*[1] S. Prager, M. S. Haynes, and M. Moghaddam, “Wireless subnanosecond RF synchronization for distributed ultrawideband software-defined radar networks,” IEEE Trans. Microw. Theory Techn., vol. 68, no. 11,pp. 4787–4804, Nov. 2020.*

*[2] R. Moddemeijer, “On the determination of the position of extrema of sampled correlators,” IEEE Trans. Signal Process., vol. 39, no. 1,pp. 216–219, Jan. 1991.*

*[3] S. R. Mghabghab and J. A. Nanzer, "Microwave ranging via least-squares estimation of spectrally sparse signals in software-defined radio", IEEE Microw. Wireless Compon. Lett., vol. 32, no. 2, pp. 161-164, Feb. 2022.*

通过低采样率实现皮秒级别（ps）同步是一个非常挑战性的问题，因为皮秒级别的时间分辨率意味着极高的精度，而低采样率通常会限制时间分辨率。但有一些方法可以克服这一挑战，利用特殊的信号处理技巧，尽管采样率低，依然能获得极高的同步精度。

以下是几种常见的实现方法：

**1. 利用插值算法提高分辨率**

尽管采样率较低，但可以通过插值算法增加信号的“有效采样点”，从而提升时间分辨率。插值算法可以从有限的采样数据中重建出更高分辨率的信号。

常用的插值方法包括：

• **线性插值**：通过连接相邻的采样点来估计信号在更细时间尺度上的值。

• **样条插值（Spline Interpolation）**：通过拟合样条曲线来进行插值，能够更平滑地恢复信号的细节，适用于复杂波形。

• **高阶插值**：比如 **拉格朗日插值** 或 **多项式插值**，这些方法能够在给定的采样点上进行更精细的信号重建，从而提供更高的时间分辨率。

通过这些插值方法，虽然实际采样率很低，但可以在时间域上“推测”出信号的更多信息，从而有效提高时间同步精度。

**2. 利用时间延迟估计（Time Delay Estimation，TDE）**

**时间延迟估计 (TDE)** 技术可以在低采样率下精确地估算信号的传播时延，尤其是在存在噪声的情况下。通过交叉相关等方法，可以根据信号的特征（如相位或幅度）进行时延估计，而不依赖于高采样率。

• **交叉相关法**：计算两个信号的交叉相关性，找到两信号最相似的时刻，从而估算信号之间的时延。

• **子采样方法**：通过估算信号的交叉相关最大值的子采样位置，可以精确估算时延。常见的方法有 **谱估计** 或 **傅里叶变换** 等。这些方法能够在较低采样率下，依然做到皮秒级别的同步精度。

**3. 相位锁定环（PLL）和相位同步技术**

相位锁定环（PLL）是通过自动调节一个振荡器的频率，使其与输入信号的频率保持同步。利用PLL可以有效提高系统的同步精度，即使采样率较低。

在高精度同步应用中，PLL 可以提供极为细致的时钟同步，通过增加反馈回路、提高调节精度，能够在低采样率下实现皮秒级别的同步。

**4. 超高精度时钟同步与时间标定**

低采样率下的皮秒级别同步还可以通过精确的硬件时钟同步和时间标定来实现。对于某些应用，硬件时钟可以与外部同步信号（如 GPS 信号、原子钟等）进行精密对接，即使在低采样率的情况下，也能保证系统在较长时间内的高精度同步。

• **利用外部同步源**：例如，使用 GPS 或原子钟等高精度时钟同步源来为系统提供精确的时间标定，通过硬件时钟的高精度同步，弥补低采样率带来的分辨率限制。

• **时间同步协议**：例如，**PTP（Precision Time Protocol）** 等高精度时间同步协议也可以在较低采样率的条件下，利用协议中的补偿机制和精密算法进行高精度同步。

**5. 信号调制与解调技术**

某些调制和解调技术，如 **频率调制（FM）** 或 **相位调制（PM）**，可以在较低的采样率下实现高精度同步。这些技术通过对信号的频率或相位变化进行编码，能够有效地传递和估计时间信息。

• **相位差估计**：通过对两信号相位差的估计，可以在低采样率下高精度地测量时延。比如通过计算不同频率成分之间的相位差来提高时延估计的精度。

**6. 采样时间密度增强（Sub-sampling）**

即使采样率低，也可以通过 **超采样技术**（例如对信号进行多次采样，在信号的特定时刻或特定特征上进行重复采样）来增强信号的时间密度，从而提升精度。这种技术可以通过多次对信号的高精度特征进行采样，从而模拟更高采样率带来的效果。

例如，在信号的特定边沿或高频成分上进行超采样，能够增强时间精度，尽管采样率保持较低。

**7. 信号编码与解码（Modulation and Demodulation）**

在某些情况下，特定的编码方案（如 **正交频分复用（OFDM）** 或 **码分多址（CDMA）**）可以通过高效的解码方法实现高精度同步。这些编码方式通过嵌入时间信息或时钟同步信号，即使在低采样率下，也能通过解码和去噪算法提取出准确的时钟同步信息。

**总结**

实现 **皮秒级别同步** 的关键在于使用先进的信号处理技术和优化算法来克服低采样率的限制。尽管采样率较低，仍可以通过：

• 插值算法

• 时间延迟估计

• 相位锁定环

• 外部同步源和高精度时钟

• 采样密度增强等方法

通过这些技术，可以有效提升同步精度，甚至达到皮秒级别的时钟同步。在实际应用中，结合硬件优化和信号处理算法，能够在低采样率下实现超高精度的时间同步。

**Peak Interpolation Techniques（峰值插值技术）**

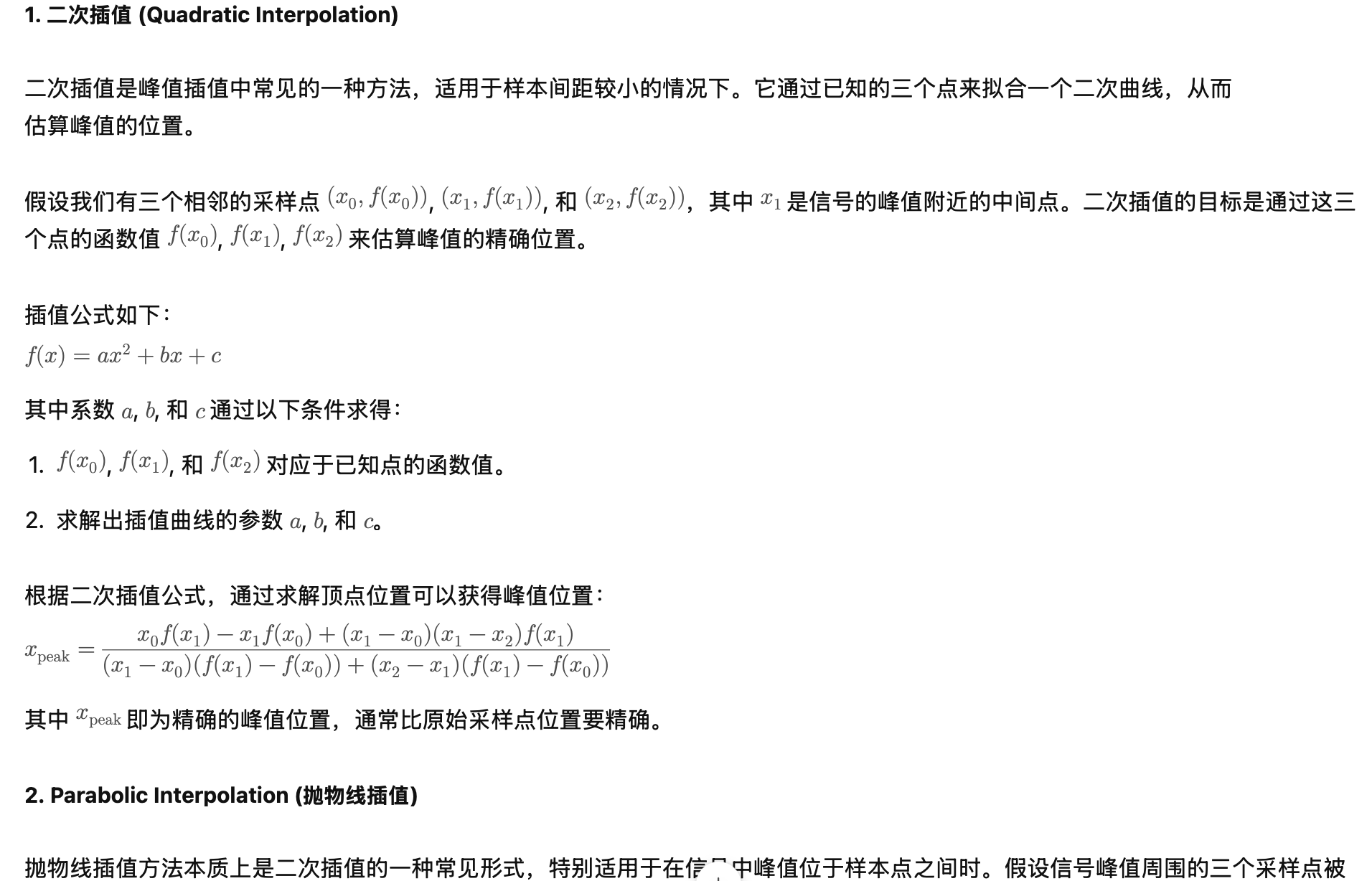
**Peak interpolation techniques** 是一种用于估计信号峰值位置或峰值特征的技术，尤其在信号处理中应用广泛。通过对信号的峰值进行插值，我们可以实现比原始采样间隔更高的时间或频率分辨率，尤其在采样点有限的情况下。这些技术通常用于 **时延估计**、**频率估计** 和 **相位估计** 等应用场景。

**数学原理**

在信号处理中，插值的目标是通过已知的采样点估算出更高精度的位置。**峰值插值** 主要应用于估计信号的极值点（峰值点）位置，常见的方法包括 **二次插值** 和 **拉格朗日插值** 等。我们通常通过插值来获取峰值的精确位置，超越原始采样间隔的分辨率。

以下是几种常见的峰值插值方法及其数学公式：

**1. 二次插值 (Quadratic Interpolation)**



二次插值是峰值插值中常见的一种方法，适用于样本间距较小的情况下。它通过已知的三个点来拟合一个二次曲线，从而估算峰值的位置。



假设我们有三个相邻的采样点 , , 和 ，其中 是信号的峰值附近的中间点。二次插值的目标是通过这三个点的函数值 , , 来估算峰值的精确位置。

插值公式如下：

其中系数 , , 和 通过以下条件求得：

1. , , 和 对应于已知点的函数值。

2. 求解出插值曲线的参数 , , 和 。

根据二次插值公式，通过求解顶点位置可以获得峰值位置：

其中 即为精确的峰值位置，通常比原始采样点位置要精确。

**2. Parabolic Interpolation (抛物线插值)**

抛物线插值方法本质上是二次插值的一种常见形式，特别适用于在信号中峰值位于样本点之间时。假设信号峰值周围的三个采样点被称为 , , 和 ，其中 是中间点。

使用以下公式可以估算信号的峰值位置：

其中：

• 是估算的峰值位置。

• , , 和 是已知点的函数值。

**3. Gaussian Fitting (高斯拟合)**

在某些情况下，信号的峰值可能呈现近似高斯形状。此时，我们可以使用 **高斯拟合** 来估算峰值。高斯函数的形式为：

其中：

• 是振幅。

• 是峰值位置（我们要估算的）。

• 是标准差，表示峰的宽度。

通过对已知的信号点进行拟合，我们可以估算出峰值的位置 ，进而得到精确的峰值估计。

**4. Spline Interpolation (样条插值)**

对于复杂信号或者在连续域中插值时，常用样条插值（例如立方样条）来平滑地估算信号波形的峰值。

**立方样条插值** 通过连接一系列多项式（每个区间内为立方多项式）来生成一个平滑的曲线，能够有效避免传统插值方法中出现的过度振荡问题。

**5. Lagrange Interpolation (拉格朗日插值)**

拉格朗日插值用于通过已知的采样点生成插值多项式来估算峰值位置。假设有 个点 , , , ，我们可以构造一个拉格朗日多项式：

\[

L(x) = \sum\_{i=0}^{n} f(x\_i) \prod\_{\substack{0 \leq j \leq n \\ j \neq i}} \frac{x - x\_j}{x\_i - x\_j}

\]

然后通过求解这个多项式的极值来得到峰值位置。

**物理意义**

峰值插值技术的核心物理意义是 **提高测量精度**。实际应用中，采样频率和采样间隔往往受硬件限制，但通过插值技术，可以利用有限的采样点估算出更精确的特征点位置，如信号峰值。峰值的位置通常代表信号的最大能量或最强反应点，特别在以下领域非常重要：

• **信号检测**：精确估算信号的峰值位置可以有效提高信号检测的灵敏度，特别是在噪声环境下。

• **时延估计**：在无线通信和定位系统中，峰值位置的精确估算有助于计算信号传播的时延，从而进行时间同步。

• **频率和相位估计**：对于频率偏移或相位误差的估算，通过峰值插值可以提高精度，特别是在频谱分析中。

• **目标定位**：在雷达、声纳和成像系统中，通过准确估算目标的峰值，可以提高目标检测与定位精度。

**总结**

峰值插值技术是一种通过对已知采样点的插值来精确估算信号极值位置的方法。常见的插值方法有二次插值、抛物线插值、高斯拟合、样条插值和拉格朗日插值等。通过这些插值技术，我们可以在低采样率下获得更高精度的信号特征位置，这对于时延估计、频率估计、目标定位等高精度信号处理任务至关重要。

**Sinc Nonlinear Least Squares (Sinc-NLS) 算法**

**Sinc Nonlinear Least Squares (Sinc-NLS)** 是一种用于峰值检测和时延估计的算法。它利用信号的 **Sinc 函数特性**（即理想低通滤波器的频率响应函数）来拟合信号中的峰值位置。该算法属于非线性最小二乘法（Nonlinear Least Squares, NLS）的一种应用，广泛用于高精度的峰值定位问题，特别是在噪声环境下进行信号峰值的精确估计时。

**Sinc 函数与其在信号处理中的意义**

**Sinc 函数**（Sin(x)/x）是傅里叶变换中非常重要的一种函数，广泛应用于数字信号处理，尤其是在频域分析中。它通常与理想低通滤波器的频率响应密切相关，用来描述有限带宽信号的理想表示。Sinc 函数具有如下形式：

在峰值检测问题中，Sinc 函数的特点可以用于精确描述信号的波形，特别是在周期性信号和带限信号的情况中，信号的形态可能接近于 Sinc 函数。

**Sinc Nonlinear Least Squares 算法的数学公式**

Sinc-NLS 算法的核心思想是将信号波形拟合为 Sinc 函数模型，使用非线性最小二乘法优化来估计信号的峰值位置、振幅和其他特征。具体的数学步骤如下：

**1. Sinc 函数模型**

假设我们有一个信号 ，该信号近似为一个 Sinc 函数的加权和。为了通过 Sinc 函数对信号进行建模，我们假设信号的形式为：

其中：

• 是信号的振幅。

• 是信号的峰值位置（我们需要估算的）。

• 是与信号宽度相关的参数（如时间带宽积）。

• 是噪声或其他干扰。

**2. 最小二乘法目标函数**

为了估算 , , 和 等参数，我们使用最小二乘法进行优化。目标函数为信号与模型之间的误差平方和：

其中：

• 是在时刻 采样得到的信号值。

• , , 和 是需要优化的参数。

**3. 非线性最小二乘法求解**

为了得到 , , 和 的最优值，我们采用 **非线性最小二乘法**。该方法通过迭代更新参数，最小化误差函数 ，最终收敛到一个局部最优解。

假设我们的初始估计值为 , , 和 *，然后通过* ***高斯-牛顿法*** *或* ***Levenberg-Marquardt 法*** *进行迭代优化，更新参数：*

其中：

• 是雅可比矩阵，它表示目标函数对参数的梯度。

• 是目标函数的残差向量。

• 是待估参数。

通过这样的迭代，算法能够不断优化参数估计，直到目标函数的值收敛到一个最小值，从而得到最精确的峰值位置。

**4. 峰值检测**

通过对信号进行 Sinc 拟合，算法可以精确估算出信号的峰值位置 。同时，Sinc-NLS 方法也可以提供峰值的振幅 和信号的宽度参数 ，这些参数对于进一步的信号分析非常重要。

**物理意义**

Sinc-NLS 算法的物理意义可以从以下几个方面来理解：

1. **信号建模**：

Sinc 函数是频率响应为理想低通滤波器的信号形态。许多通信系统和传感器系统中的信号（特别是经过低通滤波或带限信号）在时间域上会呈现类似于 Sinc 函数的形态。因此，Sinc 函数是一个自然的选择，能够有效地近似这些信号的形态。

2. **峰值精度**：

Sinc 函数通过拟合信号的精确形态（包括其宽度和位置），可以显著提高峰值定位的精度。与传统的基于采样点的简单峰值检测方法相比，Sinc-NLS 能够在较少的采样点下提供更高的时间分辨率，达到皮秒级别的精度。

3. **噪声鲁棒性**：

由于 Sinc 函数能够很好地描述信号的主波形特征，Sinc-NLS 在噪声环境下也表现得比较鲁棒。相比于简单的峰值检测方法，它能够有效抑制噪声对峰值位置估计的影响，尤其在低信噪比的条件下。

4. **非线性优化**：

该算法通过非线性最小二乘法优化信号的参数，实际上是通过不断调整信号的形状和位置来最小化信号与模型之间的误差，从而得到信号的峰值位置。这一过程体现了非线性优化在信号处理中的应用，特别是在处理复杂波形和噪声影响时，能够得到更精确的结果。

**应用场景**

Sinc-NLS 算法广泛应用于需要高精度峰值定位和时延估计的场景，特别是在以下领域：

1. **通信系统中的时延估计**：

在多用户通信或无线定位系统中，精确的信号时延估计对于系统的同步和定位非常重要。Sinc-NLS 可以用来估算信号的到达时延，从而实现高精度的时钟同步。

2. **雷达和声纳系统**：

在雷达系统中，回波信号通常呈现类似 Sinc 函数的形态，Sinc-NLS 可以用来精确估计目标的距离和速度，特别是在多径效应和噪声环境下。

3. **声学传感器网络**：

在传感器网络中，通过高精度的峰值检测和时延估计，可以实现更精确的位置估算，尤其适用于需要在低信噪比条件下工作的环境。

4. **医学信号处理**：

在医学图像和生物信号分析中，Sinc-NLS 可以用于处理类似心电图（ECG）或脑电图（EEG）等信号，以实现高精度的事件检测。

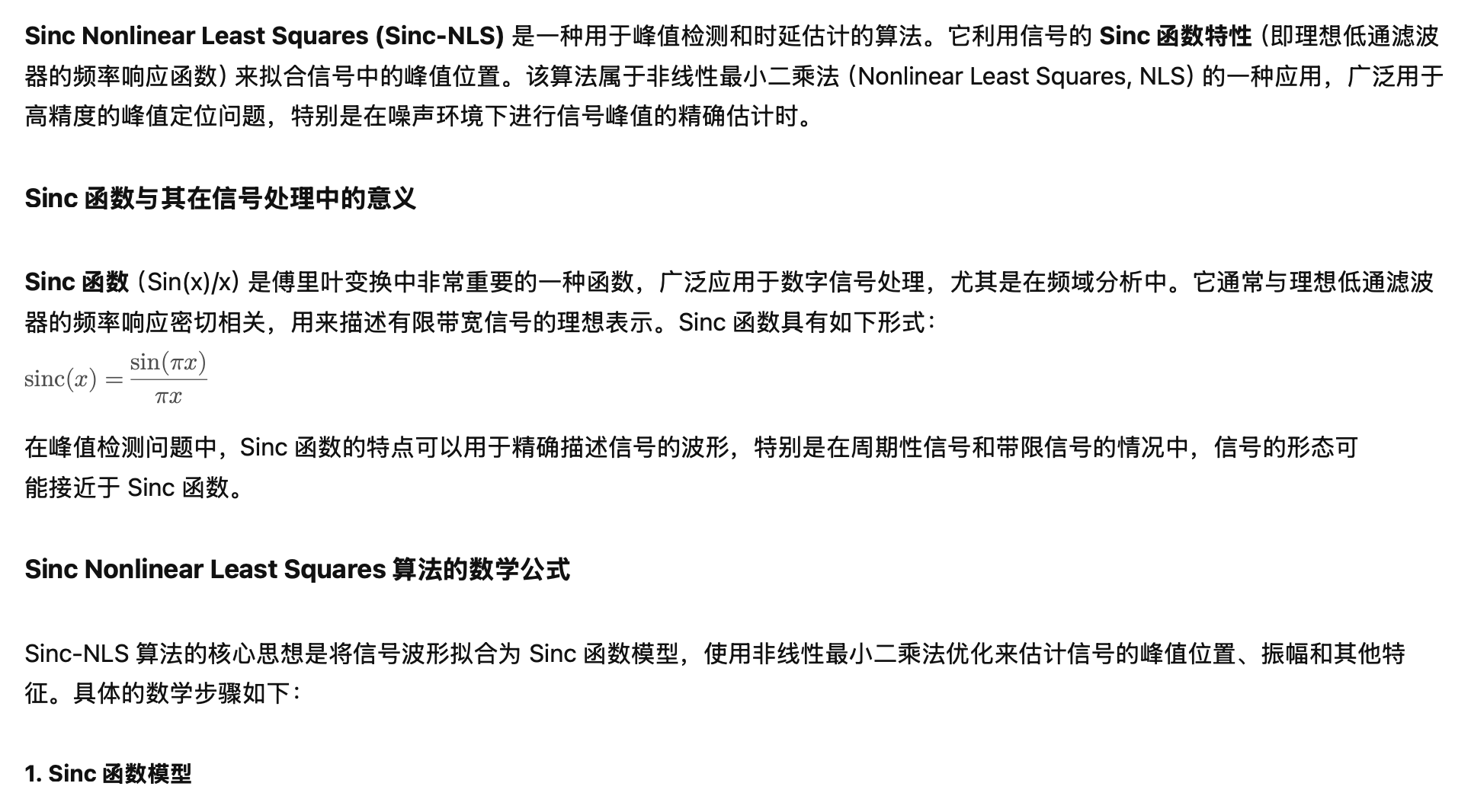
**总结**

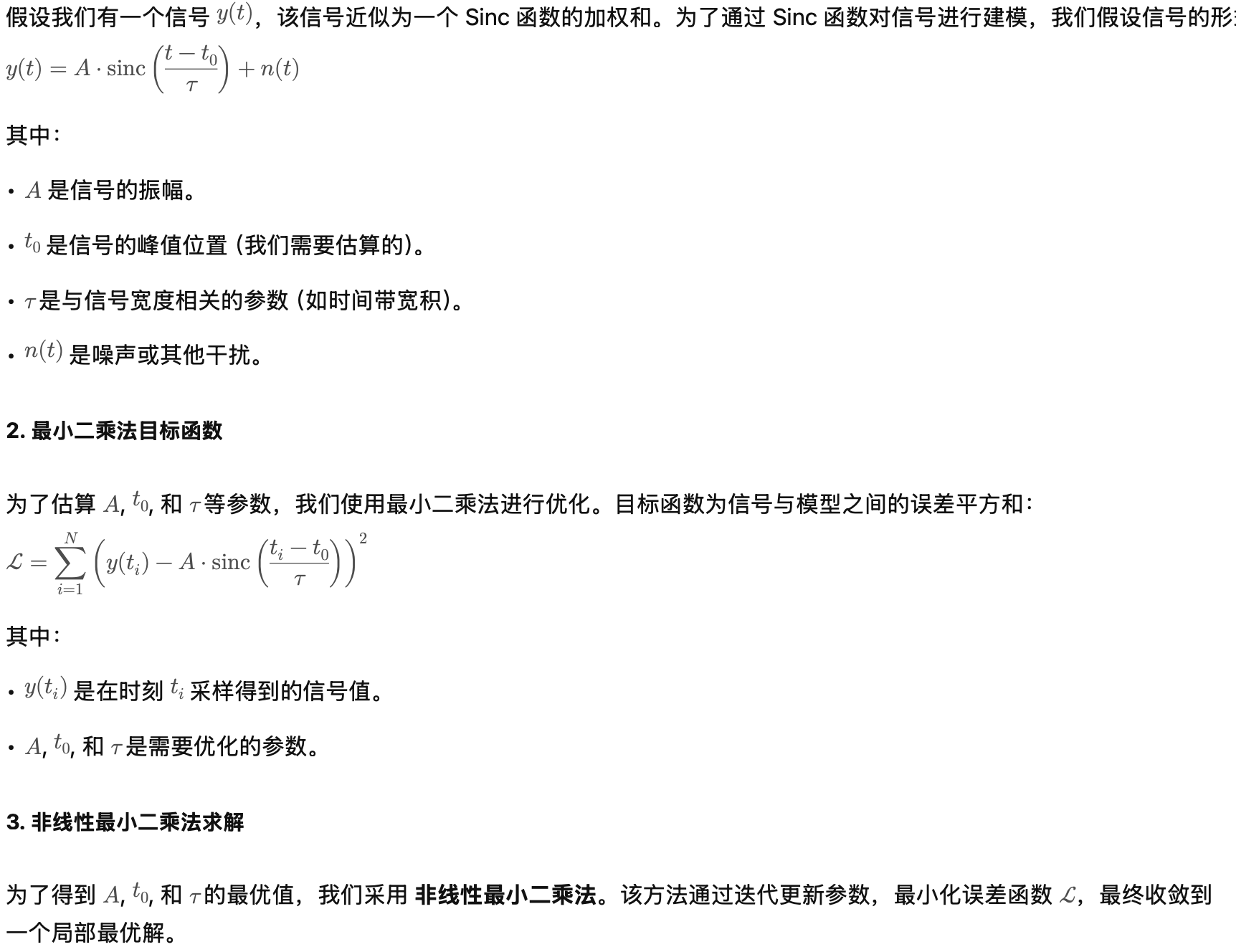
**Sinc Nonlinear Least Squares (Sinc-NLS)** 算法通过将信号模型化为 Sinc 函数并使用非线性最小二乘法优化，能够高精度地估计信号的峰值位置、振幅和宽度。这一算法的优点在于：

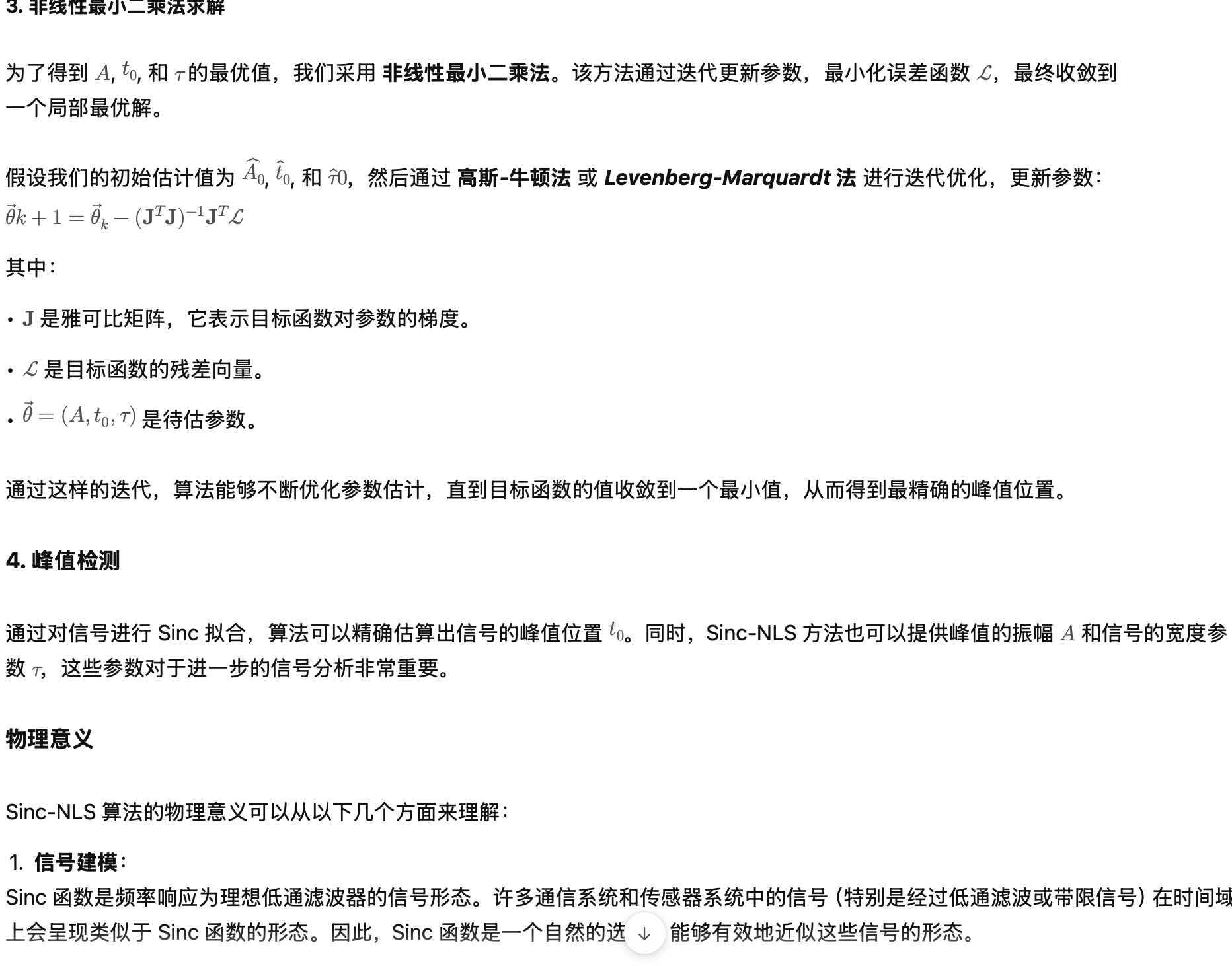
• 通过 Sinc 函数模型拟合信号，能够在噪声环境下提供精确的峰值定位。

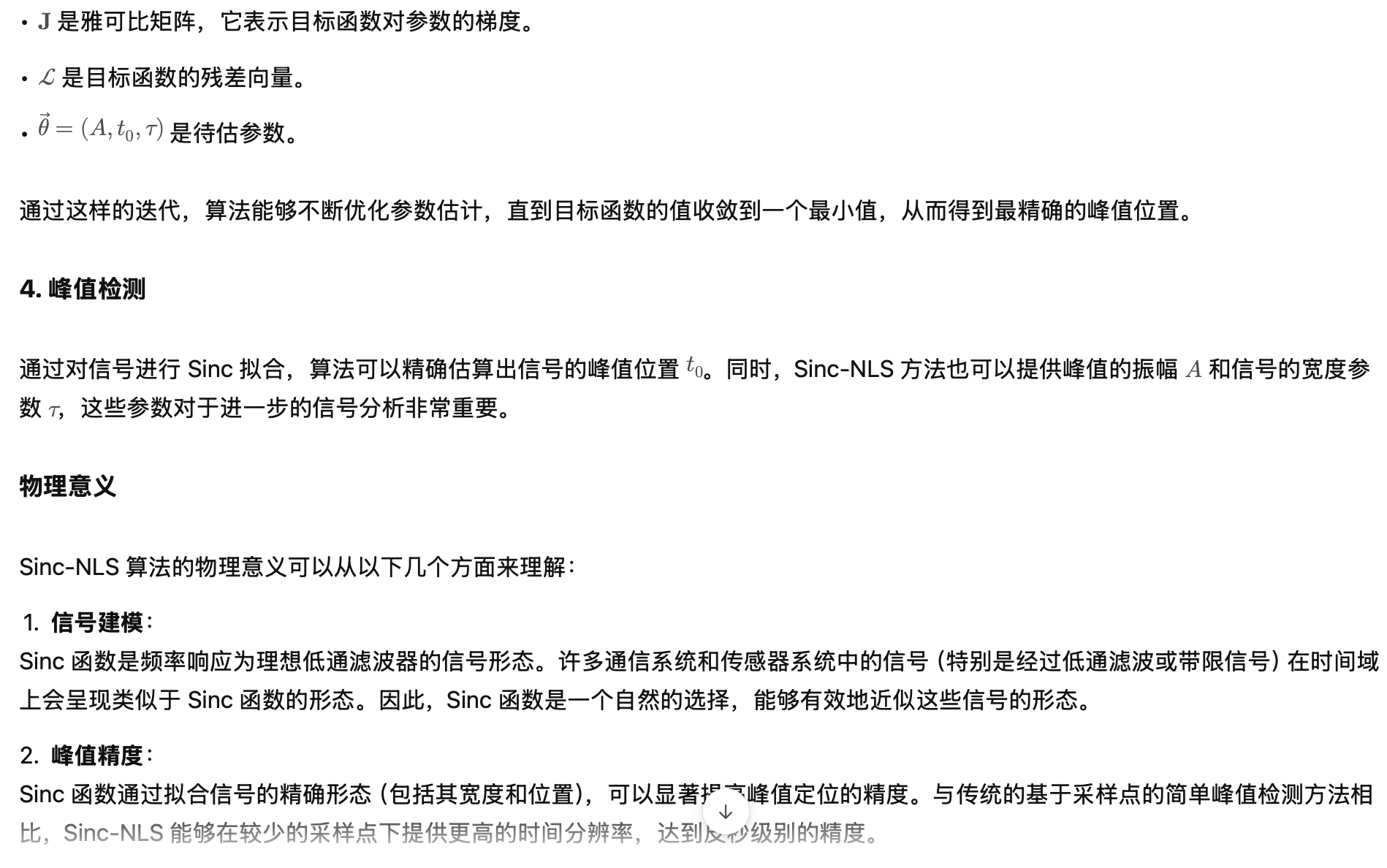
• 通过非线性最小二乘法优化，可以从较少的采样点中提取出信号的精确特征。

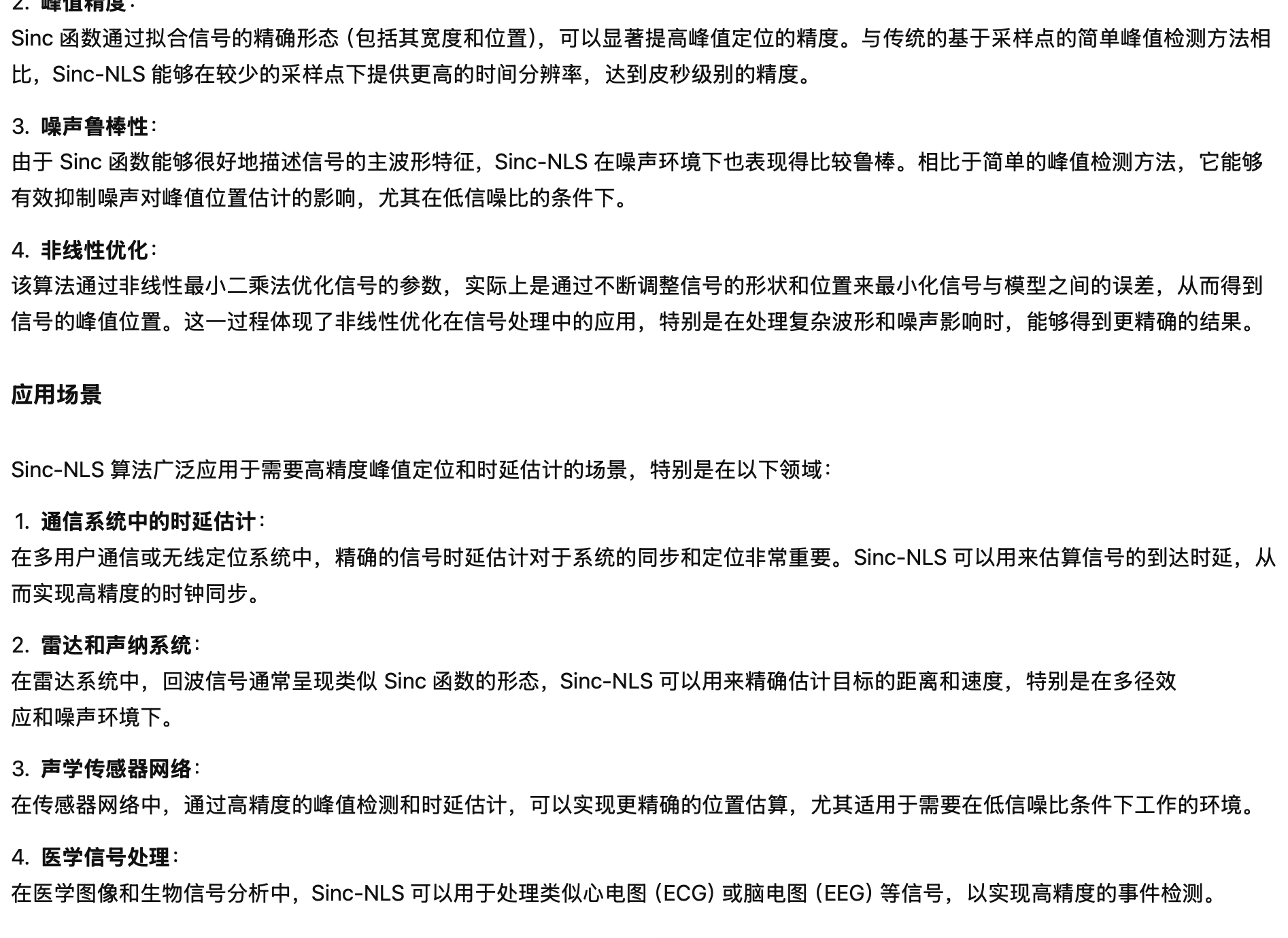
• 能够广泛应用于时延估计、信号同步、目标定位等高精度信号处理任务中。

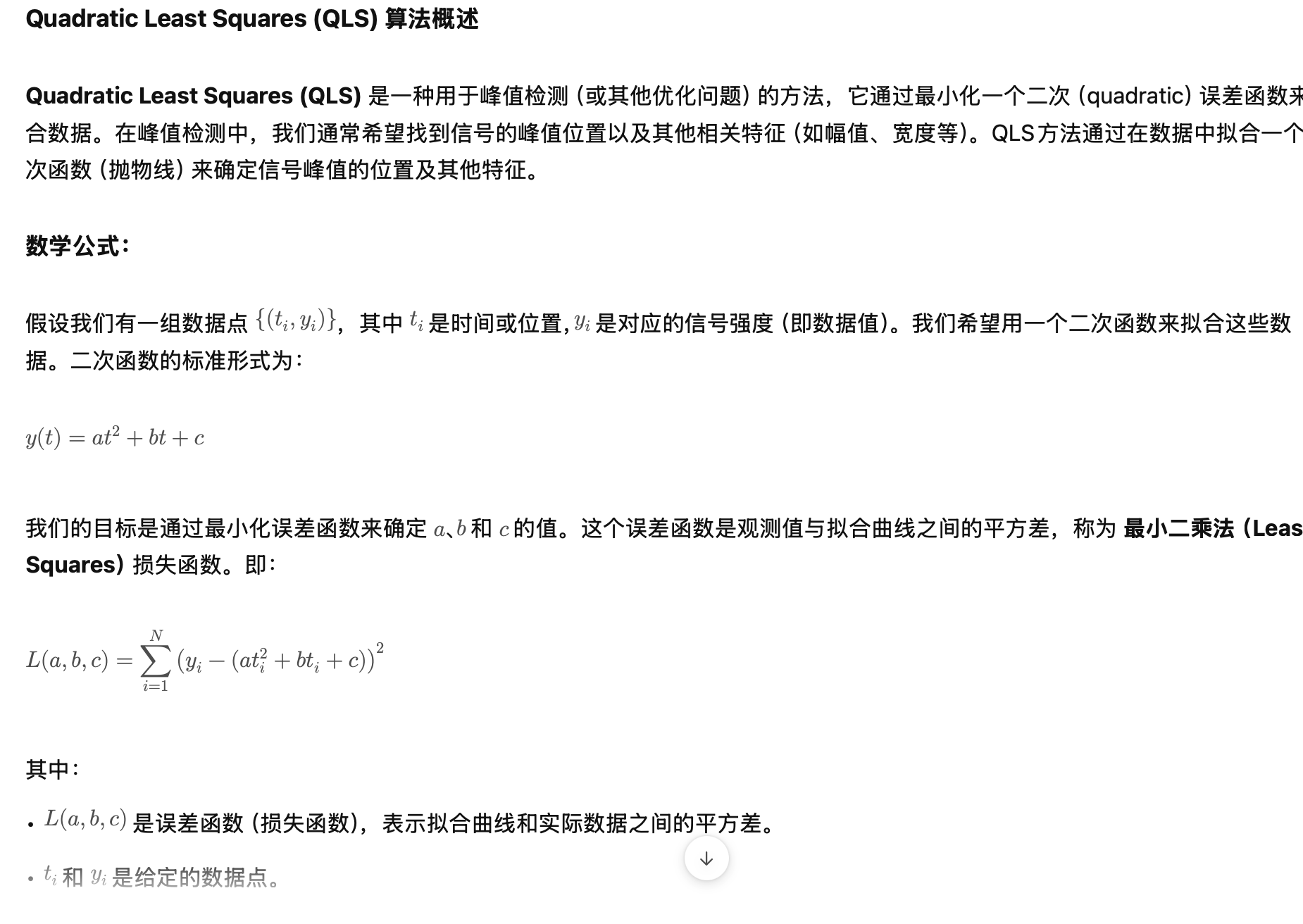


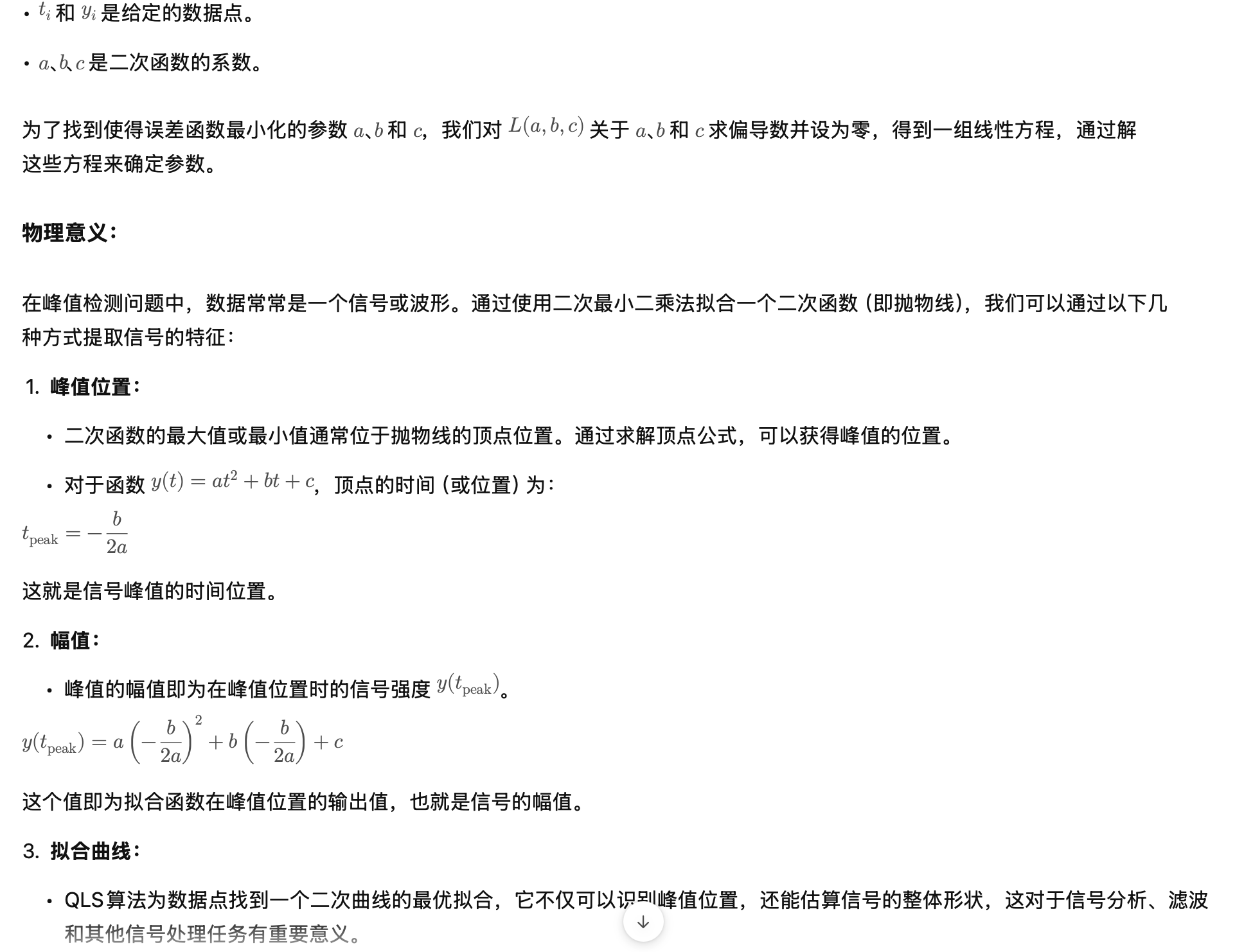
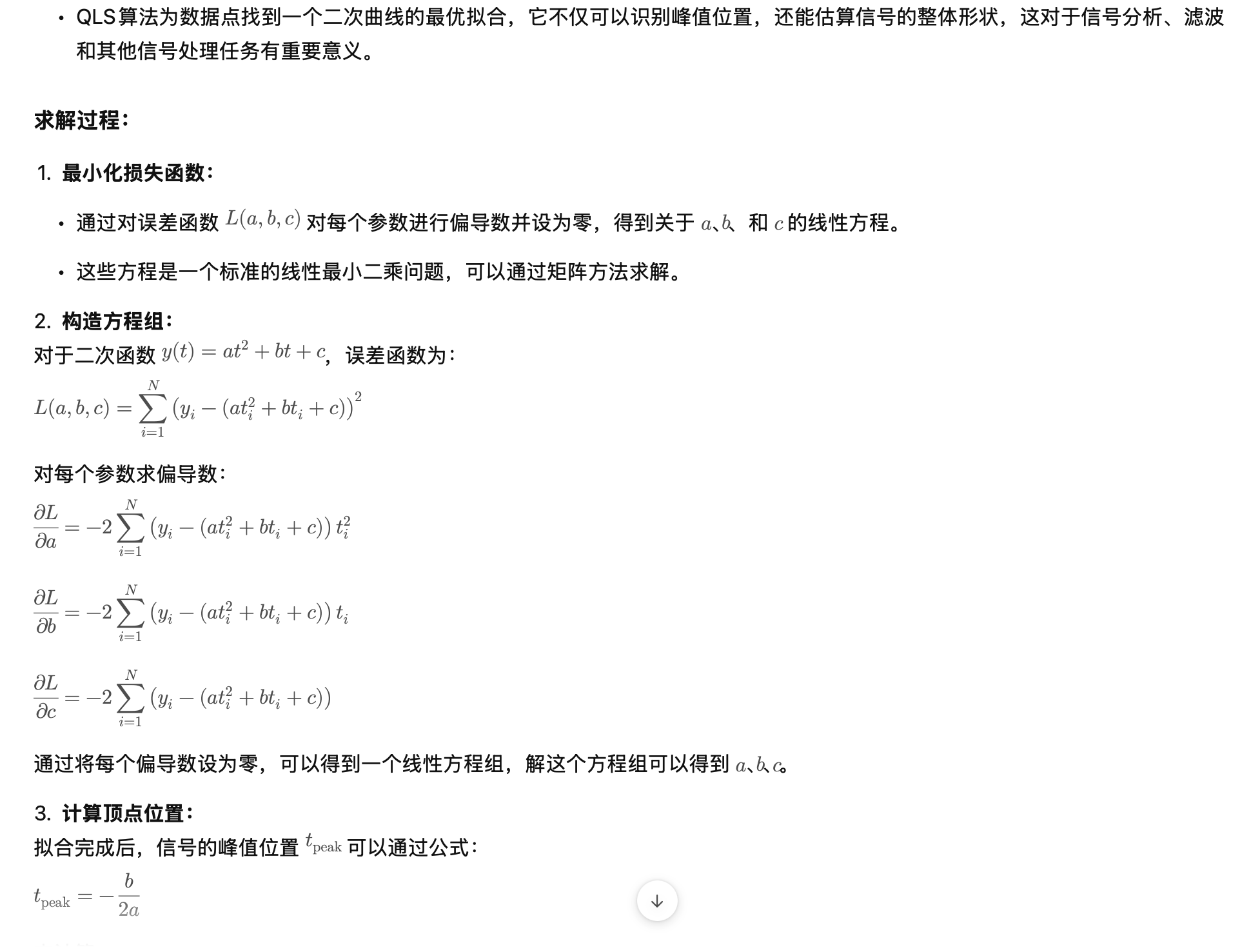


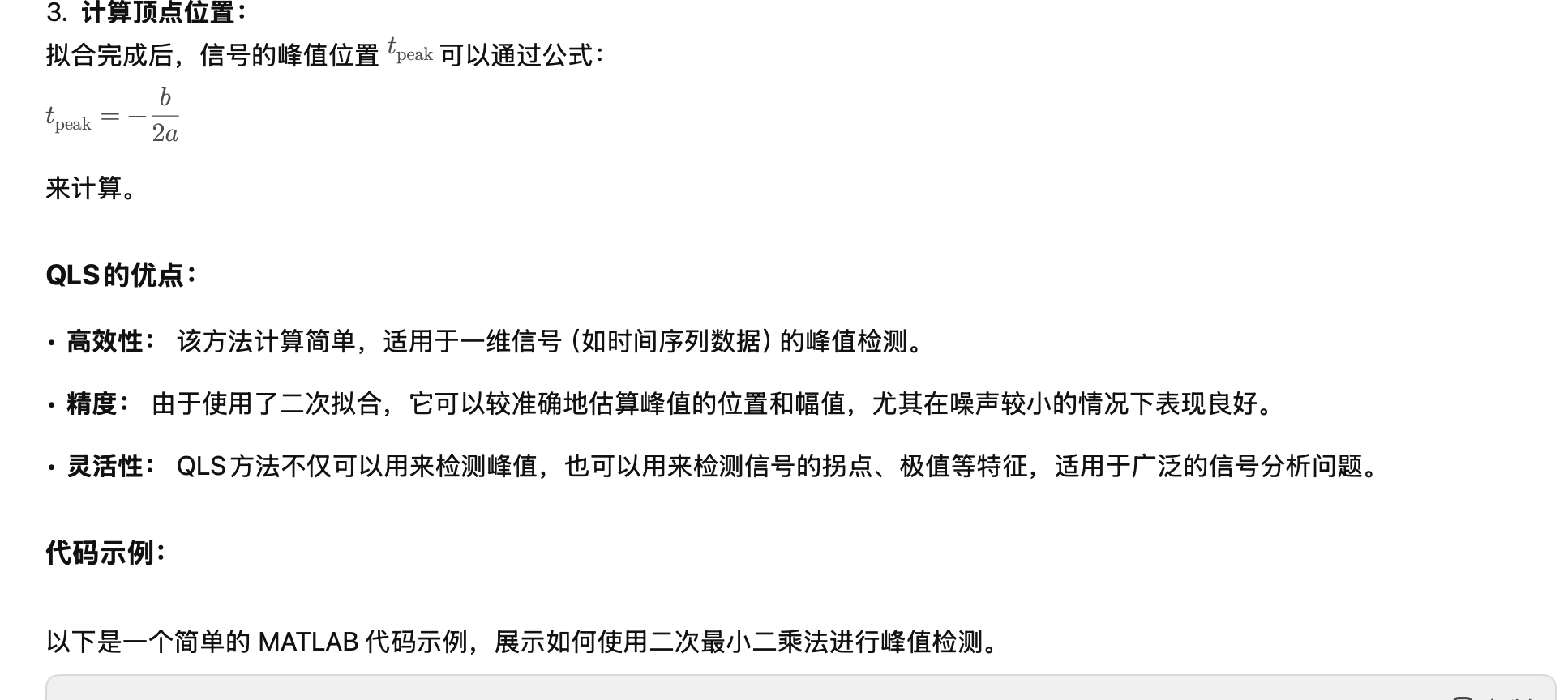












% MATLAB code for High Precision Clock Synchronization using Sinc-NLS (without fminunc)

clear all; close all;

% Step 1: Generate a simulated signal (ideal signal + noise)

Fs = 1e9; % Sampling frequency (1 GHz)

T = 1e-6; % Signal duration (1 microsecond)

t = 0:1/Fs:T-1/Fs; % Time vector

true\_peak\_position = 5e-8; % True peak position (50 ns)

tau = 1e-9; % Time width of the signal (1 ns)

A = 1; % Amplitude of the signal

% Generate a noisy Sinc signal with added noise

signal = A \* sinc((t - true\_peak\_position) / tau);

noise = 0.1 \* randn(size(t)); % Additive Gaussian noise

noisy\_signal = signal + noise;

% Step 2: Define the Sinc function model for fitting

sinc\_model = @(params, t) params(1) \* sinc((t - params(2)) / params(3)); % Sinc model

% Step 3: Define the objective function for least squares fitting

% params = [Amplitude, Peak Position (t0), Time width (tau)]

objective\_function = @(params) sum((noisy\_signal - sinc\_model(params, t)).^2);

% Step 4: Define the gradient of the objective function

% Gradient of the least squares error with respect to [A, t0, tau]

gradient\_function = @(params) [

-2 \* sum((noisy\_signal - sinc\_model(params, t)) .\* sinc((t - params(2)) / params(3))); % dL/dA

2 \* sum((noisy\_signal - sinc\_model(params, t)) .\* params(1) \* (t - params(2)) .\* sinc((t - params(2)) / params(3)).^2 / params(3)); % dL/dt0

2 \* sum((noisy\_signal - sinc\_model(params, t)) .\* params(1) \* (t - params(2)).^2 .\* sinc((t - params(2)) / params(3)).^2 / params(3).^2); % dL/dtau

];

% Step 5: Implement the Gradient Descent algorithm

% Initialize parameters

initial\_guess = [A, 5e-8, 1e-9]; % Initial guess for [Amplitude, Peak position (t0), Time width (tau)]

learning\_rate = 1e5; % Learning rate for gradient descent

max\_iterations = 200; % Maximum number of iterations

tolerance = 1e-8; % Convergence tolerance

params = initial\_guess; % Start with the initial guess

previous\_loss = objective\_function(params); % Calculate the initial loss

% Perform gradient descent

for iter = 1:max\_iterations

grad = gradient\_function(params); % Compute the gradient

params = params - learning\_rate \* grad; % Update the parameters using the gradient

% Calculate the new loss

current\_loss = objective\_function(params);

% Check for convergence (if the loss change is very small)

if abs(previous\_loss - current\_loss) < tolerance

fprintf('Converged after %d iterations.\n', iter);

break;

end

previous\_loss = current\_loss; % Update the loss for the next iteration

end

% Step 6: Extract the estimated parameters

estimated\_amplitude = params(1);

estimated\_peak\_position = params(2);

estimated\_tau = params(3);

% Step 7: Display results

fprintf('True Peak Position: %.8f s\n', true\_peak\_position);

fprintf('Estimated Peak Position: %.8f s\n', estimated\_peak\_position);

fprintf('Estimated Amplitude: %.4f\n', estimated\_amplitude);

fprintf('Estimated Time Width (tau): %.8f s\n', estimated\_tau);

% Step 8: Plot the results

figure;

plot(t \* 1e9, noisy\_signal, 'k', 'LineWidth', 1.5); % Plot noisy signal

hold on;

plot(t \* 1e9, sinc\_model(params, t), 'r--', 'LineWidth', 2); % Plot fitted Sinc model

xlabel('Time (ns)');

ylabel('Amplitude');

legend('Noisy Signal', 'Fitted Sinc Model');

title('High Precision Clock Synchronization using Sinc-NLS (Gradient Descent)');

grid on;

% MATLAB code for High Precision Clock Synchronization using Sinc-NLS

clear all; close all;

% Step 1: Generate a simulated signal (ideal signal + noise)

Fs = 1e9; % Sampling frequency (1 GHz)

T = 1e-6; % Signal duration (1 microsecond)

t = 0:1/Fs:T-1/Fs; % Time vector

true\_peak\_position = 5e-8; % True peak position (50 ns)

tau = 1e-9; % Time width of the signal (1 ns)

A = 1; % Amplitude of the signal

% Generate a noisy Sinc signal with added noise

signal = A \* sinc((t - true\_peak\_position) / tau);

noise = 0.1 \* randn(size(t)); % Additive Gaussian noise

noisy\_signal = signal + noise;

% Step 2: Define the Sinc function model for fitting

sinc\_model = @(params, t) params(1) \* sinc((t - params(2)) / params(3)); % Sinc model

% Step 3: Define the objective function for least squares fitting

% params = [Amplitude, Peak Position (t0), Time width (tau)]

objective\_function = @(params) sum((noisy\_signal - sinc\_model(params, t)).^2);

% Step 4: Initial guess for parameters

initial\_guess = [A, 5e-8, 1e-9]; % Initial guess for Amplitude, Peak position (t0), Time width (tau)

% Step 5: Use fminunc (Nonlinear optimization) to minimize the least squares error

options = optimset('Display', 'off', 'TolFun', 1e-8, 'TolX', 1e-8); % Set optimization options

estimated\_params = fminunc(objective\_function, initial\_guess, options);

% Step 6: Extract the estimated parameters

estimated\_amplitude = estimated\_params(1);

estimated\_peak\_position = estimated\_params(2);

estimated\_tau = estimated\_params(3);

% Step 7: Display results

fprintf('True Peak Position: %.8f s\n', true\_peak\_position);

fprintf('Estimated Peak Position: %.8f s\n', estimated\_peak\_position);

fprintf('Estimated Amplitude: %.4f\n', estimated\_amplitude);

fprintf('Estimated Time Width (tau): %.8f s\n', estimated\_tau);

% Step 8: Plot the results

figure;

plot(t \* 1e9, noisy\_signal, 'k', 'LineWidth', 1.5); % Plot noisy signal

hold on;

plot(t \* 1e9, sinc\_model(estimated\_params, t), 'r--', 'LineWidth', 2); % Plot fitted Sinc model

xlabel('Time (ns)');

ylabel('Amplitude');

legend('Noisy Signal', 'Fitted Sinc Model');

title('High Precision Clock Synchronization using Sinc-NLS');

grid on;