东北大学资源与土木工程学院

实 验 报 告

课程名称: 雷达干涉测量

实验项目名称: 干涉图滤波处理

学生专业:测绘工程　　学生班级:测绘 2202

学　　号:20222490　　学生姓名:栗浩宇

指导教师:魏恋欢，敖萌　　实验成绩:

教师评语:

评阅人:　　评阅时间:

目录

1 任务概述

本次编程任务分四个步骤进行，即：进行干涉生成干涉图、去平地效应、对干涉结果进行滤波处理、评价滤波质量。

2 数据介绍

ERS Data
Directory: ERS_Vesuvius
Data: im_m, im_s, dem
Samples: 600 (range)
Records: 2700 (azimuth)

- Wavelength: 0.0566 m
- Sampling frequency: 18.93 MHz
- Baseline: 251 m
- Incidence: angle 23 deg.
- Height: 780Km

本次任务使用 ERS 地球资源卫星对维苏威火山成像的数据，它的基本信息有：波长 0.0566m，采样频率 18.93MHz，基线 251 米，入射角 23°，卫星高度 780 千米。可以将这个数据导入到 matlab 中，可以看到它包括：主影像 im_m、辅影像 im_s 以及 dem 数据，他们都是 2700×600 的矩阵数据。

3 开发环境

本次编程任务使用 matlab2022b 进行，并且因为使用了 mean 和 median 等函数，所以还需要安装 matlab 中的 Image Processing Toolbox 工具箱。

4 算法原理

根据本次的四个步骤，我将本次任务分成七个模块来进行，分别是：干涉图的生成、去平地效应、Goldstein 滤波、均值滤法、中值滤波、自适应中值滤波以及精度定量评价。

4.1 干涉图的生成

SAR 影像中记录的数据是一个复数矩阵，每一个复数由实部和虚部组成，分别记录了幅度和相位信息。而雷达干涉技术是通过捕获同一地区在不同时间的两幅 SAR 影像，并对这两幅图像进行干涉处理以产生干涉图的一种方法，下面进行详细的解释：

$$u_i = |u_i|e^{j\varphi_i} = Re(u_i) + j \cdot Im(u_i)$$

其中，i取 1、2 和 int 代表两幅 SAR 影像复数以及干涉后的影像复数，$|u_i|$代表幅度，φ是相位，$Re(u_i)$代表实部，$Im(u_i)$代表虚部。int 和 1、2 两幅影像之间存在下面的关系：

$$u_{\text{int}} = u_1 u_2^*$$

其中，u_2^*为u_2的共轭复数，故u_{int}是两个复数共轭相乘的结果。相位数据φ计算公式如下：

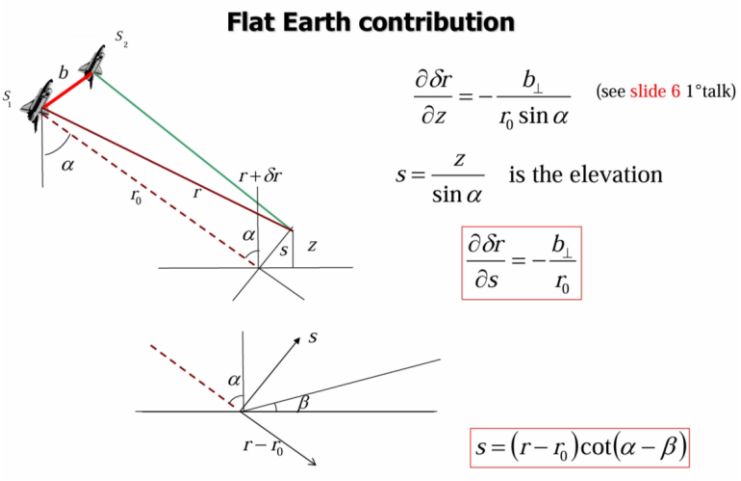
$$\varphi = \arctan\left(\frac{\text{Im}(u_{\text{int}})}{\text{Re}(u_{\text{int}})}\right)$$

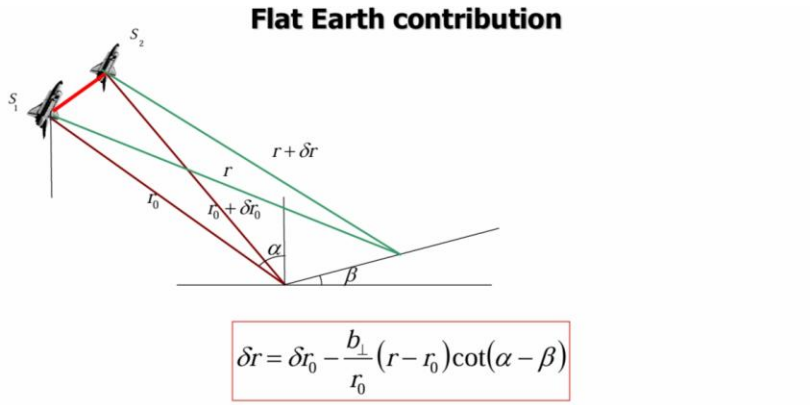
并需对其进行2π取模计算：

$$\varphi_M = \text{mod}(\varphi, 2\pi)$$

4.2 去平地效应

去平地效应可通过下面图中的公式来进行操作：





Flat Earth Removal

MATLAB EXERCISE

ERS Data
Directory: ERS_Vesuvius
Data: im_m, im_s, dem
Samples: 600 (range)
Records: 2700 (azimuth)

- Wavelength: 0.0566 m
- Sampling frequency: 18.93 MHz
- Baseline: 251 m
- Incidence: angle 23 deg.
- Height: 780Km

$$\delta r = \delta r_0 - \frac{b_\perp}{r_0}(r - r_0)\cot(\alpha - \beta)$$

4.3 改进 Goldstein 滤波算法

经典 Goldstein 滤波的原理是将干涉相位图分块，并利用傅里叶变换将干涉相位图变换到频谱域，再对干涉滤波进行平滑处理。在高相干区滤波系数小，滤波强度就小，低相干区滤波系数大，则滤波强度就大。其数学表达式为：

$$\hat{\varphi}(m,n) = IFFT2\{|S(u,v)|_m^\alpha S(u,v)\}$$

其中，$\hat{\varphi}(m,n)$表示滤波后的相位，$S(u,v)$表示干涉相位图$\varphi(m,n)$频谱，下标m表示平滑处理，u 和 v 分别为空间频率。α为滤波参数。

我则是在 Goldstein 滤波这个框架下，实现了一个简易版本。在通过局部窗口处理，将图像分块做 FFT，然后利用简单的 3×3 卷积核计算权重并施加幂运算参数，从而局部加权滤波，替代了传统 Goldstein 算法全局频域滤波的复杂步骤，实现了相位噪声的局部抑制和滤波。

4.4 均值滤波算法

均值滤波到目前为止还是最常用的 InSAR 空域滤波方法之一。其原理是在干涉相位图中取一个滑动窗口，将窗口内的中心像素取平均值，将窗口进行上下滑动，来达到平滑滤波的作用。其数学表达如下所示：

$$\bar{c}(m,n) = \frac{1}{N^2}\Sigma_{m-\frac{N-1}{2}}^{m+\frac{N+1}{2}}\Sigma_{n-\frac{N-1}{2}}^{n+\frac{N+1}{2}}A(i,j)$$

其中，(m,n)为窗口中心像素的坐标，$A(i,j)$为原始干涉纹图在(i,j)处的像素值，$\bar{c}(m,n)$为平均像素值。

均值滤波原理简单，容易实现，滤波效果也比较明显，但是由于均值滤波会

随着滤波窗口的扩大，导致干涉图的边缘保持度减小，如果滤波窗口过大就会造成干涉条纹被破坏，因此滤波窗口大小的选择需在滤波效果和边缘保持中取得平衡。

4.5 中值滤波算法

中值滤波也是目前常用的 InSAR 空域滤波方法之一。与均值滤波不同，中值滤波的基本原理是：在干涉相位图中选取一个滑动窗口，将窗口内所有像素的数值排序，然后取中间的那个值作为窗口中心像素的新值。其数学表达式可以写为：

$$\tilde{c}(m,n) = median\left\{A(i,j)\middle| i = m - \frac{N-1}{2}, \dots, m + \frac{N-1}{2}, j = n - \frac{N-1}{2}, \dots, n + \frac{N-1}{2}\right\}$$

其中，(m,n)表示窗口中心像素的坐标，$A(i,j)$为原始干涉图在处的像素值，而$\tilde{c}(m,n)$则表示中值滤波后的像素值。

中值滤波的优点在于它能够有效抑制由椒盐噪声等脉冲噪声引起的异常值，同时较好地保持图像的边缘特征。由于中值滤波不依赖于所有像素的平均信息，它在面对极端值时能够更稳健地恢复干涉图的纹理信息。然而，中值滤波的计算复杂度较均值滤波高，且在滤波窗口较大时可能会对局部细节造成一定的平滑效果。

4.6 自适应中值滤波算法

自适应中值滤波是中值滤波的一种改进方法，它不仅考虑窗口内像素的中值信息，还通过自适应地调整窗口大小来更好地平衡噪声抑制与细节保留。其基本原理如下：

自适应窗口选择：与固定大小窗口的中值滤波不同，自适应中值滤波从一个较小的窗口开始，在每个窗口内计算中值、最小值和最大值。如果当前窗口的中值既不等于窗口内的最小值也不等于最大值，则认为中值能够代表该局部区域的特征；否则，则增大窗口尺寸（直至达到预设的最大窗口尺寸）以便获得更稳定的统计值。

噪声判断与替换：在确定合适的窗口后，滤波器进一步判断窗口中心像素是否为噪声点。如果该像素超出窗口内的最小值和最大值范围，则将其替换为窗口的中值；否则，保持原值不变。数学上可以表达为：

$$\tilde{c}(m,n) = \begin{cases} median\{A(i,j)\}, & A(m,n) \le \min\{A(i,j)\} \ or \ A(m,n) \ge \max\{A(i,j)\} \\ A(m,n) & , \quad else \end{cases}$$

其中，(m,n)表示窗口中心像素的坐标，$A(i,j)$为原始干涉图在处的像素值，而$\tilde{c}(m,n)$则表示中值滤波后的像素值。

自适应中值滤波通过对局部区域统计特性的动态调整，在平滑噪声的同时有效保留了干涉图中的结构信息，是处理含有脉冲噪声干涉图的一种有效方法。

4.7 精度评价

我通过下面几个指标对我使用的滤波的效果进行了评价，下面会一一进行介绍。

4.7.1 均方误差

MSE 用于衡量经过滤波处理后的干涉图相位与原始（去平地后但未滤波）干涉图相位之间的平均误差。误差越小，说明滤波后的结果与原始参考图越接近，也就说明滤波器在保持原始信息的同时有效去除了噪声。

其计算公式为：

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(\varphi_0(i) - \varphi_f(i)\right)^2$$

其中，φ_0表示原始相位，φ_f滤波后的相位，N 为像素总数。

4.7.2 全局相位残差标准差

该指标反映了滤波后与原始相位之间残差的全局分布情况。先对相位差进行$(-\pi,\pi]$的包裹（即对差值进行模2π调整），再计算其标准差。标准差越小，说明滤波后的相位与原始相位之间的整体误差更为集中，噪声抑制效果较好。

公式：

$$STD_{globle} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\Delta\varphi(i) - \overline{\Delta\varphi})^2}$$

其中，$\Delta\varphi = WrapToPi\left(\varphi_f(i) - \varphi_0(i)\right)$是相位残差，$\overline{\Delta\varphi}$是相位残差均值。

4.7.3 局部 STD 改善率

局部标准差（局部 STD）用于反映图像中较小区域内相位的波动情况。通过在固定大小的滑动窗口内计算相位的标准差，可以评估局部噪声水平或纹理的平滑程度。而通过将前后局部 STD 对比，即可得到 STD 改善率，该指标反映了滤波后局部相位标准差的改善程度，即局部噪声水平降低了多少。改善率越高，说明滤波处理对局部噪声的抑制效果越明显。

其公式如下：

$$LocalSTD = Mean(StdFilt(\varphi(i), w \times w))$$

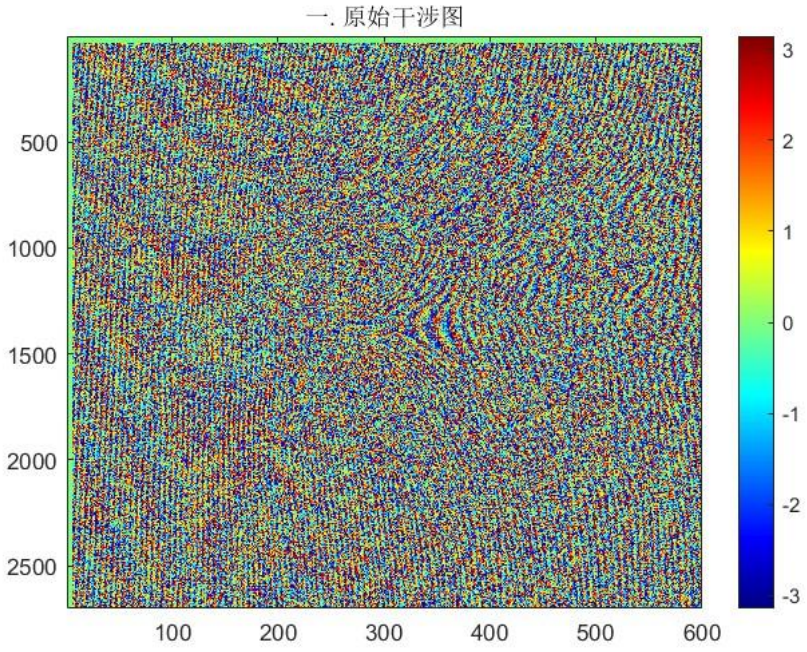
分别对滤波前后做此操作得到$LocalSTD_{before}$与$LocalSTD_{after}$，再计算出改善率：

$$ImprovementRate = \left(1 - \frac{LocalSTD_{after}}{LocalSTD_{before}}\right) \times 100\%$$

5 算法运行结果

5.1 干涉图生成

从 ers_vesuvius.mat 文件中读取主影像 (im_m) 和从影像 (im_s)，并计算干涉图（interferogram = masterImage .* conj(slaveImage)），然后通过 angle() 获取干涉相位。



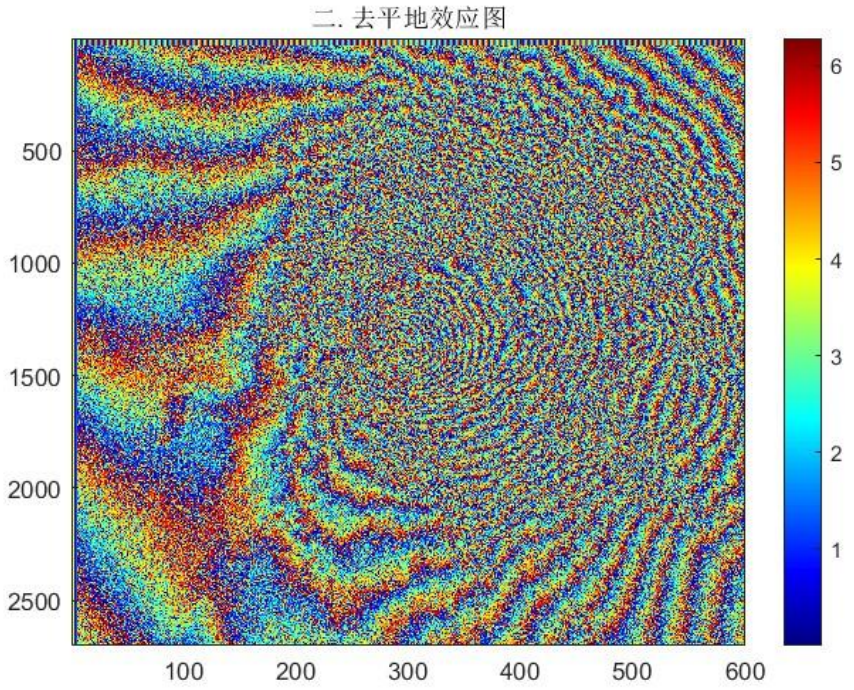
5.2 去平地效应

（1）根据角度 23°和卫星与地面距离 780000m，计算平均距离及每个像素的距离变化。

（2）构造一个距离修正数组，对图像每一列赋予正负不同的修正值。

（3）根据波长、基线长度及其他参数计算平地相位修正项，并将其从原始干涉相位中扣除，再将结果限定在 $[0,2\pi)$ 内。

（4）生成新的复数干涉图：用原始幅值和去平地后的相位重构复数矩阵。

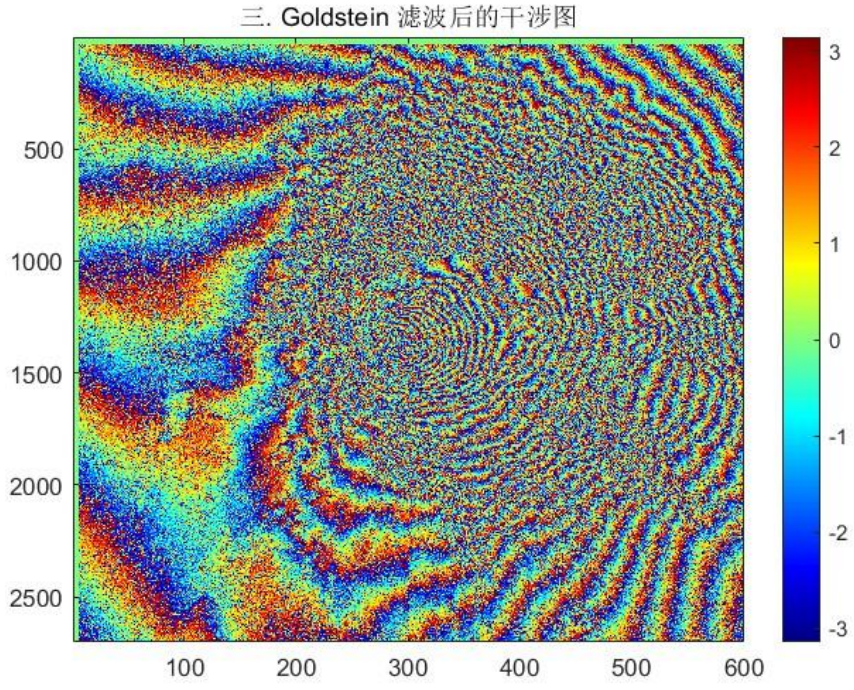


5.3 Goldstein 滤波处理

（1）定义 Goldstein 滤波函数，设置滤波参数（α值、窗口大小、步长）。

（2）在频域内利用简单卷积核构造权重，并在滑动窗口内对每个局部区域进行滤波处理。

（3）将滤波结果合成新的干涉图并显示其相位分布。

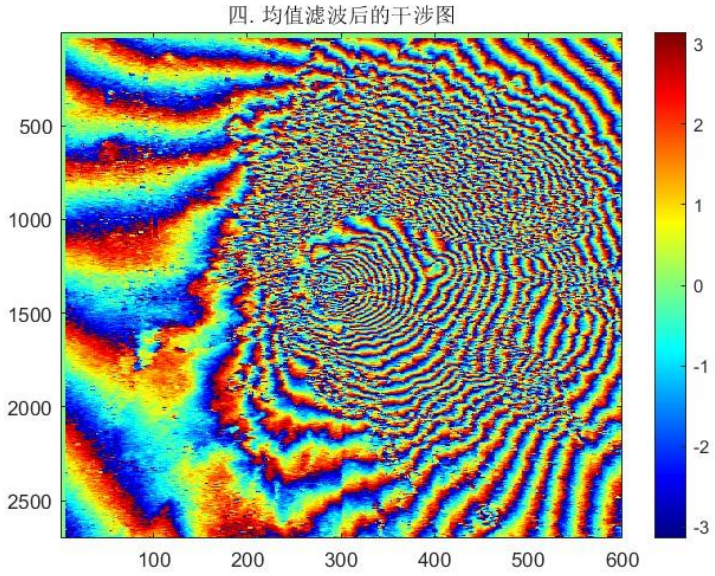


5.4 均值滤波处理

（1）在函数 mean_filter 中，针对每个像素取以窗口内所有像素实部和虚部分别求均值。

（2）考虑边界问题，采用扩充数据（复制边界值）保证所有像素均可滤波。

（3）最后输出复数滤波结果，并展示滤波后相位图。

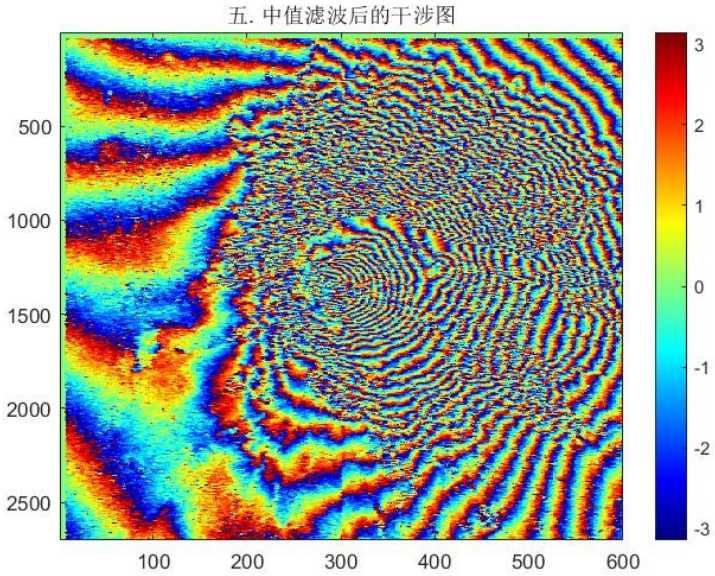


5.5 中值滤波处理

（1）在函数 median_filter 中，对输入干涉图分别对实部和虚部采用滑动窗口的中值运算。

（2）同样扩充数据以处理边界问题。

（3）输出滤波后的复数数据并显示其相位。



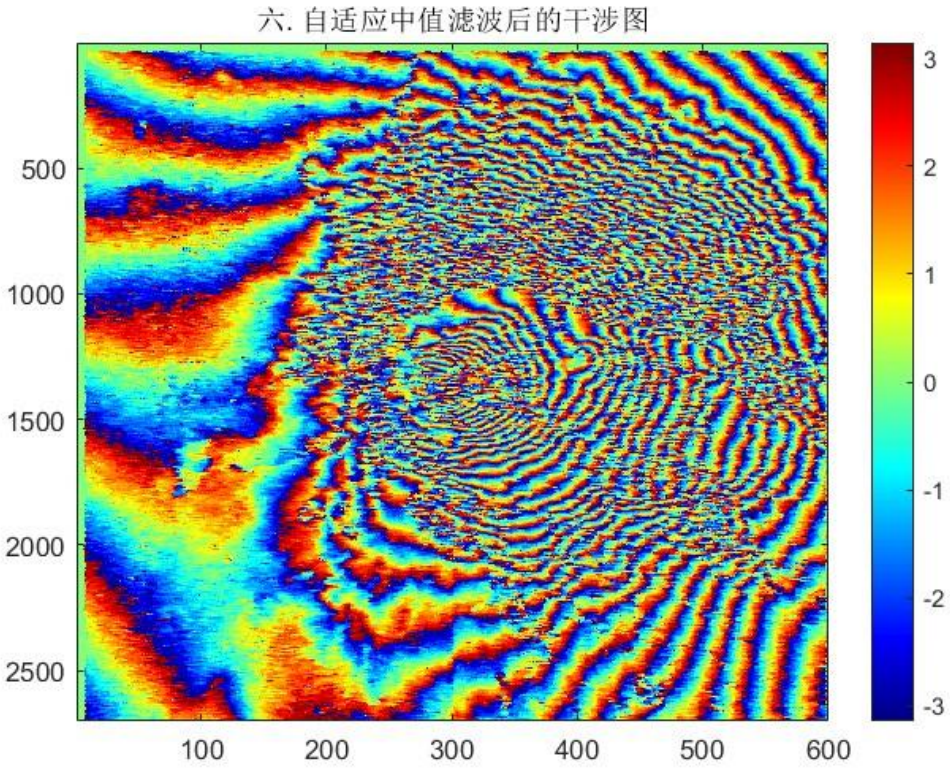
5.6 自适应中值滤波处理

（1）在 adaptive_median_filter 函数中，对每个像素分别对实部和虚部进行处理。

（2）从最小窗口尺寸开始，逐步增大窗口至最大尺寸，直至找到合适的中值（满足局部统计特性）。

（3）若当前像素处于局部极值，则替换为中值；否则保持原值或采用最大窗口下的中值。

（4）最终输出经过自适应中值滤波处理后的复数干涉图，并展示相位分布。



5.7 定量评价 – MSE 和 STD 计算

（1）以“去平地后但未滤波”的干涉图作为参考图，与滤波后的结果作比较。

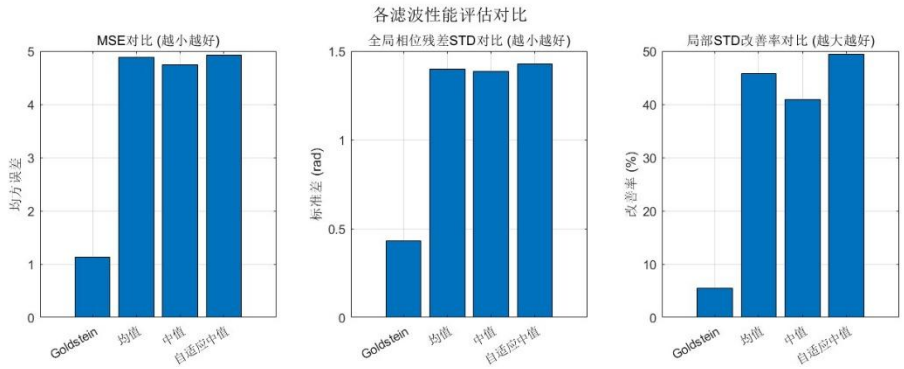
（2）MSE 计算：衡量原始与滤波相位之间的均方误差。

（3）全局相位残差 STD：通过包裹相位差计算全局误差的标准差。

（4）局部相位 STD：利用局部窗口计算滤波前后的局部标准差，并对比改善率。

（5）最后，分别打印四种滤波方法的评价指标，并绘制对比图展示 MSE、全局 STD 以及局部 STD 改善率的变化。

柱状图：



控制台打印结果：

```
Goldstein 滤波评估结果:
  - MSE: 1.1254
  - 全局相位残差 STD: 0.4313 rad
  - 局部相位 STD: 前 1.6818 → 后 1.5907
  - STD 改善率: 5.42%

均值滤波评估结果:
  - MSE: 4.8824
  - 全局相位残差 STD: 1.3981 rad
  - 局部相位 STD: 前 1.6818 → 后 0.9129
  - STD 改善率: 45.72%

中值滤波评估结果:
  - MSE: 4.7422
  - 全局相位残差 STD: 1.3837 rad
  - 局部相位 STD: 前 1.6818 → 后 0.9932
  - STD 改善率: 40.94%

自适应中值滤波评估结果:
  - MSE: 4.9284
  - 全局相位残差 STD: 1.4262 rad
  - 局部相位 STD: 前 1.6818 → 后 0.8521
  - STD 改善率: 49.33%
```

通过分析上面的结果，我得到了如下结论：

Goldstein 滤波：在保持整体相位一致性和较低全局误差方面表现突出（低 MSE 和低全局 STD），但在局部噪声抑制方面较弱，适合需要精确保持相位信息的场景。

均值、中值及自适应中值滤波：均在局部噪声抑制上有较大改善（局部 STD 大幅降低），但代价是引入较大的全局误差和更高的 MSE。自适应中值滤波在局部改善方面效果最好，适合在对局部噪声要求更高的应用中使用，但需要接受全局相位上较大变化的事实。

下面是对他们**优缺点**的总结：

Goldstein 滤波：优点：能保持与原始相位的整体吻合，MSE 和全局相位残差最小，干涉条纹细节保存度较好。缺点：局部平滑能力相对较弱，降低噪

声的效果不如均值/中值滤波明显

均值 / 中值 / 自适应中值滤波：优点：大幅降低局部相位噪声（局部 STD 改善率高），其中自适应中值滤波在去除随机噪声上最为强力。缺点：滤除噪声的同时，也会明显改变原始干涉相位，导致全局误差偏高、保真度下降（MSE 和全局 STD 较大）。

6 附录（全部代码）

```
clc;
clear;

%% 1. 干涉图生成
% --------------------------
% 说明：
%   该部分加载雷达图像数据，并根据主图像与从图像计算干涉图，
%   继而提取干涉相位，并以伪彩色图显示原始干涉图相位。
%
% 数据加载
load('C:\Users\栗浩宇\Desktop\Insar 滤波\ers_vesuvius.mat');

% 从加载的数据中提取主图像（im_m）与从图像（im_s）
masterImage = im_m;
slaveImage  = im_s;

% 计算复数形式的干涉图：主图像乘以从图像的共轭
interferogram = masterImage .* conj(slaveImage);

% 提取干涉相位（以弧度表示）
interferogram_phase = angle(interferogram);

% 显示原始干涉图相位
figure;
imagesc(interferogram_phase);
colormap('jet');
colorbar;
title('一. 原始干涉图');

%% 2. 去平地效应（消除平地效应）
% --------------------------
% 说明：
%   为消除因卫星几何结构产生的平地效应，
%   本节计算每个像素对应的距离变化，并由此生成相位修正，
%   再将其应用于原始干涉相位，确保结果相位位于 [0, 2π) 区间。
```

```
% 计算卫星至地面平均距离（考虑入射角 23°）
averageDistance = 780000 / cos(23 * pi / 180);

% 计算单个像素对应的距离变化（依据雷达波在真空中传播速度是光速 299792458m/s 及采样频率 18.93MHz）
distancePerPixel = 299792458 / (2 * 18930000);

% 获取图像的列数
numberOfColumns = size(im_m, 2);

% 初始化距离修正数组（用于存储每列的距离偏移）
distanceCorrection = zeros(1, numberOfColumns);

% 对图像每一列计算距离修正值：
%   - 左半部分为正方向修正
%   - 右半部分为负方向修正
for i = 1:(numberOfColumns / 2)
    distanceCorrection(1, numberOfColumns/2 + 1 - i) = distancePerPixel * i;  % 正向修正
    distanceCorrection(1, numberOfColumns/2 + i)     = -distancePerPixel * i; % 负向修正
end
% 将一维距离修正数组扩展为与图像大小一致的二维矩阵
distanceCorrection = repmat(distanceCorrection, size(im_m, 1), 1);

% 计算平地效应相位修正：
%   使用参数：波长 = 0.0566m，基线 = 251m，入射角 = 23°
flatPhaseCorrection = (4 * pi * 251 * distanceCorrection) ...
                    / (0.0566 * averageDistance * tan(23 * pi / 180));

% 应用相位修正，获得校正后的干涉相位
correctedInterferogramPhase = interferogram_phase - flatPhaseCorrection;
% 确保相位值落在 [0, 2π) 区间内
correctedInterferogramPhase = mod(correctedInterferogramPhase, 2 * pi);

% 显示去平地效应后的干涉图相位
figure;
imagesc(correctedInterferogramPhase);
colormap('jet');
colorbar;
title('二. 去平地效应图');

% 组合幅值与校正后的相位，构造新的复数干涉图
```

```
correctedInterferogram = abs(interferogram) .* exp(1i *
correctedInterferogramPhase);

%% 3. Goldstein 滤波处理
% --------------------------
% 说明：
%   使用 Goldstein 滤波方法对校正后的干涉图进行相位滤波，
%   以减小噪声并改善相位连续性。
%
% 设置滤波参数：
alpha       = 0.5;  % Goldstein 滤波的幂指数参数
window_size = 5;    % 滑动窗口的尺寸
step_size   = 1;    % 滑动窗口的步长

% 调用 Goldstein 滤波函数，返回滤波后的复数干涉图
filteredInterferogram_Goldstein = goldstein_filter(correctedInterferogram,
alpha, window_size, step_size);

% 显示 Goldstein 滤波后干涉图的相位
figure;
imagesc(angle(filteredInterferogram_Goldstein));
colormap('jet');
colorbar;
title('三. Goldstein 滤波后的干涉图');

%% 4. 均值滤波处理
% --------------------------
% 说明：
%   采用均值滤波对校正后的干涉图进行处理，通过计算滑动窗口内
%   实部和虚部的均值来抑制噪声。
%
% 设置均值滤波的窗口大小（可与 Goldstein 滤波的窗口大小相同或不同）
mean_window_size = 5;

% 调用均值滤波函数，返回滤波后的复数干涉图
filteredInterferogram_Mean = mean_filter(correctedInterferogram,
mean_window_size);

% 显示均值滤波后干涉图的相位
figure;
imagesc(angle(filteredInterferogram_Mean));
colormap('jet');
colorbar;
title('四. 均值滤波后的干涉图');
```

```
%% 5. 中值滤波处理
% --------------------------
% 说明：
%   使用中值滤波对校正后的干涉图进行处理，
%   分别对实部和虚部采用中值滤波以有效去除椒盐噪声。
%
% 调用中值滤波函数，返回滤波后的复数干涉图
filteredInterferogram_Median = median_filter(correctedInterferogram,
mean_window_size);

% 显示中值滤波后干涉图的相位
figure;
imagesc(angle(filteredInterferogram_Median));
colormap('jet');
colorbar;
title('五. 中值滤波后的干涉图');

%% 6. 自适应中值滤波处理
% --------------------------
% 说明：
%   自适应中值滤波根据局部统计特性自动调整窗口尺寸，
%   分别对干涉图的实部和虚部进行处理，
%   以在保留细节的同时有效抑制噪声。
%
% 设置自适应中值滤波的参数：
wmin = 5;   % 最小窗口尺寸
wmax = 7;   % 最大窗口尺寸

% 调用自适应中值滤波函数，返回滤波后的复数干涉图
filteredInterferogram_AdaptiveMedian =
adaptive_median_filter(correctedInterferogram, wmin, wmax);

% 显示自适应中值滤波后干涉图的相位
figure;
imagesc(angle(filteredInterferogram_AdaptiveMedian));
colormap('jet');
colorbar;
title('六. 自适应中值滤波后的干涉图');

%% 7. 定量评价 - 计算 MSE 和 STD 指标
% --------------------------
% 说明：
%   为比较各滤波方法的效果，本节采用以下评价指标：
```

```
%   1) 均方误差（MSE）—衡量滤波前后相位的整体误差；
%   2) 全局相位残差标准差（STD）—反映全图相位一致性；
%   3) 局部相位标准差—通过滑动窗口计算局部区域相位波动，
%      进而得到改善率。
%
% 评估窗口大小设置：
eval_window_size = 5;

% 使用“去平地后但尚未滤波”的干涉图作为参考图，
% 分别对四种滤波结果进行评估。

% 1. Goldstein 滤波的评估
[mse_goldstein, std_global_goldstein, std_local_before_goldstein,
std_local_after_goldstein] = ...
    evaluate_filter_performance(correctedInterferogram,
filteredInterferogram_Goldstein, eval_window_size);
fprintf('Goldstein 滤波评估结果:\n');
fprintf('  - MSE: %.4f\n', mse_goldstein);
fprintf('  - 全局相位残差 STD: %.4f rad\n', std_global_goldstein);
fprintf('  - 局部相位 STD: 前 %.4f rad → 后 %.4f rad\n',
std_local_before_goldstein, std_local_after_goldstein);
fprintf('  - STD 改善率: %.2f%%\n', (1 -
std_local_after_goldstein/std_local_before_goldstein) * 100);

% 2. 均值滤波的评估
[mse_mean, std_global_mean, std_local_before_mean, std_local_after_mean]
= ...
    evaluate_filter_performance(correctedInterferogram,
filteredInterferogram_Mean, eval_window_size);
fprintf('\n 均值滤波评估结果:\n');
fprintf('  - MSE: %.4f\n', mse_mean);
fprintf('  - 全局相位残差 STD: %.4f rad\n', std_global_mean);
fprintf('  - 局部相位 STD: 前 %.4f rad → 后 %.4f rad\n',
std_local_before_mean, std_local_after_mean);
fprintf('  - STD 改善率: %.2f%%\n', (1 -
std_local_after_mean/std_local_before_mean) * 100);

% 3. 中值滤波的评估
[mse_median, std_global_median, std_local_before_median,
std_local_after_median] = ...
    evaluate_filter_performance(correctedInterferogram,
filteredInterferogram_Median, eval_window_size);
fprintf('\n 中值滤波评估结果:\n');
fprintf('  - MSE: %.4f\n', mse_median);
```

```
fprintf('  - 全局相位残差 STD: %.4f rad\n', std_global_median);
fprintf('  - 局部相位 STD: 前 %.4f rad → 后 %.4f rad\n',
std_local_before_median, std_local_after_median);
fprintf('  - STD 改善率: %.2f%%\n', (1 -
std_local_after_median/std_local_before_median) * 100);

% 4. 自适应中值滤波的评估
[mse_adaptive_median, std_global_adaptive, std_local_before_adaptive,
std_local_after_adaptive] = ...
    evaluate_filter_performance(correctedInterferogram,
filteredInterferogram_AdaptiveMedian, eval_window_size);
fprintf('\n自适应中值滤波评估结果:\n');
fprintf('  - MSE: %.4f\n', mse_adaptive_median);
fprintf('  - 全局相位残差 STD: %.4f rad\n', std_global_adaptive);
fprintf('  - 局部相位 STD: 前 %.4f rad → 后 %.4f rad\n',
std_local_before_adaptive, std_local_after_adaptive);
fprintf('  - STD 改善率: %.2f%%\n', (1 -
std_local_after_adaptive/std_local_before_adaptive) * 100);

% 5. 绘制各滤波方法评估结果对比图
figure;

% 定义各滤波方法名称与对应评价指标数据
filter_names = {'Goldstein', '均值', '中值', '自适应中值'};
mse_values         = [mse_goldstein, mse_mean, mse_median,
mse_adaptive_median];
std_global_values   = [std_global_goldstein, std_global_mean,
std_global_median, std_global_adaptive];
std_improvement     = [
    (1 - std_local_after_goldstein/std_local_before_goldstein) * 100, ...
    (1 - std_local_after_mean    /std_local_before_mean)    * 100, ...
    (1 - std_local_after_median  /std_local_before_median)  * 100, ...
    (1 - std_local_after_adaptive/std_local_before_adaptive) * 100
];

% 子图 1：MSE 对比（越小越好）
subplot(1, 3, 1);
bar(mse_values);
set(gca, 'XTickLabel', filter_names);
title('MSE 对比 (越小越好)');
ylabel('均方误差');
grid on;

% 子图 2：全局相位残差 STD 对比（越小越好）
```

```
subplot(1, 3, 2);
bar(std_global_values);
set(gca, 'XTickLabel', filter_names);
title('全局相位残差 STD 对比 (越小越好)');
ylabel('标准差 (rad)');
grid on;

% 子图 3：局部 STD 改善率对比（越大越好）
subplot(1, 3, 3);
bar(std_improvement);
set(gca, 'XTickLabel', filter_names);
title('局部 STD 改善率对比 (越大越好)');
ylabel('改善率 (%)');
grid on;

% 调整整体图形尺寸与布局
set(gcf, 'Position', [100, 100, 1200, 400]);
sgtitle('各滤波方法性能评估对比');

%% 均值滤波函数
% 说明：
%   对输入的复数干涉图分别对其实部与虚部采用滑动窗口均值滤波，
%   并对边界点进行扩充处理，确保滤波结果的连续性与稳定性。
%
% 输入参数：
%   input_data  - 待滤波的干涉图（复数矩阵）
%   window_size - 滑动窗口尺寸（正整数）
%
% 输出：
%   out_data    - 均值滤波后的干涉图（复数矩阵）
function out_data = mean_filter(input_data, window_size)
    [rows, cols] = size(input_data);
    mid = floor(window_size / 2);
    out_data = input_data;  % 初始化输出矩阵

    % 创建数据副本，将 NaN 值置为 0，防止影响均值计算
    data_copy = input_data;
    data_copy(isnan(data_copy)) = 0;

    % 对图像进行边界扩充，采用复制边界法
    expanded_data = padarray(data_copy, [mid mid], 'replicate', 'both');

    % 分离扩充数据的实部与虚部
    real_part = real(expanded_data);
```

```
    imag_part = imag(expanded_data);

    % 对图像每个像素点采用滑动窗口计算均值
    for i = 1:rows
        for j = 1:cols
            real_window = real_part(i:i+2*mid, j:j+2*mid);
            imag_window = imag_part(i:i+2*mid, j:j+2*mid);

            real_mean = mean(real_window(:));
            imag_mean = mean(imag_window(:));

            out_data(i, j) = complex(real_mean, imag_mean);
        end
    end

    % 处理特殊情况：若原始相位为 0 或 NaN，则对应输出也置为 0 或 NaN
    idx = angle(input_data) == 0;
    out_data(idx) = 0;
    idx = isnan(angle(input_data));
    out_data(idx) = nan;
end

%% 中值滤波函数
% 说明：
%   分别对输入复数干涉图的实部与虚部进行滑动窗口中值滤波，
%   采用边界扩充法处理图像边缘，降低噪声影响。
%
% 输入参数：
%   input_data  - 待滤波的干涉图（复数矩阵）
%   window_size - 滑动窗口尺寸（正整数）
%
% 输出：
%   out_data    - 中值滤波后的干涉图（复数矩阵）
function out_data = median_filter(input_data, window_size)
    [rows, cols] = size(input_data);
    mid = floor(window_size / 2);
    out_data = input_data;  % 初始化输出矩阵

    % 创建数据副本，将 NaN 值置为 0
    data_copy = input_data;
    data_copy(isnan(data_copy)) = 0;

    % 对图像进行边界扩充（复制边界）
    expanded_data = padarray(data_copy, [mid mid], 'replicate', 'both');
```

```
    % 分离实部与虚部
    real_part = real(expanded_data);
    imag_part = imag(expanded_data);

    % 对每个像素使用滑动窗口分别计算实部与虚部的中值
    for i = 1:rows
        for j = 1:cols
            real_window = real_part(i:i+2*mid, j:j+2*mid);
            imag_window = imag_part(i:i+2*mid, j:j+2*mid);

            real_median = median(real_window(:));
            imag_median = median(imag_window(:));

            out_data(i, j) = complex(real_median, imag_median);
        end
    end

    % 同样处理特殊情况：相位为 0 或 NaN 的位置
    idx = angle(input_data) == 0;
    out_data(idx) = 0;
    idx = isnan(angle(input_data));
    out_data(idx) = nan;
end

%% 自适应中值滤波函数
% 说明：
%   自适应中值滤波针对每个像素根据局部窗口内的统计特性动态调整窗口尺寸，
%   分别对实部和虚部进行处理，以更好地保留图像细节并抑制噪声。
%
% 输入参数：
%   input_data - 待滤波的干涉图（复数矩阵）
%   wmin       - 最小窗口尺寸（奇数）
%   wmax       - 最大窗口尺寸（奇数）
%
% 输出：
%   out_data   - 自适应中值滤波后的干涉图（复数矩阵）
function out_data = adaptive_median_filter(input_data, wmin, wmax)
    [rows, cols] = size(input_data);
    out_data = input_data;  % 初始化输出矩阵

    % 处理 NaN 值
    data_copy = input_data;
    data_copy(isnan(data_copy)) = 0;
```

```
    % 分离实部与虚部
    real_part = real(data_copy);
    imag_part = imag(data_copy);

    % 为保证窗口完全覆盖，计算扩充尺寸
    pad_size = (wmax-1)/2;
    expand_real = padarray(real_part, [pad_size pad_size], 0, 'both');
    expand_imag = padarray(imag_part, [pad_size pad_size], 0, 'both');

    % 对图像每个像素分别自适应处理中、实部与虚部
    for i = 1:rows
        for j = 1:cols
            real_pixel_replaced = false;
            imag_pixel_replaced = false;
            real_result = real_part(i, j);
            imag_result = imag_part(i, j);

            % 对实部进行自适应中值滤波
            for n = wmin:2:wmax
                half_window = (n-1)/2;
                S_real = expand_real(i : i+2*half_window, j :
j+2*half_window);
                real_max = max(S_real(:));
                real_min = min(S_real(:));
                real_med = median(S_real(:));

                % 若中值位于极值之间，则判断当前像素是否为异常值
                if real_med > real_min && real_med < real_max
                    if real_part(i,j) <= real_min || real_part(i,j) >=
real_max
                        real_result = real_med;
                        real_pixel_replaced = true;
                    end
                    break;
                end
            end
            % 若未替换，则采用最大窗口的中值
            if ~real_pixel_replaced
                half_window = (wmax-1)/2;
                S_real = expand_real(i : i+2*half_window, j :
j+2*half_window);
                real_result = median(S_real(:));
            end
```

```
            % 对虚部进行自适应中值滤波（方法同上）
            for n = wmin:2:wmax
                half_window = (n-1)/2;
                S_imag = expand_imag(i : i+2*half_window, j :
j+2*half_window);
                imag_max = max(S_imag(:));
                imag_min = min(S_imag(:));
                imag_med = median(S_imag(:));

                if imag_med > imag_min && imag_med < imag_max
                    if imag_part(i,j) <= imag_min || imag_part(i,j) >=
imag_max
                        imag_result = imag_med;
                        imag_pixel_replaced = true;
                    end
                    break;
                end
            end
            if ~imag_pixel_replaced
                half_window = (wmax-1)/2;
                S_imag = expand_imag(i : i+2*half_window, j :
j+2*half_window);
                imag_result = median(S_imag(:));
            end

            % 合成处理后的复数结果
            out_data(i,j) = complex(real_result, imag_result);
        end
    end

    % 特殊情况处理：相位为 0 或 NaN 的位置
    idx = angle(input_data) == 0;
    out_data(idx) = 0;
    idx = isnan(angle(input_data));
    out_data(idx) = nan;
end

%% Goldstein 滤波函数
% 说明：
%   Goldstein 滤波是一种基于频域加权的方法，本函数展示一种简化的
%   反距离加权 Goldstein 滤波思路。实际应用中可能需更复杂处理。
%
% 输入参数：
```

```
%   input_data  - 待滤波的干涉图复数矩阵
%   alpha       - 滤波参数，用于控制加权因子的幂次
%   window_size - 滑动窗口尺寸
%   step_size   - 滑动窗口步长
%
% 输出:
%   out_data    - Goldstein 滤波后的干涉图复数矩阵
function out_data = goldstein_filter(input_data, alpha, window_size,
step_size)
    % 构造简易卷积核，并通过傅里叶变换获取其频域形式
    K = ones(3, 3);
    K = K / sum(K(:));
    K = fftshift(fft2(K));

    [rows, cols] = size(input_data);
    out_data = zeros(rows, cols);  % 初始化输出矩阵

    % 若 window_size 为奇数，则将其调整为偶数（便于窗口分割）
    if mod(window_size,2) ~= 0
        window_size = window_size - 1;
    end

    % 构造简化版的权重矩阵
    x = 1:window_size/2;
    [X, Y] = meshgrid(x, x);
    X = X + Y;
    weight = [X, fliplr(X)];
    weight = [weight; flipud(weight)];

    % 滑动窗口遍历整个图像
    for ii = 1:step_size:rows
        for jj = 1:step_size:cols
            mm = ii + window_size - 1;
            if mm > rows, mm = rows; end
            nn = jj + window_size - 1;
            if nn > cols, nn = cols; end

            % 取当前窗口数据
            window = input_data(ii:mm, jj:nn);
            ww = weight(1:(mm-ii+1), 1:(nn-jj+1));

            % 计算窗口数据的傅里叶变换，并移到频域中心
            H = fft2(window);
            H = fftshift(H);
```

```
            % 对幅值进行局部卷积平滑，再进行 alpha 加权
            S = conv2(abs(H), K, 'same');
            S = S / max(S(:) + eps);
            S = S .^ alpha;

            % 加权后反变换回时域
            H = H .* S;
            H = ifftshift(H);
            window_filt = ifft2(H);

            % 累计滤波结果（考虑权重叠加）
            out_data(ii:mm, jj:nn) = out_data(ii:mm, jj:nn) + window_filt .*
ww;
        end
    end

    % 特殊情况处理：相位为 0 或 NaN 的像素
    idx = angle(input_data) == 0;
    out_data(idx) = 0;
    idx = isnan(angle(input_data));
    out_data(idx) = nan;
end

%% 评估函数：计算 MSE 和 STD 指标
% 说明：
%   该函数对比原始（去平地但未滤波）干涉图与滤波后干涉图，
%   分别计算：
%     1) 均方误差（MSE），衡量整体相位误差；
%     2) 全局相位残差标准差（global_std）；
%     3) 局部相位标准差（local_std），通过滑动窗口求局部统计平均，
%        用于量化噪声抑制效果。
%
% 输入参数：
%   original_interferogram - 去平地后但未滤波的干涉图（复数矩阵）
%   filtered_interferogram - 滤波后的干涉图（复数矩阵）
%   window_size            - 计算局部标准差的窗口尺寸
%
% 输出：
%   mse             - 均方误差（MSE）
%   global_std      - 全局相位残差的标准差
%   local_std_before- 滤波前局部相位标准差的均值
%   local_std_after - 滤波后局部相位标准差的均值
```

```
function [mse, global_std, local_std_before, local_std_after] =
evaluate_filter_performance(original_interferogram, filtered_interferogram,
window_size)
    % 提取原始与滤波后干涉图的相位信息
    original_phase = angle(original_interferogram);
    filtered_phase = angle(filtered_interferogram);

    % 1. 计算均方误差（MSE）
    mse = mean((original_phase(:) - filtered_phase(:)).^2, 'omitnan');

    % 2. 计算全局相位残差标准差（对相位差进行包裹到 [-pi, pi] 范围）
    phase_diff = wrapToPi(filtered_phase - original_phase);
    global_std = std(phase_diff(:), 'omitnan');

    % 3. 计算局部相位标准差（采用 stdfilt 函数在滑动窗口内计算标准差后取均值）
    local_std_before = stdfilt(original_phase, true(window_size));
    local_std_after  = stdfilt(filtered_phase, true(window_size));

    local_std_before = mean(local_std_before(:), 'omitnan');
    local_std_after  = mean(local_std_after(:), 'omitnan');
end
```