基于差分图算法的噪声图像相位恢复

摘要

本文针对含噪声频谱的相位恢复和图像复原问题, 首先构建了基于频谱区域保护的全变分和 3D 块匹配去噪模型, 其次利用差分图(DM)算法进行相位恢复, 在此基础上提出了基于自监督峰值信噪比(S-PSNR)和自监督结构相似性(S-SSIM)的评价模型, 最后定量分析过采样率、支撑域和空域结构连续性对恢复概率的影响.

针对问题一: 我们首先通过离群点分析和 Kolmogorov-Smirnov 单样本检验, 得出噪声服从 $\sigma = 3.6936$ 的瑞利分布的结论. 其次以 99%的能量阈值划分低频区和高频区, 构建了基于区域保护的全变分去噪模型. 然后利用差分图相位恢复算法, 基于频谱幅度和随机初始化相位, 通过多次迭代实现图像重建, 有效剔除了叠加解. 在此基础上提出了基于 S-PSNR 和 S-SSIM 的评价模型, 计算得到恢复空域图像的 S-PSNR=24 dB, S-SSIM=0.57, 图像恢复效果良好. 最后我们分析了迭代次数和初始相位分布对恢复效果的影响, 得出两个结论: (1)增加迭代次数能够使恢复效果在提升的同时逐渐趋于平稳; (2)初始相位满足均匀分布时恢复效果更好.

针对问题二: 在问题一的基础上, 我们采用相同的处理思路, 构建了基于区域保护的 3D 块匹配去噪模型, 而后利用差分图算法进行相位恢复和图像重建, 最后基于 S-PSNR 和 S-SSIM 的评价模型给出图像恢复效果, 计算得到附件 2 的 S-PSNR=20.91 dB, S-SSIM=0.46, 附件 3 的 S-PSNR=26.62 dB, S-SSIM=0.75, 证明图像恢复效果良好.

针对问题三: 我们基于 Lena 图构建了包含不同类型、不同强度噪声的数据集, 从过采样率和支撑域构建(即先验信息)两个角度定量分析其对恢复概率的影响, 得出过采样率越高, 支撑域非中心对称性越强, 图像恢复概率越高的结论. 此外, 我们在考虑了实际中物像平面大小固定这一现实条件的基础上, 对图像进行分块恢复和重拼接, 并引入空域结构连续性这一先验, 进一步分析了过采样率对恢复概率的影响.

最后, 我们对提出的模型进行全面的评价: 本文的模型贴合实际, 能够在保留目标信息的同时滤除噪声, 在相位恢复中克服中心对称解问题, 具有实用性强, 算法效率高等特点.

关键词: 相位复原　图像去噪　全变分算法　BM3D 算法

目录

1 问题综述

1.1 问题背景

图像作为人类感知世界的视觉基础，是人类获取信息、表达信息和传递信息的重要手段. 其中，相位是蕴含图像信息的重要因素，包含约 3/4 的信息，而幅度仅包含 1/4 的信息[1]. 然而在许多实际问题中，例如相干衍射成像领域，由于光波的高频特性，幅度信息很容易被记录，而相位在光学器件采集过程中却难以获得或者会丢失大部分的信息，且利用全息技术实现相位捕捉也很困难. 在此情况下，仅根据缺乏相位信息的频谱复原的图像会完全失真，如图 2 所示.

此外，在图像的获取、传输、量化等过程中，由于受传感器材料特性、工作环境、电子器件和电路结构、不完善的传输媒介等影响，会不可避免地会引入噪声信号，其在图像上常表现为随机的像素亮度变化，使得图像的质量会有所降低. 因此，基于含噪声的强度或频谱信息来恢复相位，实现图像重构，已经引起了越来越多的关注.

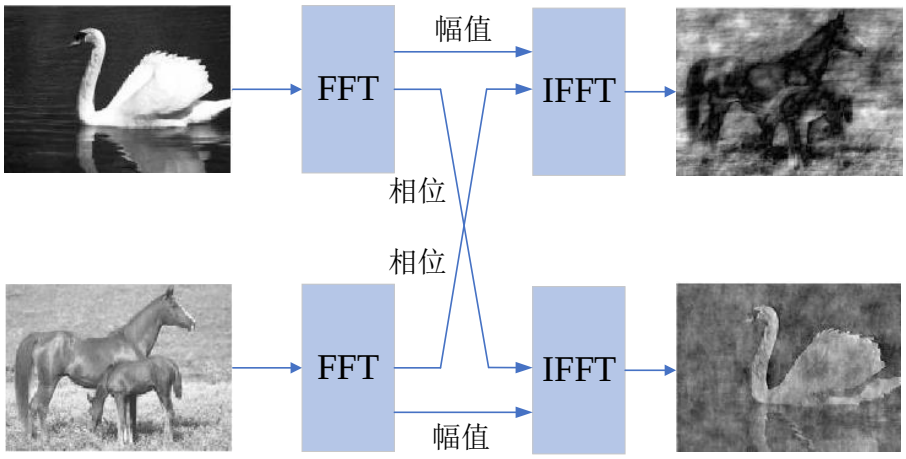
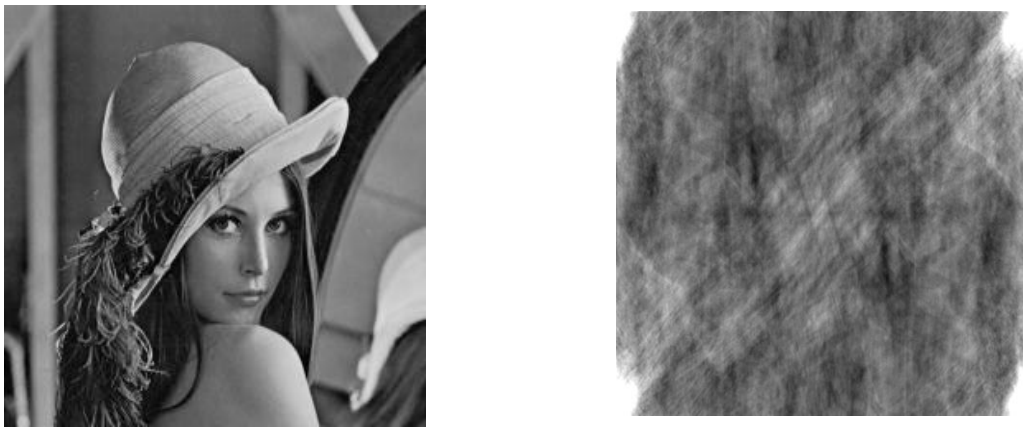


图 1 相位信息对图像恢复的影响



(a)经典 Lena 图　　(b)频域幅度谱的恢复结果

图 2 原图与仅基于频域幅度谱的恢复结果对比

相位复原算法主要分为两种[2]：第一种是利用图像在空间域和频域的反复迭代变换，并在每一步施加约束，最终实现相位复原的算法，即迭代法. Gerehberg 和 Saxton 提出的 GS 算法奠定了相位复原衍射迭代的基本模式，在 GS 算法的基础上，衍生出了 ER 算法、HIO 算法等；另一种是利用强度传输方程求解相位的方法，例如格林函数法、傅里叶变换法等.

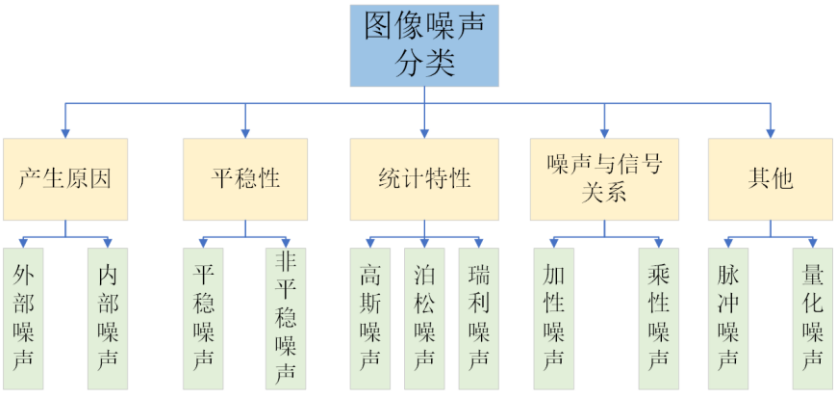


图 3 噪声分类

图像去噪(Image Denoising)的目的是根据不同的噪声类型、特点进行算法设计，尽可能滤除噪声信号，提升图像质量，便于对图像进行进一步处理和分析．当前，对图像去噪的算法研究已经较为成熟，可以分为两大类——基于人工特征的传统去噪方法和基于深度学习的去噪方法[1]．针对包含不同类型噪声的图像使用不同的方法会获得不同的去噪效果．传统的中值滤波、均值滤波等算法运算简单，但会对图像的细节和几何结构造成损坏[4]；NLM、BM3D 等方法利用自相似补丁能够在维持图像保真度的同时显著提升图像质量[5] [6]；基于深度学习的方法具有更强大的学习能力和更好的去噪效果，已成为当前的主流手段，但其需要大量图像作为数据库支撑，在本题仅由三张频谱数据的前提下并不适用．

1.2 问题提出

根据题目要求，我们需要从相位恢复模型、去噪模型构建、恢复效果评价、恢复影响因素(如过采样率、先验信息)等角度考虑，解决以下问题:

(1) 问题 1: 在没有频谱相位信息、噪声先验信息的前提下，针对附件 1 中仅含一种噪声的频谱，构建相应的去噪和恢复模型，而后设定相应的指标进行效果评价，并分析影响图像恢复效果的因素;
(2) 问题 2: 在没有频谱相位信息、噪声先验信息的前提下，针对附件 2 和附件 3 中包含多种噪声的频谱，构建相应的去噪和恢复模型，使其能在多种未知类型的噪声下均能产生较好的恢复效果;
(3) 问题 3: 通过自己构造含噪声的频谱信息数据，在不同的过采样率以及相关先验信息下，结合问题 1 和问题 2 中的模型，给出图像的恢复概率．

2 问题分析

2.1 问题一的分析

问题一要求我们在已知仅有一种噪声的前提下，仅通过频谱幅度恢复空域图像．可以将问题一的求解分为四步: 一是尽可能获取噪声先验信息，可以选取目标能量较少的高频区域，利用拟合度检验得到噪声的统计特性; 二是对频谱图采用相应的去噪算法进行降噪; 三是构建基于频谱幅度实现空域图像重建的相位恢复模型; 最后提出无参考图像的恢复效果评价指标，并定量分析不同参数对恢复效果的影响．

2.2 问题二的分析

问题二可以采取与问题一相同的处理思路，先进行频谱图去噪，再进行相位恢复．但由于频谱图均含有多种噪声，在去噪算法上需选取适应性更强、鲁棒性更好的算法．

2.3 问题三的分析

过采样是相位恢复的必要条件, 其通过在图像周围补零实现, 这些零值像素正是一种先验信息. 我们可以构造包含不同种类、强度噪声的数据集, 通过改变过采样率、支撑域形状等参数实现不同的过采样过程, 并基于问题一、二的模型进行图像恢复, 定量分析参数对恢复概率的影响.

3 模型假设与符号说明

3.1 模型基本假设

(1) 假设本题中除脉冲噪声以外, 所有噪声均为加性噪声. 加性噪声是指噪声与输入图像信号无关, 与信号直接相加的噪声; 乘性噪声指与图像信号有关, 与信号直接相乘的噪声. 脉冲噪声一般是量化成图像中的极限灰度, 在图像中表现为随机出现的黑点和白点. 本题中, 为简化分析, 我们假定其余噪声均为加性噪声.

(2) 假设每个图像像素上接收的噪声是独立同分布的, 不存在相关性.

(3) 假设入射光为相干光. 从目标上任意两点出发的散射光在成像面阵上的每个点都是相干叠加的, 采集到的图像才是完整频域信号. 若相干性不足, 仅能获取部分频谱信息, 就无法重建完整的空域图像.

(4) 假设待恢复的原始图像包含较丰富的结构信息.

3.2 符号说明

本文定义了如下 5 个使用次数较多的符号, 其余符号在使用时注明.

表 1 符号说明

符号	含义
$\mathbf{K}_l$	第 l 次迭代的空域图像
$\mathbf{G}_0$	真实空域图像
$\mathbf{G}_l$	第 l 次迭代的空域相位谱矩阵
$\mathbf{P}_l$	第 l 次迭代的频域幅度谱矩阵
$\mathbf{A}_0$	频域幅度谱矩阵
$\mathbf{S}$	支撑域掩膜矩阵

4 模型建立与求解

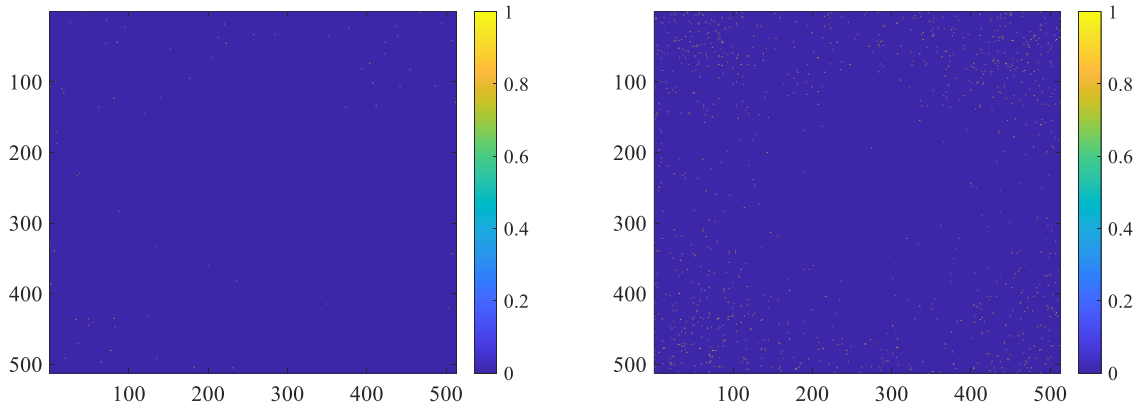
4.1 问题一的求解

4.1.1 基于 K-S 单样本检验的噪声类型判别

通过像素离群点分析, 我们首先排除脉冲噪声, 理由如下:

1、频域图像全局没有明显的随机峰值或近零值点;

2、幅度很小的像素点大多分布在高频区, 而幅度很大的像素点集中在低频区, 如图 4 所示. 而如果是脉冲噪声, 像素极值点应在低频区和高频区均随机出现.



(a) 幅度<0.25 的像素点分布　　(b) 幅度<1 的像素点分布

(c) 幅度>5000 的像素点分布　　(d) 幅度>6000 的像素点分布

图 4 附件 1 的像素离群点分析(中心化后频谱)

由于目标频谱主要分布在低频区，为了规避其影响，我们在选定区域掩膜时选取原始频谱图中心的 200×200 高频区域，如图所示. 而后统计出该区域的幅度直方图，利用 MATLAB 和 SPSS 软件进行 Kolmogorov-Smirnov 单样本检验，判别噪声分布.

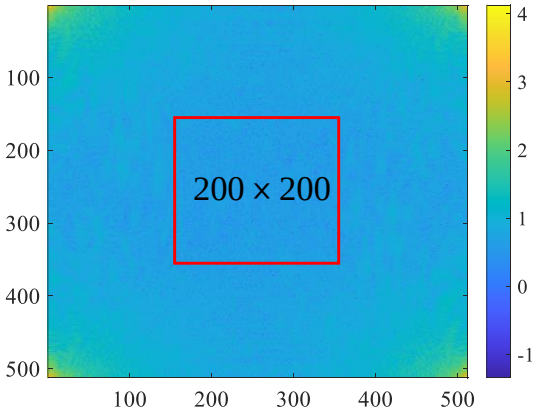


图 5 分析区域选定

对于正态分布和瑞利分布，利用 MATLAB 进行单样本检验的结果如下表所示:

表 2 正态和瑞利分布单样本检验

	瑞利分布	正态分布
h	0	1
p	0.1088	4.1574×10^{-35}

以正态分布为例:

原假设 H_0: 分布服从 $N(\mu,\sigma)$ 的正态分布;

备择假设 H_1: 分布不服从 $N(\mu,\sigma)$ 的正态分布.

h 为检验判决变量, h=0 代表接受 H_0, h=1 代表拒绝 H_0, 接受 H_1; p 为渐进显著性指数, 大于 0.05 时认为数据符合该分布函数, 且渐进指数越大, 拟合程度越高. 根据表 2 可知, 噪声服从瑞利分布, 不服从正态分布.

由于泊松变量必须为整数, 我们将其量化至 0 ~ 255 区间内的整数, 然后基于 SPSS 软件进行判别:

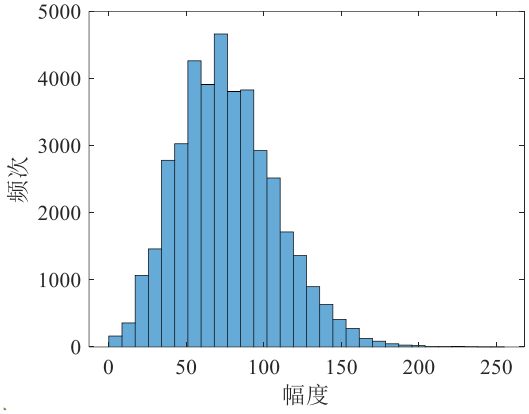


图 6 幅度化整后的直方图

表 3 Poisson 分布 K-S 检验

N		40401
Poisson 参数	均值	79.98
最极端差别	绝对值	0.347
	正	0.347
	负	-0.211
Kolmogorov-Smirnov Z		69.651
渐进显著性(双侧)		0.000

综上所示, 我们选择瑞利分布函数描述噪声分布, 其概率密度函数为

$$p(z)=\frac{z}{\sigma^2}e^{-z^2/(2\sigma^2)} \tag{1}$$

其中 $\sigma=3.6936$.

4.1.2 基于区域保护的全变分降噪模型

低频信息占据了目标能量的主体部分(高频信息体现为图像轮廓和细节), 而噪声是随机分布于频谱图之上. 如果我们直接对频谱图整体使用降噪算法, 则目标频谱在降噪后会使得傅里叶逆变换后的空域图产生较大的畸变, 丢失目标信息.

为解决上述问题, 一个直观的思路是通过设定门限, 先将 "低频区" 和 "高频区" 加以划分, 对 "高频区" 使用去噪算法, 而对 "低频区" 的主要信息加以保护. "低频区

" 和 "高频区" 可通过能量阈值η进行划分. 设定 "低频区" 边长为 R, 能量比例的阈值为η, 二者满足

$$\sum |f(x,y)|^2 = \eta E \tag{2}$$

其中$-R/2 \le x \le R/2$, $-R/2 \le y \le R/2$, E 为频谱图能量总和, R 为整数.

在频谱图上以零频点为中心, R 为边长的正方形区域为低频区, 其余为高频区. 设η=0.99, 计算可得 $R=50$. 可以理解为, 正方形区域内像素点能量之和占据了全图的 99%.

本文假设噪声均为加性噪声, 噪声频域图像的模型为:

$$F(x,y) = U(x,y) + n(x,y) \tag{3}$$

其中 $F(x,y)$ 为携带噪声的频域图像, $U(x,y)$ 为原始频域图像, $n(x,y)$ 为噪声.

基于全变分的图像去噪问题本质是求下式的最小化解[7]:

$$\min_u \int_\Omega |\nabla U| dxdy \tag{4}$$

其均值约束条件和方差约束条件分别为

$$\begin{cases} \int_\Omega U dxdy = \int_\Omega F dxdy \\ \dfrac{1}{|\Omega|}\int_\Omega (F-U)^2 dxdy = \sigma^2 \end{cases} \tag{5}$$

其中 σ^2 为噪声方差, $|\Omega|$ 为区域的面积.

通过拉格朗日乘子法, 将(4)转化为能量泛函的极小值问题:

$$\min E(U) = \alpha \int_\Omega |\nabla U| dxdy + \frac{\lambda}{2}\int_\Omega (U-F)^2 dxdy \tag{6}$$

其中 ∇U 为频域图像梯度; $\int_\Omega |\nabla U| dxdy$ 为频域图像的全变分, 为正则项; $\int_\Omega (U-F)^2 dxdy$ 表示去噪前后频谱的差异, 为频域图像保真项; α 和 λ 为调节正则项和保真项的系数.

(6)对应的 Euler-Lagrange 方程

$$\frac{\partial E(U)}{\partial U} = F_U - \frac{\partial}{\partial x}F_{U_x} - \frac{\partial}{\partial y}F_{U_y} = \lambda(U-F) - \nabla\bullet\left(\frac{\nabla U}{|\nabla U|}\right) = 0 \tag{7}$$

其中$\nabla\bullet$为散度运算符, $\nabla\bullet\left(\dfrac{\nabla U}{|\nabla U|}\right)$为扩散项, $\dfrac{1}{|\nabla U|}$为扩散系数.

若在图像中存在梯度为 0 的频点, 则将成为一个病态方程. 为避免这个问题, 我们利用正则化方法, 在全变分表达式分母中引入一个无穷小正数ε:

$$TV(U) = \int_\Omega |\nabla U| dxdy = \int_\Omega \sqrt{U_x^{\,2} + U_y^{\,2} + \varepsilon} dxdy, \varepsilon > 0 \tag{8}$$

4.1.3 基于差分图的相位恢复算法

差分图算法是一种针对一般约束满足问题的搜索算法, 综合了混合输入输出(HIO)算法和用于凸优化的 Douglas-Rachford 算法. 差分图算法以只包含幅度的频谱以及一个随机初始相位谱作为输入, 在空间谱与频谱之间反复迭代, 通过对非支撑域施加约束,

最终使得初始输入的随机相位谱近似收敛到频域的真实相位谱, 从而实现对图像空间谱的复原[8].

算法输入包括: 频谱幅度矩阵 $\mathbf{A}_0$, 支撑域掩膜矩阵 $\mathbf{S}$, (根据 $\mathbf{S}$ 算出掩膜矩阵$\mathbf{S}_2$, $\mathbf{S}_2 = 2\mathbf{S} - \mathbf{\Pi}$, 其中$\mathbf{\Pi}$为全 1 矩阵, 其维度与 $\mathbf{S}$ 相同), 均匀分布相位谱$\mathbf{G}_0$, 总迭代次数为 L, 具体流程如下:

Step 1 将迭代次数初始化$l = 0$; 将空域相位谱估计初始化为$\mathbf{G}_l = \mathbf{G}_0$;

Step 2 迭代次数$l = l + 1$;

Step 3 对估计的空域相位谱进行变换: $\mathbf{G}_l' = \mathbf{G}_l \odot \mathbf{S}_2$, 其中$\odot$为 Hadamard 积, 表示相位谱估计$\mathbf{G}_l$与掩膜矩阵$\mathbf{S}_2$逐元素相乘;

Step 4 更新频域相位谱估计$\mathbf{P}_l = \left[\text{fftshift}\left(\text{fft2}\left(\mathbf{G}_l' \right) \right) \right] \Big/ \left| \mathbf{P}_l \right|$, 即对空域相位谱估计$\mathbf{G}_l'$做二维离散傅里叶变换, 并进行中心化处理, 然后进行归一化;

Step 5 更新频域图像的频谱估计$\mathbf{F}_l = \mathbf{A}_0 \odot \mathbf{P}_l$;

Step 6 更新空域图像估计$\mathbf{K}_l = \text{ifft2}\left(\text{ifftshift}\left(\mathbf{F}_l \right) \right) - \mathbf{G}_l \odot \mathbf{S}$, 即对逆中心化的频谱估计$\mathbf{F}_l$做二维离散傅里叶逆变换再减去相位谱估计$\mathbf{G}_l$与掩膜矩阵$\mathbf{S}$的 Hadamard 积;

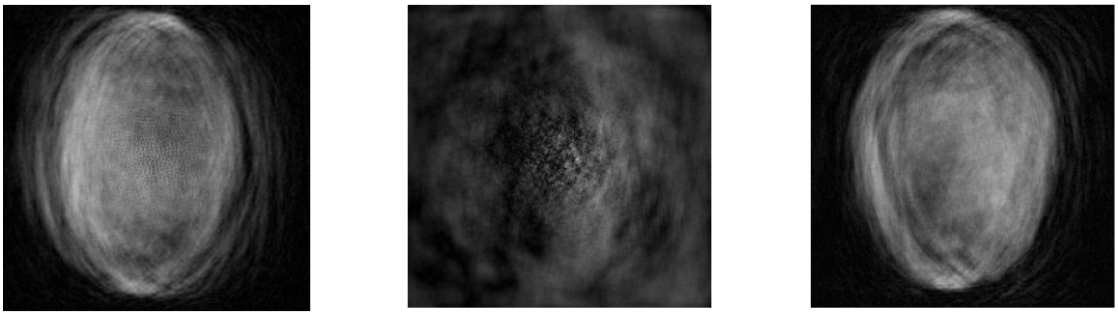
Step 7 更新空域图像的相位谱估计$\mathbf{G}_l = \mathbf{K}_l$;

Step 8 若$l < L$, 返回 Step 2, 否则结束循环. 最终输出空域图像的幅度谱估计为$\left| \mathbf{K}_l \right|$.

本题中, 支撑域掩膜矩阵 $\mathbf{S}$ 的维度为 512×512, 其中心 256×256 的元素为 1, 其余元素均为 0.

4.1.4 附件 1 图像恢复结果

我们对比了三种条件下的图像恢复效果, 分别为无降噪、无区域保护降噪(即对整体频谱图使用降噪算法)和基于区域保护的降噪(即只对 "高频区" 使用降噪算法), 结果如下.



(a) 无降噪的图像恢复结果　(b) 无区域保护的降噪后图像恢复结果　(c) 基于区域保护的降噪后图像恢复结果

图 7 空域图像恢复结果对比

由(a)、(c)对比可知, 经过降噪后的空域图像在保持了纹理、轮廓、亮度的基础上, 整体平滑度有所提升, 有效抑制了噪点; 由(b)、(c)对比可知, 直接对频谱整体进行降噪会使得目标信息基本丢失, 侧面证明了进行频谱区域保护的必要性.

附件 4 填入的是基于区域保护的降噪后图像恢复结果, 即图 7 (c).

4.1.5 基于 S-PSNR 和 S-SSIM 的评价模型

在典型的有监督图像恢复类问题(例如有监督图像超分辨问题、有监督图像降噪问题等)中, 一般通过计算恢复图像与理想图像的 PSNR 和 SSIM 来评价图像恢复质量的好坏. PSNR 是峰值信号的能量与噪声的平均能量之比; SSIM 是对样本之间的亮度、对比度和结构进行比较, 范围为 0 至 1, 越大代表图像越相似.

PSNR 的计算过程如下:

$$\mathrm{MSE}(\mathrm{G}_0,\mathrm{G}_1)=\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}[\mathbf{G}_1(i,j)-\mathbf{G}_0(i,j)]^2 \tag{9}$$

$$\mathrm{PSNR}(\mathrm{G}_0,\mathrm{G}_1)=10\cdot\log_{10}\left(\frac{MAX^2}{MSE}\right)=20\cdot\log_{10}\left(\frac{MAX^2}{\sqrt{MSE}}\right) \tag{10}$$

其中 MAX 为 G_0 的像素最大值.

SSIM 的计算过程为:

$$\mu_{\mathbf{G}_0}=\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\mathbf{G}_0(i,j) \tag{11}$$

$$\sigma_{\mathbf{G}_0}=\left[\frac{1}{M\times N-1}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(\mathbf{G}_0(i,j)-\mu_{\mathbf{G}_0}\right)^2\right]^{\frac{1}{2}} \tag{12}$$

$$\sigma_{\mathbf{G}_0\mathbf{G}_1}=\frac{1}{M\times N-1}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(\mathbf{G}_0(i,j)-\mu_{\mathbf{G}_0}\right)\left(\mathbf{G}_1(i,j)-\mu_{\mathbf{G}_1}\right) \tag{13}$$

亮度、对比度、结构相似性分别为

$$\mathrm{l}(\mathbf{G}_0,\mathbf{G}_1)=\frac{2\mu_{\mathbf{G}_0}\mu_{\mathbf{G}_1}+c_1}{\mu_{\mathbf{G}_0}^2+\mu_{\mathbf{G}_1}^2+c_1} \tag{14}$$

$$\mathrm{c}(\mathbf{G}_0,\mathbf{G}_1)=\frac{2\sigma_{\mathbf{G}_0}\sigma_{\mathbf{G}_1}+c_2}{\sigma_{\mathbf{G}_0}^2+\sigma_{\mathbf{G}_1}^2+c_2} \tag{15}$$

$$\mathrm{s}(\mathbf{G}_0,\mathbf{G}_1)=\frac{\sigma_{\mathbf{G}_0\mathbf{G}_1}+c_3}{\sigma_{\mathbf{G}_0}\sigma_{\mathbf{G}_1}+c_3} \tag{16}$$

其中 $c_1=(k_1L)^2$, $c_2=(k_2L)^2$, $c_3=c_2/2$ 均为常数, L 为像素值范围, $k_1=0.01$ 和 $k_2=0.03$ 为默认值. 则

$$\mathrm{SSIM}(\mathbf{G}_0,\mathbf{G}_1)=\left[l(\mathbf{G}_0,\mathbf{G}_1)^{\alpha}\cdot c(\mathbf{G}_0,\mathbf{G}_1)^{\beta}\cdot s(\mathbf{G}_0,\mathbf{G}_1)^{\gamma}\right] \tag{17}$$

通常设 $\alpha=\beta=\gamma=1$.

由于缺少真实空域图像 $\mathbf{G}_0$, 为了检验图像恢复的准确性, 我们借鉴深度学习中的自监督思想, 提出使用自监督峰值信噪比(S-PSNR)和自监督结构相似度(S-SSIM)的评价模型. 设第 n 轮迭代后得到空域幅度矩阵 $\mathbf{G}_n$. 为了评价恢复图像 $\mathbf{G}_1$ 的恢复效果, 需要进行 n 轮自监督迭代. 定义用于评价图像 $\mathbf{G}_1$ 的图像恢复效果的 S-PSNR 和 S-SSIM 公式如下:

$$\text{S-PSNR}(\mathbf{G}_1)=\mathrm{PSNR}(\mathbf{G}_1,\mathbf{G}_n) \tag{18}$$

$$\text{S-SSIM}(\mathbf{G}_1)=\mathrm{SSIM}(\mathbf{G}_1,\mathbf{G}_n) \tag{19}$$

这是因为只有当首轮迭代的结果$\mathbf{G}_1$中保留真实图像$\mathbf{G}_0$的结构信息越多时，经过多轮迭代后才能更有可能将这些信息保留并传递下去，从而 S-PSNR($\mathbf{G}_1$)和 S-SSIM($\mathbf{G}_1$)会更大.

基于 S-PSNR 和 S-SSIM 的自监督评价模型计算过程如下(以 n=3 为例):

Step 1 第 1 轮迭代: 由附件给出的频域幅度矩阵$\mathbf{P}_0$，通过上述图像恢复模型，得到空域幅度矩阵$\mathbf{G}_1$;

Step 2 第 2 轮迭代: 对$\mathbf{G}_1$进行过采样和 FFT 变换，得到频域幅度矩阵$\mathbf{P}_1$；由频域幅度矩阵$\mathbf{P}_1$得到空域幅度矩阵$\mathbf{G}_2$;

Step 3 第 3 轮迭代: 对$\mathbf{G}_2$进行过采样和 FFT 变换，得到频域幅度矩阵$\mathbf{P}_2$；由频域幅度矩阵$\mathbf{P}_2$得到空域幅度矩阵$\mathbf{G}_3$;

Step 4 计算 S-PSNR($\mathbf{G}_1$)和 S-SSIM($\mathbf{G}_1$)，其值越高，则可以间接证明空域图像$\mathbf{G}_1$越接近真实图像$\mathbf{G}_0$，即恢复效果越好.

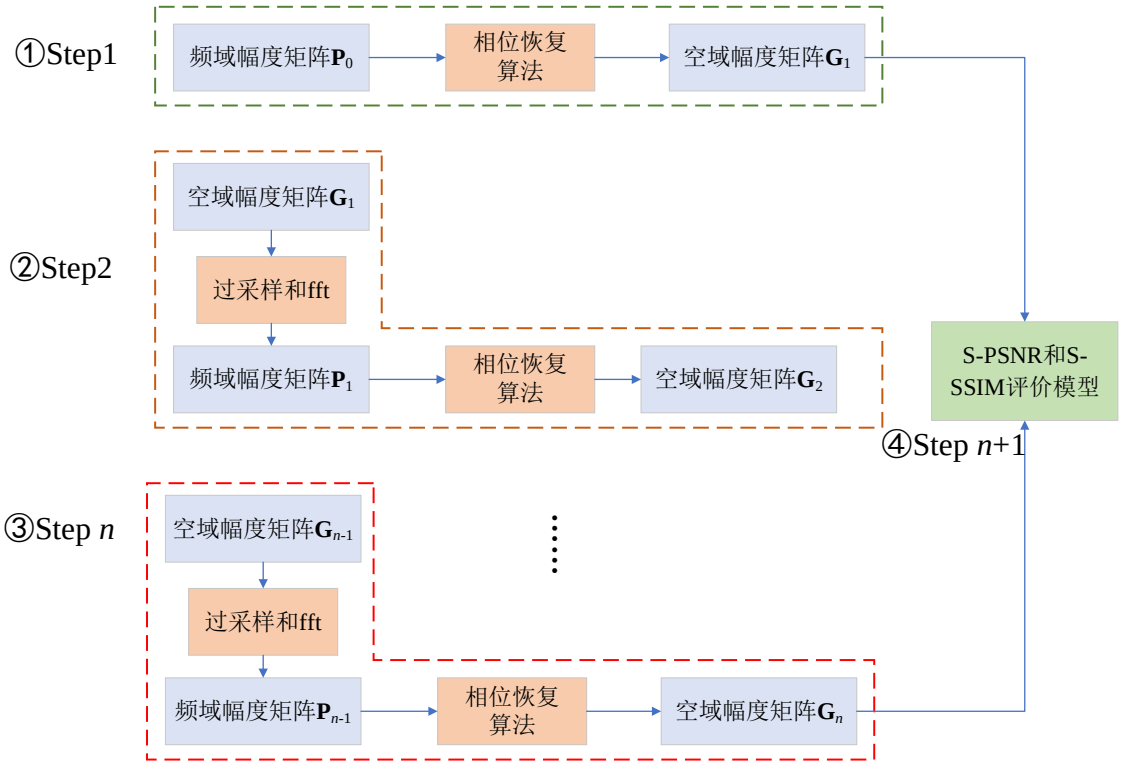


图 8 基于 S-PSNR 和 S-SSIM 的评价模型

图 7 中三种恢复图像在三轮迭代后 S-PSNR($\mathbf{G}_1$)和 S-SSIM($\mathbf{G}_1$)计算结果如下:

表 4 三种空域恢复图像的 S-PSNR 和 S-SSIM 对比

	无降噪	无区域保护的降噪	基于区域保护的降噪
S-PSNR	20	17	24
S-SSIM	0.57	0.40	0.57

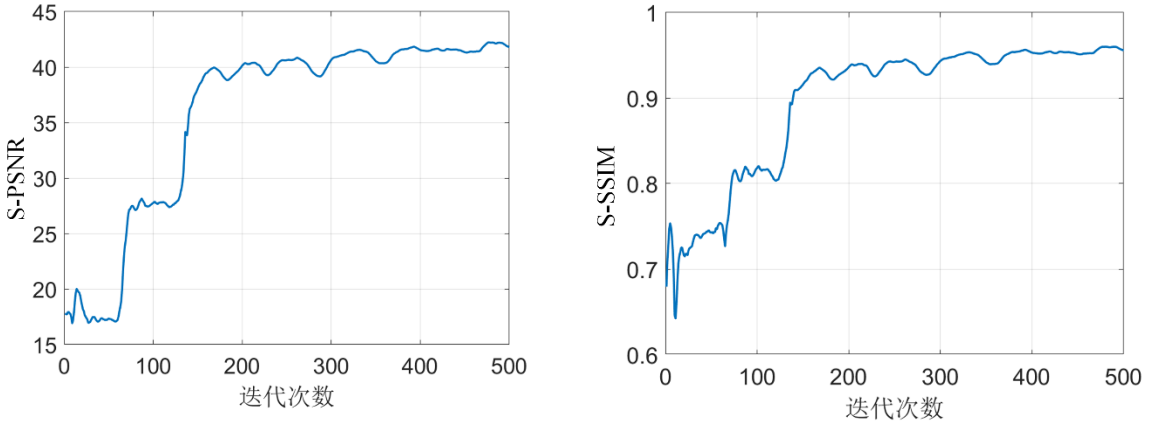
可以看到，基于区域保护降噪的图像恢复算法经过三轮迭代后的 S-SSIM($\mathbf{G}_1$)达到 57%, S-PSNR($\mathbf{G}_1$)达到 24 dB，证明经过 3 轮迭代后$\mathbf{G}_3$依然保留了$\mathbf{G}_1$中的大量信息，间接的证明了$\mathbf{G}_1$趋近于真实图像$\mathbf{G}_0$.

4.1.6 影响因素辨析

我们从迭代次数和初始相位分布两个方面对图像恢复效果进行辨析.

(一)迭代次数

下图分别为初始相位服从随机分布时, S-PSNR 与 S-SSIM 随迭代次数的变化结果. 可以看到, 两条曲线呈现出相似的变化趋势: 随着迭代次数的增加, S-PSNR 和 S-SSIM 均逐渐升高. 迭代次数较小时, 增加循环对提升恢复效果的作用较为明显; 而随着迭代次数继续增加, 对恢复效果的作改善逐渐降低, 超过 180 次时曲线基本不再上升.



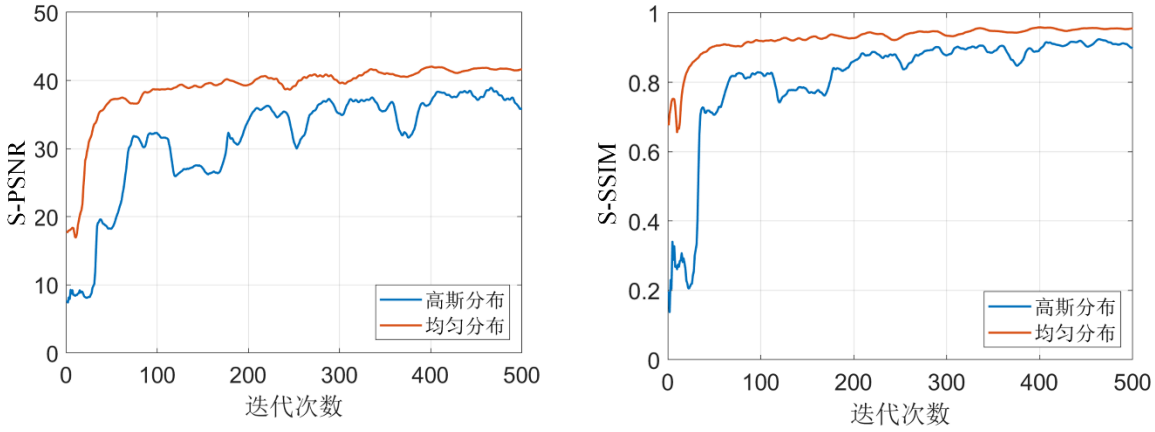
(a) S-PSNR 与迭代次数　　(b) S-SSIM 与迭代次数

图 9 迭代次数对恢复效果的影响

(二)初始相位分布

假设初始相位分别服从标准均匀分布和标准高斯分布, 图 10 列出了两种分布下 S-PSINR 和 S-SSIM 随迭代次数的变化趋势.

可以看到, 均匀分布对应的曲线始终高于高斯分布, 且更平滑, 说明初始相位服从均匀分布时图像恢复效果更好, 算法迭代稳定性更高. 此外, 均匀分布条件下仅需迭代 50 次左右即可保证结果收敛.



(a) S-PSNR 与迭代次数　　(b) S-SSIM 与迭代次数

图 10 初始相位分布对恢复效果的影响

4.2 问题二的求解

4.2.1 基于区域保护的 BM3D 去噪模型

3D 块匹配降噪(BM3D)算法是当前效果最好的传统降噪算法之一，其吸取了非局部均值滤波算法(NLM)中计算相似块的方法，又融合了小波变换域去噪[9]．BM3D 算法总共分为两大步——基础估计和最终估计．基础估计是通过硬阈值方法来进行协同滤波，而最终估计是在基础估计图像的基础上采用经验维纳收缩，根据协同变换系数的功率谱以及噪声的强度，对原始噪声图像相同位置 3D 块的协同变换系数进行收缩，然后对收缩后的系数进行反变换以及整合，得到最终降噪结果．

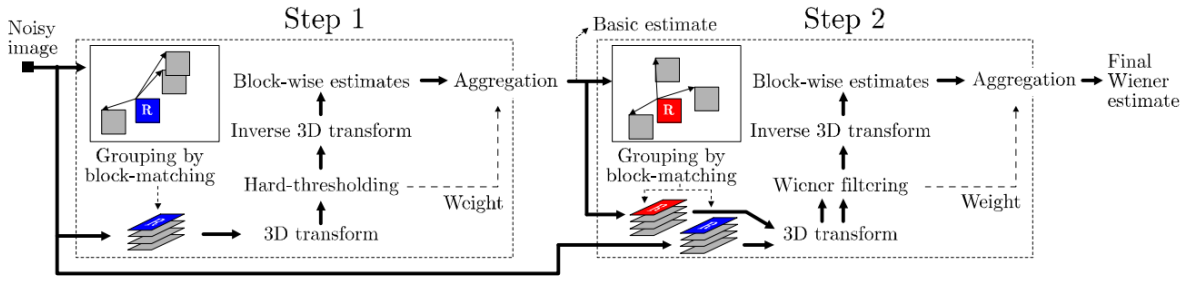


图 11 BM3D 算法流程图

上述两个步骤又包含三小步，即首先对每个参考块进行相似块分组并分别得到一个三维的组合，然后对其进行协同变换和滤波，最后对各个参考块对应组合的滤波结果进行整合，从而得到最终的降噪结果．下面重点对基础估计的三个子步骤进行说明:

Step 1 相似块分组

首先在噪声频域图像中以 3 为步长选择参照块，在参照块的周围适当大小区域内进行搜索，寻找若干个匹配误差最小的块，并将其整合成一个 3 维的矩阵．

Step 2 协同滤波

形成若干个三维的矩阵之后，对每一层进行二维变换，可采用小波变换等，更好地捕捉频谱块中周期性的信息．而后在矩阵的第三个维度进行一维变换，通过尺度缩放等更好地捕捉块与块之间的局部相似性．变换完成后对频谱块进行硬阈值处理，剔除不合格频点，然后通过反变换得到处理后的频谱块．

Step 3 聚合

将二维频谱块融合到原来的位置，对这些来自不同组合的协同滤波结果进行整合，权值取决于协同滤波后剩余的非零系数的个数．

4.2.2 附件 2 和附件 3 图像恢复结果

与问题一的解决思路相同，首先我们对 "低频区" 和 "高频区" 进行划分，对 "高频区" 使用 BM3D 算法进行降噪，而后利用差分图算法进行相位恢复、图像重建，最后基于 S-PSNR 和 S-SSIM 的评价模型给出图像恢复效果．

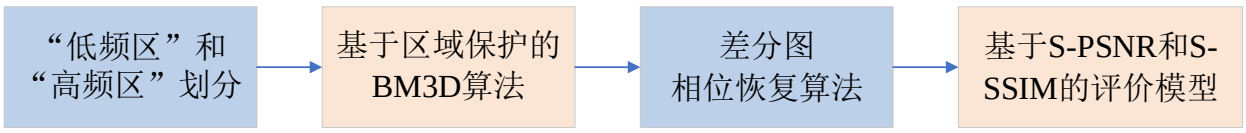
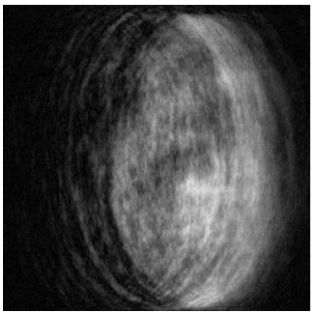
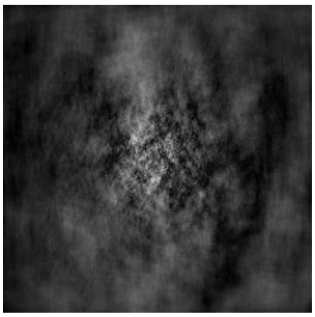


图 12 附件 2 和附件 3 图像恢复流程

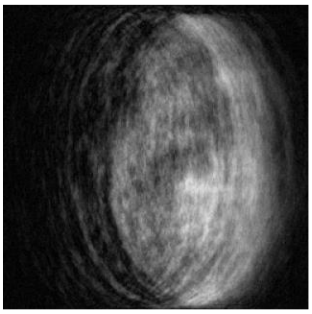
附件 2 的恢复结果和效果评价如下:



(a) 无降噪的图像恢复结果



(b) 无区域保护的降噪后图像恢复结果



(c) 基于区域保护的降噪后图像恢复结果

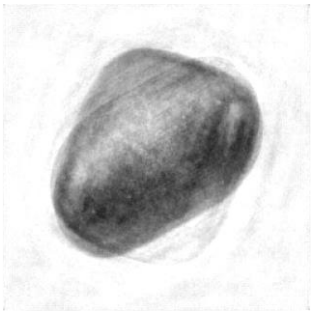
图 13 附件 2 空域图像恢复结果对比

表 5 附件 2 空域恢复图像的 S-PSNR 和 S-SSIM

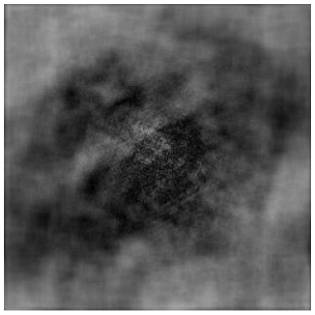
	无降噪	无区域保护的降噪	基于区域保护的降噪
S-PSNR	17.9867	15.9149	20.9131
S-SSIM	0.4202	0.3892	0.4562

附件 5 填入的是基于区域保护的降噪后图像恢复结果, 即图 13 (c).

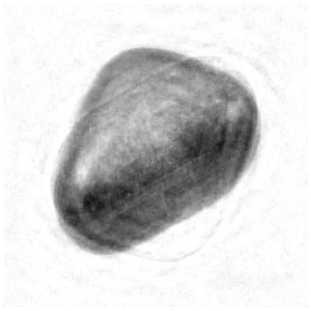
附件 3 的恢复结果和效果评价如下:



(a) 无降噪的图像恢复结果



(b) 无区域保护的降噪后图像恢复结果



(c) 基于区域保护的降噪后图像恢复结果

图 14 附件 3 空域图像恢复结果对比

表 6 附件 3 空域恢复图像的 S-PSNR 和 S-SSIM

	无降噪	无区域保护的降噪	基于区域保护的降噪
S-PSNR	25.3411	11.3868	26.6156
S-SSIM	0.7235	0.2686	0.7500

附件 6 填入的是基于区域保护的降噪后图像恢复结果, 即图 14 (c).

4.3 问题三的求解

相位恢复结果是否具有唯一性是判定恢复成功的关键指标之一. 以 $N\times N$ 的二维目标为例, 由于每个像素具有幅度和相位两个未知量, 未知量总数为 $2N^2$ 个. 而频域幅度信息提供的方程个数仅为 N^2, 小于未知数个数, 在缺少先验信息的前提下无法得出唯一解.

为了解决上述问题, 我们引入已知的先验信息, 使方程个数≥未知数个数, 并通过过采样率 β 来确定所需先验信息数目:

$$\beta = \frac{M_{\text{total}}}{M_{\text{unknown}}} \tag{20}$$

其中 M_{total} 为像素总数，M_{unknown} 为待恢复像素数. 当 $\beta \geq 2$ 时，即可得到唯一的相位恢复结果[10].

过采样过程是通过在图像周围 "补零" 实现的，这些补零区域可以视为已知先验信息，中间的待求像素区域称为 "支撑域"，如下图所示. 过采样率决定了补零区域的大小，支撑域决定了补零区域的形状.

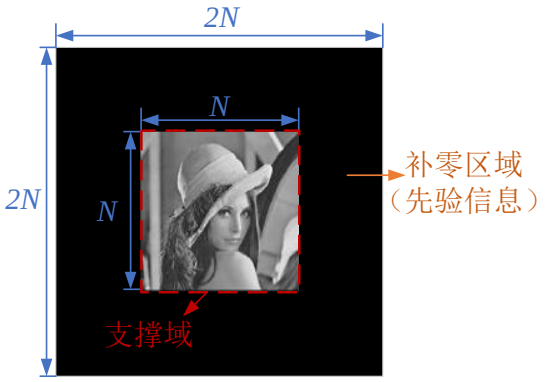


图 15 $\beta = 4$ 的支撑域和补零区域

需要注意的是，即使满足过采样条件，相位恢复仍存在颠倒、位移等模糊解问题，最终表现为多个解的叠加. 尤其是当支撑域中心对称时，模糊问题比较突出. 我们可以通过构造非中心对称支撑域来解决此问题. 可以说，提高过采样率能够提升图像清晰度，非中心对称支撑域能够减少模糊像的影响.

SSIM 能够对样本之间的亮度、对比度和结构进行综合比较，范围为 0 至 1，越大代表图像越相似，因此在本题中将 SSIM 作为图像恢复概率.

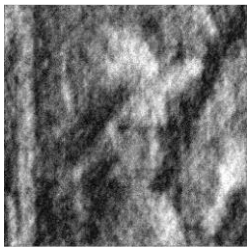
我们在 Lena 图上添加不同类型、不同噪声作为数据集，并从过采样率、支撑域两个角度分析其对图像恢复的影响.

4.3.1 过采样率分析

我们构建了三种含不同类型、不同强度噪声的图像数据集，分别为高斯噪声、脉冲噪声以及混合噪声. 噪声强度是相对归一化频谱定义的，指高斯噪声的标准差和脉冲噪声的密度. 分别计算得到了不同过采样率下的图像恢复概率，如表 7 所示，其中高斯噪声下的恢复图像由图 16 给出. 可以看到，在不同噪声下，提高过采样率均使得恢复概率显著提升，单种噪声下的恢复效果较混合噪声更好.

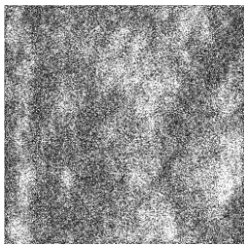
表 7 不同过采样率对应的恢复概率

		$\beta = 4$	$\beta = 16$	$\beta = 64$
脉冲噪声 (强度)	0.004	0.2443	0.5783	0.7325
	0.016	0.0312	0.1168	0.2897
高斯噪声 (强度)	0.004	0.1911	0.5991	0.7299
	0.016	0.0194	0.4053	0.5444
高斯+脉冲混合噪声(强度)	0.004	2.6686×10^{-4}	0.0421	0.2043
	0.016	4.1102×10^{-4}	0.0449	0.1306


(a) $\beta=4$，噪声强度 0.004


(b) $\beta=16$，噪声强度 0.004


(c) $\beta=64$，噪声强度 0.004


(d) $\beta=4$，噪声强度 0.016

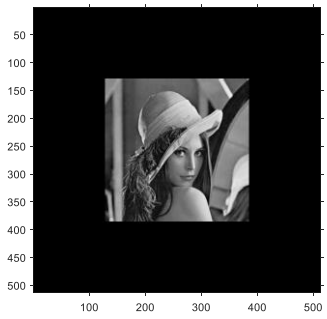

(e) $\beta=16$，噪声强度 0.016


(f) $\beta=64$，噪声强度 0.016

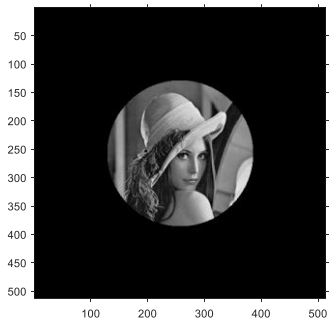
图 16 高斯噪声下，不同过采样率的恢复结果

4.3.2 中心对称解校正

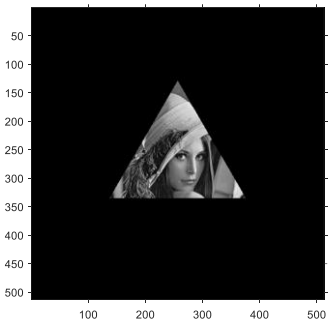
为了去除中心对称解的影响，可以采用非中心对称的支撑域，能够使其中一个解具有更大的优势. 算法会朝着优势解方向收敛，最终收敛到真值 $f(x)$ 或孪生解 $f^*(-x)$. 图 17 列出了不同支撑域的补零效果.



(a) 方形支撑域



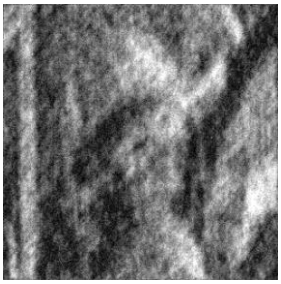
(b) 圆形支撑域

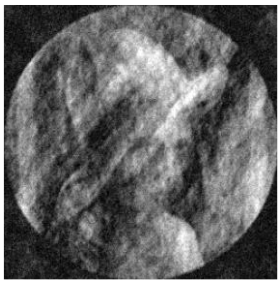


(c) 三角形支撑域

图 17 不同支撑域形状的补零示意图

下图为高斯噪声强度为 0.004、过采样率为 4 时，不同支撑域的恢复结果. 可以看到，三角形恢复效果最好，圆形次之，方形效果最差. 尤其是在混合噪声下，三角形支撑域的恢复概率远高于其余两种支撑域，验证了非中心支撑域能够有效抑制中心对称解.


(a) 方形支撑域


(b) 圆形支撑域


(c) 三角形支撑域

图 18 高斯噪声强度 0.004，不同支撑域的恢复结果

表 8 不同支撑域对应的恢复概率

		方形	圆形	三角形
脉冲噪声	0.004	0.2387	0.2630	0.3917
(强度)	0.016	0.0288	0.0505	0.1724
高斯噪声	0.004	0.1992	0.2138	0.3549
(强度)	0.016	0.0161	0.0421	0.2132
高斯+脉冲混合(强度)	0.004	3.439×10^{-4}	4.742×10^{-4}	0.0220
	0.016	4.491×10^{-5}	5.640×10^{-4}	0.0153

4.3.3 基于空域结构连续性先验的拼接校正

实际情况下, 物像平面的大小是固定的, 在过采样时并不能像 4.3.1 节这样直接在图像周围补零. 由于过采样相当于会缩小视野, 实际处理过程中对图像进行分块, 每次选定一块恢复区域, 将其他区域像素置零, 如下图所示, 最后将每一块区域的复原结果进行拼接.

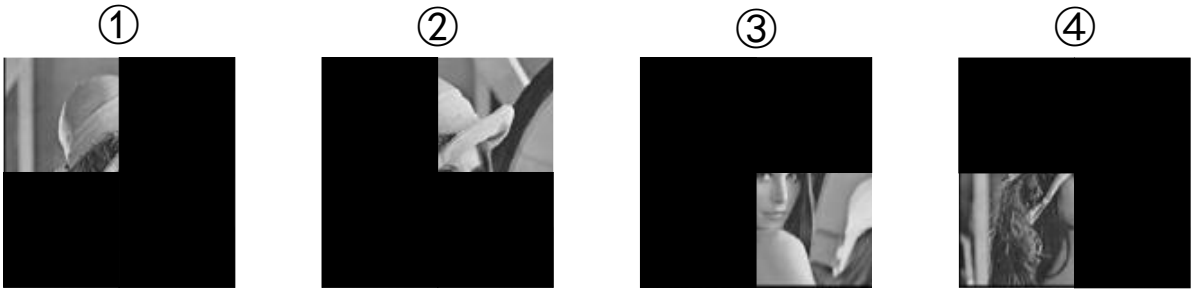


图 19 过采样率为 4 的实际处理流程

但由于中心对称解校正只能使分块复原结果收敛到真值或孪生解的其中之一, 并不能保证最终能拼接为原图, 如下图所示. 可以看到, 一些区域经相位恢复后收敛至孪生像(与原图呈 180° 旋转倒置), 最终使得恢复图像与原图产生偏差.

(a) 原图 (b) 过采样率为 4 (c) 过采样率为 16 (d) 过采样率为 64

图 20 不同过采样率的实际恢复图像

为了改善这一问题, 我们引入新的先验信息——认为图像空域结构在一定区域内具有连续性. 错误的拼接会使边缘像素梯度产生突变, 因此我们计算每一块区域在旋转前后图像的整体梯度, 以梯度变小作为旋转变换的依据. 在此基础上, 计算不同类型、强度噪声下改变过采样率对图像恢复概率的影响, 如下表所示.

表 9 不同过采样率下的恢复概率

		$\beta=4$	$\beta=16$	$\beta=64$
无噪声		0.6322	0.8204	0.7036
脉冲噪声(强度)	0.001	0.1809	0.8171	0.67
	0.004	0.1516	0.6341	0.7161
	0.016	0.0175	0.1926	0.5138
高斯噪声(强度)	0.001	0.3805	0.7563	0.6998
	0.004	0.1374	0.7114	0.6572
	0.016	0.0155	0.4287	0.6271
高斯+脉冲合噪声(强度)	0.001	1.48×10^{-4}	0.0698	0.3817
	0.004	2.11×10^{-4}	0.0699	0.3714
	0.016	9.33×10^{-5}	0.0625	0.3583

分析上表结果可以得到以下结论:

一、噪声强度越大, 图像的恢复概率越低, 且混合噪声在低采样率时基本无法恢复图像;

二、过采样率提高能显著提到各种噪声条件下图像的恢复概率. 但其并非越高越高, 这是由于过采样率太高时, 图像的分割区域太小, 使得区域旋转前后图像整体梯度基本不变, 难以保证将所有区域均恢复至正确方位.

(a) $\beta=4$ (b) $\beta=16$ (c) $\beta=64$

图 21 高斯噪声强度 0.04, 不同过采样率的恢复结果

过采样率越大, 分块区域越小, 当过采样率继续增大直至支撑域缩小为单个像素点时, 图像恢复不再存在中心对称解问题. 考虑到计算量, 我们以 64×64 的局部图为例, 当过采样率为 64×64=4096 时, 恢复概率可以达到 0.9572.

(a) 原图

(b) $\beta = 4096$ 的恢复结果

图 22 原图与恢复结果对比

5 模型评价

5.1 模型的优点

(1) 基于差分图的相位恢复算法能够很好的克服中心对称解问题, 图像恢复结果稳定, 且运算时间短、效率高;

(2) 基于区域保护的去噪算法能够在有效保护目标信息滤除大部分噪声, 对单种和多种噪声均适用, 鲁棒性好;

(3) 考虑了实际物像平面大小固定这一现实条件, 在此基础上引入空域结构连续性假设, 通过图像分块恢复、再拼接实现过采样, 处理过程更贴近实际.

5.2 模型的不足

(1) 缺乏噪声先验信息时, 若去噪算法中的参数设置不合理, 降噪效果会有所下降;

(2) 基于空域结构连续性先验的图像恢复方法对于结构连续性差的图像, 进行分块恢复后, 拼接校正效果会有所下降, 可能无法达到较好的效果.

参考文献

[1] 王金洋,曹继华.相位复原算法比较分析[J].天津职业技术师范大学报,2015,25(04):36-39.

[2] 张宇,张洪文,远国勤.相位恢复算法研究进展[J].激光与红外,2023,53(06):803-811.

[3] 张娜娜,张媛媛,丁维奇.经典图像去噪方法研究综述[J].化工自动化表,2021,48(05):409-412.

[4] 刘迪,贾金露,赵玉卿等.基于深度学习的图像去噪方法研究综述[J].计算机工程与应用,2021,57(07):1-13.

[5] Leng K. An improved non-local means algorithm for image denoising[C]//2017 IEEE 2nd International Conference on Signal and Image Processing (ICSIP). IEEE, 2017: 149-153.

[6] Yang D, Sun J. BM3D-Net: A convolutional neural network for transform-domain collaborative filtering[J]. IEEE Signal Processing Letters, 2017, 25(1): 55-59.

[7] 马晓月. 基于变分法的图像去噪算法研究[D].苏州大学,2019.

[8] Li B, Ersoy O K, Ma C, et al. Phase retrieval based on difference map and deep neural networks[J]. Journal of Modern Optics, 2021, 68(20): 1108-1120.

[9] Dabov K, Foi A, Katkovnik V, et al. Image denoising by sparse 3-D transform-domain collaborative filtering[J]. IEEE Transactions on image processing, 2007, 16(8): 2080-2095.

[10] Miao J, Sayre D, Chapman H N. Phase retrieval from the magnitude of the Fourier transforms of nonperiodic objects [J]. JOSA A, Optical Society of America, 1998,15(6):1662–1669.

附　录

附录 A：支撑材料列表

支撑材料列表

序号	文件名	材料说明
1	Code	代码集合

附录 B：主要程序/关键代码

代码环境	操作系统: Windows 10 编程语言: MATLAB 编辑器: MATLAB 2019 代码详见：附件

代码清单 1 问题一去噪算法代码

```
function  u=hpROF(u0,IterMax,lambda)
u0=double(u0);   %将原图像转化为双精度浮点型
[M  N]=size(u0);
u=u0;
[M,N]=size(u);   %【行数(高度),列数(宽度)】
h=1;%   空间离散
for  Iter=1:IterMax,       %迭代

      for  i=2:M-1,
            for  j=2:N-1,
                  %系数计算
                  ux=(u(i+1,j)-u(i,j))/h;
                  uy=(u(i,j+1)-u(i,j-1))/2*h;
                  Gradu=sqrt(ux*ux+uy*uy);
                  co1=1./Gradu;

                  ux=(u(i,j)-u(i-1,j))/h;
                  uy=(u(i-1,j+1)-u(i-1,j-1))/2*h;
                  Gradu=sqrt(ux*ux+uy*uy);
                  co2=1./Gradu;

                  ux=(u(i+1,j)-u(i-1,j))/2*h;
                  uy=(u(i,j+1)-u(i,j))/h;
                  Gradu=sqrt(ux*ux+uy*uy);
                  co3=1./Gradu;

                  ux=(u(i+1,j-1)-u(i-1,j-1))/2*h;
                  uy=(u(i,j)-u(i,j-1))/h;
                  Gradu=sqrt(ux*ux+uy*uy);
                  co4=1./Gradu;
                  u(i,j)=(u0(i,j)+(1/lambda*h*h)*(co1*u(i+1,j)+co2*u(i-1,j)+co3*u(i,j+1)+co4*u(i,j-1)))*(1/
(1+(1/(lambda*h*h)*(co1+co2+co3+co4))));
%                     co=1./(1.+(1/lambda*h*h)*(co1+co2+co3+co4));
%                     div=co1*u(i+1,j)+co2*u(i-1,j)+co3*u(i,j+1)+co4*u(i,j-1);
%                     u(i,j)=(u0(i,j)+(1/lambda*h*h)*div)*co;

            end
```

```
    end
    %边缘条件
    for i=2:M-1,
        u(i,1)=u(i,2);      %第一列的计算等于第二列
        u(i,N)=u(i,N-1);   %最后一列的计算等于倒数第二列
    end

    for j=2:N-1,
        u(1,j)=u(2,j);
        u(M,j)=u(M-1,j);   %同理, 第一行和最后一行的计算等于和它相近的行
    end

    u(1,1)=u(2,2);
    u(1,N)=u(2,N-1);
    u(M,1)=u(M-1,2);
    u(M,N)=u(M-1,N-1);      %四个角的值也近似与它相近的点
    %计算每次迭代的离散能量
    en=0.0;     %设能量初始值为0
    for i=2:M-1,
        for j=2:N-1,
            ux=(u(i+1,j)-u(i,j))/h;
            uy=(u(i,j+1)-u(i,j))/h;
            fid=(u0(i,j)-u(i,j))*(u0(i,j)-u(i,j));
            en=en+sqrt(ux*ux+uy*uy)+lambda*fid;
        end
    end
    %计算每次迭代的能量
    Energy(Iter)=en;
end
```

代码清单 2 问题一相位恢复算法代码

```
clc;close all;

N = 512; % size of the diffraction pattern image
m = 256; % size of the sample image
m2 = m/2;

sample = rgb2gray(imread('lena.jpg')); % read the lena.jpg as sample image
sample = imresize(sample,[m m]);
sample = MatMap(sample,0,1); % the sample matrix is normalized

figure;imshow(sample,'InitialMagnification',200);axis on;title('sample image');
%%
S = zeros(N,N);
% put the sample image into a zero background
S(N/2-(m2-1):N/2+1+(m2-1) ,N/2-(m2-1):N/2+1+(m2-1)) = sample;

% sup = circle_mask(N,m,N/2,N/2); % generate a circle support
% sup = triMask(N,m/2+8,N/2+10,N/2); % 要是换成一个三角形掩模效果会更好
sup = square_mask(N,m); % 要是换成一个三角形掩模效果会更好

S = S.*sup;
figure;imshow(S);axis on;title('样品被掩模遮挡部分后(支持域作用的效果)');

S = abs(fftshift(fft2(S))); % generate the modulus of the diffraction pattern
```

```
S= csv2I('附件3.csv');
S_denoise=hpROF(S,100,0.01);

% S = image;
S=abs(fftshift(S));
S_denoise=abs(fftshift(S_denoise));
S_protect = Protect_and_remake(S,S_denoise,500);
% S_protect = S_denoise;
% S=S_protect;
figure;
imagesc(log(1+S));
axis square;
title('The modulus of diffraction pattern (log)');
%%
itnum = 500; % iteration number
g = rand(N,N); % 生成初始随机g
for i = 1:itnum
    g = g + projectM(2*projectSup(g,sup)-g,S) - projectSup(g,sup);    % g进行更新, 只利用了sup掩模和频谱
    %==============display the reconstruct sample image==================
    imshow(abs(g(N/2-(m2-1):N/2+1+(m2-1),N/2-(m2-1):N/2+1+(m2-1))),'InitialMagnification',200);
    final=abs(g(N/2-(m2-1):N/2+1+(m2-1),N/2-(m2-1):N/2+1+(m2-1)));
%       title(strcat('迭代步数',num2str(i)));
%       pause(0.01); % 每一步迭代结果的显示时间
    % 显示中间的迭代结果时只显示 N x N 大小重建图像中间的 m x m 大小的区域, 旁边大片的黑色部分不显示
end
I1 = final;
%%  有了I1 , 继续使用算法计算I2

%%
sample = I1;
S = zeros(N,N);
% put the sample image into a zero background
S(N/2-(m2-1):N/2+1+(m2-1) ,N/2-(m2-1):N/2+1+(m2-1)) = sample;

% sup = circle_mask(N,m,N/2,N/2); % generate a circle support
% sup = triMask(N,m/2+8,N/2+10,N/2); % 要是换成一个三角形掩模效果会更好
sup = square_mask(N,m); % 要是换成一个三角形掩模效果会更好

S = S.*sup;
figure;imshow(S);axis on;title('样品被掩模遮挡部分后(支持域作用的效果)');

S = abs(fftshift(fft2(S))); % generate the modulus of the diffraction pattern

% S= csv2I('附件1.csv');
% S_denoise=hpROF(S,100,0.01);

% S = image;
```

```
S=abs(fftshift(S));
S_denoise=abs(fftshift(S_denoise));
% S_protect = Protect_and_remake(S,S_denoise,500);
% S_protect = S_denoise;
% S=S_protect;
figure;
imagesc(log(1+S));
axis square;
title('The modulus of diffraction pattern (log)');
%%
itnum = 500; % iteration number
g = rand(N,N); % 生成初始随机g
for i = 1:itnum
    g = g + projectM(2*projectSup(g,sup)-g,S) - projectSup(g,sup);    % g进行更新, 只利用了sup掩模和频谱
    %===============display the reconstruct sample image===================
    imshow(abs(g(N/2-(m2-1):N/2+1+(m2-1),N/2-(m2-1):N/2+1+(m2-1))),'InitialMagnification',200);
    final=abs(g(N/2-(m2-1):N/2+1+(m2-1),N/2-(m2-1):N/2+1+(m2-1)));
%     title(strcat('迭代步数',num2str(i)));
%     pause(0.01); % 每一步迭代结果的显示时间
    % 显示中间的迭代结果时只显示 N x N 大小重建图像中间的 m x m 大小的区域, 旁边大片的黑色部分不显示
end
I2 = final;
psnr(I1,I2)
psnr(I1,rot90(I2,2))

ssim(I1,I2)
ssim(I1,rot90(I2,2))

%%  有了I2 , 继续使用算法计算I3
sample = I2;
S = zeros(N,N);
% put the sample image into a zero background
S(N/2-(m2-1):N/2+1+(m2-1) ,N/2-(m2-1):N/2+1+(m2-1)) = sample;

% sup = circle_mask(N,m,N/2,N/2); % generate a circle support
% sup = triMask(N,m/2+8,N/2+10,N/2); % 要是换成一个三角形掩模效果会更好
sup = square_mask(N,m); % 要是换成一个三角形掩模效果会更好

S = S.*sup;
figure;imshow(S);axis on;title('样品被掩模遮挡部分后(支持域作用的效果)');

S = abs(fftshift(fft2(S))); % generate the modulus of the diffraction pattern

% S= csv2I('附件1.csv');
% S_denoise=hpROF(S,100,0.01);

% S = image;
S=abs(fftshift(S));
```

```
S_denoise=abs(fftshift(S_denoise));
% S_protect = Protect_and_remake(S,S_denoise,500);
% S_protect = S_denoise;
% S=S_protect;
figure;
imagesc(log(1+S));
axis square;
title('The modulus of diffraction pattern (log)');
%%
itnum = 500; % iteration number
g = rand(N,N); % 生成初始随机g
    g = g + projectM(2*projectSup(g,sup)-g,S) - projectSup(g,sup);    % g进行更新, 只利用了sup掩模和频谱
    %==============display the reconstruct sample image==================
    imshow(abs(g(N/2-(m2-1):N/2+1+(m2-1),N/2-(m2-1):N/2+1+(m2-1))),'InitialMagnification',200);
    final=abs(g(N/2-(m2-1):N/2+1+(m2-1),N/2-(m2-1):N/2+1+(m2-1)));
%       title(strcat('迭代步数',num2str(i)));
%       pause(0.01); % 每一步迭代结果的显示时间
    % 显示中间的迭代结果时只显示 N x N 大小重建图像中间的 m x m 大小的区域, 旁边大片的黑色部分不显示
end
I3 = final;
psnr(I3,I2)
psnr(I3,rot90(I2,2))

ssim(I3,I2)
ssim(I3,rot90(I2,2))

I3 = final;
psnr(I3,I1)
psnr(I3,rot90(I1,2))

ssim(I3,I1)
ssim(I3,rot90(I1,2))
```

代码清单 3 问题二去噪算法代码

```
function [varargout] = BM3D_CFA(z, sigma)
image_name = [
    'kodim07.png'
%       'kodim08.png'
%       'kodim19.png'
%       'kodim23.png'
    ];

%%%%  Quality/complexity trade-off profile selection
%%%%
%%%%  'np' --> Normal Profile (balanced quality)
```

```
if ~exist('profile','var')
    profile             = 'np'; %% default profile
end

%%%%  Specify the std. dev. of the corrupting noise
%%%%
if ~exist('sigma','var')
    sigma                   = 25; %% default standard deviation of the AWGN
end

%%%% Following are the parameters for the Normal Profile.
%%%%

%%%% Select transforms ('dct', 'dst', 'hadamard', or anything that is listed by 'help wfilters'):
transform_2D_HT_name      = 'dct'; %% transform used for the HT filt. of size N1 x N1
transform_2D_Wiener_name = 'dct';
transform_3rd_dim_name    = 'haar';      %% transform used in the 3-rd dim, the same for HT and Wiener filt.

%%%% Hard-thresholding (HT) parameters:
N1                      = 5;    %% N1 x N1 is the block size used for the hard-thresholding (HT) filtering
Nstep                   = 3;    %% sliding step to process every next reference block
N2                      = 16;   %% maximum number of similar blocks (maximum size of the 3rd dimension of a 3D array)
Ns                      = 39;   %% length of the side of the search neighborhood for full-search block-matching (BM), must be odd
lambda_thr2D            = 0;
tau_match               = 3000;%% threshold for the block-distance (d-distance)
lambda_thr3D            = 2.7; %% threshold parameter for the hard-thresholding in 3D transform domain
beta                    = 2.0; %% parameter of the 2D Kaiser window used in the reconstruction

%%%% Step 2: Wiener filtering parameters:
N1_wiener               = 6;
Nstep_wiener            = 3;
N2_wiener               = 32;
Ns_wiener               = 39;
tau_match_wiener        = 400;
beta_wiener             = 2.0;



%%%% Block-matching parameters:
stepFS                  = 1;   %% step that forces to switch to full-search BM, "1" implies always full-search
smallLN                 = 'not used in np'; %% if stepFS > 1, then this specifies the size of the small local search neighb.
stepFSW                 = 1;
smallLNW                = 'not used in np';
thrToIncStep            = 8;   % if the number of non-zero coefficients after HT is less than thrToIncStep,
```

```
                                % than the sliding step to the next reference block is incresed to (n
m1-1)


decLevel = 0;           %% dec. levels of the dyadic wavelet 2D transform for blocks (0 means f
ull decomposition, higher values decrease the dec. number)
thr mask = ones(N1); %% N1xN1 mask of threshold scaling coeff. --- by default there is no scali
ng, however the use of different thresholds for different wavelet decompoistion subbands can be d
one with this matrix


[Tfor, Tinv]    = getTransfMatrix(N1, transform 2D HT name, decLevel);       %% get (normalize
d) forward and inverse transform matrices
[TforW, TinvW] = getTransfMatrix(N1 wiener, transform 2D Wiener name, 0); %% get (normalize
d) forward and inverse transform matrices


if (strcmp(transform 3rd dim name, 'haar') == 1) | (strcmp(transform 3rd dim name(end-2:end), '1.1
') == 1),
    %%% If Haar is used in the 3-rd dimension, then a fast internal transform is used, thus no
need to generate transform
    %%% matrices.
    hadper_trans_single_den             = {};
    inverse_hadper_trans_single_den = {};
else
    %%% Create transform matrices. The transforms are later applied by
    %%% matrix-vector multiplication for the 1D case.
    for hpow = 0:ceil(log2(max(N2,N2_wiener))),
        h = 2^hpow;
        [Tfor3rd, Tinv3rd]    = getTransfMatrix(h, transform_3rd_dim_name, 0);
        hadper_trans_single_den{h}              = single(Tfor3rd);
        inverse_hadper_trans_single_den{h} = single(Tinv3rd');
    end
end


%%%% 2D Kaiser windows used in the aggregation of block-wise estimates
%%%%
if beta wiener==2 & beta==2 & N1 wiener==8 & N1==8 % hardcode the window function so th
at the signal processing toolbox is not needed by default
    Wwin2D = [ 0.1924    0.2989    0.3846    0.4325    0.4325    0.3846    0.2989    0.1
924;
        0.2989    0.4642    0.5974    0.6717    0.6717    0.5974    0.4642    0.2989;
        0.3846    0.5974    0.7688    0.8644    0.8644    0.7688    0.5974    0.3846;
        0.4325    0.6717    0.8644    0.9718    0.9718    0.8644    0.6717    0.4325;
        0.4325    0.6717    0.8644    0.9718    0.9718    0.8644    0.6717    0.4325;
        0.3846    0.5974    0.7688    0.8644    0.8644    0.7688    0.5974    0.3846;
        0.2989    0.4642    0.5974    0.6717    0.6717    0.5974    0.4642    0.2989;
        0.1924    0.2989    0.3846    0.4325    0.4325    0.3846    0.2989    0.1924];
    Wwin2D_wiener = Wwin2D;
else
    Wwin2D              = kaiser(N1, beta) * kaiser(N1, beta)'; % Kaiser window used in the ag
gregation of the HT part
    Wwin2D wiener    = kaiser(N1 wiener, beta wiener) * kaiser(N1 wiener, beta wiener)'; % K
aiser window used in the aggregation of the Wiener filt. part
end


%%%% If needed, read images, generate noise, or scale the images to the
```

```
%%%% [0,1] interval
%%%%
if ~exist('z','var')
    yRGB          = im2double(imread(image_name));  %% read a noise-free image and put in in
tensity range [0,1]
    y = zeros(size(yRGB,1), size(yRGB,2));
    y(1:2:end,1:2:end) = yRGB(1:2:end,1:2:end,2);
    y(2:2:end,2:2:end) = yRGB(2:2:end,2:2:end,2);
    y(1:2:end,2:2:end) = yRGB(1:2:end,2:2:end,1);
    y(2:2:end,1:2:end) = yRGB(2:2:end,1:2:end,3);

    randn('seed', 0);                                  %% generate seed
    z           = y + (sigma/255)*randn(size(y)); %% create a noisy image

else  % external images

    image_name = 'External image';

    % convert z to double precision if needed
    z = double(z);
    y= [];
end

if (size(z,3) ~= 1)
    error('BM3D accepts only grayscale 2D images.');
end

%%% Check whether to dump information to the screen or remain silent
if isempty(y)
    dump_output_information = false;
else
    dump_output_information = true;
end

%%% Print image information to the screen
%%%%
if dump_output_information
    fprintf('Image: %s (%dx%d), sigma: %.1f\n', image_name, size(z,1), size(z,2), sigma);
end

%%%% Step 1. Produce the basic estimate by HT filtering
%%%%
tic;
y_ht = bm3d_CFA_thr(z, hadper_trans_single_den, Nstep, N1, N2, lambda_thr2D,...
    lambda thr3D, tau match*N1*N1/(255*255), (Ns-1)/2, (sigma/255), thrToIncStep, single(Tfor), si
ngle(Tinv)', inverse_hadper_trans_single_den, single(thr_mask), Wwin2D, smallLN, stepFS );
estimate_elapsed_time = toc;

if dump_output_information
    PSNR_INITIAL_ESTIMATE = 10*log10(1/mean((y(:)-double(y_ht(:))).^2));
    fprintf('BASIC ESTIMATE, PSNR: %.2f dB\n', PSNR_INITIAL_ESTIMATE);
```

```
end

%%%% Step 2. Produce the final estimate by Wiener filtering (using the
%%%%  hard-thresholding initial estimate)
%%%%
tic;
y_wiener = bm3d_CFA_wiener(z, y_ht, hadper_trans_single_den, Nstep_wiener, N1_wiener, N2_wiener, ...
    'unused_arg', tau_match_wiener*N1_wiener*N1_wiener/(255*255), (Ns_wiener-1)/2, (sigma/255), 'unused_arg', single(TforW), single(TinvW)', inverse_hadper_trans_single_den, Wwin2D_wiener, smallLNW, stepFSW, single(ones(N1_wiener)) );
wiener_elapsed_time = toc;

%%%% Calculate the final estimate's PSNR, print it, and show the
%%%% denoised image next to the noisy one
%%%%
y_wiener = double(y_wiener);


if dump_output_information
    PSNR = 10*log10(1/mean((y(:)-y_wiener(:)).^2)); % y is valid
    fprintf('FINAL ESTIMATE (total time: %.1f sec), PSNR: %.2f dB\n', ...
        wiener_elapsed_time + estimate_elapsed_time, PSNR);

    figure, imshow(z); title(sprintf('Noisy %s, PSNR: %.3f dB (sigma: %d)', ...
        image_name(1:end-4), 10*log10(1/mean((y(:)-z(:)).^2)), sigma));

    figure, imshow(y_wiener); title(sprintf('Denoised %s, PSNR: %.3f dB', ...
        image_name(1:end-4), PSNR));

end

if nargout==0
    varargout={};
else
    varargout{1}=y_wiener;
    varargout{2}=y_ht;
end

return;

function [Tforward, Tinverse] = getTransfMatrix (N, transform_type, dec_levels)
%
% Create forward and inverse transform matrices, which allow for perfect
% reconstruction. The forward transform matrix is normalized so that the
% l2-norm of each basis element is 1.
%
% [Tforward, Tinverse] = getTransfMatrix (N, transform_type, dec_levels)
%
%   INPUTS:
%
```

```
%    N                  --> Size of the transform (for wavelets, must be 2^K)
%
%    transform_type   --> 'dct', 'dst', 'hadamard', or anything that is
%                         listed by 'help wfilters' (bi-orthogonal wavelets)
%                         'DCrand' -- an orthonormal transform with a DC and all
%                         the other basis elements of random nature
%
%    dec_levels       --> If a wavelet transform is generated, this is the
%                         desired decomposition level. Must be in the
%                         range [0, log2(N)-1], where "0" implies
%                         full decomposition.
%
%   OUTPUTS:
%
%    Tforward          --> (N x N) Forward transform matrix
%
%    Tinverse          --> (N x N) Inverse transform matrix
%

if exist('dec_levels') ~= 1,
    dec_levels = 0;
end

if N == 1,
    Tforward = 1;
elseif strcmp(transform_type, 'hadamard') == 1,
    Tforward = hadamard(N);
elseif (N == 8) & strcmp(transform type, 'bior1.5')==1 % hardcoded transform so that the wavelet toolbox is not needed to generate it
    Tforward = [ 0.353553390593274 0.353553390593274 0.353553390593274 0.35355339 0593274 0.353553390593274 0.353553390593274 0.353553390593274 0.353553390593274;
        0.219417649252501 0.449283757993216 0.449283757993216 0.219417649252501 -0.219417649252501 -0.449283757993216 -0.449283757993216 -0.219417649252501;
        0.569359398342846 0.402347308162278 -0.402347308162278 -0.569359398342846 -0.083506045090284 0.083506045090284 -0.083506045090284 0.083506045090284;
       -0.083506045090284 0.083506045090284 -0.083506045090284 0.083506045090284 0.569359398342846 0.402347308162278 -0.402347308162278 -0.569359398342846;
        0.707106781186547 -0.707106781186547 0 0
                0 0 0 0;
                    0 0 0.707106781186547 -0.707106781186547
                0 0 0 0;
                    0 0 0 0
 0.707106781186547 -0.707106781186547 0 0;
                    0 0 0 0
                0 0 0.707106781186547 -0.707106781186547];
elseif (N == 8) & strcmp(transform type, 'dct')==1 % hardcoded transform so that the signal proc essing toolbox is not needed to generate it
    Tforward = [ 0.353553390593274 0.353553390593274 0.353553390593274 0.353553390 593274 0.353553390593274 0.353553390593274 0.353553390593274 0.353553390593274;
        0.490392640201615 0.415734806151273 0.277785116509801 0.097545161008064 -0.097545161008064 -0.277785116509801 -0.415734806151273 -0.490392640201615;
        0.461939766255643 0.191341716182545 -0.191341716182545 -0.461939766255643 -0.461939766255643 -0.191341716182545 0.191341716182545 0.461939766255643;
        0.415734806151273 -0.097545161008064 -0.490392640201615 -0.277785116509801 0.277785116509801 0.490392640201615 0.097545161008064 -0.415734806151273;
```

```
        0.353553390593274 -0.353553390593274 -0.353553390593274 0.353553390593274 0.353553390593274 -0.353553390593274 -0.353553390593274 0.353553390593274;
        0.277785116509801 -0.490392640201615 0.097545161008064 0.415734806151273 -0.415734806151273 -0.097545161008064 0.490392640201615 -0.277785116509801;
        0.191341716182545 -0.461939766255643 0.461939766255643 -0.191341716182545 -0.191341716182545 0.461939766255643 -0.461939766255643 0.191341716182545;
        0.097545161008064 -0.277785116509801 0.415734806151273 -0.490392640201615 0.490392640201615 -0.415734806151273 0.277785116509801 -0.097545161008064];
elseif (N == 8) & strcmp(transform type, 'dst')==1 % hardcoded transform so that the PDE toolbox is not needed to generate it
    Tforward = [ 0.161229841765317 0.303012985114696 0.408248290463863 0.464242826880013 0.464242826880013 0.408248290463863 0.303012985114696 0.161229841765317;
        0.303012985114696 0.464242826880013 0.408248290463863 0.161229841765317 -0.161229841765317 -0.408248290463863 -0.464242826880013 -0.303012985114696;
        0.408248290463863 0.408248290463863 0 -0.408248290463863 -0.408248290463863 0 0.408248290463863 0.408248290463863;
        0.464242826880013 0.161229841765317 -0.408248290463863 -0.303012985114696 0.303012985114696 0.408248290463863 -0.161229841765317 -0.464242826880013;
        0.464242826880013 -0.161229841765317 -0.408248290463863 0.303012985114696 0.303012985114696 -0.408248290463863 -0.161229841765317 0.464242826880013;
        0.408248290463863 -0.408248290463863 0 0.408248290463863 -0.408248290463863 0 0.408248290463863 -0.408248290463863;
        0.303012985114696 -0.464242826880013 0.408248290463863 -0.161229841765317 -0.161229841765317 0.408248290463863 -0.464242826880013 0.303012985114696;
        0.161229841765317 -0.303012985114696 0.408248290463863 -0.464242826880013 0.464242826880013 -0.408248290463863 0.303012985114696 -0.161229841765317];
elseif strcmp(transform_type, 'dct') == 1,
    Tforward     = dct(eye(N));
elseif strcmp(transform_type, 'dst') == 1,
    Tforward     = dst(eye(N));
elseif strcmp(transform_type, 'DCrand') == 1,
    x = randn(N); x(1:end,1) = 1; [Q,R] = qr(x);
    if (Q(1) < 0),
        Q = -Q;
    end;
    Tforward = Q';
else %% a wavelet decomposition supported by 'wavedec'
    %%% Set periodic boundary conditions, to preserve bi-orthogonality
    dwtmode('per','nodisp');

    Tforward = zeros(N,N);
    for i = 1:N
        Tforward(:,i)=wavedec(circshift([1 zeros(1,N-1)],[dec levels i-1]), log2(N), transform type); %% construct transform matrix
    end
end

%%% Normalize the basis elements
Tforward = (Tforward' * diag(sqrt(1./sum(Tforward.^2,2))))';

%%% Compute the inverse transform matrix
Tinverse = inv(Tforward);

return;
```

代码清单 4 问题三核心代码

```
clear all;
clc;close all;

N = 512; % size of the diffraction pattern image
scale = 4;% 过采样率开根号, 边长倍数2,4,8,16
m = N/scale; % size of the sample image
m2 = m/2;

sample = rgb2gray(imread('Lena.png')); % readthe lena.jpg as sample image
% sample=sample(1+256:N+256,1+256:N+256);
% sample = imresize(sample,[m m]);
sample = MatMap(sample,0,1); % the sample matrix is normalized

figure;imshow(sample,'InitialMagnification',200);axis on;title('sample image');
%%
I_final = zeros(N,N);
I_denoise_final=zeros(N,N);
% figure(2)
noise_type = 1;%'salt';
noise_type = 2;%'gaussian';
noise_type = 3;%'gaussian+salt';
noise_type = 0;'no noise';

noise_type = 0;
noise_energy = 0.02;%噪声强度, 0.05,0.1
if noise_type == 1
    noise = imnoise(zeros(512,512),'salt & pepper',noise_energy.^2); % 添加5% 噪声, 10%噪声
else
    if noise_type ==2
        noise = imnoise(zeros(512,512), 'gaussian', 0,noise_energy.^2) ;    %均值始终为0, 方差为
0.05, 0.1
    else
        if noise_type == 3
            noise = imnoise(zeros(512,512),'salt & pepper',noise energy.^2)+imnoise(zeros(512,51
2),'gaussian',noise_energy.^2); % 加两种混合噪声
        else %noise_type == 0
            noise = zeros(512,512);
        end
    end
end

psnr_all=[4,500];ssim_all=[4,500];
for ID = 1:scale*scale
    % S = zeros(N,N);
    % put the sample image into a zero background
    % S(N/2-(m2-1):N/2+1+(m2-1) ,N/2-(m2-1):N/2+1+(m2-1)) = sample;
    S=sample;
```

```
% sup = circle_mask(N,m2,N/2+50,N/2+50); % generate a circle support
% sup = triMask(N,m/2+8,N/2+10,N/2); % 要是换成一个三角形掩模效果会更好

sup = square_mask(N,m,ID); % 矩形掩模

S = S.*sup;
%       figure;imshow(S);axis on;title('样品被掩模遮挡部分后(支持域作用的效果)');

S = abs(fftshift(fft2(S))); % generate the modulus of the diffraction pattern
S_max = max(max( S));
S_min=min(min( S));
S = (S-min(min( S)))./(max(max( S))-min(min( S)));
S = S+noise;% 加入噪声, 如果没有噪声, 加入一个空矩阵
S_denoise=hpROF(S,100,noise_energy.^2);
S = S*(S_max-S_min)+S_min;
S_denoise = S_denoise*(S_max-S_min)+S_min;
S_protect = Protect_and_remake(S,S_denoise,256);


%       S =S_protect ;
% S= csv2I('附件1.csv');
% S = image;
% S=abs(fftshift(S));
%       figure;
%       imagesc(log(1+S));
%       axis square;
%       title('The modulus of diffraction pattern (log)');
%%
for phase = 2
    itnum = 500; % iteration number
    switch phase

        case 1
            g = randn(N,N); % 生成初始随机g
        otherwise
            g = rand(N,N); % 生成初始随机g
    end

    g2 = g; % 生成初始随机g
    for i = 1:25%itnum
        g = g + projectM(2*projectSup(g,sup)-g,S) - projectSup(g,sup);    % g进行更新, 只利用了sup掩模和频谱
        g2 = g2 + projectM(2*projectSup(g2,sup)-g2,S protect) - projectSup(g2,sup);    % g进行更新, 只利用了sup掩模和频谱
        %==============display the reconstruct sample image==================
        %       imshow(abs(g(N/2-(m2-1):N/2+1+(m2-1),N/2-(m2-1):N/2+1+(m2-1))),'InitialMagnification',200);
        %           imshow(abs(g),'InitialMagnification',200);
        %           hold on;
        %       title(strcat('迭代步数',num2str(i)));
```

```
                %       pause(0.01); % 每一步迭代结果的显示时间
                % 显示中间的迭代结果时只显示 N x N 大小重建图像中间的 m x m 大小的区域,旁边大片的黑色部分不显示
                I_sup = sup.*abs(g);
                imshow(I_sup);
                I_sup = I_sup(1:N/scale,1:N/scale);
                psnr_all(phase,i)=max([psnr(I_sup,sample(1:N/scale,1:N/scale)),psnr(rot90(I_sup,2),sample(1:N/scale,1:N/scale))]);
                ssim_all(phase,i)=max([ssim(I_sup,sample(1:N/scale,1:N/scale)),ssim(rot90(I_sup,2),sample(1:N/scale,1:N/scale))]);
            end
        end
        %       g2 = ifft2(S_protect.*(fft2(g)./abs(fft2(g))));
        I_final = I_final+sup.*abs(g);
        I_denoise_final = I_denoise_final+sup.*abs(g2);
end
% figure(10)
% for phase=1:2
%       plot([1:itnum],psnr_all(phase,:),'linewidth',1.5);
%       hold on;
% end
% xlabel('迭代次数')
% ylabel('PSNR')
% set(gca,'fontsize',16);
% legend('高斯分布','均匀分布')
% figure(11)
% for phase=1:2
%       plot([1:itnum],ssim_all(phase,:),'linewidth',1.5);
%       hold on;
% end
% xlabel('迭代次数')
% ylabel('SSIM')
% set(gca,'fontsize',16);
% legend('高斯分布','均匀分布')
figure(6)
disp('生成不利用先验信息的加噪声的图像结果, ssim有两个选一个大的')
subplot(1,2,1)
imshow(I_final)
subplot(1,2,2)
imshow(I_denoise_final)

sample_rate=scale*scale% 过采样率
noise_type
noise_energy

ssim_final = ssim(I_final,sample)
ssim_final = ssim(I_final,rot90(sample,2))

ssim_final = ssim(I_denoise_final,sample)
ssim_final = ssim(I_denoise_final,rot90(sample,2))
```

```
I_final_with_rot=I_final;
I_denoise_final_with_rot=I_denoise_final;
for  ID=1:scale*scale
      [judge,I_final_with_rot]=judge_rot(I_final_with_rot,N,scale,ID);
      [judge,I_denoise_final_with_rot]=judge_rot(I_denoise_final_with_rot,N,scale,ID);
      %         judge
end
for  ID=1:scale*scale
      [judge,I_final_with_rot]=judge_rot(I_final_with_rot,N,scale,ID);
      [judge,I_denoise_final_with_rot]=judge_rot(I_denoise_final_with_rot,N,scale,ID);
      %         judge
end
for  ID=1:scale*scale
      [judge,I_final_with_rot]=judge_rot(I_final_with_rot,N,scale,ID);
      [judge,I_denoise_final_with_rot]=judge_rot(I_denoise_final_with_rot,N,scale,ID);
      %         judge
end

figure(7)
disp('生成利用先验信息的加噪声的图像恢复结果，ssim有两个选一个大的')
sample_rate=scale*scale%  过采样率
noise_type
noise_energy
subplot(1,2,1)
imshow(I_final_with_rot)
subplot(1,2,2)
imshow(I_denoise_final_with_rot)
ssim_final  =  ssim(I_final_with_rot,sample)
ssim_final  =  ssim(I_final_with_rot,rot90(sample,2))
ssim_final  =  ssim(I_denoise_final_with_rot,sample)
ssim_final  =  ssim(I_denoise_final_with_rot,rot90(sample,2))

%%%%%%%%%%%%%%%%%%%%%%%%%%拼接校正子函
数%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [judge,I_judge]  =  judge_rot(I,N,scale,ID)
%  根据不同过采样率，对图像进行分块,第ID号的分块，并且每一块旋转180度，输出判定为该图像
应不应该旋转，旋转就是0，不旋转是1
m=N/scale;%  图像的分块
windows  =  m;
X=floor((ID-1)/scale)+1;
Y=ID  -  (X-1)*scale;%  i是第M行第N列的图块
maski  =  square_mask(N,m,ID);%  计算掩模
%  I_ID  =  maski.*I;%在图像中加入图像块掩模

I_ID=I(1+windows*(X-1):windows*X,1+windows*(Y-1):windows*Y);%取出图像的ID号的掩模一小块

I_ID_rot  =  rot90(I_ID,2);%  得到ID小块图像的旋转180图像

I_with_rot  =  I;
```

```
I_with_rot(1+windows*(X-1):windows*X,1+windows*(Y-1):windows*Y)=I_ID_rot;%得到将旋转后的ID小块填充回到原图的图

%现在开始计算指标，通过SSIM计算之前之后的图像，与原图的SSIM
Grad_no_rot=sum(sum(imgradient(I)));
Grad_rot = sum(sum(imgradient(I_with_rot)));

if Grad_rot<Grad_no_rot
    judge = 1;
    I_judge=I_with_rot;
else
    judge = 0;
    I_judge=I;
end

% maski = zeros(N,N);%初始化一个mask，mask大小与原图一致
% scale = N/m;
% windows = N/scale;% 缩放后的窗宽
% mask_center=ones(windows,windows);
%
%
% X=floor((ID-1)/scale)+1;
% Y=ID - (X-1)*scale;% i是第M行第N列
% maski(1+windows*(X-1):windows*X,1+windows*(Y-1):windows*Y)=mask_center;
```