DSP大作业

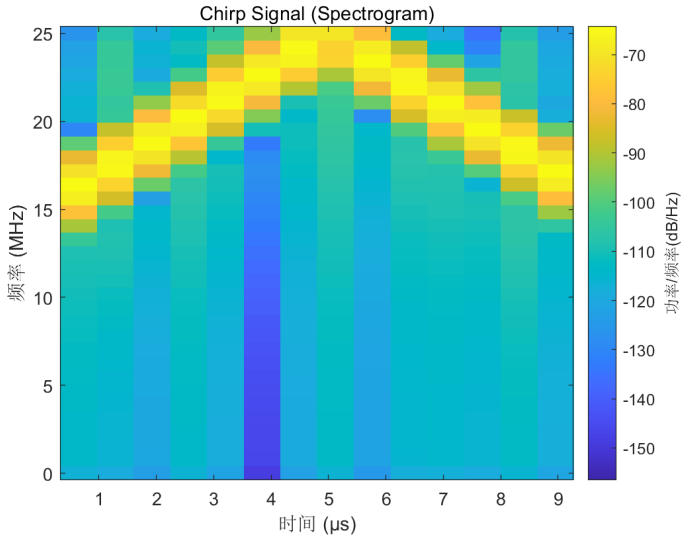
利用STFT验证信号参数与要求相同

MATLAB代码如下：

```
fs = 50e6; % 采样率为50MHz
T = 10e-6; % 脉冲持续时间为10μs
t = 0: 1/fs: T - 1/fs; % 时间向量
f0 = 15e6; % 起始频率为15MHz
f1 = 35e6; % 结束频率为35MHz

% 使用chirp函数生成线性调频信号
chirp_signal = chirp(t, f0, T, f1, 'linear', 90);
figure;
% 使用spectrogram函数进行短时傅里叶变换
window_size = 64; % 窗口大小
overlap = 32; % 重叠大小
nfft = window_size; % FFT点数
spectrogram(chirp_signal, window_size, overlap, nfft, fs, 'yaxis');
title('Chirp Signal (Spectrogram)');
```

使用了`spectrogram`函数进行了STFT，结果如下：



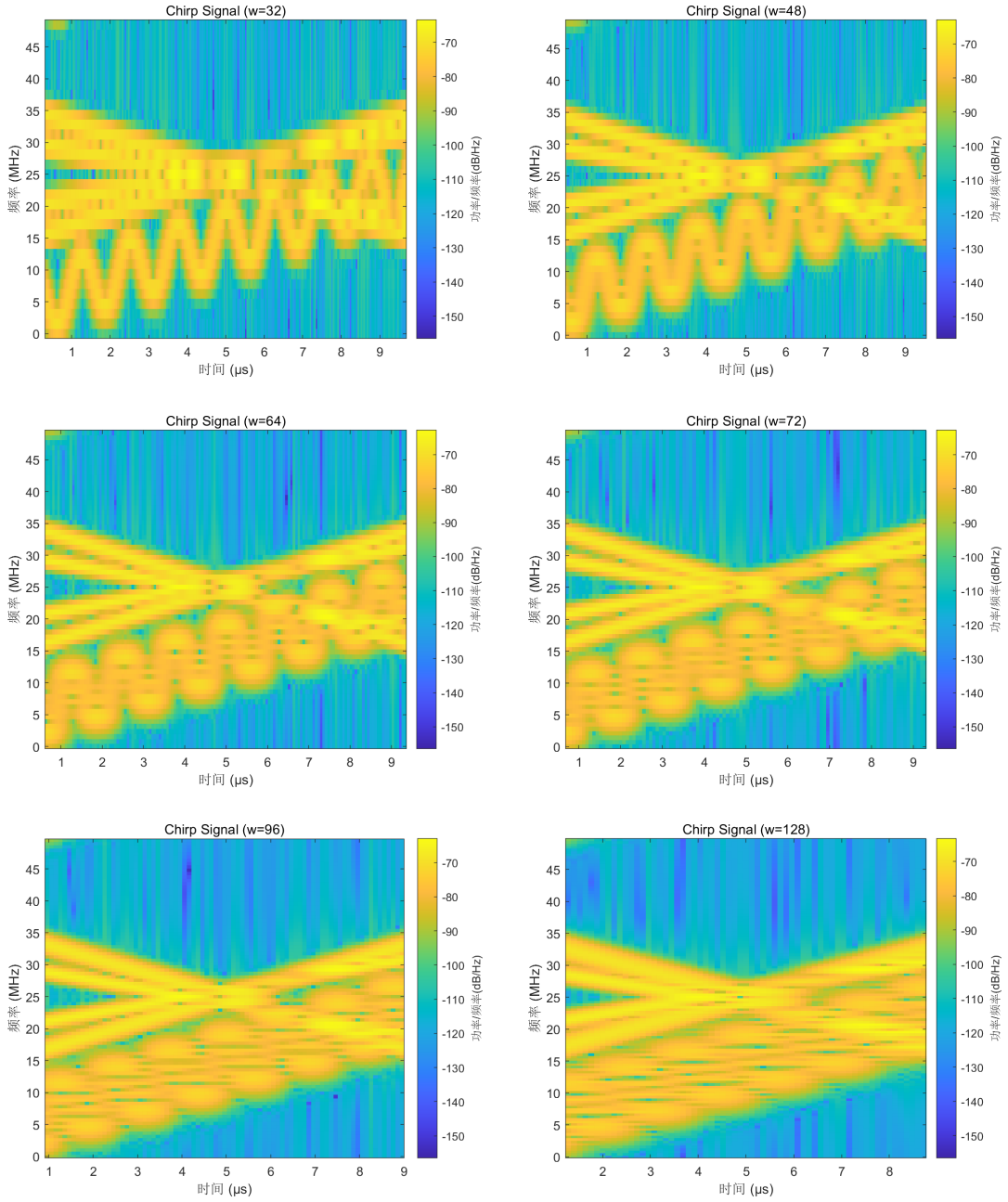
可以发现，在0-5μs，信号的频率从15MHz增加到25MHz，在5-10μs，信号的频率从25MHz下降到15MHz。由于采样率只有50MHz，故最多只能分辨25MHz的频率，对于高于25MHz的频率，将会发生混叠。25-35MHz的频率会被混叠到-25~-15MHz，也即25~15MHz，与图像相符。

分析滑动窗长度对时频域分辨率的影响

下面是仿真代码：

```
load('wave_data.mat');
chirp_signal2 = chirp(t, 20e6, T, 32e6, "linear", 90);
observed_signal = chirp_signal + chirp_signal2 + wave_data;
figure;
window_size = 128; % 窗口大小
overlap = round(0.95 * window_size); % 重叠大小
nfft = window_size; % FFT点数
spectrogram(observed_signal, window_size, overlap, nfft, fs, 'yaxis');
title("Chirp Signal (w=" + num2str(window_size) + ")");
```

仿真时，调整`window_size`的大小，可以生成不同时频分辨率的信号：



较短的滑动窗长度意味着更高的时域分辨率，可以更好地分辨信号中快速变化的瞬时特性。然而，较短的窗口长度将导致频谱的主瓣展宽，频域分辨率下降。

对于`wave_data`，它的频率大概是**线性函数加上一个正弦函数**，如果窗长过大，快变的正弦信号就会“混叠”，难以分辨；而如果窗长过短，过低的频率分辨率会使两个chirp信号难以分辨。

综上，窗长为48时较为合适。

带噪单chirp信号估计

原理

参考了TIM上的"Polynomial Chirplet Transform With Application to Instantaneous Frequency Estimation"论文，主要是想使用chirplet transform（CT）来分析chirp信号

对信号$s(t)$的CT的定义如下：

$$CT_s(t_0,\omega,\alpha;\sigma)=\int_{-\infty}^{+\infty}\overline{z}(t)w_\sigma(t-t_0)\exp(-j\omega t)dt$$

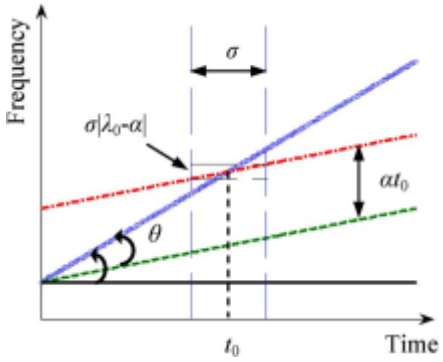
其中，

$$\begin{aligned}z(t)&=s(t)+j\mathcal{H}\{s(t)\}\\\overline{z}(t)&=z(t)\Phi_\alpha^R(t)\Phi_\alpha^M(t,t_0)\\\Phi_\alpha^R(t)&=\exp(-j\alpha t^2/2)\\\Phi_\alpha^M(t,t_0)&=\exp(j\alpha t_0 t)\\w_\sigma(t)&=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{t^2}{2\sigma^2}\right)\end{aligned}$$

$z(t)$是$s(t)$的解析信号，$w_\sigma(t)$是高斯窗函数，CT其实是对$\overline{z}(t)w_\sigma(t-t_0)$的傅里叶变换，只需要考察$\overline{z}(t)w_\sigma(t-t_0)$的性质，就能推测出CT的性质。

- 在 time–frequency distributions (TFD）上，$\Phi_\alpha^R(t)$ 是一个旋转（**R**otate）操作，如果原信号的TFD是直线的，那么乘上$\Phi_\alpha^R(t)$后，直线的斜率会减小α。
- $\Phi_\alpha^M(t,t_0)$ 是一个频移操作，它将 ω 处的频率分量平移到$\omega+\alpha t_0$频率处。
- 注意窗函数

下面的图（**摘自原论文**）较为形象地展示了chirplet变换对TFD的操作，假设原来的chirp信号的频率为$\omega_0+\lambda_0 t$ (图中的青色线)



CT先将其旋转，变成绿线，注意到直线的斜率变成了$\lambda_0-\alpha$；再进行平移操作，变成红线。在带宽为σ的高斯窗内，青色线的带宽为$\lambda_0\sigma+1/\sigma$；绿色线和红色线的带宽为$\sigma|\lambda_0-\alpha|+1/\sigma$。当$\alpha=\lambda_0$ 时，带宽会取得最小值，$1/\sigma$。这意味着CT能使得TFD的**能量最为集中**。所以，$|CT_s(t_0,\omega,\alpha;\sigma)|$在$(\omega,\alpha)=(\omega_0,\lambda_0)$ 处取得全局最大值。这是我们估计的理论基础。

算法步骤与代码实现

估计是基于**最小二乘**的**迭代**求解策略（**部分参考TIE上的Multicomponent Signal Analysis Based on Polynomial Chirplet Transform**），大致步骤如下：

- 对信号做chirplet变换（初始的$\alpha = 0$），得到TFD
- 对TFD的每个时刻找到幅频响应最大值对应的频率及其幅频响应，再找到幅频响应的全局最大值
- 忽略最大幅频响应较低的时刻（本方法中直接将小于0.5倍全局最大值的时刻丢弃）
- 认为剩下的时间就是信号的持续时间，每个时刻的频率就是前述的幅频响应最大值对应的频率。使用线性最小二乘拟合，得到斜率，就是频率随时间的变化率
- 将 α 更新成斜率，重复上述4个步骤，直到收敛

实现的代码如下：

```
start_time = 3.8e-6;
duration = 7.9e-6 - start_time;
zero_freq = 6e6;
end_freq = 23.8e6;
% 生成chirp信号
chirp_signal = chirp(t, zero_freq, T, end_freq, "linear", 90);
init_freq = start_time / 1e-5 * (end_freq - zero_freq) + zero_freq;
bandwidth = abs(duration / 1e-5 * (end_freq - zero_freq));
% 将chirp信号在时域上截断
observed_signal = truncated_signal(chirp_signal, start_time, start_time + duration,
fs);
% 信噪比
SNR = 3:30;
% 每个信噪比下重复仿真的次数
simu_num = 10;
start_time_est = zeros(simu_num, length(SNR));
init_freq_est = zeros(simu_num, length(SNR));
duration_est = zeros(simu_num, length(SNR));
bandwidth_est = zeros(simu_num, length(SNR));

for k = 1:length(SNR)
    for j = 1:simu_num
        % 添加噪声
        noisy_signal = awgn(observed_signal, SNR(k));
        % 估计参数
        [start_time_est(j, k), init_freq_est(j, k), duration_est(j, k),
bandwidth_est(j, k)] = single_chirp(noisy_signal, t, fs);
    end
end
% 计算RMSE
start_time_RMSE = sqrt(mean((start_time_est - start_time).^2, 1));
init_freq_RMSE = sqrt(mean((init_freq_est - init_freq).^2, 1));
duration_RMSE = sqrt(mean((duration_est - duration).^2, 1));
bandwidth_RMSE = sqrt(mean((bandwidth_est - bandwidth).^2, 1));
```

其中`truncated_signal`是将信号在时域上截断，使得$0 < t < t_1$和$t_2 < t < 10\mu s$的信号为0，实现如下：

```
function observed_signal = truncated_signal(signal, t1, t2, fs)
    % t1 为信号的起始时间
    % t2 为信号的结束时间
    % fs 为采样率
    len = length(signal);
    t = ones(1, len);
    index1 = floor(t1*fs) + 1;
    index2 = floor(t2*fs) + 1;
    t(1:index1 - 1) = 0;
    t(index2+1:end) = 0;
    observed_signal = signal .* t;

end
```

`single_chirp`是用于对单chirp信号估计的函数，实现如下：

```
function [start_time, init_freq, duration, bandwidth] =
single_chirp(observed_signal, t, fs)
    % 频率轴下标的个数
    fLevel = 256;
    % 高斯窗的窗长
    WinLen = 64;
    % 估计出来的频率变化率
    alpha = 0;
    alpha_pre = 0;
    % 迭代次数
    iter_cnt = 0;
    % 若前后两次估计出来的alpha相差太多，则继续迭代
    while (iter_cnt == 0 || abs(alpha - alpha_pre) / abs(alpha_pre) > 0.02)
        alpha_pre = alpha;
        % 进行chirplet变换
        % Spec为频率响应
        [Spec, Freq, ~] = Chirplet_Transform(observed_signal, fLevel, WinLen, fs,
alpha);
        spec_abs = abs(Spec);
        % max_if_per_inst：每个时刻的幅频响应的最大值
        % index: 最大值对应的频率下标
        [max_if_per_inst, index] = max(spec_abs);
        % 整个时频图中的最大值
        max_if = max(max_if_per_inst);
        % 忽略最大幅频响应较低的时刻
        observed_win = max_if_per_inst > 0.5 * max_if;
        % 取出这些时刻，认为这就是chirp信号的持续时间
        observed_time = t(observed_win);
        % 取出这些时刻的幅频响应最大值对应的频率
        observed_freq = Freq(index(observed_win));
        % 使用线性最小二乘拟合
        coefficients = polyfit(observed_time, observed_freq, 1);
```

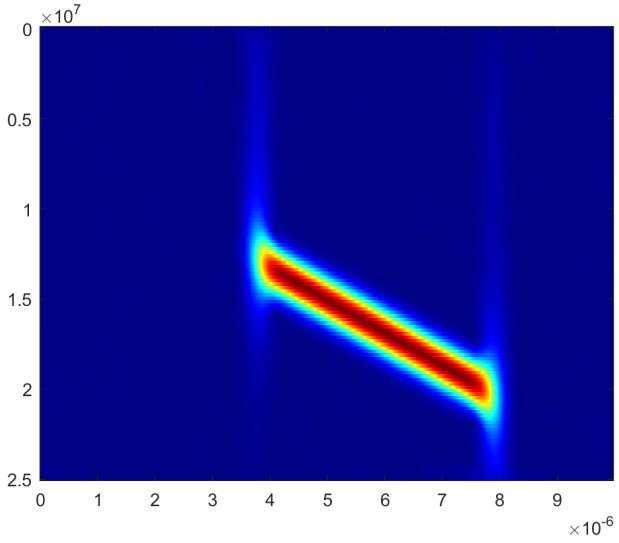
```
        % 斜率即为频率变化率
        alpha = coefficients(1);
        iter_cnt = iter_cnt + 1;
        % 至多迭代10次
        if(iter_cnt > 10)
            break;
        end
    end

    start_time = observed_time(1);
    init_freq = observed_freq(1);
    duration = observed_time(end) - observed_time(1);
    bandwidth = abs(observed_freq(end) - observed_freq(1));
```

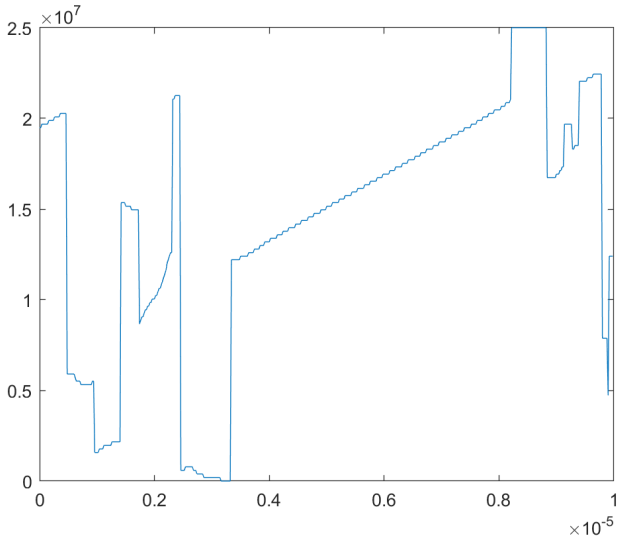
其中，`Chirplet_Transform`**直接采用现有实现**（https://ww2.mathworks.cn/matlabcentral/fileexchange/72303-chirplet-transform）。

结果和讨论

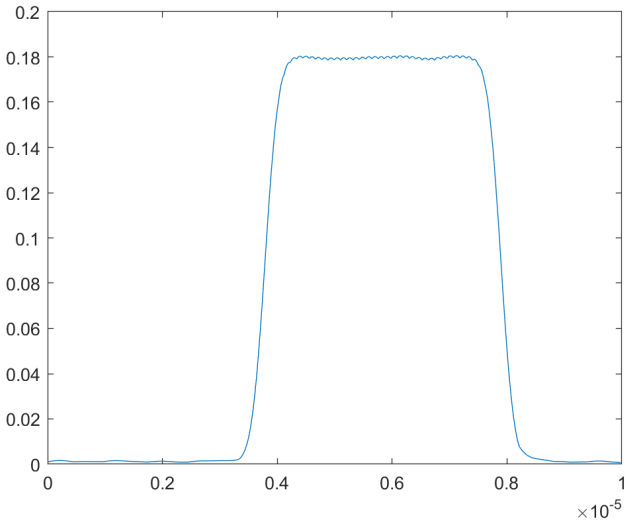
下面给出了上述信号在SNR=40dB时的一些其他结果：



上面是chirplet变换后的TFD，发现相比于STFT，能量更为集中，线性的频率变化也更加明显。

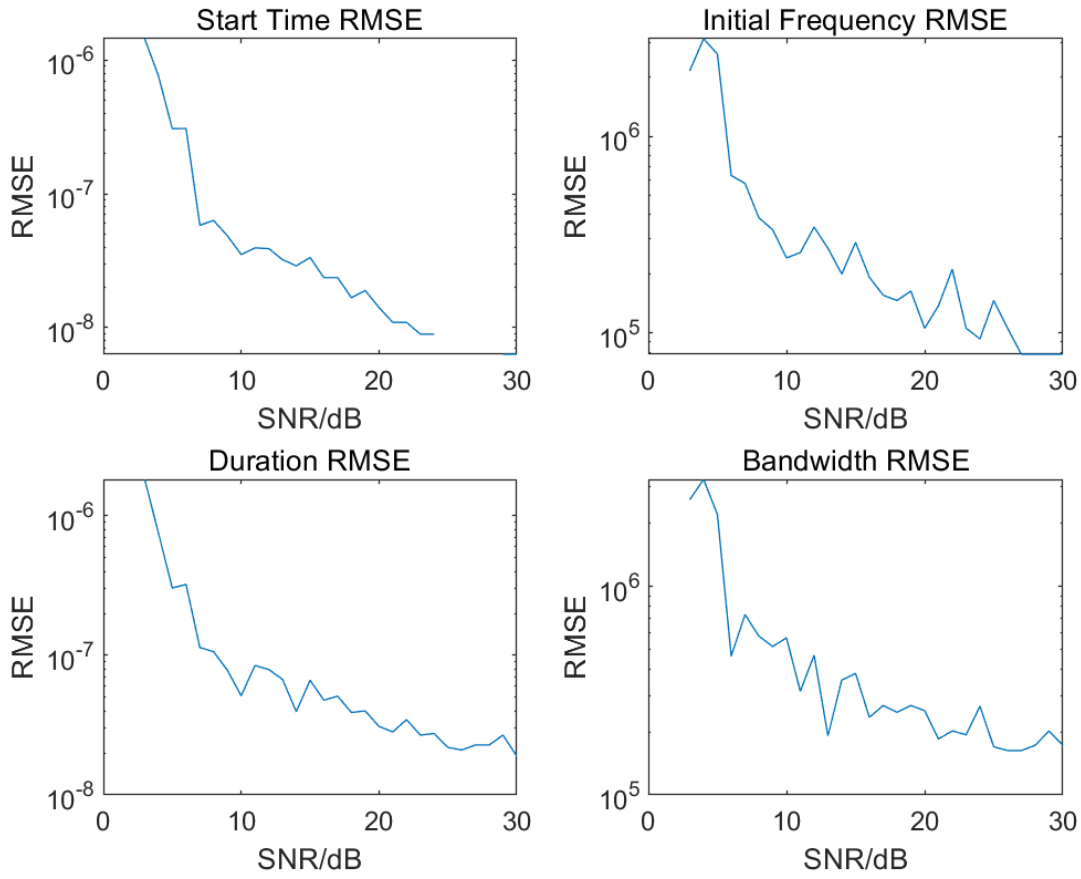


上图是每个时刻幅频响应最大值对应的频率，可以发现，在信号的持续时间内（0.38~0.79e-5s），所得频率及其变化率与真实值很接近。



上图是每个时刻幅频响应的最大值，可以发现，在信号的持续时间内，幅频响应很高且近似不变，取最大幅频响应的一半截断，得到的信号的持续时间和真实的信号的持续时间（0.38~0.79e-5s)很接近。

在上面代码中的参数设置下，各参数估计的RMSE如下：

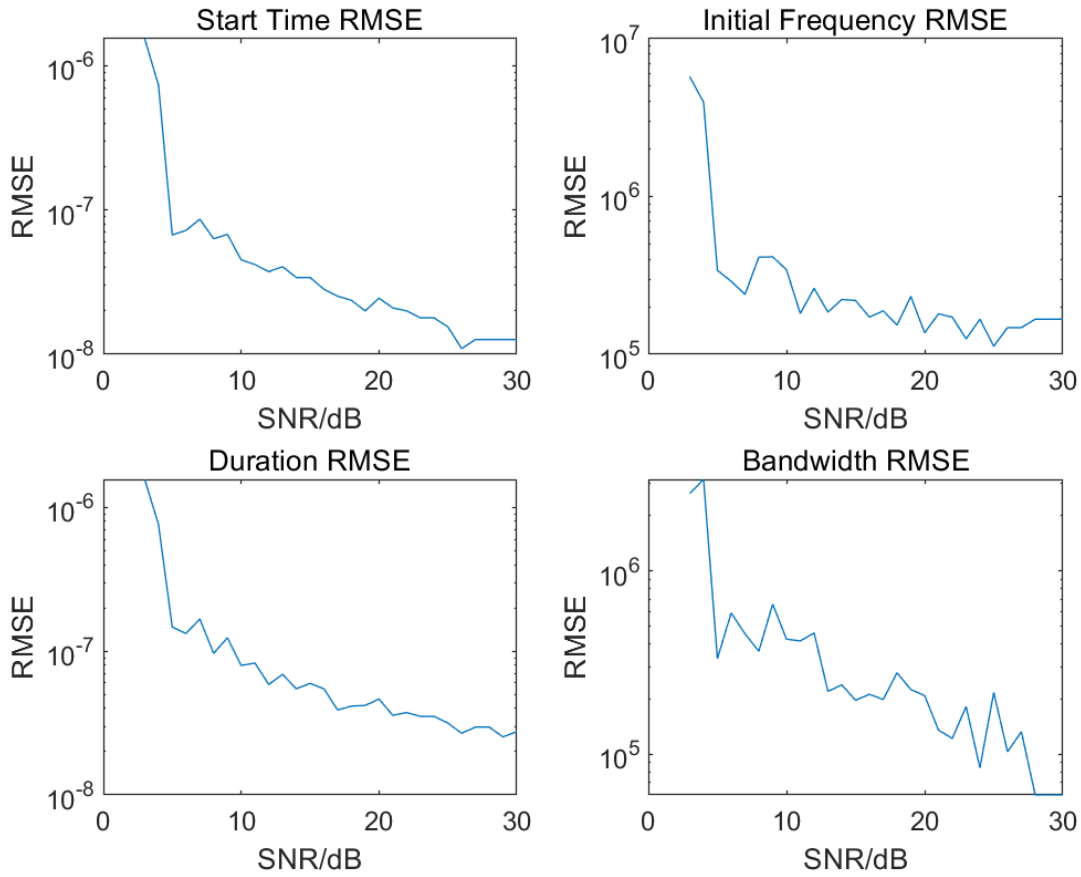


其中，信号开始时间的RMSE在SNR较大时为0，所以曲线出现了中断。可以发现，各参数的RMSE随SNR的升高呈下降趋势，但略有抖动，这可能是由于在估计信号的持续时间时，采用了小于0.5倍最大值直接截断的经验策略，而并没有给出一个理论上的性能分析。

更换一组参数：

```
start_time = 2.9e-6;
duration = 9.1e-6 - start_time;
zero_freq = 18.8e6;
end_freq = 13.1e6;
```

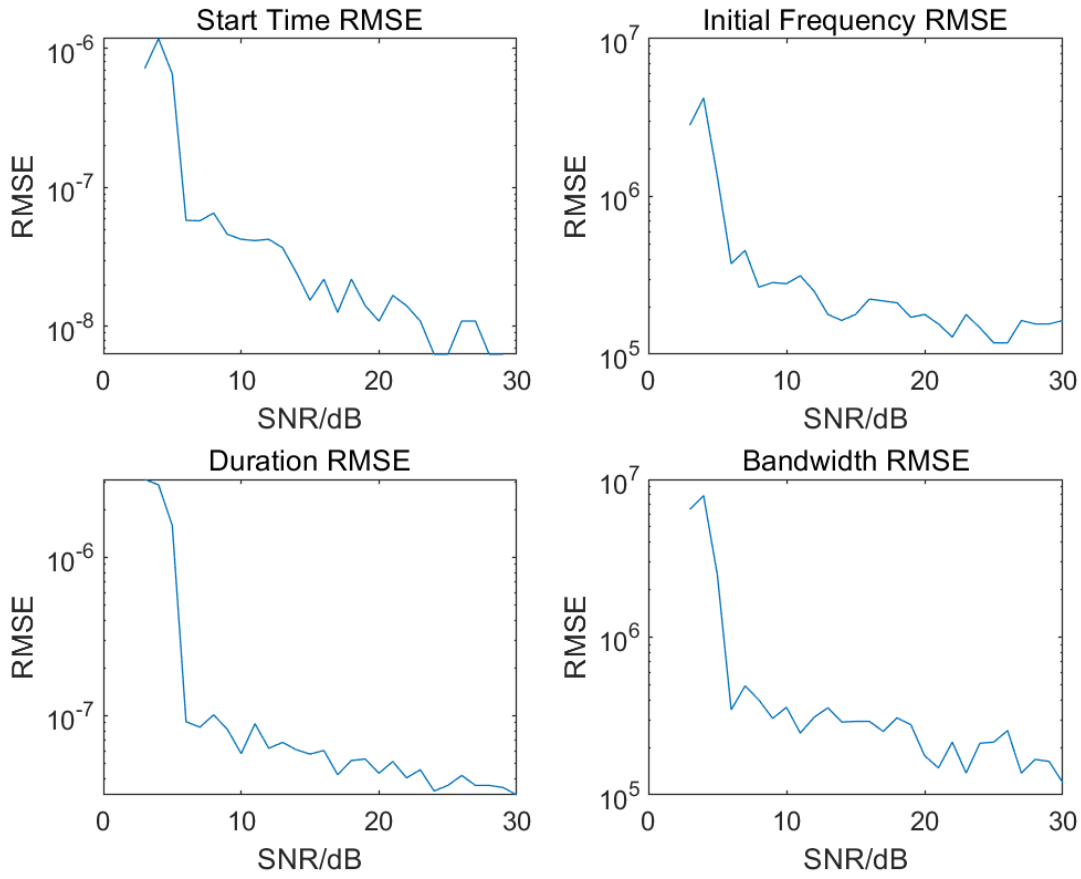
得到的结果依然较好：



再更换一组参数：

```
start_time = 2.3e-6;
duration = 4.9e-6 - start_time;
zero_freq = 18e6;
end_freq = 15.2e6;
```

得到的结果依然较好：



带噪多chirp信号估计

分析

多chirp信号估计的问题在于以下几个方面：

- 信号数量未知；即使信号数量已知，估计的结果如何保证不会出现重复或者遗漏
- 每个chirp信号的幅度是未知的
- chirp信号在TFD中可能交叠，这就意味着**单chirp信号估计的方法不再适用**（因为没有办法确定每个信号的起止时间）

我们通过以下假设（或者方法）来处理这个问题：

- 假设所有chirp信号具有相同的幅度，不妨设其为1
- 借助**图像处理**中的Hough变换来处理TFD，根据其检测到的直线进行拟合

算法步骤和代码实现

下面是具体的方法：

- 对带噪信号做chirplet变换（$\alpha = 0$），得到TFD
- 对scaled后的TFD应用Hough变换，找到较多数目（算法采用了30个）的Hough空间的峰值，及其对应的直线以及斜率
- 对上述直线进行合并和去重操作

- 以剩下的直线的斜率分别作为chirplet变换的α，比如，如果剩下17条直线，那么就做17次chirplet变换
- 对所有scaled后的TFD应用Hough变换，找到一个峰值对应的直线
- 取TFD上以该直线为轴的带状区域，采用与单chirp信号估计类似的方法，使用最小二乘拟合直线的斜率

实现的代码如下：

先生成三个截断加噪的chirp信号，绘制时频图，调用`multi_chirp`函数估计参数，将参数打印出来：

```
chirp_signal = chirp(t, 16e6, T, 23.8e6, "linear", 90);
chirp_signal2 = chirp(t, 0.7e6, T, 20e6, "linear", 90);
chirp_signal3 = chirp(t, 15e6, T, 12e6, "linear", 90);

observed_signal1 = truncated_signal(chirp_signal, 1e-6, 4e-6, fs);
observed_signal2 = truncated_signal(chirp_signal2, 2.9e-6, 7.9e-6, fs);
observed_signal3 = truncated_signal(chirp_signal3, 3.8e-6, 9e-6, fs);

observed_signal = observed_signal1 + observed_signal2 + observed_signal3;
% 信噪比为20dB
observed_signal = awgn(observed_signal, 20);
fLevel = 512;
% 在这里作chirplet变换的目的是绘制时频图
[Spec, Freq, ~] = Chirplet_Transform(observed_signal, fLevel, 64, fs, 0);
figure;
imagesc(t,Freq,abs(Spec));
JET = colormap(jet);
colormap (JET);   % Set the current colormap Style
box on;
colorbar off;
% 调用multi_chirp估计参数
[start_time, init_freq, duration, bandwidth, detected_alpha] =
multi_chirp(observed_signal, fs, fLevel);
% 将估计出来的参数输出，注意由于没有给出信号的个数，估计出来的信号个数未必等于真实信号
for k = 1:length(start_time)
    fprintf('Start time of signal %d is %fe-6s\n', k, 1e6 * start_time(k));
    fprintf('Initial frequency of signal %d is %fe6Hz\n', k, 1e-6 * init_freq(k));
    fprintf('Duration of signal %d is %fe-6s\n', k, 1e6 * duration(k));
    fprintf('Bandwidth of signal %d is %fe6Hz\n', k, 1e-6 * bandwidth(k));
    fprintf('Alpha of signal %d is %fe11Hz/s\n', k, 1e-11 * detected_alpha(k));
    fprintf('\n');
end
```

`multi_chirp`主要涉及合并Hough变换后的结果，实现如下：

```
function [start_time, init_freq, duration, bandwidth, detected_alpha] =
multi_chirp(observed_signal, fs, fLevel)
    % 对信号首先做一次chirplet变换，此时alpha = 0
    [Spec, ~, ~] = Chirplet_Transform(observed_signal, fLevel, 64, fs, 0);
    spec_abs = abs(Spec);
    max_spec_abs = max(max(spec_abs));
    % 去除噪声
    spec_abs(spec_abs < 0.5 * max_spec_abs) = 0;
```

```
    % 转化成图像，以便能利用图像处理中的技术
    scaled_data = uint8(255 * mat2gray(spec_abs));
    % 调用extract_chirp_component提取出不超过30条直线
    [~, ~, ~, ~, alpha2] = extract_chirp_component(scaled_data, fs, fLevel, 30);
    % 初始化估计参数的矩阵
    start_time = zeros(1, length(alpha2));
    init_freq = zeros(1, length(alpha2));
    duration = zeros(1, length(alpha2));
    bandwidth = zeros(1, length(alpha2));
    detected_alpha = zeros(1, length(alpha2));
    % 有效估计参数的个数
    valid_cnt = 0;
    % 遍历所有直线
    for k = 1:length(alpha2)
        isvalid = 1;
        % 调整alpha值为直线的斜率，使得对应信号的能量更集中
        alpha = alpha2(k);
        % 每个信号的迭代次数
        iter_cnt = 0;
        alpha_pre = alpha;
        % 当直线的斜率收敛时迭代停止
        while (iter_cnt == 0 || abs(alpha - alpha_pre) / abs(alpha_pre) > 0.02)
            alpha_pre = alpha;
            % 与函数开头的操作类似，只不过这里的alpha由0改成了直线的斜率
            [Spec, ~, ~] = Chirplet_Transform(observed_signal, fLevel, 256, fs,
alpha);
            spec_abs = abs(Spec);
            max_spec_abs = max(max(spec_abs));
            spec_abs(spec_abs < 0.55 * max_spec_abs) = 0;
            scaled_data = uint8(255 * mat2gray(spec_abs));
            % 只提取一个直线
            [start_time_, init_freq_, duration_, bandwidth_, alpha] =
extract_chirp_component(scaled_data, fs, fLevel, 1);
            iter_cnt = iter_cnt + 1;
            % 超过10次就不再迭代
            if(iter_cnt > 10)
                break;
            end
        end
        % 去除重复的迭代结果
        for index = 1:valid_cnt
            % 信号持续时间过短，或者估计的参数的误差都在10%以内就要删去
            if (duration_ < 2e-6 ...
                    || (abs(init_freq(index)-init_freq_)/abs(init_freq(index)) < 0.1
...
                    && abs(duration(index)-duration_)/abs(duration(index)) < 0.1 ...
                    && abs(detected_alpha(index)-alpha)/abs(alpha) < 0.1))
                isvalid = 0;
                break;
            end

        end
```

```
        % 若新检测出来的参数和之前的没有重复，则更新矩阵
        if (isvalid == 1)
            valid_cnt = valid_cnt + 1;
            start_time(valid_cnt) = start_time_;
            init_freq(valid_cnt) = init_freq_;
            duration(valid_cnt) = duration_;
            bandwidth(valid_cnt) = bandwidth_;
            detected_alpha(valid_cnt) = alpha;
        end

    end
    % 把有效的估计量提取出来
    start_time = start_time(1:valid_cnt);
    init_freq = init_freq(1:valid_cnt);
    duration = duration(1:valid_cnt);
    bandwidth = bandwidth(1:valid_cnt);
    detected_alpha = detected_alpha(1:valid_cnt);
```

`extract_chirp_component`主要涉及对缩放去噪后的时频图做Hough变换，以及根据结果拟合直线斜率，合并预测结果。

```
function [start_time, init_freq, duration, bandwidth, alpha] =
extract_chirp_component(image, fs, fLevel, peak_num)
    bwImage = image;
    % 执行霍夫变换，霍夫变换是图像处理中寻找直线的一种经典算法
    [H, theta, rho] = hough(bwImage);
    % 在霍夫空间中找到峰值
    peaks = houghpeaks(H, peak_num); % 选择peak_num个峰值点
    % 检测直线
    lines = houghlines(bwImage, theta, rho, peaks);
    % 初始化估计参数矩阵，每一行的格式为(x1,y1,x2,y2)
    detected_line = zeros(length(lines), 4);
    % 找到的直线很可能有重复，需要去重
    % 没有重复的直线数目
    detected_line_num = 0;
    % 遍历每一条Hough变换给出的直线
    for index = 1:length(lines)
        is_detected = 1;
        point1 = lines(index).point1;
        point2 = lines(index).point2;
        % 遍历之前找到的没有重复的直线
        for index2 = 1:detected_line_num
            point3 = detected_line(index2, 1:2);
            point4 = detected_line(index2, 3:4);
            % 如果两条直线的端点距离比较近，说明重复
            if (sum(abs(point1 - point3)) < 40) && (sum(abs(point2 - point4)) < 40)
                % 如果新直线的跨度比旧直线大，将旧直线更新为新直线
                if(point1(1) < point3(1) && point2(1) > point4(1))
                    detected_line(index2, :) = [point1, point2];
                end
```

```
                is_detected = 0;
                break;
            end
        end
        if(is_detected == 0)
            continue;
        end
        detected_line_num = detected_line_num + 1;
        detected_line(detected_line_num, :) = [point1, point2];
    end
    detected_line = detected_line(1:detected_line_num, :);
    % 横坐标一格代表多长时间
    time_per_x = 1 / fs;
    % 纵坐标一格代表多少频率
    freq_per_y = fs / fLevel;
    start_time = detected_line(:, 1) * time_per_x;
    init_freq = zeros(detected_line_num, 1);
    alpha = zeros(detected_line_num, 1);
    % 接下来是拟合直线的斜率
    % 没有采用Hough变换给出的斜率，而是采用了与单chirp信号类似的最小二乘估计的方法
    for k1 = 1:detected_line_num
        % 把直线的左右端点找到
        left = detected_line(k1, 1);
        right = detected_line(k1, 3);
        x = left:right;
        y = zeros(1, length(x));
        % 扫描直线的每一个时刻
        for k2 = left:right
            % 对直线进行线性插值，确定纵坐标
            range = round((k2-left)/(right-left)*(detected_line(k1, 4)-
detected_line(k1, 2))+detected_line(k1, 2));
            % 取一个带状区域[low:top]，防止取到其他信号的幅频响应
            % 预留了5的裕度
            top = min(range+5, fLevel/2);
            low = max(1, range-5);
            signal_amp = image(low:top, k2);
            [~, max_index] = max(signal_amp);
            y(k2-left+1) = max_index + low - 1;
        end
        init_freq(k1) = y(1) * freq_per_y;
        % 拟合斜率
        p = polyfit(x, y, 1);
        alpha(k1) = p(1) * freq_per_y / time_per_x;

    end
    duration = (detected_line(:, 3) - detected_line(:, 1)) * time_per_x;
    bandwidth = abs(alpha .* duration);
    % 只取一个峰
    if (peak_num == 1)
        start_time = start_time(1);
        init_freq = init_freq(1);
        duration = duration(1);
```

```
        bandwidth = bandwidth(1);
        alpha = alpha(1);
    end
```

结果和讨论

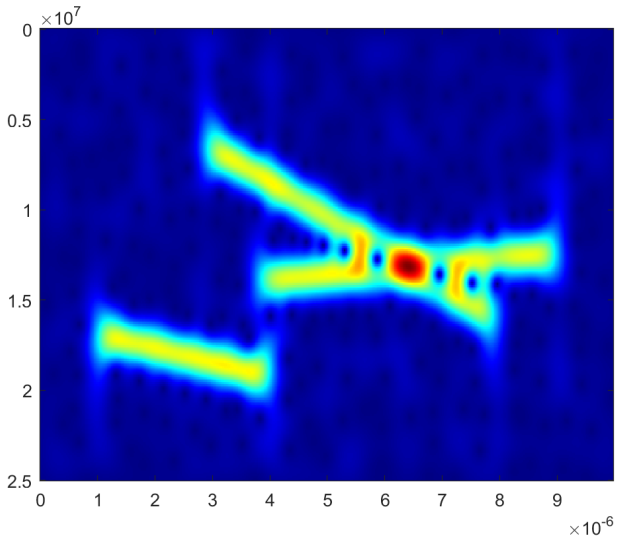
在给出结果之前，已经可以发现算法会存在若干问题：

- 无法限制信号的个数，估计出来的信号个数未必等于真实信号个数。所以，在算法中，**并未使用**到真实信号个数的条件。既然已经不能准确给出信号数量，那么RMSE的计算只能手动进行，这里为了方便，只是列出估计的参数，并没有实际计算RMSE。
- 算法核心是基于Hough变换的图像处理技术，这意味着如果两个chirp信号在时频图上有交叠，导致时频图所对应的像素发生畸变，会影响到检测效果；难以检测到持续时间较短的信号；如果两个信号在时频图上比较接近，也难以检测出来；检测出来的直线会有冗余和缺失。

下面的算法都是在**SNR=20dB**的情况下进行的

结果1

信号的时频图如下：



可以看到有两个信号有交叠，检测结果如下：

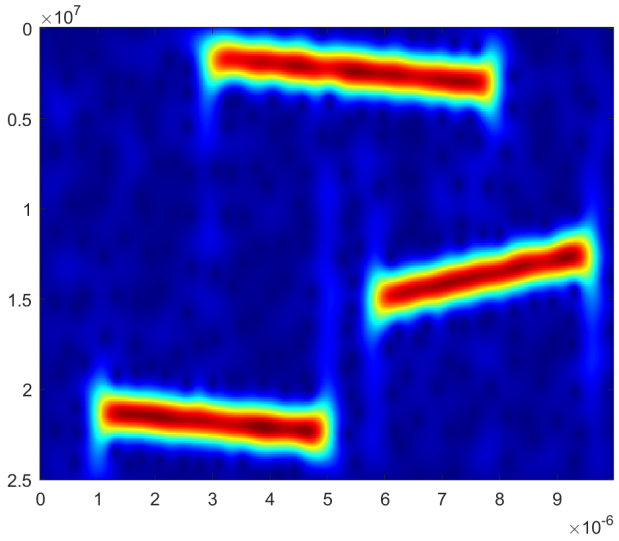
	start time(μs)	init freq(MHz)	duration(μs)	bandwidth(MHz)
信号1	1.00	16.78	3.00	2.34
估计1	1.06	16.99	2.92	2.25
误差	**6.00%**	**1.25%**	**2.67%**	**3.85%**
信号2	2.90	6.30	5.00	9.65
估计2	3.00	6.54	4.78	9.32
误差	**3.45%**	**3.81%**	**4.40%**	**3.42%**

	start time(μs)	init freq(MHz)	duration(μs)	bandwidth(MHz)
信号3	3.80	13.86	5.20	1.56
估计3	4.00	13.96	4.84	1.53
误差	**5.26%**	**0.72%**	**6.92%**	**1.92%**

恰好输出了3个检测结果，所有参数的估计误差最大约为7%

结果2

信号的时频图如下：



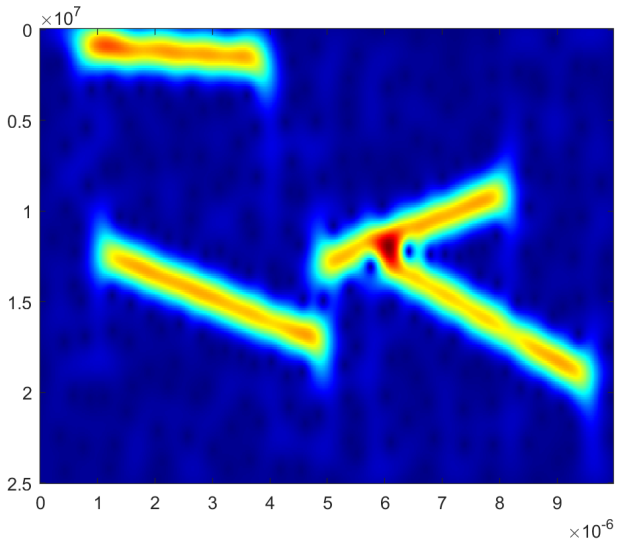
可以看到有上下两个信号的斜率类似，检测结果如下：

	start time(μs)	init freq(MHz)	duration(μs)	bandwidth(MHz)
信号1	1.00	21.10	4.00	1.20
估计1	NaN	NaN	NaN	NaN
误差	**NaN**	**NaN**	**NaN**	**NaN**
信号2	2.90	1.57	5.00	1.50
估计2	3.06	1.76	4.80	1.44
误差	**5.52%**	**12.10%**	**4.00%**	**4.00%**
信号3	5.80	14.94	3.80	2.66
估计3	5.92	14.94	3.62	2.49
误差	**2.07%**	**0.00%**	**4.74%**	**6.39%**

当chirp信号的频率变换率相近时，该检测算法会倾向于输出持续时间较长的信号（因为只选取了Hough变换的一个峰），造成漏检，这种现象与它们在持续时间上是否交叠没有关系。正确检测的信号的参数的误差多数在10%以内。

结果3

将信号数量增加到4，信号的时频图如下：



检测结果如下：

	start time(μs)	init freq(MHz)	duration(μs)	bandwidth(MHz)
信号1	1.00	12.10	4.00	5.20
估计1	1.12	12.30	3.86	5.01
误差	**12.00%**	**1.65%**	**3.50%**	**3.65%**
信号2	0.70	0.86	3.30	0.76
估计2	0.68	0.88	3.22	0.82
误差	**2.86%**	**2.33%**	**2.40%**	**7.89%**
信号3	5.80	12.81	3.80	7.41
估计3	5.44	11.33	4.14	8.00
误差	**6.21%**	**11.55%**	**8.95%**	**7.96%**
信号4	4.80	13.00	3.40	4.25
估计4	4.86	13.09	3.14	3.89
误差	**1.25%**	**0.69%**	**7.65%**	**8.47%**

算法恰好输出了4个信号，正确检测的信号的参数的误差多数在10%以内。

讨论

- 算法利用到了chirplet变换能将chirp信号的能量集中，以及Hough变换
- 本文所提出的算法并不能很好处理频率变化率类似的chirp信号，会漏检。
- 算法不能保证输出信号的个数与真实信号的个数一致