数字信号处理 大作业

2024秋季学期

姓名：陈彦旭

班级：无24

Task 1

介绍所使用的 SFFT 算法的原理与流程，画出稀疏信号的频谱真值与 SFFT 所得频谱的对比图。

选取“哈希映射法”（文献 Ref 1）。

符号说明： N 为信号长度， K 为信号频谱稀疏度， B 为分“筐”数， d 为循环时用到的参数， L 为循环估计的次数。

Step 1：频谱重排

实际中的稀疏信号大多频点较为集中，为了保证频域“分筐”的时候以极大概率将不同大值频点装入不同的筐中，需要对信号进行重排，将原本集中的大值频点分散开。利用 DFT 的循环位移性质和缩放性质，在时域对信号进行等价的操作。选取随机参数 σ, τ ，进行缩放+位移：

$$p[n] = x[((\sigma n + \tau))_N] \tag{1}$$

等价于频域有：

$$P[((\sigma k))_N] = X[k] W_N^{-\tau k} \tag{2}$$

Step 2：窗函数滤波

矩形窗函数时域为 sinc 序列，无限长，需要做截断操作，但是截断导致频域的通带和阻带变大。可以选取高斯窗函数与之时域相乘，具有平滑效果。

构造平坦的窗函数 $g[n]$ 及其频谱 $G[k]$ ，然后与信号时域相乘：

$$y[n] = p[n] \cdot g[n] \tag{3}$$

频域为两个信号频谱的循环卷积：

$$Y[k] = P[k] \circledast G[k] \tag{4}$$

由于 $P[k]$ 只在少数频点上有大值，可以看作若干不同幅度的 $\delta[k]$ 叠加，因此频谱 $Y[k]$ 近似为若干不同位置上的窗函数的叠加，且不同窗函数之间有重叠的概率较低。

Step 3：降采样

对信号 $y[n]$ 时域混叠：

$$z[n] = \sum_{j=0}^{N/B-1} y[n + Bj],\ n = 0, \ldots, N-1 \tag{5}$$

对 $z[n]$ 做 B 点 FFT ，结果实际上是对频谱 $Y[k]$ 降采样，降采样率为 N/B ：

$$Z[k] = Y[k\frac{N}{B}],\ k = 0, \ldots, B-1 \tag{6}$$

也就是将连续的 N/B 个频点装入一个筐中，一共得到 B 个筐。

Step 4：重构恢复

取出 $Z[k]$ 中幅度最大的 dK 个频点对应的坐标组成集合 J ，集合 J 是以 $z[k]$ 幅值为观测域得到的大值频点的坐标集合，每一个坐标都是一个筐，筐中装有 N/B 个频点。通过哈希函数关系可以将 J 映射回以 $X[k]$ 为观测域的大值频点的“原像”，得到集合 I ，因此集合 $I = \{k \in [0, N-1] | h_\sigma(k) \in J\}$ ，大小为 dKN/B 。

其中哈希函数的正确定义应该是（参考文献中并没有正确的表示出循环位移中“模”的关系，否则哈希函数将是单调增长的，不可能从 $[0, N-1]$ 映射到 $[0, B-1]$ ）：

$$h_\sigma(k) = \mathrm{mod}(\mathrm{round}(\mathrm{mod}(\sigma * k, N) * B/N), B),\ k = 0, \ldots, N-1 \tag{7}$$

```
hash_sigma = mod(round(mod(sigma * (0:N - 1), N) * B / N), B) + 1;
```

对于集合 J 中的每一个原像频点 k ，需要除以窗函数滤波器的增益，估计其原本的幅度值。

$$\hat{X}[k] = Z[h_\sigma(k)]W_N^{\tau k}/G[o_\sigma(k)] \tag{8}$$

所使用的偏移量函数 $o_\sigma(k)$ 将 $[0, N-1]$ 映射到 $[-N/2B, N/2B-1]$ ，如果加上模 N 的话，就是映射到 $[0, N/2B-1], [N-N/2B, N-1]$ ，这样能够直接作为数组的索引：

$$o_\sigma(k) = \mathrm{mod}(\sigma k - \mathrm{round}(\sigma k B/N) * N/B,\ N) \tag{9}$$

```
offset_sigma = mod(sigma * (0:N - 1) - round(sigma * (0:N - 1) * B / N) * N / B, N) + 1;
```

经过上述一次操作之后得到集合 I 和其对应坐标估计的幅度值。循环操作 L 次后，所有的集合 I 组成大小为 $L \times dKN/B$ 的矩阵，找出其中出现次数超过 $L/2$ 的频点坐标（或者出现次数最多的 K 个频点的坐标），作为最终所求的原始大值频点的坐标，将其在循环中出现的所有幅度值的中位数作为最终所估计的频率值。

基于上述算法，编写 MATLAB 代码如下：

```
% 稀疏傅里叶变换SFT

function X_hat = sft(x_n, N, K, B, L, d, W)
    % 参数：
    % x_n: 时域信号，大小[1,N]
    % N: 信号长度
    % K: 频谱稀疏度
    % B: 分筐的个数
    % L: 循环次数
    % d: 定位循环用到的参数
    % W: 窗函数的截断长度
    % 返回：
    % X_hat: 估计的频域信号，大小[1,N]

```

```
    if mod(N, B) ~= 0
        error('error! B should divide N.');
    end

    % 准备平坦窗函数g[n]
    rec_win = zeros(1, N);
    rec_win(1:W) = sinc(((0:W - 1) - W / 2) / B) / B;
    gauss_win = zeros(1, N);
    std_dev = B * log2(N); % 高斯窗的标准差
    gauss_win(1:W) = exp(- ((0:W - 1) - W / 2) .^ 2 / (2 * std_dev ^ 2));
    g_n = rec_win .* gauss_win;
    g_n = g_n ./ max(abs(g_n));
    G_k = fft(g_n);

    X_r = zeros(L, N); % 存储每次循环的估计值
    X_hat = zeros(1, N);

    for loop_cnt = 1:L
        % 生成随机参数sigma,tau
        % sigma为奇数，与N互素
        sigma = 2 * randi([0, N / 2 - 1]) + 1;
        tau = randi([1, N]);

        % 频谱随机重排
        p_n = x_n(mod(sigma * (0:N - 1) + tau, N) + 1); % 缩放平移
        y_n = p_n .* g_n; % 与窗函数时域相乘

        % 降采样FFT
        z_n = sum(reshape(y_n, B, []), 2);
        z_n = transpose(z_n);
        Z_k = fft(z_n, B);

        % 按幅度排序取出最大的d*K个
        [~, J] = sort(abs(Z_k), 'descend');
        J = J(1:d * K);

        % hash_sigma:[1,N]-->[1,B]
        hash_sigma = mod(round(mod(sigma * (0:N - 1), N) * B / N), B) + 1;
        % offset_sigma:[1,N]-->[1,N/2B],[N-N/2B-1,N]
        offset_sigma = mod(sigma * (0:N - 1) - round(sigma * (0:N - 1) * B / N) * N / B, N) + 1;

        % J的原像坐标集合I_r
        I_r = find(ismember(hash_sigma, J));
        % 估计幅度
        X_r(loop_cnt, I_r) = Z_k(hash_sigma(I_r)) .* exp(1j * 2 * pi * tau * (I_r - 1) / N) ./ G_k(offset_sigma(I_r));

    end

    % 找出出现次数大于L/2的频点坐标
    % nonzero_freq = find(sum(X_r ~= 0) > L / 2);

    % 找出出现次数最多的K个频点坐标
    [~, nonzero_freq] = sort(sum(X_r ~= 0), 'descend');
    nonzero_freq = nonzero_freq(1:K);
```

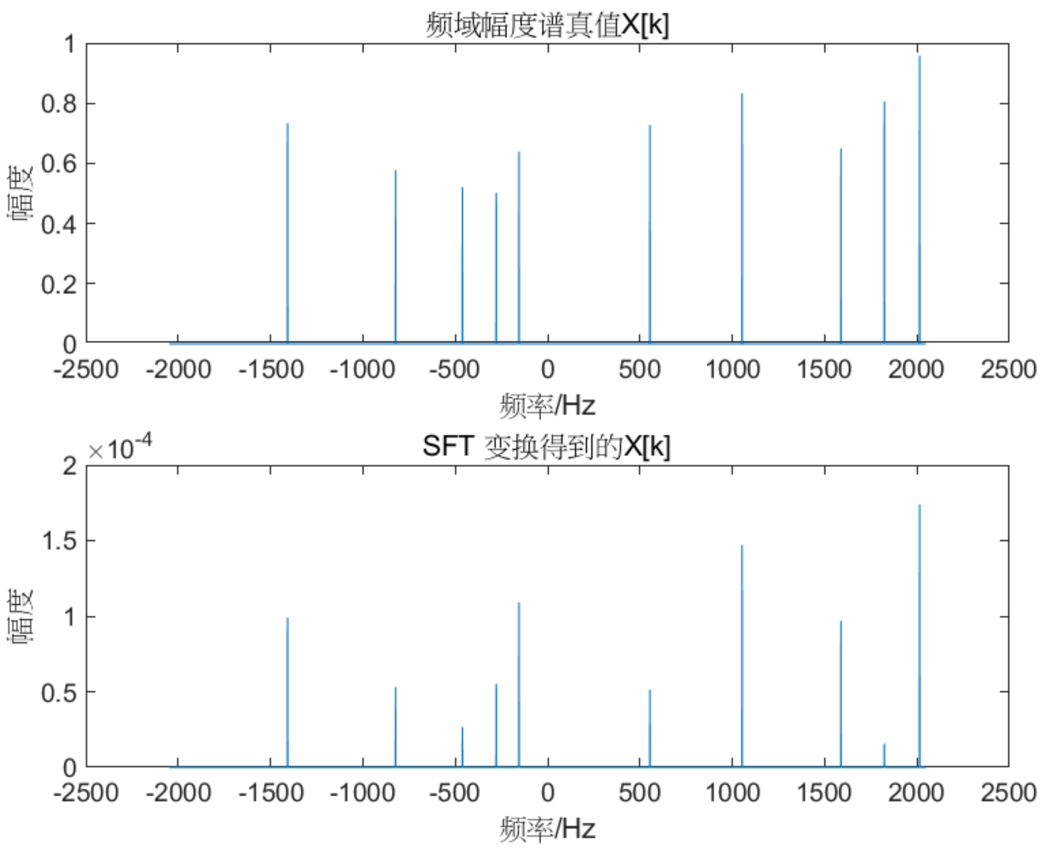
```

    for freq = nonzero_freq
        re = median(real(X_r(:, freq)));
        im = median(imag(X_r(:, freq)));
        X_hat(freq) = re + 1j * im;
    end

end
```

仿真验证

设定实验参数为：$N = 4096, K = 10, d = 4$。运行结果如下：



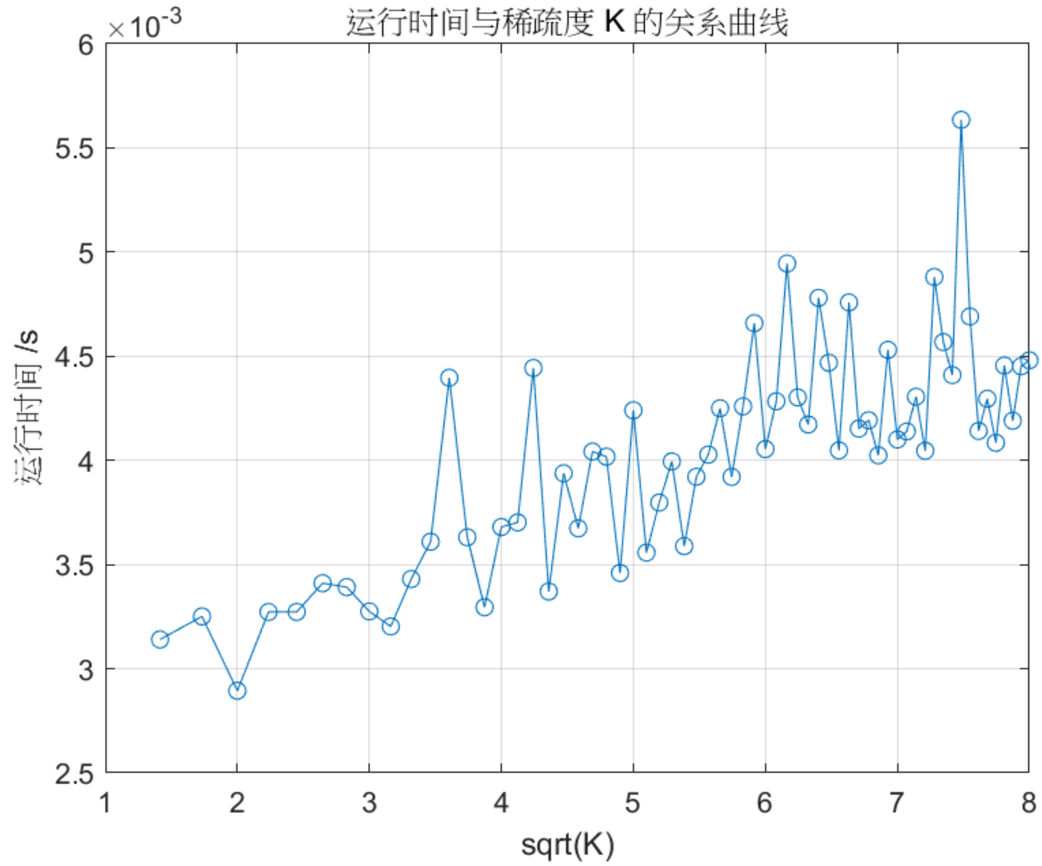
可见 SFT 算法较为准确地找出了所有稀疏的频点。幅值较小，相对大小不一致，可能是因为丢弃了大部分原始序列中的点，降采样操作也会造成能量损失。

Task 2

以 $O(f(n,k))$ 的形式给出所使用的 SFFT 算法运行的时间复杂度。分别画出 SFFT 运行时间随信号长度 N 以及信号频谱稀疏度 k 变化的关系图，利用关系图验证你的推测。

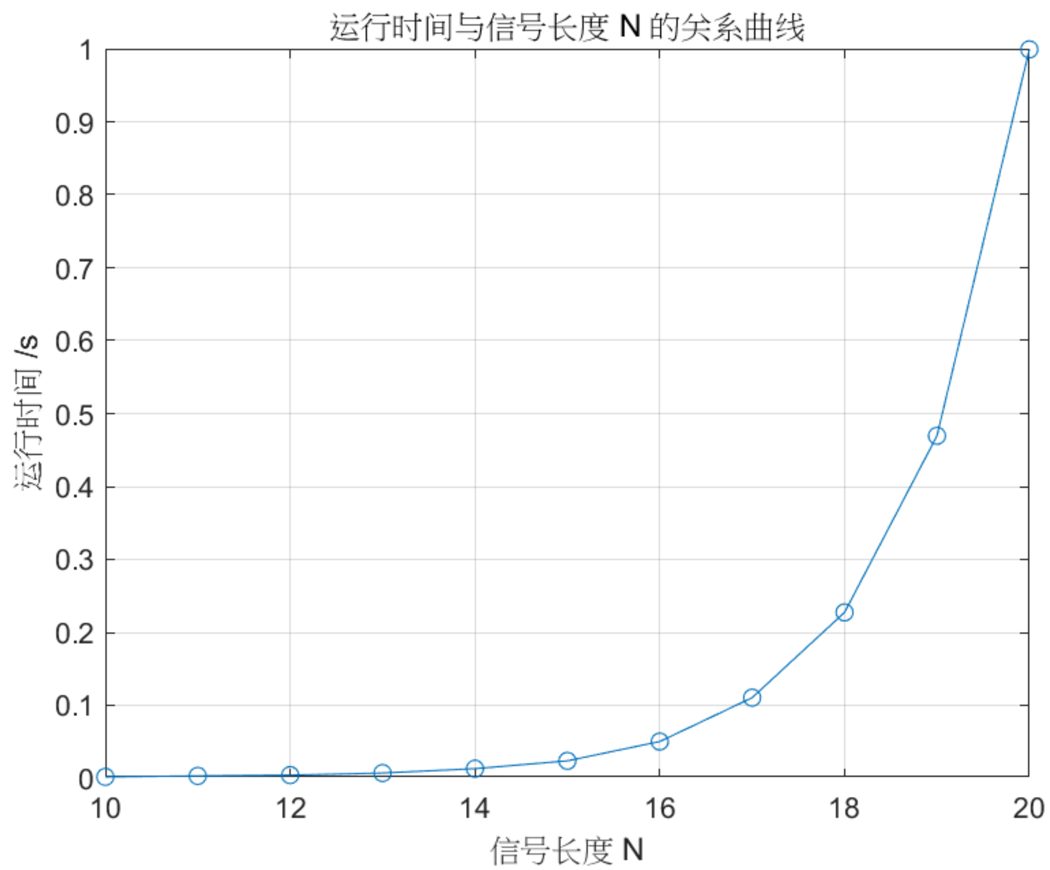
根据参考文献 Ref 2 的推导，SFFT 算法的复杂度约为 $O(\log_2 n\sqrt{nk\log_2 n})$， n 为信号长度且为 2 的次幂， k 为频域的稀疏度。

控制信号长度 n 不变时，运行时间约为正比于 $\sqrt{k}$， 仿真得到：

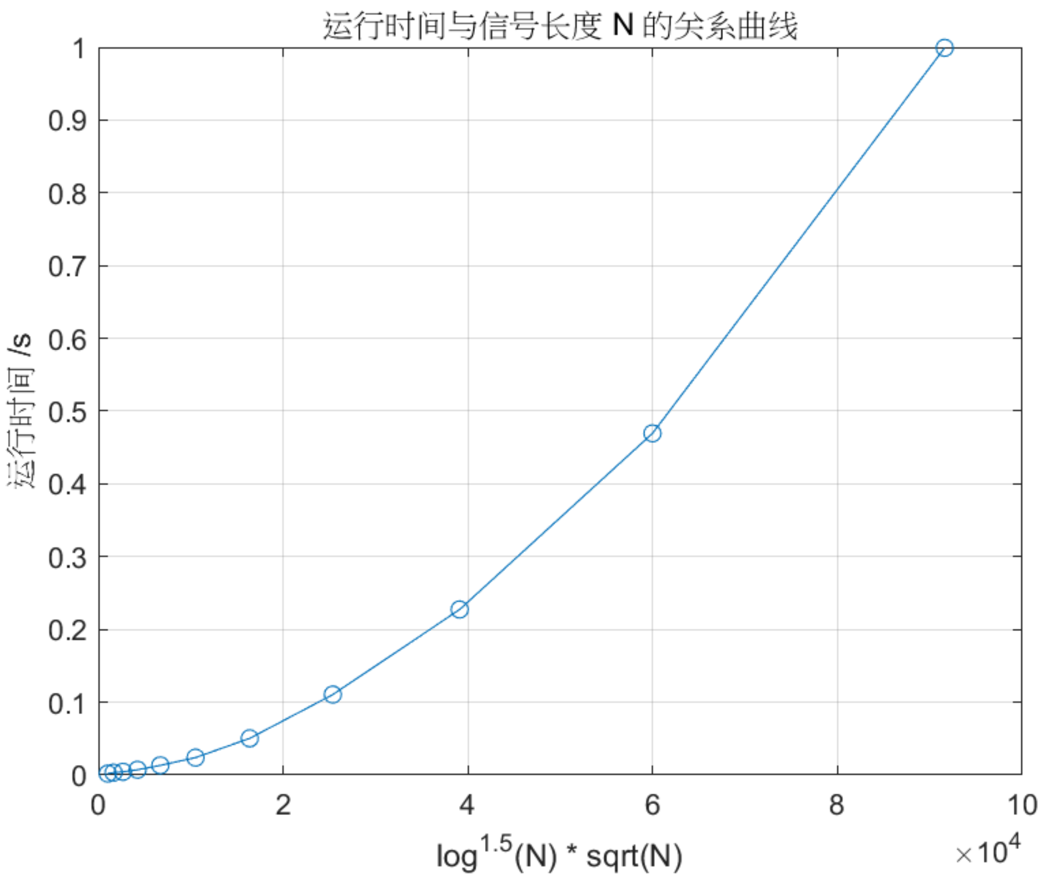


上图中横坐标为 $\sqrt{k}$ 。可见随着稀疏度变化，算法的运行时间波动很大，但总体上还是随着稀疏度的增长而增长的。

控制稀疏度 k 不变时，运行时间约为正比于 $\log_2 n\sqrt{n\log_2 n}$ ，若令 $x=\log_2 n$ ，则复杂度约为 $2^{\frac{x}{2}}x^{\frac{3}{2}}$ ，比指数增长略快。以 x 为横坐标，得到图像：



若以 $\log_2 n\sqrt{n\log_2 n}$ 为横坐标，得到：



比线性增长略快，但是几乎在 $O(\log_2 n\sqrt{n\log_2 n})$ 范围之内。

Task 3

介绍所选择压缩感知算法的流程、原理。探究至少两种因素对 SFFT 算法与压缩感知算法性能的影响，分别画出使用两种算法得到的信号频谱估计的 l_1 误差与所选择的因素之间的关系图。对比使用两种算法得到的关系图，并尝试对所得结果加以解释。

奈奎斯特采样定理告诉我们，要从采样之后的数字信号中无失真恢复出原始信号，采样频率必须大于原始信号中最高频率的两倍。但是如果信号是稀疏的，那么可以由远低于采样定理要求的频率采样和恢复。

如果使用随机亚采样，频域不再是以固定周期延拓，而是出现大量近似均匀分布的干扰值，这些干扰值是由原始信号中各个频点的非零值泄漏导致的，可以使用特别的追踪算法恢复出原信号。压缩感知的前提：信号稀疏，使用随机亚采样。

$$y=\varPhi\varPsi s=\varPhi x=\varTheta s \tag{10}$$

其中 x 为原信号，y 为采样结果，$\varPhi$ 为观测矩阵，对应随机亚采样方式，$\varPsi$ 为变换矩阵（对应 IDFT），s 为信号的稀疏表示（对应频谱），$\varTheta=\varPhi\varPsi$ 为感知矩阵。观测矩阵应满足约束等距条件（RIP），RIP 的等价条件是观测矩阵和稀疏表示基不相关。独立同分布的高斯随机测量矩阵可以作为普适的压缩感知测量矩阵。

重建过程也就是求解欠定方程 $y=\varTheta s$ 的问题，使得 0-范数（非零元素个数）最小化，即 $\min\|\hat{x}\|_0$ 。

正交匹配追踪算法（OMP）是一种贪婪算法，通过迭代中每次选择与当前残差最相关的字典元素，并通过正交投影逐步逼近真实信号。算法过程如下：

Step 1

输入观测矩阵 $A \in \mathbb{R}^{M\times N}$ ，观测信号 $y \in \mathbb{R}^{M\times 1}$ ，稀疏度 $sparsity$ 。由于我们希望得到的是信号的频谱而不是信号本身，因此这里输入的是观测矩阵实际上是高斯随机测量矩阵和 IDFT 矩阵的乘积。

Step 2

初始化残差为 $r_0 = y$ ，索引集 $S_0 = \varnothing$ ，复原信号 $\hat{x}_0 = \mathbf{0}_{N\times 1}$ 。

Step 3

迭代过程，对于每一次迭代 k ：

1. 首先选择与当前残差最相关的字典元素，计算：

$$i_k = \arg\max_{i \notin S_{k-1}} |A_i^T r_{k-1}| \tag{11}$$

2. 将索引 i_k 加入到支持集中：

$$S_k = S_{k-1} \cup \{i_k\} \tag{12}$$

3. 在当前的索引集上，通过最小二乘法求解向量 x_k ：

$$\begin{aligned} x_k(i \in S_k) &= \arg\min_x \|y - A_{S_k} x\|_2 \\ x_k(i \notin S_k) &= 0 \end{aligned} \tag{13}$$

4. 更新残差：

$$r_k = y - A_{S_k} x_k \tag{14}$$

重复 Step3 进行迭代，迭代终止条件：残差的范数小于预设阈值或者达到最大迭代次数（或者达到需要取出的频点数目）。

基于上述算法编写，编写 MATLAB 代码如下：

```
function x_hat = omp(y, measure_mtx, trans_mtx, sparsity)
    % 参数:
    % y: 观测信号，大小[M,1]
    % measure_mtx: 测量矩阵，大小[M,N]
    % trans_mtx: 变换矩阵，大小[N,N]
    % sparsity: 稀疏度
    % 返回:
    % x_hat: 估计信号，大小[N,1]

    sensing_mtx = measure_mtx * trans_mtx; % 感知矩阵
    [~, N] = size(sensing_mtx);
    x_hat = zeros(N, 1);
    residual = y; % 初始化残差r=y
    index = []; % 初始化索引集合S
    threshold = 1e-6; % 设置阈值

    for cnt = 1:sparsity
        correlations = abs(sensing_mtx.' * residual);
        correlations(index) = -inf;
        [~, i_k] = max(correlations);
        index = [index, i_k];
        x_hat(index) = pinv(sensing_mtx(:, index)) * y;
```

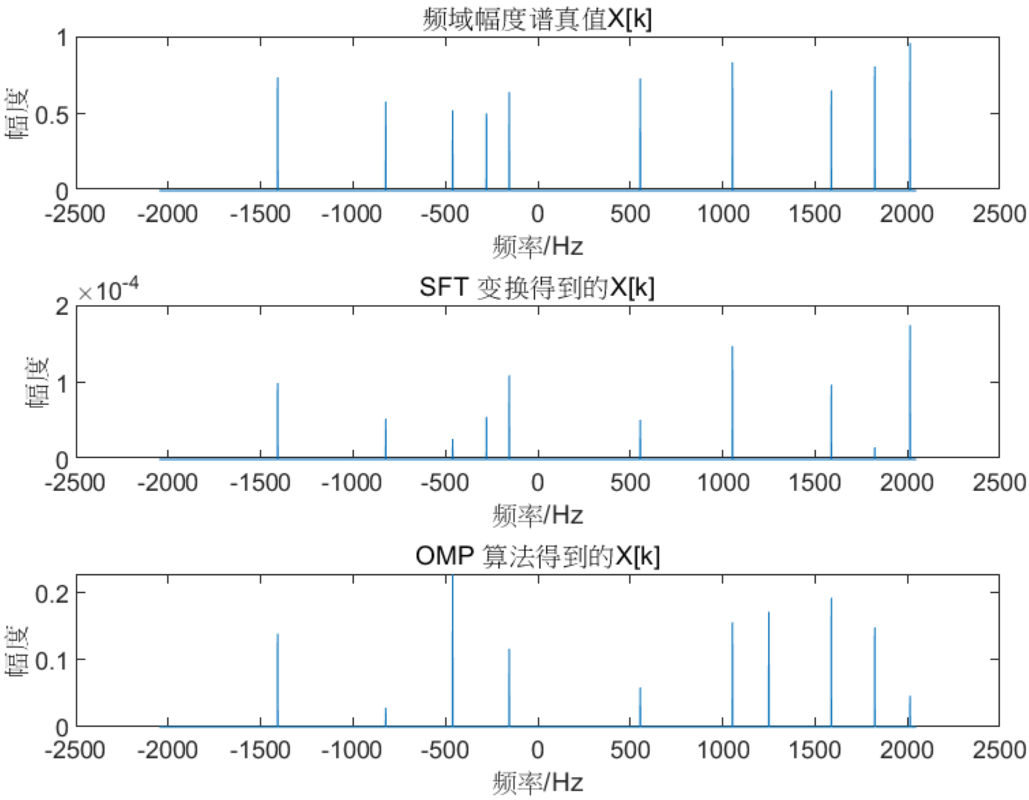
```
        residual = y - sensing_mtx * x_hat;

        if norm(residual) < threshold
            break;
        end

    end

end
```

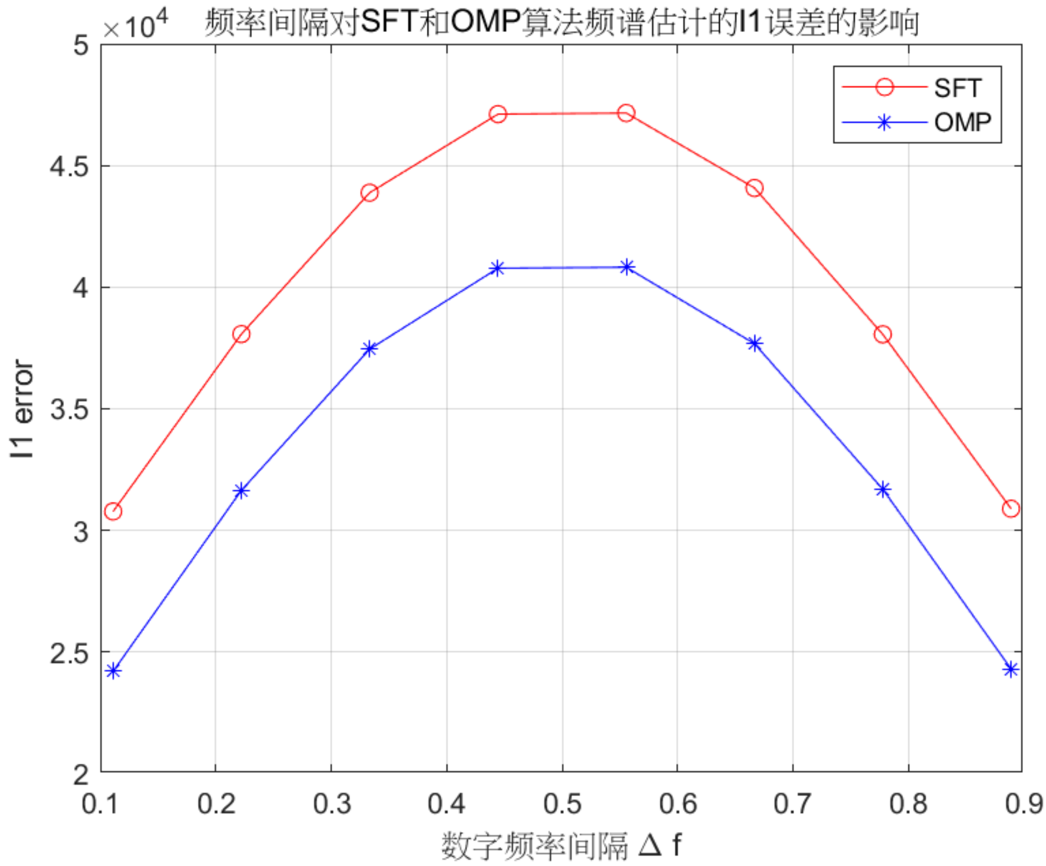
生成频域稀疏信号，分别使用 SFT 算法和 OMP 算法，得到信号的频谱，对比如下：



与 SFT 算法相比，OMP 算法遗漏了原频谱中的一个频点，错误地找出了一个原本不存在的频点，但整体效果还是不错的。

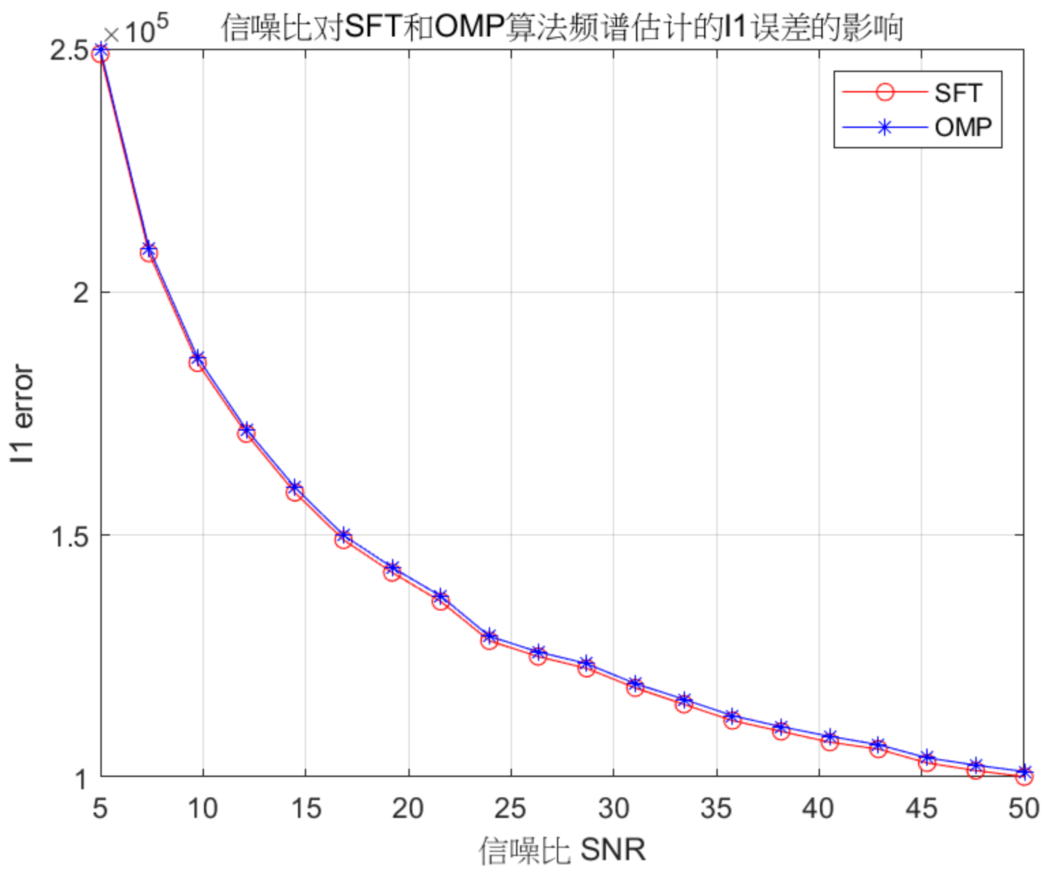
生成稀疏度为 2 的信号，并加上复高斯噪声。

其他因素不变时，两个频率分量的频率间隔对 SFT 和 OMP 算法频谱估计的 l_1 误差的影响：



在上图中，频率间隔为 0.5 左右时误差达到最大，增大或减小频率间隔时误差都将减小。我设置的第一个频率为 0.5 为高频，频率间隔为 0.5 意味着第二个数字频率在 0 频附近。频率间隔接近 0 或 1 意味着第二个数字频率也在 ± 0.5 的高频附近，此时误差较小。原因可能是两个信号频率较为接近时，能量能量更为集中，抗噪声干扰能力强，因此算法能够更准确地排除干扰并找出原有频率分量。

其他因素不变时，信噪比对 SFT 和 OMP 算法频谱估计的 l_1 误差的影响：



结果是在意料之中的，信噪比越大，噪声功率越小，造成额外干扰的频率分量的幅度越小，能够让 SFT 算法和 OMP 算法更精准地找出原本稀疏的频率分量。

文件清单

```
.
|-- exp1.m        实验1：实现SFT算法
|-- exp2_1.m      实验2-1：SFT算法运行时间与K的关系
|-- exp2_2.m      实验2-2：SFT算法运行时间与N的关系
|-- exp3_1.m      实验3-1：实现OMP算法，与SFT算法对比
|-- exp3_2.m      实验3-2：探究频率间隔对SFT和OMP算法频谱估计的l1误差的影响
|-- exp3_3.m      实验3-3：探究信噪比对SFT和OMP算法频谱估计的l1误差的影响
|-- omp.m         压缩感知OMP算法
|-- report.pdf    实验报告
`-- sft.m         稀疏傅里叶变换SFT
```

参考文献：

文档中 Ref1, Ref2, Ref5。

[1]李非凡.稀疏傅里叶变换算法的硬件实现[D].东南大学,2023.DOI:10.27014/d.cnki.gdnau.2023.000766.