E911 Report of Project 1

Author: Zhiling Li 12112748, Jinda Dong 12111829

Task 1

Problem Understanding

The aim is to develop a short-term wind power forecast model to predict wind speeds 10 minutes into the future. This forecasting is essential for efficient energy management and active control in wind farms. The prediction model in consideration is an Autoregressive (AR) model, which predicts future wind speed based on past values.

Theoretical Analysis

The AR model used for wind speed prediction is given by:

$$y(k) = a_1 y(k-1) + a_2 y(k-2) + ... + a_n y(k-n) + v(k) \tag{1}$$

where:

- $y(k)$ is the wind speed at time k,
- a_i are the coefficients of the model,
- n is the number of lag terms,
- $v(k)$ is the noise term.

The task is to determine the optimal number of regression terms n and the corresponding coefficient values $\theta^{(n)} = [a_1, a_2, ..., a_n]^T$.

Algorithm Analysis

To identify the AR parameters, the Ordinary Least Squares method is employed, which minimizes the sum of the squares of the differences between the predicted values and the observed data. The model is iteratively calculated for n ranging from 1 to 20, and the prediction error $err(n)$ is computed.

For the OLS method, we can get the matrix expression from the AR model:

$$\begin{bmatrix} y(k) \\ y(k-1) \\ \vdots \\ y(21) \end{bmatrix} = \begin{bmatrix} y(k-1) & y(k-2) & \cdots & y(k-n) \\ y(k-2) & y(k-3) & \cdots & y(k-1-n) \\ \vdots & \vdots & \ddots & \vdots \\ y(21-1) & y(21-2) & \cdots & y(21-n) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \tag{2}$$

for the above equation, we posit that we uniformly use data `21 ~ 5000`, in order to make the total prediction error meaningful, and we define

$$\hat{y} = \begin{bmatrix} y(k) \\ y(k-1) \\ \vdots \\ y(21) \end{bmatrix}$$

and

$$H = \begin{bmatrix} y(k-1) & y(k-2) & \cdots & y(k-n) \\ y(k-2) & y(k-3) & \cdots & y(k-1-n) \\ \vdots & \vdots & \ddots & \vdots \\ y(21-1) & y(21-2) & \cdots & y(21-n) \end{bmatrix}$$

from above equation, the optimal $\theta_{LS}^{(n)}$ is given by:

$$\theta_{LS}^{(n)} = (H^T H)^{-1} H^T \hat{y} \tag{3}$$

and then, the prediction y is given by:

$$\hat{y}_{pred} = H\theta_{LS}^{(n)} \tag{4}$$

Next, the total prediction error is:

$$error(n) = |\hat{y} - \hat{y}_{pred}||^2 \tag{5}$$

After comparing the errors of different n, we can choose the best n which has the smallest error.

Implementation

The implementation involves preparing the dataset for the AR model, computing the AR parameters for each n, and evaluating the prediction error. The Python package scikit-learn is utilized for fitting the linear regression model, representing the AR model for different lags.

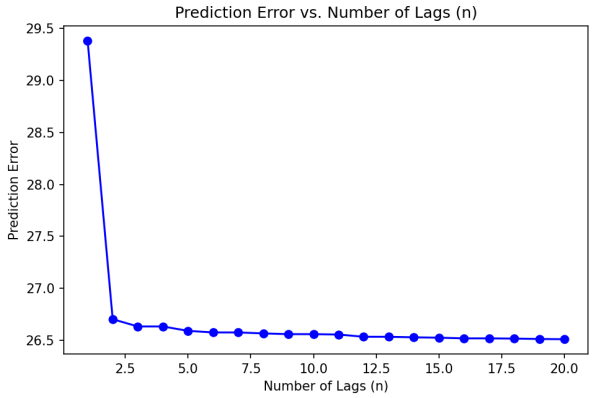
```
# Implementation code snippet
# Create dataset function
def create_ar_dataset(data, n_lags=1):
    """
    Create the dataset for the AR model with the specified number of lags.
    :param data: The time series data.
    :param n_lags: The number of lag terms to use.
    :return: A tuple of (X, y) where X is the matrix of lagged values and
     y is the target variable.
    """
    X, y = [], []
    for i in range(20, len(data)):
        X.append(data[i-n_lags:i])
        y.append(data[i])
    return np.array(X), np.array(y)

# Fit the model and calculate errors for n from 1 to 20
for n in range(1, 21):
    X, y = create_ar_dataset(wind_speed_data, n_lags=n)
    model = LinearRegression().fit(X, y)
    y_pred = model.predict(X)
    error = np.linalg.norm(y - y_pred)
    ar_models[n] = model
    prediction_errors[n] = error
```

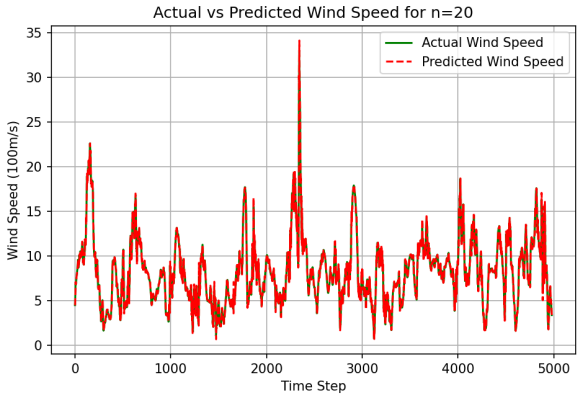
```
# Select the n with the minimum error
min_n = min(prediction_errors, key=prediction_errors.get)
min_error = prediction_errors[min_n]
```

The model is then used to predict wind speed, and the actual vs. predicted values are plotted to assess the performance of the selected model. For the purpose of comparing the total error, we uniformly use the y from data 21-5000 to calculate the prediction errors.

Results

From the optimization, the number of lags n with the smallest prediction error was found to be 20. The prediction errors for different n values and the plot of actual vs. predicted wind speeds suggest that the AR(20) model captures the dynamics of the wind speed well, despite some deviations during peak values. This model can now be utilized for making real-time wind speed predictions in wind farm management.





Additionally, if we want to use the whole 5000 data, for different lag we have to get different sizes of predictions, so we try to use the mean prediction error instead of total error to compare different lag n, and in this case, the best n is 16.

Task 2

Problem Understanding

The federal government requires that cellular network operators have the capability to locate a cell phone when an emergency call is made. The E911 system simplifies this process by using the Time of Arrival (ToA) information from several base stations to estimate the cell phone's location. The challenge is to compute the position x in two-dimensional space, assuming the elevation to be zero, and the time of transmission τ, given the ToA measurements t_i from each base station.

Theoretical Analysis

To estimate the cell phone's location and transmission time, we can formulate this problem as a nonlinear least squares problem. The ToA measurements are related to the distance between the cell phone and each base station and are affected by the noise in the measurements. The theoretical ToA t_i is given by the equation:

$$t_i = \frac{1}{c}||s^{(i)} - x|| + \tau + v_i, i = 1, ..., n \tag{6}$$

Where:

- c is the speed of light (0.3 meters per nanosecond),
- $s^{(i)}$ is the position of the i-th base station,
- x is the position of the cell phone,
- τ is the transmission time,
- v_i is the noise or error in the measured ToA.

Algorithm Analysis

The theoretical analysis for formulating the E911 problem as a nonlinear least squares problem involves defining an objective function that quantifies the difference between the measured times of signal arrival at various base stations and the theoretical times predicted by a model based on the cell phone's location and the signal transmission time. Here's the formulation in detail:

Let $\theta = [x_1, x_2, \tau]^T$, we need to estimate the θ based on the time of arrival (ToA) data t_i from multiple base stations at known positions $s^{(i)}$. Given:

- The speed of light c (0.3 meters/nanosecond in this case).
- The measured ToA t_i from base station i.
- The position $s^{(i)}$ of base station i.

The theoretical ToA for the signal to reach base station i from the cell phone is:

$$\hat{t}_i(\theta) = \frac{1}{c}\|s^{(i)} - x\| + \tau \tag{7}$$

where $||\cdot||$ denotes the Euclidean norm.

The objective function to be minimized is the sum of the squared differences between the measured ToA t_i and the theoretical ToA $\hat{t}_i$, summed over all base stations:

$$f(\theta) = \sum_{i=1}^{n}(t_i - \hat{t}_i(\theta))^2 \tag{8}$$

where n is the number of base stations.

To estimate θ, we solve the following optimization problem:

$$\min_{\theta} f(\theta) \tag{9}$$

This is a nonlinear optimization problem because $\hat{t}_i(\theta)$ is a nonlinear function due to the Euclidean norm involving the square root.

The objective function to minimize is the sum of squared differences between the measured ToAs and the theoretical ToAs, calculated from the cell phone's estimated position and the transmission time. We employ the least squares optimization method to solve this nonlinear problem. The `least_squares` function from `scipy.optimize` is used to perform the optimization.

Implementation

The implementation involves defining the objective function and running the optimization algorithm with an initial guess. The objective function calculates the residuals, which are the differences between the theoretical ToAs and the measured ToAs.

```
# Define the theoretical ToA function
def theoretical_times(theta, base_stations):
    """
    Calculate the theoretical times of arrival at the base stations
    from the estimated position and transmission time.
    :param theta: The parameter vector [x1, x2, tau] where x1 and x2 are
                  the coordinates, and tau is the transmission time.
    :param base_stations: The positions of the base stations.
    :return: Theoretical times of arrival at the base stations.
    """
    x1, x2, tau = theta
    d = np.sqrt((base_stations[0] - x1)**2 + (base_stations[1] - x2)**2)
    return d / c + tau

# Define the objective function
def objective_function(theta, base_stations, measured_times):
    return theoretical_times(theta, base_stations) - measured_times

# Run the least squares optimization
result = least_squares(objective_function, initial_vector,
                                    args=(base_stations, measured_times))
```

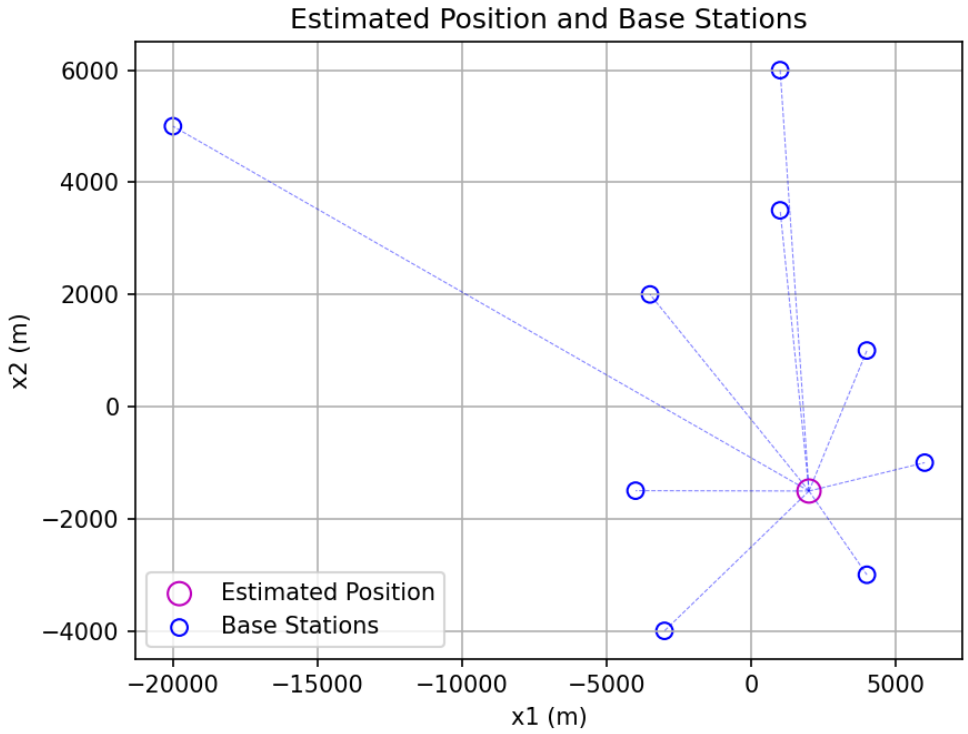
The optimization results in the estimated cell phone position and transmission time.

Results

The estimated parameters from the optimization are as follows:

- Estimated x-coordinate: 1998.33 m
- Estimated y-coordinate: -1506.25 m
- Estimated transmission time: 2986.85 ns

These estimates indicate the probable location and transmission time of the cell phone during the emergency call. The graph showing the relative positions of cell phones and base stations is shown below:



Task 3

Establishment of State Space Equation

According to the equation, we get:

$$\ddot{\theta} = M(\theta)^{-1}(\tau - c(\theta, \dot{\theta}) - g(\theta)) \tag{1}$$

Let's assume:

$$x = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{2}$$

It is easy to get:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} \tag{3}$$

and:

$$\begin{bmatrix} \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = M(\theta)^{-1}(\tau - c(\theta, \dot{\theta}) - g(\theta)) \tag{4}$$

Thus we get the state function:

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_3 \\ x_4 \\ M(\theta)^{-1}(\tau - C(\theta, \dot{\theta}) - g(\theta)) \end{bmatrix} \tag{5}$$

or we can write as:

$$\dot{x} = f(x, \tau) \tag{6}$$

In descrete time domain, we know:

$$x[k+1] - x[k] = f(x[k], \tau[k]) \cdot T \tag{7}$$

That is:

$$x[k+1] = \begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \end{bmatrix} = \begin{bmatrix} x_1[k] + x_3[k] \cdot T \\ x_2[k] + x_4[k] \cdot T \\ M(\theta)^{-1}(\tau - C(\theta, \dot{\theta}) - g(\theta)) \cdot T + \begin{bmatrix} x_3[k] \\ x_4[k] \end{bmatrix} \end{bmatrix} \tag{8}$$

And

$$y[k] = x[k] \tag{9}$$

In the following simulation, the value of T will be 0.001, that is:

$$x[k+1] = \begin{bmatrix} x_1[k] + 0.001x_3[k] \\ x_2[k] + 0.001x_4[k] \\ 0.001 \cdot M(\theta)^{-1}(\tau - C(\theta, \dot{\theta}) - g(\theta)) + \begin{bmatrix} x_3[k] \\ x_4[k] \end{bmatrix} \end{bmatrix} \tag{10}$$

System Simulation

Apply the method we mentioned in descrete time domain, imitate the free-falling model (where $\tau = 0$), we successfully applied it in jupyter notebook. Here are some portions of the simulating results, in Figure.1 and Figure.2, where we take $\theta_1 = 0, \theta_2 = \frac{\pi}{2}$ and $\theta_1 = 0, \theta_2 = -\frac{\pi}{2}$ as the initial states, with $m_1, m_2 = 1kg$, $L_1 = L_2 = 0.5m$ to be the manipulator parameters. The function is given in class `Dropping::forward`, in file `trajectory_falling.py`, and the animation of the falling manipulator is provided in the corresponding jupyter notebook.

System Identification

The system seems to be non-linear. In order to apply least square method, we're going to rectify the system:

$$\tau = M(\theta)\ddot{\theta} + c(\theta, \dot{\theta}) + g(\theta) = Ha$$

with:

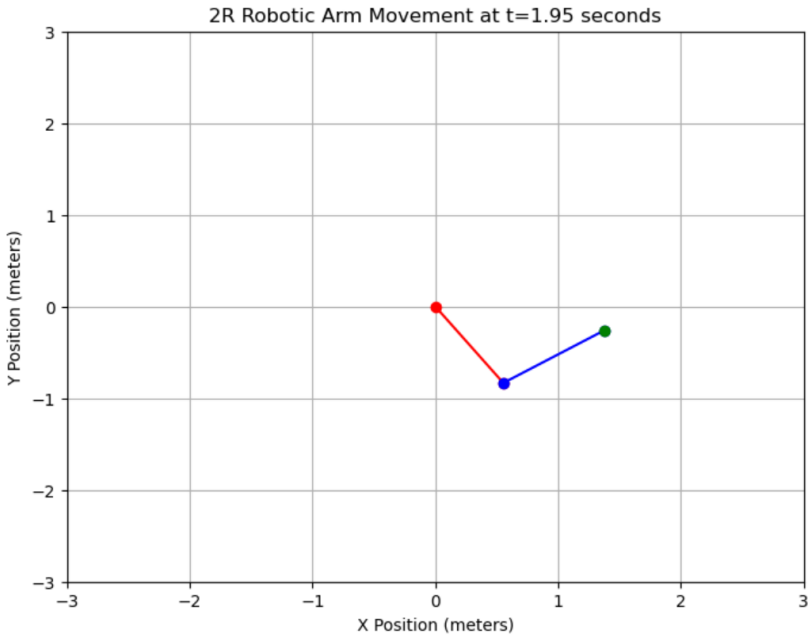


Figure 1: Simulation result when $\theta_1 = 0, \theta_2 = \frac{\pi}{2}$

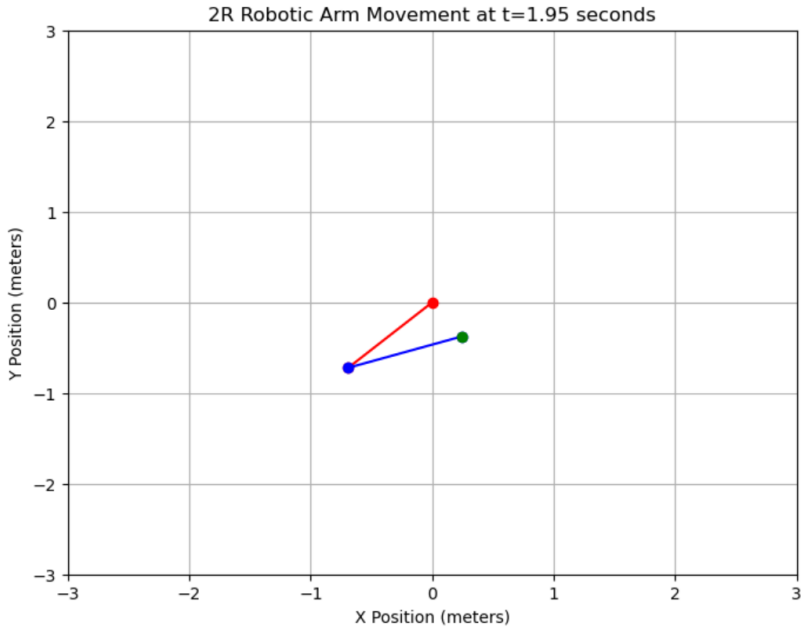


Figure 2: Simulation result when $\theta_1 = 0, \theta_2 = -\frac{\pi}{2}$

$$H = \begin{bmatrix} \ddot{\theta}_1, & \cos(2\ddot{\theta}_1 + \ddot{\theta}_2) - \sin\theta_2(2\dot{\theta}_1\dot{\theta}_2), \ddot{\theta}_1 + \ddot{\theta}_2, g\cos(\theta_1), g\cos(\theta_1 + \theta_2) \\ & 0, \cos\theta_2 \cdot \ddot{\theta}_1 + \sin\theta_2 \cdot \dot{\theta}_1^2, \ddot{\theta}_1 + \ddot{\theta}_2, 0, g\cos(\theta_1 + \theta_2) \end{bmatrix}$$

and

$$a = \begin{bmatrix} m_1 L_1^2 + m_2 L_1^2 \\ m_2 L_1 L_2 \\ m_2 L_2^2 \\ (m_1 + m_2) L_1 \\ m_2 L_2 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}$$

So we can use τ and H to estimate $\hat{a}$:

$$\hat{a} = (H^T H)^{-1} H^T \tau \tag{11}$$

Sometimes, $H^T H$ may be singular, so we can also use pseudo-inverse to solve the problem:

$$\hat{a} = H^+ \tau \tag{12}$$

Where H^+ represents the pseudo-inverse of H.

According to our definition of Matrix a, we know:

$$L_2 = \frac{a_3}{a_5} \tag{13}$$

$$m_2 = \frac{a_5}{L_2} \tag{14}$$

$$L_1 = \frac{a_2}{m_2 L_2} \tag{15}$$

$$m_1 = \frac{a_4}{L_1} - m_2 \tag{16}$$

To examine the result, we use MSE function:

$$err = (a - \hat{a})^2$$

where a is the real value of parameter set and $\hat{a}$ is the predicted parameter set.

We choose 20 samples for establishing the H matrix and τ matrix. Given $\tau = [2, 1]^T$ as constant outerier torque. To better the performance, I choose $T = 0.0001$ as the time difference, using $m_1, m_2 = 1kg,\ L_1 = L_2 = 0.5m$ as simulating parameters, and apply the method to perform least square method, the results are shown in Table.1. And the error on parameters is 9.52×10^{-5}.

	Real Value	**Predict Value**
m_1	1.00	1.00
m_2	1.00	1.00
L_1	0.50	0.50
L_2	0.50	0.50

Table 1: Examine the predictions with T = 0.0001.

	Real Value	**Predict Value**
m_1	1.00	1.00
m_2	1.00	1.00
L_1	0.50	0.50
L_2	0.50	0.49

Table 2: Examine the predictions with T = 0.001.

Compare to $T = 0.001$, the results of T = 0.001 are shown in Table.2, with $err = 0.0079$, performs worse than that of $T = 0.0001$.

The function is provided in class `Identification::forward`, in file `trajectory_identification.py`