AMATH 301 Rahman

Due Thursday, May 13th at 11:59pm

Problem 1

The file `lynx.csv` contains estimates of the population of lynx in the Bialowieza Primeval Forest (BPF) in Belarus between 1946 and 1994. (The data comes from "Population dynamics (1869-1994), demography, and home ranges of the lynx in Bialowieza Primeval Forest (Poland and Belarus)" by Jedrzejewski et al., in the journal *Ecography*, 1996.) You can load the data with the code

```
data = readmatrix('lynx.csv');
t = data(1, :);
pop = data(2, :);
```

in MATLAB or

```
data = np.genfromtxt('lynx.csv', delimiter=',')
t = data[0, :]
pop = data[1, :]
```

in python. This will create two variables: `t` contains times measured in years since 1946 (so $t = 0$ corresponds to 1946 and $t = 48$ corresponds to 1994) and `pop` contains populations. Each entry of `pop` gives the population of lynx in the corresponding year, so the first entry is the population at $t = 0$ and the last entry is the population at $t = 48$. Note that `t` and `pop` will both be 1×49 row vectors in MATLAB, but they will be 1D arrays with 49 entries in python. It is probably best to keep them in this format, and then reshape answers to 2D arrays as necessary once you are done.

Most of the data in this set came from censuses of Belarusian lynx, but two of the censuses were deemed unreliable. To correct this, the authors performed some statistical analysis (comparing populations in Belarus and Poland) and suggested that the population in 1956 should actually be 34 (instead of 14) and the population in 1974 should actually be 27 (instead of 62). Fix the corresponding entries of the vector `pop` to account for this update. Save the updated population vector `pop` as `A1`. The rest of this problem will use the updated values.

(1) Use a cubic spline to interpolate this data at $t = 24.5$. (In other words, predict the population half way between year 1970 and 1971 using a cubic spline.) Save your interpolated value (i.e., your predicted population) in a variable named `A2`.

(2) Use the `polyfit` function to find the best fit line $y = mt + b$ for this data. Save the resulting slope and y-intercept in a 1×2 row vector `[m, b]` named `A3`. (This is the format that polyfit returns in MATLAB, but in python you will need to reshape your answer). In scientific computing we usually use the l_2 norm (the square root of the error in the theory lecture) to calculate our errors (different from the error in the coding lecture called the root mean square error). Find the l_2 error, $\|y - \texttt{pop}\|_2$ using `norm(y - pop)` on MATLAB and `scipy.linalg.norm(y - pop)`, of this fit and save it in a variable named `A4`.

(3) Use the `polyfit` function to find the best fit quadratic $y = at^2 + bt + c$ for this data. Save the resulting coefficients in a 1×3 row vector `[a, b, c]` named `A5`. (This is the format that polyfit returns in MATLAB, but in python you will need to reshape your answer.) Find the l_2 error of this fit and save it in a variable named `A6`.

(4) Use the `polyfit` function to find the best fit 10th order polynomial $y = a_{10}x^{10} + a_9x^9 + \cdots a_1x^1 + a_0$ for this data. Save the resulting coefficients in a 1×11 row vector `[a10, a9, ..., a1, a0]` named `A7`. (This is the format that polyfit returns in MATLAB, but in python you will need to reshape your answer.) Find the l_2 error of this fit and save it in a variable named `A8`.

Problem 2

The file `CO2_data.csv` contains monthly averages of atmospheric CO_2 (in parts per million) measured at the Mauna Loa observatory in Hawaii. (You can find this data, along with much more information about the measurements, here.) You can load this data with the code

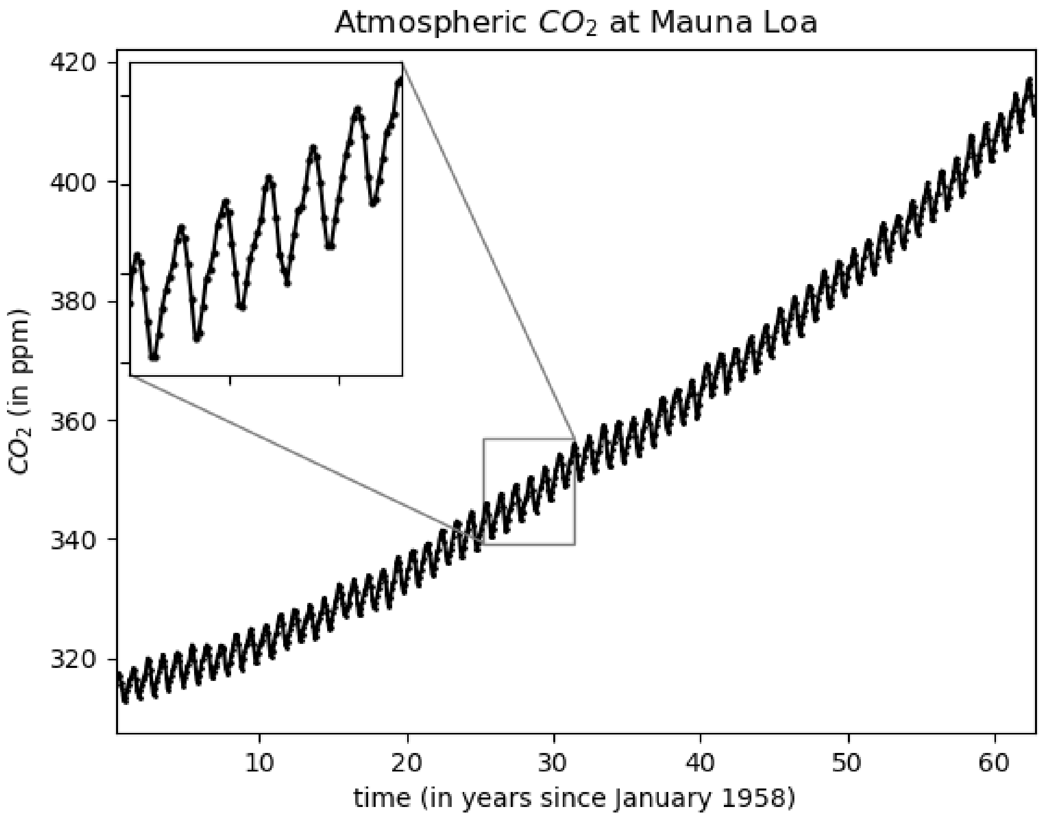
```
data = readmatrix('CO2_data.csv');
t = data(1, :);
co2 = data(2, :);
```

in MATLAB or

```
data = np.genfromtxt('CO2_data.csv', delimiter=',')
t = data[0, :]
co2 = data[1, :]
```

in python. This will create two variables: `co2` contains atmostpheric CO_2 at each month, and `t` contains time measured in years since the beginning of 1958. This means that the first entry in `t` is 3/12, because the first measurement was in March, 1958 and March is the third month of the year. The last entry in `t` is 62 and 10/12 (or 62.8333), because the last measurement I included in the data set was from October, 2020. (The year 2020 is 62 years after 1958, and October is the 10th month of the year.) Note that `t` and `co2` will both be row vectors in MATLAB, but they will be 1D arrays in python. It is probably best to keep them in this format, and then reshape answers to 2D arrays as necessary once you are done.

The CO_2 concentration has risen steadily over the last 62 years, but also undergoes distinct seasonal oscillations:



(1) Find a best fit line $f(t) = mt + b$ for this data using `polyfit`. Save the resulting slope and y-intercept in a 1×2 row vector `[m, b]` named `A9`. (This is the format that polyfit returns in MATLAB, but in python you will need to reshape your answer.) In scientific computing we usually use the l_2 norm (the square root of the error in the theory lecture) to calculate our errors (different from the error in the coding lecture called the root mean square error). Calculate the l_2 error, $\|f(t) - \texttt{co2}\|_2$ using `norm(f - co2)` on MATLAB and `scipy.linalg.norm(f - co2)`. [Caveat: only works if f has the same discretization as co2. How can you make sure of that?] Save it in a variable named `A10`.

(2) It looks like the data might be growing roughly exponentially. Further, suppose we know that before the industrial revolution, CO2 levels were steady around 260 parts per million; i.e., $\lim_{t\to-\infty} f(t) = 260$. This means that we should try to find a best fit curve of the form

$$f(t) = ae^{rt} + b = e^{rt+\ln a} + b; \qquad \lim_{t\to-\infty} f(t) = 260$$

while there are more sophisticated algorithms in MATLAB and Python packages to do this sort of fitting, let's use what we know to come up with an exponential fit ourselves instead of blindly applying a predefined function of a sophisticated algorithm. Notice, that in the coding lecture we did the fit for a function of the form $ae^{rt} = e^{rt+\ln a}$ by taking the natural log of our data using the `log` function and doing a linear fit on that. Here however we have $+b$, so lets take our data and subtract b, `co2` $- b$ and then do the exponential fit just like in the lecture. Save your answers in a 1×3 row vector `[a, r, b]` named `A11`. Find the l_2 error (as described in Part 1) of this fit and save it in a variable named `A12`.

(3) You should find that the best fit exponential curve captures the overall trend of your data well, but it does not do a good job of capturing seasonal oscillations. To fix this, we can try to find a best fit curve of the form

$$f(t) = ae^{rt} + b + A\sin(Bt).$$

There are some sophisticated data analysis algorithms for this as well, but lets think in terms of traditional scientific computing to use empirical observations to inform our models. The exponential did a decent job capturing the average trend, so lets not change that. Now, how do we find B? Well B depend on the period of the seasonal cycles. How long is one period of a seasonal cycle? What value of B will make our sine function match the period of the seasonal cycles? (You don't need to do any coding for B). For A, we want the amplitude of the sine function. So, let's isolate the sine and find the amplitude. Consider `co2` $- (ae^{rt} + b)$. We can find the amplitude of this function by taking the difference between the peaks and troughs and dividing by two. However, this function has multi-year oscillations as well, so lets average the yearly low to high range. Write a loop that goes twelve months at a time for 62 years, and sum the difference between the max and min of `co2` $- (ae^{rt} + b)$. Then to get the average divide this sum by 62: that will be your A. Save your answeres in a 1×2 row vector `[A, B]` named `A13`. Find the l_2 error of this fit and save it in a variable named `A14`.