AMATH301_Homework5_writeup

February 18, 2023

1 Homework 5 writeup solutions

1.1 Name: Aqua Karaman

1.2 Section C

1.3 Problem 1

You are going to need to load in your data again. Do that in the cell below.

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize

#################### Coding Problem 1 #########################
## Part a - Load in the data
# The data is called 'CO2_data.csv'
M = np.genfromtxt("CO2_data.csv", delimiter=",")

# Once you have M defined from CO2_data, uncomment the
# following code to define t and CO2
t = M[:, 0]
CO2 = M[:, 1]

A1 = t
A2 = CO2

## (b)
# Define the error function that calculates the sum of squared error.
# I've started this for you, but you need to fill it in and uncomment.

def sumSquaredError(a, b, r):
#    # Define the model y
    y = lambda t: a + b * np.exp(r*t)

    # Compute the error using sum-of-squared error
    error = np.sum(np.abs(y(t) - CO2)**2)
    return error
```

```python
# Check the error function by defining A3
A3 = sumSquaredError(300, 30, 0.03)
print(A3)

## (c)
# We need an adapter function to make this work with scipy.optimize.fmin
# Uncomment the line below to use the adapter function
adapter_SSD = lambda p: sumSquaredError(p[0], p[1], p[2])

# Once adapter is defined, use fmin
# We use the following guess
guess = np.array([300, 30, 0.03])
ssd_reg_param = scipy.optimize.fmin(adapter_SSD, guess)
A4 = ssd_reg_param

## (d)
# Once we have found the optimal parameters,
# find the error for those optimal parameters
A5 = adapter_SSD(ssd_reg_param)

## (e)
# Now we do the same thing except with max error.
# Your function looks similar, except use the max error
def maxError(a, b, r):
    y = lambda t: a + b * np.exp(r*t)
    max_error = np.max(np.abs(y(t) - CO2))
    return max_error

adapter_maxError = lambda p: maxError(p[0], p[1], p[2])
max_reg_param = scipy.optimize.fmin(adapter_maxError, guess, maxiter=2000)

A6 = adapter_maxError(guess)
A7 = max_reg_param
print('ssd reg param are', A4)
print('ssd error of ssd reg param is', A5)
print('max error of guess is', A6)
print('max reg param are', A7)


## (f)
# This error function has more inputs, but it's the same idea.
# Make sure to use sum of squared error!
def sumSquaredError(a, b, r, c, d, e):
    y = lambda t: a + b * np.exp(r*t) + c * np.sin(d * (t - e))
    ssd_error = np.sum(np.abs(y(t)-CO2)**2)
    return ssd_error
```

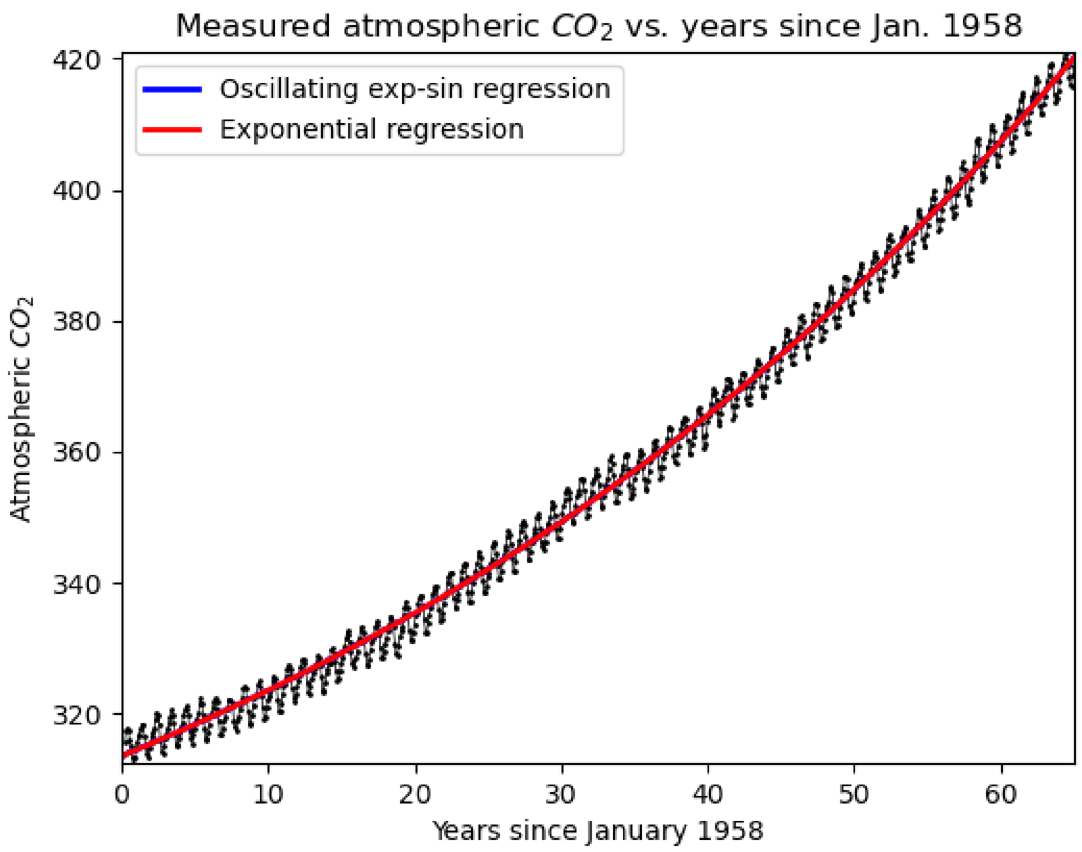
```
guess = np.array([300, 30, 0.03, -5, 4, 0])

adapter_SSD = lambda p: sumSquaredError(p[0],p[1],p[2],p[3],p[4],p[5])
A8 = adapter_SSD(guess)
print('osc reg guess error is', A8)

# And we need to make a new adapter function
# Again, this will have more inputs but will look pretty similar.
osc_reg_param = scipy.optimize.fmin(adapter_SSD, np.append(ssd_reg_param, np.
 ↪array([-5, 4, 0])), maxiter=2000)
A9 = osc_reg_param
print('osc reg param are', A9)

## (h)
# Once we have found the optimal parameters, find the associated error.
A10 = adapter_SSD(osc_reg_param)
print('ssd error of osc reg param is', A10)
```

```
1303827.5069159162
Optimization terminated successfully.
         Current function value: 3860.524834
         Iterations: 141
         Function evaluations: 250
Optimization terminated successfully.
         Current function value: 5.145382
         Iterations: 514
         Function evaluations: 922
ssd reg param are [2.56512390e+02 5.69030054e+01 1.62517171e-02]
ssd error of ssd reg param is 3860.524833526519
max error of guess is 94.02895569295083
max reg param are [2.64855032e+02 4.95219480e+01 1.76198511e-02]
osc reg guess error is 1311088.466981647
Optimization terminated successfully.
         Current function value: 3856.076132
         Iterations: 654
         Function evaluations: 1030
osc reg param are [ 2.56508543e+02  5.69060297e+01  1.62513027e-02
-1.07042079e-01
  6.91389352e+00 -1.14838941e-01]
ssd error of osc reg param is 3856.076131958911
```

1.3.1 Part a - plot

```
[ ]: # Plot data here
oscReg = lambda p,t: p[0] + p[1] * np.exp(p[2]*t) + p[3] * np.sin(p[4] * (t -␣
 ↪p[5]))
expReg = lambda p,t: p[0] + p[1] * np.exp(p[2]*t)
```

```
time = np.arange(0, 65+0.1, 0.01)

plt.figure()
plt.plot(t, CO2, '-k.', markersize=2, linewidth=0.5)
plt.plot(time, oscReg(osc_reg_param, time), '-b', linewidth=2,␣
 ↪label='Oscillating exp-sin regression')
plt.plot(time, expReg(ssd_reg_param, time), '-r', linewidth=2,␣
 ↪label='Exponential regression')

plt.xlim(0, 65)
plt.ylim(np.min(CO2), np.max(CO2))
plt.xlabel('Years since January 1958')
plt.ylabel('Atmospheric $CO_2$')
plt.legend()
plt.title('Measured atmospheric $CO_2$ vs. years since Jan. 1958')

plt.savefig('CO2fig.svg')
plt.savefig('CO2fig.png')
# Then plot exponential + sinusoidal fit - make sure you do this in the right␣
 ↪order!

# Then plot the exponential-only fit

# label, legends, etc. below
```



1.3.2 Part b - Discussion of error

```
[ ]: print('exp reg error is',adapter_SSD(np.append(ssd_reg_param, [0,0,0])))
     print('oscillating exp-sin reg error is',adapter_SSD(osc_reg_param))
```

```
exp reg error is 3860.524833526519
oscillating exp-sin reg error is 3856.076131958911
```

I think I may have a slightly unique set of circumstances on my end, so I'll try to interpret as best as I can given what I can see. Firstly, printed above are the errors for each regression; naturally, the oscillating exponential-sinusoid regression has lower error (approx. 4 units less) than the exponentional regression. I cannot see the plot very well at all on my screen from here (currently remotely connected to my desktop, which I assume skews whatever the native resolution variable Matplotlib uses to define the diensions of a plot). In SVG format, however, I can sort of see how the blue exp-sin regression plot slightly deviates from the exponential-only plot, fitting better with the oscillations and amplitudes of the data.

1.3.3 Part c - Prediction

To predict the CO_2 levels in 2023, it would be a better idea to use the exponential-sinusoid regression. This model better fits seasonal fluctuations displayed in the original data, and as such would be closer to the real life value. I will be honest though, I'm curious if there's an even better model that will fit the amplitude of the oscillations more accurately.

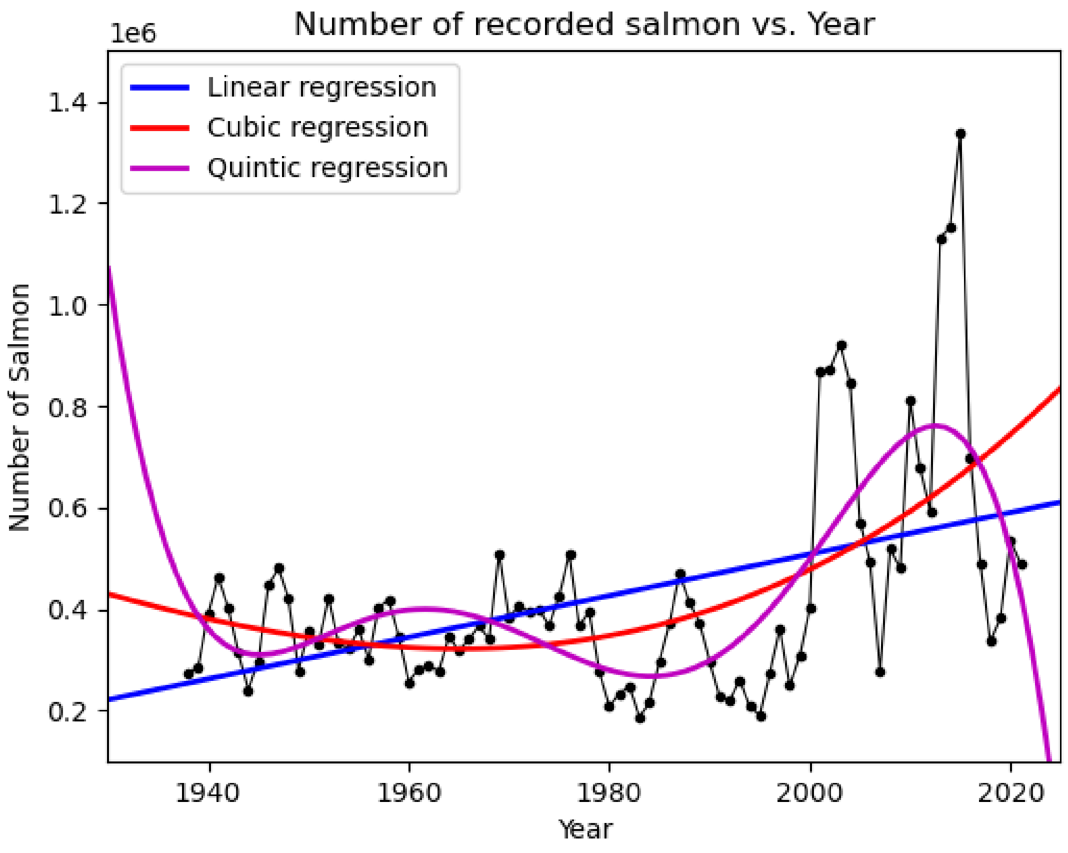
1.4 Problem 2

We'll need to load in the Salmon data again. Do that below.

```
[ ]: ######################### Coding problem 2 ###################
## Part (a)
M = np.genfromtxt('salmon_data.csv', delimiter=',', dtype=float)

year = M[:,0] #Assign the 'year' array to the first column of the data
salmon = M[:,1] #Assign the 'salmon' array to the first column of the data

## (b) - Degree-1 polynomial
linRegP = np.polyfit(year, salmon, 1)
A11 = linRegP

## (c) - Degree-3 polynomial
cubRegP = np.polyfit(year, salmon, 3)
A12 = cubRegP

## (d) - Degree-5 polynomial
qntRegP = np.polyfit(year, salmon, 5)
A13 = qntRegP

## (e) - compare to exact number of salmon
exact =  752638 # The exact number of salmon
err0 = np.abs(np.polyval(linRegP, 2022) - exact)/exact
err1 = np.abs(np.polyval(cubRegP, 2022) - exact)/exact
err2 = np.abs(np.polyval(qntRegP, 2022) - exact)/exact
errs = np.array([err0, err1, err2])
A14 = errs
```

1.4.1 Part a - plot

```
[ ]: plt.figure()

x = np.linspace(1930, 2025, 100)

plt.plot(year, salmon, '-k.', linewidth='0.8')
plt.plot(x, np.polyval(linRegP, x), '-b', label='Linear regression',␣
 ↪linewidth=2)
plt.plot(x, np.polyval(cubRegP, x), '-r', label='Cubic regression', linewidth=2)
plt.plot(x, np.polyval(qntRegP, x), '-m', label='Quintic regression',␣
 ↪linewidth=2)

plt.xlim(1930, 2025)
plt.ylim(1e5,1.5e6)
plt.xlabel('Year')
plt.ylabel('Number of Salmon')
plt.title('Number of recorded salmon vs. Year')
plt.legend()

plt.savefig('salmonfig.svg')
```



1.4.2 Part b - Line of best fit discussion

The slope of the linear regression (or line of best fit) shows that, over the range of values, the population of salmon on average increased as time passed.

1.4.3 Part c - Accuracy of predictions

Of the used regressions, the quintic model by far gave the most accurate prediction for the number of salmon in 2022. In contrast, the cubic regression interestingly gave the worst prediction.

1.4.4 Part d - Predicting Salmon populations in 2050

```
[ ]: print('The linear regression predicts the number of salmon in 2050 will be', np.
      ↪polyval(linRegP, 2050))
     print('The cubic regression predicts the number of salmon in 2050 will be', np.
      ↪polyval(cubRegP, 2050))
     print('The quintic regression predicts the number of salmon in 2050 will be',␣
      ↪np.polyval(qntRegP, 2050))
```

```
The linear regression predicts the number of salmon in 2050 will be
712512.5765549596
The cubic regression predicts the number of salmon in 2050 will be
1458352.528254509
The quintic regression predicts the number of salmon in 2050 will be
-17028352.5625
```

Although I don't really trust any of these models to predict the number of salmon in 2050, I'd rank them in the following order: I would trust the linear regression most, followed by the cubic regression, and then the quintic regression in dead last. Above are the predicted values for each regression, respectfully. Excluding possible outlier years, I find it very unlikely the population of salmon reaches about 1.5 million in 2050. There's a better chance of the population reaching 700,000 salmon in 2050, instead. And although I don't have much authority in the realm of statistics and probability, I'd say there's a 0% chance there are negative 17 million salmon in 2050.