

ME424 Project 1 Report  
11912404 江轶豪

1. Wind Power Forecast

solution:

(a) Data:

From the attachment “project1\_1.ipynb” below, we import the data of the second column in the .csv file.

```
import numpy as np
winddata = np.loadtxt("winddata.csv", delimiter=",", skiprows=1, usecols=1)
```

(b) Identification:

Assuming that for the k-th wind speed data, it is correlated with the (k-n)-th to (k-1)-th data. Then the equation can be written as below:

$$y(k) = a_1 y(k-n) + a_2 y(k-n+1) + \cdots + a_n y(k-1) + v_n$$

In the matrix form:

$$\begin{bmatrix} y(k-n) & y(k-n+1) & \cdots & y(k-1) \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \hat{y}(n)$$

And if we expand the matrix and apply first 5000 wind speed data into the matrix, we will get:

$$\begin{bmatrix} y(1) & y(2) & \cdots & y(n-2) & y(n-1) \\ y(2) & y(3) & \cdots & y(n-1) & y(n) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y(4999-n) & y(5000-n) & \cdots & y(4997) & y(4998) \\ y(5000-n) & y(5001-n) & \cdots & y(4998) & y(4999) \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} = \begin{bmatrix} y(n) \\ y(n+1) \\ \vdots \\ y(4999) \\ y(5000) \end{bmatrix}$$

$$\text{where } H = \begin{bmatrix} y(1) & y(2) & \cdots & y(n-2) & y(n-1) \\ y(2) & y(3) & \cdots & y(n-1) & y(n) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y(4999-n) & y(5000-n) & \cdots & y(4997) & y(4998) \\ y(5000-n) & y(5001-n) & \cdots & y(4998) & y(4999) \end{bmatrix}$$

$$\theta = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}$$

From n = 1 to n = 2, the error will be estimated into the formula:

$$err(n) = \sum \|y - H\hat{\theta}_{LS}^{(n)}\|$$

And then compare every  $err(n)$  to get the minimum one.

Since for every  $n = k$ , the amount of their samples may not be the same, for example, for  $n = 1$ , the amount of the samples is 4999, while for  $n = 2$ , it is 4998, it is necessary to let the first test data begin with 21-st wind speed data.

Step 1 of the code:

To define an AR model in a range of 1 to 20, and finally calculate the error and then put them into the array:

```
# define an AR model
def ARmodel():
    for n in range(1,21):
        H=np.zeros((5000-n,n))
        for row in range(0,5000-n):
            for col in range(0,n):
                H[row][col]=winddata[col+row]
        y=np.zeros((5000-n,1))
        for i in range(0,5000-n):
            y[i]=winddata[n+i]
        HT=np.conj(H).T

        if(np.linalg.det(HT.dot(H)) != 0):
            # alpha = (H.T*H)^(-1)*H.T*y to get the parameter
            alpha=((np.linalg.inv(HT.dot(H))).dot(HT)).dot(y)
            # calculate the least square predicted data
            theta=H.dot(alpha)
            # print(theta)
            # calculate the error
            for t in range(20,4999):
                error[n-1] = error[n-1]+(y[t-n]-theta[t-n])**2
        else:
            alpha=9999*np.ones((n,1))
            for t in range(20,4999):
                error[n-1] = error[n-1]+(y[t-n]-theta[t-n])**2
```

Step 2:

Get the minimum error, and also return the index of the error to get its “n”, where  $n = \text{index} + 1$ .

```
# get the index of the minimum data in the 1-d matrix
def findMin_1d(mat):
    min=mat[0]
    min_index=0
    for i in range (0,len(mat)):
        if mat[i] < min:
            min = mat[i]
            min_index = i
    return min_index
```

```
ARmodel()

print(error)
min_item = min(error)
print(min_item)
index=findMin_1d(error)
print(index+1)
num=index+1
```

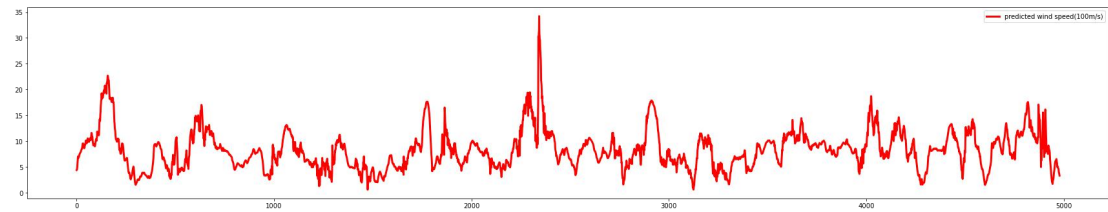
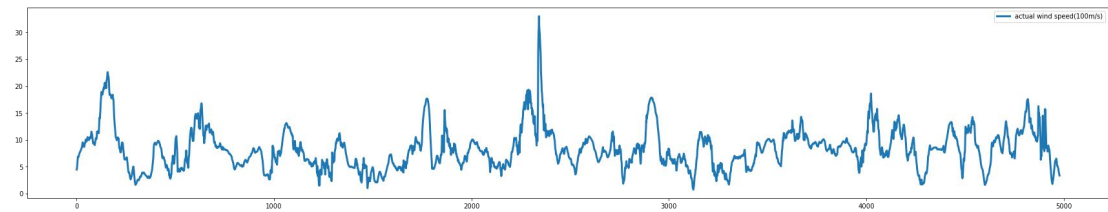
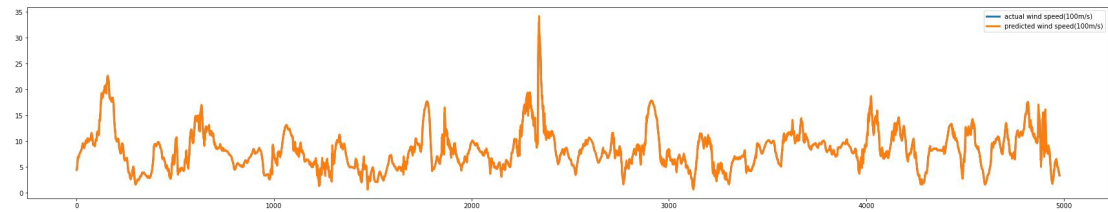
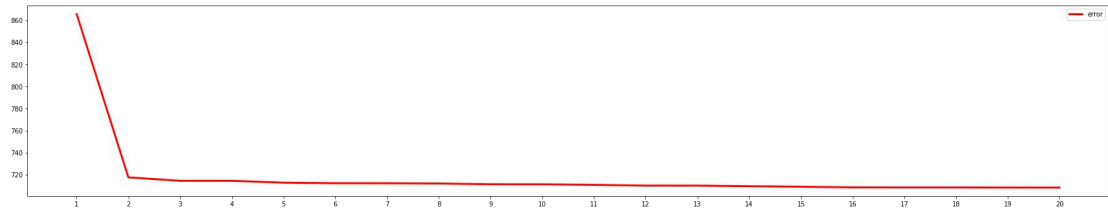
Step 3:

Get “n” and do the AR model again with the specific “n”, where n = 20.

```
H=np.zeros((5000-num,num))
for row in range(0,5000-num):
    for col in range(0,num):
        H[row][col]=winddata[col+row]
y=np.zeros((5000-num,1))
for i in range(0,5000-num):
    y[i]=winddata[num+i]
HT=np.conj(H).T
if(np.linalg.det(HT.dot(H)) != 0):
    alpha=((np.linalg.inv(HT.dot(H))).dot(HT)).dot(y)
yAR=H.dot(alpha)
print(alpha)
```

Step 4: draw the diagram project 1-1 requires: Plot err versus n

```
yreal = np.zeros((5000-num,1))
# yAR = np.zeros((4978,1))
# ypast = np.zeros((num,1))
for i in range(0,5000-num):
    yreal[i] = winddata[i+num]
t=range(0,5000-num)
# for i in range(0,4978):
#     for k in range(i,i+17):
#         ypast = winddata[i:i+20:1]
#         yAR[i][0]=ypast.T.dot(alpha);
import matplotlib.pyplot as plt
# print(yAR)
plt.figure(figsize=(30,25))
plt.subplot(411)
n=range(1,21)
plt.plot(n,error,'r',linewidth=3) #error versus n, where we can find out that n=20 is the best one
plt.xticks(n)
plt.legend(['error'])
plt.subplot(412)
plt.plot(t,yreal,linewidth=3)
plt.plot(t,yAR,linewidth=3)
plt.legend(["actual wind speed(100m/s)","predicted wind speed(100m/s)"])
plt.subplot(413)
plt.plot(t,yreal,linewidth=3)
plt.legend(["actual wind speed(100m/s)"])
plt.subplot(414)
plt.plot(t,yAR,'r',linewidth=3)
plt.legend(["predicted wind speed(100m/s)"])
```



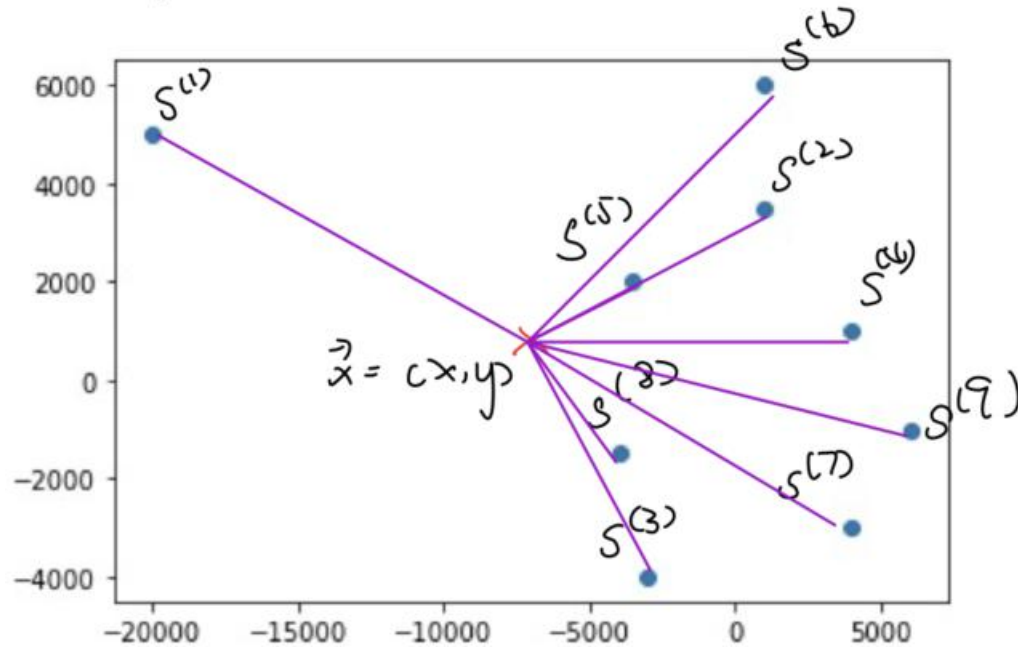
2. E911

(a)

The problem can be seen as a nonlinear least square problem, where

$$|x| \leq 3000, |y| \leq 3000, |\tau| \leq 5000$$

See the figure below:



$$C(t_i - \tau) = \|s^{(i)} - x\| + v_i, i = 1, 2, \dots, 9$$

$$\text{Let } C(t_i - \tau) = d_i, i = 1, 2, \dots, 9$$

then we can choose a cost function:

$$J(x) = \sum_{i=1}^9 (d_i - \|s^{(i)} - x\|), i = 1, 2, \dots, 9$$

Through the definition of the least square, we would find  $J(x)_{\min}$  and

$$x = \arg(J(x)_{\min})_{x \in \mathbb{R}^2}$$

(2) There are two methods to calculate the least square location:

- A. Convert the nonlinear least square problem into a linear one.
- B. Solve the nonlinear least square problem by iteration applying python minimize method.

A: Linear optimization by the least square:

$$c^2(t_i - \tau)^2 = (x_i - x)^2 + (y_i - y)^2$$

for  $i = n-1$

$$x_n^2 + x^2 - 2x_n x + y_n^2 + y^2 - 2y_n y = c^2 t_n^2 + c^2 \tau^2 - 2c^2 t_n \tau$$

for  $i = n-1$

$$x_{n-1}^2 + x^2 - 2x_{n-1} x + y_{n-1}^2 + y^2 - 2y_{n-1} y = c^2 t_{n-1}^2 + c^2 \tau^2 - 2c^2 t_{n-1} \tau$$

The first equation subtracts with the second one:

$$2(x_{n-1} - x_n)x + 2(y_{n-1} - y_n)y - 2c^2(t_{n-1} - t_n)\tau = x_{n-1}^2 - x_n^2 + y_{n-1}^2 - y_n^2 - c^2(t_{n-1}^2 - t_n^2)$$

Then we can transform it into a matrix form and expand it:

$$\begin{bmatrix} 2(x_1 - x_2) & 2(y_1 - y_2) & -2c^2(t_1 - t_2) \\ 2(x_2 - x_3) & 2(y_2 - y_3) & -2c^2(t_1 - t_2) \\ \vdots & \vdots & \vdots \\ 2(x_8 - x_9) & 2(y_8 - y_9) & -2c^2(t_1 - t_2) \end{bmatrix} \begin{bmatrix} x \\ y \\ \tau \end{bmatrix} = \begin{bmatrix} x_1^2 - x_2^2 + y_1^2 - y_2^2 - c^2(t_1^2 - t_2^2) \\ \vdots \\ x_8^2 - x_9^2 + y_8^2 - y_9^2 - c^2(t_8^2 - t_9^2) \end{bmatrix}$$

$$H = \begin{bmatrix} 2(x_1 - x_2) & 2(y_1 - y_2) & -2c^2(t_1 - t_2) \\ 2(x_2 - x_3) & 2(y_2 - y_3) & -2c^2(t_1 - t_2) \\ \vdots & \vdots & \vdots \\ 2(x_8 - x_9) & 2(y_8 - y_9) & -2c^2(t_1 - t_2) \end{bmatrix}$$

$$\theta = \begin{bmatrix} x \\ y \\ \tau \end{bmatrix}$$

$$y = \begin{bmatrix} x_1^2 - x_2^2 + y_1^2 - y_2^2 - c^2(t_1^2 - t_2^2) \\ \vdots \\ x_8^2 - x_9^2 + y_8^2 - y_9^2 - c^2(t_8^2 - t_9^2) \end{bmatrix}$$

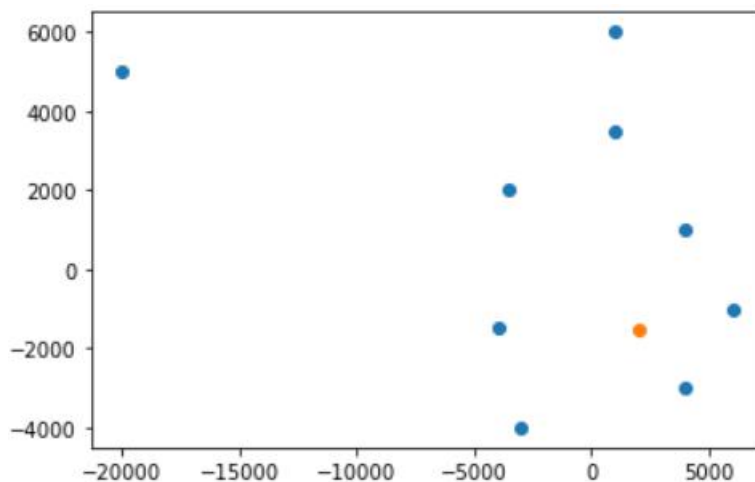
Where  $\hat{\theta}_{LS} = (H^T H)^{-1} H^T y$ , and the code is below:

```

#Linear method
H = np.zeros((8,3))
for i in range(0,8):
    H[i][0] = 2*(x[i]-x[i+1])
for i in range(0,8):
    H[i][1] = 2*(y[i]-y[i+1])
for i in range(0,8):
    H[i][2] = -2*0.3*0.3*(t[i]-t[i+1])
# print(H)
Y = np.zeros((8,1))
for i in range(0,8):
    Y[i] = (x[i]**2-x[i+1]**2)+(y[i]**2-y[i+1]**2)-0.3*0.3*(t[i]**2-t[i+1]**2)
HT=H.T
print(Y)
# print(H.T)
theta=((np.linalg.inv(HT.dot(H))).dot(HT)).dot(Y)
alpha=H.dot(theta)
inv1 = np.linalg.inv(HT.dot(H))
theta = (inv1.dot(HT)).dot(Y)
print(alpha)
print()
print(theta)

posx=np.array([1999.512])
posy=np.array([-1506.86])
plt.scatter(x, y)
plt.scatter(posx, posy)

```



B. Nonlinear method:

In this method, what it matters is the code.

In this code, I choose the method of Basinhopping

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.optimize import basinhopping

```

Since Basinhopping can help me find the global variable instead of the local extreme value.

Define the function of the error:

```
def func(s):  
    func = 0  
    for i in range(0,9):  
        func = func + 0.3*(t[0] - s[2]) - np.sqrt((x[i]-s[0])*(x[i]-s[0]) + (y[i]-s[1])*(y[i]-s[1]))  
    return func
```

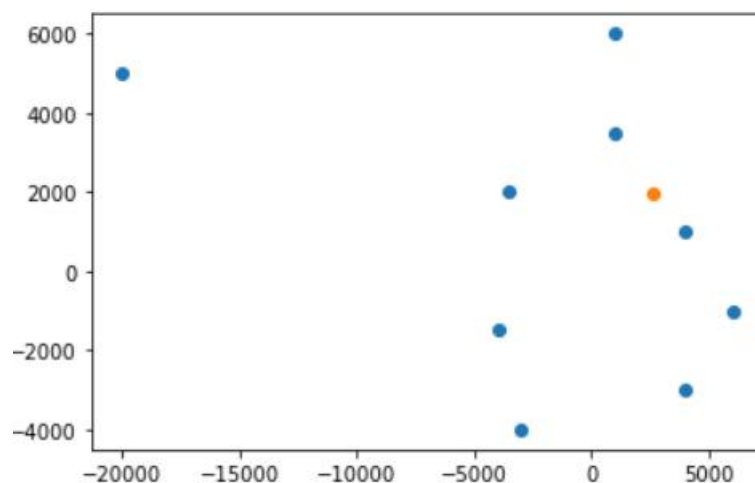
It is used to help me get the answer bounded:

```
class MyBounds:  
    def __init__(self, xmax=[3000,3000,5000], xmin=[-3000,-3000,-5000]):  
        self.xmax = np.array(xmax)  
        self.xmin = np.array(xmin)  
    def __call__(self, **kwargs):  
        x = kwargs["x_new"]  
        tmax = bool(np.all(x <= self.xmax))  
        tmin = bool(np.all(x >= self.xmin))  
        return tmax and tmin
```

Find the minimum:

```
# cons = con()  
x0 = np.array((1500,1500,2500))  
# minLocation = minimize(func(), x0, method='SLSQP', constraints=cons)  
# minLocation = minimize(func(), x0, method='trust-constr', constraints=cons)  
mybounds = MyBounds()  
# globallocation = basinhopping(func,x0,niter=10000, minimizer_kwargs={"method":"L-BFGS-B"}, accept_test=mybounds)  
globallocation = basinhopping(func,x0,niter=1000, minimizer_kwargs={"method":"BFGS"}, accept_test=mybounds)  
# print(minLocation)  
print()  
print(globallocation)  
  
posx=np.array([2601.01])  
posy=np.array([1976.92])  
  
plt.scatter([posx, posy])
```

```
x: array([2601.32769628, 1977.62175314, 4175.72331985])
```



To sum up, the nonlinear method really requires the accuracy and the iteration times. So it is better to use the linear method.

$$x = 1999.5121$$

$$y = -1506.8679$$

$$\tau = 2970.4610$$



```
[[ 1999.51209958]
 [-1506.86794166]
 [ 2970.4610061  ]]
```

### 3. Robot Dynamics Simulation and Parameter Identification: