章节 7

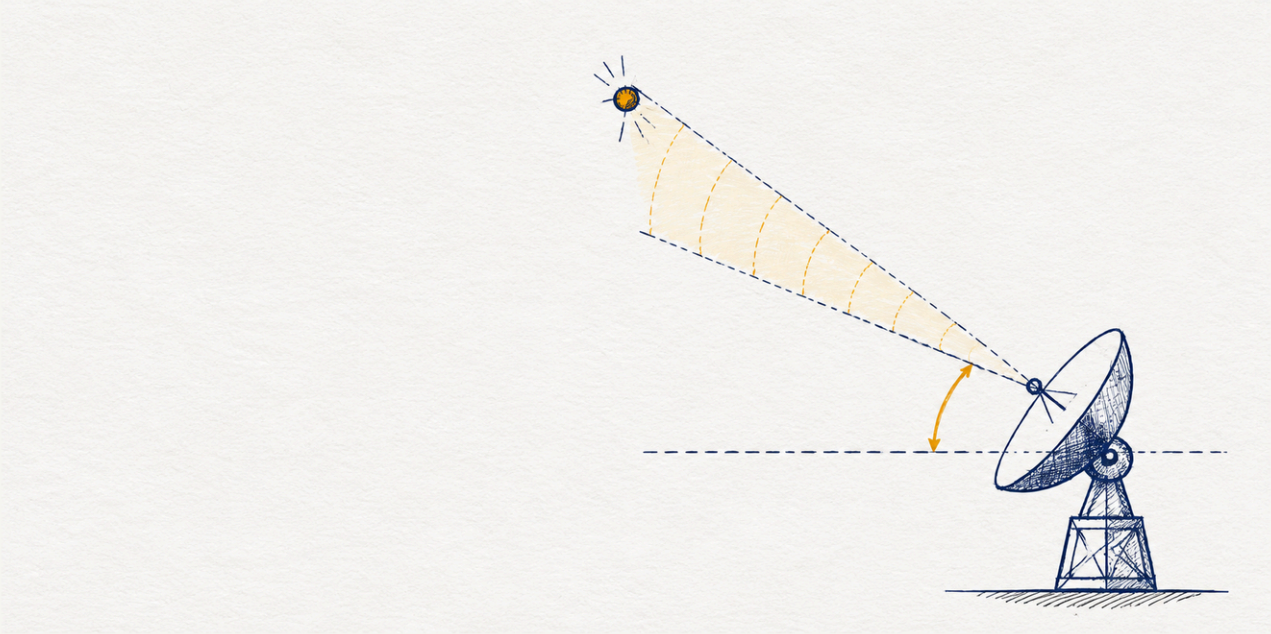
角度测量

作者：唐承乾

版本：社区版 V2

官方仓库

https://github.com/apple-art/easy-radar-tutorial



目录

CONTENTS

前面几章把雷达处理链拆开讲过，第 3 章把多发脉冲的 I/Q 样本排成 `rx_matrix`，第 4 章在快时间上做距离处理，第 5 章在慢时间上做 Doppler 处理，第 6 章把距离-速度图上的响应筛成检测结果，第 7 章说明方向信息怎样进入目标参数。

本章把这些步骤放到同一个 MATLAB 程序。每个变量都要讲清楚它的维度是什么、下一步沿哪一维处理、处理后又对应哪个物理量。

本章采用简化 pulse-Doppler 场景：两个点目标，32 发脉冲，基带 LFM 信号。完整脚本在配套 MATLAB 附件，正文只保留主要片段和检查方法。

1.1. 8.1 仿真条件

本章用两个点目标作为例子。雷达参数延续前面几章的简化设定：使用基带 LFM 脉冲，接收端得到复数 I/Q 样本，多发脉冲按发射顺序组成 `rx_matrix`。

参数	MATLAB 变量	取值	含义
光速	`c`	$3\times10^{8}\,m/s$	距离与时延换算
载频	`fc`	$10\,GHz$	决定波长
波长	`lambda`	$0.03\,m$	Doppler 到速度的换算
LFM 带宽	`B`	$10\,MHz$	影响距离分辨率
脉冲宽度	`Tp`	$20\,\mu s$	发射脉冲持续时间
采样率	`fs`	$20\,MHz$	快时间采样间隔
PRF	`PRF`	$5\,kHz$	慢时间采样率
脉冲数	`Npulse`	32	决定 CPI 和速度分辨率
快时间采样点数	`Nfast`	2048	决定接收窗口长度

两个目标的真值设定如下：

目标	距离	径向速度	幅度
A	$6\,km$	$+30\,m/s$	1.0
B	$9\,km$	$-20\,m/s$	0.75

这里的速度都落在本例的不模糊速度范围内。因为

$$v_{\max}=\frac{\lambda\,\mathrm{PRF}}{4}=\frac{0.03\times5000}{4}=37.5\,m/s$$

所以 $+30\,m/s$ 和 $-20\,m/s$ 都能在双边速度轴上显示出来。若把 PRF 降低到 $2\,kHz$，$v_{\max}$ 会变成 $15\,m/s$，目标 A 就会发生速度模糊。

scene 里的距离和速度是真值，只在仿真里存在，用来生成回波。target_list 是处理链从回波里估计出来的，真实雷达只有这一侧，没有真值可查。

1.2. 8.2 从回波矩阵到目标表

处理链写成一行数据流：

```
scene -> tx -> rx_matrix -> range_data -> rd_map -> det_mask ->
target_list
```

各变量的作用和维度如下：

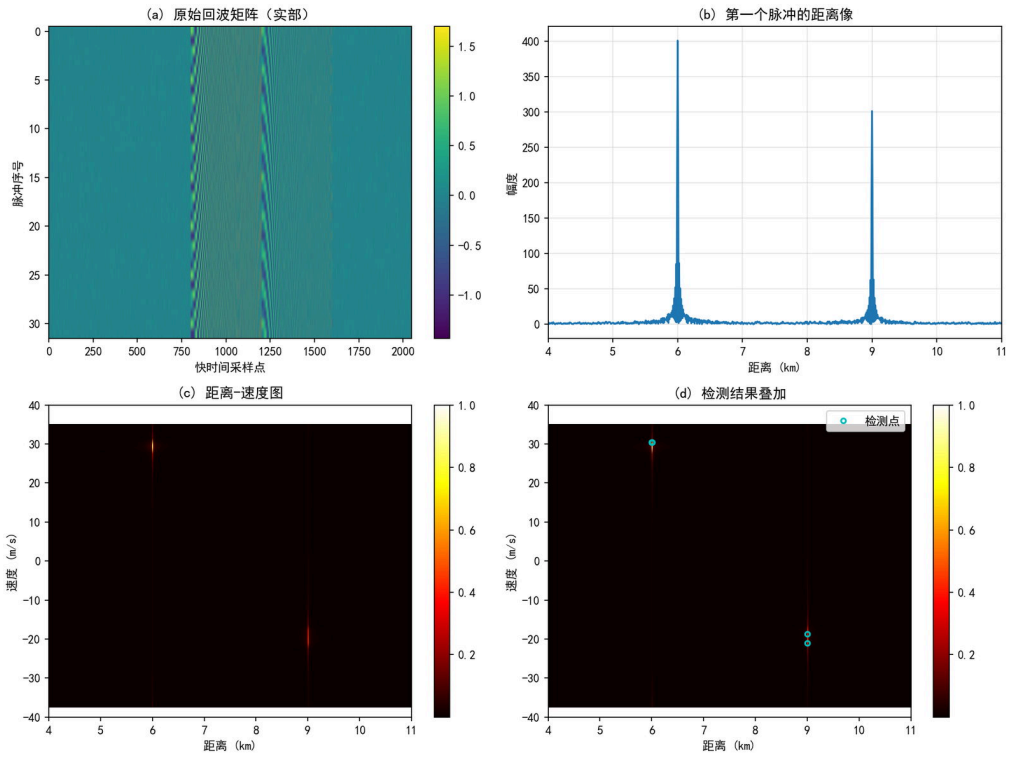
变量	典型维度	轴含义	物理含义
scene	结构体	不适用	目标真值和雷达参数
tx	1 x Nt	快时间	发射基带 LFM 脉冲
rx_matrix	Npulse x Nfast	行：脉冲序号；列：快时间采样点	原始复数回波矩阵
range_data	Npulse x Nfast	行：脉冲序号；列：距离相关采样点	距离压缩结果
rd_map	Ndoppler x Nrange	行：Doppler / 速度 bin；列：range bin	距离-速度图
det_mask	与 rd_map 相同	行列与 rd_map 对齐	检测掩码
target_list	Ntarget x 字段数	每行一个目标	距离、速度、响应强度等估计结果

顶层程序组织成下面的形式：

```
scene = ch08_default_scene();
[tx, mf, k] = ch08_make_lfm_pulse(scene.B, scene.Tp, scene.fs);
rx_matrix = ch08_simulate_echo(scene, k);
[range_data, range_axis] = ch08_range_compress(rx_matrix, mf,
scene.c, scene.fs);
[rd_map, rd_power, vel_axis] = ch08_doppler_process(range_data,
scene);
[det_mask, row_idx, col_idx] = ch08_detect_targets(rd_power, 0.35);
target_list = ch08_make_target_list(row_idx, col_idx, rd_power,
range_axis, vel_axis);
```

这段代码只看函数接口，不看函数内部。rx_matrix 是接收样本矩阵，到 target_list 才是可以汇报的目标结果，每一步都在改变数据的组织方式。

我们再检查尺寸：



完整处理链的四个阶段

```
size(rx_matrix)
size(range_data)
size(rd_map)
size(det_mask)
height(target_list)
```

如果 `rx_matrix` 和 `range_data` 的行列方向已经弄反，后面的 Doppler FFT、速度轴和检测点都会跟着错。

图 8-1 展示了完整处理链的四个阶段。

1.3.　8.3 回波矩阵的生成

1.3.1.　场景参数怎样进入回波

`scene` 里每个目标至少有三个参数：距离 R_i、径向速度 v_i 和幅度 A_i。距离决定回波晚到多少，速度决定相邻脉冲之间多转多少相位。

距离到时延的关系仍然是第 4 章的公式：

$$\tau_i = \frac{2R_i}{c}$$

速度到 Doppler 频率的关系仍然是第 5 章的公式：

$$f_{d,i} = \frac{2v_i}{\lambda}$$

第 pulse_idx 发脉冲中，目标 i 的慢时间相位写成

$$\exp\left(j2\pi f_{d,i}(\text{pulse\_ idx} - 1)T_r\right)$$

其中 $T_r = 1/\text{PRF}$。程序里只需要把这两个关系放进回波生成函数：

```
tau = 2 * R / scene.c;
fd = 2 * v / scene.lambda;
delayed_t = fast_time - tau;
valid = delayed_t >= 0 & delayed_t < scene.Tp;
echo(valid) = amp * exp(1j * pi * k * delayed_t(valid).^2) ...
    .* exp(1j * 2*pi * fd * (pulse_idx - 1) * scene.Tr);
```

前半部分 exp(1j*pi*k*delayed_t.^2) 是延迟后的基带 LFM 回波，后半部分是目标运动带来的脉冲间相位变化。第 4 章处理的是前半部分怎样形成距离峰，第 5 章处理的是后半部分怎样形成速度峰。

1.3.2. rx_matrix 的行和列

回波生成函数最终输出 rx_matrix。本章程序采用下面的约定：

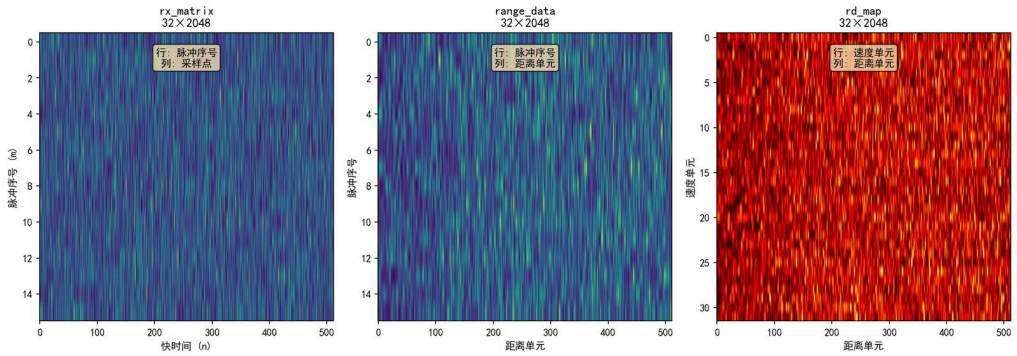
```
rx_matrix: Npulse x Nfast
```

也就是每一行是一发脉冲的快时间采样，每一列是同一个快时间位置在不同脉冲之间的慢时间序列。

方向	索引	对应物理量	后续处理
行方向	pulse_idx	慢时间 / 脉冲序号	Doppler FFT 沿这个方向做
列方向	fast_idx	快时间采样点	距离压缩沿这个方向做

这个约定和第 3 章的 rx_matrix 保持一致。若程序采用相反约定也能工作，但每一个 fft(..., dim)、conv 循环和 imagesc 坐标都要同步改变。教材里不混用两套约定。

图 8-3 展示了数据在处理链中的维度变化（图中只显示部分采样点作为示意）。rx_matrix 和 range_data 的尺寸相同，但列的物理含义已经从“采样点”变成“距离单元”。rd_map 的行方向则从“脉冲序号”变成“速度单元”。



数据维度在处理链中的变化

1.4.　8.4 距离压缩得到 range_data

1.4.1.　沿快时间做匹配滤波

rx_matrix 的每一行是一发脉冲的回波。距离压缩要在这一行内部做匹配滤波，把延迟后的 LFM 回波压成距离峰。

第 4 章中匹配滤波写成

$$h(t) = s^*(-t), \qquad y(t) = r(t)*h(t)$$

程序里的主要片段是：

```
range_data = zeros(size(rx_matrix));
for pulse_idx = 1:size(rx_matrix, 1)
    range_data(pulse_idx, :) = conv(rx_matrix(pulse_idx, :), mf,
'same');
end
```

这里的 pulse_idx 固定一发脉冲，: 取出这一发脉冲内的全部快时间采样点。conv(..., 'same') 保持输出长度和输入长度相同，所以 range_data 的尺寸仍然是 Npulse x Nfast。

处理前后最容易混淆的是列方向含义。rx_matrix 的列是快时间采样点，range_data 的列已经可以配上距离轴，但它仍然不是最终目标，只是每发脉冲上的距离压缩结果。

1.4.2.　距离轴和群延迟修正

匹配滤波器有长度。使用 conv(..., 'same') 时，输出保留中心段，距离轴要扣除匹配滤波器的中心偏移：

```
group_delay = floor(length(mf) / 2);
Nfast = size(rx_matrix, 2);
range_axis = ((0:Nfast-1) - group_delay) * scene.c / (2 * scene.fs);
```

这行距离轴来自第 4 章的测距公式：采样点先换成时间，再由时间换成距离。

$$R = \frac{c\Delta t}{2} = \frac{c(n - n_0)}{2f_s}$$

其中 n_0 就是 `group_delay`。若不扣除这个偏移，距离估计会系统性偏大。

1.5. 8.5 Doppler 处理得到 `rd_map`

1.5.1. 沿慢时间做 FFT

距离压缩完成后，`range_data` 的每一列是同一个距离单元在不同脉冲之间的慢时间序列。第 5 章已经讲过，这个序列的频谱就是 Doppler 谱。

程序里写成：

```
rd_map = fftshift(fft(range_data, [], 1), 1);
rd_power = abs(rd_map);
rd_power = rd_power / max(rd_power(:));
```

这里 `fft(..., [], 1)` 的最后一个参数 `1` 表示沿第一维处理。因为本书约定 `range_data` 的行是脉冲序号，所以 Doppler FFT 沿第 1 维做。

做完这一步后，行方向从脉冲序号变成了 Doppler bin 或速度 bin，列方向仍然对应距离。

变量	行方向	列方向
`range_data`	脉冲序号	距离相关采样点
`rd_map`	Doppler / 速度 bin	range bin

这一步容易出错。`rd_map` 看起来仍然是二维矩阵，但它已经不再是一张脉冲-快时间表。

1.5.2. 速度轴和 `fftshift`

双边复 I/Q 序列的 Doppler 频率范围写成

$$f_d \in \left[-\frac{\mathrm{PRF}}{2}, \frac{\mathrm{PRF}}{2}\right)$$

因此速度轴写成：

```
Npulse = scene.Npulse;
doppler_freq = (-Npulse/2:Npulse/2-1) * scene.PRF / Npulse;
vel_axis = doppler_freq * scene.lambda / 2;
```

`fftshift` 把零 Doppler 移到中间。没有 `fftshift` 时，速度轴仍按双边方式标，图上的速度位置会错位。

本例中速度分辨率为

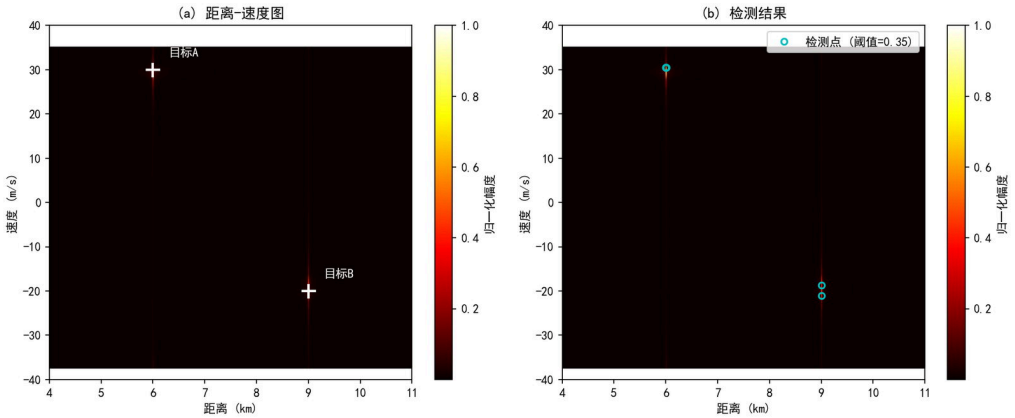
$$\Delta v = \frac{\lambda}{2T_{\mathrm{CPI}}} = \frac{\lambda\,\mathrm{PRF}}{2N}$$

代入 $\lambda = 0.03\,m$、$\mathrm{PRF} = 5000\,Hz$、$N = 32$，得到

$$\Delta v \approx 2.34\,m/s$$

所以目标速度不一定正好落在某一个速度 bin 上，实际峰值会出现在最接近的 bin 附近。

图 8-2 展示了距离-速度图的细节。左图标注了两个目标的真实位置，右图叠加了检测结果。有的目标附近会出现多个检测点，这是因为主瓣展宽和固定阈值的共同作用。



距离-速度图与检测结果

1.6.　8.6 从 rd_map 到 det_mask

1.6.1.　固定阈值先跑通

距离-速度图上的亮点还不是目标表。第 6 章已经讲过，图上的响应可能来自目标，也可能来自噪声、旁瓣或杂波。检测要把连续幅度图变成 0/1 掩码。

最小实现先用固定阈值：

```
threshold = 0.35;
det_mask = rd_power > threshold;
[row_idx, col_idx] = find(det_mask);
```

`det_mask` 和 `rd_map` 使用同一套行列坐标。`det_mask(row, col)=1` 表示 `rd_map(row, col)` 这个距离-速度单元通过检测。

固定阈值适合先跑通处理链。但背景不均匀、旁瓣高、强弱目标差距大时，一个全局阈值很难兼顾整张图。第 6 章讲过的 CFAR 可以替换这个检测模块，但第 8 章不展开二维 CFAR 代码。

1.6.2.　检测掩码的常见误读

`det_mask` 中的 1 不一定一一对应目标。一个目标的主瓣附近可能有多个相邻格子越过阈值，强目标旁瓣也可能触发检测。实际输出目标表前，通常还要做局部峰值保留或点迹凝聚。

一个简化的局部峰值保留：

```
is_peak = imregionalmax(rd_power);
det_mask = (rd_power > threshold) & is_peak;
[row_idx, col_idx] = find(det_mask);
```

这段代码需要图像处理工具箱。没有这个工具箱时，也可以用第 6 章的点迹凝聚思路：先找到通过阈值的区域，再取每个区域内响应最大的格子作为代表点。

1.7.　8.7 从检测掩码到目标表

1.7.1.　索引到物理量的换算

`find(det_mask)` 返回的是行列索引，还不是距离和速度。换算关系写成：

```
range_est = range_axis(col_idx);
vel_est = vel_axis(row_idx);
```

其中 `range_axis` 对应距离坐标轴，`vel_axis` 对应速度坐标轴。这两个轴在前面的距离压缩和 Doppler 处理中已经生成。

到这一步，前面得到的距离-速度图才真正转化为可以汇报的目标结果。

1.7.2. 目标表的组织

最简单的目标表只保留距离、速度和响应强度：

```
target_list = table(range_est, vel_est, ...
rd_power(sub2ind(size(rd_power), row_idx, col_idx)), ...
    'VariableNames', {'Range', 'Velocity', 'Power'});
```

实际工程中，目标表还会包含检测时刻、信噪比估计、角度信息等字段。第 7 章讲过的角度测量可以在这里并入目标表。

1.8. 8.8 参数变化练习

练习 1：改变目标距离。把目标 A 的距离从 $6\,km$ 改成 $7\,km$，观察 `range_data` 的距离峰和 `rd_map` 中亮点的横向位置。速度位置不应明显改变。

练习 2：改变目标速度。把目标 B 的速度从 $-20\,m/s$ 改成 $+20\,m/s$，观察 `rd_map` 中亮点从负速度一侧移动到正速度一侧。距离位置不应明显改变。

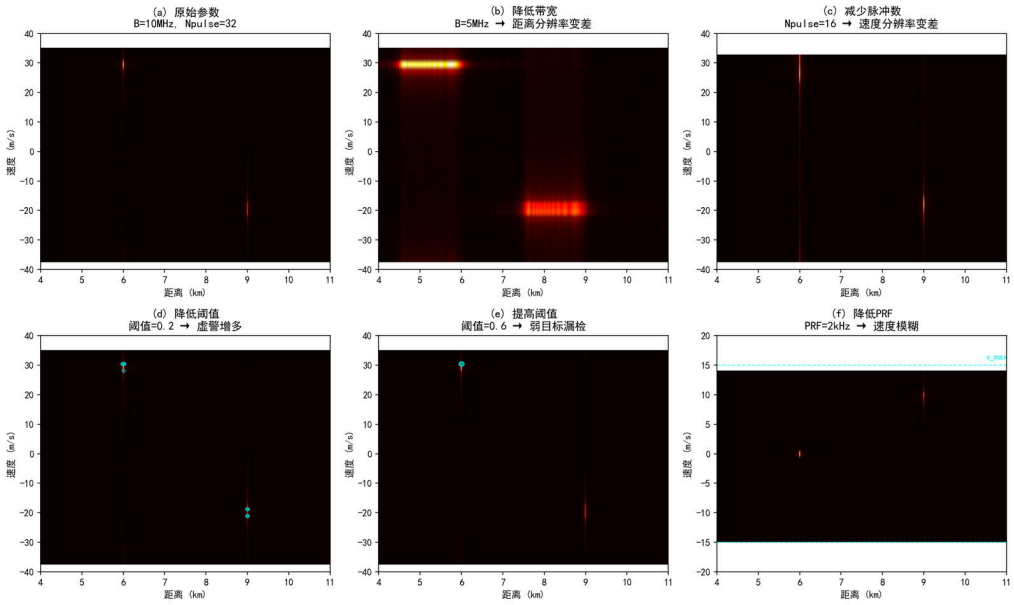
练习 3：降低脉冲数。把 `Npulse` 从 32 改成 16。速度 bin 变粗，`rd_map` 的速度向分辨能力下降。这个变化对应第 5 章的 CPI 与速度分辨率关系。

练习 4：降低 PRF。把 `PRF` 从 $5\,kHz$ 改成 $2\,kHz$。最大不模糊速度变成 $15\,m/s$，目标 A 的 $+30\,m/s$ 会折叠到 $0\,m/s$ 附近，目标 B 的 $-20\,m/s$ 会折叠到 $+10\,m/s$ 附近。观察 `vel_axis` 的范围和峰值位置变化。

练习 5：调整阈值。把 `threshold` 从 0.35 改成 0.2 和 0.6。阈值降低时检测点会增多，阈值升高时弱目标更容易消失。这个练习对应第 6 章的虚警和漏检取舍。

练习 6：加入一个强零速度杂波。在某个距离附近加入速度为 $0\,m/s$、幅度较大的回波。距离-速度图上零速度附近会出现强响应，检测模块可能优先报出它。这个练习用来连接第 5 章的零 Doppler 杂波和第 6 章的检测限制。

图 8-4 展示了不同参数设置对处理结果的影响。左列是原始设置，右列分别展示了改变带宽、脉冲数和 PRF 后的效果。



参数变化对处理结果的影响

跑完这些练习后，可以对照 `scene` 里的真值，检查 `target_list` 每一行的距离和速度估计是否落在预期位置。