Problem Chosen
A

2024
ShuWei Cup
Summary Sheet

Team Control Number
2024110230228

The Frequency Estimation Problem in Aircraft Laser Doppler Velocity Measurement

Summary

For problem one, distinguish how mixed variable data differ numerically from the computational formulas used in modeling statistical features. The basic statistics of the data such as mean, maximum and minimum, variance, skewness and kurtosis are calculated to portray the distribution and shape of the data, Fourier transform is calculated to achieve the spectral analysis, and the data is further analyzed by calculating the abnormal amplitude fluctuations and temporal irregularities. It is found that the noise sequence is an independent sequence obeying a normal distribution with a mean of 0.00584 and a variance of 3.9904. Moreover, the noise series approximates the white noise data to a large extent and has the characteristics of white noise data.

For problem two, first process the time series data containing the time and value of the received signal. Isolated forest algorithm is used to identify outliers in the data. In order to deeply analyze the data and detect possible small shifts or trend changes, sliding window technique and CUSUM (cumulative sum) method are further introduced to detect the frequency period. Finally, the Hidden Markov Model was introduced to estimate the frequency of the non-noise portion and the frequency with the largest amplitude in the FFT results was found to be 41 × 106 Hz.

For problem three, this paper uses FFT, PSD, MUSIC algorithm and autocorrelation function method to estimate the frequency of period 3 signal data. The signal quality is improved by removing the DC component, smoothing and filtering, and the signal-to-noise ratio reaches 20.69dB. The analysis shows that the main frequency of the signal is about 35MHz, in which FFT, PSD and MUSIC algorithm are estimated to be 34.99986MHz, 35.16MHz and 35.4MHz respectively. The results of all methods are generally consistent, which verifies the reliability of frequency estimation and provides a reference for high-precision signal analysis.

For problem four,this paper proposes two methods for estimating the frequency of intermittent signals received during the fourth flight period. The first method is the Kalman filter, which utilizes a recursive algorithm to predict and update the signal, ultimately determining the frequency to be 56.29 MHz. This result is further validated using FFT spectrograms,demonstrating its reliability. The second method is wavelet transform, which calculates the main frequency from the center frequency to obtain the frequency of the intermittent signal as 55.56 MHz. Both methods indicate that the frequency estimation is reliable.

Key word: Probability density function, white noise, Hidden Markov, FFT, Kalman Filter,Wavelet Transform

Content

1. Introduction

1.1 Background

Flight safety is closely related to airspeed measurement. Traditional methods of airspeed measurement have limitations, whereas laser velocimetry emerges as an ideal choice due to its high precision and non-contact advantages. The challenge in laser velocimetry lies in accurately extracting Doppler shift information from noise to calculate airspeed. This paper delves into the issue of frequency estimation in laser velocimetry, analyzes the characteristics of signal noise, and designs frequency estimation algorithms tailored for different scenarios. We will analyze the signal noise during flight cycle 1 and design algorithms to estimate the signal frequencies during flight cycles 2 and 3. Additionally, we will investigate frequency estimation methods in intermittent reception mode. By studying the problem of frequency estimation, we aim to improve the accuracy of airspeed measurement and ensure flight safety.

1.2 Our Work

- Analysis of Noise Characteristics: Conduct an analysis of the noise characteristics of the received signals during flight cycle 1, providing a foundation for the subsequent design of frequency estimation algorithms.
- Design of Frequency Estimation Algorithms: Develop different frequency estimation algorithms for flight cycles 2 and 3, taking into account the differences in signal and noise characteristics.
- Analysis of Intermittent Reception Mode: Analyze the intermittent reception mode during flight cycle 4 and design corresponding frequency estimation algorithms to address situations with insufficient information.

2. Problem analysis

2.1 Data analysis

The time series of several variables are first plotted, that is, the original data is split, and the signal data possessing noise appears as the X(t) variable, and it is known that the amplitude of the non-noise portion of the received signal is 4, the frequency is 30*106 Hz, and the phase is 45○ . The noise sequence and the pure signal sequence (i.e., the sequence with no noise interference) can be obtained. While graphing the time series, plot an image of the probability density function of each variable to visually explore the statistical properties of the series.

As can be seen from the above figure, the noise sequence is very similar to the original

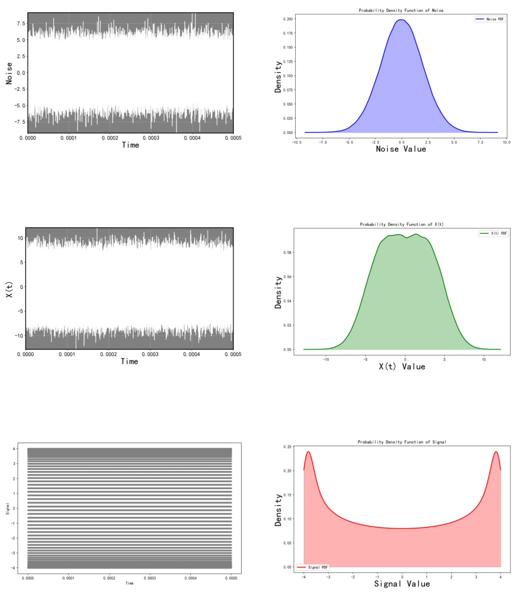


Figure 1 Plot of time series and probability density function

time series plot of the signal sequence, the fluctuations present the same state, only the values differ, whereas the pure signal sequence behaves differently and it is not possible to plot a line graph, or rather, the data points of the plotted line graph cover the entire image, so a scatter plot is considered to represent it. Scatter plots of pure signal sequences show a specific tendency that the numerical points are not fully covered in the range but some specific points are more numerous, thus forming horizontal lines.

Considering that the scatterplot was plotted for the pure signal sequence, in order to find out the pattern between the variables, the scatterplot was similarly plotted for the other two variables, and it was found that the scatterplot of the noise sequence and the original signal sequence differed from the pure signal sequence.

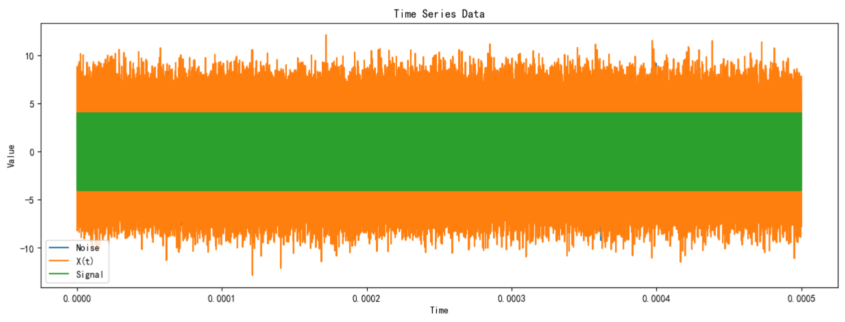


Figure 2 Line graph of three variables

Plotting the images individually reveals the trend characteristics of the individual variables, whereas plotting three variables on a single graph enables comparison of the differences between the variables. There are three variables in the graph above, but there are only two visible colors, orange and green, with the noise data overlaid. What is peculiar is that the pure signal data as a whole has a rectangular shape, with barely discernible fluctuations, but rather is average and stable like a uniform distribution.

Table 1 Statistical characteristics of flight cycle 1 data variables

Statistics	Noise	X(t)	Signal
average value	0.00584	0.00585	0.00001
upper quartile	0.00363	-0.00166	0.12564
(statistics) standard deviation	1.99547	3.46016	2.82843
extremely poor	18.35326	24.94152	7.99605
coefficient of variation	341.75083	591.45285	250001.50001
skewness	-0.00153	-0.00428	-0.00001
kurtosis	-2.98696	0.00585	0.00001

The table highlights significant differences among the noise sequence, original signal sequence, and pure signal sequence, reflecting distinct periodic fluctuations and trends. This distinction aids in analyzing noise characteristics, emphasizing the need for independent examination of the noise sequence.

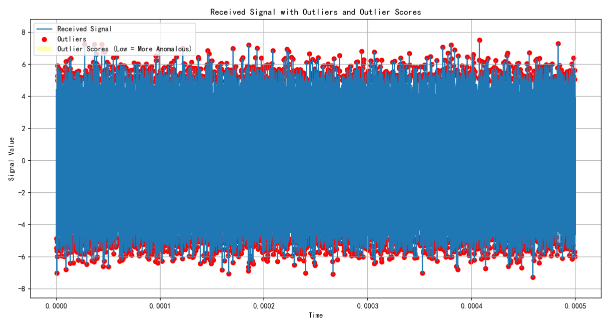


Figure 3 Sliding window method outliers

In the separation of outliers, it first calculated the signal mean (and optionally standard deviation) within each of the sliding windows that slid over the data for local analysis. It then calculated normalized deviations based on the mean and standard deviation of each window and calculated CUSUM values based on the accumulation of these deviations. By setting a threshold, it was able to flag as outliers those points where the CUSUM value exceeded that threshold, which could represent trend changes or persistent small shifts in the data.

The above figure and table, show the original signal, the outliers detected by the isolated

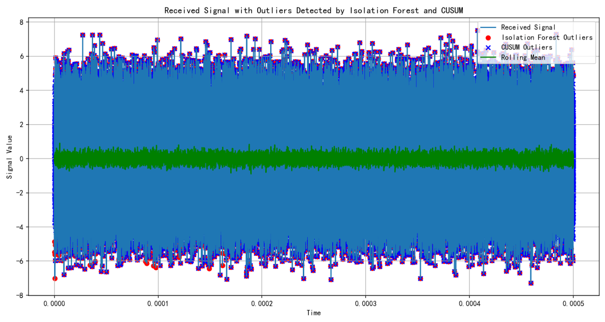


Figure 4 CUSUM method outliers

Table 2 Sliding window periodic change detection table

Index	Window Start	Window End	Mean	Standard Deviation	CUSUM Outlier
0	0.000000e+00	1.000000e-07	0.038215	2.100516	0.0
1	2.000000e-09	1.020000e-07	-0.008944	2.080996	0.0
2	4.000000e-09	1.040000e-07	-0.036687	2.036863	0.0
3	6.000000e-09	1.060000e-07	-0.035542	2.036609	0.0
4	8.000000e-09	1.080000e-07	-0.059198	2.036020	0.0
⋮	⋮	⋮	⋮	⋮	⋮
249946	4.998920e-04	4.999920e-04	-0.409042	1.877887	1.0
249947	4.998940e-04	4.999940e-04	-0.460952	1.865919	1.0
249948	4.998960e-04	4.999960e-04	-0.455193	1.862366	1.0
249949	4.998980e-04	4.999980e-04	-0.430861	1.830697	1.0
249950	4.999000e-04	5.000000e-04	-0.436416	1.833177	1.0

forest, and the anomalies detected by the CUSUM method. In addition, some of the solution results are also shown in the Sliding Window Periodic Change Detection Table and the CUSUM Periodic Change Detection Table to provide additional information about the overall trend of the signal. In this way, it not only helped us to identify isolated outliers in the data, but also revealed possible trending anomalies.

In this paper, the signal data for period 3 is processed as follows: Remove the DC component: Subtract the average value of the data from each data point. Smooth processing: Sliding average method can be used to smooth data and reduce noise. Filter: Select a low-pass or band-pass filter as needed to remove high-frequency noise, and the result after processing is shown as follows:

The signal amplitude in the figure is large, especially in the original signal and the signal after removing the DC component, showing large fluctuations. This indicates that the signal contains high-frequency noise components. Compared with the signal without DC component,

Table 4 CUSUM cycle change detection table

Primary Sequence	Test Results	Primary Sequence	Test Results	Primary Sequence	Test Results
1	False	18934	True	159132	True
2	False	18935	True	159133	True
3	False	18936	True	159134	True
4	False	18937	False	159135	True
5	False	18938	False	159136	True

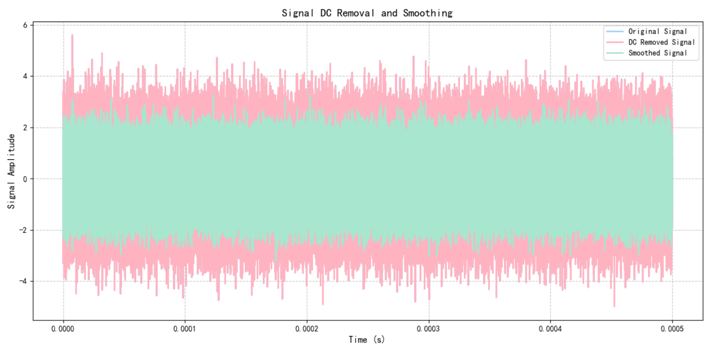


Figure 5 Signal DC Removal and Smoothing

the fluctuation of the smoothed signal is significantly reduced, indicating that the smoothed signal effectively reduces the noise component and makes the signal more observable.

This paper analyzes the signal data for period 4 by addressing the intermittent reception model and designing a frequency estimation method suited for discontinuous signals. The study investigates the uniformity or gaps in signal acquisition during the fourth flight period through visualizing the relationship between received signal time and time differences. The visualization results are as follows:

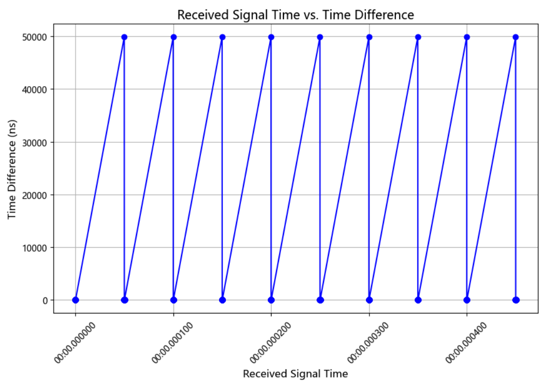


Figure 6 Relationship between received signal time and time difference

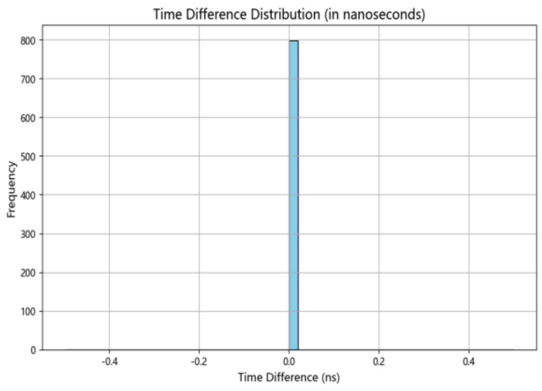


Figure 7 Time difference distribution

The relationship between the received signal time and time difference in the figure shows that the line is not steadily increasing but has obvious fluctuations, indicating that there are gaps in the signal reception time. The time difference distribution graph shows that the time interval between signal receptions is very small, resulting in a vertical shape of the time difference distribution.

Additionally, we have statistically analyzed the time difference information for the fourth stage in Attachment 1 as follows:

Table 5 Time Difference Statistical Information

Statistics	Value	Statistics	Value
Count	799	Min	1.00×10^{-9}
Mean	5.63×10^{-7}	25%	2.00×10^{-9}
Std Dev	5.26×10^{-6}	Median (50%)	2.00×10^{-9}
Max	4.98×10^{-5}	75%	2.00×10^{-9}

According to the statistical information, we find that most of the time, the received signal is continuous and stable. However, in some cases, there are longer time intervals between signals. We need to correctly handle these unstable time intervals to accurately estimate the frequency.

2.2 Analysis of question one

In Flight Cycle 1 of Appendix 1, it is known that the non-noise portion of the received signal has an amplitude of 4, a frequency of 30 × 106 Hz, and a phase of 45○. Please analyze the noise z (t) characteristics of the received data for flight cycle 1. For this problem, we first preprocess the data, split the data sequence into noise sequence and pure signal sequence, and after analyzing the original sequence, noise sequence and pure signal sequence independently, we statistically characterize the values of the three variables, and obtain the noise sequence characteristics through the probability density function.

2.3 Analysis of question two

In flight cycle 2 of Annex 1, the actual received signal is known to have an amplitude of 2 and a phase of 0○ . The noise sequence can be assumed to be two cases, white noise and non-white noise, and the frequencies are estimated for each of the two cases by identifying outliers, detecting the frequency period, and building a Hidden Markov Model to estimate the frequency of the non-noise part.

2.4 Analysis of question three

This paper employs various frequency estimation methods, including FFT, PSD, MUSIC, and autocorrelation function, to analyze period 3 signal data. Preprocessing steps such as DC removal, smoothing, and filtering enhance the signal's quality, characterized by a high sampling rate (500MHz) and a strong SNR (20.69dB). Frequency analysis reveals consistent main frequency estimates around 35MHz (e.g., FFT: 34.99986MHz, PSD: 35.16MHz, MUSIC: 35.4MHz), confirming the reliability of the methods despite minor differences. The study highlights the effectiveness of these techniques in accurately identifying the signal's dominant frequency.

2.5 Analysis of question four

The fourth flight period employs an intermittent reception mode. The challenge with this mode is that the received signal data may contain gaps, leading to information loss and affecting the accuracy of frequency estimation. To address this issue, it is necessary to analyze the sampling interval of the signal, identify data gaps, and design a reasonable method for frequency estimation.

3. Symbol and Assumptions

3.1 Symbol Description

Table 6 Symbol Description Table

Symbol	Description	Symbol	Description
f_k	Frequency estimation	$\dot{f}_k$	Rate of frequency change
W_{k-1}	Process noise	Q	Covariance matrix
H	Observed frequency component	$\bar{x}_k$	Predicted state estimation
$\bar{P}_k$	Predicted error covariance	$\mathcal{W}(a,b)$	Wavelet Transform
$x(t)$	Input signal	Δt	Time interval

3.2 Fundamental assumptions

- The process noise conforms to a zero-mean Gaussian distribution, where the noise introduced during the state transition is random and its distribution follows a Gaussian (normal) distribution with a mean of zero.

- The observation noise conforms to a zero-mean Gaussian distribution. Similarly, the noise introduced during the observation process is also random, and its distribution also follows a Gaussian distribution with a mean of zero.
- The fourth flight signal is obtained through scale and displacement transformations of a set of wavelet basis functions. Wavelet transform achieves this by continuously subdividing the original signal and selecting wavelet bases to perform multiple scale and displacement transformations, thereby obtaining a set of distinct functions.

4. Model

4.1 Modeling and Solving Problem 1

The first step in the model building of Problem 1 is the feature mathematical model, and differentiate how different types of mixed data are numerically different from the computational equations in building the feature model. The basic statistics of the data such as mean, maximum and minimum, variance, skewness and kurtosis are calculated to portray the distribution and shape of the data, the Fourier transform is calculated to realize the spectral analysis, and the abnormal amplitude fluctuations and temporal irregularities are calculated through the analysis of the data to further establish the characteristic mathematical model.

Fourier transform:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-\frac{j2\pi kn}{N}}, \quad k = 0, 1, \ldots, N-1$$

$x[n]$ is the input signal, $X[k]$ is the input spectrum, N is the signal length, and $e^{-\frac{j2\pi kn}{N}}$ is a complex exponential function.

Describe the degree and location of concentration:

$$SampleMean : \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Describe the degree of variation:

$$Coefficientofvariation : CV = \frac{s}{\overline{x}} \times 100\%$$

Describe the shape of the distribution:

$$\text{Skewness: } v_1 = \frac{1}{s^3}\sum_{i=1}^{n}(x_i - \overline{x})^3$$

$$\text{Peakness: } v_2 = \frac{1}{s^4}\sum_{i=1}^{n}(x_i - \overline{x})^4$$

Descriptive Relevance:

$$\text{Correlation coefficient: } p_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

With the help of data visualization and analysis to intuitively understand the characteristics of the data, to form an impression of the data as a whole, to check whether the data as a whole is missing, the type of data and the characteristics of the data. On the basis of the previous steps, we carry out feature exploration, calculate the spectrum with Fourier transform, draw fluctuation amplitude images, explore the existence of seasonality and periodicity from the point of view of time series, and draw autocorrelation and partial autocorrelation functions for the noisy series.

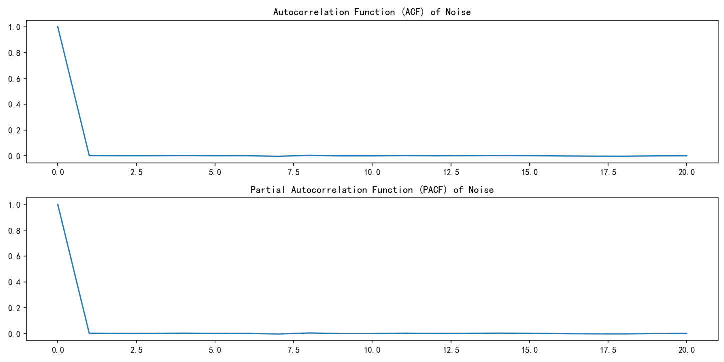


Figure 8 Noise sequence ACF and PACF plots

The ACF and PACF line plots for noise illustrate their behavior with varying lag values. The ACF (top subplot) shows a rapid decline of autocorrelation coefficients to near zero, indicating that periodicity or trends in the data diminish quickly. Similarly, the PACF (bottom subplot) displays strong correlations at lower lags, which also quickly decay to near zero, highlighting limited predictive power at higher lags.

Overall, the two charts together reveal the non-stationarity and lack of long-term memory effects that characterize this time series data.

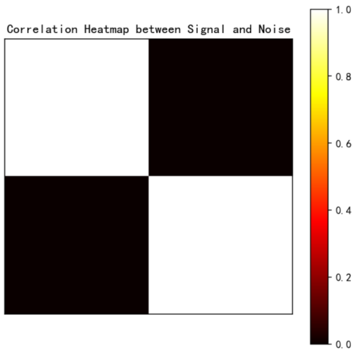


Figure 9 Correlation between pure signal and noise

In the previous data visualization and analysis, we found different trends and characteristics for each of the three variables, and now a more accurate way to calculate the Pearson correlation coefficients reveals that the pure signal sequence and the noise sequence really do not exist, which validates the previous conjecture that the sequences are independent of each other.

We have already established that the noise sequence is normally distributed, and hypothesis testing further allows us to guess that the noise sequence may also be a white noise sequence, which is random and memoryless. In order to determine whether the data in the column ”noise” is white noise, the following analysis is performed:

Autocorrelation function (ACF) values were calculated for observing the autocorrelation of the data at different lags.

A Ljung-Box test was performed to assess whether the data exhibited randomness. A unit root ADF test was performed to check the smoothness of the data.

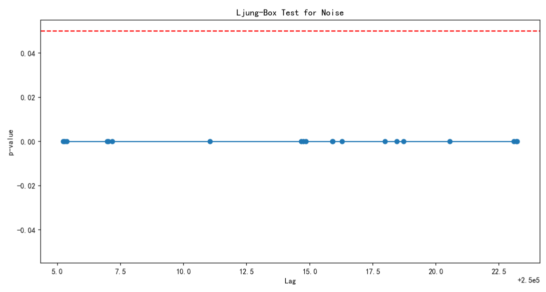


Figure 10 Ljung-Box Test for Noise

The upper graph is a scatter plot to show the results of the Ljung-Box test. There is a red dotted line in the graph indicating the critical value at a significance level of 0.05, while the solid blue circles represent the p-values at different lag orders (lag). As can be seen from the figure, the p-value gradually approaches 0 as the lag order increases, indicating that in most cases we cannot reject the original hypothesis that the series are independent.

Autocorrelation function (ACF) values: between lags 1 and 20, most of the ACF values are within the confidence intervals, indicating that autocorrelation is not strong at these lags. Ljung-Box test p-value: p-value is high for most of the lags, which means we cannot reject the original hypothesis that the data is random. ADF test statistic and p-value: a negative ADF statistic and a small p-value indicate that the data is smooth.

Based on the above analysis, the following is the judgment of whether the data in the ”noise” column is white noise or not:

The value of autocorrelation function (ACF) shows that autocorrelation is not strong at most lags, which is consistent with the characterization of white noise.The high p-value of the Ljung-Box test indicates that the data may be random, which is also a characteristic of white noise.The ADF test indicates that the data is smooth, which is also a necessary condition for white noise data.

Taking these three points together, it can be concluded that the ”NOISE” column is characterized by white noise. However, it should be noted that although these tests support the hypothesis of white noise, there is no actual data that exactly matches the ideal white noise model. Therefore, we can say that the data in the ”noise” column largely approximates white noise.

In addition the data obeyed a normal distribution with a mean of 0.00584 and a variance of 3.9904.

$$f(x \mid \mu, \sigma) = \frac{1}{3.9904\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-0.00584}{3.9904}\right)^2}$$

4.2 Modeling and Solving Problem 2

Three steps are implemented in the solution of problem two, in the first step, the isolated forest algorithm is used to correct the data by calculating the outlier degree to determine whether there are outliers in the data. In the second step, the frequency is detected by using sliding window and CUSUM method to detect the change in signal period. In the third step, the Hidden Markov Model (HMM) is considered to estimate the frequency of the non-noise part of the received signal for flight period 2.

The **Isolation Forest (IForest)** algorithm detects outliers by isolating them through randomly constructed trees. Outliers are quickly isolated to leaf nodes, resulting in shorter average path lengths across trees, which measure their degree of anomaly.

The **sliding window approach** identifies local features in time series data by moving a fixed-size window step by step. This generates localized subsequences, enabling targeted analysis within each window.

The CUSUM method is a method for detecting change points in a data series. It monitors the degree of data deviation by calculating the cumulative sum of the data series and triggers an event when the cumulative sum exceeds a preset threshold.

Hidden Markov Model (HMM) is a statistical model which is used to describe a Markov process with implicitly unknown parameters.The HMM model consists of 2 sets of states and 3 probability matrices.The specific computational steps of the model are shown below:

Consider two cases. The first, the white noise case: the Fourier transform is applied to estimate the frequency during the time period identified as the signal state. Since white noise is uniform in the frequency domain, it does not significantly affect the estimation of the signal frequency. Second, non-white noise case: in this case, a more complex processing may be required, considering Hidden Markov estimation of frequency.

A simplified HMM-based frequency estimation process involves: preprocessing the signal and segmenting it into multiple segments. Define two HMM states—noise and signal—and select emission probabilities based on signal statistics (e.g., mean, variance). Use the Baum-Welch algorithm to estimate HMM parameters and apply the Viterbi or forward-backward algorithm for state decoding. This results in a state sequence identifying noise and signal distribution within the data.

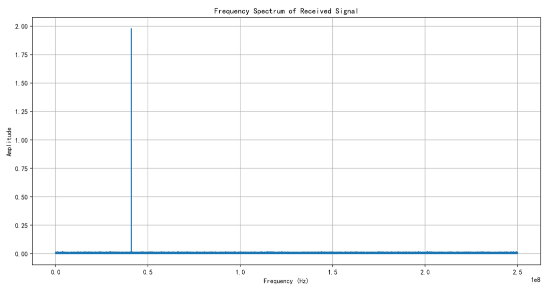


Figure 11 Magnitude spectrum of FFT results

The dominant frequency is determined as $41 * 10^6$ Hz using FFT. Introducing the Hidden Markov Model (HMM) for frequency estimation focuses on modeling the signal's potential states (e.g., noise and signal states) and their transitions. HMM does not estimate frequency directly but aids in distinguishing signal components. Combined with techniques like Fourier transform or autoregressive modeling, HMM enhances the accuracy of indirect frequency estimation.

4.3 Modeling and Solving Problem 3

After the signal is preprocessed, PSD estimation is used to analyze the spectrum characteristics of the signal. By calculating the power spectral density, the energy distribution of the signal at each frequency can be obtained. PSD estimation helps analyze the strength of the dominant frequency component in the signal and provides frequency domain characteristics for subsequent frequency estimation. It can also help distinguish the difference between noise and signal. If there is a strong noise component in the signal, PSD estimation helps to identify the frequency range where the noise is located.

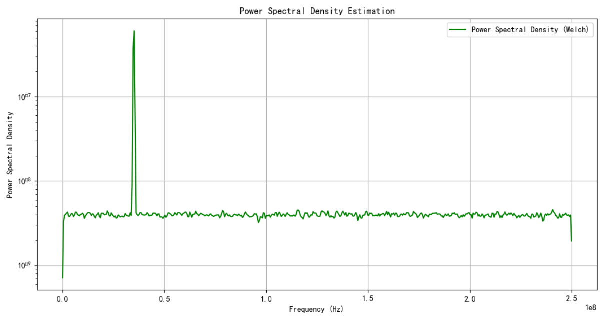


Figure 12 Power Spectral Density Estimation

The power spectral density (PSD) analysis reveals a significant peak near 0.5 Hz, representing the main frequency of the signal where its energy is most concentrated. Other frequencies exhibit low and flat PSD values, indicating random noise. In the high-frequency region, a decline in power density is observed, likely due to signal bandwidth limitations or high-frequency noise attenuation. Using a logarithmic Y-axis highlights a sharp increase in PSD around the main frequency, ranging from 10^{27} to 10^{29}, confirming its prominence. The PSD plot shows a single dominant frequency peak, suggesting the signal is likely a single-frequency sine wave. By applying Python's `find_peaks` function, the main frequency is estimated at 35.16 MHz, representing the strongest frequency component in the signal.

The frequency of the signal is estimated without knowing the amplitude and phase of the signal. Since the noise characteristics of the signal may be different and the signal frequency may change with time, the simple frequency estimation method may not accurately capture the instantaneous frequency of the signal. Therefore, a central part of the problem is dealing with the non-stationarity of the signal, that is, the frequency may not be fixed.

The short-time Fourier transform (STFT) divides the signal into several small time periods, and then conducts frequency analysis in each time period respectively. In this way, the frequency components of the signal in different time periods can be obtained, and the frequency changes over time can be detected. STFT does not depend on the amplitude and phase information of the signal, but can analyze the instantaneous frequency components directly based on the spectrum information.

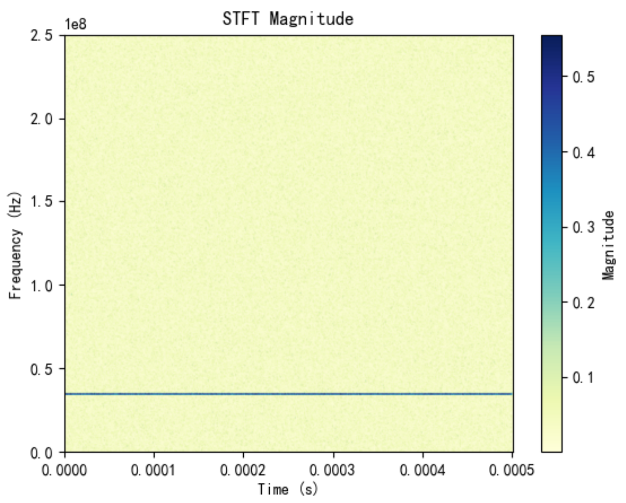


Figure 13 STFT diagram

From the STFT diagram, it can be seen that the main frequency of the signal is stable in the whole time range, and the frequency is almost no fluctuation. With STFT, the previous frequency estimate (around 35.16MHz) can be further verified and the frequency is confirmed to be stable in time.

PSD estimation provides a global frequency domain feature for frequency analysis, which helps to determine the main frequency of the signal and its significance. STFT provides an in-depth examination of frequency changes over time and is particularly suitable for detecting non-stationary signals. The frequency of the signal in this paper is stable in time, which further verifies the accuracy of frequency estimation.

The principle of fast Fourier transform (FFT) for frequency estimation is based on converting a time-domain signal into a frequence-domain representation in order to analyze the frequency components in the signal. Fourier transform is a method of converting a signal from the time domain to the frequency domain, which can decompose the different frequency components of the signal. The mathematical formula for the Fourier transform is as follows:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft}\, dt$$

$X(f)$ is a complex value in the frequency domain, representing the signal strength at frequency f, $x(t)$ is a time domain signal, $e^{-j2\pi ft}$ is a complex exponential function, corresponding to each frequency component. The Fourier transform decomposes the signal into the sum of sinusoidal waves of different frequencies and provides amplitude and phase information for each frequency component.

For digital signals in practical applications, we cannot compute a continuous Fourier transform, but instead use a discrete Fourier transform (DFT). DFT is used for sampled signals of finite length, and the formula is:

$$X[K] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$$

The DFT provides the frequency component of the signal over a limited frequency range.

Fast Fourier Transform (FFT) is an optimization algorithm of DFT, which reduces the complexity to O(NlogN) by division-and-conquer algorithm, greatly speeding up the calculation speed, so FFT is usually used in the actual frequency estimation.

The frequency estimation of FFT does not need the amplitude and phase information, but only needs to use the time series of the signal to estimate the frequency. FFT frequency estimation is suitable for processing single frequency sine wave and multi-frequency component superposition signals, which meets the general frequency estimation requirements. The main frequency components can be identified in the spectrogram, while the interference of random noise can be suppressed, and the accuracy of estimation can be improved.

As can be seen from the result table, the main frequency is 35MHz, indicating that the energy of the signal is mainly concentrated at that frequency, which may be the basic component of the signal or the main modulation frequency. The corresponding normalized amplitude is close to 0.5, indicating that the frequency component in the signal is relatively large. The sampling

Table 7 FFT frequency estimation results

Metric	Value
Sampling Interval	0.000000002
Sampling Rate	500000000
Signal Length	250001

Metric	Value
Dominant Frequency	34999860
Dominant Amplitude	0.496802953

frequency is 500MHz, much more than twice the main frequency (35MHz×2=70MHz), which conforms to the Nyquist sampling theorem and ensures that there is no aliasing problem at the main frequency. 250,000 sampling points, with a sampling interval of 2ns and a signal duration of about 0.5 seconds, provide enough time information to support stable frequency analysis.

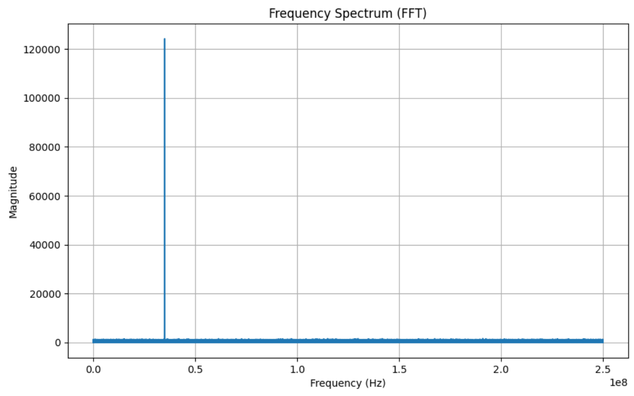


Figure 14 FFT Frequency Spectrum

According to the results of the frequency estimation of the FFT, the estimated main frequency is about 34.99986MHz, which can be considered to be about 35MHz. In the FFT spectrum diagram, the main frequency has a significant peak near 35MHz, indicating that this frequency component is the strongest in the signal, and other frequencies may be noise or minor components.

BlindSourceSeparation based on blind source separation (BSS) technology of frequency estimation, is separated from the mixed signal independent components to estimate the frequency of each signal source. The core of BSS technology is to use statistical characteristics to restore the mixed signal to the original signal source when the mixing mode is unknown. The BSS technique is used to separate the useful information in the mixed signal and then estimate the signal frequency. Independent Component Analysis (ICA) is one of the most commonly used algorithms in BSS, and its basic assumption is that the source signals are statistically independent and non-Gaussian.

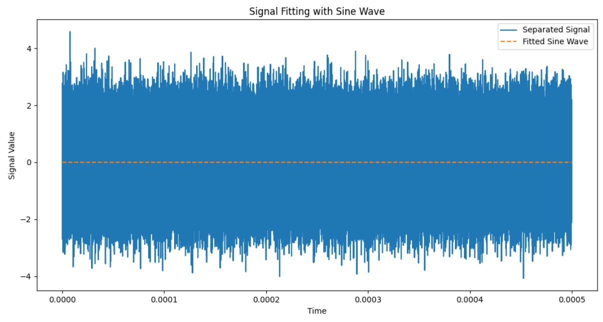
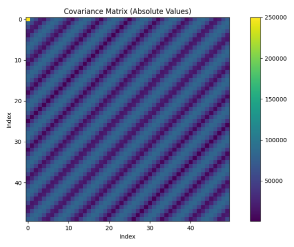


Figure 15 Siginal Fitting with Sine Wave

Although the separated signal contains noise, it shows a certain periodic structure, which indicates that some frequency components can still be identified.

MUSIC (MultipleSignalClassification) algorithm is a kind of high resolution frequency estimation method, especially suitable for analysis of signal spectrum in the noise of the larger environment, often used to estimate multiple narrowband signal frequency. MUSIC finds the main frequency by analyzing the covariance matrix of the signal and separating the signal component from the noise component.



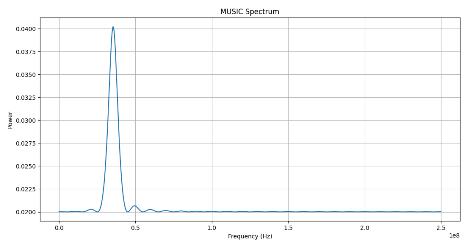


Figure 16 Signal information covariance matrix

Figure 17 MUSIC algorithm frequency estimation case graph

The structure of the covariance matrix shows the periodic and frequency characteristics of the signal, indicating that the signal has strong correlation and periodic components.

The spectrum shows a prominent peak at approximately 0.4×10^8 Hz (40 MHz), with low and stable power at other frequencies, indicating effective noise suppression by the MUSIC algorithm. The frequency estimate, computed using Python, is approximately 35.4 MHz, demonstrating the high resolution and reliability of the MUSIC method.

The autocorrelation method is different from other algorithms in that it does not rely on frequency domain conversion, but directly estimates the frequency based on the time domain characteristics of the signal. This method can be used as a supplementary method for frequency estimation after blind source separation, and is suitable for signals with strong periodicity. The formula for calculating the autocorrelation function is used to measure the similarity of the signal under different time delays (or hysteresis). For discrete signals, the formula for calculating the autocorrelation function is:

$$R_x(k) = \sum_{n=0}^{N-k-1} x(n) \cdot x(n+k)$$

$R_x(k)$ represents the **autocorrelation** value of the signal $x(n)$ under delay k, N is the length of the signal $x(n)$, and the **autocorrelation function** calculates the overlapping similarity of the signal under different delays. By multiplying and summing the signal point by point with its delayed version, the degree of similarity under different delays can be obtained. By analyzing the peak value of the **autocorrelation function**, the period T of the signal is obtained, and the frequency is the reciprocal of the period to obtain the frequency.

The 'np.correlate' function in python programming language is used to calculate the autocorrelation function, and the delay corresponding to the first significant peak is found by using the 'find_peaks' function, which is regarded as the period of the signal, and the main frequency of the signal is calculated by the reciprocal of the period, so as to achieve frequency estimation. The frequency of the calculated autocorrelation function is estimated in the following cases.

One period of the signal detected by the autocorrelation function is about 58 nanoseconds, and the main frequency of the signal is 17.24MHz, which is the frequency component with the most concentrated energy in the signal.

Table 8 Frequency estimation results of different methods

Method	Main Frequency Estimation (MHz)
FFT	34.99986 MHz
PSD	35.16 MHz
MUSIC	35.4 MHz
Autocorrelation function method	17.24 MHz

By preprocessing the signal data of period 3 (removing DC component, smoothing and filtering), and using FFT, PSD, autocorrelation function and MUSIC algorithm to estimate the frequency, the results show that the main frequency of the signal is concentrated around 35MHz. Each of these methods has its advantages and disadvantages in noise suppression, frequency resolution and signal type adaptability, but the overall results are consistent, which proves the reliability of frequency estimation.

4.4 Modeling and Solving Problem 4

The Kalman filter is a recursive algorithm that combines predictions and observations to estimate the system state. Its advantage is that it can effectively handle noisy signals and is a good method for calculating frequency.

The **Kalman filter** is a recursive algorithm that combines predictions and observations to estimate the system state. Its advantage is that it can effectively handle noisy signals and is a good method for calculating frequency.

$$\boldsymbol{x}_k = \begin{bmatrix} f_k \\ \dot{f}_k \end{bmatrix}$$

Where f_k is the frequency estimate at time step k, and $\dot{f}_k$ is the frequency rate of change. If the frequency changes slowly, the state transition equation is:

$$\boldsymbol{x}_k = F\boldsymbol{x}_{k-1} + \boldsymbol{W}_{k-1}$$

$$F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$$

Where $\boldsymbol{W}_{k-1}$ represents process noise, which follows a Gaussian distribution with a mean of zero, and the covariance matrix is Q. The observed signal value is denoted as: $y_k = H\boldsymbol{x}_k + v_k$ Where H represents the observation frequency component, and v_k is the observation noise, which follows a Gaussian distribution with a mean of zero. The **Kalman filter** algorithm mainly consists of two steps: prediction and update. In the prediction step, the following is typically used:

$$\boldsymbol{x}_k^- = F\boldsymbol{x}_{k-1}, \quad P_k^- = FP_{k-1}F^T + Q$$

Wherebmx_k is the predicted state estimate, and P_k^- is the predicted error covariance.In the update step, the following is generally used:

$$K_k = \frac{P_k^- H^T}{HP_k^- H^T + R}, \quad \boldsymbol{x}_k = \boldsymbol{x}_k^- + K_k\left(y_k - H\boldsymbol{x}_k^-\right), \quad P_k = (I - K_k H)\, P_k^-$$

Where I is the identity matrix.

In this experiment, we use the Kalman filter algorithm to estimate the signal frequency in aircraft laser speed measurement. This method is a recursive algorithm. In the process of selecting parameters, the noise covariance matrix Q of the state transition equation and the observation noise covariance matrix R will directly affect the estimation accuracy. We determine the final values of Q and R through experiments.

$$\mathbf{Q} = \begin{bmatrix} 1 \cdot 10^{-3} & 0 \\ 0 & 1 \cdot 10^{-3} \end{bmatrix}, \quad R = 1 \cdot 10^{-1}$$

After determining the process covariance matrix and noise covariance, we continue to calculate. We use Python to implement the algorithm and finally obtain a frequency of 56.25 MHz.

$$\hat{f}_k = \hat{f}_{k-1} + K_k * (z_k - H * \hat{f}_{k-1}) = 56.29\,\text{MHz}$$

To further verify the rationality of this value, we take the average of the last 30 frequency values and draw the Kalman filter estimated frequency change graph and FFT spectrogram.

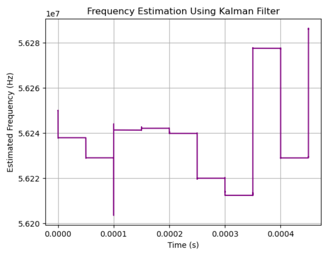


Figure 18 Kalman filter estimated frequency change graph

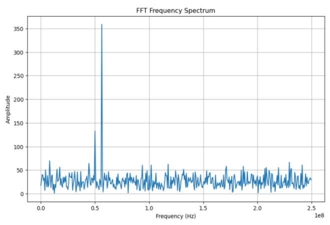


Figure 19 FFT spectrogram

The left figure shows the Kalman filter's estimated frequency, stable around 56.25 MHz with minor fluctuations. The right figure, from FFT analysis, confirms a main frequency peak near 56.25 MHz, aligning with the Kalman filter's result, demonstrating its accuracy in noisy environments.

Therefore, the frequency of the received signal during the fourth flight period is 56.25 MHz.

Wavelet transform (Wavelet Transform) can effectively estimate the frequency of intermittent signals. Wavelet transform can provide local time-frequency information, which is very helpful for irregularly spaced signals.

(1) Continuous Wavelet Formula: Used to transform the time-domain signal into the time-frequency domain

$$W(a,b) = \int_{-\infty}^{\infty} x(t)\psi^*_{a,b}(t)\,dt$$

Where $W(a,b)$ is the wavelet transform, $x(t)$ is the input signal, and

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi\left(\frac{t-b}{a}\right)$$

is the conjugate of the wavelet function.

(2) **Principal Frequency Extraction Formula:**

$$f_{\text{peak}} = \arg\max \int_t \left|W(\alpha(f),t)\right|^2\,dt$$

Considering using wavelet transform to process the intermittent data in the fourth flight stage of Attachment 1.

By bringing the intermittent data from Attachment 4 into the wavelet transform algorithm and using Python to implement it, we can get: is the center frequency of the wavelet, is 0.849, so the frequency is:

$$f_{\text{peak}} = \frac{f_c}{a_{\text{peak}} \cdot \Delta t} = 55.56\,\text{MHz}$$

To further verify our calculated result, we use the Wavelet Transform Power Spectrum and 3D wavelet energy spectrum to further determine the estimated frequency.

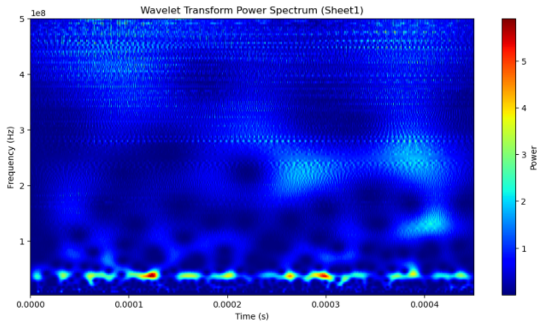


Figure 20 Wavelet Transform Power

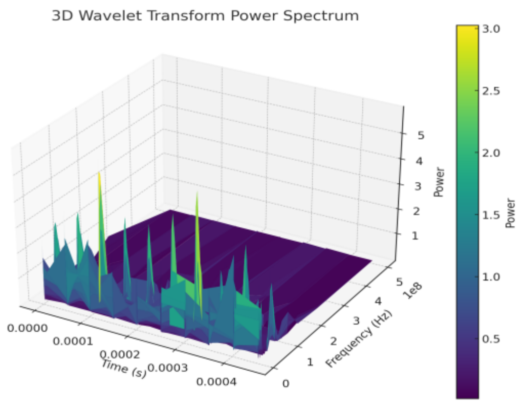


Figure 21 3D Wavelet Energy Spectrum

The 3D wavelet energy spectrum reveals that the signal's energy is concentrated near 55 MHz, with a stable and continuous distribution over time, verifying its time-domain stability. The main frequency during the fourth flight period is determined to be approximately 55.56 MHz.

To intuitively reflect the effect of the two methods on frequency estimation, we present the frequency estimates obtained by the two methods as follows:

Table 9 Frequency estimation comparison table

Method	Frequency Estimate (MHz)
Kalman Filter	56.25
Wavelet Transform	55.56

The estimated frequency by Kalman filter is 56.25 MHz, and the estimated frequency by wavelet transform is 55.56 MHz. Compared with wavelet transform, Kalman filter is more suitable for dynamic systems and can handle noise and uncertainty in time series data. While wavelet transform is more suitable for non-stationary signals and can provide local frequency information in different frequency ranges.

Sensitivity analysis helps understand how different parameters in the Kalman filter affect the frequency estimation results. Different parameters can have varying impacts, so we conduct a sensitivity analysis.

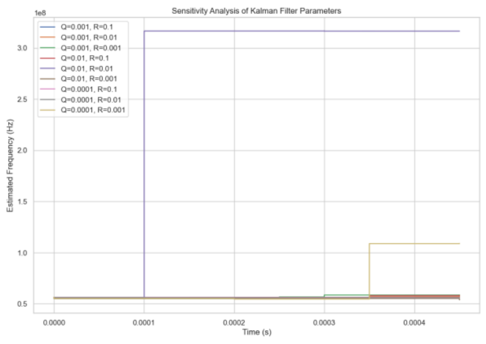


Figure 22 Kalman Sensitivity Analysis

Similarly, sensitivity analysis is performed for the wavelet transform: The frequency variation in the figure shows almost step-like abrupt changes, which may be due to the Kalman filter parameters being too sensitive or the initial state not being well set. This may require further optimization of the filter's initial conditions or reducing such drastic fluctuations by adjusting the values of Q and R. For the combination of Q=0.0001, R=0.1 and Q=0.0001, R=0.001, the frequency variation is more stable, showing a more reasonable estimation, indicating that under lower process noise, the Kalman filter can better balance prediction and measurement.

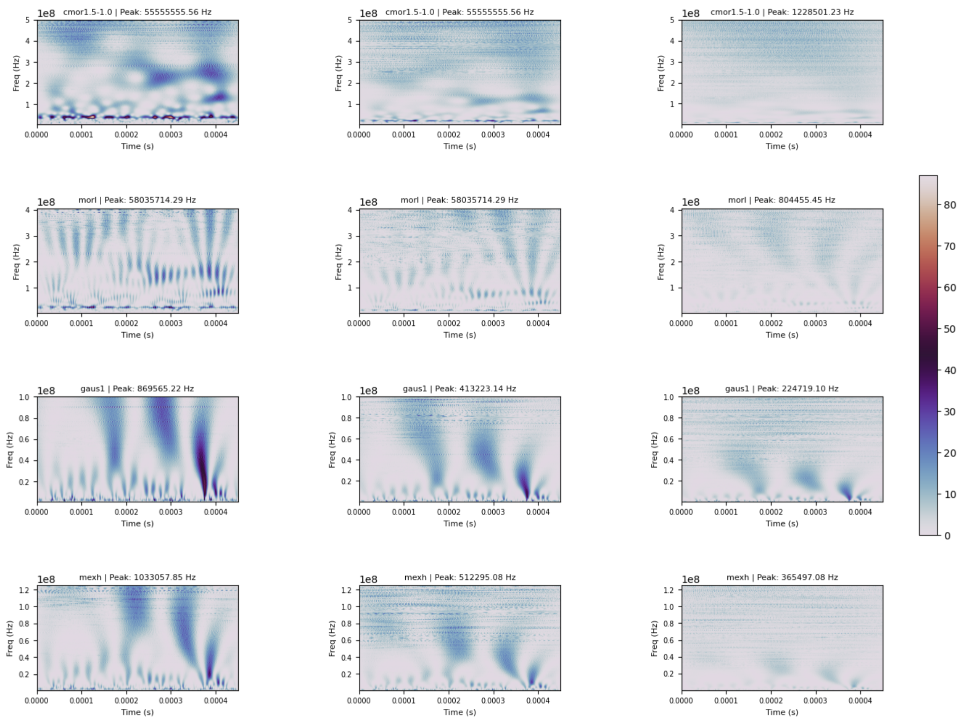


Figure 23 Wavelet Transform Sensitivity Analysis

The cmor1.5-1.0 wavelet is suitable for signals with strong periodicity and smooth frequency changes, while the mexh wavelet is better suited for signals with sharp frequency changes, as it captures sudden frequency components more effectively. Additionally, the larger the scale, the lower the frequency resolution of the wavelet, which is more suitable for detecting low-frequency components. Smaller scales offer higher resolution, making them more suitable for detecting high-frequency signals.

5. Strengths and Weakness

5.1 Strengths

One can try to reduce the effect of noise by using a band-pass filter or an adaptive filter before estimating the frequency. Another approach is to use an autoregressive model (e.g., an AR model) to fit the signal and estimate the frequency from the parameters of the model. More advanced signal processing techniques, such as wavelet transforms or empirical mode decomposition (EMD), can also be considered to separate the signal from the noise. Compare the frequency estimation results for the white noise and non-white noise cases and analyze the effect of the accuracy of HMM state decoding on the frequency estimation.

Kalman filter is a state estimation method that continuously updates the state estimate value when dealing with dynamic systems, so it has higher accuracy and is suitable for linear systems.Wavelet transform can process non-stationary data and can effectively process non-stationary signals and provide time-frequency localized analysis.

5.2 Weakness

FFT and PSD methods are sensitive to noise, and the frequency resolution is limited by signal length and sampling frequency, so it is difficult to resolve the approximate frequency components. The MUSIC algorithm is complex in calculation, sensitive to the selection of order of covariance matrix, and has limited application scope. The method of autocorrelation function has poor precision and is difficult to process multi-frequency signals when the signal periodicity is not obvious or the noise is large. Different methods have their own characteristics and should be selected or combined according to the signal characteristics and actual needs.

6. Conclusion

In summary, the first question examines the differences in computational formulas for mixed variable data in modeling statistical features. Basic statistics like mean, max/min, variance, skewness, and kurtosis are computed to describe the data's distribution and shape. Fourier transform is applied for spectral analysis, and additional calculations reveal abnormal amplitude fluctuations and temporal irregularities. The noise sequence is identified as an independent, normally distributed sequence with a mean of 0.00584 and variance of 3.9904, resembling white noise characteristics.

For the second problem, time series data of received signals are processed to identify outliers using the Isolated Forest algorithm. To analyze the data further for small shifts or trend changes, the sliding window technique and CUSUM method are used to detect the frequency period. The Hidden Markov Model is then applied to estimate the frequency of the non-noise portion, with the largest amplitude in the FFT results indicating a frequency of 41 × 106 Hz.

For the third problem ,consistently estimated the signal's main frequency at approximately 35 MHz through various methods (FFT, PSD, MUSIC), demonstrating reliable accuracy after effective preprocessing. Each method's strengths confirmed the robustness of the frequency estimation process.

The paper discusses frequency estimation of intermittent signals, comparing Kalman filtering and wavelet transform methods. Kalman filtering predicts 56.29 MHz, while wavelet transform analyzes to 55.56 MHz. Kalman filtering is recursive and complex, fitting dynamic systems, whereas wavelet transform suits non-stationary signals but is parameter-sensitive. Sensitivity analysis shows parameter effects, with the choice between methods depending on application and data specifics.

References

[1] Kouakou H., Goulart M. D. H. J., Vitale R., et al. On-the-fly spectral unmixing based on Kalman filtering[J]. *Chemometrics and Intelligent Laboratory Systems*, 2024, 255:105252.

[2] Peng J., Wu D., Yao P., et al. Schuler period oscillation error suppression for inertial navigation systems based on reverse navigation and wavelet transforms[J]. *Measurement*, 2025, 242(PD):115862.

[3] Shu S ,Lai J ,Wang Y , et al.High precision phase difference calculation based on adaptive mixed-radix All-phase FFT for Tokamak plasma electron density measurement[J].Journal of Instrumentation,2024,19(10):P10011-P10011.

[4] Garrison H L ,Mackey F D ,Shih H Y , et al.nifty-ls: Fast and Accurate Lomb–Scargle Periodograms Using a Non-uniform FFT[J].Research Notes of the AAS,2024,8(10):250-250.

[5] Rasmussen N ,Rizk R ,Matoo O , et al.DeepWhaleNet: Climate Change-Aware FFT-Based Deep Neural Network for Passive Acoustic Monitoring[J].International Journal of Pattern Recognition and Artificial Intelligence,2024,(prepublish):

[6] Shu S ,Lai J ,Wang Y , et al.High precision phase difference calculation based on adaptive mixed-radix All-phase FFT for Tokamak plasma electron density measurement[J].Journal of Instrumentation,2024,19(10):P10011-P10011.

Appendix

Listing 1: Part of the code for questions one and two

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde, kurtosis, skew, shapiro
from scipy.fft import fft, fftfreq
from statsmodels.tsa.stattools import acf, pacf, adfuller, q_stat
from sklearn.ensemble import IsolationForest
from hmmlearn import hmm

plt.rcParams['font.sans-serif'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False

# File path
file_path = 'C:/Users/25833/Desktop/noise_data.xlsx'
data = pd.read_excel(file_path)

# Extracting columns
times = data['Time']
noises = data['Noise']
xt = data['X(t)']
signal = data['Signal']

#%% Plot noise over time
fig = plt.figure(figsize=(10, 6), facecolor='white', dpi=150)
ax = fig.add_subplot(111, facecolor='gray')
ax.plot(times, noises, linestyle='-', color='white')
ax.set_xlabel('Time', color='black', fontsize=24)
ax.set_ylabel('Noise', color='black', fontsize=24)
ax.spines['bottom'].set_color('black')
ax.spines['left'].set_color('black')
for spine in ax.spines.values():
spine.set_linewidth(2)
plt.margins(x=0, y=0)
ax.tick_params(axis='both', which='major', labelcolor='black',
    colors='black', labelsize=15)
ax.grid(True, color='lightgray', linestyle='--', linewidth=0.5)
plt.show()
```

```
#%% KDE for X(t)
noise_values = xt
kde = gaussian_kde(noise_values, bw_method='scott')
x = np.linspace(min(noise_values), max(noise_values), 1000)
pdf_values = kde(x)
plt.figure(figsize=(10, 6), dpi=150)
plt.plot(x, pdf_values, color='green', lw=2, label='X(t) PDF')
plt.fill_between(x, pdf_values, color='green', alpha=0.3)
plt.title('Probability Density Function of X(t)')
plt.xlabel('X(t) Value', fontsize=24)
plt.ylabel('Density', fontsize=24)
plt.legend()
plt.show()

#%% Signal statistics
df = pd.DataFrame(data)
mean_value = df['Signal'].mean()
median_value = df['Signal'].median()
std_dev = df['Signal'].std()
range_value = df['Signal'].max() - df['Signal'].min()
cv = std_dev / mean_value
skewness = skew(df['Signal'])
kurtosis_value = kurtosis(df['Signal']) - 3
print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Standard Deviation: {std_dev}")
print(f"Range: {range_value}")
print(f"Coefficient of Variation: {cv}")
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurtosis_value}")

#%% ACF and PACF plots for noise
lag_acf = acf(noises, nlags=20)
lag_pacf = pacf(noises, nlags=20, method='ols')
plt.figure(figsize=(12, 6), dpi=150)
plt.subplot(211)
plt.plot(lag_acf)
plt.title('Autocorrelation Function (ACF) of Noise')
plt.subplot(212)
plt.plot(lag_pacf)
plt.title('Partial Autocorrelation Function (PACF) of Noise')
plt.tight_layout()
plt.show()

#%% FFT of noise
fft_noise = fft(noises)
frequencies = fftfreq(len(noises))
plt.figure(figsize=(12, 6), dpi=150)
plt.plot(frequencies, np.abs(fft_noise))
plt.title('Frequency Spectrum of Noise')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.tight_layout()
plt.show()

#%% Correlation heatmap
correlation_matrix = np.corrcoef(signal, noises)
plt.figure(figsize=(6, 6), dpi=150)
plt.imshow(correlation_matrix, cmap='hot', interpolation='nearest')
plt.colorbar()
plt.title('Correlation Heatmap between Signal and Noise')
plt.xticks([])
plt.yticks([])
plt.show()

#%% Ljung-Box test
lb_value, p_value = q_stat(lag_acf, len(noises))
plt.figure(figsize=(12, 6), dpi=150)
plt.plot(lb_value[1:], p_value[1:], 'o-')
plt.title('Ljung-Box Test for Noise')
plt.xlabel('Lag')
plt.ylabel('p-value')
plt.axhline(y=0.05, color='red', linestyle='--')
plt.show()

#%% ADF test
result = adfuller(noises)
```

```
print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:')
for key, value in result[4].items():
print(f'\t{key}: {value:.3f}')

#%% Shapiro-Wilk test
stat, p_value = shapiro(noises)
print(f"Shapiro-Wilk p-value: {p_value}")
if p_value > 0.05:
print("Noise follows a normal distribution.")
else:
print("Noise does not follow a normal distribution.")

mean_noise = data['Noise'].mean()
std_noise = data['Noise'].std()
print(f"Mean of 'Noise': {mean_noise}")
print(f"Standard Deviation of 'Noise': {std_noise}")

#%% Outlier detection
file_path = 'C:/Users/25833/Desktop/period2.xlsx'
data = pd.read_excel(file_path)
times = data['Received Signal Time'].values
signals = data['Received Signal Value'].values
iso_forest = IsolationForest(contamination=0.01)
outliers = iso_forest.fit_predict(signals.reshape(-1, 1))

plt.figure(figsize=(14, 7), dpi=150)
plt.plot(times, signals, label='Received Signal')
plt.scatter(times[outliers == -1], signals[outliers == -1], color='red',
    label='Outliers')
plt.xlabel('Time')
plt.ylabel('Signal Value')
plt.title('Received Signal with Outliers')
plt.legend()
plt.grid(True)
plt.show()

#%% FFT for received signals
```

```
N = len(signals)
T = times[-1] - times[0]
yf = fft(signals)
xf = fftfreq(N, T / N)
plt.figure(figsize=(14, 7), dpi=150)
plt.plot(xf[:N//2], 2.0/N * np.abs(yf[:N//2]))
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('Frequency Spectrum of Received Signal')
plt.grid(True)
plt.show()

main_freq_idx = np.argmax(2.0/N * np.abs(yf[:N//2]))
main_freq = xf[main_freq_idx]
print(f"Estimated Main Frequency: {main_freq:.6f} Hz")

#%% HMM frequency estimation
signals = (signals - np.mean(signals)) / np.std(signals)
n_components = 2
model = hmm.GaussianHMM(n_components=n_components, covariance_type="diag",
    n_iter=1000)
model.fit(signals.reshape(-1, 1))
logprob, state_sequence = model.decode(signals.reshape(-1, 1),
    algorithm="viterbi")
signal_indices = np.where(state_sequence == 1)[0]
extracted_signal = signals[signal_indices]

if len(extracted_signal) < 100:
raise ValueError("Extracted signal segment is too short for frequency
    estimation.")

N = len(extracted_signal)
yf = fft(extracted_signal)
xf = fftfreq(N, d=times[1] - times[0])
frequencies = xf[:N//2]
amplitudes = 2.0/N * np.abs(yf[:N//2])
plt.plot(frequencies, amplitudes)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('Frequency Spectrum of Extracted Signal')
plt.grid(True)
plt.show()

main_freq_idx = np.argmax(amplitudes)
main_freq = frequencies[main_freq_idx]
print(f"Estimated Main Frequency: {main_freq:.6f} Hz")
```

Listing 2: Part of the code for questions three

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt, welch

# Read the Excel file and extract data
file_path = "D:/Personal/Desktop/New Folder/Attachment 1.xlsx" # Ensure the
    file path is correct
data = pd.read_excel(file_path, sheet_name='Flight Period 3')
time = data['Received Signal Time'].values
signal = data['Received Signal Value'].values

# Parameter settings
sampling_interval = time[1] - time[0]
sampling_rate = 1 / sampling_interval # Sampling frequency

# 1. Time-domain analysis of the original signal
plt.figure(figsize=(10, 6))
plt.plot(time, signal, label='Original Signal')
plt.title('Original Signal in Time Domain')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

# 2. Remove DC component
signal_dc_removed = signal - np.mean(signal)
plt.figure(figsize=(10, 6))
plt.plot(time, signal_dc_removed, label='DC Removed Signal')
plt.title('Signal after Removing DC Component')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

# 3. Smoothing (moving average)
window_size = 101 # Window size
smoothed_signal = np.convolve(signal_dc_removed, np.ones(window_size) /
    window_size, mode='same')
plt.figure(figsize=(10, 6))
plt.plot(time, smoothed_signal, label='Smoothed Signal', color='green')
plt.title('Signal after Smoothing')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

# 4. Filtering (low-pass filter)
def butter_lowpass_filter(data, cutoff, fs, order=4):
nyquist = 0.5 * fs
normal_cutoff = cutoff / nyquist
b, a = butter(order, normal_cutoff, btype='low', analog=False)
y = filtfilt(b, a, data)
return y

cutoff_frequency = 50e6 # 50 MHz
filtered_signal = butter_lowpass_filter(smoothed_signal, cutoff_frequency,
    sampling_rate)

plt.figure(figsize=(10, 6))
plt.plot(time, filtered_signal, label='Filtered Signal', color='red')
plt.title('Signal after Lowpass Filtering')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

# 5. Frequency analysis (FFT)
fft_values = np.fft.fft(filtered_signal)
```

```
fft_frequencies = np.fft.fftfreq(len(filtered_signal), sampling_interval)
fft_amplitude = np.abs(fft_values) / len(filtered_signal)

# Retain only positive frequencies
positive_frequencies = fft_frequencies[:len(fft_frequencies) // 2]
positive_amplitude = fft_amplitude[:len(fft_amplitude) // 2]

plt.figure(figsize=(10, 6))
plt.plot(positive_frequencies, positive_amplitude, label='FFT Spectrum')
plt.title('Frequency Spectrum')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.legend()
plt.grid()
plt.show()

# 6. Power spectral density (PSD) analysis
frequencies, psd = welch(filtered_signal, fs=sampling_rate, nperseg=1024)
plt.figure(figsize=(10, 6))
plt.semilogy(frequencies, psd, label='Power Spectral Density (PSD)')
plt.title('Power Spectral Density (PSD)')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power/Frequency (dB/Hz)')
plt.legend()
plt.grid()
plt.show()

# 7. Signal-to-noise ratio (SNR) analysis
signal_power = np.sum(filtered_signal ** 2) / len(filtered_signal)
noise_power = np.sum((filtered_signal - smoothed_signal) ** 2) /
    len(filtered_signal)
snr = 10 * np.log10(signal_power / noise_power)
print(f"Signal-to-Noise Ratio (SNR): {snr:.2f} dB")

# Summary of signal characteristics
results = {
   'Metric': ['Sampling Interval', 'Sampling Rate', 'Signal Length', 'Signal
      Power', 'Noise Power', 'SNR'],
   'Value': [sampling_interval, sampling_rate, len(signal), signal_power,
      noise_power, snr],
   'Unit': ['Seconds', 'Hz', 'Samples', 'Power', 'Power', 'dB']
}
results_df = pd.DataFrame(results)

# Save the results table to an Excel file
output_file = "D:/Personal/Desktop/Signal_Analysis_Results.xlsx"
results_df.to_excel(output_file, index=False)
print(f"Results saved to: {output_file}")

# Print results table
print("\nSignal Analysis Summary:")
print(results_df)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import FastICA

# Read the Excel file
file_path = "D:/Personal/Desktop/New Folder/Attachment 1.xlsx" # Ensure the
    file path is correct
data = pd.read_excel(file_path, sheet_name='Flight Period 3')

# Extract time and signal data
time = data['Received Signal Time'].values
signal = data['Received Signal Value'].values

# Use ICA to separate signals
X = np.vstack([signal]).T
ica = FastICA(n_components=1)
S_ = ica.fit_transform(X)
separated_signal = S_.flatten()

# Parameter settings
sampling_rate = 1 / (time[1] - time[0]) # Calculate the sampling rate
frequency_range = np.linspace(0, sampling_rate / 2, 5000) # Frequency search
    range
order = 50 # Covariance matrix order
```

```
# Compute covariance matrix
def covariance_matrix(signal, order):
"""
Generate the covariance matrix of the signal
"""
N = len(signal)
covariance = np.zeros((order, order), dtype=complex)
for i in range(order):
for j in range(order):
if i + j < N:
covariance[i, j] = np.dot(signal[:N - i - j], signal[i + j:])
return covariance

cov_matrix = covariance_matrix(separated_signal, order)

# Print covariance matrix
print("Covariance Matrix:")
print(cov_matrix)

# Visualize covariance matrix
plt.figure(figsize=(8, 6))
plt.imshow(np.abs(cov_matrix), cmap='viridis')
plt.colorbar()
plt.title('Covariance Matrix (Absolute Values)')
plt.xlabel('Index')
plt.ylabel('Index')
plt.show()

# Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Separate signal and noise subspaces
eigenvectors = eigenvectors[:, np.argsort(np.abs(eigenvalues))[::-1]]
noise_space = eigenvectors[:, 1:] # Assume one signal source

# Compute MUSIC spectrum
def music_spectrum(noise_space, frequency_range, sampling_rate):
spectrum = []
for freq in frequency_range:
```

```
omega = np.exp(-2j * np.pi * freq * np.arange(order) / sampling_rate)
P = np.abs(1 / (omega.conj().T @ noise_space @ noise_space.conj().T @
    omega.T))
spectrum.append(P)
return np.array(spectrum)

spectrum = music_spectrum(noise_space, frequency_range, sampling_rate)

# Plot MUSIC spectrum
plt.figure(figsize=(10, 6))
plt.plot(frequency_range, spectrum)
plt.title('MUSIC Spectrum')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power')
plt.grid()
plt.show()

# Find the dominant frequency
dominant_frequency = frequency_range[np.argmax(spectrum)]
print(f"Estimated Dominant Frequency using MUSIC: {dominant_frequency} Hz")
```

Listing 3: Part of the code for questions four

```
  import pandas as pd
  import matplotlib.pyplot as plt

  # Load data into DataFrame df, including 'Received Signal Time' column
  df = pd.read_excel('C:/Users/DELL/Desktop/period_four.xlsx') # Read data

  # Print the first few rows of the original data and the data type of the
      'Received Signal Time' column to check the data structure
  print("Original 'Received Signal Time' column data type:", df['Received
      Signal Time'].dtype)
  print("First few rows of original 'Received Signal Time' column data:")
  print(df['Received Signal Time'].head())

  # Convert 'Received Signal Time' from float to timedelta format (in seconds)
  df['Received Signal Time'] = pd.to_timedelta(df['Received Signal Time'],
      unit='s')
```

```
# Calculate time difference (difference between adjacent time points)
df['time_diff'] = df['Received Signal Time'].diff()

# Print 'time_diff' column statistics to check the time differences
print(f"Time difference statistics:\n{df['time_diff'].describe()}")

# Visualize time difference distribution
plt.figure(figsize=(10, 6))
plt.hist(df['time_diff'].dt.total_seconds() * 1e9, bins=50,
    color='skyblue', edgecolor='black')
plt.title('Time Difference Distribution (in nanoseconds)', fontsize=14)
plt.xlabel('Time Difference (ns)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(True)
plt.show()

# Set time gap threshold (e.g., 10 nanoseconds)
threshold = pd.Timedelta(seconds=10e-9) # 10 nanoseconds

# Find gaps greater than the threshold
long_gap = df[df['time_diff'] > threshold]

# Output data points with time gaps exceeding the threshold
print(f"Data points with time gaps exceeding {threshold}:")
print(long_gap)

# Data density: Calculate the actual sampling time and the total time span
    of the signal
total_time = df['Received Signal Time'].max() - df['Received Signal
    Time'].min()
sampled_time = df['time_diff'].sum()

# Ensure total_time and sampled_time are both Timedelta types
total_time = pd.to_timedelta(total_time, unit='s') # Convert total_time to
    Timedelta type
sampled_time = pd.to_timedelta(sampled_time, unit='s') # Convert
    sampled_time to Timedelta type
```

```
# Data density: The proportion of sampling time to total time
data_density = (sampled_time / total_time) * 100 if total_time >
    pd.Timedelta(0) else 0

# Output total data time span, actual sampling time, and data density
print(f"Total data time span: {total_time}")
print(f"Actual sampling time: {sampled_time}")
print(f"Data density: {data_density:.2f}%")

# Visualize the time of receiving the signal
plt.figure(figsize=(10, 6))
plt.plot(df['Received Signal Time'], df['time_diff'].dt.total_seconds() *
    1e9, marker='o', linestyle='-', color='blue')
plt.title('Received Signal Time vs. Time Difference', fontsize=14)
plt.xlabel('Received Signal Time', fontsize=12)
plt.ylabel('Time Difference (ns)', fontsize=12)
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from numpy.fft import fft, fftfreq

# Read data
file_path = "C:/Users/DELL/Desktop/period_four.xlsx"
data = pd.read_excel(file_path)

time = data['Received Signal Time']
signal = data['Received Signal Value']

# Calculate time interval
dt = time[1] - time[0]

# Frequency analysis: Use FFT for preliminary frequency estimation of the
    signal
N = len(signal)
yf = fft(signal)
xf = fftfreq(N, dt)[:N//2]
```

```
initial_frequency = xf[np.argmax(np.abs(yf[:N//2]))] # Frequency
    corresponding to the maximum amplitude
print(f"Initial Frequency from FFT: {initial_frequency} Hz")

# State space model for the Kalman filter
def kalman_filter(signal, dt, initial_frequency):
n = len(signal)

# Initial state estimate
x = np.array([initial_frequency, 0]) # Initial frequency and rate of change
    of frequency
P = np.eye(2) * 1e3 # Initial error covariance

# Process noise covariance matrix
Q = np.array([[1e-3, 0], [0, 1e-3]]) # Adjust process noise

# Observation noise covariance matrix
R = 1e-1 # Increase observation noise

# Store estimation results
frequencies = []

# Kalman filter process
for t in range(1, n):
# Prediction
x_pred = np.array([x[0] + x[1] * dt, x[1]]) # Predict the frequency and
    rate of change of frequency at the next time step
P_pred = P + Q # Predicted error covariance

# Update
y = signal[t] - np.sin(2 * np.pi * x_pred[0] * time[t] + x_pred[1]) #
    Measurement residual
S = R + np.cos(2 * np.pi * x_pred[0] * time[t])**2 * P_pred[0, 0] #
    Residual covariance
K = P_pred[0, 0] / S # Kalman gain

# Update state estimate
x[0] = x_pred[0] + K * y
x[1] = x_pred[1] # Rate of change of frequency remains unchanged
```

```
# Update error covariance
P = P_pred - K * S * K

# Store estimated frequency
frequencies.append(x[0])

return frequencies

# Use the Kalman filter for frequency estimation
frequencies = kalman_filter(signal, dt, initial_frequency)

# Smoothing: Take the average of the last 30 frequency values
smoothed_frequency = np.mean(frequencies[-30:])
print(f"Smoothed Estimated Frequency: {smoothed_frequency:.2f} Hz")

# Plot frequency change graph
plt.plot(time[1:], frequencies, color='purple')
plt.xlabel("Time (s)")
plt.ylabel("Estimated Frequency (Hz)")
plt.title("Frequency Estimation Using Kalman Filter")
plt.grid(True)
plt.show()

# Plot FFT spectrum
N = len(signal)
yf = fft(signal)
xf = fftfreq(N, dt)[:N//2]

plt.figure(figsize=(10, 6))
plt.plot(xf, np.abs(yf[:N//2]))
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.title("FFT Frequency Spectrum")
plt.grid(True)
plt.show()
```