COMS 3251 Spring 2021: Lab 5

Instructions: Your TAs will go over the tutorial sections of this lab, cover examples, and answer questions. After that, there will be allotted time for you and your partner to complete the exercises in the second half of the notebook. During this time, the TAs will answer individual questions to the extent possible. Your group should try to submit the completed notebook on Gradescope prior to the end of your section time.

COVID-19 Vaccinations

In this lab you will be looking at real-time data tracking the progress of COVID-19 vaccinations in the United States. There are several public data sets; we will be using the one from Our World in Data. Each row of this data set provides vaccination numbers on a given date of a particular state or the US itself; each column indicates metrics such as daily vaccinations or people fully vaccinated.

First import the latest data set by uploading the provided CSV file under Files to the left and then running the code cell below. We will be using pandas, Python Data Analysis Library, to import and read the data.

```
import pandas as pd
import numpy as np
import numpy.linalg as nla
import matplotlib.pyplot as plt

ts = pd.read_csv('us_state_vaccinations.csv')
data = ts.values
pd.DataFrame(data)
```

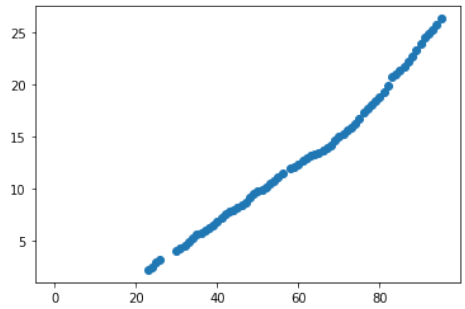
	0	1	2	3	4	5	6	7	8	9	10	11	
0	2021-01-12	Alabama	78134	377025	70861	0.15	1.59	7270	1.45	7.69	NaN	NaN	N
1	2021-01-13	Alabama	84040	378975	74792	0.19	1.71	9245	1.53	7.73	5906	5906	1:
2	2021-01-14	Alabama	92300	435350	80480	NaN	1.88	NaN	1.64	8.88	8260	7083	1·
3	2021-01-15	Alabama	100567	444650	86956	0.28	2.05	13488	1.77	9.07	8267	7478	1:
4	2021-01-16	Alabama	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	7557	7498	1:
...	...	...	...	...	...	...	...	...	...	...	...	...	
4763	2021-03-21	Wyoming	221336	314015	134703	14.77	38.24	85508	23.27	54.26	343	1347	2:
4764	2021-03-22	Wyoming	221452	314015	134800	14.78	38.26	85556	23.29	54.26	116	1279	2:

Data Extraction

`data` contains all the information contained in the CSV file. We will look at the portion of the data pertaining to the US as a whole. We will first use `numpy.where` to extract the correct row indices. The example below then further extracts the data of column `8`, which corresponds to the number of people who have received at least one vaccine dose per 100 people. Finally, this information is plotted on a scatter plot.

```
indices = np.where(data == 'United States')[0]

# people vaccinated per hundred
vax = data[indices,8]
n = vax.size
days = np.arange(n)

print(vax)
plt.scatter(days, vax)
```

```
[nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan nan 2.23 2.41 2.92 3.19 nan nan nan 4.1 4.3 4.53 4.89
 5.24 5.57 5.8 5.99 6.23 6.54 6.89 7.25 7.59 7.84 7.96 8.18 8.41 8.71 9.11
 9.51 9.74 9.9 10.18 10.46 10.79 11.16 11.53 nan 11.95 12.13 12.36 12.64
 12.89 13.14 13.29 13.42 13.63 13.88 14.21 14.59 14.99 15.28 15.59 15.92
 16.28 16.73 17.28 17.73 18.07 18.4 18.81 19.3 19.87 20.75 21.02 21.4
 21.73 22.19 22.74 23.26 23.91 24.52 24.93 25.28 25.74 26.31]
<matplotlib.collections.PathCollection at 0x7f8d47f77490>
```

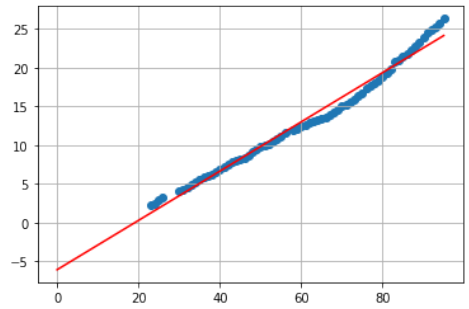


Real-world data is messy. Notice in particular the missing data points, most of which is at the beginning. The value 0 on the horizontal axis is associated with December 20, 2020, while the value 23 is associated with January 12, 2021 and the first available datum.

Linear Regression

We can perform linear regression on this data and fit a trend line, as long as we are careful with the missing data. We first form the design matrix and output vector using the data structures above. We then remove the rows from both structures containing missing data. After solving for the model parameters, we can then compute and plot the fitted line on the data.

```
nan_idx = np.where(pd.isnull(vax))
X = np.column_stack((np.ones(n), days))
X = np.delete(X, nan_idx, 0)
b = np.delete(vax, nan_idx)

fit = nla.lstsq(X, b.astype(float), rcond=None)
beta = fit[0]
plt.scatter(days, vax)
plt.plot(days, np.column_stack((np.ones(n), days)) @ beta, 'r')
plt.grid()
```



The fitted line is shown in red above. It should look like a pretty good fit for the most part, with a couple of caveats. The most recent data should appear to be veering away from a linear trend to a steeper rise (a good thing!). On the other end of the domain, you should see that the line extends downward into "negative people vaccinated" at the beginning. Of course this makes no sense, and one takeaway is that this particular data has not always been well described by the same linear trend.

You will be exploring other aspects of this data set in the exercises below.

Exercises

PROBLEM 1 (20 points)

Let's now investigate the number of people who are *fully* vaccinated per hundred. This data is given in column index `5`. If you replicate the process in the example above, you will see that the fit again extends into the negative for the first few weeks.

We can try to fix this problem by making the following modifications. First, set the first value of the data (on day 0) to be 0 so that it's no longer `nan`. Second, a linear fit may not be best. Instead, we can use polynomial or exponential basis functions, the latter of which can be approximated by a sum of increasing powers of the data values.

Use the hypothesis

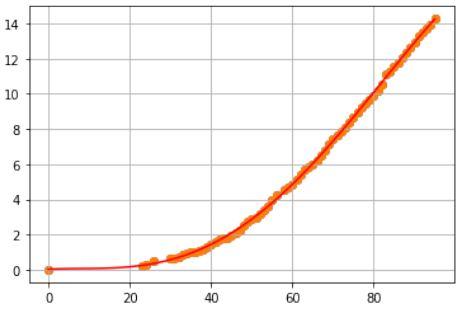
$$\hat{f}(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4.$$

As a hint, this will require you to include additional columns in the design matrix X. Each additional column should correspond to one of the new basis functions of x. Output a plot with the fit overlaid on top of the scatter plot of points, and then answer the questions below.

```
# YOUR CODE HERE
indices = np.where(data == 'United States')[0]
# full vaccinated per 100
vax = data[indices,5]
vax[0] = 0
n = vax.size
days = np.arange(n)
#vax[0] = 0

print(vax)
plt.scatter(days, vax)
```

```
nan_idx = np.where(pd.isnull(vax))
X = np.column_stack((np.ones(n), days, days ** 2, days ** 3, days ** 4))
X = np.delete(X, nan_idx, 0)
b = np.delete(vax, nan_idx)

b = np.delete(vax, nan_idx)

fit = nla.lstsq(X, b.astype(float), rcond=None)
beta = fit[0]
plt.scatter(days, vax)
plt.plot(days, np.column_stack((np.ones(n), days, days ** 2, days ** 3, days ** 4)) @ beta, 'r')
plt.grid()
print(beta)
```

```
[0 nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan nan nan 0.24 0.31 nan 0.49 nan nan nan 0.61 0.65 0.72 0.83 0.91
 0.97 1.01 1.05 1.14 1.28 1.44 1.58 1.7 1.79 1.83 1.94 2.09 2.26 2.51 2.76
 2.87 2.96 3.15 3.37 3.64 3.94 4.24 nan 4.52 4.66 4.87 5.13 5.39 5.68 5.86
 5.99 6.21 6.49 6.81 7.14 7.46 7.67 7.88 8.12 8.37 8.65 8.97 9.24 9.49
 9.67 9.91 10.2 10.54 11.12 11.28 11.55 11.76 12.05 12.34 12.63 12.96 13.3
 13.53 13.72 13.97 14.28]
[ 2.49667632e-02  1.20279884e-02 -1.35838967e-03  6.24586014e-05
 -3.46218259e-07]
```



1. Why is this fit able to avoid the problem that the original linear fit encountered for the first 23 days of the domain? Other than this problem, what other metric(s) can you compare to show that this hypothesis may be better than a strictly linear one?
2. Report the β_i coefficients. Looking at their magnitudes, which of the basis functions of the hypothesis carry the most "weight"? If we have to drop one or two of the basis functions, which ones should we drop first?

ENTER YOUR RESPONSES HERE

1. The problen with a linear fit is that it includes negative values when it should not. The linear fit cannot avoid the problem because f(x) is always positive. For a nonlinear scatter plot it makes more sense to also use a polynomial fit rather than linear expecially for this data set. Linear fits cannot account for varying rates of change over time.

You can also use the sum of squares. $\sum_i (y_i - (\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4))^2$

2. $\beta = [2.49667632e-02, 1.20279884e-02, -1.35838967e-03, 6.24586014e-05, -3.46218259e-07]$

Looking at the magnitudes, β_a carries the moist weight. β_4 should be dropped because it has the smallest magnitude and weight.

x^3 and x^4 carry the least weight.

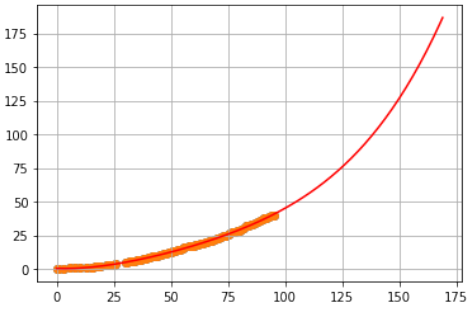
PROBLEM 2 (20 points)

Regression is typically used to infer correlations and causal relationships between variables, but it can also be used for prediction. Let's now look at total vaccinations (first and second doses combined) per 100 people. This data is given in column index `6`.

Replicate the entire procedure of the previous problem on this data, although you will not have to explicitly set the value at 0. Then when overlaying the fitted curve on the scatter plot, extend the curve all the way to day 170 instead of stopping at the last day of the data.

```
# YOUR CODE HERE
indices = np.where(data == 'United States')[0]

# people fully vaccinated per 100
vax = data[indices,6]
n = vax.size
days = np.arange(n)

print(vax)
plt.scatter(days, vax)

nan_idx = np.where(pd.isnull(vax))
X = np.column_stack((np.ones(n), days, days ** 2, days ** 3, days ** 4))
X = np.delete(X, nan_idx, 0)
b = np.delete(vax, nan_idx)

fit = nla.lstsq(X, b.astype(float), rcond=None)
beta = fit[0]
plt.scatter(days, vax)
days = np.arange(170)
plt.plot(days, np.column_stack((np.ones(170), days, days ** 2, days ** 3, days ** 4)) @ beta, 'r')
plt.grid()
print(beta)
```

```
[0.17 0.18 nan 0.3 nan nan 0.59 nan 0.64 nan 0.84 nan nan 1.27 nan 1.37
 1.46 1.6 1.78 2.01 nan nan 2.71 2.81 3.1 3.36 3.7 nan nan nan 4.73 4.98
 5.29 5.76 6.19 6.58 6.85 7.09 7.43 7.89 8.4 8.91 9.37 9.71 9.87 10.2 10.6
 11.09 11.76 12.41 12.78 13.01 13.49 13.97 14.58 15.25 15.93 nan 16.63
 16.95 17.39 17.95 18.46 19.0 19.33 19.59 20.02 20.56 21.22 21.93 22.66
 23.16 23.68 24.26 24.87 25.61 26.48 27.21 27.74 28.22 28.83 29.58 30.46
 31.84 32.25 32.86 33.36 34.05 34.86 35.64 36.58 37.49 38.11 38.62 39.3
 40.15]
[ 3.86038522e-01 -5.63855544e-02  8.84013059e-03 -7.62711145e-05
  3.81765789e-07]
```



1. On approximately what day do we expect to have administered 100 total vaccinations per 100 people in the US? How about 200? Give a reason for why 200 vaccinations may never actually be attained. For reference, all current vaccines require no more than 2 doses, and they have only been approved for adults (which comprise about 74% of the total US population).
2. Briefly describe what you expect the actual vaccination curve to look like over the next 100 days. What should we be mindful of when using regression for prediction, especially for systems with exponential growth?

ENTER YOUR RESPONSES HERE

1. 100 total vaccines by 100 people would be expected on the 135th day and 200 vaccines by the 185th day. Because only 74% of the population currently is eligible for the vaccine, this implies that the maximum of 148 doses can be given.

2. The actual vaccine curve should be closer to linear as opposed to exponentially increasing. The rate of available vaccinations is increasing, but the total number of people seeking for a vaccine is decreasing. Also, not the whole population is eligible or seeking. Therefore, the growth is expected to stay constant or maybe diminish/ stay the same (sigmoid function). When appkying regression to systens with exponentiak growth, you need to pay attention to overfitting curves to early data, because the conditions may change over time.

PROBLEM 3 (20 points)

Oftentimes, a hypothesis described solely by a few polynomial basis functions is not sufficient. A more flexible hypothesis is the auto-regressive model, which explicitly uses past data to make future predictions:

$$\hat{x}_{t+1} = \beta_1 x_t + \cdots + \beta_M x_{t-M+1}$$

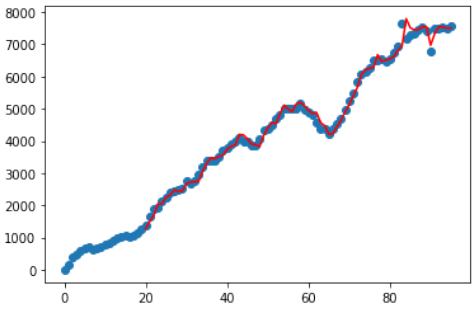
M denotes the memory length of the model; e.g., if $M = 3$ then $\hat{x}_{t+1} = \beta_1 x_t + \beta_2 x_{t-1} + \beta_3 x_{t-2}$. The data that we are fitting will thus appear in both the output vector as well as the design matrix. The following would be the linear regression problem for $M = 3$:

$$\begin{bmatrix} x_3 & x_2 & x_1 \\ x_4 & x_3 & x_2 \\ \vdots & \vdots & \vdots \\ x_{n-2} & x_{n-3} & x_{n-4} \\ x_{n-1} & x_{n-2} & x_{n-3} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} x_4 \\ x_5 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}$$

While you can construct these quantities for any M using loops, we are providing you with the `AR` procedure below to form them so that you can focus more on the outputs and analysis. For this last problem, you will look at daily vaccinations per million (column index `12`). The code below will run the regression for a given value of M and plot the output.

```
def AR(data, M):
  n = data.size
  idx,_ = np.mgrid[0:n-M, 0:M] + np.arange(0,M)
  X = vax[idx].astype(float)
  b = vax[M:].astype(float)
  return X, b

# daily vaccinations per million
vax = data[indices,12]
vax[0] = 0
n = vax.size
days = np.arange(n)

M = 20
X, b = AR(vax, M)
fit = nla.lstsq(X, b, rcond=None)
beta = fit[0]
plt.scatter(days, vax)
plt.plot(np.arange(M,n), X @ beta, 'r')
```

```
[<matplotlib.lines.Line2D at 0x7f8d4798e590>]
```



You should see that the auto-regressive model generates a very close fit of the data regardless of its trends. One open question is the value of M. The best value is the one that minimizes the root-mean-square (RMS) error, or the norm of the difference between the predicted value and the true value divided by the size of the fitted data. In other words,

$$RMS = \frac{||\mathbf{b} - X\hat{\beta}||}{\sqrt{n - M}}$$

where n is the size of the original data set. Note that $n - M$ becomes smaller as M gets larger, since the first $M - 1$ data have no predictions.

Write code below to compute the RMS for values of M ranging from 1 to 50. You will have to use a loop and fit an auto-regressive model for each value. Generate a single plot of RMS versus M as your output.

```
# YOUR CODE HERE
import math


y_values = []
x_values = []

for M in range(1, 51):
    X, b = AR(vax, M)
    fit = nla.lstsq(X, b, rcond=None)
    beta = fit[0]

    result = b - X.dot(beta)
    RMS = nla.norm(result) / math.sqrt(data.size - M)

    #append
    y_values.append(RMS)
    x_values.append(M)
plt.scatter(x_values, y_values)
print (beta)
```

```
[-0.07015705  0.18197279  1.86058143 -0.00978171 -0.25214249 -0.21928394
  0.54714586 -0.21262555  0.0777477  -0.42956826 -0.11379527 -1.02986142
 -0.11352967  1.28285501 -1.01979142 -1.11951828  0.67854001 -0.27201625
 -0.17269144  0.0404771   0.33150609 -0.74863404 -0.09955585  0.72833496
  0.09688849  0.84926772 -0.90644891  0.52710069  0.75054103 -0.63833125
  0.46078967 -0.01999314 -0.36785486  0.05750671  0.0778314   0.87747872
 -0.52059681  0.29977817 -0.05190013  0.0152768  -0.50507562  0.22020753
  0.37698747 -0.48408875 -0.10640451  0.59004136 -0.13954264 -0.49258319
  0.48847053  0.25698673]
```