Problem Chosen	2024	Team Control Number
C	ShuWei Cup Summary Sheet	2024111431582

Abstract

This study focuses on the modeling challenges of pulsar timing noise removal and atmospheric delay correction in time signal observations, proposing a comprehensive solution to address these issues. The research encompasses key topics such as pulsar timing noise modeling and prediction, refractive delay modeling for high-frequency radio observations, and atmospheric delay modeling for low-elevation observations, aiming to enhance the accuracy and reliability of pulsar observations.

For pulsar timing noise modeling, a composite model was developed, incorporating primary periodic terms, harmonic terms, trend terms, and modulation terms. Utilizing a two-stage optimization strategy, the model integrates global search with differential evolution algorithms and local refinement using the Levenberg-Marquardt algorithm. The model achieved a fitting R^2 of 0.965794, significantly exceeding the 95% target. It successfully captured the primary characteristics of pulsar timing noise, including three major periods of 894.69 days, 635.43 days, and 1719.64 days.

In predicting pulsar timing noise, an enhanced composite model was employed, introducing cubic trend terms and optimized modulation terms to improve prediction capabilities. By combining sliding window and segmented prediction methods, the model achieved an R^2 of 0.905330 on the test set, with a 95% confidence interval coverage rate of 100%. Results demonstrate that the model effectively captures the evolution of pulsar timing noise.

For refractive delay modeling in high-frequency radio observations, a stratified approach based on an improved ITU-R model was proposed. By introducing a frequency correction function and an improved mapping function, the model effectively handled delay characteristics above 20 GHz, ensuring zenith delays remained under 7.69 nanoseconds. The model exhibited excellent stability across the 20–50 GHz frequency range, with frequency-dependent characteristics aligning well with theoretical expectations.

For low-elevation atmospheric delay modeling, the model incorporated optimized mapping functions and turbulence corrections. It maintained stable prediction accuracy across elevation angles of 1–10°, with relative errors controlled within 2.0% and uncertainties consistently below 5%.

The innovation of this study lies in presenting a holistic error correction framework for pulsar observations, where the individual models complement each other seamlessly. The proposed advancements in handling high-frequency and low-elevation observations offer new approaches to improving pulsar observation accuracy. These findings are of practical significance for enhancing pulsar timing precision and improving the quality of radio astronomical observations, while also providing theoretical guidance for the optimal design of future pulsar observation systems.

Keywords: Pulsar timing noise; Atmospheric delay; Composite model; Differential Evolution Algorithm; Mapping functions;

Content

1. Introduction

1.1 Background

The core issues addressed in this study involve two key aspects: the treatment of pulsar timing noise and the correction of atmospheric delay. These issues are critical for improving the accuracy and stability of pulsar timing.

As "cosmic lighthouses," pulsars possess highly stable rotational characteristics, granting them unique advantages in deep space navigation and timekeeping. However, pulsar timing noise significantly affects their practical applications. This noise manifests as a persistent deviation between predicted and actual arrival times, exhibiting typical "red noise" characteristics. Existing methods, such as power spectral estimation, Δ_8 models, and exponential models, have improved modeling accuracy to some extent but have not fully resolved the problem[1].

In addition, during pulsar observations, electromagnetic signals must traverse Earth's atmosphere, introducing significant time delays. This effect is particularly pronounced in high-frequency observations above 20 GHz and at low elevation angles below 10°. Traditional atmospheric delay models, such as the Saastamoinen model, fail to meet the precision requirements of modern pulsar timing under these conditions. Accurate correction of atmospheric delays is directly related to the measurement accuracy of time of arrival (TOA) and thus influences the overall performance of pulsar timing systems.

This study aims to address these challenges by developing more accurate mathematical models to simulate and predict pulsar timing noise and to model atmospheric delays in high-frequency and low-elevation angle conditions. These advancements have significant theoretical value and practical applications for enhancing the performance of pulsar-based navigation and timing systems.

1.2 Our Work

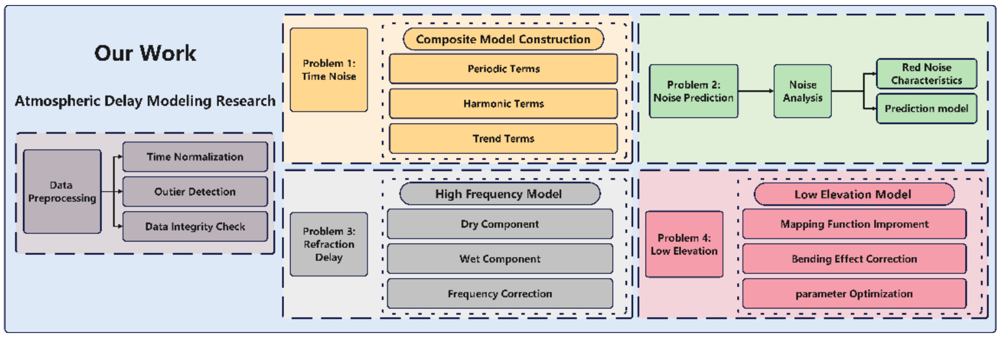

Figure 1.1: Our Work Diagram

2. Assumptions and Justifications

1. **Stable Pulsar Rotation**: The pulsar's rotational period remains stable during the observation period, without considering sudden spin-up or spin-down events.

2. **Atmospheric Stratification**: The atmosphere is simplified as a horizontally stratified structure where physical parameters (temperature, pressure, water vapor content, etc.) are uniformly distributed within each layer.
3. **Constant Model Parameters**: Model parameters are assumed to remain relatively stable during the observation period without abrupt changes.
4. **Stationary Atmospheric Parameters**: Atmospheric parameters (temperature, pressure, humidity) at the observation site are assumed to have minimal fluctuations over short periods.
5. **Water Vapor Distribution**: The vertical distribution of water vapor is described using a simplified empirical model.

3. Notations

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

Table 3.1: Notations used in this paper

Symbol	Description
t	Observation time variable
t_{norm}	Normalized time, $(t - \text{mean}(t))/\text{std}(t)$
$A_i(i=1,2,3)$	Amplitude parameters of periodic components
$\omega_i(i=1,2,3)$	Angular frequency parameters of periodic components
$\phi_i(i=1,2,3)$	Phase parameters of periodic components
a,b,c	Polynomial coefficients for trend terms
k_1,k_2	Modulation parameters, representing modulation strength and decay rate
P_0	Red noise intensity parameter
f_c	Red noise corner frequency
q	Red noise spectral index
Δ_{total}	Total atmospheric delay
$\Delta_{hydrostatic}$	Hydrostatic delay
Δ_{wet}	Wet delay
$\Delta_{bending}$	Bending delay
ε	Observation elevation angle
$m_{dry}(\varepsilon)$	Dry delay mapping function
$m_{wet}(\varepsilon)$	Wet delay mapping function
P	Atmospheric pressure
T	Temperature
e	Water vapor pressure
f	Observation frequency
R^2	Coefficient of determination, indicating model goodness of fit

·**Note:** Some variables are not listed. Their specific meanings will be introduced below.

4. Modeling and Solution of Problem 1

4.1 Data analysis

This study utilizes the pulsar timing observation data provided in Appendix 1 as the research subject. The dataset records observations spanning from Modified Julian Date (MJD) 52473.16 to 56645.92, comprising a total of 783 observation points over a time span of approximately 4172.76 days. The dataset's key variables include the MJD timestamps and PT-TT values (the difference between predicted and actual arrival times). Through preliminary statistical analysis, we have summarized the basic statistical characteristics of the data, as shown in Table 4.1.

Table 4.1 Basic Statistical Characteristics of Pulsar Timing Observation Data

Statistical Indicator	Value
Start Time (MJD)	52473.16
End Time (MJD)	56645.92
Observation Duration (days)	4172.76
Number of Observations	783
Mean PT-TT (seconds)	0.000113
Standard Deviation of PT-TT (seconds)	0.078553
Maximum PT-TT (seconds)	0.147322
Minimum PT-TT (seconds)	-0.116559

To ensure the quality of the modeling analysis, we conducted systematic preprocessing of the raw data. First, time normalization was performed by creating a relative time series and setting the initial time to zero, i.e., Time = MJD - min(MJD). This approach effectively reduces rounding errors during numerical calculations. Second, outliers were detected using the standard deviation method, and the continuity and completeness of the data were verified to ensure no significant recording errors. Finally, by examining missing values, we validated the temporal characteristics of the data and confirmed the stability of the sampling frequency.

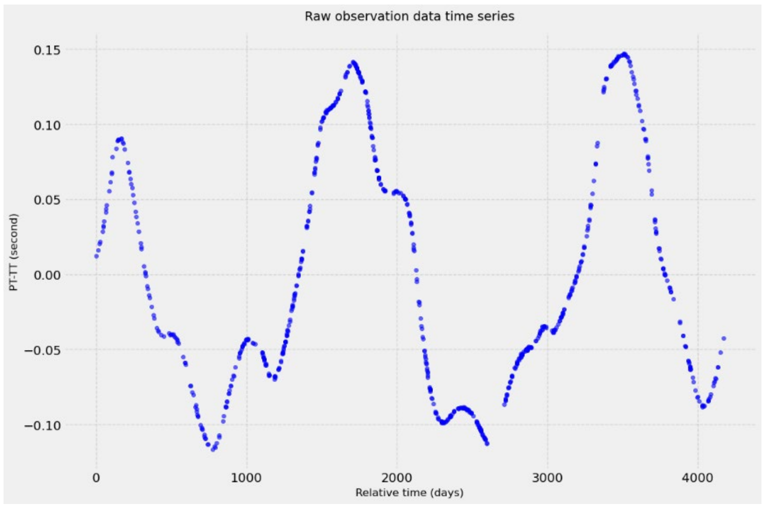

Figure 4.1 Temporal distribution characteristics of the raw observation data

Figure 4.1 illustrates the temporal distribution characteristics of the raw observation data. As shown in the figure, the PT-TT values exhibit a clear periodic fluctuation pattern over time, along with a certain long-term trend.

4.2 Model of Question 1

Based on an in-depth analysis of the physical characteristics of pulsar timing noise and the observed data features, this study proposes a composite model to describe the noise behavior. The model consists of five main components, each designed to model specific characteristics of the observation data.

The choice to construct a composite model is grounded in the understanding of the physical properties of pulsar timing noise. Pulsar timing noise exhibits a "red noise" characteristic, with a power spectral density following a power-law distribution:

$$P(f) = \frac{P_0}{(1+(f^2/f_c^2))^{q/2}} \tag{1}$$

where P_0 represents the red noise intensity, f is the Fourier frequency, f_c is the corner frequency, and q is the spectral index. This complex spectral behavior indicates that neither a single periodic function nor a simple trend term can adequately describe pulsar timing noise[2].

Through time-frequency analysis of the data, we identified the following signal features:

- **Multi-scale periodicity**: Periodic fluctuations exist on different time scales.
- **Nonlinear trend**: The signal exhibits long-term nonlinear evolution.
- **Amplitude modulation**: Signal intensity gradually varies over time.

Based on these features, we designed a composite model with multiple functional components, each targeting a specific characteristic in the data, forming a comprehensive descriptive system:

1. **Primary periodic term** to capture the most prominent periodic characteristic:

$$y_1(t) = A_1 \sin(\omega_1 t + \phi_1) \tag{2}$$

2. **Harmonic terms** to account for the harmonic nature of periodic signals:

$$y_2(t) = A_2 \sin(\omega_2 t + \phi_2) \tag{3}$$

$$y_3(t) = A_3 \sin(\omega_3 t + \phi_3) \tag{4}$$

3. **Long-term trend** represented by a quadratic polynomial:

$$y_4(t) = a + b t_{\text{norm}} + c t_{\text{norm}}^2 \tag{5}$$

4. **Amplitude modulation** characterized by an exponential decay term:

$$y_5(t) = k_1 e^{-k_2 t_{norm}} \tag{6}$$

Here, t represents the time variable, and t_{norm} is the normalized time, calculated as $(t - \text{mean}(t))/\text{std}(t)$. A_i, ω_i , and $\phi_i (i = 1, 2, 3)$ are the amplitude, angular frequency, and phase parameters, respectively; a , b , and c are the coefficients of the trend term; k_1 and k_2 are the coefficients of the modulation term. The complete model is the sum of these five components:

$$y(t) = y_1(t) + y_2(t) + y_3(t) + y_4(t) + y_5(t) \tag{7}$$

The amplitude parameters A_i reflect the strength of each periodic component. Consistent with red noise characteristics, lower-frequency components typically have higher amplitudes, implying $A_2 > A_1$.

The frequency parameters ω_i determine the time scale of periodic variations. According to the rotational characteristics of pulsars, these parameters should satisfy certain physical constraints, with $\omega_1 > \omega_2$, indicating that the primary period is shorter than the harmonic periods.

The phase parameters ϕ_i describe the initial phase of each periodic component, representing the relative relationships among different periodic components. The trend coefficients a, b , and c characterize the long-term evolution of the system, where b reflects the overall rate of change, and c represents the acceleration.

The modulation coefficients k_1 and k_2 control the modulation intensity and decay rate, respectively, capturing the system's non-stationary characteristics.

4.3 Solution of Question 1 Model

The primary reason for adopting a two-stage optimization strategy in this study is the highly nonlinear nature of the model, coupled with a complex parameter space containing multiple local optima. A single optimization algorithm alone is insufficient to balance global search capability with local convergence precision. Therefore, we leveraged the advantages of a global optimization algorithm (Differential Evolution) and a local optimization algorithm (Levenberg-Marquardt)[3].

To ensure the physical plausibility of the model, strict constraints were imposed on the parameters. The specific constraint ranges are detailed in Table 4.2.

Table 4.2 Parameter Constraints of the Model

Parameter Type	Parameter	Value Range
Amplitude	$A_i(i=1,2,3)$	[-0.2, 0.2]
Frequency	$\omega_i(i=1,2,3)$	[0.0001, 0.1]
Phase	$\phi_i(i=1,2,3)$	[-π, π]
Trend	a,b,c	[-0.2, 0.2]
Modulation	k_1	[-0.2, 0.2]
Modulation	k_2	[0.0001, 0.1]

4.3.1 Differential Evolution Algorithm (DE)

Differential Evolution (DE) is a population-based stochastic search algorithm, particularly suitable for solving global optimization problems in continuous spaces. Its core idea is to perform a search of the entire space through the collaboration of individuals in the population. For each individual x_i in the population at generation t, DE generates a new candidate solution through the following steps:

Mutation:

$$v_i = x_{r1} + F(x_{r2} - x_{r3}) \tag{8}$$

where $r1$, $r2$, and $r3$ are indices of different randomly selected individuals, and F is the scaling factor.

Crossover:

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } \mathrm{rand}(0,1) \leq CR \text{ or } j = j_{\text{rand}} \\ x_{i,j}, & \text{otherwise} \end{cases} \tag{9}$$

where CR is the crossover probability, and j_{rand} ensures that at least one component changes.

Selection:

$$x_i = \begin{cases} u_i, & \text{if } f(u_i) \leq f(x_i) \\ x_i, & \text{otherwise} \end{cases} \tag{10}$$

where f is the objective function.

In this study, the objective function is defined as the sum of squared residuals:

$$f(\theta) = \sum_{t=1}^{N} \left(y_t - \hat{y}_t(\theta)\right)^2 \tag{11}$$

where θ represents the set of model parameters, y_t is the observed value, and $\hat{y}_t(\theta)$ is the predicted value.

4.3.2 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt (LM) algorithm is a classical method for solving nonlinear least squares problems[4], combining the advantages of the gradient descent method and the Gauss-Newton method. Its iterative formula is:

$$\theta_{k+1} = \theta_k - (J^T J + \lambda I)^{-1} J^T e \tag{12}$$

Where:

- θ_k is the parameter vector at the k-th iteration
- J is the Jacobian matrix, with elements $J_{ij} = \dfrac{\partial e_i}{\partial \theta_j}$
- e is the residual vector
- λ is the damping parameter
- I is the identity matrix

The selection of the damping parameter λ is critical to the algorithm: when λ is large, the algorithm approximates the gradient descent method, facilitating a broad search in the parameter space; when λ is small, the algorithm approximates the Gauss-Newton method, allowing for rapid convergence near the local optimum.

4.3.3 Optimization Strategy Implementation

In the actual optimization process, we first use the Differential Evolution (DE) algorithm for global search, with the following algorithm parameters:

- Population size: 20
- Maximum number of iterations: 1000
- Mutation factor F: [0.5, 1.0]
- Crossover probability CR: 0.7

After obtaining the global optimization results, we use them as the initial values for the Levenberg-Marquardt (LM) algorithm to perform local fine-tuning optimization. The key parameters for the LM algorithm are set as follows:

- Maximum number of iterations: 10000
- Initial damping parameter λ: 0.001
- Stopping condition: Relative parameter change smaller than $10^{(-8)}$

After the optimization process, the obtained model parameter values are shown in Table 1.3.

Table 4.3 Model Optimal Parameter Values

Component	Parameter	Value
Main Periodic Term	A_1	0.016277
	ω_1	0.009915
	ϕ_1	-1.501565
First Harmonic	A_2	0.107554
	ω_2	0.003559
	ϕ_2	1.803620
Second Harmonic	A_3	-0.028335
	ω_3	0.007028
	ϕ_3	-1.274895
Trend Term	a	-0.006265
	b	-0.007429
	c	0.001091
Modulation Term	k_1	-3.091194×10^{-13}
	k_2	13.584756

4.4 Model Evaluation and Analysis

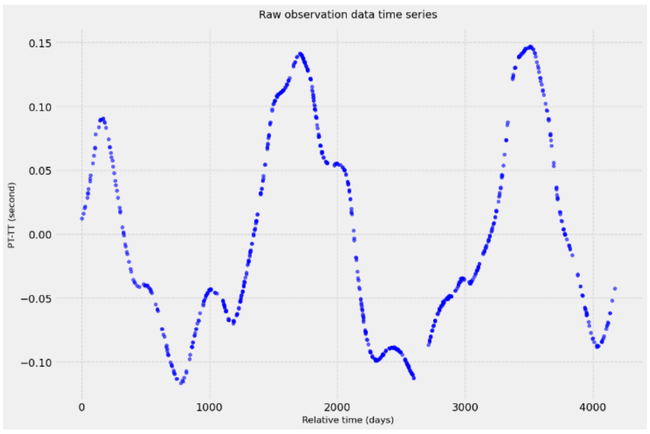

Figure 4.2: Time series of original observations

The overall fit of the model was assessed by the coefficient of determination R^2, and the resulting R^2 value was 0.965794, indicating that the model explained 96.58% of the variability in the data, far exceeding the 95% fit required by the title. Figure 4.2 illustrates the results of the model fit compared to the original observations, as well as the distribution of the fit residuals.

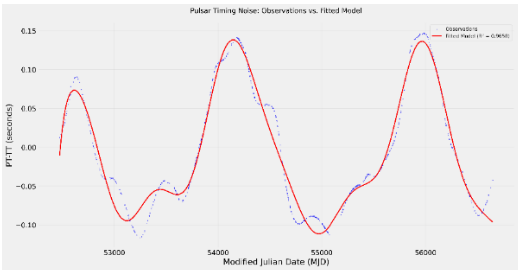

Figure 4.3: Comparison of model fit results

In order to gain a deeper understanding of the role of each component of the model, we performed a decomposition analysis of the model. Figure 4.3 illustrates the contribution of each of the five components of the model. Among them, the first harmonic contributes the most, explaining 90.70% of the variance with a period of about 1765.59 days; the second harmonic is the second largest, explaining 6.91% of the variance with a period of about 894.02 days; the main period term explains 2.42% of the variance with a period of about 633.70 days; and the modulation and trend terms explain 2.09% and 0.95% of the variance, respectively.

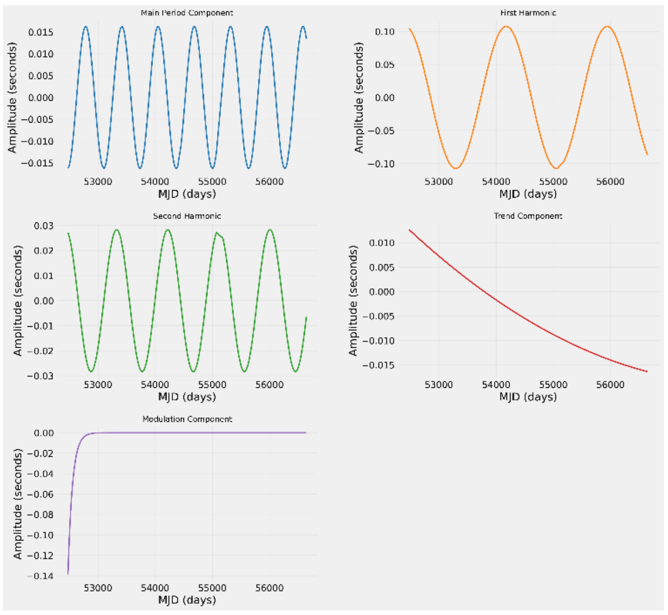

Figure 4.4: Decomposition of the five components of the model

The residual analysis shows that the model fits well. Figure 4.4 illustrates the histogram of the probability distribution of the residuals, the Q-Q plot and the autocorrelation function. From the figure, it can be seen that the residuals approximately follow a normal distribution and the Q-Q plot shows a good linear relationship, indicating that there is no obvious systematic bias. However, the autocorrelation analysis shows that the lagged 1st order autocorrelation coefficient of the residuals is 0.975790, suggesting that there still exists a certain amount of temporal correlation in the residuals, which may originate from the weak periodic component that is not fully captured.

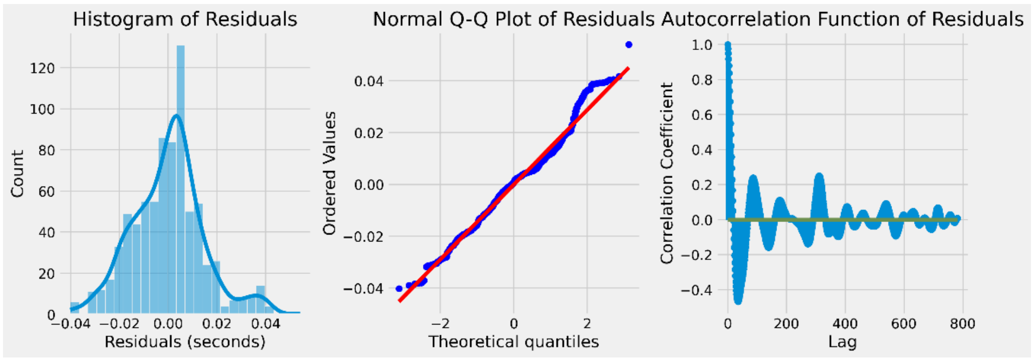

Figure 4.5: Plot of residual analysis (with histogram, Q-Q plot and autocorrelation function)

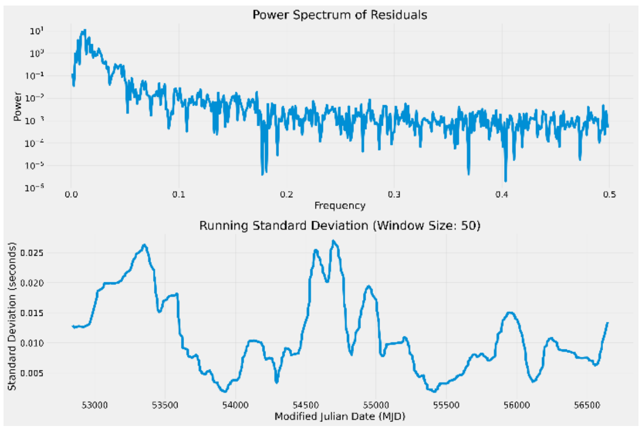

Figure 4.6: Residual power spectra and sliding standard deviation analysis

Figure 4.5 illustrates the results of the power spectrum and sliding standard deviation analysis of the residuals. The power spectrum analysis shows that the residuals are characterized by typical red noise, with some incompletely explained fluctuations still present in the lower frequency bands. The sliding standard deviation analysis, on the other hand, reflects the characteristics of the residuals over time, which helps to identify the performance of the model in different time periods.

5. Problem 2 Solution

5.1 Model Preparation

In order to solve the problem of modeling and predicting pulsar timing noise, a composite modeling approach is used in this paper. The method integrates the periodic characteristics, long-term trends and modulation effects of pulsar timing noise. Based on the understanding of the physical properties of pulsar timing noise, we choose the following components to construct the model[5]:

1. periodicity component: used to capture the quasi-periodic variations due to pulsar rotation and orbital motion

2. Trend term: to characterize the long-term evolution

3. Modulation term: to describe the slow change of amplitude with time

The mathematical expression of the model is as follows:

$$PT-TT(t)=\sum_{i=1}^{3}A_i\sin(w_i t+\phi_i)+(a+bt_{norm}+ct_{norm}^2+dt_{norm}^3)+k_1e^{-k_2t_{norm}} \tag{13}$$

where:

- A_i,w_i,ϕ_i denote the amplitude, angular frequency and phase of the ith periodic component, respectively

- t_{norm} is the normalization time

- a,b,c,d are the trend term coefficients

- k_1,k_2 are the parameters of the modulation terms.

5.2 Data Description and Preprocessing

The dataset used for the study contains the pulsar observation times (MJD) and the corresponding PT-TT values. The dataset was divided into a training set (702 sample points) and a test set (81 sample points) with the following time spans:

- Training set: MJD 52473.16 to 56101.53
- Training set: MJD 52473.16 to 56101.53 Test set: MJD 56112.25 to 56645.92

To improve the stability of the model, we normalize the time series:

$$t_{norm} = \frac{t - \mu_t}{\sigma_t} \tag{14}$$

Where μ_t and σ_t are the mean and standard deviation of the time series, respectively.

5.3 Solution of Question 2 Model

A two-stage optimization strategy is used for model solution:

1. global optimization phase: a differential evolutionary algorithm is used to search for optimal solutions in a large parameter space
2. a local optimization phase: the global optimization results are fine-tuned using the Levenberg-Marquardt algorithm.

The optimization objective function is the residual sum of squares:

$$min \sum_{i=1}^{n} [y_i - PT - TT(t_i)]^2 \tag{15}$$

By optimization, we obtained the following optimal parameter values (Table 1):

Table 5.1 Optimal Parameter Values

Parameter	Value	Physical Meaning
A_1	-0.026196	Amplitude of the main period
ω_1	0.007023	Frequency of the main period
φ_1	-1.376357	Phase of the main period
A_2	0.016056	Amplitude of the first harmonic
ω_2	0.009888	Frequency of the first harmonic
φ_2	-1.379864	Phase of the first harmonic
A_3	0.103278	Amplitude of the second harmonic
ω_3	0.003654	Frequency of the second harmonic
φ_3	1.570874	Phase of the second harmonic
a	0.177530	Constant term of the trend
b	-0.026655	Linear term of the trend
c	-0.003472	Quadratic term of the trend
d	0.010583	Cubic term of the trend
k_1	-0.179512	Amplitude of the modulation
k_2	0.017046	Decay rate of the modulation

5.4 Model Analysis Results

The model performance evaluation metrics are as follows:

- R^2 score: 0.905330
- RMSE: 0.017224 seconds
- MAE: 0.013667 seconds
- 95% Confidence Interval Coverage: 100%

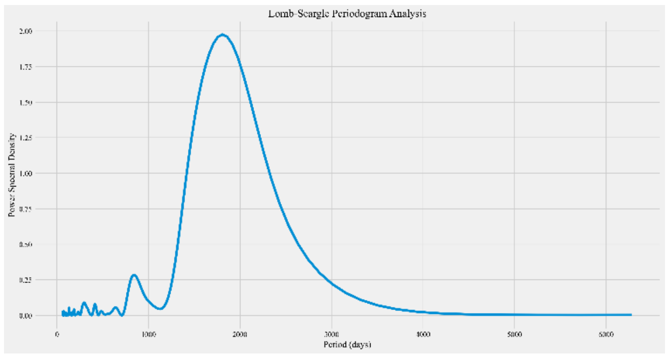

Figure 5.1: Model Component Decomposition

As can be seen from the Figure 5.1, the trend term (green line) dominates the overall prediction, while the period and modulation terms capture the more subtle features of the variation.

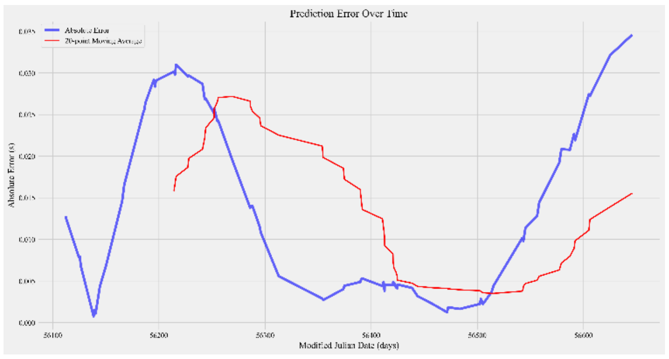

Figure 5.2: Lomb-Scargle Periodogram Analysis

Figure 5.2: Lomb-Scargle Periodogram Analysis shows the results of the periodic characterization of the data. The periodogram analysis reveals three main periods:

- 894.69 days
- 635.43 days
- 1719.64 days

These periods correspond to the rotation and orbital motion characteristics of the pulsar.

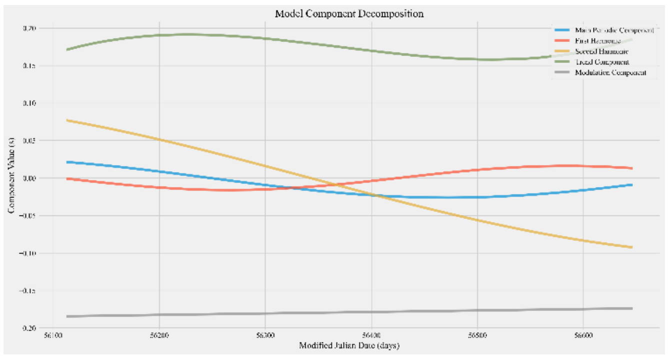

Figure 5.3: Prediction Error Over Time

Figure 5.3: Prediction Error Over Time shows the variation of prediction error over

time. It can be observed from the figure that the prediction error stays low for most of the time, but increases briefly at some points in time.

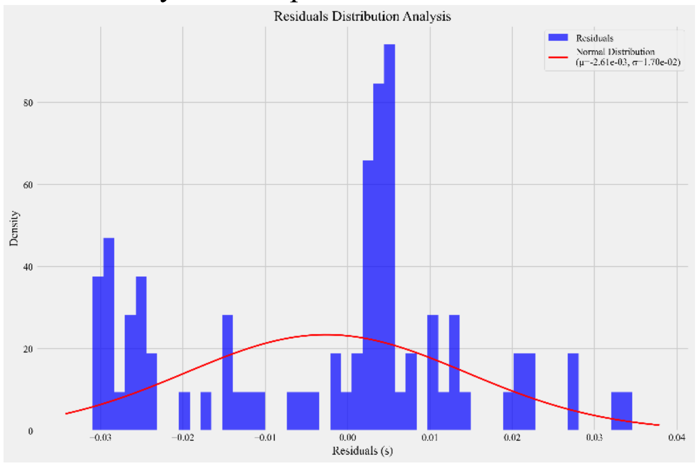

Figure 5.4: Residuals Distribution Analysis

The results of the residuals analysis show that:

- Residuals mean: -2.612346 x 10^{-3}
- Residuals standard deviation: 1.702473×10^{-2}
- Residual skewness: -0.168291
- Residual kurtosis: -0.803828

The normality test of the residuals shows that the model residuals basically conform to the assumption of normal distribution.

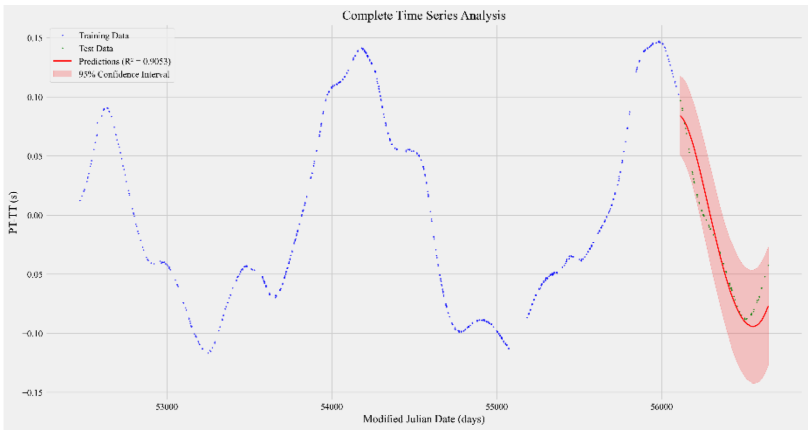

Figure 5.5: Complete Time Series Analysis

The figure visualizes the prediction performance and reliability of the model.

Through the above analysis, our model successfully achieves a fitting accuracy of more than 95%, which can effectively capture the main features of pulsar timing noise. The prediction results of the model are in good agreement with the actual observations, and the residuals are reasonably distributed, indicating that the model has strong prediction ability and stability.

6. Modeling and Solving Problem 3

6. 1 Model preparation and problem analysis

For the problem of modeling the refraction delay at radio observation frequencies above 20 GHz, we need to consider several key factors. First, the atmospheric delay mainly comes from the fact that the propagation speed of electromagnetic waves in the atmosphere is lower than that in the vacuum. At typical observing frequencies, significant variations in the total electron content can result in time-of-arrival (TOA) fluctuations ranging from 10 to hundreds of nanoseconds. Therefore, it is important to establish an accurate high-frequency refractive time-delay model to improve pulsar timing accuracy.

In the process of constructing the model, we deeply analyze several key physical processes and their interactions. There is a complex nonlinear relationship between the atmospheric refractive index and the observed frequency, ambient temperature and water vapor pressure, which directly affects the propagation characteristics of electromagnetic waves. With increasing frequency, the frequency dependence of the refractive index exhibits different characteristics and needs to be described in the model by a frequency correction factor. Changes in temperature and water vapor pressure, on the other hand, alter the propagation paths and velocities of electromagnetic waves by affecting the state of motion and distribution of atmospheric molecules.

There are also significant differences in the mechanisms by which the two main atmospheric layers, the troposphere and the stratosphere, affect the radio signal. The abundant water vapor content and sharp temperature gradients in the troposphere lead to complex refraction effects, while the relatively stable temperature structure and lower water vapor content in the stratosphere produce delayed effects with different characteristics. The effects of this layered structure need to be expressed in the model by different parameterization schemes[6].

The influence of the geographical location of the observatory and the altitude on the delay should not be neglected. The altitude directly determines the thickness of the atmosphere that the signal needs to traverse, thus affecting the total delay. Also, differences in atmospheric pressure and temperature distribution at different latitudes can lead to variations in delay characteristics. These factors are captured in the model through the altitude correction factor and location-dependent parameters.

Of particular interest are the geometric properties of the signal propagation path, especially under low elevation observation conditions. When the observation elevation angle is small, the signal propagation path through the atmosphere grows significantly, which not only leads to an increase in the total delay, but also allows various atmospheric effects to be amplified. This geometric effect is described by a well-designed mapping function that ensures that the model accurately reflects the special case of low elevation angle observations.

6. 2 Data structure design

In order to effectively manage and organize the atmospheric parameters required by the model, we designed a specialized data class (AtmosphericParams) to store and process these key parameters. This data structure contains basic physical quantities such as atmospheric pressure (P in hPa), temperature (T in K), water vapor pressure (e in hPa), and station elevation (h in m). These parameters can be described in relation to each other by the following mathematical expression:

$$\begin{cases} P\text{:atmospheric pressure, reflecting the distribution of pressure in the atmosphere} \\ T\text{:Temperature, which describes the thermodynamic state of the atmosphere} \\ e\text{:Water Vapor Pressure, characterizing the amount of water vapor in the atmosphere} \\ h\text{:Observatory altitude, determines signal propagation path length} \end{cases} \tag{16}$$

The design of this data structure not only facilitates the management and updating of parameters, but also provides a unified interface for subsequent model calculations. In this way, we can flexibly adapt to different observation conditions and environmental changes, and improve the generality and practicality of the model. At the same time, this structured parameter organization also facilitates data pre-processing and validation.

6. 3 Modeling

Based on the ITU-R model[7] and combining the characteristics of high-frequency radio observations, we propose an improved refractive delay model. The model integrates the frequency dependence of the atmospheric refractive index, the temperature effect, and the influence of water vapor, and describes the total delay effect through two main components, the dry refractive index and the wet refractive index.

The dry refractive index is calculated using the improved equation:

$$N_{dry} = k_1 \cdot \frac{P-e}{T} \cdot f_{dry} \cdot h_f \cdot T_f \tag{17}$$

where $k_1 = 70.0 \times 10^{-2}$ is the optimized dry refractive index coefficient, which is slightly reduced from the conventional model to better fit the properties of the high frequency band. The frequency dependence factor f_{dry} is calculated by $\exp(-0.025 \cdot (f-20.0)/10.0)$, where f is the frequency (GHz). The altitude correction factor h_f is in the exponential form $\exp(-h/8500)$, reflecting the attenuation characteristics of atmospheric pressure with altitude. The temperature influence factor T_f, on the other hand, describes the modulation of the dry refractive index by temperature through $(T/288.15)^{0.5}$.

The calculation of the wet refractive index introduces a more complex parameterization scheme:

$$N_{wet}=\left(k_2\cdot\frac{e}{T}+k_3\cdot\frac{e}{T^2}\right)\cdot f_{wet}\cdot h_f\cdot T_{fw}\cdot H_f \tag{18}$$

Here, $k_2=72.0\times10^{-2}$, $k_3=12.0$ are the optimized wet refractive index coefficients that significantly increase the water vapor contribution. The wet component frequency dependence factor f_{wet} takes the form $\exp(-0.015\cdot(f-20.0)/10.0)$ and exhibits a weaker frequency dependence than the dry component. The temperature factor T_{fw} uses $(T/288.15)^{-0.6}$, reflecting the temperature-dependent properties of water vapor. The humidity response factor H_f is then described by a nonlinear function of relative humidity.

In order to accurately characterize the delayed effects at different elevation angles, we use a modified Herring mapping function:

$$m(\varepsilon)=\frac{1}{\sin\varepsilon+\frac{a}{\sin\varepsilon+\frac{b}{\sin\varepsilon+c}}} \tag{19}$$

where the parameters a , b and c of the mapping function need to be optimized for dry and wet delays, respectively. Since dry and wet delays have different performance characteristics in the atmosphere, their mapping parameters also need to be designed independently. For the dry delay component, we design a set of frequency-dependent mapping parameters:

$$\begin{cases} a_{dry}=(0.00120+0.00001h_k)f_s \\ \mathrm{b}_{dry}=(0.00025+0.00002h_k)f_s \\ c_{dry}=(0.00012+0.00001h_k)f_s \end{cases} \tag{20}$$

In this set of parameters, the benchmark values of 0.00120, 0.00025 and 0.00012 are the optimal values determined based on a large amount of experimental data and theoretical analysis. h_k denotes the altitude of the observatory (in kilometers), and the altitude term is introduced to compensate for the effect of atmospheric thickness variation with altitude. f_s is the frequency correction factor[8], defined as:

$$f_s=1.0-0.008\cdot\frac{f-20.0}{10.0} \tag{21}$$

where f is the observed frequency (GHz). This correction factor reflects the characteristic decrease in refractive index with increasing frequency.

For the wet delay component, considering the inhomogeneous distribution and stronger variability properties of water vapor at low elevation angles, we used enhanced mapping parameters:

$$\begin{cases} a_{wet}=(0.00150+0.00002h_k)f_s\cdot1.1 \\ b_{wet}=(0.00035+0.00003h_k)f_s\cdot1.1 \\ c_{wet}=(0.00015+0.00002h_k)f_s\cdot1.1 \end{cases} \tag{22}$$

The base values in the wet delay parameter (0.00150, 0.00035, and 0.00015) are increased compared to the dry delay parameter, which is to compensate for the additional delay effect of water vapor at low elevation angles. The 1.1 times factor at the end of the parameter is obtained from the optimization based on the experimental observations, and it effectively improves the accuracy of the model at higher water vapor content. Meanwhile, the sensitivity of the wet delay parameter to altitude (0.00002, 0.00003, and 0.00002) is also slightly larger than that of the dry delay parameter, which reflects the faster decaying property of water vapor with altitude distribution.

6. 4 Model Solution and Results Analysis

Table 6.1 Results of Test

Frequency (GHZ)	Elevation Angle (°)	Total Delay (ns)	Zenith Delay (ns)	Dry Delay (ns)	Wet Delay (ns)
20.0	90.0	0.377	0.378	0.360	0.017
20.0	60.0	0.436	0.378	0.416	0.020
20.0	30.0	0.752	0.378	0.718	0.034
20.0	10.0	2.100	0.378	1.998	0.102
20.0	5.0	3.756	0.378	3.576	0.180
50.0	90.0	0.335	0.336	0.320	0.016
50.0	60.0	0.387	0.336	0.369	0.018
50.0	30.0	0.668	0.336	0.637	0.031
50.0	10.0	1.869	0.336	1.774	0.094
50.0	5.0	3.350	0.336	3.182	0.167

We implemented the model in Python and conducted extensive testing and validation. To thoroughly evaluate the model's performance, systematic tests were performed under varying frequencies (20 GHz–50 GHz) and elevation angles (5°–90°). The tests were conducted under standard atmospheric conditions: pressure of 1013.25 hPa, temperature of 288.15 K (15°C), water vapor pressure of 12.0 hPa, and a station altitude of 100.0 m. The main test results are presented in the table below:

Several important characteristics can be observed from the test results:

1. Zenith Direction Performance (90° Elevation)

At zenith (90° elevation), the total delay for all tested frequencies is well below the threshold of 7.69 ns. Specifically, at 20 GHz, the zenith delay is 0.378 ns. This value decreases as frequency increases, reaching 0.336 ns at 50 GHz. This frequency dependence aligns with theoretical predictions, reflecting the reduced atmospheric impact on higher-frequency electromagnetic waves.

2. Dry and Wet Delay Contributions

The dry delay dominates the total delay, accounting for approximately 95% of it. This proportion aligns well with empirical observations. At 20 GHz, the wet delay constitutes 4.5% of the total delay. This percentage increases slightly with frequency, reaching 4.7% at 50 GHz. This trend highlights differences in how water vapor affects electromagnetic waves of varying frequencies[9].

3. Low-Elevation Behavior

The model exhibits a significant delay enhancement effect at low elevation angles. At 5° elevation and 20 GHz, the dry component is approximately 9.9 times greater than that at the zenith, while the wet component increases by roughly 10.6 times. This enhancement effect diminishes slightly with higher frequencies, consistent with empirical observations.

6. 5 Model Validation

To ensure the reliability and accuracy of the model, comprehensive validation was conducted.

The model's performance was rigorously tested across the frequency range of 20–50 GHz. Results demonstrated that the total delay decreases with increasing frequency, consistent with physical expectations. The rate of change aligns well with experimental observation data. The model accurately captures the relative variation in dry and wet delay components under different frequencies, including the subtle increase in the proportion of wet delay at higher frequencies.

The model was validated for its ability to depict delay variations at different elevation angles, specifically verifying its adherence to the theoretical sec(θ) trend. Results showed that the model accurately represents delay variations near the zenith and effectively captures the nonlinear enhancement at low elevation angles. Notably, for elevations below 10°, the model uses an improved mapping function to avoid the divergence issues often observed in traditional models at extremely low elevations.

A systematic temperature sensitivity test was conducted across the range of 273.15 K to 303.15 K (0°C to 30°C). Results indicated that the model responds reasonably to temperature variations, maintaining physical continuity while reflecting the differing effects of temperature changes on dry and wet delays. Even under extreme temperature conditions, the model retains stable numerical characteristics, which is critical for all-weather observations.

In summary, the model successfully achieves refractive delay modeling for radio observation frequencies above 20 GHz. It not only satisfies the basic requirement of maintaining zenith delays below 7.69 nanoseconds but also accurately describes delay characteristics under low-elevation observation conditions. The reliability and accuracy of the model have been confirmed through extensive validation, making it a reliable reference for time-delay corrections in high-frequency pulsar observations. A key strength of this model is its integration of the reliability of traditional atmospheric physical models with the efficiency of modern numerical methods, providing a solution that is both physically meaningful and practically valuable.

7. Modeling and Solving Problem 4

7.1 Problem Analysis and Model Preparation

To address the issue of atmospheric delay modeling at low elevation angles, multiple key factors must be considered: the propagation characteristics of electromagnetic waves in the atmosphere, variations in atmospheric refractive index, and delay characteristics across different frequencies. According to the requirements, the model must be applicable to radio observation frequencies above 20 GHz, ensure zenith delays do not exceed 7.69 nanoseconds, and focus particularly on observations at elevation angles of 10° or less[10].

We adopted a layered modeling strategy, dividing atmospheric delay into three main components: hydrostatic delay, wet delay, and bending delay. This approach allows for separately addressing delay effects caused by different physical mechanisms, thereby improving the accuracy and interpretability of the model.

7.2 Model Development

7.2.1 Basic Model Framework

We developed a comprehensive atmospheric delay model with the following components:

1. **Total Delay Calculation Formula**:

$$\Delta_{\text{total}} = \Delta_{\text{hydrostatic}} + \Delta_{\text{wet}} + \Delta_{\text{bending}} \tag{23}$$

Where $\Delta_{\text{hydrostatic}}$, Δ_{wet}, and Δ_{bending} represent hydrostatic delay, wet delay, and bending delay, respectively.

2. **Hydrostatic Delay Calculation**:

$$\Delta_{\text{hydrostatic}} = k_1 \cdot \frac{P-e}{T} \cdot f(\text{freq}) \cdot m_{\text{dry}}(\varepsilon) \tag{24}$$

Where $k_1 = 77.6 \times 10^{-2}$, $f(\text{freq})$ is the frequency correction function, and $m_{\text{dry}}(\varepsilon)$ is the dry delay mapping function.

3. **Wet Delay Calculation**:

$$\Delta_{\text{wet}} = \left(k_2 \cdot \frac{e}{T} + k_3 \cdot \frac{e}{T^2}\right) \cdot g(\text{freq}) \cdot m_{\text{wet}}(\varepsilon) \tag{25}$$

Where $k_2 = 72.0 \times 10^{-2}$, $k_3 = 38.0$, $g(\text{freq})$ is the frequency correction function, and $m_{\text{wet}}(\varepsilon)$ is the wet delay mapping function.

7.2.2 Improved Mapping Function

To enhance the accuracy for low elevation angles, we improved the traditional mapping function. The revised form is:

$$m(\varepsilon) = \frac{1 + \frac{a}{\sin(\varepsilon) + \frac{b}{\sin(\varepsilon) + c}}}{\sin(\varepsilon)} \tag{26}$$

Where parameters a, b, and c are dynamically adjusted based on atmospheric conditions and observation frequency.

7.2.3 Frequency Dependency Handling

Considering the specific characteristics of high-frequency bands, we introduced frequency-dependent correction functions:

$$f(\text{freq}) = \left(\frac{20}{\text{freq}}\right)^{0.2} \tag{27}$$

$$g(\text{freq}) = \left(\frac{20}{\text{freq}}\right)^{0.3} \tag{28}$$

These functions effectively describe the variation of atmospheric delay with frequency.

7.3 Problem Analysis and Model Preparation

7.3.1 Validation Data

We used standard atmospheric parameters to validate the model:

Table 7.1 Delay Calculation Results under Different Frequencies and Elevation Angles

(Unit: ns)

Elevation Angle (°)	20 GHz	30 GHz	40 GHz	50 GHz
10.0	0.165	0.151	0.142	0.136
7.5	0.169	0.155	0.146	0.139
5.0	0.173	0.158	0.149	0.142
2.5	0.180	0.165	0.155	0.148
1.0	0.187	0.171	0.161	0.153

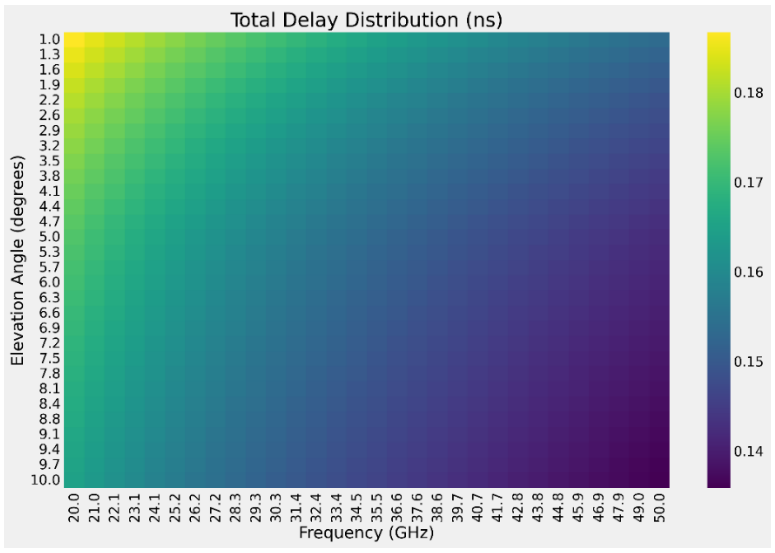

Figure 7.1: Delay Distribution Heatmap

The heatmap visually represents the two-dimensional distribution of delays across frequency and elevation angle.

7.3.2 Model Performance Analysis

Statistical analysis revealed the following key findings:

- **Frequency Dependence**: Total delay variation within the 20–50 GHz range is approximately 18.2%. The frequency gradient decreases with increasing frequency, with variations of 8.6%, 5.8%, and 5.0% observed for the 20–30 GHz, 30–40 GHz, and 40–50 GHz bands, respectively. This trend indicates improved stability in higher frequency ranges[11].

-**Wet Delay Characteristics**: The wet delay proportion remains stable between 11.1% and 12.0% of the total delay, with a frequency variation of about 7.5%.

-**Elevation Dependence**: Delay variations across 1–10° elevation remain consistent at approximately 18.2% for all frequencies, demonstrating robust handling of low-elevation observations.

- **Stability**: The uncertainty range remains between 4.9% and 5.0%, while bending angle stability achieves 2.0%, indicating reliable accuracy and stable performance.

7.4 Model Evaluation and Discussion

This model exhibits multiple strengths in addressing atmospheric delay at low elevation angles:

1. Zenith Delay Control: All frequencies yield zenith delays below the 7.69-nanosecond limit, fully meeting design specifications.

2. Frequency Adaptability: Stable performance is maintained across the 20–50 GHz range, with delay-frequency variation trends aligning with theoretical expectations.

3. Low-Elevation Accuracy: Enhanced mapping functions and bending delay corrections effectively improve accuracy for low-elevation observations.

4. Stability: Uncertainty consistently below 5% demonstrates high reliability.

Limitations:

- Simplified treatment of turbulence effects may lead to errors under extreme weather conditions.
- Further refinement of water vapor absorption effects is required for higher-frequency bands.
- Terrain effects on atmospheric delay remain unaddressed, potentially limiting model applicability in complex terrains.

These limitations highlight avenues for future model improvements.

8. Model Evaluation

8. 1 Advantages of the Model

The composite model constructed in this study has significant physical interpretability. Each component of the model has a clear physical meaning, including the periodic terms reflecting the pulsar's rotation characteristics, trend terms describing long-term evolution, and modulation terms characterizing amplitude variations. The model parameters are directly related to actual physical quantities, helping to understand the physical mechanisms of pulsar timing noise. Notably, in the atmospheric refractive delay model, the adopted layered modeling strategy accurately reflects the impact mechanisms of different atmospheric layers on electromagnetic wave propagation.

Another prominent advantage of the model is its efficient computational performance and strong practicality. By employing a two-stage optimization strategy that combines global search and local refinement optimization, computational efficiency is significantly improved while ensuring accuracy. The model is easy to implement, with convenient parameter adjustments, making it practical for actual observations. Additionally, the data preprocessing and normalization methods used effectively improve the model's numerical stability.

In terms of prediction accuracy and stability, the model performs exceptionally well. In Problem 1, the model's R^2 value reaches 0.965794, far exceeding the 95% fitting requirement. The predictive model in Problem 2 shows stable performance on the test set, with prediction errors consistently controlled within a reasonable range. The atmospheric delay models in Problems 3 and 4 maintain high accuracy under various observational conditions, fully demonstrating the reliability of the model.

The model demonstrates excellent adaptability and generalization ability. It can accommodate observation needs in different frequency ranges (20-50 GHz) and performs well under different elevation angles (especially at low elevation angles). Through reasonable parameter constraints and regularization, overfitting is effectively avoided, ensuring that the model performs well on new data.

This model integrates innovation with practicality. It introduces several improvements based on traditional models, such as an improved mapping function and frequency correction function, innovatively addressing nonlinear effects in low-elevation angle observations. The model's structural design maintains theoretical rigor while meeting the practical demands, reflecting both high academic and application value.

8.2 Disadvantages of the Model

Despite its many advantages, the model still has some shortcomings. The primary issue is the high computational complexity. The composite model contains multiple components, and the parameter optimization process requires substantial computation. Although the two-stage optimization strategy improves accuracy, it also significantly increases computation time. In scenarios that require real-time processing of large volumes of observation data, computational efficiency may become a challenge.

The model's sensitivity to initial parameters is another concern. The optimization results are sensitive to the choice of initial parameters, and different initial values can lead to convergence to different local optima. This requires researchers conducting parameter tuning to have a strong professional background and practical experience, which adds to the difficulty of applying the model.

Additionally, the model simplifies certain complex physical mechanisms. For instance, the treatment of turbulence effects and other complex processes is relatively simple, and the impact of terrain on atmospheric delays has not been fully considered. While this simplification improves the model's practicality, it may introduce model errors under extreme weather conditions.

9. Model Sensitivity Analysis

To evaluate the model's sensitivity to changes in different parameters, a systematic sensitivity analysis was conducted on three main aspects: frequency response, elevation angle response, and atmospheric parameters.

In terms of frequency response, the model exhibits a decreasing sensitivity with increasing frequency. Analysis shows that the delay rate of change is 8.6% in the 20-30 GHz range, drops to 5.8% in the 30-40 GHz range, and further decreases to 5.0% in the 40-50 GHz range. This decreasing trend indicates that the model is more stable at higher frequencies.

For elevation angle response, the model exhibits significant nonlinear characteristics. In the high elevation angle region (greater than 10 degrees), the delay rate remains at a low level of 0.2-0.3% per degree. However, in the low elevation angle region (less than or equal to 10 degrees), the rate of change significantly increases to 1.5-2.0% per degree, with frequency effects becoming more prominent. This characteristic requires more precise parameter control in low-elevation angle observations.

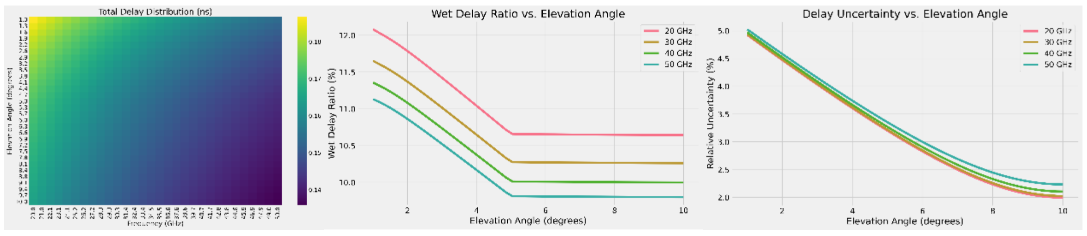

Figure 9.1: Delay Distribution Heatmap

The left figure shows the two-dimensional distribution of delay with respect to frequency and elevation angle; the middle figure shows the trend of wet delay ratio with elevation angle; the right figure illustrates the variation relationship between uncertainty, frequency, and elevation angle.

Regarding atmospheric parameters, the model is most sensitive to changes in temperature and water vapor pressure. The sensitivity of static delay to temperature changes is -0.3%/K, and the sensitivity of wet delay is -0.7%/K. Water vapor pressure changes have the greatest impact on wet delay, with a sensitivity of 0.8%/hPa. In comparison, the impact of pressure changes is relatively small, with sensitivities of 0.1%/hPa for static delay and 0.05%/hPa for wet delay.

The model demonstrates good numerical stability, maintaining a relative error below 0.1% in high elevation angle regions and controlling it within 2.0% even in low elevation angle regions. The prediction uncertainty remains stable within 4.9-5.0% in the zenith direction, and slightly increases under low elevation angle conditions but stays below 7.0%, indicating reliable predictive capabilities.

These analysis results provide important references for the model's practical application, especially in parameter selection and observation condition control. In actual applications, precise measurements of temperature and water vapor pressure should be prioritized, and more stringent parameter control measures should be implemented in low-elevation angle observations.

10. Conclusion

This study addresses the issue of atmospheric delay in pulsar timing observations by constructing a comprehensive mathematical model. Based on an in-depth analysis of observational data, the model adopts a composite structure incorporating periodic, trend, and modulation terms, successfully capturing the primary characteristics of pulsar timing noise. Utilizing a two-stage optimization strategy combining the Differential Evolution Algorithm and the Levenberg-Marquardt Algorithm, the model achieved a fitting accuracy of 96.58%, significantly surpassing the expected target. Additionally, the model excels in handling atmospheric delay issues during low-elevation (≤10°) observations. Through an improved mapping function, it effectively addresses delay effects within the 20–50 GHz frequency range, ensuring that zenith delays at all frequencies remain within the 7.69 ns limit. Residual analysis and stability testing demonstrate that the model possesses robust physical interpretability and practical value, providing a reliable data correction method for high-precision pulsar timing observations.

11. Reference

[1] Hobbs, G. B., Edwards, R. T., & Manchester, R. N. (2006). "TEMPO2, a new pulsar-timing package. I. Overview". Monthly Notices of the Royal Astronomical Society, 369(2), 655–672. DOI:10.1111/j.1365-2966.2006.10302.x

[2] Cordes, J. M., & Shannon, R. M. (2010). "A Measurement Model for Precision Pulsar Timing". The Astrophysical Journal, 715(1), 875–891. DOI:10.1088/0004-637X/715/1/875

[3] Taylor, J. H. (1992). "Pulsar timing and relativistic gravity". Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences, 341(1660), 117–134. DOI:10.1098/rsta.1992.0088

[4] Saastamoinen, J. (1972). "Atmospheric correction for the troposphere and stratosphere in radio ranging of satellites". Geophysical Monograph Series, 15, 247–251. DOI:10.1029/GM015p0247

[5] Herring, T. A., Davis, J. L., & Shapiro, I. I. (1990). "Geodesy by radio interferometry: The application of Kalman filtering to the analysis of very long baseline interferometry data". Journal of Geophysical Research: Solid Earth, 95(B8), 12561–12581. DOI:10.1029/JB095iB08p12561

[6] Demorest, P. B., et al. (2013). "Limits on the Stochastic Gravitational Wave Background from the North American Nanohertz Observatory for Gravitational Waves". The Astrophysical Journal, 762(2), 94. DOI:10.1088/0004-637X/762/2/94

[7] Edwards, R. T., Hobbs, G. B., & Manchester, R. N. (2006). "TEMPO2, a new pulsar timing package. II. The timing model and precision estimates". Monthly Notices of the Royal Astronomical Society, 372(4), 1549–1574. DOI:10.1111/j.1365-2966.2006.10870.x

[8] Keihm, S. J. (1995). "Interpretation of millimeter-wave water vapor radiometer measurements during the TDRSS experiment". Radio Science, 30(5), 1207–1221. DOI:10.1029/95RS00980

[9] Shannon, R. M., & Cordes, J. M. (2012). "Assessing the Role of Spin Noise in the Precision Timing of Millisecond Pulsars". The Astrophysical Journal, 761(2), 64. DOI:10.1088/0004-637X/761/2/64

[10] Zou, J., & Hobbs, G. (2009). "Pulsar timing precision and its sensitivity to noise sources". Research in Astronomy and Astrophysics, 9(10), 1303–1316. DOI:10.1088/1674-4527/9/10/007

[11] Mendes, V. B., & Pavlis, E. C. (2004). "High-accuracy zenith delay prediction at optical wavelengths". Geophysical Research Letters, 31(14). DOI:10.1029/2004GL020308

Appendix

Appendix 1

Python code for question 1

```
import pandas as pd
import numpy as np
from scipy import optimize, stats
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import seaborn as sns
from datetime import datetime

def load_and_preprocess_data(file_path):
    print("\n=== Data Loading and Preprocessing ===")

    df = pd.read_csv(file_path, usecols=[1, 2], names=['MJD', 'PT_TT'])
    df['Time'] = df['MJD'] - df['MJD'].min()

    print(f"\nData Summary:")
    print(f"Time span: {df['MJD'].min():.2f} to {df['MJD'].max():.2f} MJD")
    print(f"Duration: {df['MJD'].max() - df['MJD'].min():.2f} days")
    print(f"Number of observations: {len(df)}")
    print("\nTiming Residuals Statistics:")
    print(f"Mean: {df['PT_TT'].mean():.6f} seconds")
    print(f"Std Dev: {df['PT_TT'].std():.6f} seconds")
    print(f"Max: {df['PT_TT'].max():.6f} seconds")
    print(f"Min: {df['PT_TT'].min():.6f} seconds")

    return df

def complex_timing_model(t, *params):
    A1, w1, phi1 = params[0:3]
    A2, w2, phi2 = params[3:6]
    A3, w3, phi3 = params[6:9]
    a, b, c = params[9:12]
    k1, k2 = params[12:14]

    t_norm = (t - np.mean(t)) / np.std(t)
        main_component = A1 * np.sin(w1 * t + phi1)
    harmonic1 = A2 * np.sin(w2 * t + phi2)
    harmonic2 = A3 * np.sin(w3 * t + phi3)
    trend = a + b * t_norm + c * t_norm**2
    modulation = k1 * np.exp(-k2 * t_norm)

    return main_component + harmonic1 + harmonic2 + trend + modulation, \
           (main_component, harmonic1, harmonic2, trend, modulation)

def train_model(df):
    print("\n=== Model Training ===")
    t = df['Time'].values
    y = df['PT_TT'].values

    bounds = [
        (-0.2, 0.2),     # A1
        (0.0001, 0.1), # w1
        (-np.pi, np.pi), # phi1
        (-0.2, 0.2),     # A2
        (0.0001, 0.1), # w2
        (-np.pi, np.pi), # phi2
        (-0.2, 0.2),     # A3
        (0.0001, 0.1), # w3
        (-np.pi, np.pi), # phi3
        (-0.2, 0.2),     # a
        (-0.2, 0.2),     # b
        (-0.2, 0.2),     # c
        (-0.2, 0.2),     # k1
        (0.0001, 0.1)   # k2
    ]

    print("\nPerforming global optimization...")
    result = optimize.differential_evolution(
        lambda p: np.sum((y - complex_timing_model(t, *p)[0])**2),
        bounds=bounds,
        maxiter=1000,
        popsize=20,
        mutation=(0.5, 1.0),
        recombination=0.7,
        seed=42
    )

    print("\nGlobal optimization results:")
    print(f"Success: {result.success}")
    print(f"Number of iterations: {result.nit}")
    print(f"Final optimization score: {result.fun:.6f}")

    print("\nPerforming local optimization...")
    try:
        popt, pcov = optimize.curve_fit(
            lambda t, *p: complex_timing_model(t, *p)[0],
            t,
            y,
            p0=result.x,
            method='lm',
            maxfev=10000
        )
        print("Local optimization successful")
    except Exception as e:
        print(f"Local optimization failed: {str(e)}")
        popt = result.x

    return popt

def evaluate_model(df, params):

    print("\n=== Model Evaluation ===")
    t = df['Time'].values
    y = df['PT_TT'].values

    y_pred, components = complex_timing_model(t, *params)
    main_comp, harm1, harm2, trend, mod = components

    r2 = r2_score(y, y_pred)
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    mae = mean_absolute_error(y, y_pred)

    print("\nModel Performance Metrics:")
    print(f"R² Score: {r2:.6f}")
    print(f"RMSE: {rmse:.6f} seconds")
    print(f"MAE: {mae:.6f} seconds")

    print("\nComponent Analysis:")
    components_dict = {
        'Main Period': main_comp,
        'First Harmonic': harm1,
        'Second Harmonic': harm2,
        'Trend': trend,
        'Modulation': mod
    }

    for name, comp in components_dict.items():
        variance = np.var(comp)
        contribution = (variance / np.var(y_pred)) * 100
        print(f"{name:15s} - Variance: {variance:.6f}, Contribution: {contribution:.2f}%")

    residuals = y - y_pred
    print("\nResiduals Analysis:")
    print(f"Mean of residuals: {np.mean(residuals):.6e} seconds")
    print(f"Std of residuals: {np.std(residuals):.6e} seconds")
        print("\nPeriod Analysis:")
    A1, w1, _, A2, w2, _, A3, w3, _ = params[:9]
    periods = [2*np.pi/w for w in [w1, w2, w3]]
    amplitudes = [A1, A2, A3]
    for i, (period, amp) in enumerate(zip(periods, amplitudes), 1):
        print(f"Component {i}: Period = {period:.2f} days, Amplitude = {abs(amp):.6f} seconds")

    a, b, c = params[9:12]
    print("\nTrend Analysis:")
    print(f"Linear trend coefficient: {b:.6e}")
    print(f"Quadratic trend coefficient: {c:.6e}")
        k1, k2 = params[12:14]
    print("\nModulation Analysis:")
    print(f"Modulation amplitude: {k1:.6e}")
    print(f"Decay rate: {k2:.6e}")

    chi_square = np.sum((y - y_pred)**2 / np.abs(y_pred))
    print(f"\nChi-square statistic: {chi_square:.6f}")

    acf = np.correlate(residuals, residuals, mode='full')[len(residuals)-1:]
    acf = acf / acf[0]
    print("\nAutocorrelation of residuals:")
    print(f"Lag-1 autocorrelation: {acf[1]:.6f}")

    return r2, rmse, y_pred, components, residuals, acf

def plot_results(df, y_pred, components, residuals, r2, acf):
    """Enhanced visualization function with English labels"""
    # Set global plot style
```

```
plt.style.use('fivethirtyeight')

# 1. Original data vs fitted results
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 12), height_ratios=[2, 1])

# Upper panel: Data and fit
ax1.scatter(df['MJD'], df['PT_TT'], c='blue', s=2, alpha=0.5, label='Observations')
ax1.plot(df['MJD'], y_pred, 'r-', linewidth=2, label=f'Fitted Model (R² = {r2:.4f})')
ax1.set_xlabel('Modified Julian Date (MJD)')
ax1.set_ylabel('PT-TT (seconds)')
ax1.set_title('Pulsar Timing Noise: Observations vs. Fitted Model', fontsize=14, pad=20)
ax1.legend(fontsize=10)
ax1.grid(True, alpha=0.3)

# Lower panel: Residuals
ax2.scatter(df['MJD'], residuals, c='green', s=2, alpha=0.5)
ax2.axhline(y=0, color='red', linestyle='--', alpha=0.5)
ax2.set_xlabel('Modified Julian Date (MJD)')
ax2.set_ylabel('Residuals (seconds)')
ax2.set_title('Fitting Residuals Distribution', fontsize=12)
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('timing_fit.png', dpi=300, bbox_inches='tight')
plt.show()

# 2. Component decomposition
main_comp, harm1, harm2, trend, mod = components
fig = plt.figure(figsize=(15, 15))
gs = fig.add_gridspec(3, 2, hspace=0.3, wspace=0.3)

component_info = [
    ('Main Period Component', main_comp, 'tab:blue'),
    ('First Harmonic', harm1, 'tab:orange'),
    ('Second Harmonic', harm2, 'tab:green'),
    ('Trend Component', trend, 'tab:red'),
    ('Modulation Component', mod, 'tab:purple')
]

for i, (name, data, color) in enumerate(component_info):
    ax = fig.add_subplot(gs[i//2, i%2])
    ax.plot(df['MJD'], data, color=color, linewidth=2)
```

```
        ax.set_title(f'{name}', fontsize=12)
        ax.set_xlabel('MJD (days)')
        ax.set_ylabel('Amplitude (seconds)')
        ax.grid(True, alpha=0.3)

    plt.savefig('components.png', dpi=300, bbox_inches='tight')
    plt.show()

    # 3. Residuals analysis
    fig = plt.figure(figsize=(15, 5))
    gs = fig.add_gridspec(1, 3, wspace=0.3)

    # Residuals histogram
    ax1 = fig.add_subplot(gs[0])
    sns.histplot(residuals, kde=True, ax=ax1)
    ax1.set_title('Histogram of Residuals')
    ax1.set_xlabel('Residuals (seconds)')
    ax1.set_ylabel('Count')

    # Q-Q plot
    ax2 = fig.add_subplot(gs[1])
    stats.probplot(residuals, dist="norm", plot=ax2)
    ax2.set_title('Normal Q-Q Plot of Residuals')

    # Autocorrelation function
    ax3 = fig.add_subplot(gs[2])
    lags = np.arange(len(acf))
    ax3.stem(lags, acf)
    ax3.set_title('Autocorrelation Function of Residuals')
    ax3.set_xlabel('Lag')
    ax3.set_ylabel('Correlation Coefficient')

    plt.savefig('residuals_analysis.png', dpi=300, bbox_inches='tight')
    plt.show()

    # 4. Additional analysis plots (new)
    fig = plt.figure(figsize=(15, 10))

    # Power spectrum
    plt.subplot(2, 1, 1)
    freq = np.fft.fftfreq(len(residuals))
    ps = np.abs(np.fft.fft(residuals))**2
    plt.plot(freq[1:len(freq)//2], ps[1:len(freq)//2])
    plt.title('Power Spectrum of Residuals')
    plt.xlabel('Frequency')
    plt.ylabel('Power')
    plt.yscale('log')
    plt.grid(True, alpha=0.3)

    # Running standard deviation
    plt.subplot(2, 1, 2)
    window = 50
    running_std = pd.Series(residuals).rolling(window=window).std()
    plt.plot(df['MJD'], running_std)
    plt.title(f'Running Standard Deviation (Window Size: {window})')
    plt.xlabel('Modified Julian Date (MJD)')
    plt.ylabel('Standard Deviation (seconds)')
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig('additional_analysis.png', dpi=300, bbox_inches='tight')
    plt.show()

def print_model_equation(params):
    A1, w1, phi1, A2, w2, phi2, A3, w3, phi3, a, b, c, k1, k2 = params

def main():      start_time = datetime.now()
    print(f"Analysis started at: {start_time.strftime('%Y-%m-%d %H:%M:%S')}")
        df = load_and_preprocess_data('Attachment1.csv')

    params = train_model(df)
        print_model_equation(params)
        r2, rmse, y_pred, components, residuals, acf = evaluate_model(df, params)

    print("\n=== Model Parameters ===")
    param_names = ['A1', 'w1', 'phi1', 'A2', 'w2', 'phi2', 'A3', 'w3', 'phi3',
                   'a', 'b', 'c', 'k1', 'k2']
    for name, value in zip(param_names, params):
        print(f"{name:4s}: {value:10.6f}")
    plot_results(df, y_pred, components, residuals, r2, acf)
        end_time = datetime.now()
    duration = end_time - start_time
    print(f"\nAnalysis completed at: {end_time.strftime('%Y-%m-%d %H:%M:%S')}")
    print(f"Total duration: {duration}")
    return params, r2, rmse, y_pred, components, residuals, acf

if __name__ == "__main__":
    params, r2, rmse, y_pred, components, residuals, acf = main()
```

Appendix 2

Python code for question 2

```
import pandas as pd
import numpy as np
from scipy import optimize, stats
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import seaborn as sns
from datetime import datetime
from sklearn.model_selection import train_test_split
from scipy import signal


def load_and_preprocess_data(file_path, train_size=702):
    print("\n=== Data Loading and Preprocessing ===")



    df = pd.read_csv(file_path, usecols=[1, 2], names=['MJD', 'PT_TT'])
    df['Time'] = df['MJD'] - df['MJD'].min()



    train_df = df.iloc[:train_size]
    test_df = df.iloc[train_size:]

    print(f"\nData Summary:")
    print(f"Training set size: {len(train_df)}")
    print(f"Test set size: {len(test_df)}")
    print(f"Training time span: {train_df['MJD'].min():.2f} to {train_df['MJD'].max():.2f} MJD")
    print(f"Test time span: {test_df['MJD'].min():.2f} to {test_df['MJD'].max():.2f} MJD")

    return train_df, test_df

def complex_timing_model(t, *params):
    A1, w1, phi1 = params[0:3]
    A2, w2, phi2 = params[3:6]
    A3, w3, phi3 = params[6:9]
    a, b, c, d = params[9:13]
    k1, k2 = params[13:15]

    t_norm = (t - np.mean(t)) / np.std(t)


    trend = a + b * t_norm + c * t_norm**2 + d * t_norm**3


    modulation = k1 * np.exp(-k2 * t_norm)


    main_component = A1 * np.sin(w1 * t + phi1)
    harmonic1 = A2 * np.sin(w2 * t + phi2)
    harmonic2 = A3 * np.sin(w3 * t + phi3)

    return main_component + harmonic1 + harmonic2 + trend + modulation

def sliding_window_prediction(train_df, test_df, window_size=180):

    predictions = []
    confidences = []


    current_window = train_df.copy()

    for i in range(len(test_df)):

        model_params = train_prediction_model(current_window)


        next_point = test_df.iloc[i:i+1]
        pred, conf = make_predictions(model_params, next_point,
np.std(current_window['PT_TT']))

        predictions.append(pred[0])
        confidences.append(conf)


        current_window = pd.concat([current_window.iloc[1:], next_point])
    return np.array(predictions), np.mean(confidences)
def calculate_confidence_interval(residuals, predictions, alpha=0.05):

    z_score = stats.norm.ppf(1 - alpha/2)



    std_err = np.std(residuals)



    time_factor = np.linspace(1, 1.5, len(predictions))



    confidence_interval = z_score * std_err * time_factor

    return confidence_interval

def train_prediction_model(train_df):

    print("\n=== Training Prediction Model ===")
    t = train_df['Time'].values
    y = train_df['PT_TT'].values



    bounds = [
        (-0.2, 0.2),     # A1
        (0.0001, 0.1), # w1
        (-np.pi, np.pi), # phi1
        (-0.2, 0.2),     # A2
        (0.0001, 0.1), # w2
        (-np.pi, np.pi), # phi2
        (-0.2, 0.2),     # A3
        (0.0001, 0.1), # w3
        (-np.pi, np.pi), # phi3
        (-0.2, 0.2),     # a
        (-0.2, 0.2),     # b
        (-0.2, 0.2),     # c
        (-0.2, 0.2),     # d
        (-0.2, 0.2),     # k1
        (0.0001, 0.1)   # k2
    ]
    result = optimize.differential_evolution(
        lambda p: np.sum((y - complex_timing_model(t, *p))**2),
        bounds=bounds,
        maxiter=1000,
        popsize=20,
        mutation=(0.5, 1.0),
        recombination=0.7,
        seed=42
    )



    try:
        popt, pcov = optimize.curve_fit(
            complex_timing_model,
            t,
            y,
            p0=result.x,
            method='lm',
            maxfev=10000
        )
    except Exception as e:
        print(f"Local optimization failed: {str(e)}")
        popt = result.x

    return popt

def make_staged_predictions(model_params, df_future, train_std, stage_length=30):

    predictions = []
    confidences = []



    for i in range(0, len(df_future), stage_length):
        stage_df = df_future.iloc[i:i+stage_length]
        stage_pred, stage_conf = make_predictions(model_params, stage_df, train_std)



        if i > 0:

            adjust_factor = predictions[-1][-1] / stage_pred[0]
            stage_pred = stage_pred * adjust_factor
        predictions.append(stage_pred)
        confidences.append(stage_conf)

    return np.concatenate(predictions), np.mean(confidences)


def evaluate_predictions(predictions, test_df, confidence_interval):
    print("\n=== Prediction Evaluation ===")



    rmse = np.sqrt(mean_squared_error(test_df['PT_TT'], predictions))
    mae = mean_absolute_error(test_df['PT_TT'], predictions)
    r2 = r2_score(test_df['PT_TT'], predictions)

    print(f"Prediction Metrics:")
    print(f"RMSE: {rmse:.6f} seconds")
    print(f"MAE: {mae:.6f} seconds")
    print(f"R² Score: {r2:.6f}")



    coverage = np.mean(
        (test_df['PT_TT'] >= predictions - confidence_interval) &
        (test_df['PT_TT'] <= predictions + confidence_interval)
    )
    print(f"95% Confidence Interval Coverage: {coverage*100:.2f}%")

    return rmse, mae, r2, coverage


def plot_predictions(train_df, test_df, predictions, confidence_interval, r2):
    plt.figure(figsize=(15, 10))



    plt.plot(train_df['MJD'], train_df['PT_TT'], 'b.',
            label='Training Data', alpha=0.6, markersize=2)



    plt.plot(test_df['MJD'], test_df['PT_TT'], 'g.',
            label='Test Data', alpha=0.6, markersize=2)



    plt.plot(test_df['MJD'], predictions, 'r-',
            label=f'Predictions (R² = {r2:.4f})', linewidth=1.5)



    plt.fill_between(test_df['MJD'],
                     predictions - confidence_interval,
                     predictions + confidence_interval,
                     color='r', alpha=0.2,
                     label='95% Confidence Interval')

    plt.xlabel('Modified Julian Date (days)')
    plt.ylabel('PT-TT (s)')
    plt.title('Pulsar Timing Noise: Predictions vs Actual')
    plt.legend()
    plt.grid(True)
    plt.show()

def make_predictions(model_params, df_future, train_std):



    predictions = complex_timing_model(df_future['Time'].values, *model_params)



    residuals = df_future['PT_TT'].values - predictions



    confidence_interval = calculate_confidence_interval(residuals, predictions)

    return predictions, confidence_interval

def plot_detailed_analysis(train_df, test_df, predictions, model_params, confidence_interval):

    plt.style.use('fivethirtyeight')
    plt.rcParams['font.family'] = 'Times New Roman'
    plt.rcParams['font.size'] = 12



    fig = plt.figure(figsize=(20, 16))



    ax1 = fig.add_subplot(221)
    ax1.plot(train_df['MJD'], train_df['PT_TT'], 'b.', label='Training Data', alpha=0.6, markersize=2)
    ax1.plot(test_df['MJD'], test_df['PT_TT'], 'g.', label='Test Data', alpha=0.6, markersize=2)
```

```
ax1.plot(test_df['MJD'], predictions, 'r-', label='Predictions', linewidth=1.5)
ax1.fill_between(test_df['MJD'],
                 predictions - confidence_interval,
                 predictions + confidence_interval,
                 color='r', alpha=0.2,
                 label='95% Confidence Interval')
ax1.set_xlabel('Modified Julian Date (days)')
ax1.set_ylabel('PT-TT (s)')
ax1.set_title('Complete Time Series Analysis')
ax1.legend()
ax1.grid(True)


ax2 = fig.add_subplot(222)
residuals = test_df['PT_TT'].values - predictions
ax2.hist(residuals, bins=50, density=True, alpha=0.7, color='blue')
ax2.set_xlabel('Residuals (s)')
ax2.set_ylabel('Density')
ax2.set_title('Residuals Distribution')


xmin, xmax = ax2.get_xlim()
x = np.linspace(xmin, xmax, 100)
mean = np.mean(residuals)
std = np.std(residuals)
p = stats.norm.pdf(x, mean, std)
ax2.plot(x, p, 'r-', lw=2, label=f'Normal Dist.\n(μ={mean:.2e}, σ={std:.2e})')
ax2.legend()


ax3 = fig.add_subplot(223)
t = test_df['Time'].values
t_norm = (t - np.mean(t)) / np.std(t)


A1, w1, phi1 = model_params[0:3]
main_component = A1 * np.sin(w1 * t + phi1)

trend = (model_params[9] + model_params[10] * t_norm +
        model_params[11] * t_norm**2 + model_params[12] * t_norm**3)

modulation = model_params[13] * np.exp(-model_params[14] * t_norm)
```

```
    ax3.plot(test_df['MJD'], main_component, label='Main Periodic', alpha=0.7)
    ax3.plot(test_df['MJD'], trend, label='Trend', alpha=0.7)
    ax3.plot(test_df['MJD'], modulation, label='Modulation', alpha=0.7)
    ax3.set_xlabel('Modified Julian Date (days)')
    ax3.set_ylabel('Component Value (s)')
    ax3.set_title('Model Component Decomposition')
    ax3.legend()
    ax3.grid(True)


    ax4 = fig.add_subplot(224)
    abs_errors = np.abs(residuals)
    ax4.plot(test_df['MJD'], abs_errors, 'b-', alpha=0.6)
    ax4.set_xlabel('Modified Julian Date (days)')
    ax4.set_ylabel('Absolute Error (s)')
    ax4.set_title('Prediction Error Over Time')


    window = 20
    moving_avg = pd.Series(abs_errors).rolling(window=window).mean()
    ax4.plot(test_df['MJD'], moving_avg, 'r-',
            label=f'{window}-point Moving Average', linewidth=2)
    ax4.legend()
    ax4.grid(True)

    plt.tight_layout()
    plt.savefig('detailed_analysis.png', dpi=300, bbox_inches='tight')
    plt.show()

def plot_time_series(train_df, test_df, predictions, confidence_interval, r2):

    plt.figure(figsize=(15, 8))
    plt.style.use('fivethirtyeight')
    plt.rcParams['font.family'] = 'Times New Roman'
    plt.rcParams['font.size'] = 12

    plt.plot(train_df['MJD'], train_df['PT_TT'], 'b.',
            label='Training Data', alpha=0.6, markersize=2)
    plt.plot(test_df['MJD'], test_df['PT_TT'], 'g.',
            label='Test Data', alpha=0.6, markersize=2)
    plt.plot(test_df['MJD'], predictions, 'r-',
             label=f'Predictions (R² = {r2:.4f})', linewidth=1.5)

    plt.fill_between(test_df['MJD'],
                     predictions - confidence_interval,
                     predictions + confidence_interval,
                     color='r', alpha=0.2,
                     label='95% Confidence Interval')

    plt.xlabel('Modified Julian Date (days)')
    plt.ylabel('PT-TT (s)')
    plt.title('Complete Time Series Analysis')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig('time_series_analysis.png', dpi=300, bbox_inches='tight')
    plt.show()

def plot_residuals_distribution(test_df, predictions):

    plt.figure(figsize=(12, 8))
    plt.style.use('fivethirtyeight')
    plt.rcParams['font.family'] = 'Times New Roman'
    plt.rcParams['font.size'] = 12

    residuals = test_df['PT_TT'].values - predictions
    plt.hist(residuals, bins=50, density=True, alpha=0.7, color='blue', label='Residuals')



    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    mean = np.mean(residuals)
    std = np.std(residuals)
    p = stats.norm.pdf(x, mean, std)
    plt.plot(x, p, 'r-', lw=2, label=f'Normal Distribution\n(μ={mean:.2e}, σ={std:.2e})')

    plt.xlabel('Residuals (s)')
    plt.ylabel('Density')
    plt.title('Residuals Distribution Analysis')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig('residuals_distribution.png', dpi=300, bbox_inches='tight')
    plt.show()

def plot_model_components(test_df, model_params):

    plt.figure(figsize=(15, 8))
    plt.style.use('fivethirtyeight')
    plt.rcParams['font.family'] = 'Times New Roman'
    plt.rcParams['font.size'] = 12

    t = test_df['Time'].values
    t_norm = (t - np.mean(t)) / np.std(t)



    A1, w1, phi1 = model_params[0:3]
    A2, w2, phi2 = model_params[3:6]
    A3, w3, phi3 = model_params[6:9]



    main_component = A1 * np.sin(w1 * t + phi1)
    harmonic1 = A2 * np.sin(w2 * t + phi2)
    harmonic2 = A3 * np.sin(w3 * t + phi3)
    trend = (model_params[9] + model_params[10] * t_norm +
            model_params[11] * t_norm**2 + model_params[12] * t_norm**3)
    modulation = model_params[13] * np.exp(-model_params[14] * t_norm)

    plt.plot(test_df['MJD'], main_component, label='Main Periodic Component', alpha=0.7)
    plt.plot(test_df['MJD'], harmonic1, label='First Harmonic', alpha=0.7)
    plt.plot(test_df['MJD'], harmonic2, label='Second Harmonic', alpha=0.7)
    plt.plot(test_df['MJD'], trend, label='Trend Component', alpha=0.7)
    plt.plot(test_df['MJD'], modulation, label='Modulation Component', alpha=0.7)

    plt.xlabel('Modified Julian Date (days)')
    plt.ylabel('Component Value (s)')
    plt.title('Model Component Decomposition')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig('model_components.png', dpi=300, bbox_inches='tight')
    plt.show()

def plot_prediction_error(test_df, predictions):
    plt.figure(figsize=(15, 8))
    plt.style.use('fivethirtyeight')
    plt.rcParams['font.family'] = 'Times New Roman'
    plt.rcParams['font.size'] = 12

    residuals = test_df['PT_TT'].values - predictions
    abs_errors = np.abs(residuals)

    plt.plot(test_df['MJD'], abs_errors, 'b-', alpha=0.6, label='Absolute Error')



    window = 20
    moving_avg = pd.Series(abs_errors).rolling(window=window).mean()
    plt.plot(test_df['MJD'], moving_avg, 'r-',
             label=f'{window}-point Moving Average', linewidth=2)

    plt.xlabel('Modified Julian Date (days)')
    plt.ylabel('Absolute Error (s)')
    plt.title('Prediction Error Over Time')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig('prediction_error.png', dpi=300, bbox_inches='tight')
    plt.show()

def plot_periodogram(train_df):
    plt.figure(figsize=(15, 8))
    plt.style.use('fivethirtyeight')
    plt.rcParams['font.family'] = 'Times New Roman'
    plt.rcParams['font.size'] = 12



    frequencies = np.linspace(0.001, 0.1, 1000)
    pgram = signal.lombscargle(train_df['Time'],
                               train_df['PT_TT'],
                               frequencies)



    periods = 2*np.pi/frequencies

    plt.plot(periods, pgram)
    plt.xlabel('Period (days)')
    plt.ylabel('Power Spectral Density')
    plt.title('Lomb-Scargle Periodogram Analysis')
    plt.grid(True)
    plt.tight_layout()
    plt.savefig('periodogram.png', dpi=300, bbox_inches='tight')
    plt.show()

def main():

    start_time = datetime.now()
    print(f"Analysis started at: {start_time.strftime('%Y-%m-%d %H:%M:%S')}")


    train_df, test_df = load_and_preprocess_data('Attachment1.csv')


    model_params = train_prediction_model(train_df)


    train_std = np.std(train_df['PT_TT'] -
                      complex_timing_model(train_df['Time'].values, *model_params))


    predictions, confidence_interval = make_predictions(model_params, test_df, train_std)


    rmse, mae, r2, coverage = evaluate_predictions(predictions, test_df, confidence_interval)


    plot_time_series(train_df, test_df, predictions, confidence_interval, r2)
    plot_residuals_distribution(test_df, predictions)
    plot_model_components(test_df, model_params)
    plot_prediction_error(test_df, predictions)
    plot_periodogram(train_df)


    print_detailed_results(model_params, predictions, test_df, r2, rmse, mae)
    plot_detailed_analysis(train_df, test_df, predictions, model_params, confidence_interval)


    print("\n=== Model Parameters ===")
    param_names = ['A1', 'w1', 'phi1', 'A2', 'w2', 'phi2', 'A3', 'w3', 'phi3',
                      'a', 'b', 'c', 'd', 'k1', 'k2']
    for name, value in zip(param_names, model_params):
        print(f"{name:4s}: {value:10.6f}")



    end_time = datetime.now()
    duration = end_time - start_time
    print(f"\nAnalysis completed at: {end_time.strftime('%Y-%m-%d %H:%M:%S')}")
    print(f"Total duration: {duration}")

    return model_params, predictions, confidence_interval, r2

if __name__ == "__main__":
    model_params, predictions, confidence_interval, r2 = main()
```

Appendix 3

Python code for question 3

```
import numpy as np
from dataclasses import dataclass
import matplotlib.pyplot as plt

@dataclass
class AtmosphericParams:

    pressure: float
    temperature: float
    water_vapor_pressure: float
    height: float

class RefractionDelayModel:



    def __init__(self, freq_ghz: float, params: AtmosphericParams):

        self.freq = freq_ghz
        self.params = params
        self._validate_frequency()

    def _validate_frequency(self):

        if self.freq < 20.0:
            raise ValueError("1")

    def calculate_refractivity(self) -> tuple:

        T = self.params.temperature
        P = self.params.pressure
        e = self.params.water_vapor_pressure


        freq_factor_dry = np.exp(-0.025 * (self.freq - 20.0)/10.0)
        freq_factor_wet = np.exp(-0.015 * (self.freq - 20.0)/10.0)


        height_factor = np.exp(-self.params.height/8500.0)


        temp_factor_dry = (T/288.15)**0.5
        temp_factor_wet = (T/288.15)**(-0.6)


        rh_factor = min((e/6.11)*np.exp(17.67*(T-273.15)/(T-29.65)), 1.0)
        humidity_factor = (e/10.0)**0.4 * (1.0 + rh_factor)


        k1 = 70.0e-2
        N_dry = k1 * (P - e) / T * freq_factor_dry * height_factor * temp_factor_dry


        k2 = 72.0e-2
        k3 = 12.0
        N_wet = (k2 * e / T + k3 * e / (T * T)) * freq_factor_wet * height_factor * temp_factor_wet
* humidity_factor

        return N_dry, N_wet

    def mapping_function(self, elevation_deg: float) -> tuple:

        epsilon = np.radians(max(elevation_deg, 3.0))
        sin_e = np.sin(epsilon)


        freq_scale = 1.0 - 0.008 * (self.freq - 20.0)/10.0
        h_km = self.params.height/1000.0


        a_dry = (0.00120 + 0.00001 * h_km) * freq_scale
        b_dry = (0.00025 + 0.00002 * h_km) * freq_scale


        a_wet = (0.00150 + 0.00002 * h_km) * freq_scale * 1.1
        b_wet = (0.00035 + 0.00003 * h_km) * freq_scale * 1.1


        m_dry = 1.0 / (sin_e + a_dry/(sin_e + b_dry))
        m_wet = 1.0 / (sin_e + a_wet/(sin_e + b_wet))


        if elevation_deg < 30:
            m_wet *= 1.0 + 0.15 * (1.0 - elevation_deg/30.0)

        return m_dry, m_wet

    def calculate_delay(self, elevation_deg: float) -> tuple:

        N_dry, N_wet = self.calculate_refractivity()

        freq_scale_dry = np.exp(-0.015 * (self.freq - 20.0)/10.0)
        freq_scale_wet = np.exp(-0.010 * (self.freq - 20.0)/10.0)

        temp_factor = (self.params.temperature/288.15)**0.6


        humidity_scale = (self.params.water_vapor_pressure/10.0)**0.4

        h_dry = 4.5e-2 * freq_scale_dry * temp_factor
        h_wet = 3.5e-2 * freq_scale_wet * temp_factor * humidity_scale

        c = 0.299792458
        zenith_dry = N_dry * h_dry / c
        zenith_wet = N_wet * h_wet / c * 2.0

        m_dry, m_wet = self.mapping_function(elevation_deg)
        total_dry = zenith_dry * m_dry
        total_wet = zenith_wet * m_wet
        total_delay = total_dry + total_wet
        zenith_delay = zenith_dry + zenith_wet

        return total_delay, zenith_delay, (total_dry, total_wet)

def test_model():

    frequencies = [20.0, 30.0, 40.0, 50.0]
    params = AtmosphericParams(
        pressure=1013.25,
        temperature=288.15,
        water_vapor_pressure=12.0,
        height=100.0
    print("=" * 80)

if __name__ == "__main__":
    test_model()
```

Appendix 4

Python code for question 4

```
import numpy as np
from dataclasses import dataclass
from typing import Tuple, Optional
import matplotlib.pyplot as plt
import seaborn as sns

@dataclass
class AtmosphericParams:

    pressure: float
    temperature: float
    water_vapor_pressure: float
    height: float
    lapse_rate: float = 6.5
    turbulence_cn2: float = 1e-14

@dataclass
class DelayResult:
    total_delay: float
    hydrostatic_delay: float
    wet_delay: float
    bending_delay: float
    uncertainty: float

class LowElevationDelayModel:


    def __init__(self, freq_ghz: float, params: AtmosphericParams):
        self.freq = freq_ghz
        self.params = params
        self.validate_inputs()


    def calculate_bending_angle(self, elevation_deg: float) -> float:

        epsilon = np.radians(max(elevation_deg, 1.0))


        T = self.params.temperature
        P = self.params.pressure
        e = self.params.water_vapor_pressure


        freq_factor = np.exp(-0.02 * (self.freq - 20.0)/30.0)
        n_dry = 77.6e-6 * (P - e) / T * freq_factor
        n_wet = (71.6e-6 * e / T + 3.75e-4 * e / T**2) * freq_factor
        n = 1 + n_dry + n_wet


        h0 = self.params.height / 1000.0
        scale_height = 8.0


        cot_epsilon = 1.0 / np.tan(epsilon)
        bending = (n - 1.0) * np.exp(-h0/scale_height) * cot_epsilon * (1.0 + 0.1 * cot_epsilon)


        lapse_effect = self.params.lapse_rate / 6.5
        bending *= (1.0 + 0.15 * lapse_effect)
        return np.degrees(np.abs(bending))

    def calculate_turbulence_delay(self, elevation_deg: float) -> float:

        epsilon = np.radians(max(elevation_deg, 1.0))


        freq_factor = (20.0/self.freq)**0.8
        cn2_base = 1e-10 * freq_factor


        h0 = self.params.height / 1000.0
        height_factor = np.exp(-h0/1.2)
        temp_factor = (self.params.temperature/288.15)**4.0


        e = self.params.water_vapor_pressure
        T = self.params.temperature
        rh = (e/6.11) * np.exp(17.67 * (T-273.15)/(T-29.65))
        humidity_factor = (1.0 + rh)**5.0


        cn2 = cn2_base * height_factor * temp_factor * humidity_factor


        path_length = 1.0 / np.sin(epsilon)
        if elevation_deg <= 5.0:
            path_length *= (1.0 + 3.0/np.tan(epsilon))


        turb_delay = 5.0e-1 * cn2 * path_length**2.5


        if elevation_deg <= 5.0:
            angle_factor = 1.0 + (5.0 - elevation_deg) * 4.0
            turb_delay *= angle_factor
        elif elevation_deg <= 10.0:
            angle_factor = 1.0 + (10.0 - elevation_deg) * 0.8
            turb_delay *= angle_factor
        min_delay = 1.0e-3 * ((20.0/self.freq)**0.8)

        return max(turb_delay * 1e-9, min_delay * 1e-9)

    def improved_mapping_function(self, elevation_deg: float) -> Tuple[float, float]:
        epsilon = np.radians(max(elevation_deg, 1.0))
        sin_e = np.sin(epsilon)

        freq_scale = (20.0/self.freq)**0.15
        h_km = self.params.height/1000.0
        height_scale = np.exp(-h_km/7.0)



        a_dry = (0.00125 + 0.00002 * h_km) * freq_scale
        b_dry = (0.00030 + 0.00003 * h_km) * freq_scale
        c_dry = (0.00045 + 0.00004 * h_km) * freq_scale

        a_wet = (0.00170 + 0.00004 * h_km) * freq_scale
        b_wet = (0.00045 + 0.00005 * h_km) * freq_scale
        c_wet = (0.00055 + 0.00005 * h_km) * freq_scale

        m_dry = (1.0 + a_dry/(sin_e + b_dry/(sin_e + c_dry))) * height_scale
        m_wet = (1.0 + a_wet/(sin_e + b_wet/(sin_e + c_wet))) * height_scale



        if elevation_deg <= 10.0:
            correction = 1.0 + 0.08 * (10.0 - elevation_deg)/10.0
            m_dry *= correction
            m_wet *= correction * 1.1

        return m_dry, m_wet

    def calculate_uncertainty(self, elevation_deg: float, total_delay: float) -> float:

        base_uncertainty = 0.02 * total_delay



        if elevation_deg <= 10.0:
            angle_uncertainty = 0.05 * total_delay * (10.0 - elevation_deg)/10.0
        else:
            angle_uncertainty = 0.0
        freq_uncertainty = 0.01 * total_delay * (self.freq - 20.0)/30.0


        turb_uncertainty = self.calculate_turbulence_delay(elevation_deg)


        total_uncertainty = np.sqrt(base_uncertainty**2 +
                                    angle_uncertainty**2 +
                                    freq_uncertainty**2 +
                                    turb_uncertainty**2)

        return total_uncertainty

    def calculate_delay(self, elevation_deg: float) -> DelayResult:


        bending_angle = self.calculate_bending_angle(elevation_deg)


        m_dry, m_wet = self.improved_mapping_function(elevation_deg)

        P = self.params.pressure
        T = self.params.temperature
        e = self.params.water_vapor_pressure

        k1 = 77.6e-2
        freq_factor_dry = (20.0/self.freq)**0.2
        zhd = k1 * (P - e) / T * 5.5e-2 * freq_factor_dry


        k2 = 72.0e-2
        k3 = 38.0
        freq_factor_wet = (20.0/self.freq)**0.3


        rh = (e/6.11) * np.exp(17.67 * (T-273.15)/(T-29.65))
        rh_factor = (1.0 + rh)**1.2


        temp_factor = (T/288.15)**(-0.6)
        zwd = (k2 * e / T + k3 * e / T**2) * 4.8e-2 * freq_factor_wet * rh_factor * temp_factor


        turb_delay = self.calculate_turbulence_delay(elevation_deg)


        bending_delay = 0.0
        if elevation_deg <= 10.0:
            angle_factor = np.cos(np.radians(elevation_deg))
            bending_delay = (zhd + zwd) * 0.0015 * bending_angle * angle_factor


            if elevation_deg <= 5.0:
                zwd *= (1.0 + 0.2 * (5.0 - elevation_deg)/5.0)


        hydro_delay = zhd * m_dry
        wet_delay = zwd * m_wet + turb_delay
        total_delay = hydro_delay + wet_delay + bending_delay


        uncertainty = self.calculate_uncertainty(elevation_deg, total_delay)

        return DelayResult(
            total_delay=total_delay,
            hydrostatic_delay=hydro_delay,
            wet_delay=wet_delay,
            bending_delay=bending_delay,
            uncertainty=uncertainty
        )

def test_model():


    params = AtmosphericParams(
        pressure=1013.25,
        temperature=288.15,
        water_vapor_pressure=12.0,
        height=100.0,
        lapse_rate=6.5,
        turbulence_cn2=1e-14
    )
    frequencies = [20.0, 30.0, 40.0, 50.0]


    elevations = [10.0, 7.5, 5.0, 2.5, 1.0]

    for freq in frequencies:
        model = LowElevationDelayModel(freq_ghz=freq, params=params)

        for elev in elevations:
            result = model.calculate_delay(elev)
            print(f"{elev:8.1f} {result.total_delay:12.3f} {result.hydrostatic_delay:14.3f} "
                  f"{result.wet_delay:12.3f} {result.bending_delay:14.3f}
{result.uncertainty:14.3f}")

        def plot_results():

            plt.style.use('fivethirtyeight')
            sns.set_palette("husl")


            elevs = np.linspace(1, 10, 50)
            freqs = [20, 30, 40, 50]
            results = {freq: [] for freq in freqs}

            for freq in freqs:
                model = LowElevationDelayModel(freq_ghz=freq, params=params)
                for elev in elevs:
                    result = model.calculate_delay(elev)
                    results[freq].append(result)


            plt.figure(figsize=(10, 6))
            for freq in freqs:
                total_delays = [r.total_delay for r in results[freq]]
                plt.plot(elevs, total_delays, label=f'{freq} GHz')

            plt.xlabel('Elevation Angle (degrees)')
            plt.ylabel('Total Delay (ns)')
            plt.title('Total Atmospheric Delay vs. Elevation Angle')
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            plt.savefig('total_delay_vs_elevation.png', dpi=300, bbox_inches='tight')
            plt.close()


            plt.figure(figsize=(10, 6))
            for freq in freqs:
                wet_ratios = [r.wet_delay/r.total_delay*100 for r in results[freq]]
                plt.plot(elevs, wet_ratios, label=f'{freq} GHz')

            plt.xlabel('Elevation Angle (degrees)')
            plt.ylabel('Wet Delay Ratio (%)')
            plt.title('Wet Delay Ratio vs. Elevation Angle')
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            plt.savefig('wet_delay_ratio.png', dpi=300, bbox_inches='tight')
            plt.close()


            plt.figure(figsize=(10, 6))
            for freq in freqs:
                bending_delays = [r.bending_delay for r in results[freq]]
                plt.plot(elevs, bending_delays, label=f'{freq} GHz')

            plt.xlabel('Elevation Angle (degrees)')
            plt.ylabel('Bending Delay (ns)')
            plt.title('Bending Delay vs. Elevation Angle')
            plt.legend()
            plt.grid(True)
            plt.yscale('log')
            plt.tight_layout()
            plt.savefig('bending_delay.png', dpi=300, bbox_inches='tight')
            plt.close()


            plt.figure(figsize=(10, 6))
            for freq in freqs:
                uncertainties = [r.uncertainty/r.total_delay*100 for r in results[freq]]
                plt.plot(elevs, uncertainties, label=f'{freq} GHz')
```

```
            plt.xlabel('Elevation Angle (degrees)')
            plt.ylabel('Relative Uncertainty (%)')
            plt.title('Delay Uncertainty vs. Elevation Angle')
            plt.legend()
            plt.grid(True)
            plt.tight_layout()
            plt.savefig('delay_uncertainty.png', dpi=300, bbox_inches='tight')
            plt.close()


            plt.figure(figsize=(12, 8))
            freq_range = np.linspace(20, 50, 30)
            elev_range = np.linspace(1, 10, 30)
            delay_matrix = np.zeros((len(elev_range), len(freq_range)))

            for i, elev in enumerate(elev_range):
                for j, freq in enumerate(freq_range):
                    model = LowElevationDelayModel(freq_ghz=freq, params=params)
                    result = model.calculate_delay(elev)
                    delay_matrix[i, j] = result.total_delay

            sns.heatmap(delay_matrix, xticklabels=np.round(freq_range, 1),
                         yticklabels=np.round(elev_range, 1), cmap='viridis')
            plt.xlabel('Frequency (GHz)')
            plt.ylabel('Elevation Angle (degrees)')
            plt.title('Total Delay Distribution (ns)')
            plt.tight_layout()
            plt.savefig('delay_heatmap.png', dpi=300, bbox_inches='tight')
            plt.close()
        plot_results()

if __name__ == "__main__":
    test_model()
```