Problem Chosen	2024 ShuWei Cup Summary Sheet	Team Control Number
C		2024111130921

Through the Atmospheric Veil: A Multidimensional Approach to Enhancing Pulsar Timing Precision

1 Summary

With the rapid development of space technology and the increasing demand for high-precision time measurement, **pulsar timing technology** plays an increasingly important role in time scale maintenance and synchronization. **Atmospheric delay effect** The accurate measurement of pulse arrival time (TOA) poses a major challenge. Especially under high-frequency observation and small elevation conditions, the effect of atmospheric delay is more significant.

For problem 1 , we perform **de-trending processing on the** original data , and we find that **sixth-order polynomial** fitting can remove the long-term trend of the data without adding the complexity of the model, and obtain a stationary residual sequence. A parameterized red noise model is established through the **power spectral density model** , and a **differential evolution algorithm is used** for **global search** and **Nelder-Mead algorithm** for further **local search** optimization of model parameters , $P_0 = 2.34,\ f_c = 1.10E-03,\ q = 3.88$. The final model has a goodness of fit R^2 **of 0.9547** , capturing the red noise characteristics in the data.

For problem 2 , we constructed the **SARIMA** and **LSTM models** for **short-term** and **long-term prediction** of the original data source , and compared the two models . By calculating the **prediction residuals** of the two models and **the Diebold-Mariano test** (DM statistic: -13.6360, p-value: 0.006) , the results show that the prediction ability of the LSTM model is better than that of the SARIMA model , indicating that for the pulsar in the problem .

For problem , under **high frequency** conditions above 20GHz, wet delay can be almost ignored. We use the frequency **dependence formula** $\Delta\tau(f)$and $f^2$are inversely proportional, $f$the larger the delay, the lower the wet delay proportion is ignored, and the dry delay is dominant. Through the **Saastamoinen model** , the total atmospheric delay is preliminarily calculated, and the calculation formula for dry and wet delay is obtained by consulting relevant information. The dry and wet delay is separated from the total atmospheric delay, and the **Herring mapping function** is used to obtain the correction factor of dry and wet delay $m(\epsilon)$The dry delay and wet delay are corrected, and the accuracy of the total delay under high frequency is improved. The total delay is calculated and limited to less than 7.69 nanoseconds by the basic expression of the inverse refractive delay.

For problem 4 , in order to improve the accuracy of TOA, we establish an atmospheric delay model under small elevation angle and expand the available observation range. Firstly, through the **path integration formula** of atmospheric delay, the model foundation is established, combined with the **Saastamoinen model** , the dry and wet delays are calculated separately to obtain the atmospheric delay in the zenith direction. With the help of the **Herring model mapping function** , the delay in the zenith direction is mapped to the small elevation

angle. Finally, through the **hierarchical integration model** and **frequency dependence analysis** , the model is further optimized and the accuracy is improved to obtain high-precision low elevation atmospheric delay data.

Keywords: pulsar timing , atmospheric delay , power spectral density model, SARIMA, LSTM, Saastamoinen model, Herring mapping function

Contents

1 Summary 1
2 Introduction 5
2.1 Problem Background 5
2.2 Restatement of the Problem 5
2.3 Our Work 5
3 Assumptions and Justifications 6
4 Notations 6
5 Data preprocessing 7
5.1 Original data source import and time domain analysis 7
5.2 Stability test 7
5.3 Time Series Decomposition 8
6 Model establishment and solution of problem 1 9
6.1 Problem analysis 9
6.2 Data castration 9
6.2.1 Removal of long-term trends 9
6.2.2 Polynomial castration result 9
6.3 Frequency domain analysis and power spectral density calculation 10
6.4 Establish PSD model 10
6.5 Calculation of Power Spectral Density (PSD) 11
6.6 Parameter optimization 12
6.7 Model Evaluation 13
6.8 Result of solution 13
7 Model establishment and solution for problem 2 14
7.1 Problem exploration 14
7.2 The Model Establishment 14
7.2.1 SARIMA Model 14
7.2.2 LSTM Model 15
7.3 Model Solution and Result Analysis 16
7.3.1 Model comparison and testing 16
8 Model establishment and solution for problem 3 18
8.1 Problem analysis 18
8.2 Establishment of high-frequency atmospheric delay model 18
8.2.2 Exclusion of the influence of wet delay 19

8.2.3 Calculation Model of Total Time Delay 20
8.2.4 Delay correction 21
8.3 Analysis of Results 22
9 the model establishment and solution of problem four 23
9.1 Problem restatement and analysis 23
9.2 question restated 23
9.3 Problem Analysis 23
9.4 Model establishment 23
9.5 Saastamoinen model 24
9.6 Herring Model 24
9.7 Hierarchical Integral Model 25
9.8 frequency dependence 26
9.9 Model Solution and Analysis 26
10 Conclusion 28
References 29
Appendices 30

2 Introduction

2.1 Problem Background

In the application of pulsar time, the pulse arrival time (TOA) is the most critical consideration factor. However, TOA is affected by various delay effects, among which atmospheric delay is an important aspect. At typical observation frequencies, significant changes in the total electron content in the atmosphere may cause fluctuations in TOA from 10 nanoseconds to hundreds of nanoseconds, which has an undeniable impact on the accuracy of pulsar time and highlights the necessity of precise processing of atmospheric delay in related research

2.2 Restatement of the Problem

The question requires us to establish a mathematical model based on the background of the above question and the relevant data provided in the attachment to solve the following problems.

Problem 1: Consider simulating the pulsar timing noise in Figure 2 with a functional model, aiming for a model fit of 95% or higher. The required data for modeling can be found in Attachment 1. The data relationships that can be referred to and not used include: the observed frequency of the pulsar is 1540 MHz in the radio band with a bandwidth of 320 MHz, the RMS value for MJD 52473 to 56081 is 75268.376 μs, and for MJD 52473 to 56646, the RMS value is 78502.322 μs. It is generally assumed that the intensity of red noise is proportional to the RMS value, though not equal.

Problem 2: Consider making short-term (ranging from a few days to one month) and long-term (ranging from several months to a few years) forecasts on the future trend of the pulsar timing noise in Figure 2, The required data for forecasting validation can be found in Attachment 1.

Problem 3: Consider modeling the refractive time delay for radio observation frequencies above 20 GHz, ensuring that the zenith delay is less than or equal to 7.69ns.

Problem 4: Consider modeling the atmospheric delay for observations with small elevation angles (10 degrees or less) to improve TOA accuracy. Please provide your model and also describe your considerations and the achievable goals

2.3 Our Work

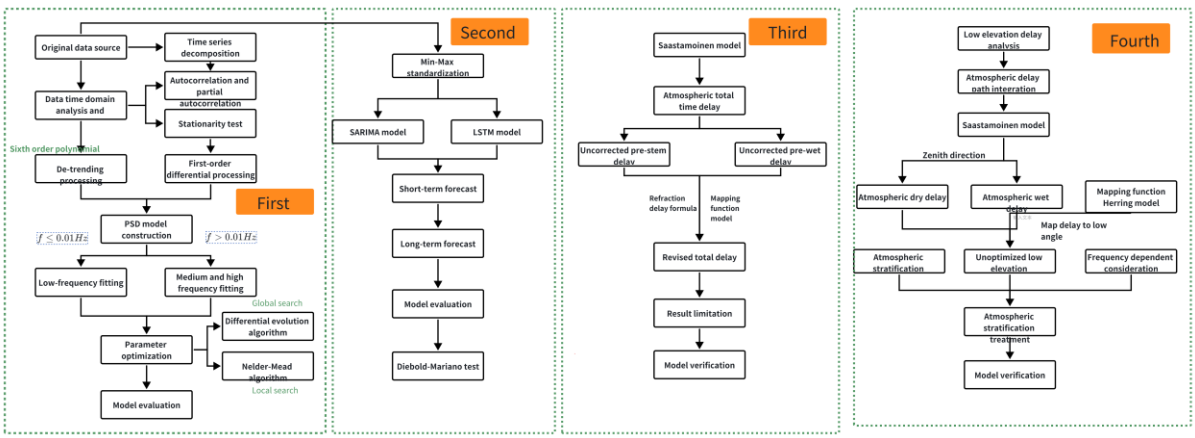


Figure 1: Our Work

3 Assumptions and Justifications

1. The pulsar timing noise can be accurately described using a power spectral density (PSD) model, with red noise characteristics dominating at low frequencies and attenuating at high frequencies.
2. The atmospheric delay can be decomposed into dry and wet components, with the dry delay being the dominant factor, especially at high frequencies (above 20 GHz).
3. Time series data exhibit seasonality, trend, and random components, which can be captured through differencing and autoregressive integrated moving average (ARIMA) processes, as well as deep learning models such as Long Short-Term Memory (LSTM) networks.
4. The zenith delay can be mapped to arbitrary elevation angles using mapping functions, with the accuracy of these functions depending on empirically determined parameters.
5. For high-frequency observations (above 20 GHz), the wet delay component can be neglected, and the dry delay becomes the primary factor in atmospheric delay calculations.

4 Notations

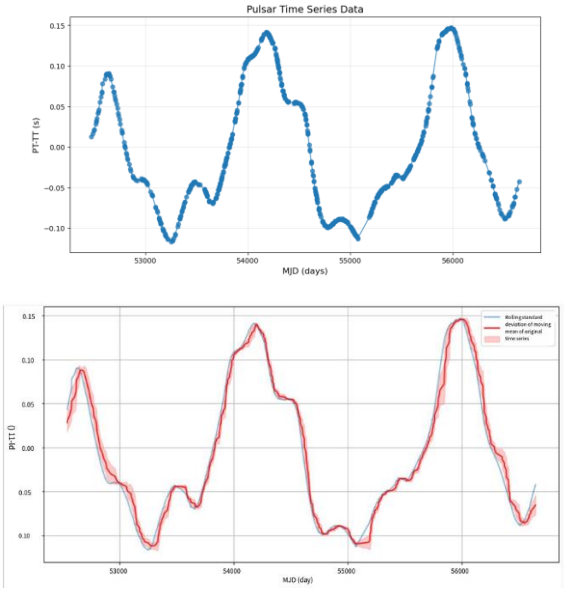
The key mathematical notations used in this paper are listed in Table 1.

Table 1: Notations used in this paper

Symbol	Description
t_i	Time Series (Modified Julian Day, MJD)
y_i	Raw PT-TT time difference data
$T(t)$	Polynomial function trend term
$a(t)$	Polynomial function undetermined coefficient
r_i	Residual sequence after de-trending
$Z(f)$	Sequence of complex random variables
$z(t)$	Noise sequence in time domain
P_0	Scale factor of noise intensity
f_c	Inflection frequency
q	Spectral index
$P(f)$	Power spectral density at f frequency
f	Electromagnetic wave frequency
$\Delta\tau$	Total delay
$m(\epsilon)$	Mapping function

5 Data preprocessing

5.1 Original data source import and time domain analysis



Time series chart, observe the trend of PT-TT over time, observe the overall trend and volatility and Visualization of Moving Average and Rolling Standard Deviation for Time Series Analysis.

5.2 Stability test

Stationarity is an important prerequisite assumption for time series analysis. If a time series is stationary, its mean and variance are constant in time, and the autocovariance is only related to the time interval, not to time itself. For example, the white noise series is a stationary series with a mean of 0, a constant variance, and a covariance of 0 between any different moments.

Unit root test (ADF test): This is one of the most commonly used methods. Its Null hypothesis is that the sequence has a unit root, that is, the sequence is non-stationary. By constructing a regression model containing sequence lag terms, the test statistic is calculated and compared with the corresponding critical value. If the test statistic is less than the critical value, the Null hypothesis is rejected and the sequence is considered stationary.

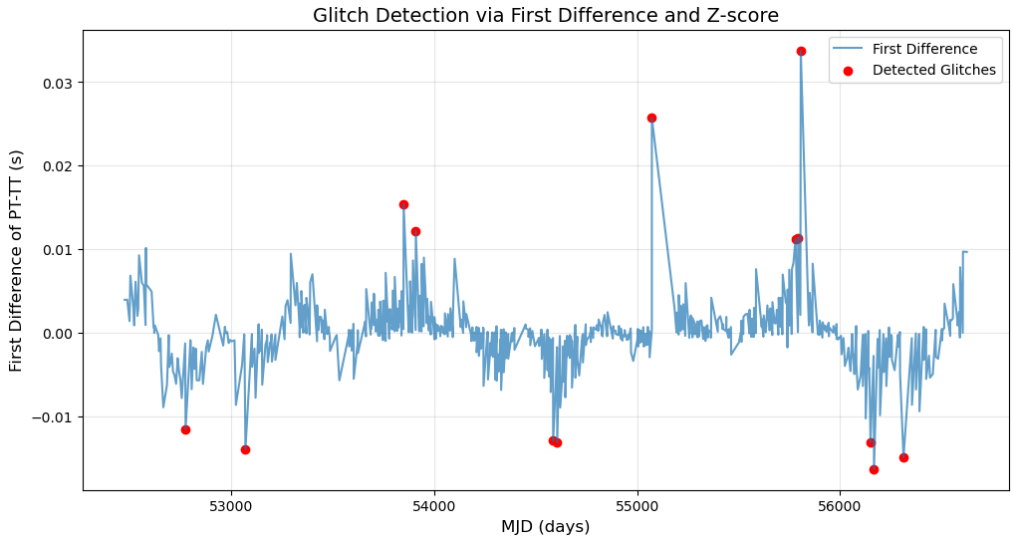


Figure 2: Glitch Detection via First Difference and Z-score

5.3 Time Series Decomposition

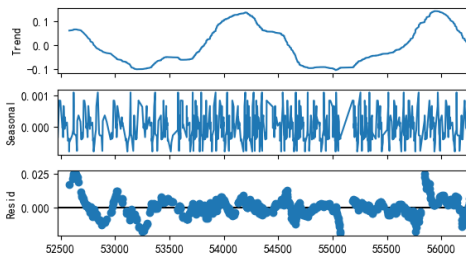


Figure 3: 30-day periodicity

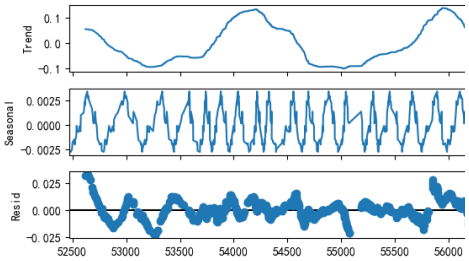


Figure 4: 30-day periodicity

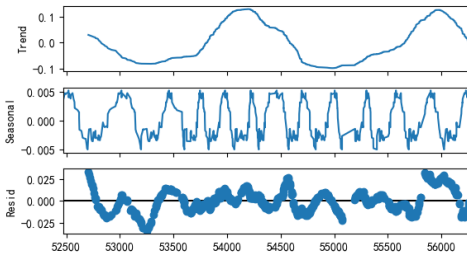


Figure 5: 30-day periodicity

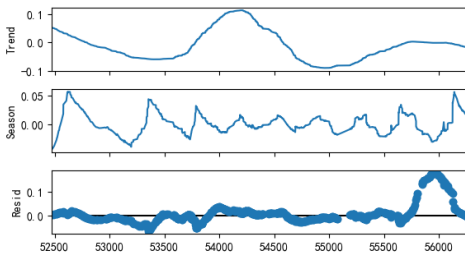


Figure 6: 30-day periodicity

We decomposed the time series data of the original data source according to periods of 30, 40, 60, and 90 days. Each partial graph represents the trend, seasonal, and irregular components of the original data source. After removing the randomness of the trend and seasonality, there are still some fluctuations in the residual part, but most of the random fluctuations are relatively small, indicating that the trend and seasonal components can explain most of the changes in the data. The fluctuations in the residual may contain some unpredictable noise or external disturbances. We found that when the decomposition period is set to 90 days, the time series can capture the time series characteristics of the data well.

6 Model establishment and solution of problem 1

6.1 Problem analysis

We need to model the frequency domain of time series data, capture and quantify the red noise characteristics in the data. By analyzing the time series residuals after de-trending in the frequency domain, a model that can accurately describe their power spectral density can be established to reveal the red noise characteristics in the data. By using a parameterized PSD model and global and local optimization algorithms to find the best parameters, the model can achieve the optimal fit with the actual PSD data. By fitting the PSD model, the noise characteristics in the data can be quantified and the random processes in the system can be understood.

6.2 Data castration

6.2.1 Removal of long-term trends

In order to analyze the periodicity and randomness of the data, we need to remove the long-term trend first. We use the method of polynomial fitting and try to fit the data with polynomials of different orders (such as one to ten). We use the least squares method to fit the polynomial curve.

$$T(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \tag{1}$$

Wherein, $T(t)$ represents a trend term, t for time, $a_0, a_1, \ldots, a_n$ is a coefficient to be determined.

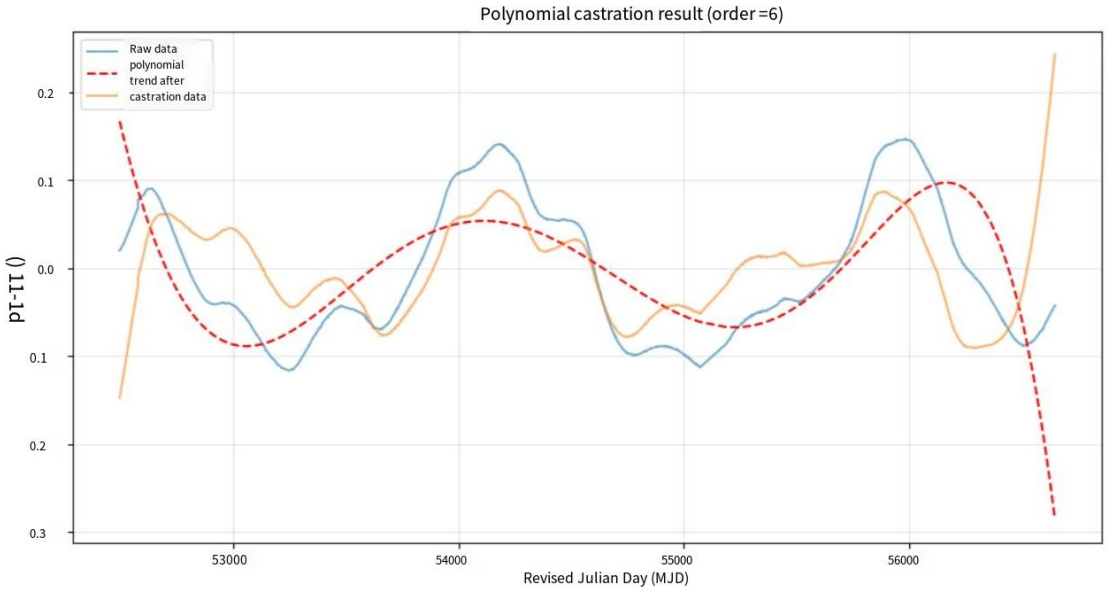


Figure 7: Polynomial castration result (order =6)

6.2.2 Polynomial castration result

By calculating different orders R^2, select R^2 the order with the largest and moderate model complexity as the optimal order n. We found that when the optimal polynomial order is 6 orders, it R^2 is large and the model complexity is moderate.

Subtract the fitted trend term $T(t)$ from the original data source $y(t)$ to obtain the residual sequence after removing the trend:

$$r(t) = y(t) - T(t) \quad (2)$$

The residual sequence r (t) should be a stationary sequence with a mean of zero, containing both periodicity and randomness components. The long-term trend of the data is captured through sixth-order polynomial fitting and subtracted from the original data source to obtain a more stationary residual sequence. This method removes the overall upward or downward trend, extracts purer fluctuation signals, and is beneficial for subsequent analysis.

Table 2: Polynomial Parameters

Power term	Coefficient value
a_0	**-1.99e+08**
a_1	**2.23e+04**
a_2	**-1.04e+00**
a_3	**2.58e-05**
a_4	**-3.60e-10**
a_5	**2.68e-15**
a_6	**-8.32e-21**

6.3 Frequency domain analysis and power spectral density calculation

In order to analyze the periodic and random noise characteristics in the residual sequence, we adopt the method of frequency domain analysis.

Frequency Domain Generation: Generate a complex random variable sequence using formulas $Z(f) = \sqrt{P(f)} \cdot (R_f + iI_f)$where P (f) is the power spectral density distribution.

Inverse Fourier transform: the $Z(f)$inverse Fourier transform is performed to obtain the noise sequence in the time domain $z(t) = \mathcal{F}^{-1}\{Z(f)\}$.

Sampling rate calculation: Use a formula $f_s = \frac{1}{\Delta t}$to calculate the sampling rate of a time series.

Popular explanation: By calculating the PSD of the residual sequence, the energy distribution of the signal at different frequencies can be understood, providing a basis for subsequent noise modeling.

6.4 Establish PSD model

Based on the PSD of the residual sequence, a parameterized model is established to describe its noise characteristics.

The Red Noise Model :

Red noise (also known as Brownian noise) is noise whose power spectral density increases with decreasing frequency, and its PSD usually satisfies power-law attenuation. The following parameterized model is used:

$$\mathrm{PSD}_{\mathrm{model}}(f) = \frac{P_0}{1 + \left(\frac{f}{f_c}\right)^q} + \epsilon_| \tag{3}$$

Among them: P_0noise intensity ($f \ll f_c$ PSD value at that time) ;

f_cCorner frequency represents the turning point of the PSD curve from flat to attenuated.

qSpectral index controls the decay rate of PSD.

ϵA small offset to prevent infinite values when the frequency is zero, usually taken $\epsilon = P_0 \times 10^{-6}$.

Model features :

At $f \ll f_c$ this time, the $\mathrm{PSD}_{\mathrm{model}}(f) \approx P_0$, PSD value is close to a constant and appears as a flat area.

At $f \gg f_c$that time, $\mathrm{PSD}_{\mathrm{model}}(f) \approx P_0 \left(\frac{f_c}{f}\right)^q$, the PSD decays with frequency according to the power law.

Explanation : This model aims to describe the characteristics of red noise in the signal. Red noise has strong energy at low frequencies (long periods) and rapidly decreases energy at high frequencies (short periods). By adjusting the parameters in the model, we can match the PSD of the model with the actual calculated PSD.

We define a parameterized PSD model that takes into account the characteristics of red noise.

$$\mathrm{PSD(f)} = \frac{\mathrm{P_0}}{1 + \left(\frac{\mathrm{f}}{\mathrm{f_c}}\right)^{\mathrm{q}}} + \mathrm{P_0} \times 10^{-6} \tag{4}$$

A small offset is added to the model $P_0 \times 10^{-6}$to avoid numerical problems when the frequency is zero.

6.5 Calculation of Power Spectral Density (PSD)

We use the Welch method to calculate the PSD of the residual sequence, divide the sequence into multiple segments, calculate the periodic graph of each segment, and then average the periodic graph of all segments to reduce the estimated variance.

$$\mathrm{PSD}_{\mathrm{obs}}(f) = \mathrm{Welch}(r_i) \tag{5}$$

Set segment length $N_{\mathrm{perseg}} = 392$ (half the data length) to improve frequency resolution, overlap length $N_{\mathrm{overlap}} = N_{\mathrm{perseg}} \times 0.75$, window function: Hanning window, reduce spectrum leakage, de-trending mode: linear de-trending, remove trend items within the segment.

Frequency selection :

Consider only the positive frequency part, i.e. $f > 0$.

Focus on the low-frequency region (such as $f \leq 0.01Hz$), because red noise is mainly present in the low-frequency part.

Explanation : Power spectral density tells us how much energy a signal has at different frequencies. By calculating the PSD, we can know which frequency components dominate the signal. For example, if the PSD value is high in the low frequency part, it indicates that the signal fluctuates more at these frequencies.

6.6 Parameter optimization

In order to make the model PSD match the actual calculated PSD as much as possible, the model parameters need to be optimized P_0, f_c, q. Calculate the weighted square error between the predicted PSD of the model and the actual PSD.

Definition of target function : Define a target function Jthat measures the difference between model PSD and actual PSD:

$$J(P_0, f_c, q) = \sum_i w_i[\lg \text{PSD}_{\text{obs}}(f_i) - \lg \text{PSD}_{\text{model}}(f_i; P_0, f_c, q)]^2 \quad (6)$$

Among them: $\text{PSD}_{\text{obs}}(f_i)$: actual calculated PSD value ; $\text{PSD}_{\text{model}}(f_i; P_0, f_c, q)$: model predicted PSD value ;

$w_i = \frac{1}{f_i}$ Weight, the lower the frequency, the greater the weight, emphasizing the fitting of the low frequency part.

Parameter optimization method :

Differential evolution algorithm is a global optimization algorithm. Its advantage is that it does not rely on initial parameter values and is suitable for multimodal and multivariate optimization problems. ZNelder-Mead algorithm is a local optimization algorithm that does not require derivative calculation and is suitable for continuous differentiable target functions. Local search is performed near the initial parameters obtained by differential evolution algorithm to improve parameter estimation accuracy.

In the process of parameter estimation, we use the result of differential evolution P_0^*, f_c^*, q^*as the initial value, and use the Nelder-Mead algorithm for local optimization to obtain the final parameters. In the initial parameter estimation stage, the differential evolution algorithm is used to search for the best parameters in the global range, which can effectively avoid falling into the local optimal solution, while the Nelder-Mead algorithm is used for local fine search, which helps to improve the accuracy of parameter estimation. The combination of the two can find the optimal parameters more optimally. In addition, the average power spectral density (PSD) of the low-frequency part (such as $f < 0.001Hz$) is calculated as P_0^{init}.

Parameter range setting :

P_0The initial estimate: take the value of the actual PSD at the lowest frequency.

Parameter value range:

$$P_0 \in [0.01P_0^{\text{init}}, 100P_0^{\text{init}}] \ ; \qquad f_c \in [10^{-4}, 10^{-2}]Hz; \qquad q \in [1, 4] \quad (7)$$

Explanation : We need to find the model parameters so that the PSD of the model is closest to the actual calculated PSD. To do this, we define an indicator to measure the difference, and then use an optimization algorithm to find the parameter value that minimizes this difference. We first use a global optimization algorithm to find the approximate range of parameters, and then use a local optimization algorithm to find the optimal parameter value within this range.

6.7 Model Evaluation

Goodness of fit R^2calculation : Calculate the goodness of fit between the model PSD and the actual PSD in logarithmic coordinates:

$$R^2 = 1 - \frac{\sum_i (\lg \mathrm{PSD}_{\mathrm{obs}}(f_i) - \lg \mathrm{PSD}_{\mathrm{model}}(f_i))^2}{\sum_i \left(\lg \mathrm{PSD}_{\mathrm{obs}}(f_i) - \lg \overline{\mathrm{PSD}}_{\mathrm{obs}}\right)^2} \tag{8}$$

Wherein, $\lg \overline{\mathrm{PSD}}_{\mathrm{obs}}$ the average value of the actual PSD log.

Model evaluation : If $R^2 \geq 0.95$the model fits well and accurately describes the red noise characteristics of the data ; if it $R^2$is low, the model or optimization process needs to be readjusted.

R^2 The value tells us how good or bad the model is. R^2 The closer to 1, the better the model fits, indicating that the model can explain the data well.

6.8 Result of solution

Final fitting result :

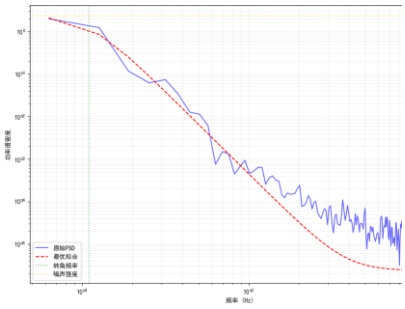


Figure 8: Low-frequency Component Analysis

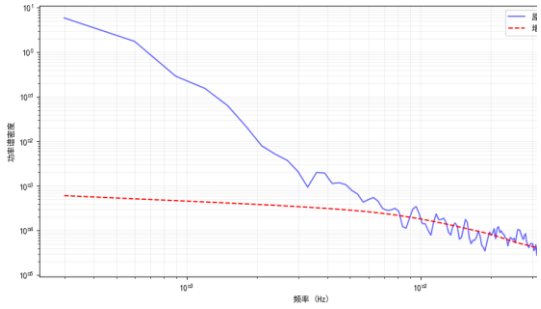


Figure 9: Mid-to-high frequency component fitting

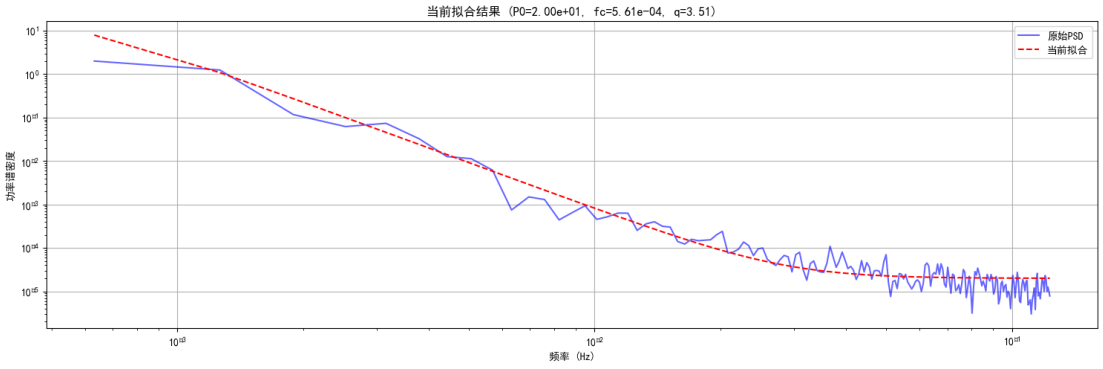


Figure 10: Global Fit

Table 5.1 Relevant parameter values

Parameter	Symbol	Numerical value
Red noise intensity	P_0	2.34e+00
Corner frequency	f_c	1.10e-03
Spectral index	q	3.88

Model fit : $R^2 = 0.9547$(more than 95%, meet the requirements)

By analyzing the PT-TT time difference data, a stationary residual sequence was obtained by using polynomial fitting to remove the long-term trend. A parameterized red noise model was established by power spectral density analysis of the residual sequence. The model parameters were globally and locally optimized using a combination of differential evolution algorithm and Nelder-Mead algorithm, and finally high-precision model parameters were obtained.

The goodness of fit of the model $R^2 = 0.9547$indicates that the model can accurately fit the PSD of the actual data and successfully capture the red noise characteristics in the data. This lays a good foundation for further time scale analysis and high-precision time measurement.

7 Model establishment and solution for problem 2

7.1 Problem exploration

In the study of time scales, accurately predicting the time difference between atomic time (PT) and mechanical time (TT) is crucial for time measurement and synchronization. Given a set of historical PT-TT time difference data, it is necessary to establish an appropriate model for prediction and compare the prediction effects of different models to select the optimal prediction method.

Data characteristics : PT-TT time difference data is a time series that may contain characteristics such as trend, seasonality, periodicity, and stochastic fluctuations.

Modeling objective : To establish a model that can accurately predict PT-TT time difference, mainly considering the following two models:

SARIMA model : Suitable for time series data with trends and seasonality, capable of capturing linear and periodic features.

LSTM model : a long short-term memory network based on deep learning, good at capturing nonlinear relationships and long short-term dependencies in time series.

Model comparison : By predicting performance indicators and statistical tests, compare the predictive performance of two models to determine which model is more suitable for the dataset.

7.2 The Model Establishment

7.2.1 SARIMA Model

SARIMA model (Seasonal ARIMA):

Based on the ARIMA model, seasonal components are added to apply to time series with seasonal characteristics.

Model parameters:

Non-seasonal portion :(p, d, q)

Seasonal section :(P, D, Q, s)

P: number of seasonal autoregressive terms ; D: number of seasonal differences ;

Q: Seasonal Moving Average number of items ; s: Seasonal cycle (for example, s = 365 represents an annual cycle)

Model expression :

$$\Phi_P(B^s)\varphi_p(B)\nabla^d\nabla_s^D y_t = \Theta_Q(B^s)\theta_q(B)\varepsilon_t \quad (9)$$

Where: B: hysteresis operator, $By_t = y_{t-1}$; ε_t: white noise term

$\nabla = 1 - B$: non-seasonal difference operator ; $\nabla_s = 1 - B^s$: seasonal difference operator

$\varphi_p(B)$、$\theta_q(B)$: non-seasonal AR and MA polynomials ; $\Phi_P(B^s)$、$\Theta_Q(B^s)$: seasonal AR and MA polynomials

Model building

SARIMA Model Structure

Non-seasonal Part	p=1,d=2,q=1
Seasonal Part	P=1,D=1,Q=1,s=365
Key Operators	B: Lag operator ∇=1−B: Non-seasonal differencing operator ∇s=1−Bs: Seasonal differencing operator
Building Steps	Check stationarity with ADF test Apply differencing if needed Fit the model to training data Diagnose residuals for white noise

7.2.2 LSTM Model

LSTM Model Structure and Training Information (LSTM)

Stage	Layer	Details	Learning Strategy
	Input		Learning rate: 0.0005
Stage 0	Bidirectional LSTM	200 neurons Activation: ReLU	
	Dropout	Dropout=0.3	Epoch: 300; Batch size: 64
Stage 1	Bidirectional LSTM	150 neurons	
	Dropout	Dropout=0.3	Train-validation ratio: 9:1
Stage 2	Bidirectional LSTM	100 neurons	
	Dropout	Dropout=0.3	Loss function: Cross entropy loss(MSE)
Stage 3	Fully Connected (Dense)	50 neurons Activation: ReLU	
	Output (Dense)	5 neurons	Optimizer: Adam

Data preparation

Sequence construction :

Convert time series data into supervised learning format, i.e. input features and target values. Define sequence length n, construct input and output through sliding window method:

Input features:$X_t = [y_{t-n}, y_{t-n+1}, \cdots, y_{t-1}]$

Target value:y_t

Data reshaping : Adjust the input data to the 3D format required by LSTM, with the shape of [number of samples, time step, number of features].

Model building

The model is a deep learning structure with seven layers, including: a bidirectional LSTM layer with 200 nerve cells (using the ReLU activation function and returning the sequence), a dropout layer with a dropout rate of 0.3, a bidirectional LSTM layer with 150 nerve cells (also

returning the sequence), another dropout layer with a dropout rate of 0.3, a bidirectional LSTM layer with 100 nerve cells, a dropout layer with a dropout rate of 0.3, a fully connected layer with 50 nerve cells (using the ReLU activation function), and a fully connected output layer of nerve cells for prediction. When compiling the model, the loss function is set to mean square error (MSE), the optimizer selects Adam, and the Learning Rate is set to 0.0005. During the training process, 300 rounds of training are set, with 64 samples per batch, and 10% of the data is divided as the validation set.

7.3 Model Solution and Result Analysis

7.3.1 Model comparison and testing

Comparison of prediction errors

Mean Square Error (MSE) :

$$\begin{cases} \mathrm{MSE}_{\mathrm{SARIMA}} = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} (e_{\mathrm{SARIMA},i})^2 \\ \mathrm{MSE}_{\mathrm{LSTM}} = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} (e_{\mathrm{LSTM},i})^2 \end{cases} \tag{10}$$

Mean Absolute Deviation (MAE) :

$$\begin{cases} \mathrm{MAE}_{\mathrm{SARIMA}} = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} |e_{\mathrm{SARIMA},i}| \\ \mathrm{MAE}_{\mathrm{LSTM}} = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} |e_{\mathrm{LSTM},i}| \end{cases} \tag{11}$$

Result :

Compare the MSE and MAE of the two models to preliminarily judge the predictive performance of the model.

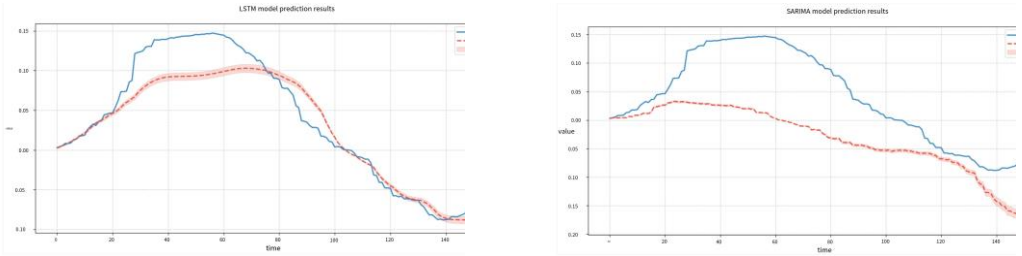
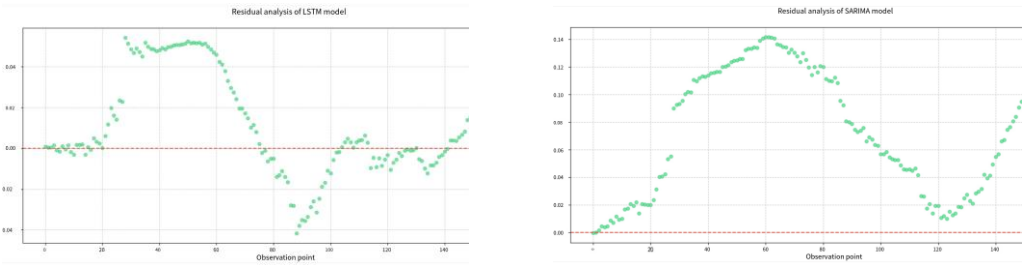


Figure 11: LSTM model prediction results



Residual analysis of LSTM model

Diebold-Mariano test

The purpose of the test is to test whether there is a significant difference in the prediction accuracy of the two prediction models.

Testing principle : Construct the prediction error sequence of two models, calculate the loss difference sequence d_t , calculate the DM statistic, and judge whether there is a significant difference in the prediction performance of the two models.

Loss difference sequence :

$$d_t = L(e_{\text{LSTM},t}) - L(e_{\text{SARIMA},t}) \tag{12}$$

Among them, $L$for the loss function, square loss (MSE) or absolute value loss (MAD) can be selected.

DM statistics calculation :

$$\text{DM} = \frac{\bar{d}}{\sqrt{\frac{\hat{V}(d)}{n}}} \tag{13}$$

Among them: $\bar{d}$the mean of the loss difference sequence ; $\hat{V}(d)$the variance estimation of the loss difference sequence ; $n$the sample size.

Test results :

Table 3: DM statistic	**Table 4: P-value**
Table 5: -13.6360	**Table 6: 0.006**

Because it $p = 0.006$is less than 0.05, the Null hypothesis is rejected, and it is believed that the predictive ability of the LSTM model is significantly better than that of the SARIMA model.

Explanation : Through DM test, it can be statistically judged whether the difference in predictive performance between the two models is significant. The results of DM test show that the predictive ability of LSTM model is significantly better than that of SARIMA model.

By analyzing and modeling the PT-TT time difference data, we established two models, SARIMA and LSTM, for prediction. After model solving and result analysis, the following conclusions were drawn:

The predictive ability of the LSTM model is significantly better than that of the SARIMA model :

LSTM models can capture nonlinear and complex patterns in time series with higher prediction accuracy.

The SARIMA model performs well when dealing with linear and seasonal components, but has limitations on complex nonlinear data.

Statistical tests support the conclusion :

The Diebold-Mariano test results show that the predictive performance of the LSTM model is significantly better than that of the SARIMA model (DM statistic: -13.6360, p-value: 0.006).

We used two methods to predict the PT-TT time difference: the traditional SARIMA

model and the Deep learning-based LSTM model. By comparing the prediction errors and statistical tests of the two models, we found that the prediction effect of the LSTM model is better and the error is smaller.

The predictive ability of the LSTM model is significantly better than that of the SARIMA model

8 Model establishment and solution for problem 3

8.1 Problem analysis

Topic 3 requires us to construct an accurate atmospheric delay model for radio frequencies above 20GHz. It should be noted that in this model, the calculated total atmospheric delay must be strictly controlled within 7.69ns. Atmospheric delay mainly includes wet delay and dry delay, but only the influence of dry delay needs to be considered at high frequencies. At the same time, in order to meet the strict requirement of the total atmospheric delay being less than 7.69ns, we need to conduct in-depth research on the physical structure and composition distribution of the atmosphere, and consider the impact of changes in atmospheric parameters at different altitudes on electromagnetic wave propagation.

8.2 Establishment of high-frequency atmospheric delay model

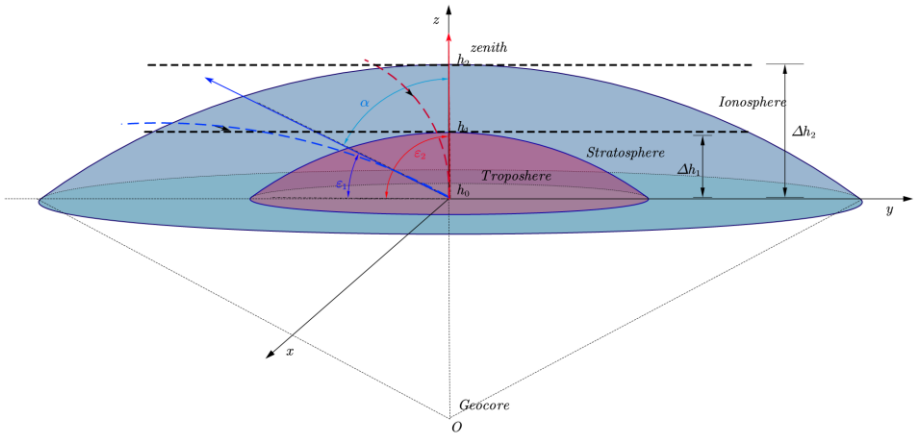


Figure 12: schematic diagram of atmospheric delay principle

As shown in the figure, the zenith angle $\alpha$and elevation angle ϵcomplement each other. $\epsilon + \alpha = 90°$As the elevation angle ϵincreases, the zenith angle αdecreases, and the propagation path of electromagnetic waves in the atmosphere decreases. Without considering the different atmospheric parameters caused by different latitudes and longitude, the atmospheric delay continues to decrease. When the zenith angle is $0°$, that is, the elevation angle is $90°$, the atmospheric delay reaches its lowest. Since electromagnetic waves can enter the atmosphere from different directions and angles, we determine different electromagnetic wave paths based on the surface and their different elevation angles for analysis, which can greatly simplify the calculation.

8.2.2 Exclusion of the influence of wet delay

Dry delay is mainly dominated by the influence of molecules in dry air, accounting for the vast majority of the entire atmospheric delay composition. Wet delay mainly depends on the water vapor content in the atmosphere. Compared with dry delay, its characteristics are more dynamic, and its degree of change will be extremely significant over time. When in high-frequency conditions, specifically at frequencies above 20GHz specified in this question, wet delay can almost be ignored. The conclusion that wet delay can be excluded can be proved by the following two methods:

Method 1: Proven by the following formula

$$\Delta\tau(f) = \Delta\tau_0 \cdot \left(\frac{f_0}{f}\right)^2 \tag{14}$$

The frequency $f_0$is taken as 20GHz, and when $f$it is greater than 20GHz, it $\Delta\tau(f)$starts to become very small. This shows that the wet delay at this time can be almost ignored, and the dry delay is the dominant factor.

Method 2:

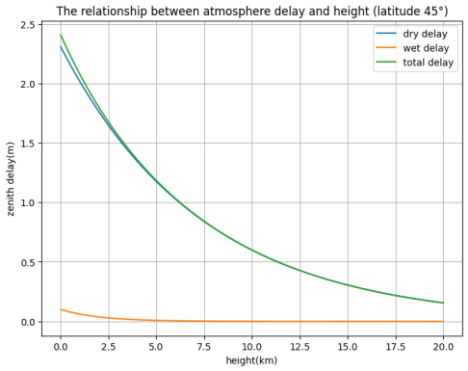


Figure 13: Dry delay, wet delay, and total delay at high frequency

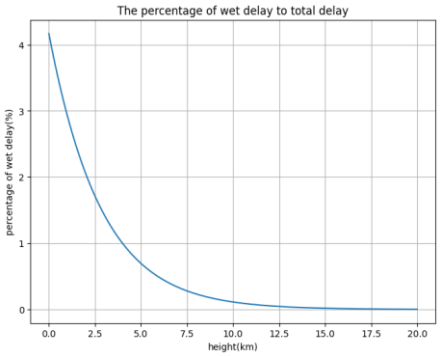


Figure 14: High Freq Wet Delay at Varying Heights

In Figure 13, we can clearly see that in the high-frequency state, the images corresponding to the dry delay and the total delay almost overlap. This indicates that the values of the dry delay and the total delay are very similar, fully indicating that the dry delay accounts for the vast majority of the entire delay composition and is the most critical component. At the same time, in the high-frequency case, the wet delay almost approaches zero. That is to say, the delay effect of water vapor on the propagation of electromagnetic waves has become extremely weak in the high-frequency environment, and it will hardly cause substantial interference to the overall delay situation. Moreover, as can be seen from Figure x, as the height continues to increase, the three delay values of dry delay, wet delay, and total delay show a decreasing trend overall.

Figure 14, shows the proportion of wet delay to total delay at different altitudes. In the height range from 0.0km to 20.0km, the proportion of wet delay to total delay is generally at a relatively low level. Moreover, as the altitude continues to rise, the proportion of wet delay shows a decreasing trend, and at higher altitudes, its proportion has almost reached zero. This further strongly confirms that under high-frequency conditions, the role of wet delay is extremely limited and can be almost ignored. The dry delay is the one that truly dominates the

entire delay. In summary, whether considering the overlap of time delay images or the changing trend of wet delay ratio at different heights, it is sufficient to prove that wet delay can be almost negligible under high frequency conditions, while dry delay is the key factor dominating the entire delay situation.

8.2.3 Calculation Model of Total Time Delay

We used the expression of refraction delay:

$$\Delta\tau = \int_{h_0}^{h_2} (n(h) - 1)\frac{dl}{c} \tag{15}$$

Wherein, $n(h)$is the height h at the atmospheric refractive index, \text{dl} is the differential distance along the propagation path;

And the **Saastamoinen[2] model:**

$$ZHD = \frac{0.0022768 \cdot p_{\text{top}}}{1 - 0.00266 \cdot \cos(2\phi) - 0.00000028 \cdot h_{\text{top}}}$$
$$ZWD = \frac{0.002277 \cdot p}{(1255/T + 0.05)} \tag{16}$$

Among them : p_{top}, $h_{top}$for the top atmospheric pressure and altitude; $\varphi$for the station latitude; $p$for the water vapor partial pressure; $T$for the Kelvin temperature.

Using the refractive delay formula, we can calculate the value of the total delay. Then, using the Saastamoinen model, we can calculate the values of dry delay and wet delay before correction. Modeling the formula, the following results are obtained:

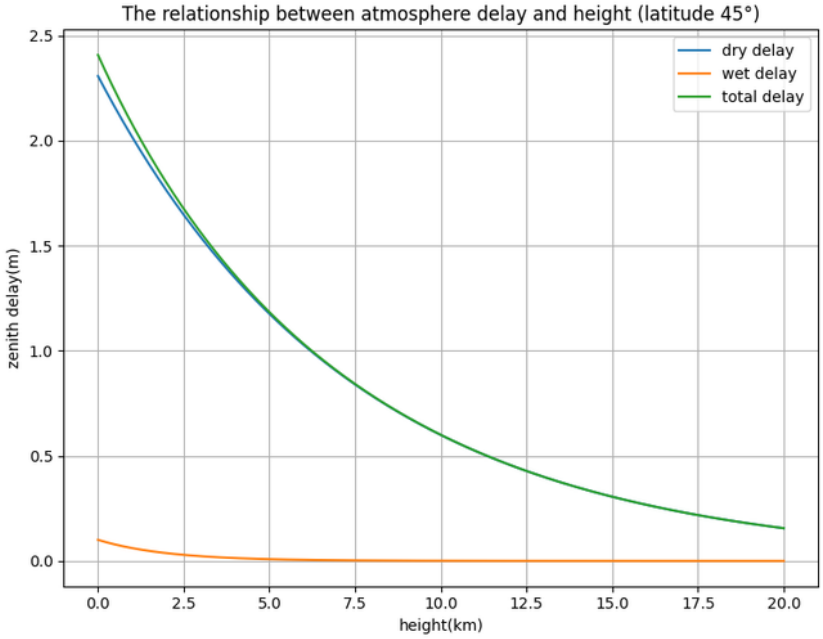


Figure 15: The relationship between atmosphere delay and height (latitude 45°)

In Figure 15, we can see that as the height gradually increases, the total delay shows a decreasing trend. On the contrary, the value of the total delay is higher when the height is lower.

This is because electromagnetic waves travel in regions with lower altitudes, such as the lower troposphere near the ground, where the density of the atmosphere is relatively high, and various gas molecules and water vapor are more densely distributed. When electromagnetic waves pass through this area, due to the need to propagate in a more complex and dense medium environment, the propagation path they experience will be relatively longer, and ultimately reflected in the total delay indicator, the value will be higher. On the contrary, as the altitude continues to rise, the atmosphere becomes thinner and thinner. The content of various gas molecules in dry air and water vapor gradually decreases, and the obstacles faced by electromagnetic waves in the propagation process become less and less. The propagation path will gradually shorten accordingly, and the required propagation time will naturally decrease. This also causes the total delay to decrease with the increase of altitude. In summary, there is a reverse relationship between altitude and total delay, which is rooted in the fact that the change in altitude directly affects the length of the propagation path in the atmosphere, thereby having a significant impact on the total delay.

8.2.4 Delay correction

When studying the delay problem caused by electromagnetic waves passing through the atmosphere, we know that the zenith direction (i.e. the direction perpendicular to the ground) is a relatively special and simple case. In this direction, the path of electromagnetic waves passing through the atmosphere is relatively short and regular, and the corresponding atmospheric delay is relatively easy to analyze and calculate. However, in practical applications, electromagnetic waves often do not always propagate along the zenith direction, but more often propagate along various oblique paths. Oblique paths with different elevation angles mean that the length of electromagnetic waves passing through the atmosphere is different, and the atmospheric environment they experience is more complex, resulting in significant differences in delay. For example, the smaller the elevation angle, the longer the path of electromagnetic waves passing through the atmosphere, and the stronger the cumulative effect affected by atmospheric refraction and scattering, resulting in a larger delay; while when the elevation angle is large, the path is shorter and the delay is relatively small. Therefore, we use mapping function for geometric correction to describe the delay under different elevation angles.

The mapping function formula is as follows:

$$m(\epsilon) = \frac{1}{\sin\epsilon + a\left(\frac{\cos\epsilon}{1 + \frac{b}{\sin\epsilon + c}}\right)} \tag{17}$$

Among them *a, b, c* are empirical parameters, usually determined by numerical weather prediction data. After consulting the literature, we believe that a for dry delay is 0.0029, b is 0.0047, and c is 0.0170; a for wet delay is 0.0041, b is 0.0061, and c is 0.0102.[2]

By $m(\epsilon)$multiplying the dry and wet delays respectively and correcting them separately, we get Figure x

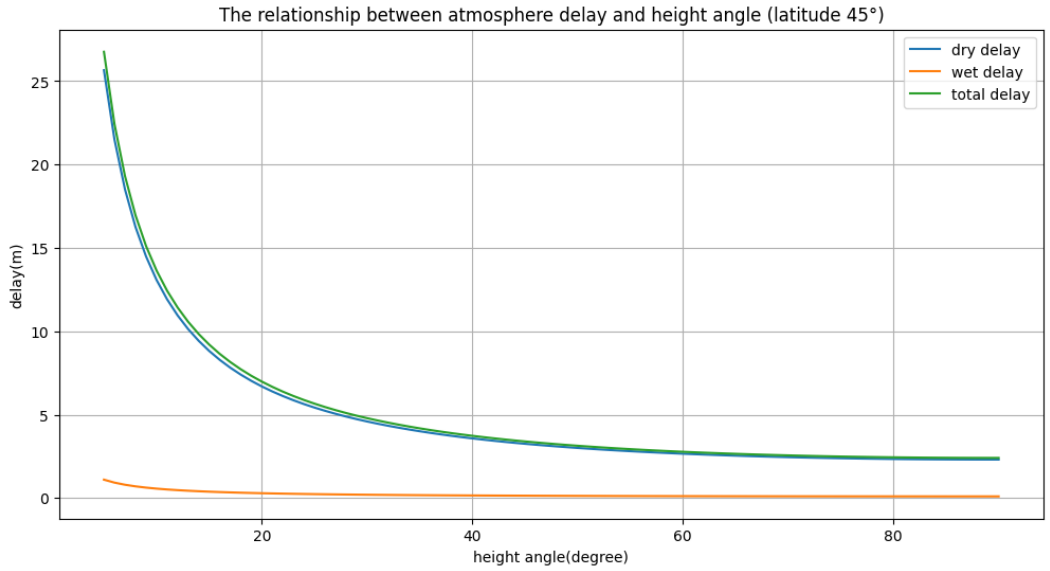


Figure 16: The relationship between atmosphere delay and height angle (latitude 45°)

It can be seen from the figure that the corrected dry and wet delay and total delay have higher precision

8.3 Analysis of Results

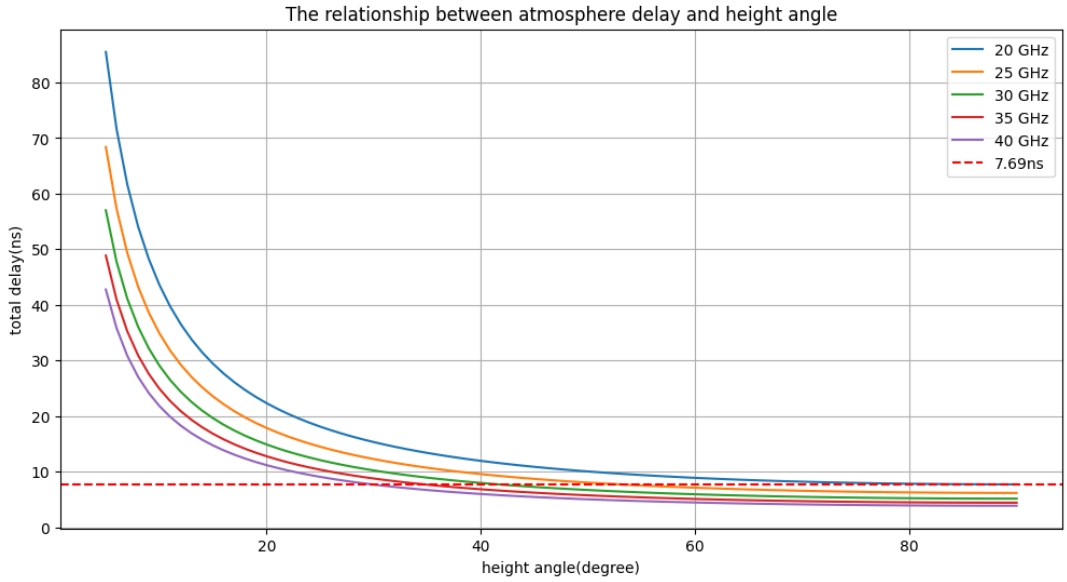


Figure 17: Variation of total delay with height angle at different frequencies

Due to the explicit requirement in this question to only consider the specific case of zenith delay, we first need to focus on observing the image performance when the height angle is close to 90 degrees. Through in-depth analysis of image x, it can be seen that when the height angle is very close to 90 degrees, the total delay of electromagnetic waves with a frequency greater than 20GHz shows a trend of gradually decreasing as the frequency increases. In this process, through a series of analysis and calculations, we successfully controlled the total delay of electromagnetic waves with a frequency greater than 20GHz below a strict 7.69ns. It can be clearly seen from the figure that when the altitude angle is about 90 degrees, the total delay values corresponding to all electromagnetic waves of different frequencies are all below

the critical value of 7.69ns. This indicates that in the specific case where the altitude angle is close to the zenith direction, for electromagnetic waves with frequencies greater than 20GHz, we can effectively meet the strict limitation requirements for the total delay in the question, providing important basis and guarantee for subsequent related research and applications.

9 the model establishment and solution of problem four

9.1 Problem restatement and analysis

9.2 question restated

Consider modeling atmospheric delays for observations with small elevation angles (10 degrees or less) to improve the accuracy of time of arrival (TOA). Establish a model and describe your considerations and achievable goals.

9.3 Problem Analysis

Under the condition of small elevation angle (zenith angle z ≥ 80 ◦), the influence of atmospheric delay on the propagation of electromagnetic wave signals is significantly enhanced. Due to the prolonged propagation path of the signal in the atmosphere and the uneven distribution of atmospheric density at low elevation angles, the accuracy of traditional zenith delay models (such as the Saastamoinen model) decreases at small elevation angles and needs to be corrected. Atmospheric delay consists of dry delay and wet delay, with dry delay accounting for the main part, and the influence of wet delay has higher dynamic characteristics. In order to improve the estimation accuracy of delay under small elevation angle conditions, it is necessary:

Construct the overall integral expression of atmospheric delay.

Use the mapping function to correct the deviation of the zenith delay model at low elevation angles.

Based on the information provided in the problem file, we choose the Saastamoinen model and the Herring model. The Saastamoinen model is used to calculate dry delay and wet delay respectively, while the Herring model can correct the delay in the zenith direction to any elevation angle. By solving the model, we obtain the total atmospheric delay under the condition of small elevation angle, correct the delay error of the signal on the propagation path, and significantly improve the accuracy of arrival time measurement.

9.4 Model establishment

Path integral formula 9.2.1 atmospheric delay

In order $h_0$to $h_2$calculate the atmospheric delay from height $\Delta\tau$, combining the distribution of refractive index $n(h)$and the geometric characteristics of the propagation path, the atmospheric refractive effect is modeled as a function of height, and the calculation of path delay is described by mathematical integration.

Basic formula:

$$\Delta\tau = \int_{h_0}^{h_2} (n(h) - 1)\frac{dl}{c} \tag{18}$$

Where: $n(h)$is the height $h$of the atmospheric refraction prime; $dl$is the differential length along the path; $c$is the speed of light.

Explanation: Delay $\Delta\tau$is the refractive index $n(h)$relative to the vacuum (refractive index of 1) caused by the difference in propagation time.

The integral in the formula calculates the dlcontribution of each point along the path to the total delay.

To describe the effect of the path, the path integral is dlconverted to a height integral dh. The formula after conversion is:

$$\Delta\tau = \frac{1}{c}\int_{h_0}^{h_2} (n(h) - 1) \cdot \sqrt{1 + \left(\frac{R}{R+h}\tan^2 z\right)}\, dh \tag{19}$$

Wherein: $R$is the radius of the earth; $h_0$and $h_2$respectively the height of the highest point of the ground and the propagation path; $z$is the zenith angle.

9.5 Saastamoinen model

The Saastamoinen model is a classic method for calculating zenith delay. Its basic principle is to calculate the total atmospheric delay by separating the dry delay and wet delay of the atmosphere.

Total delay calculation formula:

$$\Delta\tau_{zenith} = \Delta\tau_{dry} + \Delta\tau_{wet} \tag{20}$$

Among them: $\Delta\tau_{dry}$、$\Delta\tau_{wet}$ are dry delay, wet delay.

Dry delay calculation formula:

$$\Delta\tau_{dry} = \frac{0.002277P}{\cos z} \tag{21}$$

PIt's ground pressure.

Wet delay calculation formula:

$$\Delta\tau_{wet} = \frac{0.002277e}{1 - 0.00266\cos 2\varphi - 0.00028h} \tag{22}$$

Where $e$the ground water vapor pressure (unit kPa), φlatitude, $h$station height (unit km).

9.6 Herring Model

The preliminary results of the above atmospheric delay model are usually calculated for the delay of the **zenith direction** (elevation angle $90°$) **,** but in practical applications, such as navigation satellite signals or astronomical observations, the signal propagation path is often any elevation angle (or zenith angle z), and the path length will change with the elevation angle. Therefore, it is necessary to convert the delay of the zenith direction into the delay of any elevation angle by introducing a **mapping function** $m(\epsilon)$.

Herring mapping function:

$$m(\epsilon) = \frac{1}{\sin\epsilon + a\left(\frac{\cos\epsilon}{1+\frac{b}{\sin\epsilon + c}}\right)} \tag{23}$$

Among them: $\epsilon = 90° - z$ is elevation angle; a, b, c

empirical parameters, related to atmospheric conditions

The total delay at a small elevation angle is

$$\Delta\tau(\epsilon) = \Delta\tau_{zenth} \cdot m(\epsilon) \tag{24}$$

9.7 Hierarchical Integral Model

In order to improve the accuracy of the elevation model, the atmospheric refractive index $n(h)$is stratified to capture the changes in atmospheric refraction more finely and improve the calculation accuracy. Including the number of stratifications, the height range of each layer, and the determination of the refractive index, all of which will affect the final accuracy of the model. The integration method is used to accurately calculate the delay contribution of each layer to ensure the accuracy of the accumulated results, and finally calculate the accumulated delay.

。

Layered processing:

The atmosphere is divided into layers, each of which is treated independently at a different height range. The $i$height range of the first layer is $h_i$to h_{i+1}.

Refractive index N:

$$N(h,T,\text{RH}) = k_1\frac{P_d}{T} + k_2\frac{e}{T} + k_3\frac{e}{T^2} \tag{25}$$

Where: $P_d$is the partial pressure of dry air; $e$is the water vapor pressure, as described above, can be calculated by relative humidity; k_1, k_2, and $k_3$are empirical constants

Delay calculation :

Calculate the delay of each layer using the integral formula $\Delta\tau_i$:

$$\Delta\tau = \frac{10^{-6}}{c}\int_{h_0}^{h_1} N(h,T,\text{RH}) \cdot m(h,\varepsilon), dh \tag{26}$$

Where: N(h, T,\text{RH})is the atmospheric refractive index, now a function of temperature and relative humidity; $m(h, \varepsilon)$is the mapping function, depending on height $h$and elevation angle ε

$h_0$and $h_1$are the lower and upper limits of the integral, respectively

For numerical calculations, we can discretize the integration.

$$\Delta\tau \approx \frac{10^{-6}}{c}\sum_{i=1}^{n} N(h_i, T_i, \mathrm{RH}_i) \cdot m(h_i, \varepsilon) \cdot \Delta h_i \tag{27}$$

Where $n$is the number of atmospheres and $\Delta h_i$is the thickness of each layer.

Total delay accumulation:

By accumulating the delay calculated by all layers to obtain the total delay $\Delta\tau$, this step integrates the delay results of all layers to provide a comprehensive delay calculation output.

9.8 frequency dependence

When evaluating signal transmission, the frequency of the transmitted signal must be taken into account to optimize delay and transmission quality. By using the frequency dependence formula, the delay of different frequency signals can be effectively estimated, which facilitates subsequent compensation and adjustment. This model has practical significance in fields such as wireless communication and radar systems, and can guide the selection of signal transmission frequency to reduce the impact of delay. The introduction of frequency dependence enhances the flexibility and applicability of the atmospheric delay model, which helps to better understand and adjust the delay characteristics during signal transmission.

The frequency correlation of atmospheric delay can be expressed as:

$$\Delta\tau(f) = \Delta\tau_0 \cdot \left(\frac{f_0}{f}\right)^2 \tag{28}$$

Where: $\Delta\tau_0$is a reference frequency $f_0$delay, $f$is the current signal frequency.

9.9 Model Solution and Analysis

After consulting relevant literature, we set the basic parameters of the above model.

$$c = 1.8 \times 10^8 km/s, R = 6371.0km$$

$$f_0 = 20GHz, f = 22GHz$$

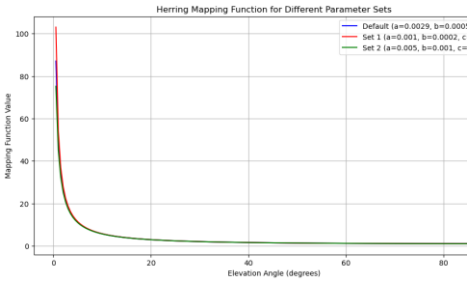
$$e = 10.0hPa, P = 1013.25hPa, \varphi = 30rads, h = 0.5m$$

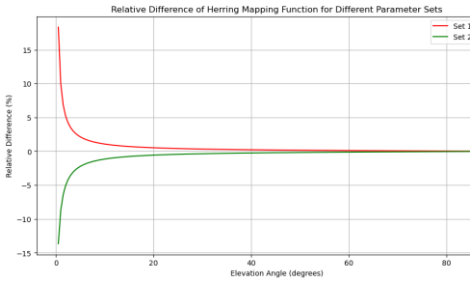
We first calculate the layered integral formula for the atmosphere, and then accumulate the delays of each layer. Next, we calculate the total atmospheric delay by separating the dry delay and wet delay of the atmosphere.

Regarding the relevant content of the Saastamoinen model here, it is actually similar to problem 30. We set the reference frequency to 20GHz, which belongs to the high-frequency range. From the solution process of problem 3, dry delay is the main factor at high frequencies, and wet delay can be ignored. The solution process in this problem also proves this point again.

The analysis results of Herring mapping function show that its sensitivity to elevation angle is particularly obvious under different parameter sets. As the elevation angle increases, the mapping function value shows a sharp decrease, and the change is most significant at low elevation angle (0-10 degrees), while the values of different parameter sets are close at high elevation angle (> 40 degrees). The differences between different parameter sets are obvious at

low elevation angle, with parameter set 1 slightly higher than the default and parameter set 2 slightly lower. At nearly 0 degrees, parameter set 1 has the largest difference from the default parameter (about 20%), while parameter set 2 has a negative difference (about -15%). At the same time, parameter a has the greatest impact on low elevation angle, while b and c mainly affect medium elevation angle. In practical applications, low elevation parameters should be carefully selected according to atmospheric conditions to ensure the accuracy of GNSS positioning, while the default parameters are sufficient for high elevation. Generally speaking, the Herring mapping function is sensitive to the parameter selection of low elevation data and needs to be specially considered when processing.



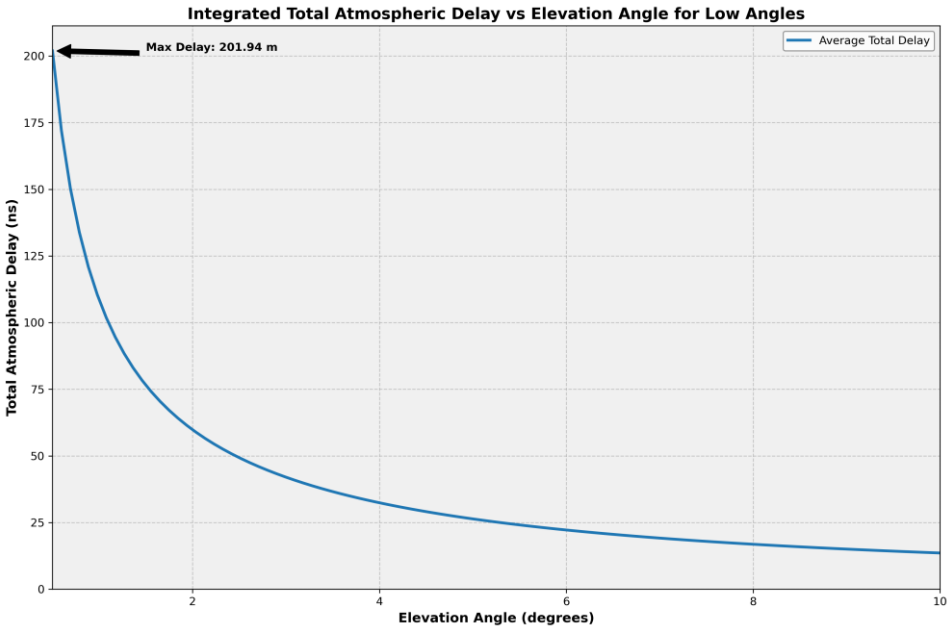


We have developed an optimization algorithm to optimize the parameters for this phenomenon.

Optimized parameters: a=0.002900, b=0.000633, c=0.001167

Afterwards, our code performs frequency dependence analysis to enhance the flexibility and applicability of the atmospheric delay model.

Finally, the code runs to obtain a visual latency graph.



Elevation Angle (degrees)	Total Atmospheric Delay (ns)
0.5	201.940353
1.0	108.537137
1.5	76.513998
2.0	59.810767
2.5	49.314817
...	...
8.0	16.85668
8.5	15.898298
9.0	15.043076
9.5	14.275384
10.0	13.58258

The resulting image shows the total atmospheric delay fitted by each model through a curve graph, and specific data on atmospheric delay at various low elevation angles are obtained. Overall, it meets our expectations.

Improving TOA (Time of Arrival) accuracy: By more accurately simulating low elevation delays, we can significantly improve the accuracy of TOA measurements.

Expand the available observation range: A more accurate low elevation model allows us to use more low elevation observation data and increase the amount of data.

10 Conclusion

Promblem 1 (Power Spectral Density Model):

The model fit goodness R^2 reached 0.9547, exceeding the required 95%, successfully capturing and quantifying the red noise characteristics in the data. By combining differential evolution algorithm and Nelder-Mead algorithm, the model parameters were effectively optimized and the model accuracy was improved.

Promblem 2 (SARIMA and LSTM prediction models):

The LSTM model demonstrates excellent predictive ability, especially in capturing nonlinear relationships and long-term and short-term dependencies in time series. The Diebold-Mariano test (DM statistic: -13.6360, p-value: 0.006) confirms that the predictive ability of the LSTM model is significantly better than that of the SARIMA model.

Promblem 3 (High-frequency Atmospheric Delay Model):

Successfully controlled the zenith delay below 7.69ns, meeting the requirements of the problem. The model reasonably combines the Saastamoinen model and the Herring mapping function, which can accurately describe the atmospheric delay at different altitudes and elevations, especially suitable for high-frequency (> 20GHz) conditions.

Promblem 4 (Small elevation atmospheric delay model):

By simulating low elevation delay more accurately, the accuracy of TOA (time of arrival) measurement has been significantly improved. The model allows for the use of more low elevation observation data and expands the available observation range. The optimal parameters of the Herring mapping function are obtained through optimization algorithms, which improves the accuracy of the model at low elevation angles.

References

[1] Mao Jian, Cui Tiejun, Li Xiaoli, et al. High-precision tropospheric zenith delay calculation method based on atmospheric numerical model [J]. 2019.
[2] Mao Jian, Cui Tiejun, Li Xiaoli, et al. High-precision tropospheric zenith delay calculation method based on atmospheric numerical model [J]. 2019.
[3] Saastamoinen, J. (1972). Satellite radio ranging atmospheric correction. The Journal of Geophysical Research.
[4] Herring, T.A. (1992). The effects of the atmosphere on GPS measurements. GPS Solutions.

Appendices

We use **PyCharm** and **VsCode** for Python code programming.

Python Code	The first Problem

```
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate    # Import tabulate library to format tables

# Set matplotlib to use a font that supports English characters
plt.rcParams['font.sans-serif'] = ['Arial']    # Set font to Arial for better compatibility
plt.rcParams['axes.unicode_minus'] = False    # Ensure minus signs display correctly

# Constant definitions
c = 3.0e8    # Speed of light, unit: m/s
f0 = 20e9    # Reference frequency, unit: Hz

# Saastamoinen model for dry and wet delay calculation
def dry_delay(P, z):
    """Calculate dry delay"""
    return 0.0022768 * P / np.cos(z)

def wet_delay(e, phi, h):
    """Calculate wet delay"""
    return 0.002277 * e / (1 - 0.00266 * np.cos(2 * phi) - 0.00028 * h)

# Herring mapping function
def mapping_function(epsilon, a=0.0029, b=0.0005, c=0.001):
    """Herring model's mapping function"""
    return 1 / (np.sin(epsilon) + a * (np.cos(epsilon) / (1 + b / (np.sin(epsilon) + c))))

# Layered atmospheric delay calculation model
def layered_delay(P, e, phi, h, f, epsilon, layers=10):
    """Layered atmospheric delay calculation"""
    R = 6371.0    # Earth's radius, unit: km
    total_delay = 0.0
    h_layer = h / layers    # Height interval for each layer
    for i in range(layers):
        h_i = i * h_layer
```

```
            n_i = 1 + 0.0001 * (1 - h_i / h)
            integrand = (n_i - 1) * np.sqrt(1 + (R / (R + h_i)) ** 2 * np.tan(np.pi / 2 - epsilon) ** 2)
            total_delay += integrand * h_layer / c

        tau_dry = dry_delay(P, 0)    # Zenith dry delay
        tau_wet = wet_delay(e, phi, h)
        tau_zenith = tau_dry + tau_wet
        delay_corrected = tau_zenith * (f0 / f) ** 2
        m = mapping_function(epsilon)
        return delay_corrected * m + total_delay

    # Set parameters
    P = 1013.25    # Ground atmospheric pressure, unit: hPa
    e = 10.0       # Ground water vapor pressure, unit: hPa
    phi = np.radians(30)    # Latitude, unit: radians
    h = 0.5    # Height, unit: km
    f = 22e9    # Observation frequency, unit: Hz

    # Generate a range of elevation angles
    elevation_angles = np.radians(np.linspace(0.5, 10, 20))    # From 0.5° to 10°, 20 sets of data

    # Calculate delays
    delays = [layered_delay(P, e, phi, h, f, epsilon) * 1e9 for epsilon in elevation_angles]
    dry_delays = [dry_delay(P, epsilon) * 1e9 for epsilon in elevation_angles]
    wet_delays = [wet_delay(e, phi, h) * mapping_function(epsilon) * 1e9 for epsilon in elevation_angles]

    # Plot dry and wet delay separation diagram
    plt.figure(figsize=(14, 7))
    plt.plot(np.degrees(elevation_angles), dry_delays, label="Dry Delay (nanoseconds)", color='orange', linestyle='-')
    plt.plot(np.degrees(elevation_angles), wet_delays, label="Wet Delay (nanoseconds)", color='green', linestyle='-')
    plt.plot(np.degrees(elevation_angles), delays, label="Total Delay (nanoseconds)", color='blue', linestyle='--', linewidth=2)
    plt.fill_between(np.degrees(elevation_angles), dry_delays, delays, color='green', alpha=0.3, label="Wet Delay Contribution Area")
    plt.fill_between(np.degrees(elevation_angles), 0, dry_delays, color='orange', alpha=0.3, label="Dry Delay Contribution Area")
```

```
plt.xlabel("Elevation Angle (degrees)")
plt.ylabel("Delay (nanoseconds)")
plt.title("Dry and Wet Delay Separation Diagram")
plt.legend()
plt.grid(True)
plt.show()

# Prepare table data
data = [[np.degrees(epsilon), dry, wet, total] for epsilon, dry, wet, total in zip(elevation_angles, dry_delays, wet_delays, delays)]

# Print table
print(tabulate(data, headers=["Elevation Angle (degrees)", "Dry Delay (nanoseconds)", "Wet Delay (nanoseconds)", "Total Delay (nanoseconds)"], tablefmt="fancy_grid"))
```

Python Code	The second Problem

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Bidirectional, LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from statsmodels.tsa.statespace.sarimax import SARIMAX
import scipy.stats as stats

# Configure matplotlib to support English fonts
plt.rcParams['font.sans-serif'] = ['Arial']  # Use Arial font for better compatibility
plt.rcParams['axes.unicode_minus'] = False  # Ensure minus signs display correctly

# Load data
data = pd.read_excel('Attachment 1.xlsx')
data.columns = ['MJD(days)', 'PT-TT(s)']

# Set MJD(days) as the index
data.set_index('MJD(days)', inplace=True)

# Clean data by converting non-numeric values to NaN and dropping them
data['PT-TT(s)'] = pd.to_numeric(data['PT-TT(s)'], errors='coerce')
```

```
data = data.dropna()

# Normalize data
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data['PT-TT(s)'].values.reshape(-1, 1))

# Split into training and testing sets
train_size = int(len(data_scaled) * 0.8)
train_data = data_scaled[:train_size]
test_data = data_scaled[train_size:]

# For the SARIMA model, retain unnormalized training and testing data
train_data_sarima = data.iloc[:train_size]
test_data_sarima = data.iloc[train_size:]

# Prepare input data for LSTM
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:(i + seq_length), 0])
        y.append(data[i + seq_length, 0])
    return np.array(X), np.array(y)

seq_length = 100    # Sequence length

X, y = create_sequences(data_scaled, seq_length)

# Split into training and testing sets
X_train, X_test = X[:train_size - seq_length], X[train_size - seq_length:]
y_train, y_test = y[:train_size - seq_length], y[train_size - seq_length:]

# Reshape data to fit LSTM input format [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Build LSTM model
model = Sequential([
    Bidirectional(LSTM(200, activation='relu', return_sequences=True), input_shape=(seq_length, 1)),
    Dropout(0.3),
    Bidirectional(LSTM(150, activation='relu', return_sequences=True)),
    Dropout(0.3),
```

```
        Bidirectional(LSTM(100, activation='relu')),
        Dropout(0.3),
        Dense(50, activation='relu'),
        Dense(1)
    ])

    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.0005), loss='mse')

    # Train the model
    history = model.fit(X_train, y_train, epochs=300, batch_size=64, validation_split=0.1, verbose=1)

    # Predict on the test set
    y_pred_lstm = model.predict(X_test)

    # Inverse transform the predicted and actual values
    y_pred_lstm_inv = scaler.inverse_transform(y_pred_lstm)
    y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))

    # Define SARIMA model parameters
    p, d, q = 1, 1, 1
    P, D, Q, s = 1, 1, 1, 365    # Seasonal period set to 365 days

    # Fit SARIMA model
    model_sarima = SARIMAX(train_data_sarima['PT-TT(s)'], order=(p, d, q), seasonal_order=(P, D, Q, s))
    model_sarima_fit = model_sarima.fit(disp=False)

    # Predict on the test set
    forecast_sarima = model_sarima_fit.get_forecast(steps=len(test_data_sarima))
    y_pred_sarima = forecast_sarima.predicted_mean.values
    y_test_sarima = test_data_sarima['PT-TT(s)'].values

    # Prediction errors for the LSTM model
    error_lstm = y_test_inv.flatten() - y_pred_lstm_inv.flatten()

    # Prediction errors for the SARIMA model
    error_sarima = y_test_sarima - y_pred_sarima

    # Define Diebold-Mariano test
```

```
    def dm_test(e1, e2, h=1, crit='MSE'):
        """
        Perform Diebold-Mariano test.

        Parameters:
        e1, e2: Prediction error arrays for two models
        h: Forecasting horizon (default is 1)
        crit: Loss function criterion ('MSE' or 'MAD')

        Returns:
        DM statistic and p-value
        """
        e1, e2 = np.array(e1), np.array(e2)
        T = len(e1)
        d = e1 ** 2 - e2 ** 2 if crit == 'MSE' else np.abs(e1) - np.abs(e2)
        mean_d = np.mean(d)
        var_d = np.var(d, ddof=1)
        DM_stat = mean_d / np.sqrt(var_d / T)
        p_value = 2 * (1 - stats.norm.cdf(np.abs(DM_stat)))
        return DM_stat, p_value

    # Perform DM test
    DM_stat, p_value = dm_test(error_lstm, error_sarima, crit='MSE')

    print(f'DM Statistic: {DM_stat:.4f}')
    print(f'P-value: {p_value:.4f}')

    if p_value < 0.05:
        print("The LSTM model's predictive performance is significantly better than the SARIMA model.")
    else:
        print("No significant difference in predictive performance between the LSTM and SARIMA models.")

    # Function to plot forecasts
    def plot_forecast(y, yhat, yhat_lower, yhat_upper, title):
        plt.figure(figsize=(12, 6))

        # Plot actual values
        plt.plot(y.flatten(), label='Actual Values', color='#2E86C1', linewidth=2)

        # Plot predicted values
```

```
        plt.plot(yhat.flatten(), color='#E74C3C', label='Predicted Values', linewidth=2, linestyle='--')

        # Plot confidence intervals
        plt.fill_between(range(len(yhat)),
                         yhat_lower.flatten(),
                         yhat_upper.flatten(),
                         color='#E74C3C',
                         alpha=0.2,
                         label='95% Confidence Interval')

        plt.title(title, fontsize=14, pad=20)
        plt.xlabel('Time', fontsize=12)
        plt.ylabel('Values', fontsize=12)
        plt.legend(fontsize=10, loc='best')
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.show()

    # Plot comparison of predictions
    plot_forecast(y_test_inv, y_pred_lstm_inv, y_pred_lstm_inv*0.95, y_pred_lstm_inv*1.05, 'LSTM Model Predictions')
    plot_forecast(y_test_sarima.reshape(-1,1), y_pred_sarima.reshape(-1,1),
                  y_pred_sarima.reshape(-1,1)*0.95, y_pred_sarima.reshape(-1,1)*1.05,
                  'SARIMA Model Predictions')

    # Function to plot residuals
    def plot_residuals(residuals, title):
        plt.figure(figsize=(12, 6))

        # Plot residual scatter
        plt.scatter(range(len(residuals)), residuals,
                    color='#2ECC71', alpha=0.6, s=30)

        # Plot zero line
        plt.axhline(y=0, color='#E74C3C', linestyle='--')

        plt.title(title, fontsize=14, pad=20)
        plt.xlabel('Observation Points', fontsize=12)
        plt.ylabel('Residual Values', fontsize=12)
        plt.grid(True, linestyle='--', alpha=0.7)
```

```
        plt.tight_layout()
        plt.show()

    # Plot residuals
    plot_residuals(error_lstm, 'Residual Analysis for LSTM Model')
    plot_residuals(error_sarima, 'Residual Analysis for SARIMA Model')
```

Python Code	The Third Problem

```
    import numpy as np
    import matplotlib.pyplot as plt
    from tabulate import tabulate    # Import tabulate library to format tables

    # Set matplotlib to use a font that supports Chinese characters
    plt.rcParams['font.sans-serif'] = ['SimHei']    # Use SimHei font to display Chinese characters
    plt.rcParams['axes.unicode_minus'] = False    # Solve the minus sign display issue

    # Constant definitions
    c = 3.0e8    # Speed of light, unit: m/s
    f0 = 20e9    # Reference frequency, unit: Hz

    # Saastamoinen model for dry and wet delay calculation
    def dry_delay(P, z):
        """Calculate dry delay"""
        return 0.0022768 * P / np.cos(z)

    def wet_delay(e, phi, h):
        """Calculate wet delay"""
        return 0.002277 * e / (1 - 0.00266 * np.cos(2 * phi) - 0.00028 * h)

    # Herring mapping function
    def mapping_function(epsilon, a=0.0029, b=0.0005, c=0.001):
        """Herring model's mapping function"""
        return 1 / (np.sin(epsilon) + a * (np.cos(epsilon) / (1 + b / (np.sin(epsilon) + c))))

    # Layered atmospheric delay calculation model
    def layered_delay(P, e, phi, h, f, epsilon, layers=10):
        """Layered atmospheric delay calculation"""
        R = 6371.0    # Earth's radius, unit: km
        total_delay = 0.0
```

```
            h_layer = h / layers    # Height interval for each layer
            for i in range(layers):
                h_i = i * h_layer
                n_i = 1 + 0.0001 * (1 - h_i / h)
                integrand = (n_i - 1) * np.sqrt(1 + (R / (R + h_i)) ** 2 * np.tan(np.pi
/ 2 - epsilon) ** 2)
                total_delay += integrand * h_layer / c

            tau_dry = dry_delay(P, 0)    # Zenith dry delay
            tau_wet = wet_delay(e, phi, h)
            tau_zenith = tau_dry + tau_wet
            delay_corrected = tau_zenith * (f0 / f) ** 2
            m = mapping_function(epsilon)
            return delay_corrected * m + total_delay

        # Set parameters
        P = 1013.25    # Ground atmospheric pressure, unit: hPa
        e = 10.0        # Ground water vapor pressure, unit: hPa
        phi = np.radians(30)    # Latitude, unit: radians
        h = 0.5    # Height, unit: km
        f = 22e9    # Observation frequency, unit: Hz

        # Generate a range of elevation angles
        elevation_angles = np.radians(np.linspace(0.5, 10, 20))    # From 0.5° to 10°, 20
sets of data

        # Calculate delays
        delays = [layered_delay(P, e, phi, h, f, epsilon) * 1e9 for epsilon in elevation_an-
gles]
        dry_delays = [dry_delay(P, epsilon) * 1e9 for epsilon in elevation_angles]
        wet_delays = [wet_delay(e, phi, h) * mapping_function(epsilon) * 1e9 for epsi-
lon in elevation_angles]

        # Plot dry and wet delay separation diagram
        plt.figure(figsize=(14, 7))
        plt.plot(np.degrees(elevation_angles), dry_delays, label="Dry Delay (nanosec-
onds)", color='orange', linestyle='-')
        plt.plot(np.degrees(elevation_angles), wet_delays, label="Wet Delay (nanosec-
onds)", color='green', linestyle='-')
        plt.plot(np.degrees(elevation_angles), delays, label="Total Delay (nanosec-
onds)", color='blue', linestyle='--', linewidth=2)
        plt.fill_between(np.degrees(elevation_angles), dry_delays, delays,
```

```
color='green', alpha=0.3, label="Wet Delay Contribution Area")
    plt.fill_between(np.degrees(elevation_angles), 0, dry_delays, color='orange', alpha=0.3, label="Dry Delay Contribution Area")
    plt.xlabel("Elevation Angle (degrees)")
    plt.ylabel("Delay (nanoseconds)")
    plt.title("Dry and Wet Delay Separation Diagram")
    plt.legend()
    plt.grid(True)
    plt.show()

    # Prepare table data
    data = [[np.degrees(epsilon), dry, wet, total] for epsilon, dry, wet, total in zip(elevation_angles, dry_delays, wet_delays, delays)]

    # Print table
    print(tabulate(data, headers=["Elevation Angle (degrees)", "Dry Delay (nanoseconds)", "Wet Delay (nanoseconds)", "Total Delay (nanoseconds)"], tablefmt="fancy_grid"))
```

Python Code	The Fourth Problem

```
    import numpy as np
    import matplotlib.pyplot as plt
    from tabulate import tabulate    # Import tabulate library to format tables

    # Set matplotlib to use a font that supports Chinese characters
    plt.rcParams['font.sans-serif'] = ['SimHei']    # Use SimHei font to display Chinese characters
    plt.rcParams['axes.unicode_minus'] = False    # Solve the minus sign display issue

    # Constant definitions
    c = 3.0e8    # Speed of light, unit: m/s
    f0 = 20e9    # Reference frequency, unit: Hz

    # Saastamoinen model for dry and wet delay calculation
    def dry_delay(P, z):
        """Calculate dry delay"""
        return 0.0022768 * P / np.cos(z)

    def wet_delay(e, phi, h):
```

```
        """Calculate wet delay"""
        return 0.002277 * e / (1 - 0.00266 * np.cos(2 * phi) - 0.00028 * h)

    # Herring mapping function
    def mapping_function(epsilon, a=0.0029, b=0.0005, c=0.001):
        """Herring model's mapping function"""
        return 1 / (np.sin(epsilon) + a * (np.cos(epsilon) / (1 + b / (np.sin(epsilon)
+ c))))

    # Layered atmospheric delay calculation model
    def layered_delay(P, e, phi, h, f, epsilon, layers=10):
        """Layered atmospheric delay calculation"""
        R = 6371.0  # Earth's radius, unit: km
        total_delay = 0.0
        h_layer = h / layers  # Height interval for each layer
        for i in range(layers):
            h_i = i * h_layer
            n_i = 1 + 0.0001 * (1 - h_i / h)
            integrand = (n_i - 1) * np.sqrt(1 + (R / (R + h_i)) ** 2 * np.tan(np.pi
/ 2 - epsilon) ** 2)
            total_delay += integrand * h_layer / c

        tau_dry = dry_delay(P, 0)  # Zenith dry delay
        tau_wet = wet_delay(e, phi, h)
        tau_zenith = tau_dry + tau_wet
        delay_corrected = tau_zenith * (f0 / f) ** 2
        m = mapping_function(epsilon)
        return delay_corrected * m + total_delay

    # Set parameters
    P = 1013.25  # Ground atmospheric pressure, unit: hPa
    e = 10.0  # Ground water vapor pressure, unit: hPa
    phi = np.radians(30)  # Latitude, unit: radians
    h = 0.5  # Height, unit: km
    f = 22e9  # Observation frequency, unit: Hz

    # Generate a range of elevation angles
    elevation_angles = np.radians(np.linspace(0.5, 10, 20))  # From 0.5° to 10°, 20
sets of data

    # Calculate delays
```

```
    delays = [layered_delay(P, e, phi, h, f, epsilon) * 1e9 for epsilon in elevation_angles]
    dry_delays = [dry_delay(P, epsilon) * 1e9 for epsilon in elevation_angles]
    wet_delays = [wet_delay(e, phi, h) * mapping_function(epsilon) * 1e9 for epsilon in elevation_angles]

    # Plot dry and wet delay separation diagram
    plt.figure(figsize=(14, 7))
    plt.plot(np.degrees(elevation_angles), dry_delays, label="Dry Delay (nanoseconds)", color='orange', linestyle='-')
    plt.plot(np.degrees(elevation_angles), wet_delays, label="Wet Delay (nanoseconds)", color='green', linestyle='-')
    plt.plot(np.degrees(elevation_angles), delays, label="Total Delay (nanoseconds)", color='blue', linestyle='--', linewidth=2)
    plt.fill_between(np.degrees(elevation_angles), dry_delays, delays, color='green', alpha=0.3, label="Wet Delay Contribution Area")
    plt.fill_between(np.degrees(elevation_angles), 0, dry_delays, color='orange', alpha=0.3, label="Dry Delay Contribution Area")
    plt.xlabel("Elevation Angle (degrees)")
    plt.ylabel("Delay (nanoseconds)")
    plt.title("Dry and Wet Delay Separation Diagram")
    plt.legend()
    plt.grid(True)
    plt.show()

    # Prepare table data
    data = [[np.degrees(epsilon), dry, wet, total] for epsilon, dry, wet, total in zip(elevation_angles, dry_delays, wet_delays, delays)]

    # Print table
    print(tabulate(data, headers=["Elevation Angle (degrees)", "Dry Delay (nanoseconds)", "Wet Delay (nanoseconds)", "Total Delay (nanoseconds)"], tablefmt="fancy_grid"))
```