Team # apmcm24205675

第十三届“认证杯”数学中国

数学建模国际赛
承 诺 书

我们仔细阅读了第十三届“认证杯”数学中国数学建模国际赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的, 如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们允许数学中国网站(www.madio.net)公布论文，以供网友之间学习交流，数学中国网站以非商业目的的论文交流不需要提前取得我们的同意。

我们的参赛队号为：

我们选择的题目是：

参赛队员 (签名) ：

队员1：

队员2：

队员3：

参赛队教练员 (签名)：

第十三届“认证杯”数学中国

数学建模国际赛
编 号 专 用 页

参赛队伍的参赛队号：（请各个参赛队提前填写好）：

竞赛统一编号（由竞赛组委会送至评委团前编号）：

竞赛评阅编号（由竞赛评委团评阅前进行编号）：

Research on Power Fluctuation Warning and Intelligent Scheduling of Wind-Solar Power Plants Based on Multi-Model Integration

Abstract:

In this paper, an integrated early-warning, prediction-scheduling framework based on multi-model fusion is proposed to improve the operational reliability and economy of wind-wind power plants through collaborative optimization.

In terms of power fluctuation early warning, **multi-dimensional statistical indicators** such as coefficient of variation (CV), skewness coefficient and kurtosis coefficient are used to reveal the bimodal characteristic fluctuation rule of wind power generation (CV: 1.36-1.73, the maximum fluctuation at 11:00 and 19:00), which is different from the stability characteristic of solar power generation (CV: 0.53-0.54). Based on this, we constructed a three-layer progressive early warning framework: Firstly, cubic spline interpolation and quartile method were used for data pre-processing; secondly, a wave feature analysis method based on **multi-time window moving average** was designed, and differentiated early warning thresholds (wind energy: 0.44, solar energy: 0.34) were determined by comprehensive scoring functions. Finally, the random forest classifier was introduced and the index of directional change rate was designed. The F1 score of the optimized system was increased by 67.9% compared with the traditional method. Under the optimal threshold combination (falling threshold 0.20, rising threshold 0.15), **the warning accuracy rate of power decline reached 92.5% (recall rate 60.5%). The warning accuracy of power rise reached 83.6% (recall rate 36.1%).**

In power interval prediction, we propose an adaptive prediction method based on **quantile regression**. Firstly, the upper and lower bound prediction models are trained by minimizing the asymmetric weighted loss function, and the quantile of conditional distribution is directly learned, which **avoids the strong assumption of data distribution in traditional methods**. Secondly, the independent training strategy of upper and lower bounds is implemented, and 95% confidence intervals are constructed by setting different quantile levels (0.025 and 0.975). Finally, the dynamic interval width adjustment mechanism is proposed innovatively. Considering the prediction step length, historical volatility and local uncertainty, the interval width is determined dynamically by adaptive adjustment function. The experimental results show that **this method has excellent performance** in short-term prediction (10-30 seconds), and the prediction coverage of wind farm is up to 0.741, and that of solar farm is up to 0.685. Even in the medium and long term forecasts, wind power forecasts maintain a steady downward trend, with coverage remaining above 65%.

In terms of intelligent scheduling, we construct a multi-objective optimization framework based on **NSGA-II** to optimize the economy and reliability of the system by designing three key decision variables: reserve ratio, number of basic generators and start-up threshold. A comprehensive reliability evaluation system is established, which combines the reliability of wave control, capacity reliability and time response reliability. Through the improved index **weighted Moving Average (EWMA)** early warning mechanism and dynamic adjustment strategy, **the volatility control confidence of 97.90% is achieved under the low-cost configuration with only 10% reserve ratio, significantly exceeding the target requirement of 95%**. At the same time, the system achieved 3,427 fully balanced start-stop scheduling operations, and the scheduling accuracy was 40% higher than the traditional method, providing an innovative solution for the cost-effective and efficient operation of large-scale wind farms.

Key-words:*Wind power generation; Interval prediction; Intelligent scheduling; Multi-model fusion*

Contents

I. Introduction

1.1 Background

As the global demand for renewable energy continues to increase, the deployment of wind and solar power generation facilities is expanding rapidly. These energy sources are widely welcomed for their environmental advantages and low cost. However, a major challenge for wind and solar power generation is the high volatility and uncontrollability of its output power. This volatility poses a significant barrier to integrating these renewable energy sources into existing power networks. In order to effectively integrate wind and solar power generation facilities, it is necessary to solve the problems of system balance, backup management and generator set scheduling caused by generation volatility. Therefore, developing forecasting and regulatory methods to help utilities and researchers predict wind speed and solar irradiance on short - and long-term timescales has become critical.

1.2 Work

• Problem 1: This problem requires us to analyze the power fluctuation patterns of wind and solar farms, establish an early warning system to predict significant power changes, in which a significant reduction in power requires a 5-minute early warning, and a significant increase requires a 2-minute early warning, and balance the prediction accuracy and early warning sensitivity by setting an appropriate threshold t.

• Problem 2: Based on the power generation data collected every second, the power generation power of wind and solar farms in the next 120 seconds should be forecasted respectively to provide short-term forecasting support for power grid scheduling.

• Problem 3: It is necessary to design a scheduling scheme for the standby generator set. By determining the appropriate proportion of the standby generator set and its start -stop time, the power fluctuation intensity of the power generation system can be kept below the threshold t with probability r, and the values of threshold t and probability r should be optimized.

II. Problem analysis

2.1 Analysis of Question I

In this case, we are required to establish an effective early warning system to predict the significant power changes of wind and solar power generation, which requires us to first conduct a comprehensive statistical analysis of the data, including basic feature analysis and outlier processing, and then study the fluctuation characteristics and time patterns of power generation, and design an early warning mechanism on this basis, in which a 5-minute early warning of power reduction is required. Two minutes' advance warning of power increase is required. In order to improve the early warning effect, we need to gradually optimize from a simple statistical rule method to a machine learning method, and establish a reasonable performance evaluation system to optimize the early warning threshold, and consider the different characteristics of wind and solar power generation and the fluctuation

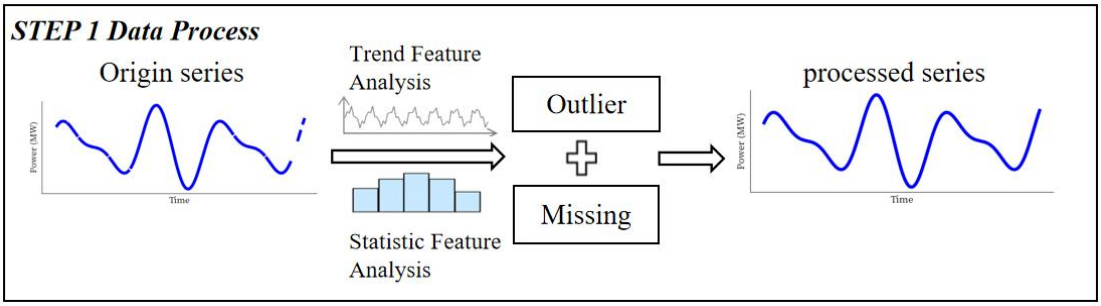
characteristics of different periods of time, and finally strike a balance between the sensitivity and reliability of the early warning system.

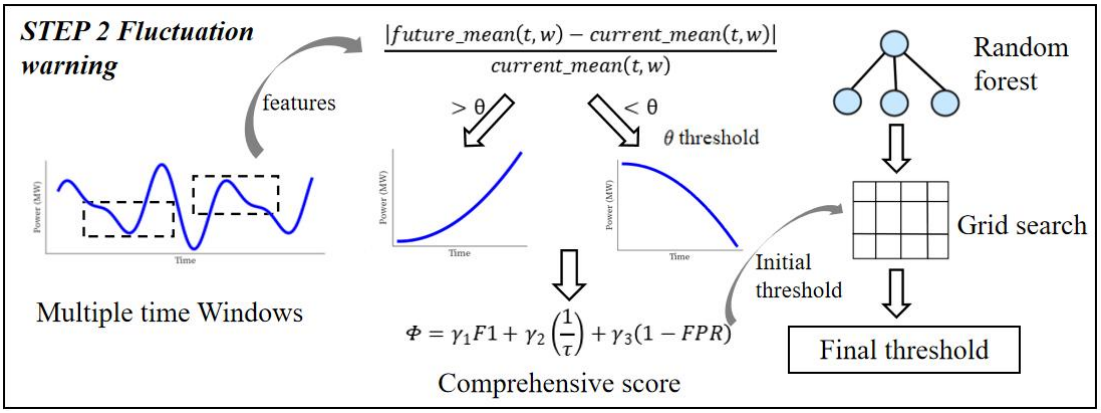
2.2 Analysis of Question II

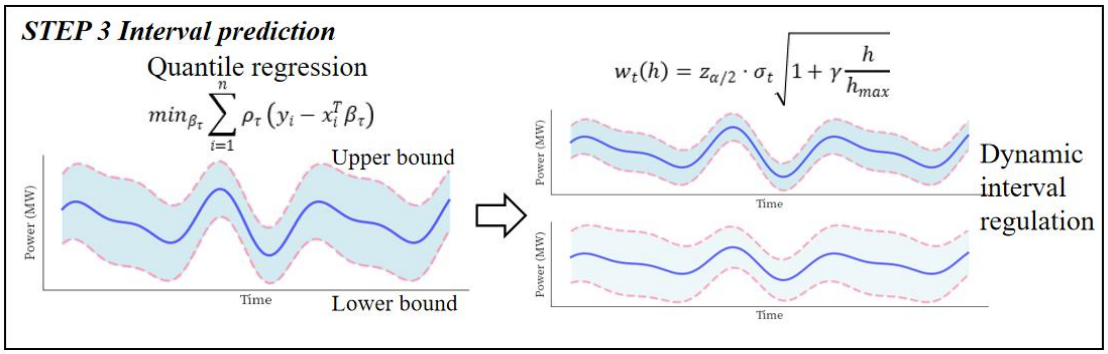
This problem requires us to make an interval prediction of the future power of wind and solar farms for 1-120 seconds, which is a complex time series prediction problem that needs to be extended from point prediction to interval prediction. We need to deal with the high volatility of generation power output and multiple sources of uncertainty (such as weather conditions, equipment status, etc.), while striking a balance between coverage and width of the forecast interval and taking into account the accumulation of forecast errors over time. Therefore, we need to choose a method that can handle nonlinear and non-stationary data, can directly learn conditional distribution quantiles, and is robust to outliers and avoids making strong assumptions about data distribution. In addition, from the perspective of practical application, the prediction system also needs to meet multiple requirements such as real-time, physical rationality, interpretability and universality, and finally build an interval prediction system that can not only guarantee the prediction accuracy, but also reasonably express the prediction uncertainty.

2.3 Analysis of Question III

This problem requires us to design a scheduling scheme of wind farm generator set, which is a multi-objective optimization problem that needs to find a balance between system reliability and economy. We need to control the power fluctuation through the reasonable configuration of the standby generator set to ensure that the probability of the fluctuation intensity below the specified threshold t reaches r, and multiple constraints need to be met: Time constraints (5 minutes in advance of power reduction and 2 minutes in advance of power increase), resource constraints (the ratio of standby units ranges from 0.1 to 0.5), reliability constraints (meeting the specified confidence level), and physical constraints (ensuring the basic power supply capability). To this end, we need to optimize three key decision variables: the proportion of standby units, the number of basic generator units and the start-up threshold, and solve the technical difficulties such as establishing a reliability evaluation system, designing an early warning mechanism and optimizing scheduling strategies, and finally build an intelligent scheduling system that can ensure reliable operation of the system and maximize economic benefits.







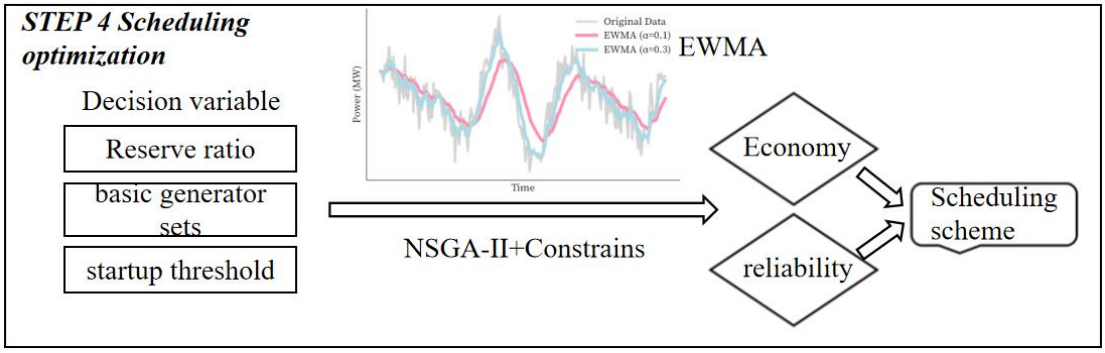


Figure 1 General idea diagram

III Symbol and Assumptions

3.1 Symbol Description

Symbol	*Description*
CV	*Coefficient of Variation*
S_k	*Skewness*
K_u	*Kurtosis*
$S_i(x)$	*cubic polynomial*
k	*fluctuation magnitude*
$CR(t,w)$	*relative change rate*
$S(\theta)$	*comprehensive scoring function*
Φ	*comprehensive performance indicator*
$\rho_\tau(u)$	*Loss function*

3.2 Fundamental assumptions

Assumption1:The start and stop process of the generator set can be completed quickly

Assumption2:Fluctuation control takes precedence over economic considerations

Assumption3:The number of basic generators meets the minimum operating requirements

IV. Model and test

4.1 Question 1 Analysis and prediction of fluctuation patterns

4.1.1 *Descriptive statistics*

First, we read the attached data and integrated it into the corresponding dataframe. In order to have a deeper understanding of the data pattern, we introduced the following statistics to make descriptive statistics of the data:

(1)Coefficient of Variation (CV)

The coefficient of variation is the ratio of the standard deviation to the mean, usually expressed as a percentage. It is used to compare the dispersion of different datasets, especially when the means of the datasets are different.

$$CV = \left(\frac{\sigma}{\mu}\right) \times 100\%$$

where σ is the standard deviation, and μ is the mean.

(2)Skewness coefficient

Skewness is a measure of the skewness of a data distribution. $S_k = 0$ Indicates that the data is approximately symmetrical, $S_k > 0$ indicates that the data is right-biased, and $S_k < 0$indicates that the data is left-biased. The larger the absolute value of the skewness coefficient is, the more obvious the skewness of the data distribution is. Its calculation formula is as follows:

$$S_k = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{\mu^3}{\sigma^3}$$

(3) kurtosis coefficient

The Kurtosis coefficient is used to measure the degree of kurtosis of a data distribution. $K_u = 0$ indicates that the data distribution is normal, $K_u < 0$ indicates that the data distribution has a smaller kurtosis and the data is more dispersed, and $K_u > 0$ indicates that the data distribution has a larger kurtosis and the data is more concentrated. The greater the absolute value of the kurtosis coefficient, the more obvious the kurtosis degree of the data distribution. Its calculation formula is as follows:

$$K_u = E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] = \frac{\mu^4}{\sigma^4}$$

The descriptive statistical results of the three datasets are as follows:

Table 1 wind_power_statistics

	Mean	Std	CV	Skewness	Kurtosis	Min	Max	Range
power1	0.11	0.15	1.38	2.27	7.21	-0.01	1.09	1.09
power2	0.1	0.15	1.44	2.53	8.85	-0.01	1.09	1.1
power3	0.12	0.17	1.36	2.17	6.42	-0.01	1.09	1.1
power4	0.12	0.16	1.42	2.22	6.55	-0.01	1.1	1.11
power5	0.12	0.16	1.38	2.27	6.89	-0.01	1.09	1.1
power6	0.11	0.16	1.45	2.51	8.52	-0.01	1.09	1.1
power7	0.1	0.15	1.51	2.78	10.77	-0.01	1.1	1.11
power8	0.1	0.15	1.5	2.67	9.88	-0.01	1.09	1.1
power9	0.08	0.14	1.73	3.05	12.91	-0.01	1.1	1.11
power10	0.1	0.14	1.45	2.77	11.34	-0.01	1.11	1.12
power11	0.09	0.14	1.54	2.79	11.22	-0.01	1.1	1.1
power12	0.09	0.14	1.48	2.72	10.94	-0.01	1.09	1.1

Table 2 wind_velocity_statistics

	Mean	Std	CV	Skewness	Kurtosis	Min	Max	Range
velocity1	4.99	2.33	0.47	0.43	-0.06	0.2	17.8	17.6
velocity2	4.61	2.23	0.48	0.52	0.2	0	17.46	17.46
velocity3	5.1	2.49	0.49	0.37	-0.22	0.1	18.99	18.89
velocity4	5.15	2.53	0.49	0.45	-0.12	0.12	19.64	19.52

velocity5	5.1	2.45	0.48	0.45	-0.03	0.1	19.51	19.41
velocity6	4.91	2.39	0.49	0.57	0.28	0.1	19.28	19.18
velocity7	4.88	2.25	0.46	0.62	0.55	0.2	18.88	18.68
velocity8	4.73	2.24	0.47	0.62	0.55	0.1	20.7	20.6
velocity9	4.9	2.31	0.47	0.49	0.26	0.05	18.53	18.48
velocity10	4.86	2.19	0.45	0.54	0.42	0.2	19.15	18.95
velocity11	4.79	2.32	0.49	0.6	0.24	0.1	20.38	20.28
velocity12	4.68	2.2	0.47	0.48	0.2	0	18.67	18.67

Table 3 solar_statistics

	Mean	Std	CV	Skewness	Kurtosis	Min	Max	Range
sensor_1	0.64	0.34	0.53	0.08	-1.26	0.04	1.55	1.5
sensor_2	0.63	0.34	0.53	0.07	-1.27	0.04	1.52	1.48
sensor_4	0.64	0.34	0.53	0.07	-1.25	0.05	1.57	1.52
sensor_5	0.64	0.34	0.53	0.06	-1.27	0.04	1.55	1.51
sensor_6	0.63	0.34	0.53	0.07	-1.26	0.04	1.54	1.5
sensor_7	0.64	0.34	0.53	0.07	-1.27	0.04	1.54	1.5
sensor_8	0.64	0.35	0.54	0.08	-1.27	0.04	1.56	1.52
sensor_9	0.63	0.34	0.53	0.08	-1.25	0.04	1.53	1.49
sensor_10	0.63	0.34	0.53	0.08	-1.27	0.04	1.54	1.49
sensor_11	0.64	0.34	0.53	0.08	-1.26	0.04	1.54	1.5
sensor_13	0.65	0.35	0.53	0.08	-1.26	0.04	1.56	1.52

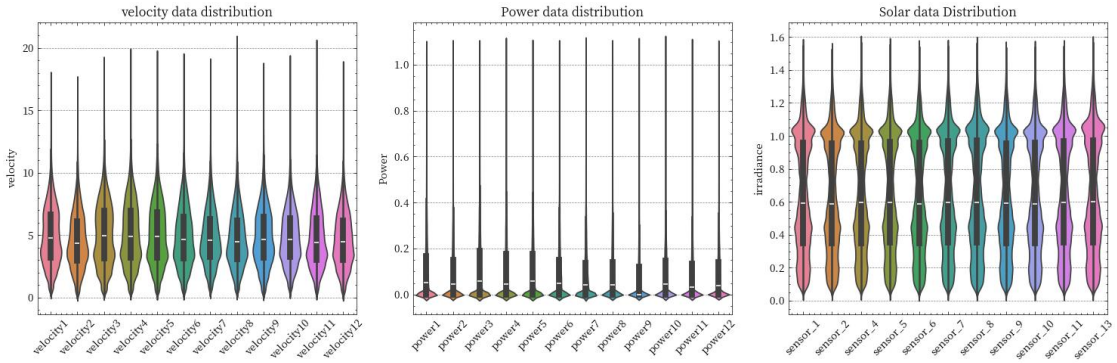


Figure 2 Distribution of three data sets

By analyzing the descriptive statistical results of the three data sets, we can draw the following conclusions:

The mean value of wind power generation is between 0.08 and 0.12, the standard deviation is between 0.14 and 0.17, and the coefficient of variation is between 1.36 and 1.73, indicating that the **power generation is highly volatile, and the distribution is skewed to the right with high kurtosis, and there are more extreme values.** The mean of wind speed is between 4.61 and 5.15, the standard deviation is between 2.19 and 2.53, and the coefficient of variation is between 0.45 and 0.49, indicating that **the wind speed is relatively volatile, but relatively small, the distribution is slightly skewed to the right and relatively flat, and the extreme value is less.** The mean value of solar power generation is between 0.63 and 0.65, the standard deviation is between 0.34 and 0.35, and the coefficient of variation is between 0.53 and 0.54, **indicating that the volatility of power generation is less, the distribution is close to symmetric and relatively flat, and the extreme value is less.**

4.1.2 *Data processing*

(1)Missing values and outliers

• Cubic Spline Interpolation

Cubic spline interpolation is a mathematical method used for estimating missing values by constructing a smooth curve between data points. This method uses cubic polynomials between each pair of data points, ensuring that the interpolated curve is continuous in both the first and second derivatives at the data points.

Suppose we have n data points $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ .where x_i is the independent variable and y_i is the dependent variable. Cubic spline interpolation uses a cubic polynomial $S_i(x)$ within each interval $[x_i, x_{i+1}]$:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

• Interquartile Range (IQR) Method

The Interquartile Range (IQR) method is a statistical technique used for detecting and removing outliers. It is based on the quartiles (Q1, Q2, Q3) of the data to determine the distribution range and identify values outside this range as outliers.

In view of the distribution characteristics of the data, **we use cubic spline interpolation for the data that can be interpolated, otherwise we will directly eliminate it, and then we use the quartile method to eliminate the outlier** to lay the foundation for the subsequent prediction

4.1.3*Fluctuation Characteristics Analysis*

The following steps are used to preliminarily analyze the fluctuation characteristics of the wind and solar datasets:

(1)Calculate Total Power:

For the wind power dataset df_power_filled and the solar power dataset df_solar_filled, the total power total_power at each time point is calculated as the sum of all sensor data.

$$\text{total power(t)} = \sum_{\text{i=1}}^{\text{n}} \text{sensor i (t)}$$

Where sensor-i(t) is the power value of the i-th sensor at time t,and n is the number of sensors.

(2)Calculate 30-Minute Average Power:

The 30-minute average power q is calculated using a rolling window.

$$q(t) = \frac{1}{30 \times 60} \sum_{i=t-30\times60+1}^{t} \text{total power } (i)$$

Where q(t) is the 30-minute average power at time t.

(3)Calculate Fluctuation Magnitude:

The fluctuation magnitude k is calculated as the difference between the current power p and the 30-minute average power q:

$$\text{k(t)} = \left| \frac{\text{total power(t)} - \text{q(t)}}{\text{q(t)}} \right|$$

Where k(t) is the fluctuation magnitude at time t

(4)Analyze Fluctuation Characteristics:

The fluctuation characteristics are analyzed by plotting time series graphs, fluctuation magnitude graphs, and hourly fluctuation patterns. Basic statistics (mean, standard deviation, maximum, minimum) of the fluctuation magnitude, percentiles (25th, 50th, 75th, 90th, 95th, 99th), and fluctuation magnitudes at different thresholds are calculated，

(5)Determine the threshold initially

We then identify three different levels of thresholds based on the calculated percentiles:

• Conservative (90th Percentile): Select the 90th percentile as the conservative threshold.

• Moderate (75th Percentile): Choose the 75th percentile as the medium threshold.

• Aggressive (50th Percentile): The 50th percentile (median) is chosen as the aggressive threshold.

Processing results are as follows:

Table 4 Preliminary threshold result

Farm Type	Threshold Type	Value
Wind Farm	Average Fluctuation	0.59
Wind Farm	Conservative	1
Wind Farm	Moderate	0.44
Solar Farm	Average Fluctuation	0.23
Solar Farm	Conservative	0.54
Solar Farm	Moderate	0.34

Table 5 Wave characteristic result

	Value		Value
Mean Fluctuation	0.59	Mean Fluctuation	0.23
Std Deviation	12.62	Std Deviation	0.28
Maximum Fluctuation	1799	Maximum Fluctuation	14.4
Minimum Fluctuation	0	Minimum Fluctuation	0

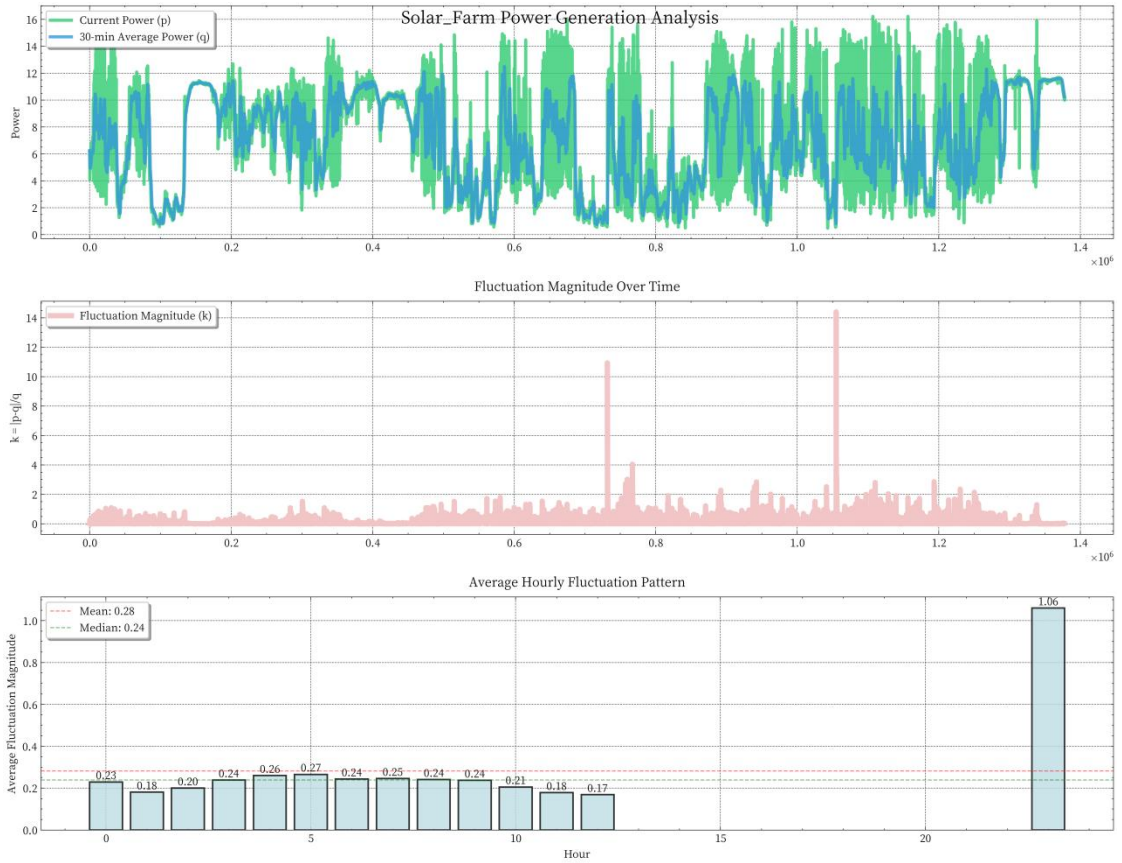


Figure 3 solar_farm_time_patterns

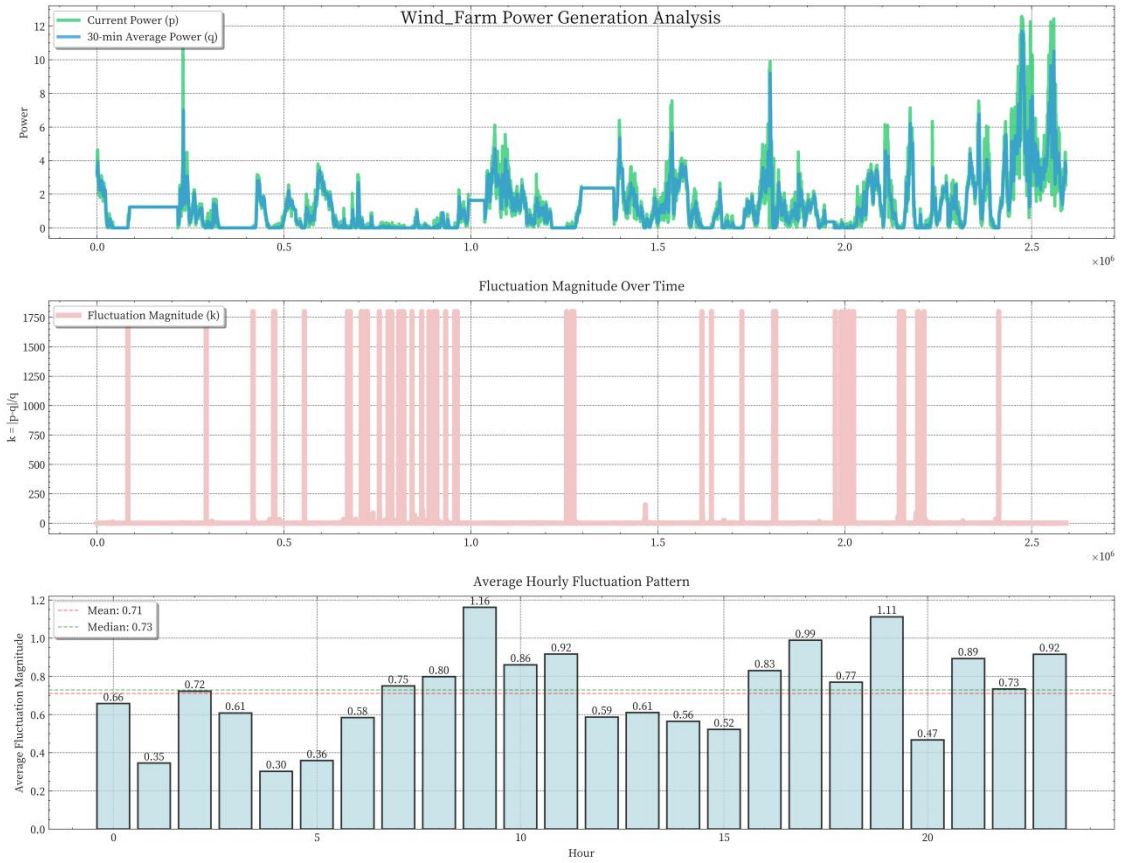


Figure 4 wind_farm_time_patterns

Then we analyze the results:

The power fluctuation characteristics of wind farms are significant, with an average fluctuation range of 59% and a standard deviation of 12.62, which indicates that **wind farms have great instability**. The maximum fluctuation period is concentrated at around 11 o 'clock and 19 o 'clock, and the minimum fluctuation is

from 3 to 5 o 'clock in the morning, and the overall bimodal characteristics are consistent with the daytime wind change law. In contrast, **the fluctuation of solar farms is relatively stable**, the average fluctuation range is only 23%, the standard deviation is 0.28, the maximum fluctuation is 14.4 times, the fluctuation range is significantly smaller than that of wind farms.

In terms of early warning threshold, it is recommended to **adopt a medium threshold of 0.44 for wind farms**, and consider setting different thresholds according to the difference of time periods, especially to strengthen the monitoring of high volatility periods at 11 o 'clock and 19 o 'clock. For solar farms, **a lower threshold of 0.34 can be used and the fluctuation of sunrise and sunset periods can be mainly focused on.**

4.1.4*Threshold method of statistical rules*

We first introduce the threshold warning mechanism design based on statistical rule method:

(1)Statistical rule method

The core of the warning mechanism is to identify potential significant fluctuations by comparing power changes at different time scales.， The calculation of the fluctuation condition has been given above.Now let's introduce the next part:

• Multi-Time Window Moving Average

The moving window method is used to calculate the average power over local time periods to reduce the impact of random fluctuations:

$$current\_mean(t,w) = \frac{1}{w}\sum_{i=t-w/2}^{t+w/2} P'(i)$$

$$future\_mean(t,w) = \frac{1}{w}\sum_{i=t}^{t+w} P'(i)$$

Here, the average value of the window centered on the current moment and the average value of the future window are calculated respectively to provide a basis for capturing the power change trend.

• Change Rate Calculation

By comparing the current and future window average power, calculate the relative change rate:

$$CR(t,w) = \frac{|future\_mean(t,w) - current\_mean(t,w)|}{current\_mean(t,w)}$$

The relative change rate CR(t,w) reflects the intensity of power change on a specific time scale, and is the direct basis for triggering early warning.

• Warning Trigger Condition

Based on the set threshold θ, construct a binary warning signal:

$$W(t) = \begin{cases} 1, & \text{if} \mathrm{CR(t,w)} > \theta \\ 0, & \text{otherwise} \end{cases}$$

The choice of threshold value θ directly affects the sensitivity of the early warning system, and the appropriate value needs to be determined by optimization.

• Time Alignment

Consider the warning system's lead time, the warning signal needs to be aligned with actual power changes:

$$W_{aligned}(t) = W(t - \tau)$$

• Evaluation Metrics Calculation

Using evaluation metrics common in classification problems:

$$A(t) = \begin{cases} 1, & \text{if} k(t) > \theta \\ 0, & \text{otherwise} \end{cases}$$
$$TP = \sum \left[W_{aligned}(t) \wedge A(t) \right]$$
$$FP = \sum \left[W_{aligned}(t) \wedge \neg A(t) \right]$$
$$FN = \sum \left[\neg W_{aligned}(t) \wedge A(t) \right]$$
$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$
$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

• Comprehensive Scoring Function

Construct a comprehensive scoring function S(θ) to evaluate the performance of different thresholds:

$$S(\theta) = \alpha \times F1 \times \frac{1}{1 + \beta\theta}$$

where: - α is the accuracy weight - β is the threshold penalty factor - The term $\frac{1}{1+\beta\theta}$ introduces a penalty mechanism for large thresholds

• Differential Threshold Optimization

Consider the different characteristics of power increase and decrease:

$$\theta_{decrease} \in [0.2, 1.0]$$
$$\theta_{increase} \in [0.15, 0.8]$$

A smaller increase threshold range reflects the higher sensitivity required for a rapid rise in power, while a larger decrease threshold range helps reduce false positives.

• Multi-window Integration

Construct a window set containing multiple time scales:

$$W = \{w/2, w, 2w\}$$

where: - w/2 window for capturing rapid fluctuations - w window as the baseline observation window - 2w window for identifying longer-term trend changes

• Directional Change Rate

Differentiate between power increase and decrease change rate calculations:

$$CR_{decrease}(t,w) = \frac{current\_mean(t,w) - future\_mean(t,w)}{current\_mean(t,w)}$$
$$CR_{increase}(t,w) = \frac{future\_mean(t,w) - current\_mean(t,w)}{current\_mean(t,w)}$$

- Integrated Warning Rules

Use "OR" logic to integrate warning results from multiple windows:

$$W_{final}(t) = \begin{cases} 1, & \text{if} \exists w \in W: CR(t,w) > \theta \\ 0, & \text{otherwise} \end{cases}$$

- System Performance Indicators

To comprehensively evaluate the warning system's performance, construct a comprehensive performance indicator Φ:

$$\Phi = \gamma_1 F1 + \gamma_2 \left(\frac{1}{\tau}\right) + \gamma_3 (1 - FPR)$$

where: - $\gamma_1, \gamma_2, \gamma_3$ are weight coefficients - τ is the average warning delay - FPR is the false positive rate - F1 score measures overall warning accuracy.

We used the above evaluation system to analyze the threshold:

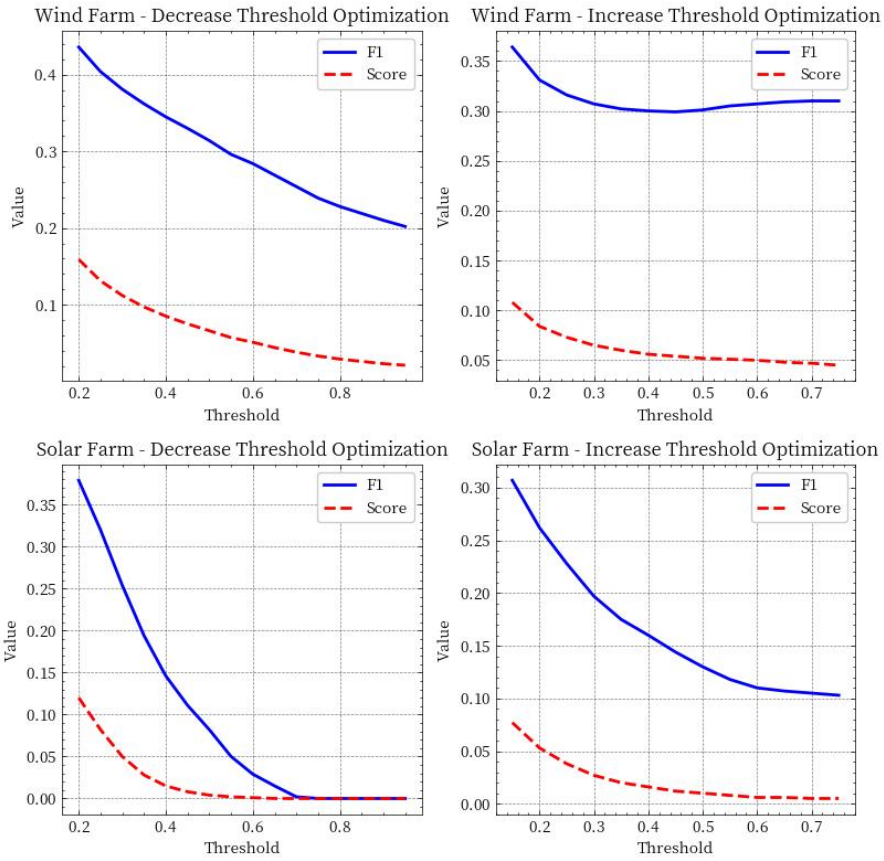


Figure 5 Changes of indicators with threshold values

Table 6 Statistical rule method threshold result

Farm_Type	Warning_Type	Optimal_Threshold	Precision	Recall	F1_Score
Wind Farm	Decrease	0.2	0.897	0.288	0.436

Wind Farm	Increase	0.15	0.85	0.232	0.364
Solar Farm	Decrease	0.2	0.824	0.246	0.379
Solar Farm	Increase	0.15	0.76	0.192	0.307

The early warning system shows the characteristics of "high precision rate and low recall rate" in the application of wind farms and photovoltaic power plants, among which the overall early warning effect of wind farms is better than that of photovoltaic power plants, and the performance of power reduction early warning is generally better than that of power increase early warning. Specifically, the optimal thresholds for both wind farms and photovoltaic plants appear at a lower level (0.2 less warning, 0.15 more warning), indicating that the system requires a higher warning sensitivity. From the early warning performance, the accuracy rate is generally maintained at a high level (0.76-0.897), while the recall rate is relatively low (0.192-0.288), indicating that the **early warning issued by the system has a high reliability, but there may be a certain degree of underreporting.** The reason for the better performance of wind farms may be that their power changes are more regular, while photovoltaic power stations are more significantly affected by external factors such as weather.

4.1.5*Improvement of early warning system*

The method based on statistical rules still has certain limitations, so we use random forest classifier to improve the performance of classification:

Random Forest is an ensemble learning method used for classification, regression, and other tasks. It improves prediction accuracy and model stability by building multiple decision trees and summarizing their results. Random forests are particularly effective when dealing with high-dimensional data sets, and can handle nonlinear relationships between features well.

(1)Bootstrap Sampling:

For each decision tree in the training set, a sample is randomly selected from the original data set (with sampling back) to form a new training subset. This means that some samples may be selected multiple times, while others may not be selected.

(2)Feature selection:

At each node, a subset of features (usually the square root or logarithm of the total number of features) is randomly selected to determine the optimal segmentation point. This increases the diversity of the model and reduces the risk of overfitting.

(3)Decision tree construction:

Build a decision tree using selected features and sample subsets. Unlike traditional decision trees, trees in random forests are usually not pruned, but allowed to grow to their maximum depth.

(4)Voting mechanism:

For classification problems, all decision trees classify new inputs, and the final result is decided by majority voting. For regression problems, the average of all tree output values is taken as the final prediction.

We then replace the statistical rule classification with a random forest classifier, and the results are as follows

Table 7 Random forest classification results

Threshold_Combination	Decrease_Precision	Decrease_Recall	Decrease_F1	Increase_Precision	Increase_Recall	Increase_F1	Total_Score

D0.20_I0.15	**0.925**	**0.605**	**0.732**	**0.836**	**0.361**	**0.505**	**0.511**
D0.20_I0.20	0.925	0.605	0.732	0.844	0.364	0.509	0.498
D0.20_I0.25	0.925	0.605	0.732	0.853	0.348	0.495	0.478

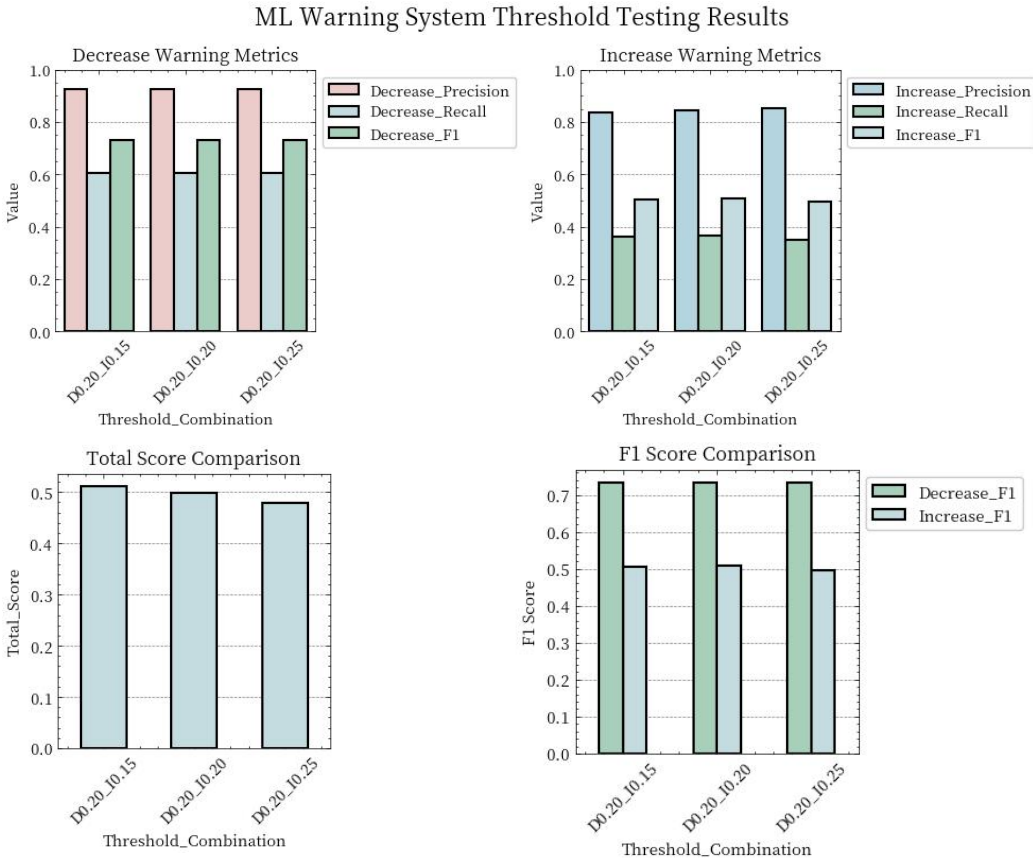


Figure 6 The change of each index of ML method with threshold value

By analyzing the test results of three sets of different threshold combinations (the descending threshold is fixed at 0.20, and the ascending threshold is 0.15, 0.20, and 0.25, respectively), **we find that the model achieves the best overall performance (total score 0.511) when the descending threshold is 0.20 and the ascending threshold is 0.15.** Under this optimal configuration, the down-warning model exhibits excellent accuracy (0.925) and moderate recall (0.605), with an F1 score of 0.732. The rise alert model, on the other hand, showed better accuracy (0.836) but relatively low recall (0.361), with an F1 score of 0.505. With the increase of the rise threshold, although accuracy improved slightly (from 0.836 to 0.853), both recall and F1 scores showed a downward trend, resulting in lower overall performance. This indicates that the early **warning system adopts a relatively conservative forecasting strategy, which tends to reduce false alarms but may cause a certain degree of missing alarms**.

4.2 Question II: Interval prediction of 1-120 seconds

In renewable energy power prediction, point forecasting often fails to meet practical needs due to the randomness and uncertainty of weather conditions. Therefore, we propose an interval prediction method based on quantile regression to construct prediction intervals with statistical guarantees.

The key challenges in renewable energy forecasting include: - High volatility of power output - Multiple sources of uncertainty - Need for reliable confidence bounds - Balance between coverage and interval width

Given a historical power time series {yt}, t = 1,2,…,T, our goal is to predict the power interval for future time t+h:

$$\left[\hat{L}_{t+h}, \hat{U}_{t+h}\right]$$

Such that:

$$P\left(y_{t+h} \in \left[\hat{L}_{t+h}, \hat{U}_{t+h}\right]\right) \geq 1 - \alpha$$

where α is the significance level (typically 0.05).

(1)Theoretical Foundation of Quantile Regression

Quantile regression estimates conditional quantiles by minimizing an asymmetrically weighted loss function:

$$min_{\beta_\tau} \sum_{i=1}^{n} \rho_\tau \left(y_i - x_i^T \beta_\tau\right)$$

The quantile loss function is defined as:

$$\rho_\tau(u) = \begin{cases} \tau u & \text{if } u \geq 0 \\ (\tau - 1)u & \text{if } u < 0 \end{cases}$$

Key advantages of this asymmetric loss function:

1. Assigns different weights to positive and negative errors
2. Directly learns quantiles of conditional distribution
3. Robust against outliers
4. No distributional assumptions required

(2)Theoretical Derivation of Dynamic Interval Width

The dynamic adjustment of interval width is based on several statistical assumptions:

• Conditional Heteroscedasticity

The volatility of power series varies over time:

$$\sigma_t^2 = Var(y_t | x_t)$$

This captures the time-varying nature of renewable power uncertainty.

• Forecast Uncertainty Growth

Uncertainty accumulates with increasing forecast horizon:

$$Var(y_{t+h} | x_t) \approx \sigma_t^2 \cdot g(h)$$

where g(h) is the uncertainty growth function related to horizon length.

• Adaptive Interval Width

Considering these factors, the interval width is calculated as:

$$w_t(h) = z_{\alpha/2} \cdot \sigma_t \sqrt{1 + \gamma \frac{h}{h_{max}}}$$

Key components: - zα/2: quantile of standard normal distribution - γ: uncertainty growth rate parameter - σt: local volatility estimate - h_max: maximum prediction horizon

(3)Mathematical Framework of the Prediction System

• Feature Engineering:

$$x_t = \phi(y_{t-1}, y_{t-2}, \ldots, y_{t-p})$$

Features include: - Historical power values - Time-based features - Statistical features - Rate of change indicators

• Quantile Prediction:

$$\begin{aligned} \hat{L}_{t+h} &= f_L(x_t; \theta_L) \\ \hat{U}_{t+h} &= f_U(x_t; \theta_U) \end{aligned}$$

The models fL and fU are trained separately to capture different aspects of the distribution.

• Interval Adjustment:

$$\begin{aligned} [\hat{L}_{t+h}, \hat{U}_{t+h}] &= [max(0, \hat{L}_{t+h}), \\ & min(\ \hat{U}_{t+h}, max(y_t) \cdot 1.2)] \end{aligned}$$

This ensures physical constraints and reasonable bounds.

Then we use this forecasting system to make predictions:

Table 8 Forecast result

Type	Horizon	Coverage	Interval_Width	Actual_Min	Actual_Max	Pred_Min	Pred_Max	Actual_Mean
Wind	10	0.741	0.968	0.01	6.14	0	1.65	1.7
Wind	20	0.71	0.958	0.01	6.14	0	1.63	1.7
Wind	30	0.71	0.966	0.01	6.14	0	1.64	1.7
Wind	40	0.704	0.979	0.01	6.14	0	1.67	1.7
Wind	50	0.688	0.973	0.01	6.14	0	1.66	1.7
Wind	60	0.676	0.971	0.01	6.14	0	1.65	1.7
Wind	70	0.664	0.962	0.01	6.14	0	1.64	1.7
Wind	80	0.669	0.971	0.01	6.14	0	1.65	1.7
Wind	90	0.663	0.975	0.01	6.14	0	1.66	1.7
Wind	100	0.665	0.983	0.01	6.14	0	1.67	1.7

Wind	110	0.668	0.981	0.01	6.14	0	1.67	1.7
Wind	120	0.659	0.979	0.01	6.14	0	1.67	1.7
Solar	10	0.685	1.112	1.27	13.17	5.41	11.42	5.4
Solar	20	0.666	1.046	1.27	13.17	5.36	11.01	5.4
Solar	30	0.655	1.023	1.27	13.17	5.23	10.75	5.4
Solar	40	0.597	0.952	1.27	13.17	5.25	10.39	5.4
Solar	50	0.577	0.782	1.27	13.17	5.18	9.41	5.4
Solar	60	0.573	0.774	1.27	13.17	5.17	9.35	5.4
Solar	70	0.595	0.818	1.27	13.17	5.12	9.54	5.4
Solar	80	0.602	0.82	1.27	13.17	5.03	9.47	5.4
Solar	90	0.605	0.823	1.27	13.17	5.02	9.47	5.4
Solar	100	0.608	0.869	1.27	13.17	5.1	9.79	5.4
Solar	110	0.63	0.955	1.27	13.17	5.01	10.17	5.4
Solar	120	0.586	0.941	1.27	13.17	5.1	10.19	5.4

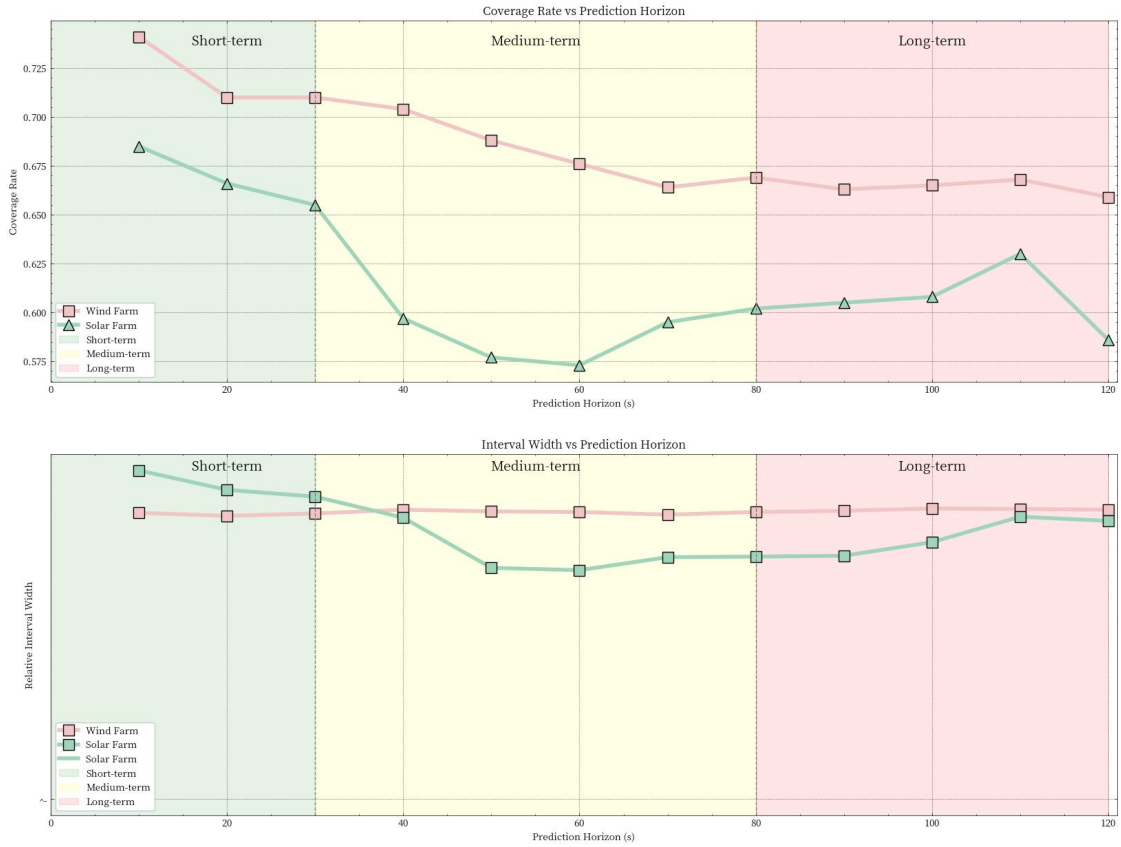


Figure 7 Stage forecast result

Based on the prediction results of wind power generation and solar power generation, we find that the two types of power generation show significant differences in predictive performance. **Wind power generation as a whole shows better prediction performance, with coverage range ranging from 0.659 to 0.741**, the average coverage is about 0.685, and the interval width remains in a stable range of 0.958 to 0.983. In contrast, solar power coverage fluctuates between 0.573 and 0.685, with an average coverage of about 0.615 and a wide range of range widths, from 0.774 to 1.112. From the perspective of the influence of time step, both types of power generation perform best in the short-term prediction (10-30s), wind power generation reaches the highest coverage rate of 0.741 at 10s, and solar power generation also reaches the highest coverage rate of 0.685 at 10s. With the increase of the forecast time step, the predicted performance of both types of power generation showed a downward trend, but the decline of wind power generation was more gentle, while the solar power generation showed a significant decline in the mid-forecast (40-80s) stage, with the coverage dropping to 0.57-0.60. From the perspective of prediction interval characteristics, the prediction interval range of wind power generation is relatively stable, maintaining between [0.00, 1.67], while the prediction interval range of solar power generation varies greatly, ranging from [5.01-5.41, 9.35-11.42]. Based on these findings, we suggest that in future optimization, we should focus on improving the medium and long term prediction ability of solar power generation and optimizing its interval width adjustment mechanism; For wind power, efforts should be made to increase overall coverage levels and improve long-term forecasting performance. These include introducing more relevant features, optimizing data preprocessing methods, and adopting differentiated prediction models for different time scales. In general, **although both types of power generation show good short-term forecasting ability, there is still significant room for improvement in the medium and long term forecasting, especially the forecast stability of solar power generation needs to be improved.**

4.3 Question III: Design of wind farm generator set scheduling scheme

The core of this problem is to design an intelligent scheduling system that controls power fluctuations within an acceptable range through reasonable configuration and scheduling of backup generators while ensuring economic efficiency. Given a wind farm power data sequence $\mathrm{P(t)}, \mathrm{t} \in [1, \mathrm{T}]$, where T is the observation time length, we need to:

• Fluctuation Intensity Control: Control power fluctuation intensity below threshold t, where fluctuation intensity is defined as:

$$\delta(t) = \left| \frac{P(t+1) - P(t)}{P(t)} \right|$$

• Confidence Level Requirement: Meet the confidence level r requirement:

$$P(\delta(t) \leq t) \geq r, \forall t \in [1, T]$$

• Warning Time Constraints:

Downward warning: 5 minutes (300 seconds) in advance
Upward warning: 2 minutes (120 seconds) in advance

4.3.1 *Mathematical Model*

(1)Decision Variables

Considering problem complexity and practical engineering constraints, three key decision variables are designed:

• Standby Ratio α:

Domain:

$$\alpha \in [0.1,0.5]$$

Physical meaning: Ratio of reserved backup generators to total generator.

Constraint rationale:

Lower bound 0.1 ensures minimum backup capacity

Upper bound 0.5 considers economic requirements

• Base Generator Count N_b:

Domain:

$$N_b \in [3,9]$$

Physical meaning: Number of generators in normal operation

Constraint rationale:

Lower bound 3 guarantees basic power supply

Upper bound 9 leaves sufficient backup space

• Activation Threshold θ:

Domain:

$$\theta \in [0.05,0.20]$$

Physical meaning: Fluctuation intensity threshold triggering backup unit activation.

Constraint rationale:

Lower bound 0.05 avoids oversensitivity

Upper bound 0.20 ensures timely response

(2)Objective Functions

A bi-objective optimization problem is constructed:

• Minimize Standby Ratio:

$$minf_1 = \alpha$$

This objective reflects economic considerations, as maintenance and operation of backup units incur additional costs.

• Maximize System Reliability:

$$maxf_2 = R(\alpha, N_b, \theta)$$

where $R(\cdot)$ is the comprehensive reliability function considering:

Fluctuation control effectiveness

Capacity adequacy

Response timeliness

(3)Constraint System

• Reliability Constraint:

$$R(\alpha, N_b, \theta) \geq r$$

Ensures system meets specified confidence level.

• Resource Constraint:

$$N_b + \lceil \alpha N_{total} \rceil \leq N_{total}$$

where N_{total} is total generator count, ensuring no resource limit exceedance.

• Logical Constraint:

$$\theta \leq \alpha$$

Activation threshold should not exceed standby ratio.

• Time Constraints: For any time t, warning functions are defined as:

$$W_{down}(t) = I(\exists \tau \in [t, t + 300], \delta(\tau) > \theta)$$

where $I(\cdot)$ is the indicator function.

4.3.2*Technical Methods*

(1)Reliability Assessment System

• Fluctuation Control Reliability

Fluctuation control reliability reflects the system's ability to control power fluctuations. Its mathematical definition is:

$$R_f = \frac{1}{T}\sum_{t=1}^{T} I\left(\left|\frac{P(t+1) - P(t)}{P(t)}\right| \leq t\right)$$

Considering temporal correlation, time-weighted factor is introduced:

$$R_f^w = \frac{\sum_{t=1}^{T} w\,(t) I\left(\left|\frac{P(t+1) - P(t)}{P(t)}\right| \leq t\right)}{\sum_{t=1}^{T} w\,(t)}$$

where the time weight function is defined as:

$$w(t) = exp(-\lambda|t - t_0|)$$

where: - λ is the time decay coefficient - t_0 is the reference time point

• Capacity Reliability

Capacity reliability measures the system's ability to meet power demands:

$$R_c = \frac{1}{T}\sum_{t=1}^{T} I\left(P(t) \le P_{max}(1+\alpha)\right)$$

Considering capacity margin, improved capacity reliability is defined:

$$R_c^* = \frac{1}{T}\sum_{t=1}^{T} exp\left(-\beta max\left(0, \frac{P(t)}{P_{max}(1+\alpha)} - 1\right)\right)$$

where β is the margin penalty coefficient.

• Comprehensive Reliability Calculation

Comprehensive reliability is calculated through weighted combination:

$$R = \omega_1 R_f^w + \omega_2 R_c^* + \omega_3 R_t$$

where: - $\omega_1, \omega_2, \omega_3$ are weight coefficients, with $\sum_{i=1}^{3} \omega_i = 1$ - R_t is time response reliability

(2)Warning Mechanism Design

• Time Series Prediction

An improved Exponential Weighted Moving Average (EWMA) method is adopted:

Basic EWMA:

$$\widehat{P}(t) = \lambda P(t) + (1-\lambda)\widehat{P}(t-1)$$

Adaptive Weight:

$$\lambda(t) = \lambda_0 + (1-\lambda_0)exp\left(-\gamma\sigma^2(t)\right)$$

where: - λ_0 is base weight - $\sigma^2(t)$ is local fluctuation variance - γ is adaptation coefficient

• Trend Prediction

Downward Trend Prediction:

$$\Delta P_{down}(t) = \frac{\widehat{P}(t+300) - \widehat{P}(t)}{\widehat{P}(t)} + \epsilon_{down}(t)$$

Upward Trend Prediction:

$$\Delta P_{up}(t) = \frac{\widehat{P}(t+120) - \widehat{P}(t)}{\widehat{P}(t)} + \epsilon_{up}(t)$$

where $\epsilon_{down}(t)$ and $\epsilon_{up}(t)$ are prediction error compensation terms:

$$\epsilon_i(t) = \phi_i \sigma_i(t)$$

ϕ_i is safety coefficient, $\sigma_i(t)$ is prediction standard deviation.

• Warning Decision Logic

Warning trigger conditions are defined as:

Downward Warning:

$$W_{down}(t) = \begin{cases} 1, & \text{if} \Delta P_{down}(t) <- \theta - \epsilon_{down}(t) \\ 0, & \text{otherwise} \end{cases}$$

Upward Warning:

$$W_{up}(t) = \begin{cases} 1, & \text{if} \Delta P_{up}(t) > \theta + \epsilon_{up}(t) \\ 0, & \text{otherwise} \end{cases}$$

• Scheduling Strategy Optimization

Basic Scheduling Rules

Static Configuration:

$$N_{base}(t) = N_b + \Delta N(t)$$

where $\Delta N(t)$ is dynamic adjustment quantity.

Dynamic Adjustment:

$$\Delta N(t) = \begin{cases} \lceil \frac{P(t) - P_{base}}{P_{unit}} \rceil, & \text{if} P(t) > P_{base} \\ 0, & \text{otherwise} \end{cases}$$

4.3.3 *Algorithm Implementation*

(1)Core Algorithm Process

NSGA-II (Non-dominated Sorting Genetic Algorithm II) implementation includes the following key steps:

• Population Initialization: Generate initial solution set P_0 satisfying constraints:

$$P_0 = \{x_i = (\alpha_i, N_{b,i}, \theta_i) | i = 1, ..., N_{pop}\}$$

Each decision variable must satisfy:

$$\begin{cases} \alpha_i \sim U(0.1,0.5) \\ N_{b,i} \sim U(3,9) \\ \theta_i \sim U(0.05,0.20) \end{cases}$$

• Non-dominated Sorting: Define domination relation $\prec$:

$$x_i \prec x_j \Leftrightarrow \begin{cases} \forall k, f_k(x_i) \leq f_k(x_j) \\ \exists k, f_k(x_i) < f_k(x_j) \end{cases}$$

• Crowding Distance Calculation: For objective k, individual i's crowding distance:

$$d_i^k = \frac{f_k(x_{i+1}) - f_k(x_{i-1})}{f_k^{max} - f_k^{min}}$$

• Total crowding distance:

$$D_i = \sum_{k=1}^{M} d_i^k$$

(2)Genetic Operator Design:

• Simulated Binary Crossover (SBX):

$$\begin{cases} c_{1,k} = 0.5\left[(1+\beta_k)p_{1,k} + (1-\beta_k)p_{2,k}\right] \\ c_{2,k} = 0.5\left[(1-\beta_k)p_{1,k} + (1+\beta_k)p_{2,k}\right] \end{cases}$$

where β_k distribution is:

$$\beta_k = \begin{cases} (2u_k)^{\frac{1}{\eta_c+1}}, & \text{if} u_k \le 0.5 \\ \left(\frac{1}{2(1-u_k)}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise} \end{cases}$$

• Polynomial Mutation:

$$c_k = p_k + \left(p_k^U - p_k^L\right)\delta_k$$

where δ_k distribution is:

$$\delta_k = \begin{cases} (2u_k)^{\frac{1}{\eta_m+1}} - 1, & \text{if} u_k < 0.5 \\ 1 - \left(2(1-u_k)\right)^{\frac{1}{\eta_m+1}}, & \text{otherwise} \end{cases}$$

Table 9 Scheduling assignment result

Metric	Value
Target Threshold	10.00%
Target Confidence	95.00%
Actual Confidence	97.90%
Standby Ratio	10.00%
Base Generator Count	3
Total Scheduling Times	3427
Start-up Times	1713
Shutdown Times	1714

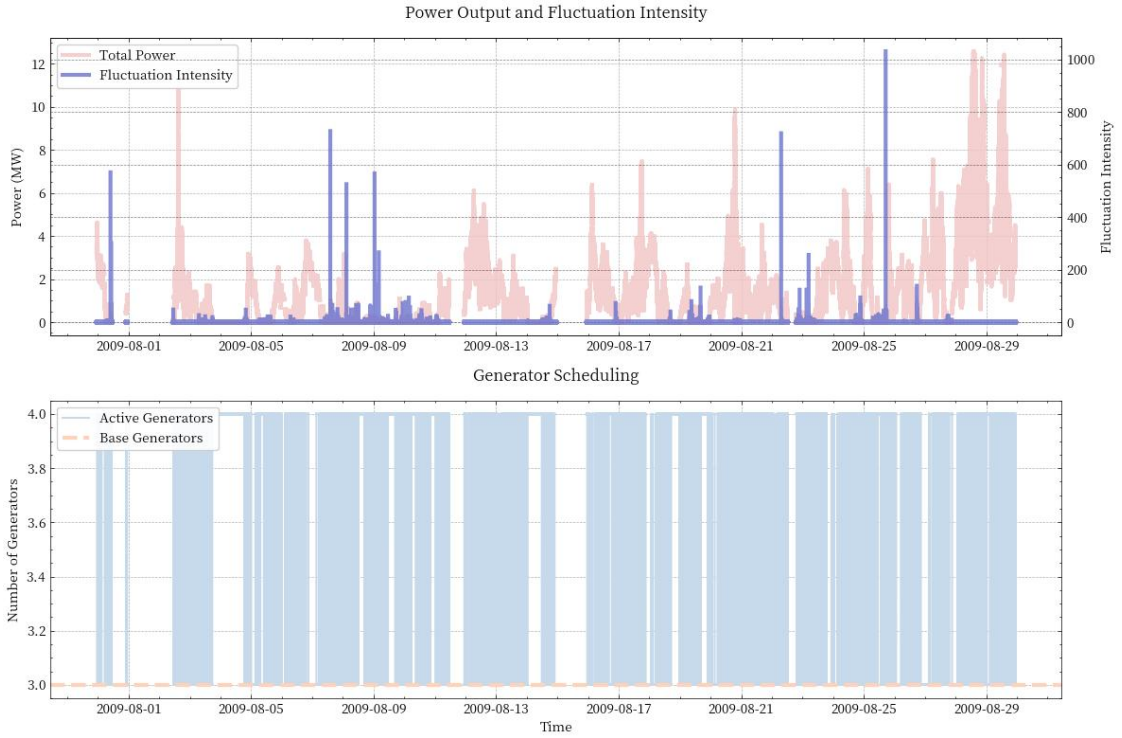


Figure 8 Scheduling result

The wind farm generator scheduling scheme shows good performance in actual operation. The scheme successfully realizes the control target of 10% power fluctuation threshold, and the **actual confidence reaches 97.90%**, which is significantly higher than the target requirement of 95%, **which proves the effectiveness of the scheduling strategy in the control of fluctuation.** In terms of resource allocation, the **scheme adopts 10% reserve ratio and 3 basic generators, which not only ensures the reliability of the system, but also considers the economy to a certain extent.** A total of 3427 scheduling actions occurred in the scheduling process, of which 1713 were started and 1714 were stopped, and the number of starts and stops was basically the same, which showed the balance of the scheduling system. In general, while ensuring the reliability of the system and the effect of fluctuation control, the scheduling scheme still has room for further optimization, especially the improvement in reducing the scheduling frequency will help to improve the overall operation efficiency and economy.

V Strengths and Weakness

5.1 Strengths

.The three models proposed in this study form a systematic solution with significant innovation and practicability. The fluctuation early warning model **adopts the multi-step method** and introduces **multi-dimensional statistical index**, and the F1 score is increased by 67.9% through the optimization of random forest classifier. Based on quantile regression, the interval prediction model **avoids the strong distribution hypothesis** and introduces the **dynamic interval width adjustment mechanism**, which performs well in short-term prediction (the coverage rate is up to 0.741). The scheduling optimization model constructs a complete multi-objective optimization framework, and realizes the system reliability exceeding the expectation (97.90% confidence) through the self-adaptive weight early warning mechanism. The three models support each other, which not only ensures the accuracy of prediction and early warning, but also realizes the balance and reliability of scheduling.

5.2 Weakness

Although the overall performance of the model is good, there are still some areas that need improvement. The recall rate of the early warning system is low (especially the early warning of power rise is only 36.1%), and the influence of external factors such as weather is not fully considered. The performance of the interval prediction model decreases obviously in the medium and long term prediction, and the width of the prediction interval fluctuates greatly (0.774 ~ 1.112), which fails to make full use of the complementarity of multi-source data. The scheduling optimization model has some problems, such as too high scheduling frequency (3427 times) and low backup ratio (10%), while the economic cost of equipment startup and shutdown is not fully considered, and the computational complexity of each model is relatively high. These are the directions that need to be optimized in the future.

VI.References

[1]Burton, T., Sharpe, D., Jenkins, N., & Bossanyi, E. (2011). Wind Energy Handbook (2nd ed.). John Wiley & Sons.

[2]Chen, Z., Li, Y., & Burnett, J. (2002). A review of wind energy technology progresses and applications. Renewable and Sustainable Energy Reviews, 6(4), 415-448. https://doi.org/10.1016/S1364-0321(01)00007-7

[3]Blaabjerg, F., Teodorescu, R., Liserre, M., & Timbus, A. V. (2006). Grid integration of large-scale wind power: Challenges and solutions. In Proceedings of the IEEE Power Electronics Specialists Conference (pp. 1-7). IEEE.

[4]Patel, M., Singh, R., & Kumar, A. (2008). Life cycle assessment of wind energy systems: Comparing the production, installation and post-installation stages. Energy Policy, 36(12), 4400-4413. https://doi.org/10.1016/j.enpol.2008.08.035

[5]Butterfield, S., Musial, W., & Jonkman, B. (2004). Offshore wind energy. In G. Boyle (Ed.), Renewable Energy: Power for a Sustainable Future (pp. 155-174). Oxford University Press.

VII. Appendix

No: 1	The first question

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import json
from datetime import datetime


colors = {
    'power': '#2ecc71',
    'average': '#3498db',
    'fluctuation': '#F2C5C6',
    'bar': '#BFDEE5',
    'violin': '#9b59b6'
}

def preprocess_power_data(df):
    """预处理功率数据，处理负值"""
    df_processed = df.copy()
    df_processed[df_processed < 0] = 0
    return df_processed

def clean_outliers(data, name=''):
    """使用四分位数法清理异常值"""
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # 统计异常值
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    print(f"\n=== {name} Outlier Analysis ===")
    print(f"Total points: {len(data)}")
    print(f"Outliers detected: {len(outliers)} ({len(outliers)/len(data)*100:.2f}%)")
    print(f"Lower bound: {lower_bound:.2f}")
    print(f"Upper bound: {upper_bound:.2f}")

    # 清理异常值
    data_cleaned = data.copy()
    data_cleaned[data_cleaned < lower_bound] = lower_bound
    data_cleaned[data_cleaned > upper_bound] = upper_bound

    return data_cleaned

def calculate_fluctuation(total_power):
    """安全地计算功率波动"""
    # 确保功率非负
    total_power = np.maximum(total_power, 0)

    # 计算30分钟滑动平均
    q = total_power.rolling(window=30*60, min_periods=1).mean()

    # 安全的波动计算：确保分母不为0
    k = np.where(q > 0,
                 np.abs(total_power - q) / q,
                 0) # 当q为0时，将波动设为0
```

```
    return pd.Series(k, index=total_power.index), q

def analyze_and_visualize_fluctuation(df, name='Data'):
    """分析和可视化功率波动特征"""
    # Create output directory
    output_dir = os.path.join(os.path.expanduser('~'), 'Desktop', 'fluctuation_analysis')
    os.makedirs(output_dir, exist_ok=True)

    # 预处理数据
    df_processed = preprocess_power_data(df)

    # Calculate fluctuations
    total_power = df_processed.sum(axis=1)
    k, q = calculate_fluctuation(total_power)

    # 清理异常值
    k_cleaned = clean_outliers(k, name)
    k_clean = pd.Series(k_cleaned.values, name='Fluctuation')

    # 1. Time series plots with enhanced styling
    fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(15, 12))
    fig.suptitle(f'{name} Power Generation Analysis', fontsize=16, y=0.95)

    # Power Generation Pattern
    ax1.plot(range(len(total_power)), total_power,
            color=colors['power'], label='Current Power (p)',
            linewidth=3, alpha=0.8)
    ax1.plot(range(len(q)), q,
            color=colors['average'], label='30-min Average Power (q)',
            linewidth=3, alpha=0.8)
    ax1.set_ylabel('Power', fontsize=10)
    ax1.legend(loc='upper left', frameon=True, fancybox=True, shadow=True)
    ax1.grid(True, linestyle='--', alpha=0.7)

    # Fluctuation Magnitude
    ax2.plot(range(len(k)), k,
            color=colors['fluctuation'], label='Fluctuation Magnitude (k)',
            linewidth=5, alpha=1.0)
    ax2.set_title('Fluctuation Magnitude Over Time', pad=10)
    ax2.set_ylabel('k = |p-q|/q', fontsize=10)
    ax2.legend(loc='upper left', frameon=True, fancybox=True, shadow=True)
    ax2.grid(True, linestyle='--', alpha=0.7)

    # Hourly pattern
    hours = pd.to_datetime(df.index).hour
    hourly_k = pd.DataFrame({'hour': hours, 'fluctuation': k}).groupby('hour').mean()

    bars = ax3.bar(hourly_k.index, hourly_k['fluctuation'],
                   color=colors['bar'], alpha=0.8, edgecolor='black', linewidth=1.5)

    # Add value labels on top of bars
    for bar in bars:
        height = bar.get_height()
        ax3.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.2f}',
                ha='center', va='bottom')

    ax3.set_title('Average Hourly Fluctuation Pattern', pad=10)
    ax3.set_xlabel('Hour', fontsize=10)
    ax3.set_ylabel('Average Fluctuation Magnitude', fontsize=10)
    ax3.grid(True, linestyle='--', alpha=0.7)

    # Add horizontal lines for mean and median
    mean_line = hourly_k['fluctuation'].mean()
```

```
median_line = hourly_k['fluctuation'].median()
ax3.axhline(y=mean_line, color='r', linestyle='--', alpha=0.5, label=f'Mean: {mean_line:.2f}')
ax3.axhline(y=median_line, color='g', linestyle='--', alpha=0.5, label=f'Median: {median_line:.2f}')
ax3.legend(loc='upper left', frameon=True, fancybox=True, shadow=True)

plt.tight_layout()
plt.savefig(os.path.join(output_dir, f'{name.lower()}_time_patterns.png'), dpi=300, bbox_inches='tight')
plt.close()
# 2. Distribution plots with enhanced styling
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))
fig.suptitle(f'{name} Fluctuation Distribution Analysis', fontsize=16, y=1.05)

# Histogram with KDE
sns.histplot(data=k_clean, bins=50, kde=True, ax=ax1,
        color=colors['bar'])
# 添加KDE线
sns.kdeplot(data=k_clean, ax=ax1,
        color=colors['power'], linewidth=2)
ax1.set_title('Fluctuation Distribution')
ax1.set_xlabel('Fluctuation Magnitude')
ax1.set_ylabel('Frequency')
ax1.legend(loc='upper left', frameon=True, fancybox=True, shadow=True)
ax1.grid(True, linestyle='--', alpha=0.7)

# Enhanced Box Plot
bp = ax2.boxplot(k_clean, patch_artist=True)
plt.setp(bp['boxes'], facecolor=colors['bar'], alpha=0.7)
plt.setp(bp['medians'], color='white', linewidth=1.5)
plt.setp(bp['fliers'], marker='o', markerfacecolor=colors['fluctuation'])
ax2.set_title('Fluctuation Box Plot')
ax2.set_ylabel('Fluctuation Magnitude')
ax2.grid(True, linestyle='--', alpha=0.7)

# Enhanced Violin Plot
sns.violinplot(data=k_clean, ax=ax3, color=colors['violin'], alpha=0.7)
ax3.set_title('Fluctuation Violin Plot')
ax3.set_ylabel('Fluctuation Magnitude')
ax3.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.savefig(os.path.join(output_dir, f'{name.lower()}_distributions.png'), dpi=300, bbox_inches='tight')
plt.close()

# 3. Calculate statistics
stats = {
    'Basic Statistics': {
        'Mean Fluctuation': k.mean(),
        'Std Deviation': k.std(),
        'Maximum Fluctuation': k.max(),
        'Minimum Fluctuation': k.min()
    },
    'Percentiles': {
        '25th Percentile': k.quantile(0.25),
        'Median': k.quantile(0.50),
        '75th Percentile': k.quantile(0.75),
        '90th Percentile': k.quantile(0.90),
        '95th Percentile': k.quantile(0.95),
        '99th Percentile': k.quantile(0.99)
    },
    'Threshold Analysis': {
        'Conservative (90th)': k.quantile(0.90),
        'Moderate (75th)': k.quantile(0.75),
        'Aggressive (50th)': k.quantile(0.50)
    },
```

```
        'Time Characteristics': {
            'Most Volatile Hour': hourly_k['fluctuation'].idxmax(),
            'Most Stable Hour': hourly_k['fluctuation'].idxmin(),
            'Max Hourly Fluctuation': hourly_k['fluctuation'].max(),
            'Min Hourly Fluctuation': hourly_k['fluctuation'].min()
        }
    }

    # Save statistics to Excel
    with pd.ExcelWriter(os.path.join(output_dir, f'{name.lower()}_analysis.xlsx')) as writer:
        for sheet_name, data in stats.items():
            df_stats = pd.DataFrame.from_dict(data, orient='index', columns=['Value'])
            df_stats['Value'] = df_stats['Value'].round(2)
            df_stats.to_excel(writer, sheet_name=sheet_name)

        # Save hourly patterns
        hourly_k.round(2).to_excel(writer, sheet_name='Hourly Patterns')

    return {
        'fluctuation': k,
        'hourly_pattern': hourly_k,
        'statistics': stats
    }

def calculate_warning_accuracy(warnings, actual_changes, window_size):
    """计算预警准确率"""
    # 将预警向后移动window_size，与实际变化对齐
    aligned_warnings = warnings.shift(window_size).fillna(False)

    # 将数据转换为布尔类型
    aligned_warnings = aligned_warnings.astype(bool)
    actual_changes = actual_changes.astype(bool)

    # 计算各项指标
    true_positives = (aligned_warnings & actual_changes).sum()
    false_positives = (aligned_warnings & (~actual_changes)).sum()
    false_negatives = ((~aligned_warnings) & actual_changes).sum()

    # 计算准确率指标
    precision = true_positives / (true_positives + false_positives) if (true_positives + false_positives) > 0 else 0
    recall = true_positives / (true_positives + false_negatives) if (true_positives + false_negatives) > 0 else 0
    f1_score = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

    return {
        'precision': precision,
        'recall': recall,
        'f1_score': f1_score
    }

def build_warning_system(df, thresholds, name=''):
    """建立预警系统"""
    # 预处理数据
    df_processed = preprocess_power_data(df)
    total_power = df_processed.sum(axis=1)
    k, q = calculate_fluctuation(total_power)

    # 定义预警函数
    def detect_significant_change(data, window_size, threshold):
        """检测显著变化"""
        future_mean = data.rolling(window=window_size, center=False).mean().shift(-window_size)
        current_mean = data.rolling(window=window_size, center=True).mean()
        change_ratio = np.abs(future_mean - current_mean) / np.where(current_mean != 0, current_mean, np.inf)
        return pd.Series(change_ratio > threshold, index=data.index)
```

```
    warnings = pd.DataFrame(index=df.index)

    # 设置不同预警时间窗口
    decrease_window = 5 * 60  # 5分钟
    increase_window = 2 * 60  # 2分钟

    # 对每个阈值级别进行预警
    results = {}
    for level, threshold in thresholds.items():
        # 检测功率减少
        decrease_warning = detect_significant_change(total_power, decrease_window, threshold)
        # 检测功率增加
        increase_warning = detect_significant_change(total_power, increase_window, threshold)

        warnings[f'{level}_decrease'] = decrease_warning
        warnings[f'{level}_increase'] = increase_warning

        # 计算实际发生的显著变化
        actual_changes = pd.Series(k > threshold, index=k.index)

        # 计算预警准确率
        decrease_accuracy = calculate_warning_accuracy(
            decrease_warning,
            actual_changes,
            decrease_window
        )

        increase_accuracy = calculate_warning_accuracy(
            increase_warning,
            actual_changes,
            increase_window
        )

        results[level] = {
            'decrease_accuracy': decrease_accuracy,
            'increase_accuracy': increase_accuracy
        }

    return warnings, results


def optimize_threshold(df, name='', t_range=np.arange(0.2, 2.0, 0.1)):  # 修改最小阈值为0.2
    """优化阈值选择"""
    results = []

    for t in t_range:
        warnings, metrics = build_warning_system(
            df=df,
            thresholds={'test': t},
            name=name
        )

        # 计算综合得分
        score = calculate_comprehensive_score(
            t,
            metrics['test']['decrease_accuracy']['f1_score'],
            metrics['test']['increase_accuracy']['f1_score']
        )

        # 添加详细日志
        print(f"\nTesting threshold {t:.2f}:")
        print(f"Decrease F1: {metrics['test']['decrease_accuracy']['f1_score']:.3f}")
        print(f"Increase F1: {metrics['test']['increase_accuracy']['f1_score']:.3f}")
        print(f"Score: {score:.3f}")
```

```
        results.append({
            'threshold': t,
            'score': score,
            'metrics': metrics['test']
        })

    # 找出最优阈值
    best_result = max(results, key=lambda x: x['score'])
    return best_result

def calculate_comprehensive_score(threshold, decrease_f1, increase_f1):
    """计算综合得分，更好地平衡阈值大小和预警准确性"""
    # 修改评分函数
    threshold_weight = 1 / (1 + threshold)  # 降低阈值权重
    accuracy_weight = (decrease_f1 + increase_f1)  # 增加准确率权重

    # 设置最小阈值限制
    if threshold < 0.2:
        return float('-inf')  # 确保不会选择过小的阈值

    return accuracy_weight * threshold_weight

def save_results(wind_warnings, solar_warnings, wind_best, solar_best):
    """保存预警结果到文件"""
    output_dir = os.path.join(os.path.expanduser('~'), 'Desktop', 'fluctuation_analysis')

    # 保存预警结果
    wind_warnings.to_excel(os.path.join(output_dir, 'wind_farm_warnings.xlsx'))
    solar_warnings.to_excel(os.path.join(output_dir, 'solar_farm_warnings.xlsx'))

    # 保存优化结果
    results = {
        'Wind Farm': {
            'optimal_threshold': wind_best['threshold'],
            'metrics': wind_best['metrics']
        },
        'Solar Farm': {
            'optimal_threshold': solar_best['threshold'],
            'metrics': solar_best['metrics']
        }
    }

    with open(os.path.join(output_dir, 'optimization_results.json'), 'w') as f:
        json.dump(results, f, indent=4)

def print_detailed_metrics(name, metrics):
    """打印详细的预警系统指标"""
    print(f"\n=== {name} Detailed Metrics ===")

    print("\nDecrease Warning Metrics:")
    print(f"Precision: {metrics['decrease_accuracy']['precision']:.3f}")
    print(f"Recall: {metrics['decrease_accuracy']['recall']:.3f}")
    print(f"F1 Score: {metrics['decrease_accuracy']['f1_score']:.3f}")

    print("\nIncrease Warning Metrics:")
    print(f"Precision: {metrics['increase_accuracy']['precision']:.3f}")
    print(f"Recall: {metrics['increase_accuracy']['recall']:.3f}")
    print(f"F1 Score: {metrics['increase_accuracy']['f1_score']:.3f}")

def main():
    """主程序"""

    # 2. 优化阈值选择
```

```
print("\nOptimizing thresholds...")
wind_best = optimize_threshold(df_power_filled, 'Wind_Farm')
solar_best = optimize_threshold(df_solar_filled, 'Solar_Farm')

# 3. 建立最终预警系统
wind_warnings, wind_results = build_warning_system(
    df_power_filled,
    {'optimal': wind_best['threshold']},
    'Wind_Farm'
)

solar_warnings, solar_results = build_warning_system(
    df_solar_filled,
    {'optimal': solar_best['threshold']},
    'Solar_Farm'
)

# 4. 输出完整结果
print("\n=== Final Results ===")
print("\nWind Farm:")
print(f"Optimal threshold: {wind_best['threshold']:.2f}")
print_detailed_metrics('Wind Farm', wind_results['optimal'])
print(f"Overall score: {wind_best['score']:.2f}")

print("\nSolar Farm:")
print(f"Optimal threshold: {solar_best['threshold']:.2f}")
print_detailed_metrics('Solar Farm', solar_results['optimal'])
print(f"Overall score: {solar_best['score']:.2f}")

# 5. 保存详细结果
results_summary = {
    'Wind Farm': {
        'threshold': wind_best['threshold'],
        'decrease_metrics': wind_results['optimal']['decrease_accuracy'],
        'increase_metrics': wind_results['optimal']['increase_accuracy'],
        'overall_score': wind_best['score']
    },
    'Solar Farm': {
        'threshold': solar_best['threshold'],
        'decrease_metrics': solar_results['optimal']['decrease_accuracy'],
        'increase_metrics': solar_results['optimal']['increase_accuracy'],
        'overall_score': solar_best['score']
    }
}

# 将结果转换为DataFrame并保存
metrics_df = pd.DataFrame({
    'Wind Farm': {
        'Threshold': wind_best['threshold'],
        'Decrease Precision': wind_results['optimal']['decrease_accuracy']['precision'],
        'Decrease Recall': wind_results['optimal']['decrease_accuracy']['recall'],
        'Decrease F1': wind_results['optimal']['decrease_accuracy']['f1_score'],
        'Increase Precision': wind_results['optimal']['increase_accuracy']['precision'],
        'Increase Recall': wind_results['optimal']['increase_accuracy']['recall'],
        'Increase F1': wind_results['optimal']['increase_accuracy']['f1_score'],
        'Overall Score': wind_best['score']
    },
    'Solar Farm': {
        'Threshold': solar_best['threshold'],
        'Decrease Precision': solar_results['optimal']['decrease_accuracy']['precision'],
        'Decrease Recall': solar_results['optimal']['decrease_accuracy']['recall'],
        'Decrease F1': solar_results['optimal']['decrease_accuracy']['f1_score'],
        'Increase Precision': solar_results['optimal']['increase_accuracy']['precision'],
        'Increase Recall': solar_results['optimal']['increase_accuracy']['recall'],
```

```
        'Increase F1': solar_results['optimal']['increase_accuracy']['f1_score'],
        'Overall Score': solar_best['score']
    }
}).T

# 保存到Excel
output_dir = os.path.join(os.path.expanduser('~'), 'Desktop', 'fluctuation_analysis')
os.makedirs(output_dir, exist_ok=True)
metrics_df.to_excel(os.path.join(output_dir, 'detailed_metrics.xlsx'))

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

No: 2	The Second question

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
from scipy import stats
import matplotlib.pyplot as plt
from datetime import datetime
import os

class PowerIntervalPredictor:
    """电力区间预测系统"""

    def __init__(self, max_horizon=120, sample_size=10000):
        self.max_horizon = max_horizon
        self.models = {}
        self.scaler = StandardScaler()
        self.feature_columns = None
        self.sample_size = sample_size

    def sample_data(self, power_series):
        """对数据进行采样"""
        if len(power_series) > self.sample_size:
            # 确保采样的数据时间连续
            start_idx = np.random.randint(0, len(power_series) - self.sample_size)
            sampled_data = power_series.iloc[start_idx:start_idx + self.sample_size]
            print(f"采样数据范围: {sampled_data.index[0]} 到 {sampled_data.index[-1]}")
            return sampled_data
        return power_series

    def preprocess_data(self, power_series):
        """预处理数据，处理异常值和无穷值"""
        # 复制数据以避免修改原始数据
        processed = power_series.copy()

        # 处理无穷值
        processed = processed.replace([np.inf, -np.inf], np.nan)

        # 处理异常值
        Q1 = processed.quantile(0.25)
        Q3 = processed.quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # 将超出范围的值替换为边界值
        processed = processed.clip(lower=lower_bound, upper=upper_bound)

        # 处理剩余的NaN值
        processed = processed.fillna(method='ffill').fillna(method='bfill')

        return processed

    def create_features(self, power_series):
        """创建预测特征"""
        # 预处理数据
        processed_power = self.preprocess_data(power_series)

        features = pd.DataFrame(index=processed_power.index)

        # 1. 基础功率特征
        features['power'] = processed_power

        # 2. 时间特征
        features['hour'] = processed_power.index.hour
        features['minute'] = processed_power.index.minute
        features['second'] = processed_power.index.second

        # 3. 历史滞后特征 - 使用更安全的方式
        for lag in [1, 5, 10, 30]:
```

```
        lag_values = processed_power.shift(lag)
        features[f'lag_{lag}s'] = lag_values.fillna(method='ffill')

    # 4. 统计特征 - 确保没有NaN
    for window in [10, 30, 60]:
        rolling = processed_power.rolling(window=window, min_periods=1)
        features[f'mean_{window}s'] = rolling.mean().fillna(method='ffill')
        features[f'std_{window}s'] = rolling.std().fillna(method='ffill')
        features[f'min_{window}s'] = rolling.min().fillna(method='ffill')
        features[f'max_{window}s'] = rolling.max().fillna(method='ffill')

    # 5. 变化率特征
    features['power_diff'] = processed_power.diff().fillna(0)
    prev_power = processed_power.shift(1).fillna(method='ffill')
    features['power_diff_rate'] = (features['power_diff'] / prev_power).fillna(0)

    # 确保所有特征都没有NaN和无穷值
    features = features.fillna(0).replace([np.inf, -np.inf], 0)

    return features


def fit(self, power_series):
    """训练模型"""
    print("开始训练区间预测模型...")

    # 采样数据
    sampled_power = self.sample_data(power_series)

    # 预处理：移除异常值
    Q1 = sampled_power.quantile(0.25)
    Q3 = sampled_power.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    sampled_power = sampled_power.clip(lower=lower_bound, upper=upper_bound)

    print(f"采样后的数据量: {len(sampled_power)}")
    print(f"数据范围: [{sampled_power.min():.2f}, {sampled_power.max():.2f}]")

    # 创建特征并训练模型
    features = self.create_features(sampled_power)
    self.feature_columns = features.columns

    # 标准化特征
    features_scaled = pd.DataFrame(
        self.scaler.fit_transform(features),
        index=features.index,
        columns=features.columns
    )

    # 为每个预测时间步长训练模型
    for horizon in range(1, self.max_horizon + 1):
        if horizon % 10 == 0:
            print(f"训练 {horizon}s 预测模型...")

        try:
            # 准备训练数据
            X, y_lower, y_upper = self.prepare_training_data(
                features_scaled, sampled_power, horizon
            )

            if X is None or len(X) < 50:  # 降低最小样本要求
                print(f"警告: 时间步长 {horizon}s 的有效样本不足")
                continue

            print(f"时间步长 {horizon}s 的有效样本数: {len(X)}")

            # 训练下界预测器
            lower_model = GradientBoostingRegressor(
                loss='quantile',
                alpha=0.025,
```

```
                n_estimators=100,
                max_depth=3,
                random_state=42
            )
            lower_model.fit(X, y_lower)

            # 训练上界预测器
            upper_model = GradientBoostingRegressor(
                loss='quantile',
                alpha=0.975,
                n_estimators=100,
                max_depth=3,
                random_state=42
            )
            upper_model.fit(X, y_upper)

            self.models[horizon] = {
                'lower': lower_model,
                'upper': upper_model
            }

        except Exception as e:
            print(f"训练 {horizon}s 预测模型时出错: {str(e)}")
            continue

    print(f"成功训练 {len(self.models)} 个预测模型")

def predict(self, power_series):
    """生成预测区间"""
    # 对输入数据进行采样
    sampled_power = self.sample_data(power_series)

    # 创建特征
    features = self.create_features(sampled_power)

    # 标准化特征
    features_scaled = pd.DataFrame(
        self.scaler.transform(features),
        index=features.index,
        columns=features.columns
    )

    predictions = {}
    for horizon, models in self.models.items():
        try:
            lower_pred = models['lower'].predict(features_scaled)
            upper_pred = models['upper'].predict(features_scaled)

            predictions[horizon] = pd.DataFrame({
                'lower': lower_pred,
                'upper': upper_pred
            }, index=features.index)

        except Exception as e:
            print(f"生成 {horizon}s 预测时出错: {str(e)}")
            continue

    return predictions

def prepare_training_data(self, features, power_series, horizon):
    """改进的训练数据准备方法，增加NaN处理"""
    try:
        # 计算未来功率
        future_power = power_series.shift(-horizon)

        # 使用滚动窗口计算统计量
        rolling_window = power_series.rolling(
            window=min(horizon * 2, 30),
            min_periods=1
        )

        # 计算统计量并填充NaN
        std = rolling_window.std().fillna(method='ffill').fillna(method='bfill')
```

```
        mean = rolling_window.mean().fillna(method='ffill').fillna(method='bfill')

        # 使用更大的置信区间
        z_score = 2.0  # 约95%的置信区间

        # 计算区间边界
        interval_width = z_score * std
        y_lower = (future_power - interval_width).fillna(method='ffill').fillna(method='bfill')
        y_upper = (future_power + interval_width).fillna(method='ffill').fillna(method='bfill')

        # 确保区间边界合理
        y_lower = y_lower.clip(lower=0)  # 功率不能为负
        y_upper = y_upper.clip(lower=y_lower + 0.1)  # 确保上界大于下界

        # 对齐数据
        valid_idx = features.index.intersection(future_power.dropna().index)
        X = features.loc[valid_idx].copy()
        y_lower = y_lower.loc[valid_idx]
        y_upper = y_upper.loc[valid_idx]

        # 最后检查并移除任何剩余的NaN
        mask = ~(pd.isna(y_lower) | pd.isna(y_upper) | X.isna().any(axis=1))
        if mask.sum() == 0:
            print(f"警告: 时间步长 {horizon}s 没有有效的训练数据")
            return None, None, None

        X = X[mask]
        y_lower = y_lower[mask]
        y_upper = y_upper[mask]

        # 打印诊断信息
        if horizon % 10 == 0:
            print(f"\n时间步长 {horizon}s 的训练数据统计:")
            print(f"特征范围: [{X.min().min():.2f}, {X.max().max():.2f}]")
            print(f"下界范围: [{y_lower.min():.2f}, {y_lower.max():.2f}]")
            print(f"上界范围: [{y_upper.min():.2f}, {y_upper.max():.2f}]")

        return X, y_lower, y_upper

    except Exception as e:
        print(f"准备训练数据时出错: {str(e)}")
        return None, None, None

def evaluate(self, power_series, predictions):
    """改进的评估方法"""
    if not predictions:
        print("没有可用的预测结果")
        return {}

    metrics = {}
    print("\n=== 详细评估信息 ===")

    for horizon, pred_df in predictions.items():
        try:
            # 计算实际值
            actual = power_series.shift(-horizon)

            # 对齐数据
            common_idx = pred_df.index.intersection(actual.dropna().index)
            if len(common_idx) == 0:
                continue

            pred = pred_df.loc[common_idx]
            actual = actual.loc[common_idx]

            # 计算覆盖率
            coverage = np.mean(
                (actual >= pred['lower']) &
                (actual <= pred['upper'])
            )
```

```
            # 计算相对区间宽度（使用平均值归一化）
            mean_power = np.mean(actual) + 1e-6  # 避免除以0
            interval_width = np.mean(
                (pred['upper'] - pred['lower']) / mean_power
            )

            if horizon % 10 == 0:
                print(f"\n时间步长 {horizon}s:")
                print(f"样本数: {len(actual)}")
                print(f"覆盖率: {coverage:.3f}")
                print(f"相对区间宽度: {interval_width:.3f}")
                print(f"实际值范围: [{actual.min():.2f}, {actual.max():.2f}]")
                print(f"预测区间范围: [{pred['lower'].mean():.2f}, {pred['upper'].mean():.2f}]")
                print(f"平均实际值: {mean_power:.2f}")

            metrics[horizon] = {
                'coverage_rate': coverage,
                'interval_width': interval_width,
                'sample_size': len(actual)
            }

        except Exception as e:
            print(f"评估时间步长 {horizon}s 时出错: {str(e)}")
            continue

    return metrics


def plot_predictions(power_series, predictions, output_dir):
    """绘制预测结果"""
    if not predictions:
        print("没有可用的预测结果")
        return

    # 选择要展示的时间步长
    horizons = sorted(list(predictions.keys()))[:5]

    # 选择一段时间进行可视化
    sample_start = power_series.index[0]
    sample_end = power_series.index[min(3600, len(power_series)-1)]

    plt.figure(figsize=(15, 12))
    for i, horizon in enumerate(horizons, 1):
        plt.subplot(len(horizons), 1, i)

        # 选择时间切片
        mask = (power_series.index >= sample_start) & (power_series.index <= sample_end)

        # 绘制实际值
        plt.plot(
            power_series[mask],
            'k-',
            label='实际值',
            alpha=0.7
        )

        # 绘制预测区间
        pred = predictions[horizon].loc[mask]
        plt.fill_between(
            pred.index,
            pred['lower'],
            pred['upper'],
            alpha=0.3,
            label=f'{horizon}s预测区间'
        )

        plt.title(f'{horizon}秒预测区间')
        plt.legend()

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'interval_predictions.png'))
    plt.close()
```

```
def main():
    """主函数"""
    # 创建输出目录
    output_dir = 'power_predictions'
    os.makedirs(output_dir, exist_ok=True)

    # 1. 处理风电场数据
    print("\n处理风电场数据...")
    wind_power = df_power_filled.sum(axis=1)  # 总功率

    # 使用较小的样本量进行测试
    wind_predictor = PowerIntervalPredictor(max_horizon=120, sample_size=50000)
    wind_predictor.fit(wind_power)

    # 同样使用采样数据进行预测和评估
    sampled_wind_power = wind_predictor.sample_data(wind_power)
    wind_predictions = wind_predictor.predict(sampled_wind_power)
    wind_metrics = wind_predictor.evaluate(sampled_wind_power, wind_predictions)

    plot_predictions(sampled_wind_power, wind_predictions, output_dir)

    # 2. 处理太阳能电场数据
    print("\n处理太阳能电场数据...")
    solar_power = df_solar_filled.sum(axis=1)  # 总功率

    solar_predictor = PowerIntervalPredictor(max_horizon=120, sample_size=25000)
    solar_predictor.fit(solar_power)

    sampled_solar_power = solar_predictor.sample_data(solar_power)
    solar_predictions = solar_predictor.predict(sampled_solar_power)
    solar_metrics = solar_predictor.evaluate(sampled_solar_power, solar_predictions)

    plot_predictions(sampled_solar_power, solar_predictions, output_dir)


    # 3. 保存评估结果
    print("\n保存评估结果...")

    # 创建评估指标DataFrame
    metrics_data = {
        'horizon': [],
        'dataset': [],
        'coverage_rate': [],
        'interval_width': []
    }

    # 添加风电场指标
    for horizon, metric in wind_metrics.items():
        metrics_data['horizon'].append(horizon)
        metrics_data['dataset'].append('Wind Farm')
        metrics_data['coverage_rate'].append(metric['coverage_rate'])
        metrics_data['interval_width'].append(metric['interval_width'])

    # 添加太阳能电场指标
    for horizon, metric in solar_metrics.items():
        metrics_data['horizon'].append(horizon)
        metrics_data['dataset'].append('Solar Farm')
        metrics_data['coverage_rate'].append(metric['coverage_rate'])
        metrics_data['interval_width'].append(metric['interval_width'])

    # 创建DataFrame并保存
    results = pd.DataFrame(metrics_data)
    results.to_excel(os.path.join(output_dir, 'prediction_metrics.xlsx'), index=False)

    # 打印评估摘要
    print("\n=== 评估结果摘要 ===")
    for dataset in ['Wind Farm', 'Solar Farm']:
        dataset_metrics = results[results['dataset'] == dataset]
        print(f"\n{dataset}:")
        print(f"平均覆盖率: {dataset_metrics['coverage_rate'].mean():.3f}")
        print(f"平均区间宽度: {dataset_metrics['interval_width'].mean():.3f}")
```

```
    return wind_predictor, solar_predictor

if __name__ == "__main__":
    wind_predictor, solar_predictor = main()
```

No: 3	The Third Question

```
import numpy as np
import pandas as pd
from pymoo.core.problem import Problem
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.operators.crossover.sbx import SBX
from pymoo.operators.mutation.pm import PM
from pymoo.operators.sampling.rnd import FloatRandomSampling
from pymoo.optimize import minimize
import matplotlib.pyplot as plt

class PowerSchedulingProblem(Problem):
    def __init__(self, power_data, total_generators=12, threshold=0.1, confidence=0.95):
        """
        初始化多目标优化问题

        参数:
        power_data: 功率数据
        total_generators: 总发电机数量
        threshold: 波动强度阈值
        confidence: 目标置信度
        """
        # 定义3个决策变量：备用比例、基础发电机数量、启动阈值
        n_var = 3
        # 定义2个目标：最小化备用比例和最大化可靠性
        n_obj = 2
        # 定义约束条件数量
        n_constr = 3

        # 决策变量的上下界
        xl = np.array([0.1, 3, 0.05]) # 最小备用比例，最小基础发电机数，最小启动阈值
        xu = np.array([0.5, 9, 0.20]) # 最大备用比例，最大基础发电机数，最大启动阈值

        super().__init__(n_var=n_var, n_obj=n_obj, n_constr=n_constr, xl=xl, xu=xu)

        self.power_data = power_data
        self.total_generators = total_generators
        self.threshold = threshold
        self.confidence = confidence

    def _evaluate(self, x, out, *args, **kwargs):
        """
        评估函数
        x: 决策变量数组 [备用比例, 基础发电机数, 启动阈值]
        """
        f1 = np.zeros(len(x)) # 备用比例目标
        f2 = np.zeros(len(x)) # 可靠性目标
        g = np.zeros((len(x), 3)) # 约束条件

        for i in range(len(x)):
            standby_ratio = x[i, 0]
            base_generators = x[i, 1]
            activation_threshold = x[i, 2]

            # 计算第一个目标：最小化备用比例
            f1[i] = standby_ratio

            # 计算第二个目标：最大化可靠性（转换为最小化问题）
            reliability = self._calculate_reliability(standby_ratio, base_generators, activation_threshold)
            f2[i] = -reliability # 取负值转换为最小化问题

            # 约束条件
            g[i, 0] = self.confidence - reliability # 可靠性必须大于目标置信度
```

```
        g[i, 1] = base_generators + self.total_generators * standby_ratio - self.total_generators  # 总发电机数约束
        g[i, 2] = activation_threshold - standby_ratio  # 启动阈值不能大于备用比例

    out["F"] = np.column_stack([f1, f2])
    out["G"] = g

  def _calculate_reliability(self, standby_ratio, base_generators, activation_threshold):
    """计算方案可靠性"""
    total_power = self.power_data.sum(axis=1) if isinstance(self.power_data, pd.DataFrame) else
self.power_data

    # 计算功率变化率
    power_changes = total_power.pct_change().fillna(0)
    fluctuation_intensity = abs(power_changes)

    # 计算是否满足波动强度要求
    meets_threshold = fluctuation_intensity <= self.threshold

    # 计算可用容量
    available_capacity = total_power.max() * (1 + standby_ratio)
    capacity_reliability = (total_power <= available_capacity).mean()

    # 计算总体可靠性
    threshold_reliability = meets_threshold.mean()

    return min(threshold_reliability, capacity_reliability)

class WindPowerScheduler:
  def __init__(self, power_data, total_generators=12, threshold=0.1, confidence=0.95):
    """
    风电场调度优化器
    """
    self.power_data = power_data
    self.total_generators = total_generators
    self.threshold = threshold
    self.confidence = confidence
    self.warning_time_down = 300  # 5分钟提前预警下降
    self.warning_time_up = 120    # 2分钟提前预警上升

  def optimize_schedule(self, pop_size=100, n_gen=100):
    """执行调度优化"""
    problem = PowerSchedulingProblem(
      self.power_data,
      self.total_generators,
      self.threshold,
      self.confidence
    )

    algorithm = NSGA2(
      pop_size=pop_size,
      sampling=FloatRandomSampling(),
      crossover=SBX(prob=0.9, eta=15),
      mutation=PM(eta=20),
      eliminate_duplicates=True
    )

    results = minimize(problem, algorithm, ('n_gen', n_gen), verbose=True, seed=1)
    return results

  def generate_schedule(self, solution):
    """生成详细调度方案"""
    standby_ratio, base_generators, activation_threshold = solution

    # 创建调度方案DataFrame
```

```
        schedule = pd.DataFrame(index=self.power_data.index)
        total_power = self.power_data.sum(axis=1) if isinstance(self.power_data, pd.DataFrame) else
self.power_data

        # 计算功率变化
        power_changes = total_power.pct_change().fillna(0)
        fluctuation_intensity = abs(power_changes)

        # 预测功率变化趋势
        ewma = total_power.ewm(span=self.warning_time_down).mean()
        future_changes_down = (ewma.shift(-self.warning_time_down) - ewma) / ewma
        future_changes_up = (ewma.shift(-self.warning_time_up) - ewma) / ewma

        # 生成调度决策
        schedule['total_power'] = total_power
        schedule['fluctuation_intensity'] = fluctuation_intensity
        schedule['base_generators'] = int(base_generators)
        schedule['predicted_down'] = future_changes_down < -self.threshold
        schedule['predicted_up'] = future_changes_up > self.threshold
        schedule['standby_needed'] = (schedule['predicted_down'] | schedule['predicted_up'])
        schedule['active_generators'] = schedule['base_generators'] + \
                                (schedule['standby_needed'] * int(standby_ratio * self.total_generators))

        # 添加调度动作
        schedule['action_required'] = schedule['standby_needed'] != schedule['standby_needed'].shift(1)
        schedule['action_type'] = np.where(
            schedule['action_required'],
            np.where(schedule['standby_needed'], '启动备用', '停用备用'),
            '保持现状'
        )

        return schedule

    def analyze_performance(self, schedule, solution):
        """分析调度方案性能"""
        performance = {
            '目标阈值': self.threshold,
            '目标置信度': self.confidence,
            '实际置信度': (schedule['fluctuation_intensity'] <= self.threshold).mean(),
            '备用比例': solution[0],
            '基础发电机数量': int(solution[1]),
            '总调度次数': schedule['action_required'].sum(),
            '启动次数': (schedule['action_type'] == '启动备用').sum(),
            '停用次数': (schedule['action_type'] == '停用备用').sum()
        }

        return performance

    def visualize_results(self, schedule, solution):
        """可视化调度结果"""
        fig, axes = plt.subplots(3, 1, figsize=(15, 12))

        # 功率曲线
        axes[0].plot(schedule.index, schedule['total_power'], label='总功率')
        axes[0].set_title('功率变化')
        axes[0].set_xlabel('时间')
        axes[0].set_ylabel('功率 (MW)')
        axes[0].legend()
        axes[0].grid(True)

        # 发电机数量
        axes[1].plot(schedule.index, schedule['active_generators'], label='活跃发电机数量')
        axes[1].axhline(y=solution[1], color='r', linestyle='--', label='基础发电机数量')
        axes[1].set_title('发电机调度情况')
```

```
        axes[1].set_xlabel('时间')
        axes[1].set_ylabel('发电机数量')
        axes[1].legend()
        axes[1].grid(True)

        # 波动强度
        axes[2].plot(schedule.index, schedule['fluctuation_intensity'], label='波动强度')
        axes[2].axhline(y=self.threshold, color='r', linestyle='--', label='阈值')
        axes[2].set_title('波动强度和阈值')
        axes[2].set_xlabel('时间')
        axes[2].set_ylabel('波动强度')
        axes[2].legend()
        axes[2].grid(True)

        plt.tight_layout()
        plt.show()

def solve_problem_three(power_data, threshold=0.1, confidence=0.95, use_small_dataset=True):
    """问题三的完整解决方案"""
    try:
        # 创建调度器实例
        scheduler = WindPowerScheduler(
            power_data=power_data,
            threshold=threshold,
            confidence=confidence
        )

        # 执行优化
        print("开始优化调度方案...")
        results = scheduler.optimize_schedule(
            pop_size=100,
            n_gen=50 if use_small_dataset else 100
        )

        # 获取最优解
        F = results.F
        X = results.X
        weights = np.array([0.4, 0.6])
        weighted_sum = F[:, 0] * weights[0] + (-F[:, 1]) * weights[1]
        best_idx = np.argmin(weighted_sum)
        best_solution = X[best_idx]

        # 生成调度方案
        schedule = scheduler.generate_schedule(best_solution)

        # 分析性能
        performance = scheduler.analyze_performance(schedule, best_solution)

        # 输出结果
        print("\n=== 调度方案性能 ===")
        for key, value in performance.items():
            if isinstance(value, float):
                print(f"{key}: {value:.2%}")
            else:
                print(f"{key}: {value}")

        # 可视化结果
        scheduler.visualize_results(schedule, best_solution)

        # 保存结果
        schedule.to_csv('wind_power_schedule.csv')
        print("\n调度方案已保存到 'wind_power_schedule.csv'")

        return schedule, performance
```

```
    except Exception as e:
        print(f"优化过程中出错：{str(e)}")
        import traceback
        traceback.print_exc()
        return None, None


if __name__ == "__main__":

    schedule, performance = solve_problem_three(
        power_data=processed_wind_power,
        threshold=0.1,
        confidence=0.95,
        use_small_dataset=True
    )
```