%%%%%%%%%%%%%%%%%%%%%%%%%%%%

多视处理

“非相干叠加”

报告 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

仿真程序基于：

1. 基于 RD 算法：正侧视，不考虑 SRC，点目标仿真。采用 RDA 成像，进行“非相干叠加”的多视处理（四视处理）；
2. 基于 CS 算法：利用 Radarsat-1 的实际数据，通过 CSA 成像，分别进行以下两种方式的“非相干叠加”多视处理：
   1) 选取 3×3 窗口，进行平滑滤波（滑动平均），以此抑制相干斑；
   2) 进行“非相干叠加”的多视处理（四视处理），方法同 RD 的点目标仿真。

参考书籍：

1. 《合成孔径雷达成像——算法与实现》，（美）卡明等著；洪文等译；电子工业出版社；
2. 《合成孔径雷达——系统与信号处理》，（美）柯兰德等著；韩传钊等译；电子工业出版社。

WD

2014.10.31.晚

一、 理论原理

“在 SAR 系统中，多个独立的视可以由飞行载体以不同的方位角通过观察点时获得。第一视由天线沿方位向第一个前向四分之一波束部分产生，下一视则来自下一个四分之一波束，以此类推。然后，由于来自波束各部分的信号到达雷达接收机是重叠在一起的，所以在时域或者空域上无法对数据进行分离。然而，具有高方位时间带宽积的一个实用 SAR 系统是将时间和频率两者绑定在一起的，在多普勒域内包含了各视的所有信息。也就是说，具有较高多普勒频率的数据一定是由方位向波束前缘触及到的地形点产生的，而当同一地点出在方位波束后缘四分之一时，产生了多普勒频段低四分之一部分。”[1]

“为了在多普勒域中产生这些独立的视，在完成距离徙动校正之后，在做过距离压缩的每个距离单元上分析多普勒频谱，即在方位压缩之前要分析频谱。也就是说，在方位压缩前用滤波器将频谱分成四个子频带，以避免在方位向时间上出现旁瓣，而又要有某种程度的重叠，以避免损失太多的信号能量，但为了保持各视之间的独立性，重叠的程度还不能太大。”[1]

此外，参考书[1]中在 150 页“**多视处理**”部分叙述了两种情况（我认为是两种不同的实现方式），分别如下：

1. “假如处理器要求具备产生单视图像的能力，则必须在方位时间变量上进行足够长度的 FFT 运算，以产生多普勒全带宽的数据”，“每视使用单视全频带压缩滤波器，以形成 L 视滤波器并用于方位多普勒数据。由于在多普勒频率上被压缩的标称数据只有单视数据带宽的$1/L$，所以对于单视图像只要采用$1/L$的采样率就足够了。减小该采样率的方法是对被压缩数据进行N/L点的 IFFT，在此假设原来的单视频谱是用 N 点变换得到的”，“**在这种方法中，各单视图像的慢时间配准是自动完成的，**因为每视的压缩滤波器精确地保留着合适的相位函数，使图像像素能够放置在合适的方位位置上。”[1]

   对于这种情况，我认为简单来说，方法（理论）就是：

   方位 FFT 用全频带，并进行方位匹配滤波（即在方位 IFFT 前和单视处理是完全一致的）。

   然后，对距离多普勒域的频谱进行多视分割（比如分成 N 份），再进行N/L点的 IFFT。

   这样便能得到 N 个子视图像。并且，由上所述，这种情况下的各子视图像是自动配准的。因此不需要额外操作。这个时候只需要非相干叠加，即可得到多视处理后的图像。

我认为这个方法和卡明的书中叙述的多视处理[2]的方法是一致的（见 179 页到 188 页）。其中一些关键的内容如下：

“如图 6.24 所示，方位多视压缩与单视处理的区别在于**子视抽取、检测及求和**。**首先对全部频谱进行匹配滤波**。这仅为一个“相位滤波器”，目的是从全部频谱中取出二次相位。**然后对各子视进行“幅度滤波”（见图 6.22），以提取所需的频谱部分。**幅度函数包含了用于控制旁瓣的加权，子视见的重叠一般在 10%~20%”，“此后对每视进行 IFFT，以完成压缩” [2]

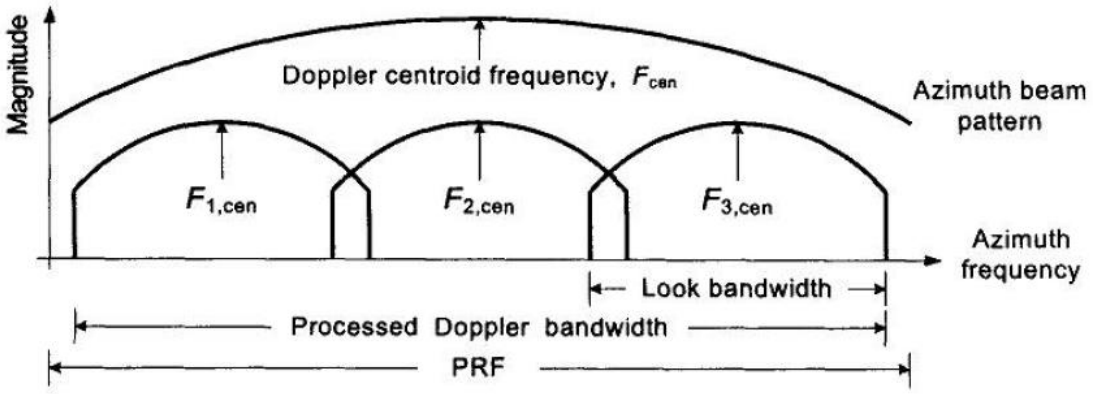


Figure 6.22: Location of the look extraction filters in the azimuth frequency domain (three-look case).

图 6.22 方位频域中的子视滤波器位置（三视）[2]

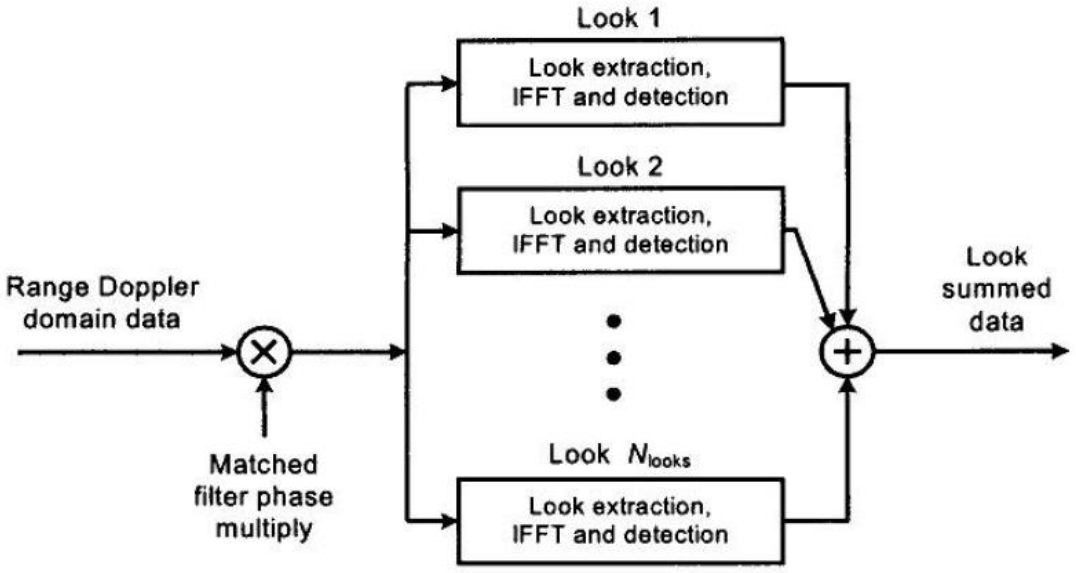


Figure 6.24: Processing steps in multilook azimuth compression.

图 6.24 方位多视压缩处理步骤[2]

“由于全部频谱中的匹配滤波相位使用的是共同的相位函数，只要匹配滤波器调频率正确，每块子视数据就会自动对齐”[2]

由参考书[1, 2]，我们很清楚的了解了这种方法，流程如下：

1) 在方位 IFFT 前，与单视处理完全相同。即直到在距离多普勒域，进行了相位相乘（补偿二次相位，实现方位 MF）。此时只剩下方位 IFFT。若直接进行方位 IFFT，即为单视处理；
2) 对距离多普勒域频谱进行频谱划分，得到相应的 N 个子视频谱；
3) 分别进行 IFFT，得到 N 幅子视图像；
4) 对得到的 N 幅子视图像进行非相干叠加，即得到最终的“多视处理”后的图像。

非相干叠加可以采取以下两种方式。另 $s(k,n)$ 为第 k 视中第 n 个采样点上的复数据，全部视数为 N_{looks}， $s_{ls}(n)$ 是子视求和后的数据。方法如下：

a) 一种方法是对每视中的像素幅度进行求和，再平均：

$$s_{ls}(n)=\frac{1}{N_{looks}}\sum_{k=1}^{N_{looks}}\left|s(k,n)\right|$$

b) 另一种方法是对子视幅度平方求和，平均，再开方

$$s_{ls}(n)=\sqrt{\frac{1}{N_{looks}}\sum_{k=1}^{N_{looks}}\left|s(k,n)\right|^2}=\frac{1}{\sqrt{N_{looks}}}\sqrt{\sum_{k=1}^{N_{looks}}\left|s(k,n)\right|^2}$$

这两种方法来源与参考书[2]182 页上的公式（6.37）和公式（6.38），但我进行了一些修改。值得说明的是，我在这次仿真中，采用的就是这种方法。

后续点目标仿真和 Radarsat-1 的成像仿真都基于这种方法。

2. “另一方面，如果不需要形成单视图像，则可以节省处理器的一些计算量和存储器。在这种情况下，任意时刻所需的最大一组多普勒频率数据是与多视图像中某一视图像的带宽相对应的。”[1]

我认为，这就是说，这种方法需要提前在时域进行多视数据选取，然后 FFT 到距离多普勒域时已经是按照子视进行的了。也就是说，FFT 和 IFFT 都是长度为 N/L（子视长度）。

“假如此采用上一段所述的分段实现的方法，则在叠加各视图像之前必须根据各视图像各自的频段进行补偿”[1]。

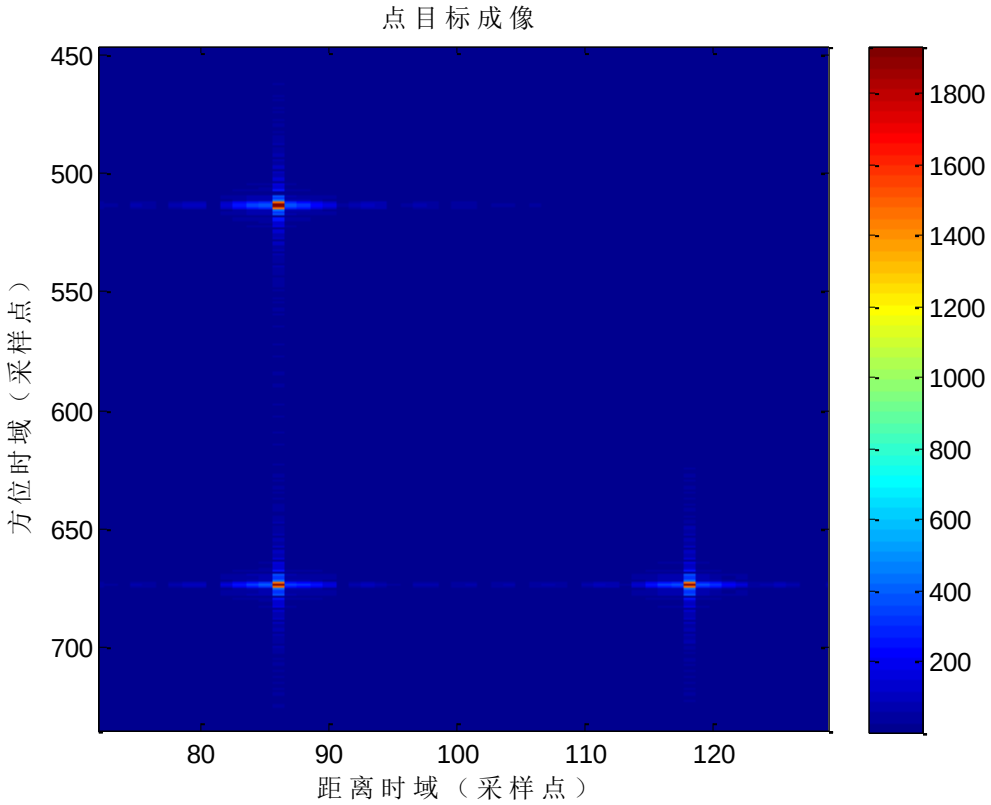
这种方法，我暂时还没有完全明白，也没有进行相应的仿真。

二、 利用 RDA，进行正侧视情况下，忽略 SRC 的点目标仿真：仿真“非相干叠加”多视处理

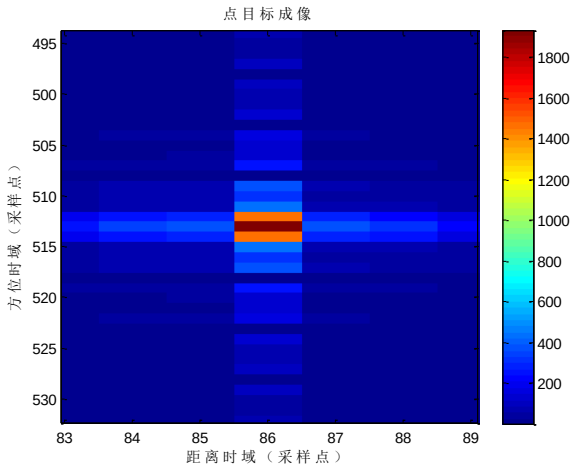
仿真条件和所有参数同 RDA 点目标仿真时相关内容，这里不再赘述。

仿真 3 个点目标，结果如下：

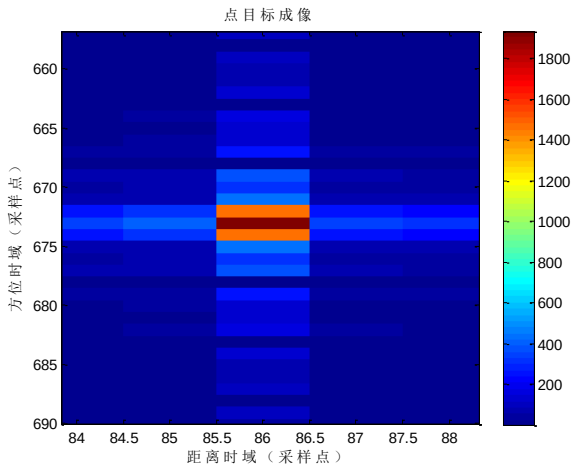
1. 单视处理结果：



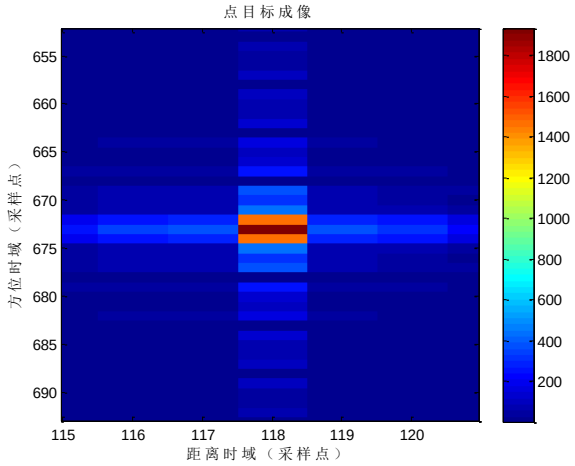
（a）总的结果



（b）点目标 A



（c）点目标 B



（d）点目标 C

图 2.1 点目标 RDA，单视处理结果

2. 多视处理结果

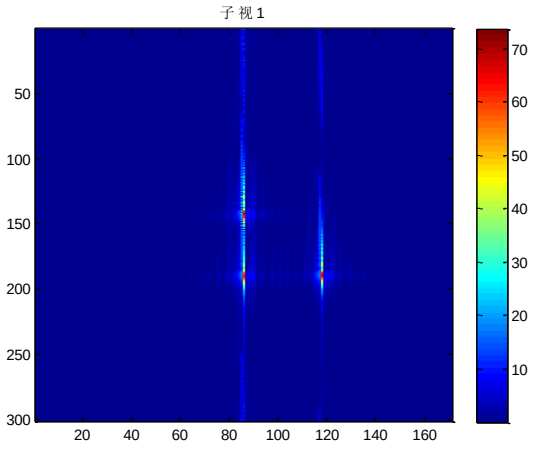


图 2.2 子视 1

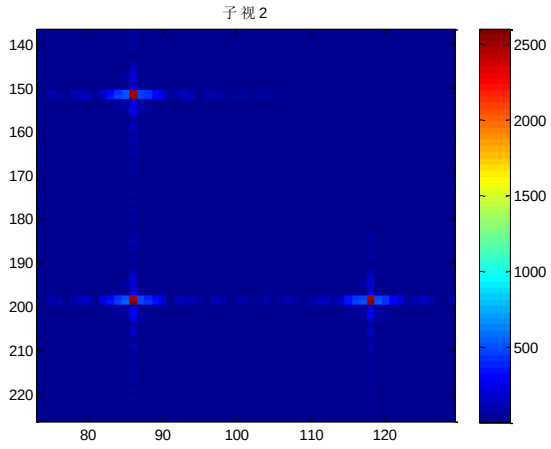


图 2.3 子视 2（局部放大）

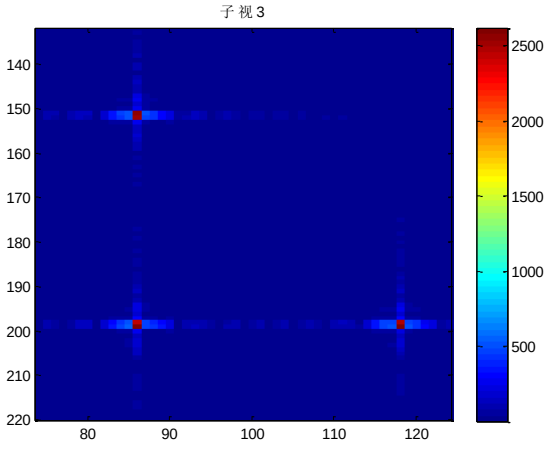


图 2.4 子视 3（局部放大）

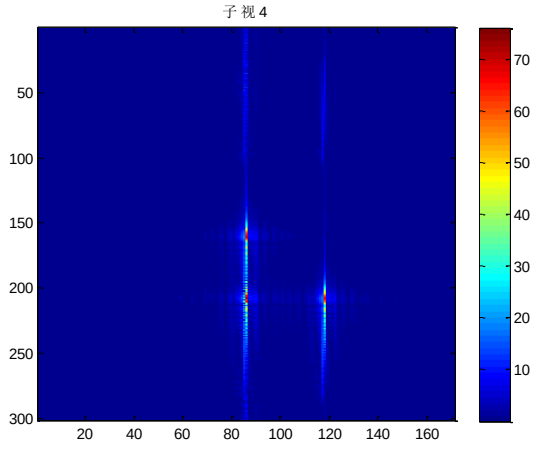


图 2.5 子视 4

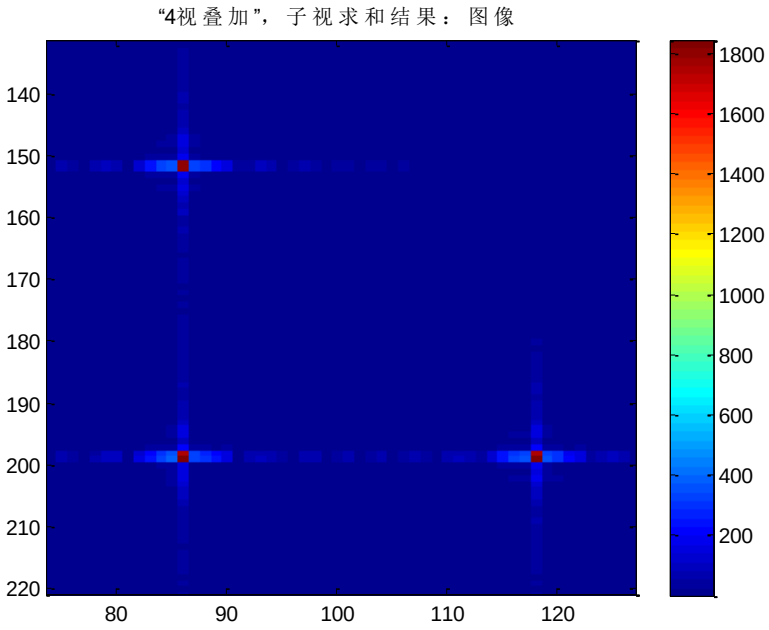
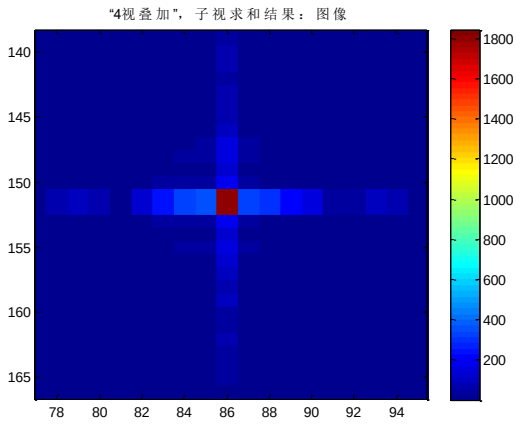
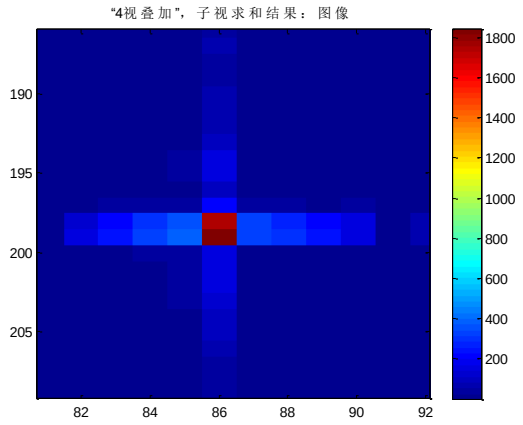


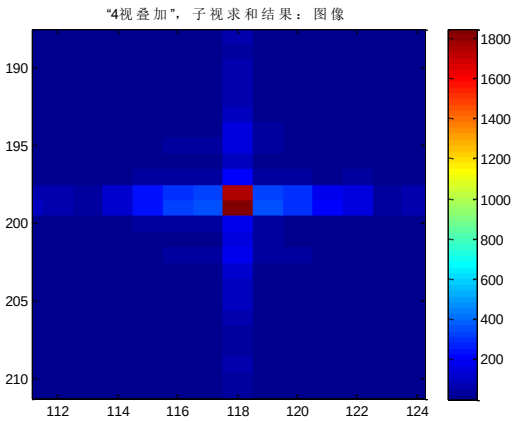
图 2.6 四视叠加结果



（a）点目标 A



（b）点目标 B



（c）点目标 C

图 2.7 四视叠加结果，三个点目标

说明：

1) 方位向长度（距离多普勒域频谱长度）共 $N_{az}=1024$，四个子视的选择为：

   子视 1： $(1:301)$

   子视 2： $(241:541)$

   子视 3： $(481:781)$

   子视 4： $(724:1024)$

   四个子视之间互相重叠长度：59 个采样点。

2) 采用 $\beta=2$ 的 Kaiser 窗对每个子视进行加权（即多普勒域截取频谱时进行加权）。

“多视叠加”这部分的程序如下：

```
%%
% ------------------------------------------------------------------
% 利用“4 视叠加”
% 抑制相干斑噪声
% ------------------------------------------------------------------
S_rd_c_fftshift = fftshift(S_rd_c,1);
w1 = kaiser(Naz/4,2);          % 当四个子视之间没有重叠部分时使用，长度为 Naz/4。
w1 = w1*ones(1,N_rg);

w2 = kaiser(301,2);            % 当四个子视之间有重叠部分时使用，长度为 301 个点。
w2 = w2*ones(1,N_rg);

% 子视 1
% S_rd_c_1 = w1.*S_rd_c_fftshift(1:Naz/4,:);        % 此时四个子视之间没有重叠部分
S_rd_c_1 = w2.*S_rd_c_fftshift(1:301,:);            % 此时四个子视，互相重叠共 59 个点
s_ac1 = ifft(S_rd_c_1);
figure;
```

```
imagesc(abs(s_ac1));
title('子视 1');

%  子视 2
% S_rd_c_2 = w1.*S_rd_c_fftshift(Naz/4+1:2*Naz/4,:);
S_rd_c_2 = w2.*S_rd_c_fftshift(241:541,:);
s_ac2 = ifft(S_rd_c_2);
figure;
imagesc(abs(s_ac2));
title('子视 2');

%  子视 3
% S_rd_c_3 = w1.*S_rd_c_fftshift(2*Naz/4+1:3*Naz/4,:);
S_rd_c_3 = w2.*S_rd_c_fftshift(481:781,:);
s_ac3 = ifft(S_rd_c_3);
figure;
imagesc(abs(s_ac3));
title('子视 3');

%  子视 4
% S_rd_c_4 = w1.*S_rd_c_fftshift(3*Naz/4+1:Naz,:);
S_rd_c_4 = w2.*S_rd_c_fftshift(724:Naz,:);
s_ac4 = ifft(S_rd_c_4);
figure;
imagesc(abs(s_ac4));
title('子视 4');

%  子视求和
s_ac_look = sqrt((abs(s_ac1).^2+abs(s_ac2).^2+abs(s_ac3).^2+abs(s_ac4).^2)./4);
```

```
% 对子视幅度平方求和再开方
figure;
imagesc(abs(s_ac_look));
title('“4 视叠加”，子视求和结果：图像');
```

三、 利用 CSA，对 Radarsat-1 的数据进行成像，仿真：

仿真说明如下：

1) 仿真程序采用对 Radarsat-1 数据成像的 CSA 仿真程序；
2) 分别进行了两种方式的多视处理：

   一种用平滑滤波（即对一个像素，在它3×3的范围内求幅度均值，以此对整个图像进行平滑处理）；

   另一种和上面点目标仿真时的完全相同，为“非相干叠加”多视处理；
3) 子视选择如下：
   a) 对于分区1和分区2，方位向长度（距离多普勒域）都是 $N_{az}=2048$，故四个子视选取为：

      子视1： $(1:569)$

      子视2： $(494:1062)$

      子视3： $(987:1555)$

      子视4： $(1480:2048)$

      各子视，重叠部分长度：76 个采样点（重叠比例：$\frac{76}{569}\approx 13.36\%$）
   b) 对于分区1和分区2合成的一整个数据块，方位向长度（距离多普勒域）是 $N_{az}=2\times 2048$，故四个子视选取为：

      子视1： $(1:1135)$

      子视2： $(988:2122)$

      子视3： $(1975:3109)$

      子视4： $(2962:4096)$

      各子视，重叠部分长度：148 个采样点（重叠比例：$\frac{148}{1135}\approx 13.04\%$）

仿真如下：

1. 分区 1

1) 单视处理结果：

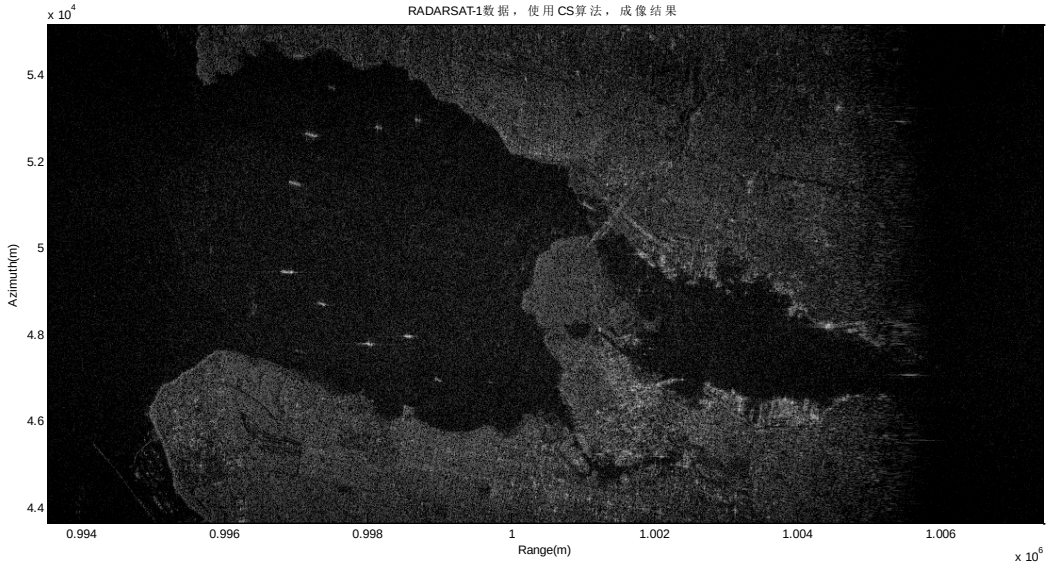


图 3.1.1 分区 1，单视处理结果

2) 选取3×3窗口，进行平滑滤波（滑动平均），以此抑制相干斑；

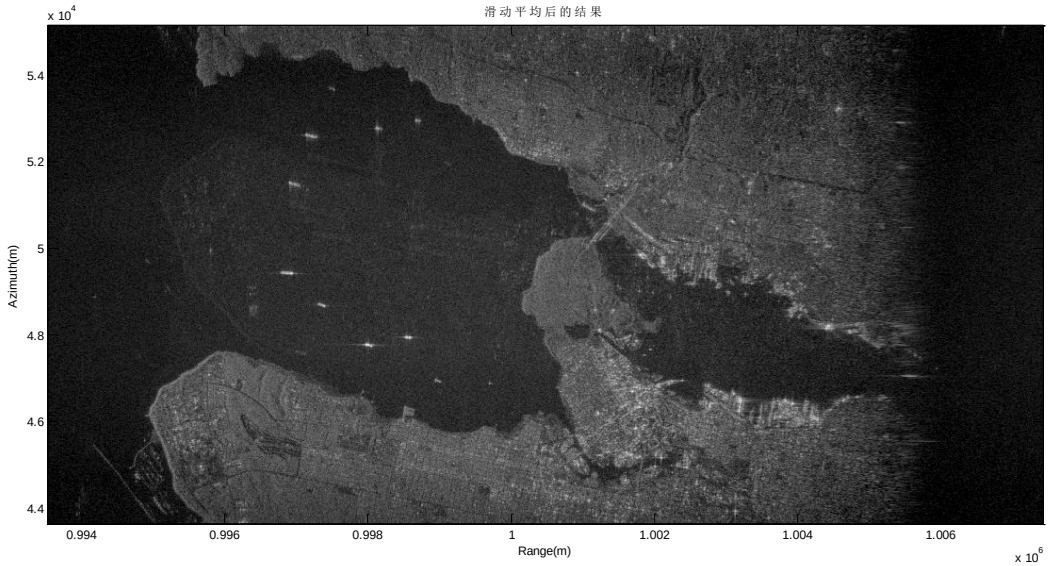


图 3.1.2 分区 1，平滑滤波处理结果

3) 进行“非相干叠加”的多视处理（四视处理），方法同RD的点目标仿真。

子视选取：

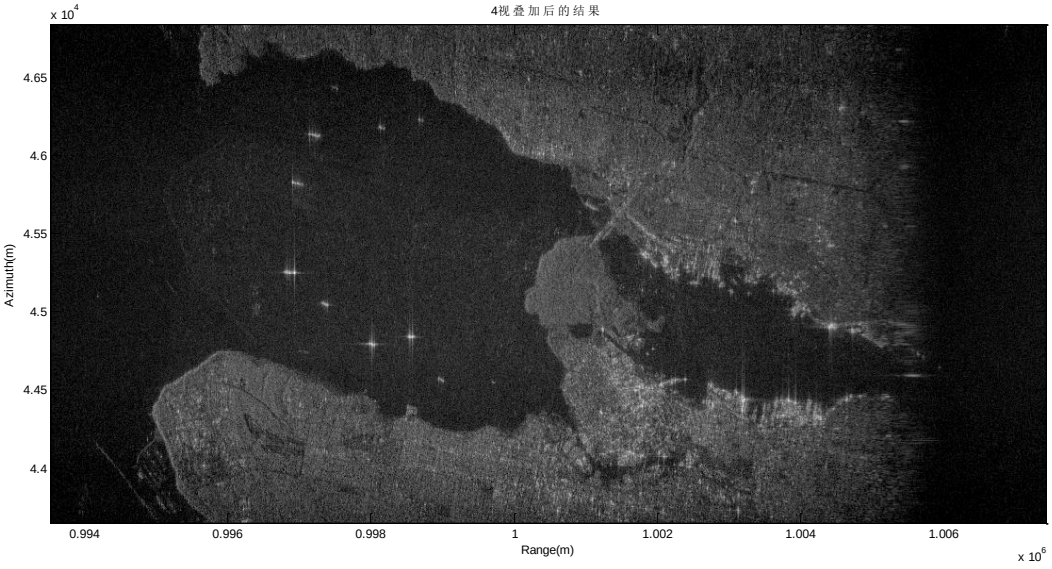
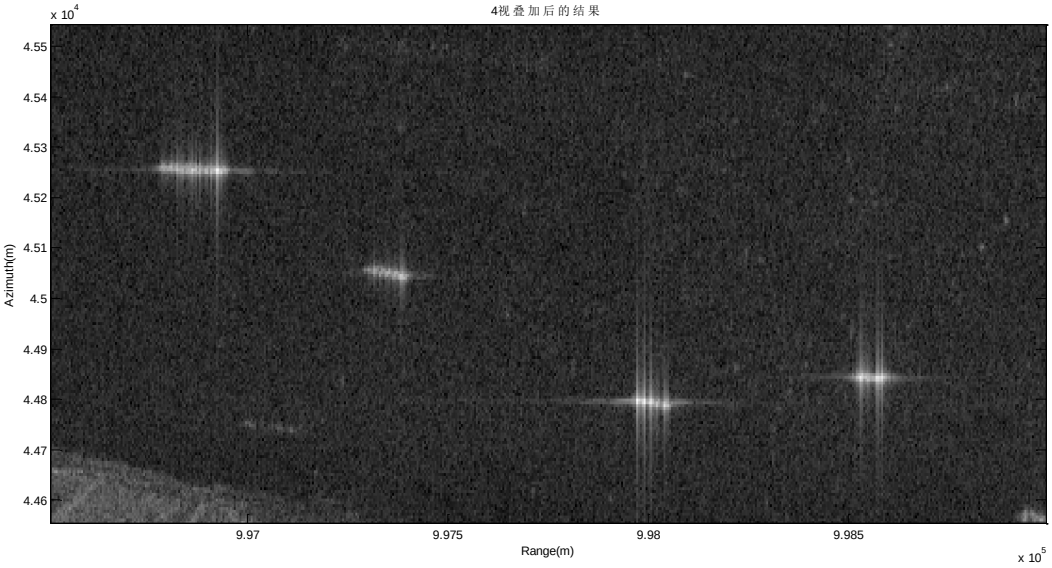


图 3.1.3 分区 1，“非相干叠加”多视处理结果

可以看到这时候的仿真结果中：原来单视结果中的特显点（比如那几艘船只），在方位向变的有很明显的旁瓣，在方位向被扩展了（我不知道该怎么形容）。如下图：



这个问题是这种方法的本质造成的，还是我在使用中哪里有问题，我目前还不知道。

——待解决

2. 分区 2

1) 单视处理结果：

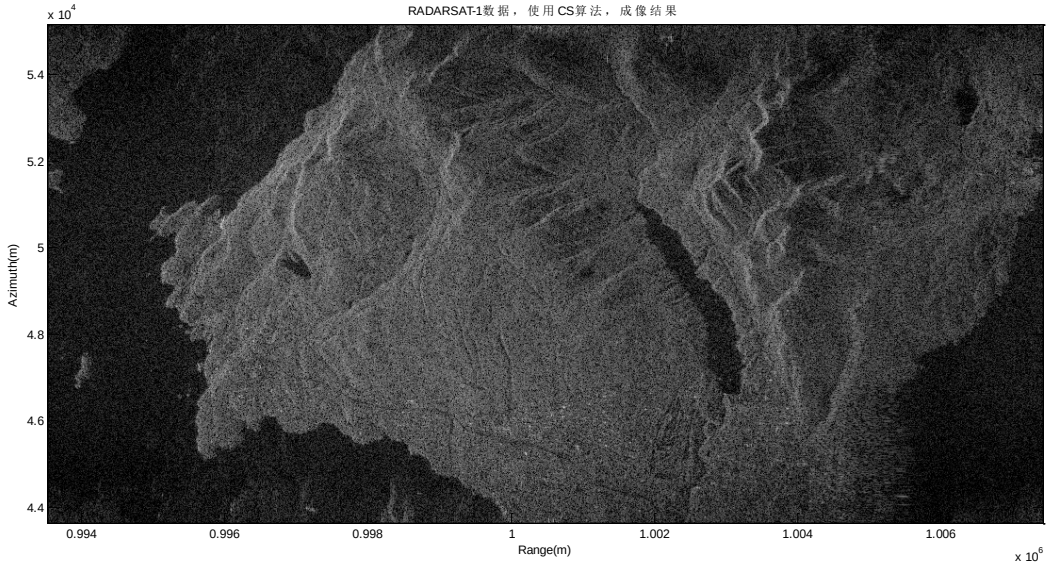


图 3.2.1 分区 2，单视处理结果

2) 选取3×3窗口，进行平滑滤波（滑动平均），以此抑制相干斑；

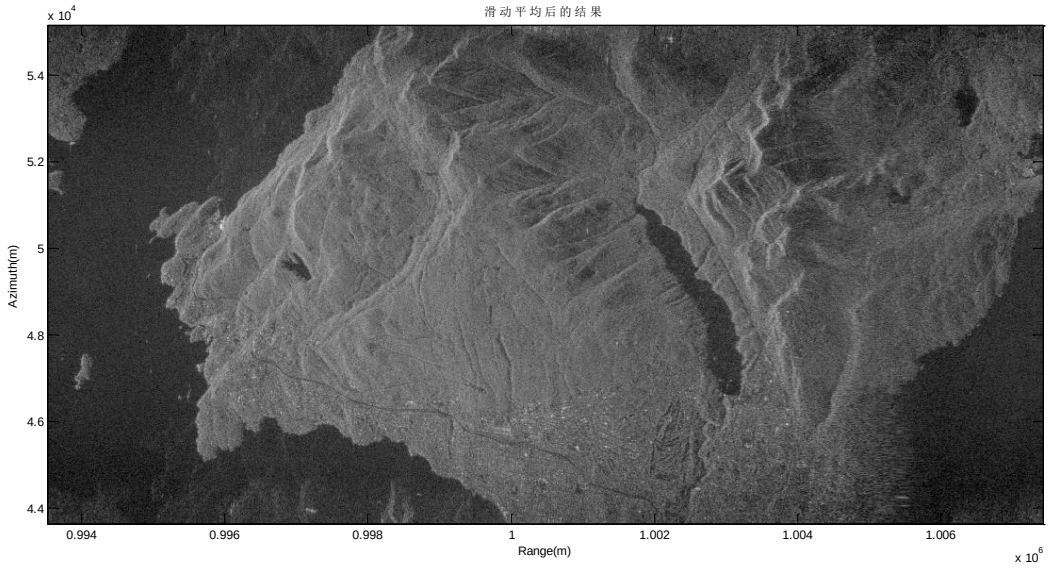


图 3.2.2 分区 2，平滑滤波处理结果

3) 进行“非相干叠加”的多视处理（四视处理），方法同RD的点目标仿真。

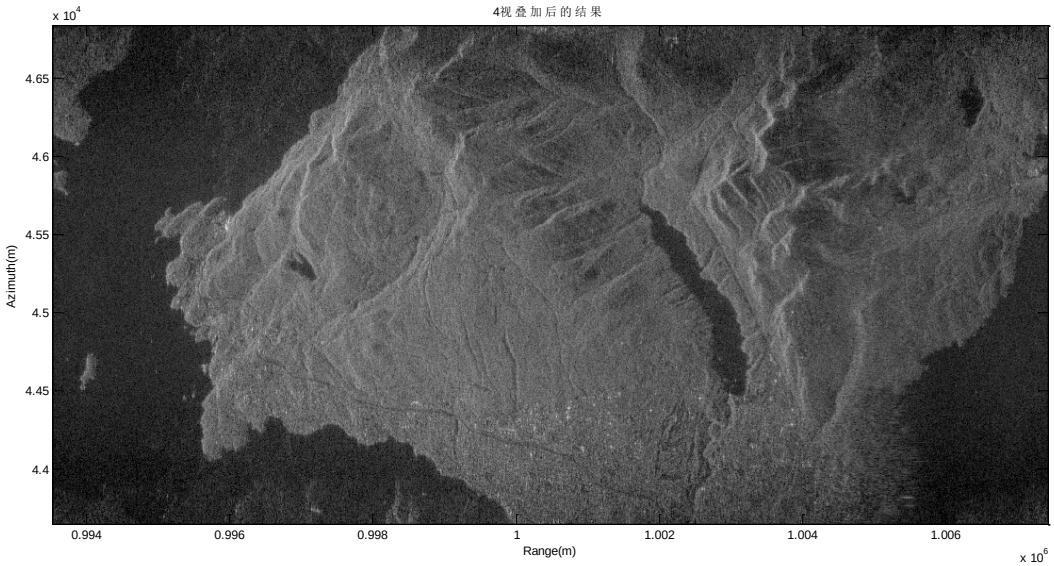


图 3.2.3 分区 2，“非相干叠加”多视处理结果

3. 分区 1 和分区 2 合成一个数据块，共同处理

1) 单视处理结果：

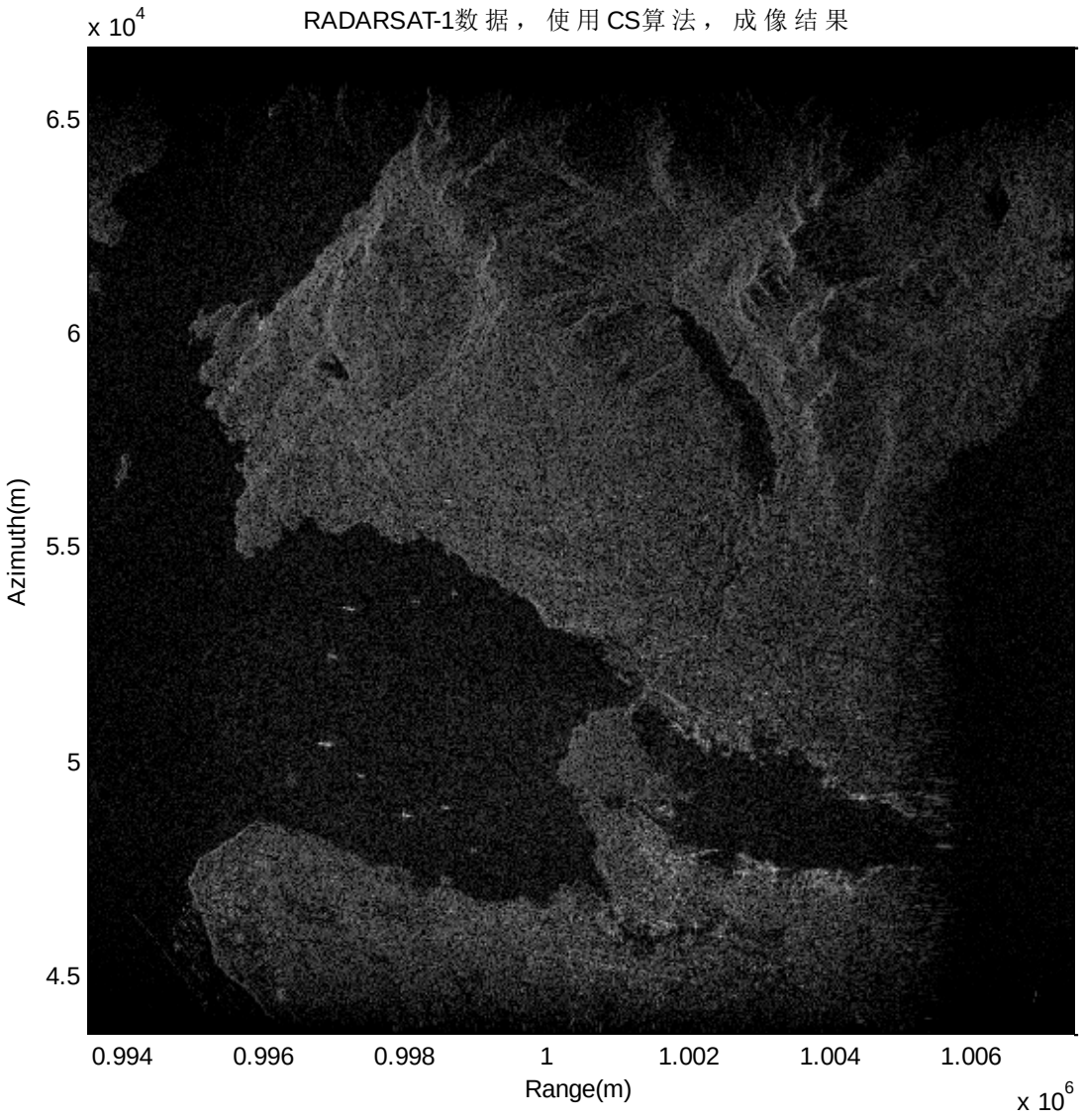


图 3.3.1 分区 3，单视处理结果

2) 选取3×3窗口，进行平滑滤波（滑动平均），以此抑制相干斑；

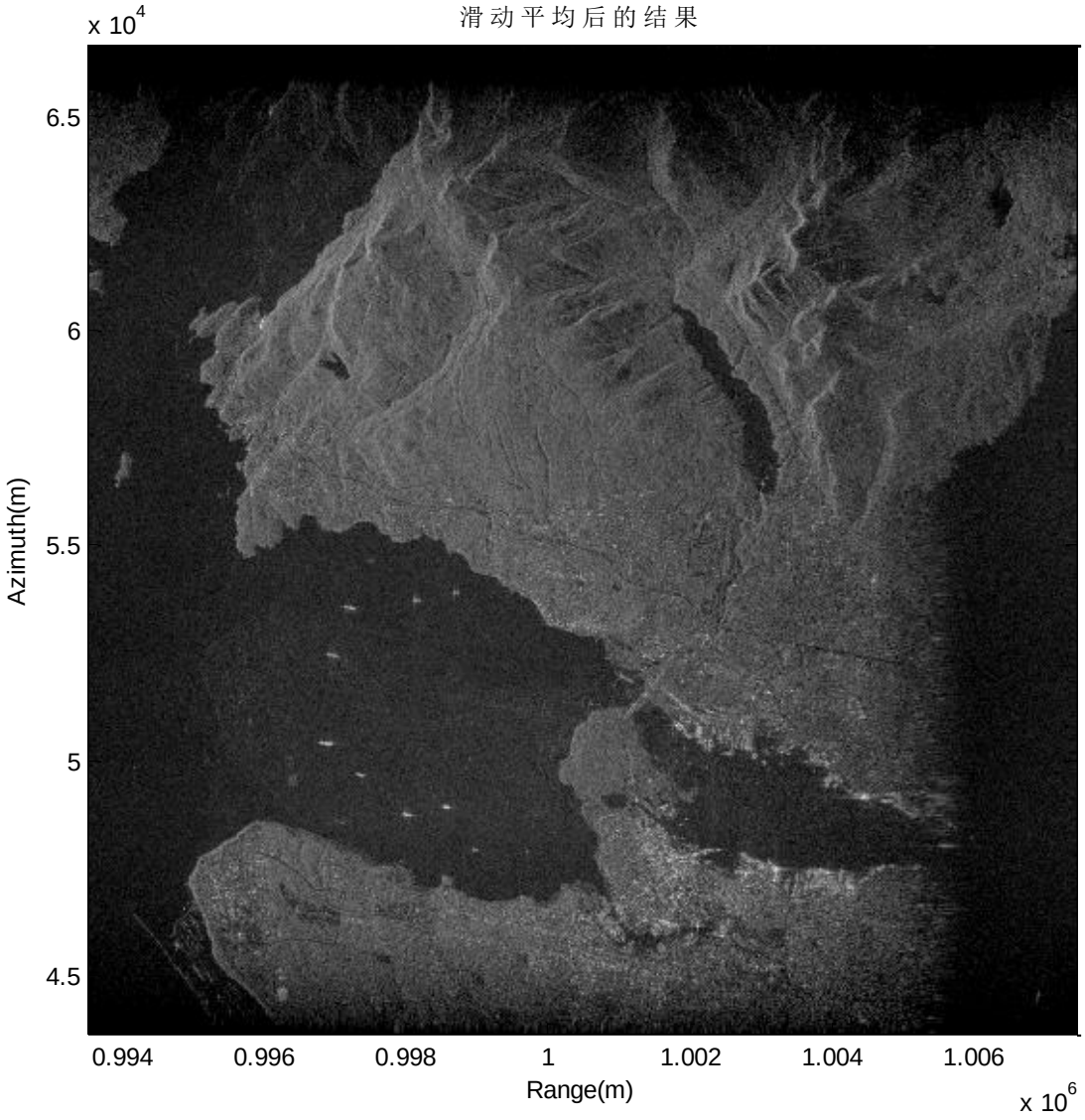


图 3.3.2 分区 3，平滑滤波处理结果

3) 进行“非相干叠加”的多视处理（四视处理），方法同RDA的点目标仿真。

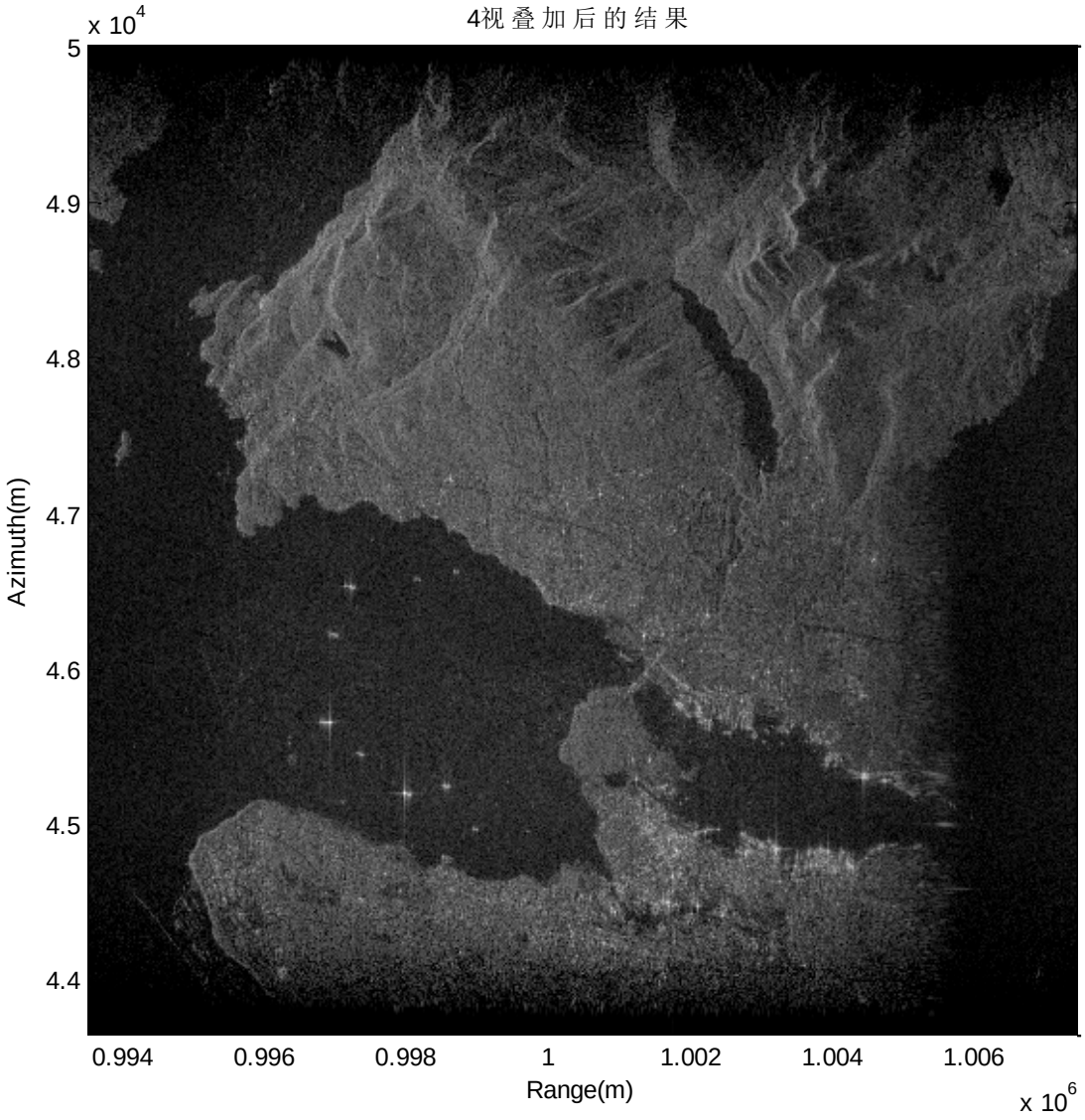
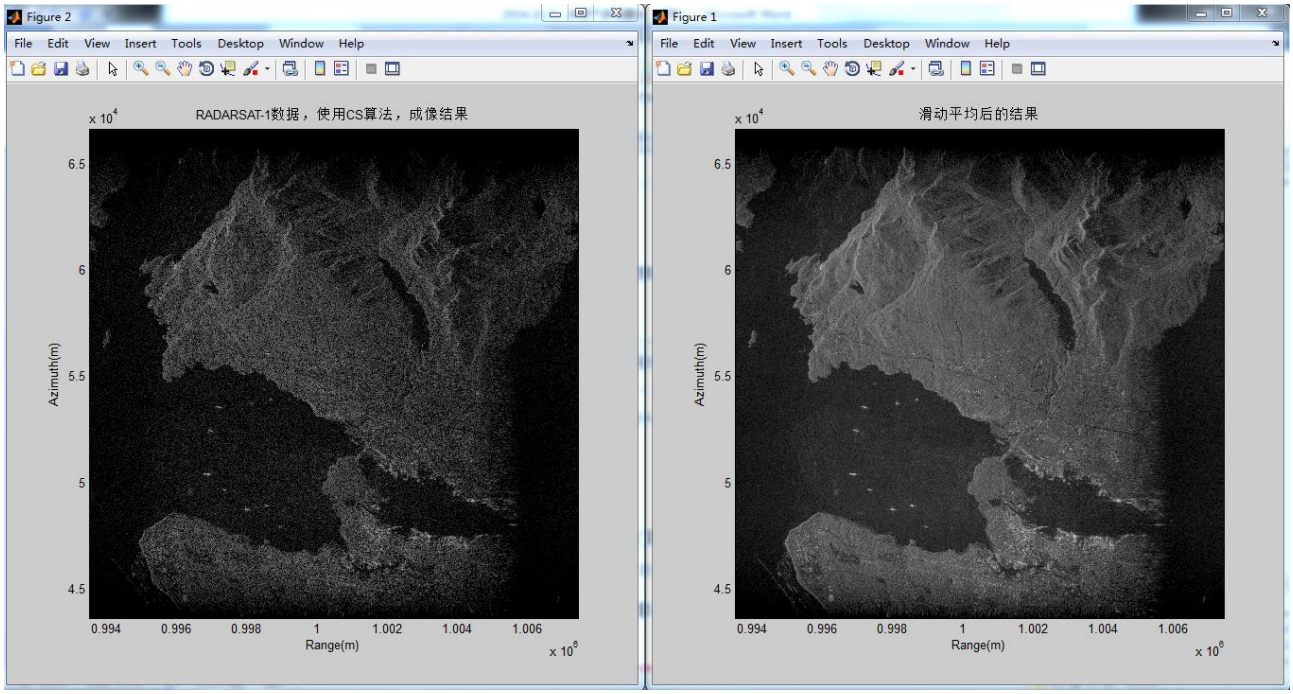


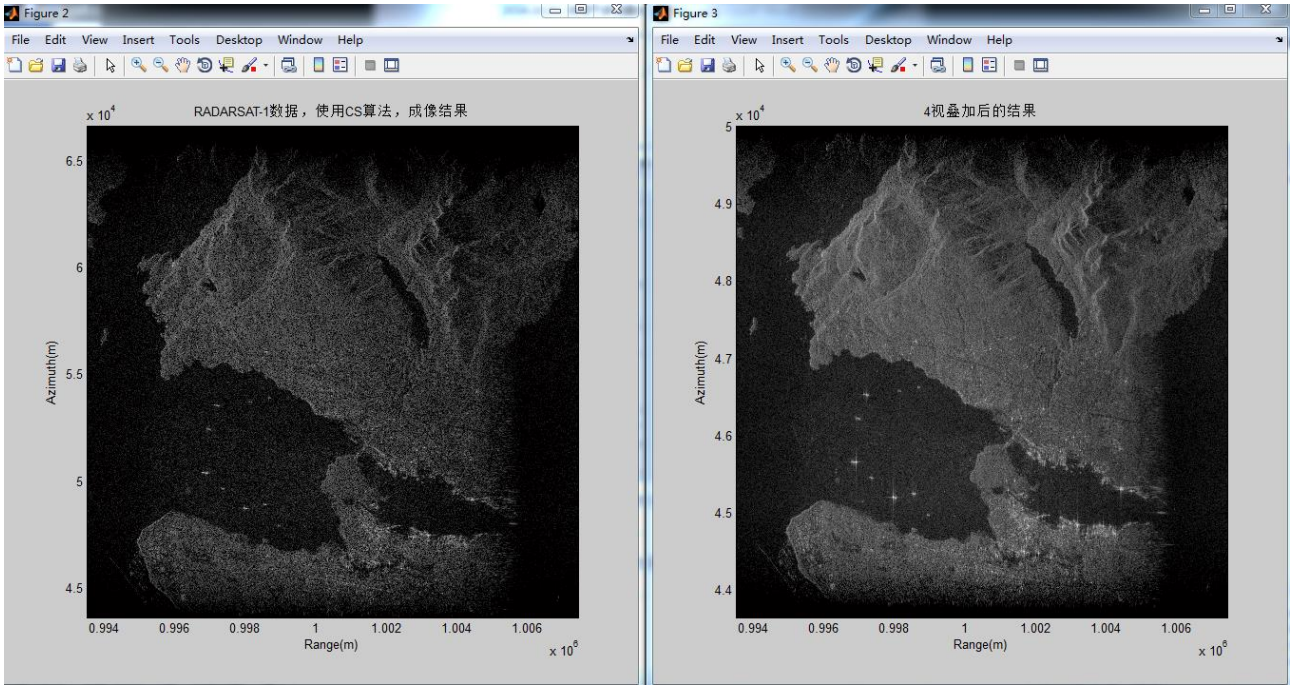
图 3.3.3 分区 3，“非相干叠加”多视处理结果

最后，将 Radarsat-1 在单视处理，滑动平均处理，和“非相干叠加”多视处理的结果，分别简单对比如下：

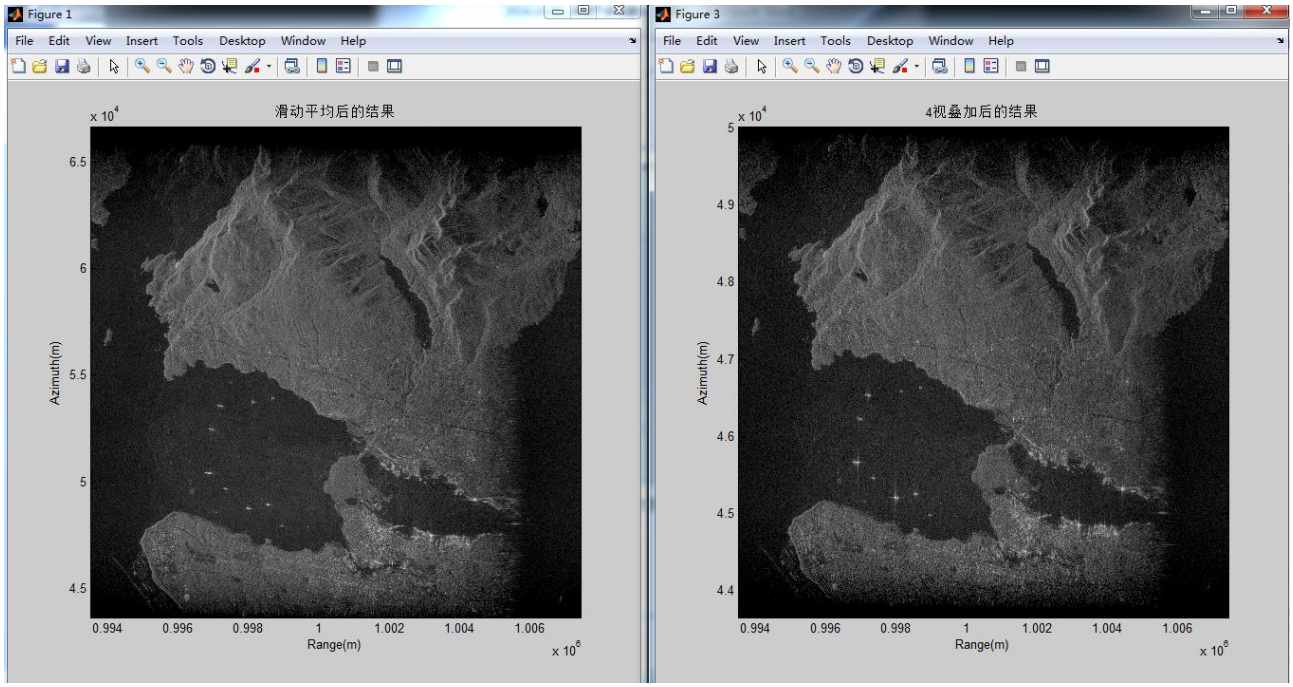
（1）单视处理，和滑动平均处理的对比



（2）单视处理，和“非相干叠加”多视处理的对比



（3）滑动平均处理，和“非相干叠加”多视处理的对比



程序源代码：

1. 选取3×3窗口，进行平滑滤波（滑动平均），程序源代码

```
%%
% ------------------------------------------------------------------
% 利用 3*3 的窗口，进行滑动平均
% 以此来抑制相干斑噪声
% ------------------------------------------------------------------
s_image_look = zeros(Naz,Nrg);              % 用来存放滑动平均后的结果
for p = 1 : Naz
    for q = 1 : Nrg
        count = 0;
        s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p,q));
        count = count + 1;
        if p>2 && p<(Naz-1) && q>2 && q<(Nrg-1)
            s_image_look(p,q) = s_image_look(p,q)+abs(s_image(p-1,q-1))+...
                abs(s_image(p-1,q))+abs(s_image(p-1,q+1))+abs(s_image(p,q-1))+...
                abs(s_image(p,q+1))+abs(s_image(p+1,q-1))+abs(s_image(p+1,q+1))+...
                abs(s_image(p+1,q+1));
            count = 9;
        else
            if (p-1) > 0
                s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p-1,q));
                count = count + 1;
                if (q-1) > 0
                    s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p-1,q-1));
                    count = count+1;
                end
                if (q+1) <= Nrg
                    s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p-1,q+1));
```

```
                count = count+1;
            end
        end

        if (p+1) <= Naz
            s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p+1,q));
            count = count + 1;
            if (q-1) > 0
                s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p+1,q-1));
                count = count+1;
            end
            if (q+1) <= Nrg
                s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p+1,q+1));
                count = count+1;
            end
        end

        if (q-1) > 0
            s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p,q-1));
            count = count+1;
        end

        if (q+1) <= Nrg
            s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p,q+1));
            count = count+1;
        end
    end
    s_image_look(p,q) = s_image_look(p,q)/count;       % 取平均
end
```

```
end
% 到此为止，我们得到了滑动平均后的结果： s_image_look
disp('利用 3*3 的窗口，进行滑动平均，得到抑制相干斑后的结果');

% 下面进行显示
clear sout;clear G;clear clim;
clear tmp;clear s_tmp;clear ss_tmp
% 抑制相干斑后的结果
% 对亮度进行非线性变换，减小对比度，显示图像
sout = abs(s_image_look)/max(max(abs(s_image_look)));
G = 20*log10(sout+eps);                    % dB显示
clim = [-55 0];                            % 动态显示范围

tmp = round(2*(R0/D_fn_ref_Vr-R0)/c*Fr);
s_tmp(:,1:Nrg-tmp+1) = G(:,tmp:end);
s_tmp(:,Nrg-tmp+1+1:Nrg) = G(:,1:tmp-1);
if b == 1 || b == 2
    % 对单一分区（比如，分区1或者分区2），使用这部分程序来显示图像
    figure;

imagesc(((0:Nrg-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz-1)+first_rg_line)/Fa*Vr,fftshift(s_tmp,1),clim
);
    axis xy;
    title('滑动平均后的结果');
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
end
```

```
if b == 3
    % 对两个分区一起成像时，使用这部分来成像。
    % 作用是：将上下部分进行一定的移位
    %          （ 原来的图像的第2900行到最后一行应该在新图像的最开头 ）
    ss_tmp(1:Naz-2900+1,:) = s_tmp(2900:Naz,:);
    ss_tmp(Naz-2900+1+1:Naz,:) = s_tmp(1:2900-1,:);
    figure;
    imagesc(((0:Nrg-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz-1)+first_rg_line)/Fa*Vr,ss_tmp,clim);
    axis xy;
    title('滑动平均后的结果');
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
end
```

2. “非相干叠加”多视处理（四视处理），程序源代码

```
%%
% --------------------------------------------------------------------
%  进行“4 视叠加”
%  抑制相干斑
% --------------------------------------------------------------------
S_rd_c_fftshift = fftshift(S_RD_3,1);

%  子视 1
if b == 1 || b == 2
    S_rd_c_1 = S_rd_c_fftshift(1:569,:);
end
if b == 3
    S_rd_c_1 = S_rd_c_fftshift(1:1135,:);
end
s_ac1 = ifft(S_rd_c_1);
% figure;
% imagesc(abs(s_ac1));
% title('子视 1');

%  子视 2
if b == 1 || b == 2
    S_rd_c_2 = S_rd_c_fftshift(494:1062,:);
end
if b == 3
    S_rd_c_2 = S_rd_c_fftshift(988:2122,:);
end
s_ac2 = ifft(S_rd_c_2);
% figure;
```

```
% imagesc(abs(s_ac2));
% title('子视 2');

%  子视 3
if b == 1 || b == 2
    S_rd_c_3 = S_rd_c_fftshift(987:1555,:);
end
if b == 3
    S_rd_c_3 = S_rd_c_fftshift(1975:3109,:);
end
s_ac3 = ifft(S_rd_c_3);
% figure;
% imagesc(abs(s_ac3));
% title('子视 3');

%  子视 4
if b == 1 || b == 2
    S_rd_c_4 = S_rd_c_fftshift(1480:2048,:);
end
if b == 3
    S_rd_c_4 = S_rd_c_fftshift(2962:4096,:);
end
s_ac4 = ifft(S_rd_c_4);
% figure;
% imagesc(abs(s_ac4));
% title('子视 4');

%  子视求和
s_ac_look = sqrt(abs(s_ac1).^2 + abs(s_ac2).^2 + abs(s_ac3).^2 + abs(s_ac4).^2);
```

```
% 对子视幅度平方求和再开方
% 到此为止，我们得到了 4 视叠加后的结果： s_ac_look
disp('进行了 4 视叠加，得到抑制相干斑后的结果');



% 下面进行显示
clear sout;clear G;clear clim;
clear tmp;clear s_tmp;clear ss_tmp
% 抑制相干斑后的结果
% 对亮度进行非线性变换，减小对比度，显示图像
sout = abs(s_ac_look)/max(max(abs(s_ac_look)));
G = 20*log10(sout+eps);                    % dB 显示
clim = [-55 0];                            % 动态显示范围

[Naz1,Nrg1] = size(s_ac_look);
tmp = round(2*(R0/D_fn_ref_Vr-R0)/c*Fr);
s_tmp(:,1:Nrg1-tmp+1) = G(:,tmp:end);
s_tmp(:,Nrg1-tmp+1+1:Nrg1) = G(:,1:tmp-1);
if b == 1 || b == 2
    % 对单一分区（比如，分区 1 或者分区 2），使用这部分程序来显示图像
    figure;

imagesc(((0:Nrg1-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz1-1)+first_rg_line)/Fa*Vr,fftshift(s_tmp,1),cl
im);
    axis xy;
    title('4 视叠加后的结果');
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
```

```
end


if b == 3
    % 对两个分区一起成像时，使用这部分来成像。
    % 作用是：将上下部分进行一定的移位
    %          （ 原来的图像的第 780 行到最后一行应该在新图像的最开头 ）
    ss_tmp(1:Naz1-780+1,:) = s_tmp(780:Naz1,:);
    ss_tmp(Naz1-780+1+1:Naz1,:) = s_tmp(1:780-1,:);
    figure;
    imagesc(((0:Nrg1-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz1-1)+first_rg_line)/Fa*Vr,ss_tmp,clim);
    axis xy;
    title('4 视叠加后的结果');
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
end
```

3. 用CSA对Radarsat-1数据成像，并加上多视处理，完整程序源代码

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                                Radarsat_1  光盘中数据
%                                     CSA  成像
%
%
%                                      WD
%                                2014.10.31. 23:40 p.m.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 程序说明：
% 主程序是：     Radarsat_1_CSA.m
%
% （1）原始数据说明：
% 文件夹中的 data_1 和 data_2 是已经经过下列方法得到的原始数据，
% 可以直接进行后续成像
% ------------------------------------------------------
% 使用现成的程序‘compute.azim.spectra.m’中读出数据的方法；
% 利用函数 'laod_DATA_block.m'，实现
%                       - reads /loads data for a block
%                       - converts to floating point
%                       - compansates for the receiver attenuation
% 变量 b -- 需要设置数据取自哪个分区
%                       - b = 1 , from CDdata1
%                       - b = 2 , from CDdata2
% 得到所需要的数据，也即可以直接进行后续 processing 的数据 data。
% ------------------------------------------------------
% 因此，文件夹中的 data_1和data_2 分别是分区1和分区2的数据，经过了下变频，
%          转换为了浮点数，进行了AGC增益补偿，最后转换为了double双精度浮点数。
```

```
%          因此，直接载入这两个数据就可以进行后续成像。
% 注：
%    这里还附加了一步：对原始数据进行补零，再进行后续处理。（补零是非常必要的!!）
%
% （2） 本文件夹中还有一个文件：CD_run_params
%              ——这里面是仿真中需要用的许多参数，直接载入即可。
%
% （3）成像程序说明：
%          由CSA的点目标程序修改而来；
%
% （4）成像流程：
%    ——原始数据
%    ——经过方位向FFT，变换到距离多普勒域，进行“补余RCMC”
%    ——经过距离向FFT，变换到二维频域，进行“距离压缩”、“SRC”、“一致RCMC”
%    ——经过距离向IFFT，变换到距离多普勒域，进行“方位压缩”和“附加相位校正”
%    ——经过方位向IFFT，回到图像域，成像结束。
%
% （5）分别采用了两种方式抑制相干斑：
%          a）“利用3*3的窗口，进行滑动平均，以抑制相干斑”;
%          b）进行“4视叠加”，来抑制相干斑；
%
% 本程序修改截止到： 2014.10.31. 23:40 p.m.


%%
clear;
clc;
close all;
% ------------------------------------------------------
% 得到可以进行后续信号处理的原始数据data（s_echo）
```

```
% ------------------------------------------------------
% 载入参数
load CD_run_params;

% 载入数据
b = 1;                  % 选择对于哪一部分成像
% b = 1，则对分区1成像
% b = 2，则对分区2成像
% b = 3，则对整个数据（分区1和分区2）成像

if b == 1
    load data_1;                % 分区1的数据
    s_echo = data_1;            % 原始数据记为s_echo，用于后续成像。
end
clear data_1;                   % 清除data_1，以腾出内存

if b == 2
    load data_2;                % 分区2的数据
    s_echo = data_2;            % 原始数据记为s_echo，用于后续成像。
end
clear data_2;                   % 清除data_2，以腾出内存

if b == 3
    load data_1;                % 分区1的数据
    s_echo1 = data_1;
    load data_2;                % 分区2的数据
    s_echo2 = data_2;
    s_echo = [s_echo1;s_echo2]; % 将分区1和分区2的数据合成整个数据块，用于成像
end
```

```
clear data_1;clear data_2;clear s_echo1;clear s_echo2;


%{
%  作图显示
figure;
imagesc(abs(s_echo));
title('原始数据');                    %  原始回波数据（未处理）的幅度图像
% colormap(gray);
%}


%%
% ------------------------------------------------------------------
%  定义一些参数
% ------------------------------------------------------------------
Kr = -Kr;                             %  将调频率Kr改成负值
BW_range = 30.111e+06;                %  脉冲宽度
Vr = 7062;                            %  有效雷达速率
Ka = 1733;                            %  方位调频率
fnc = -6900;                          %  多普勒中心频率
Fa = PRF;                             %  方位向采样率
lamda = c/f0;                         %  波长
T_start = 6.5959e-03;                 %  数据窗开始时间

Nr = round(Tr*Fr);                    %  线性调频信号采样点数
Nrg = Nrg_cells;                      %  距离线采样点数
if b == 1 || b == 2
    Naz = Nrg_lines_blk;              %  每一个数据块的距离线数
else
    Naz = Nrg_lines;                  %  两个数据块，总共的距离线数
```

```
end
NFFT_r = Nrg;                          % 距离向FFT长度
NFFT_a = Naz;                          % 方位向FFT长度

R_ref = R0;                            % 参考目标选在场景中心，其最近斜距为 R_ref
fn_ref = fnc;                          % 参考目标的多普勒中心频率

%%
%
% ----------------------------------------------------------------
% 对原始数据进行补零
% ----------------------------------------------------------------
if b == 1 || b == 2
    data = zeros(1*2048,3000);
else
    data = zeros(2*2048,3000);
end
data(1:Naz,1:Nrg) = s_echo;
clear s_echo;
s_echo = data;
clear data;
[Naz,Nrg] = size(s_echo);

NFFT_r = Nrg;                          % 距离向FFT长度
NFFT_a = Naz;                          % 方位向FFT长度

% 作图显示
figure;
imagesc(abs(s_echo));
```

```
title('补零后的原始数据');                % 补零后的原始回波数据（未处理）的幅度图像
%}

%%
% --------------------------------------------------------------------
% 距离（方位）向时间，频率相关定义
% --------------------------------------------------------------------
% 距离
tr = 2*R0/c + ( -Nrg/2 : (Nrg/2-1) )/Fr;                    % 距离时间轴
fr = ( -NFFT_r/2 : NFFT_r/2-1 )*( Fr/NFFT_r );              % 距离频率轴
% 方位
ta = ( -Naz/2: Naz/2-1 )/Fa;                                % 方位时间轴
fa = fnc + fftshift( -NFFT_a/2 : NFFT_a/2-1 )*( Fa/NFFT_a );  % 方位频率轴

% 生成距离（方位）时间（频率）矩阵
tr_mtx = ones(Naz,1)*tr;      % 距离时间轴矩阵，大小：Naz*Nrg
ta_mtx = ta.'*ones(1,Nrg);    % 方位时间轴矩阵，大小：Naz*Nrg
fr_mtx = ones(Naz,1)*fr;      % 距离频率轴矩阵，大小：Naz*Nrg
fa_mtx = fa.'*ones(1,Nrg);    % 方位频率轴矩阵，大小：Naz*Nrg

%%
% --------------------------------------------------------------------
% 变换到距离多普勒域，进行“补余RCMC”
% --------------------------------------------------------------------
s_rd = s_echo.*exp(-1j*2*pi*fnc.*(ta.'*ones(1,Nrg)));       % 数据搬移
S_RD = fft(s_rd,NFFT_a,1);    % 进行方位向傅里叶变换，得到距离多普勒域频谱

D_fn_Vr = sqrt(1-lamda^2.*(fa.').^2./(4*Vr^2));             % 大斜视角下的徙动因子，列向量
D_fn_Vr_mtx = D_fn_Vr*ones(1,Nrg);    % 形成矩阵，大小：Nrg*Naz
```

```
D_fn_ref_Vr = sqrt(1-lamda^2*fn_ref^2/(4*Vr^2));     % 参考频率fn_ref处的徙动因子，是常数。

K_src = 2*Vr^2*f0^3.*D_fn_Vr.^3./(c*R_ref*(fa.').^2);       % 列向量，使用R_ref处的值
K_src_mtx = K_src*ones(1,Nrg);        % 形成矩阵
Km = Kr./(1-Kr./K_src_mtx);           % 矩阵，这是变换到距离多普勒域的距离调频率。
                                      % 使用 R_ref 处的值

% 下面生成 变标方程 s_sc
s_sc = exp(1j*pi.*Km.*(D_fn_ref_Vr./D_fn_Vr_mtx-1).*(tr_mtx-2*R_ref./(c.*D_fn_Vr_mtx)).^2);

% 下面将距离多普勒域的信号与变标方程相乘，实现“补余RCMC”
S_RD_1 = S_RD.*s_sc;                  % 相位相乘，实现“补余RCMC”

disp(' 距离多普勒域，完成“补余RCMC” ');
%{
% 作图
figure;
imagesc(abs(S_RD));
title('原始数据变换到距离多普勒域，幅度');
figure;
imagesc(abs(S_RD_1));
title('距离多普勒域，补余RCMC后，幅度');
%}
clear S_RD;



%%
% ------------------------------------------------------------------
```

```
% 变换到二维频域，进行“距离压缩，SRC，一致RCMC”
% ----------------------------------------------------------------
S_2df_1 = fft(S_RD_1,NFFT_r,2);     % 进行距离向FFT，变换到二维频域。距离零频在两端

% 完成距离压缩，SRC，一致RCMC这三者相位补偿的滤波器为：
H1 = exp(1j*pi.*D_fn_Vr_mtx./(D_fn_ref_Vr.*Km).*fr_mtx.^2)...
      .*exp(1j*4*pi/c.*(1./D_fn_Vr_mtx-1/D_fn_ref_Vr).*R_ref.*fr_mtx);
% 上面的H1距离零频在中心
W_ref = ones(Naz,1)*(kaiser(Nrg,3).');
% 距离向，构建Kaiser窗，此为矩阵形式，距离零频在中心。
% H1 = W_ref.*H1;                   % 加入距离平滑窗，以抑制旁瓣，距离零频在中心。
% 下面通过fftshift将H1的距离零频调整到两端
H1 = fftshift(H1,2);                % 左右半边互换，距离零频在两端。

S_2df_2 = S_2df_1.*H1;       % 在二维频域，相位相乘，实现距离压缩，SRC，一致RCMC

S_RD_2 = ifft(S_2df_2,NFFT_r,2);
% 进行距离IFFT，回到距离多普勒域，完成所有距离处理。

disp('  在二维频域进行相位相乘，完成距离压缩，SRC，一致RCMC后，回到距离多普勒域  ');
%{
% 作图
figure;
imagesc(abs(S_2df_1));
title('变换到二维频域');
figure;
imagesc(abs(S_2df_2));
title('相位相乘，实现距离压缩，SRC，一致RCMC后，二维频域');
%
```

```
figure;
imagesc(abs(S_RD_2));
title('完成距离压缩，SRC，一致RCMC后，距离多普勒域');
%}
clear S_RD_1;
clear S_2df_1;
clear H1;
clear S_2df_2;

%%
% ----------------------------------------------------------------
% 距离多普勒域，完成“方位压缩”和“附加相位校正”
% ----------------------------------------------------------------
R0_RCMC = (c/2).*tr;    % 随距离线变化的R0，记为R0_RCMC，用来计算方位MF。

% 生成方位向匹配滤波器
Haz = exp(1j*4*pi.*(D_fn_Vr*R0_RCMC).*f0./c);                % 方位MF

% 附加相位校正项
H2 = exp(-1j*4*pi.*Km./(c^2).*(1-D_fn_Vr_mtx./D_fn_ref_Vr)...
      .*((1./D_fn_Vr)*R0_RCMC-R_ref./D_fn_Vr_mtx).^2);      % 附加相位校正项

% 下面进行相位相乘，在距离多普勒域，同时完成方位MF和附加相位校正
S_RD_3 = S_RD_2.*Haz.*H2;                % 距离多普勒域，相位相乘

% 最后通过IFFT回到图像域，完成方未处理
s_image = ifft(S_RD_3,NFFT_a,1);          % 完成成像过程，得到成像结果为：s_image

disp(' 完成“方位压缩”和“附加相位校正” ');
```

```
disp('  成像结束  ');
%{
%  作图
figure;
imagesc(abs(S_RD_3));
title('距离多普勒域，进行了相位相乘后（方位MF和附加相位校正）');
%}
clear S_RD_2;
clear Haz;
clear H2;
% clear S_RD_3;        %  后面进行多视叠加需要距离多普勒域频谱，因此这里不要清除。

%%
%  下面对亮度进行非线性变换，减小对比度
sout = abs(s_image)/max(max(abs(s_image)));
G = 20*log10(sout+eps);                    % dB显示
clim = [-55 0];                            %  动态显示范围
%{
figure;
imagesc(((0:Nrg-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz-1)+first_rg_line)/Fa*Vr,G,clim);
axis xy;
title('RADARSAT-1数据，使用CS算法，成像结果')
xlabel('Range(m)')
ylabel('Azimuth(m)')
% colormap(gray);
%}

%  将图像向左移位：
%     基于CSA算法的成像位置是压至参考频率对应的距离单元，而非压至零多普勒处
```

```
%    得到的图像结果相比于压至零多普勒，是向右偏移的
% 因此进行以下向左移位
% 此外，还要进行上下半边互换
% 经过以上操作后，得到结果：
tmp = round(2*(R0/D_fn_ref_Vr-R0)/c*Fr);
s_tmp(:,1:Nrg-tmp+1) = G(:,tmp:end);
s_tmp(:,Nrg-tmp+1+1:Nrg) = G(:,1:tmp-1);

if b == 1 || b == 2
    % 对单一分区（比如，分区1或者分区2），使用这部分程序来显示图像
    figure;

imagesc(((0:Nrg-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz-1)+first_rg_line)/Fa*Vr,fftshift(s_tmp,1),clim
);
    axis xy;
    title('RADARSAT-1数据，使用CS算法，成像结果')
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
end

if b == 3
    % 对两个分区一起成像时，使用这部分来显示图像。
    % 作用是：将上下部分进行一定的移位
    %          （ 原来的图像的第2900行到最后一行应该在新图像的最开头 ）
    ss_tmp(1:Naz-2900+1,:) = s_tmp(2900:Naz,:);
    ss_tmp(Naz-2900+1+1:Naz,:) = s_tmp(1:2900-1,:);
    figure;
    imagesc(((0:Nrg-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz-1)+first_rg_line)/Fa*Vr,ss_tmp,clim);
```

```
    axis xy;
    title('RADARSAT-1数据，使用CS算法，成像结果')
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
end



%%
% ------------------------------------------------------------------
% 利用 3*3 的窗口，进行滑动平均
% 以此来抑制相干斑噪声
% ------------------------------------------------------------------
s_image_look = zeros(Naz,Nrg);          % 用来存放滑动平均后的结果
for p = 1 : Naz
    for q = 1 : Nrg
        count = 0;
        s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p,q));
        count = count + 1;
        if p>2 && p<(Naz-1) && q>2 && q<(Nrg-1)
            s_image_look(p,q) = s_image_look(p,q)+abs(s_image(p-1,q-1))+...
                abs(s_image(p-1,q))+abs(s_image(p-1,q+1))+abs(s_image(p,q-1))+...
                abs(s_image(p,q+1))+abs(s_image(p+1,q-1))+abs(s_image(p+1,q+1))+...
                abs(s_image(p+1,q+1));
            count = 9;
        else
            if (p-1) > 0
                s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p-1,q));
                count = count + 1;
```

```
        if (q-1) > 0
            s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p-1,q-1));
            count = count+1;
        end
        if (q+1) <= Nrg
            s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p-1,q+1));
            count = count+1;
        end
    end

    if (p+1) <= Naz
        s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p+1,q));
        count = count + 1;
        if (q-1) > 0
            s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p+1,q-1));
            count = count+1;
        end
        if (q+1) <= Nrg
            s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p+1,q+1));
            count = count+1;
        end
    end

    if (q-1) > 0
        s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p,q-1));
        count = count+1;
    end

    if (q+1) <= Nrg
```

```
                    s_image_look(p,q) = s_image_look(p,q) + abs(s_image(p,q+1));
                    count = count+1;
                end
            end
            s_image_look(p,q) = s_image_look(p,q)/count;       % 取平均
        end
    end
% 到此为止，我们得到了滑动平均后的结果： s_image_look
disp('利用 3*3 的窗口，进行滑动平均，得到抑制相干斑后的结果');

% 下面进行显示
clear sout;clear G;clear clim;
clear tmp;clear s_tmp;clear ss_tmp
% 抑制相干斑后的结果
% 对亮度进行非线性变换，减小对比度，显示图像
sout = abs(s_image_look)/max(max(abs(s_image_look)));
G = 20*log10(sout+eps);                    % dB显示
clim = [-55 0];                            % 动态显示范围

tmp = round(2*(R0/D_fn_ref_Vr-R0)/c*Fr);
s_tmp(:,1:Nrg-tmp+1) = G(:,tmp:end);
s_tmp(:,Nrg-tmp+1+1:Nrg) = G(:,1:tmp-1);
if b == 1 || b == 2
    % 对单一分区（比如，分区1或者分区2），使用这部分程序来显示图像
    figure;

imagesc(((0:Nrg-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz-1)+first_rg_line)/Fa*Vr,fftshift(s_tmp,1),clim
);
    axis xy;
```

```
    title('滑动平均后的结果');
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
end

if b == 3
    % 对两个分区一起成像时，使用这部分来成像。
    % 作用是：将上下部分进行一定的移位
    %          （ 原来的图像的第2900行到最后一行应该在新图像的最开头 ）
    ss_tmp(1:Naz-2900+1,:) = s_tmp(2900:Naz,:);
    ss_tmp(Naz-2900+1+1:Naz,:) = s_tmp(1:2900-1,:);
    figure;
    imagesc(((0:Nrg-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz-1)+first_rg_line)/Fa*Vr,ss_tmp,clim);
    axis xy;
    title('滑动平均后的结果');
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
end


%%
% ------------------------------------------------------------------
% 进行“4视叠加”
% 抑制相干斑
% ------------------------------------------------------------------
S_rd_c_fftshift = fftshift(S_RD_3,1);
```

```
%  子视1
if b == 1 || b == 2
    S_rd_c_1 = S_rd_c_fftshift(1:569,:);
end
if b == 3
    S_rd_c_1 = S_rd_c_fftshift(1:1135,:);
end
s_ac1 = ifft(S_rd_c_1);
% figure;
% imagesc(abs(s_ac1));
% title('子视1');

%  子视2
if b == 1 || b == 2
    S_rd_c_2 = S_rd_c_fftshift(494:1062,:);
end
if b == 3
    S_rd_c_2 = S_rd_c_fftshift(988:2122,:);
end
s_ac2 = ifft(S_rd_c_2);
% figure;
% imagesc(abs(s_ac2));
% title('子视2');

%  子视3
if b == 1 || b == 2
    S_rd_c_3 = S_rd_c_fftshift(987:1555,:);
end
if b == 3
```

```
    S_rd_c_3 = S_rd_c_fftshift(1975:3109,:);
end
s_ac3 = ifft(S_rd_c_3);
% figure;
% imagesc(abs(s_ac3));
% title('子视3');

%  子视4
if b == 1 || b == 2
    S_rd_c_4 = S_rd_c_fftshift(1480:2048,:);
end
if b == 3
    S_rd_c_4 = S_rd_c_fftshift(2962:4096,:);
end
s_ac4 = ifft(S_rd_c_4);
% figure;
% imagesc(abs(s_ac4));
% title('子视4');

%  子视求和
s_ac_look = sqrt(abs(s_ac1).^2 + abs(s_ac2).^2 + abs(s_ac3).^2 + abs(s_ac4).^2);
%  对子视幅度平方求和再开方
%  到此为止，我们得到了4视叠加后的结果：  s_ac_look
disp('进行了4视叠加，得到抑制相干斑后的结果');


%  下面进行显示
clear sout;clear G;clear clim;
clear tmp;clear s_tmp;clear ss_tmp
```

```
% 抑制相干斑后的结果
% 对亮度进行非线性变换，减小对比度，显示图像
sout = abs(s_ac_look)/max(max(abs(s_ac_look)));
G = 20*log10(sout+eps);                    % dB显示
clim = [-55 0];                            % 动态显示范围

[Naz1,Nrg1] = size(s_ac_look);
tmp = round(2*(R0/D_fn_ref_Vr-R0)/c*Fr);
s_tmp(:,1:Nrg1-tmp+1) = G(:,tmp:end);
s_tmp(:,Nrg1-tmp+1+1:Nrg1) = G(:,1:tmp-1);
if b == 1 || b == 2
    % 对单一分区（比如，分区1或者分区2），使用这部分程序来显示图像
    figure;

imagesc(((0:Nrg1-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz1-1)+first_rg_line)/Fa*Vr,fftshift(s_tmp,1),cl
im);
    axis xy;
    title('4视叠加后的结果');
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
end

if b == 3
    % 对两个分区一起成像时，使用这部分来成像。
    % 作用是：将上下部分进行一定的移位
    %          （ 原来的图像的第780行到最后一行应该在新图像的最开头 ）
    ss_tmp(1:Naz1-780+1,:) = s_tmp(780:Naz1,:);
    ss_tmp(Naz1-780+1+1:Naz1,:) = s_tmp(1:780-1,:);
```

```
    figure;
    imagesc(((0:Nrg1-1)+first_rg_cell)/Fr*c/2+R0,((0:Naz1-1)+first_rg_line)/Fa*Vr,ss_tmp,clim);
    axis xy;
    title('4视叠加后的结果');
    xlabel('Range(m)')
    ylabel('Azimuth(m)')
    colormap(gray)
end
```

参考文献

[1] Curlander John C., McDonough Robert N. 合成孔径雷达——系统与信号处理[M]. 韩传钊等, 译. 电子工业出版社, 2014.

[2] Cumming Ian G., Wong Frank H. 合成孔径雷达成像——算法与实现[M]. 洪文, 胡东辉等, 译. 电子工业出版社, 2007.