(FN,A1,4*N);
33.             [beta12,beta22] = find_best_beta(FN,A2,4*N);
34.             R1 = rmse2(FN,A1,beta21,4*N);
35.             R2 = rmse2(FN,A2,beta22,4*N);
36.             dC = R2 - R1;
```

```
37.             if dC<0
38.                 S = S2;
39.                 R = R2;
40.             elseif exp(-dC/t)>=rand
41.                 S = S2;
42.                 R = R2;
43.             else
44.                 S = S1;
45.                 R = R2;
46.             end
```

10.new_answer.m

```
1. function S2 = new_answer(S1,t)
2.     temp11 = randi(length(S1));
3.     temp12 = randi(length(S1));
4.
5.     S2 = S1;
6.     temp21 = randi(length(t));
7.     temp22 = randi(length(t));
8.
9.     S2(temp11) = t(temp21);
10.                S2(temp12) = t(temp22);
11.
12.            end
```

11.Q1.M

```
1. clear
2. clc
3. close all
4. %% 测试 beta1 的效果
5. % 当 t=1 时，
6. t = 3;
7. N = 2^t;
8. Astar = get_fft_matrix(1/2,N);
9. A = flattenCellArray(Astar);
10.            w = exp(-(2*pi*1i)/N);
11.            DFT = dftmtx(N);
12.            FN = (1/sqrt(N))*DFT;
13.            Ax = eye(N);
14.            for i = 1:length(A)
15.                Ax = Ax*A{i};
16.            end
17.            [beta1,beta2] = find_best_beta(FN,Ax,N);
```

```
r1 = [];
r2 = [];
for b = 0.01:0.01:7
    r1 = [r1,rmse1(FN,Ax,b,N)];
    r2 = [r2,rmse2(FN,Ax,b,N)];
end
plot(0.01:0.01:7,r1);
hold on
plot(0.01:0.01:7,r2);
best_mse1 = rmse1(FN,Ax,beta1,N);
best_mse2 = rmse2(FN,Ax,beta2,N);
hold on
plot(beta1,best_mse1,'o','Color','r');
hold on
plot(beta2,best_mse2,'o','Color','g');
legend("\beta1","\beta2","best \beta1","best \beta2","FontSize",10)
xlabel("\beta","FontName","Times New Roman","FontSize",14)
ylabel("RMSE","FontName","Times New Roman","FontSize",14)
title("t="+string(t)+"时,\beta 与 RMSE 的关系","FontSize",16)
grid on

%% 结果分析
clear
clc
close all
r1 = zeros(1,8);
r2 = zeros(1,8);
best_beta1 = zeros(1,8);
best_beta2 = zeros(1,8);
cpx = zeros(1,8);
for t = 1:8
    N = 2^t;
    Astar = get_fft_matrix(1/2,N);
    A = flattenCellArray(Astar);
    w = exp(-(2*pi*1i)/N);
    DFT = dftmtx(N);
    FN = (1/sqrt(N))*DFT;

    A_new = {};
    for i = 1:t-1
        A_new{i} = A{i};
    end
    A_new{end+1} = A{t};
    for i = 1:t+1
```

```
            A_new{end} = A_new{end}*A{t+i};
        end
        Ax = eye(N);
        total_L = 0;
        for i = 1:length(A_new)
            [~,L] = compute_cpx(Ax,A_new{i},N);
            Ax = Ax*A_new{i};
            total_L = total_L + L;
        end
        cpx(t) = 16*total_L;
        [best_beta1(t),best_beta2(t)] = find_best_beta(FN,Ax,N);
        r1(t) = rmse1(FN,Ax,best_beta1(t),N);
        r2(t) = rmse2(FN,Ax,best_beta2(t),N);
    end

```

12.Q2.m

```
clear
clc
close all
%% 设置初始条件
t = 3;
N = 2^t;
Astar = get_fft_matrix(1/2,N);
A = flattenCellArray(Astar);
A_new = {};
    for i = 1:t-1
        A_new{i+1} = A{i};
    end
    a = zeros(N);
    for i = 1:N/2
        a(i,i) = 1;
        a(i+N/2,i) = 1;
        a(i,i+N/2) = 1;
        a(i+N/2,i+N/2) = -1;
    end
    A_new{1} = a;
    w = exp(-(2*pi*1i)/N);
    b = zeros(N);
    for i = 1:N/2
        b(i,i) = 1;
        b(i+N/2,i+N/2) = w^(i-1);
    end
```

```
A_new{2} = b;
A_new{end+1} = A{t};
for i = 1:t+1
    A_new{end} = A_new{end}*A{t+i};
end
w = exp(-(2*pi*1i)/N);
DFT = dftmtx(N);
FN = (1/sqrt(N))*DFT;

%% 寻找最优分解向量
max_k = 3;
Q = [-4 -2 -1 0 -4 -2 -1 0];
best_des = {};
for i = 1:N/2
    best_des{1,i} = [0 0;0 0;0 0];
end
for i = 1:N/2
    min_arg{}
turn = zeros(1,N/2);    % 旋转矩阵
for i = 1:length(turn)
    turn(i) = b(i+N/2,i+N/2);
end
for i = 1:length(Q)
    for j = 1:length(Q)
        for k = 1:length(Q)

        end
    end
end
end
a = 1; % 3 + 2i 表示复数 3 + 2i
b = -1i; % -1 - 4i 表示复数 -1 - 4i
% 计算复数向量的相位角
theta1 = angle(a); % 向量 a 的相位角
theta2 = angle(b); % 向量 b 的相位角
% 计算夹角
angle_degrees = rad2deg(theta2 - theta1); % 转换为角度

% 打印夹角
fprintf('夹角：%f 度\n', angle_degrees);

% Ax = eye(N);
% cpx = 0;
% for i = 1:length(A)
```

```
71.         %     cpx = cpx + compute_cpx(Ax,A{i},N);
72.         %     Ax = Ax*A{i};
73.         % end
```

13.Q3.m

```
1. clear
2. clc
3. close all
4. %% 设置初始条件
5. t = 3;
6. N = 2^t;
7. Astar = get_fft_matrix(1/2,N);
8. A = flattenCellArray(Astar);
9. A_new = {};
10.         for i = 1:t-1
11.             A_new{i+1} = A{i};
12.         end
13.         a = zeros(N);
14.         for i = 1:N/2
15.             a(i,i) = 1;
16.             a(i+N/2,i) = 1;
17.             a(i,i+N/2) = 1;
18.             a(i+N/2,i+N/2) = -1;
19.         end
20.         A_new{1} = a;
21.         w = exp(-(2*pi*1i)/N);
22.         b = zeros(N);
23.         for i = 1:N/2
24.             b(i,i) = 1;
25.             b(i+N/2,i+N/2) = w^(i-1);
26.         end
27.         A_new{2} = b;
28.         A_new{end+1} = A{t};
29.         for i = 1:t+1
30.             A_new{end} = A_new{end}*A{t+i};
31.         end
32.         w = exp(-(2*pi*1i)/N);
33.         DFT = dftmtx(N);
34.         FN = (1/sqrt(N))*DFT;
35.
36.         %% 寻找最优分解向量
37.         max_k = 3;
38.         Q = [0 1 -1 2 -2 4 -4];
```

```
best_des = {};
for i = 1:N/2
    best_des{1,i} = [1 0;1 0;1 0];
end

target = zeros(1,N/2);    % 旋转矩阵
for i = 1:length(target)
    target(i) = b(i+N/2,i+N/2);
end
min_arg = zeros(1,N/2);
for i = 1:N/2
    best_ang = des2ang(best_des{1,i});
    theta1 = angle(best_ang);
    theta2 = angle(target(i));
    min_arg(i) = abs(rad2deg(theta2 - theta1));
end
for i1 = 1:length(Q)
    for i2 = 1:length(Q)
        for j1 = 1:length(Q)
            for j2 = 1:length(Q)
                for k1 = 1:length(Q)
                    for k2 = 1:length(Q)
                        if(norm(des2ang([Q(i1),Q(i2);Q(j1),Q(j2);Q(k1),Q(k2)])))
                            for i = 1:N/2
                                best_ang = des2ang([Q(i1),Q(i2);Q(j1),Q(j2);Q(k1),Q(k2)]);
                                theta1 = angle(best_ang);
                                theta2 = angle(target(i));
                                ang = abs(rad2deg(theta2 - theta1));
                                if(ang<min_arg(i))
                                    best_des{1,i} = [Q(i1),Q(i2);Q(j1),Q(j2);Q(k1),Q(k2)];
                                    min_arg(i) = ang;
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end


%% 最优偶矩阵求解
max_M = 3;
```

```
w2 = zeros(1,N/2);    % 分解后的向量
O = zeros(1,N/2);     % 模长
for i = 1:N/2
    w2(i) = des2ang(best_des{1,i});
    O(i) = norm(w2(i));
end
W = zeros(N,N);    % 按照向量拟合的矩阵
for i = 1:N/2
    W(i,i) = 1;
    W(i+N/2,i+N/2) = w2(i);
end

t = [1,2,5,17];
w2 = zeros(1,N/4*max_M);
t0 = 1000; %初始温度
tend = 1e-3;%终止温度
L = 100; %链长
q = 0.98; %降温速度
S1 = init(N/4,t,max_M);
syms Time
Time = ceil(double(solve(t0*(q)^Time == tend)));
count=0; %迭代计数
Obj = zeros(Time,1); %目标值矩阵初始化
track = zeros(Time,(N/4)*max_M); %每代的最优 x 矩阵初始化
best_T1 = ones(1,N/4);
while t0>tend
    count = count+1;
    temp = zeros(L,(N/4)*max_M+1);
    for k = 1:L
        S2 = new_answer(S1,t);%产生新解
        [S1,R] = Metropolis(S1,S2,N,max_M,A_new,W,t0);
        temp(k,:) = [S1 R];
    end
     % 记录每次迭代过程的最优路线
    [d0,index] = min(temp(:,end)); %找出当前温度下最优路线
    if count==1 || d0 < Obj(count-1)
        Obj(count) = d0;
        best_T1 = temp(index,1:end-1);
    else
        Obj(count) = Obj(count-1);
    end
    track(count,:) = temp(index,1:end-1);
    t0=q*t0; %降温
    fprintf(1,'%d\n',count) %输出当前迭代次数
```

```
end
figure
plot(1:count,Obj)
title("模拟退火算法迭代过程")
xlabel("迭代次数")
ylabel("目标函数")
grid on

T1 = reshape(best_T1,[max_M,N/4]);
T1 = T1';

for i = 1:N/4
    T11(i) = prod(T1(i,:));
end
Tr1 = zeros(N,N);
for j = 1:N/2
    Tr1(j,j) = T11(1);
end
for j = 1:N/4
    Tr1(j+N/2,j+N/2) = T11(j);
    Tr1(j+N*3/4,j+N*3/4) = T11(j);
end
L = 0;
for i = 1:length(best_T1)
    if(best_T1(i)~=1 && best_T1(i)~=2)
        if(i<=max_M)
            L = L+1+N/2;
        else
            L = L+2;
        end
    end
end


A1 = A_new{1}*W*Tr1;
for j = 3:length(A_new)
    A1 = A1*A_new{j};
end
DFT = dftmtx(N);
FN = (1/sqrt(N))*DFT;
[beta11,beta21] = find_best_beta(FN,A1,N);
best_r = rmse2(FN,A1,beta21,N);
C = L*3;
```

14.Q4.m

```
clear
clc
close all
%% 设置初始条件
t = 3;
N = 2^t;
Astar = get_fft_matrix(1/2,N);
A = flattenCellArray(Astar);
A_new = {};
    for i = 1:t-1
        A_new{i+1} = A{i};
    end
    a = zeros(N);
    for i = 1:N/2
        a(i,i) = 1;
        a(i+N/2,i) = 1;
        a(i,i+N/2) = 1;
        a(i+N/2,i+N/2) = -1;
    end
    A_new{1} = a;
    w = exp(-(2*pi*1i)/N);
    b = zeros(N);
    for i = 1:N/2
        b(i,i) = 1;
        b(i+N/2,i+N/2) = w^(i-1);
    end
    A_new{2} = b;
    A_new{end+1} = A{t};
    for i = 1:t+1
        A_new{end} = A_new{end}*A{t+i};
    end
    diagA = zeros(4*N);
    diagB = zeros(4*N);
    diagC = zeros(4*N);
    diagD = zeros(4*N);
    for i = 1:4
        diagA(N*(i-1)+1:N*i,N*(i-1)+1:N*i) = A_new{1};
        diagB(N*(i-1)+1:N*i,N*(i-1)+1:N*i) = A_new{2};
        diagC(N*(i-1)+1:N*i,N*(i-1)+1:N*i) = A_new{3};
        diagD(N*(i-1)+1:N*i,N*(i-1)+1:N*i) = A_new{4};
    end
    DFT4 = dftmtx(4);
```

```
F4 = (1/sqrt(4))*DFT4;
DFT8 = dftmtx(8);
F8 = (1/sqrt(8))*DFT8;
FN = kron(F4,F8);
I = zeros(8,8);
for i = 1:8
    I(i,i) = 1;
end
P1 = zeros(32,32);
P1(1:8,1:8) = I;
P1(1:8,17:24) = I;
P1(9:16,9:16) = I;
P1(9:16,25:32) = -1i*I;
P1(17:24,1:8) = I;
P1(17:24,17:24) = -I;
P1(25:32,9:16) = I;
P1(25:32,25:32) = 1i*I;
P2 = zeros(32,32);
P2(1:8,1:8) = I;
P2(1:8,9:16) = I;
P2(9:16,1:8) = I;
P2(9:16,9:16) = -I;
P2(17:24,17:24) = I;
P2(17:24,25:32) = I;
P2(25:32,17:24) = I;
P2(25:32,25:32) = -I;
P3 = zeros(32,32);
P3(1:8,1:8) = I;
P3(9:16,17:24) = I;
P3(17:24,9:16) = I;
P3(25:32,25:32) = I;
%% 最优偶矩阵求解
w2 = [1,1-1i,-1i,-1-1i];
W1 = zeros(N,N); % 按照向量拟合的矩阵
for i = 1:N/2
    W1(i,i) = 1;
    W1(i+N/2,i+N/2) = w2(i);
end
W = zeros(4*N);
for i = 1:4
    W(N*(i-1)+1:N*i,N*(i-1)+1:N*i) = W1;
end
max_M = 5;
t = [1,2,5,17];
```

```
w2 = zeros(1,N/4*max_M);
t0 = 10000; %初始温度
tend = 1e-6;%终止温度
L = 50; %链长
q = 0.98; %降温速度
S1 = init(N/4,t,max_M);
syms Time
Time = ceil(double(solve(t0*(q)^Time == tend)));
count=0; %迭代计数
Obj = zeros(Time,1); %目标值矩阵初始化
track = zeros(Time,(N/4)*max_M); %每代的最优 x 矩阵初始化
best_T1 = ones(1,N/4);
while t0>tend
count = count+1;
temp = zeros(L,(N/4)*max_M+1);
for k = 1:L
    S2 = new_answer(S1,t);%产生新解
    [S1,R] = Metropolis2(S1,S2,N,FN,max_M,P1,P2,P3,diagA,diagC,diagD,W,t0);
    temp(k,:) = [S1 R];
end
% 记录每次迭代过程的最优
[d0,index] = min(temp(:,end)); %找出当前温度下最优路线
if count==1 || d0 < Obj(count-1)
    Obj(count) = d0;
    best_T1 = temp(index,1:end-1);
else
    Obj(count) = Obj(count-1);
end
  track(count,:) = temp(index,1:end-1);
  t0=q*t0; %降温
  fprintf(1,'%d\n',count) %输出当前迭代次数
end
figure
plot(1:count,Obj)
title("问题四双扰动模拟退火算法迭代过程(max_q=3)")
xlabel("迭代次数")
ylabel("目标函数(RMSE)")
grid on
T1 = reshape(best_T1,[max_M,N/4]);
T1 = T1';
for i = 1:N/4
    T11(i) = prod(T1(i,:));
end
Tr1 = zeros(N,N);
```

```
131.        for j = 1:N/2
132.            Tr1(j,j) = T11(1);
133.        end
134.        for j = 1:N/4
135.            Tr1(j+N/2,j+N/2) = T11(j);
136.            Tr1(j+N*3/4,j+N*3/4) = T11(j);
137.        end
138.        T_1 = zeros(4*N,4*N);
139.        for i = 1:4
140.            T_1(N*(i-1)+1:N*i,N*(i-1)+1:N*i) = Tr1;
141.        end
142.        L = 0;
143.        for i = 1:length(best_T1)
144.            if(best_T1(i)~=1 && best_T1(i)~=2)
145.                if(i<=max_M)
146.                    L = L+(1+N/2)*4;
147.                else
148.                    L = L+2*4;
149.                end
150.            end
151.        end
152.        A1 = P1*P2*P3*diagA*T_1*W*diagC*diagD;
153.        [beta11,beta21] = find_best_beta(FN,A1,4*N);
154.        best_r = rmse2(FN,A1,beta21,4*N);
155.        C = L*3;
```

15.Q5.m

```
1. clear
2. clc
3. close all
4. %% 设置初始条件
5. t = 3;
6. N = 2^t;
7. Astar = get_fft_matrix(1/2,N);
8. A = flattenCellArray(Astar);
9. A_new = {};
10.        for i = 1:t-1
11.            A_new{i+1} = A{i};
12.        end
13.        a = zeros(N);
14.        for i = 1:N/2
15.            a(i,i) = 1;
16.            a(i+N/2,i) = 1;
```

```
        a(i,i+N/2) = 1;
        a(i+N/2,i+N/2) = -1;
    end
    A_new{1} = a;
    w = exp(-(2*pi*1i)/N);
    b = zeros(N);
    for i = 1:N/2
        b(i,i) = 1;
        b(i+N/2,i+N/2) = w^(i-1);
    end
    A_new{2} = b;
    A_new{end+1} = A{t};
    for i = 1:t+1
        A_new{end} = A_new{end}*A{t+i};
    end
    w = exp(-(2*pi*1i)/N);
    DFT = dftmtx(N);
    FN = (1/sqrt(N))*DFT;
    Result = {};
    jishu = 1;
    %% 寻找最优分解向量
    for max_q = 1:8
        max_k = 3;
        Q = [0];
        for i = 1:max_q
            Q = [Q 2^(i-1) -2^(i-1)];
        end
        best_des = {};
        for i = 1:N/2
            best_des{1,i} = [1 0;1 0;1 0];
        end

        target = zeros(1,N/2);    % 旋转矩阵
        for i = 1:length(target)
            target(i) = b(i+N/2,i+N/2);
        end
        min_arg = zeros(1,N/2);
        for i = 1:N/2
            best_ang = des2ang(best_des{1,i});
            theta1 = angle(best_ang);
            theta2 = angle(target(i));
            min_arg(i) = abs(rad2deg(theta2 - theta1));
        end
        for i1 = 1:length(Q)
```

```
                for i2 = 1:length(Q)
                    for j1 = 1:length(Q)
                        for j2 = 1:length(Q)
                            for k1 = 1:length(Q)
                                for k2 = 1:length(Q)
                                    if(norm(des2ang([Q(i1),Q(i2);Q(j1),Q(j2);Q(k1),Q(k2)])))
                                        for i = 1:N/2
                                            best_ang = des2ang([Q(i1),Q(i2);Q(j1),Q(j2);Q(k1),Q(k2)]);
                                            theta1 = angle(best_ang);
                                            theta2 = angle(target(i));
                                            ang = abs(rad2deg(theta2 - theta1));
                                            if(ang<min_arg(i))
                                                best_des{1,i} = [Q(i1),Q(i2);Q(j1),Q(j2);Q(k1),Q(k2)];
                                                min_arg(i) = ang;
                                            end
                                        end
                                    end
                                end
                            end
                        end
                    end
                end
            end

            %% 最优偶矩阵求解
            for max_M = 1:30
                w2 = zeros(1,N/2);    % 分解后的向量
                O = zeros(1,N/2);     % 模长
                for i = 1:N/2
                    w2(i) = des2ang(best_des{1,i});
                    O(i) = norm(w2(i));
                end
                W = zeros(N,N);    % 按照向量拟合的矩阵
                for i = 1:N/2
                    W(i,i) = 1;
                    W(i+N/2,i+N/2) = w2(i);
                end
                Q(find(Q==0)) = [];   %去除 0 元素
                Q =  unique(Q);
                t = Q;
                for i = 1:length(Q)
                    for j = i+1:length(Q)
                        t = [t Q(i)^2+Q(j)^2];
```

```
105.                 end
106.             end
107.             w2 = zeros(1,N/4*max_M);
108.             t0 = 1000; %初始温度
109.             tend = 1e-3;%终止温度
110.             L = 10; %链长
111.             q = 0.98; %降温速度
112.             S1 = init(N/4,t,max_M);
113.             syms Time
114.             Time = ceil(double(solve(t0*(q)^Time == tend)));
115.             count=0; %迭代计数
116.             Obj = zeros(Time,1); %目标值矩阵初始化
117.             track = zeros(Time,(N/4)*max_M); %每代的最优 x 矩阵初始化
118.             best_T1 = ones(1,N/4);
119.             while t0>tend
120.                 count = count+1;
121.                 temp = zeros(L,(N/4)*max_M+1);
122.                 for k = 1:L
123.                     S2 = new_answer(S1,t);%产生新解
124.                     [S1,R] = Metropolis(S1,S2,N,max_M,A_new,W,t0);
125.                     temp(k,:) = [S1 R];
126.                 end
127.                  % 记录每次迭代过程的最优路线
128.                 [d0,index] = min(temp(:,end)); %找出当前温度下最优路线
129.                 if count==1 || d0 < Obj(count-1)
130.                     Obj(count) = d0;
131.                     best_T1 = temp(index,1:end-1);
132.                 else
133.                     Obj(count) = Obj(count-1);
134.                 end
135.                 track(count,:) = temp(index,1:end-1);
136.                 t0=q*t0; %降温
137.                 fprintf(1,'%d\n',count) %输出当前迭代次数
138.             end
139.         %     figure
140.         %     plot(1:count,Obj)
141.         %     title("模拟退火算法迭代过程")
142.         %     xlabel("迭代次数")
143.         %     ylabel("目标函数")
144.         %     grid on
145.
146.             T1 = reshape(best_T1,[max_M,N/4]);
147.             T1 = T1';
148.
```

```
149.                for i = 1:N/4
150.                    T11(i) = prod(T1(i,:));
151.                end
152.                Tr1 = zeros(N,N);
153.                for j = 1:N/2
154.                    Tr1(j,j) = T11(1);
155.                end
156.                for j = 1:N/4
157.                    Tr1(j+N/2,j+N/2) = T11(j);
158.                    Tr1(j+N*3/4,j+N*3/4) = T11(j);
159.                end
160.                L = 0;
161.                for i = 1:length(best_T1)
162.                    if(best_T1(i)~=1 && best_T1(i)~=2)
163.                        if(i<=max_M)
164.                            L = L+1+N/2;
165.                        else
166.                            L = L+2;
167.                        end
168.                    end
169.                end
170.
171.
172.                A1 = A_new{1}*W*Tr1;
173.                for j = 3:length(A_new)
174.                    A1 = A1*A_new{j};
175.                end
176.                DFT = dftmtx(N);
177.                FN = (1/sqrt(N))*DFT;
178.                [beta11,beta21] = find_best_beta(FN,A1,N);
179.                best_r = rmse2(FN,A1,beta21,N);
180.                C = L*max_q;
181.                Result{jishu} = [best_r,C,max_q,max_M];
182.                jishu = jishu+1;
183.            end
184.        end
185.        %% 结果分析
186.        obj1 = zeros(1,length(Result));
187.        obj2 = zeros(1,length(Result));
188.        for i = 1:length(Result)
189.            obj1(1,i) = Result{i}(1);
190.            obj2(1,i) = Result{i}(2);
191.        end
192.        plot(obj1,obj2,'o');
```

```
193.        xlabel("RMSE");
194.        ylabel("硬件复杂度 C");
195.        title("所有方案中两个目标函数的关系");
196.        grid on
```

16.rmse.m

```
1.    function r = rmse(FN,moni,beta,N)
2.    dis = beta*FN-moni;
3.    mse = norm(dis,"fro");
4.    r = mse/N;
5.    end
```

17.rmse1.m

```
1. function r = rmse1(FN,moni,beta,N)
2. dis = beta*FN-moni;
3. mse = norm(dis,"fro");
4. r = mse/N;
5. end
```

18.rmse2.m

```
1.    function r = rmse2(FN,moni,beta,N)
2.    dis = FN-beta*moni;
3.    mse = norm(dis,"fro");
4.    r = mse/N;
5.    end
```