MATH 3310 Mathematical Modeling
Instructor: Korytowski
Midterm Exam
Student First/Given name: **THOMAS**
Student Last Name: **NEYMAN**
Note if you aren't printing this off you are still required to rewrite the Honor statement like this, and sign the Honor statement. If you are emailing in and not printing, you can type your name to constitute a signature.
Honor Statement: By signing below I confirm that I have neither given nor received any unauthorized assistance on this exam. Furthermore, I agree not to discuss this exam with anyone until the exam testing period is over.
Signature: Thomas Neyman
Date: March 4th 2022

# Instructions

- There are 6 questions worth a total of 50 marks. For any question you want you can choose to do it by hand or by code, but if you do it by hand show work.
- This exam is due Sunday March 6th by 11:59pm, submit it by email like you have been doing for assignments.

1.

a)

30% of the value is decreased each iteration, so 70% is left over. That leaves an equation where:

$y_{(n+1)} = y_n * 0.7$

b)

$y_0 = 90$

$y_1 = 90 * 0.7 = 63$

$y_2 = 63 * 0.7 = 44.1$

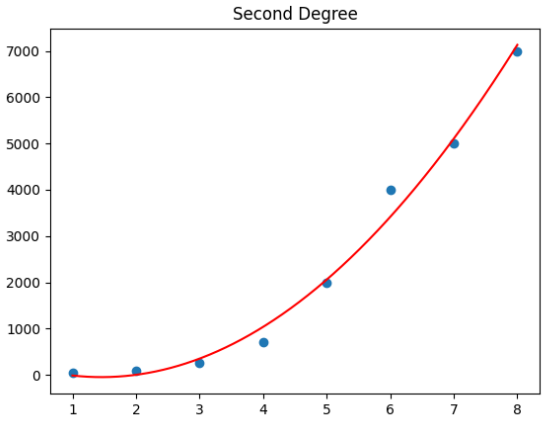
$y_3 = 44.1 * 0.7 = 30.87$

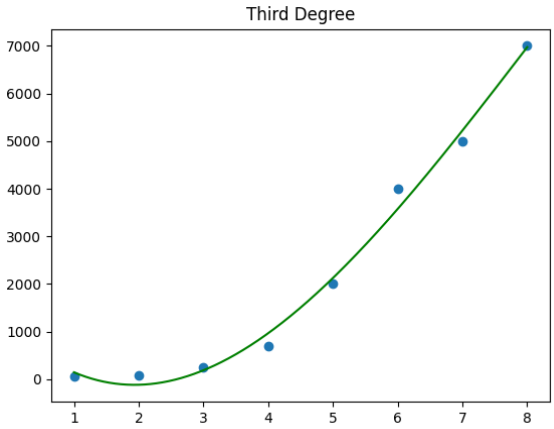
$y_4 = 30.87 * 0.7 = 21.609$

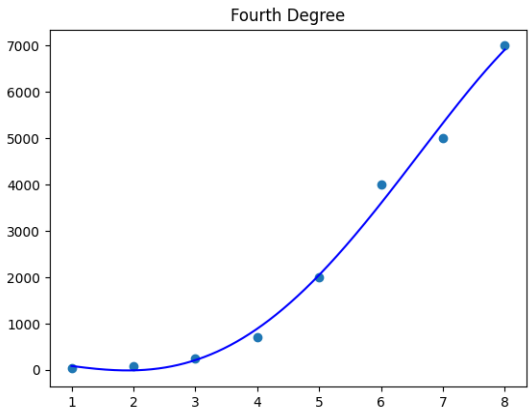
$y_5 = 21.609 * 0.7 = 15.1263$

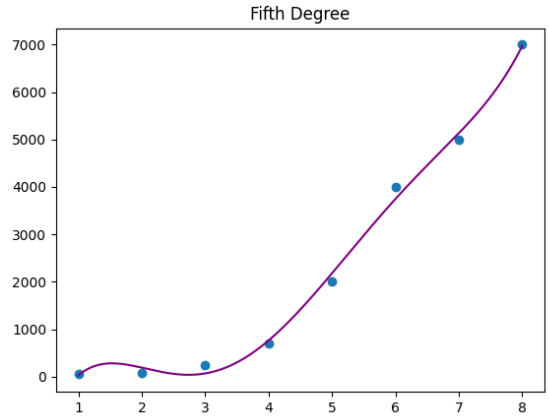
2.

a) Finding a fit with degrees 2, 3, 4, and 5:



```
===Second Degree Polynomial===
Coef: 1 = 167.7559469373667
Coef: 2 = -488.410657616929
Coef: 3 = 304.94630253712586
R squared: 0.9896379445511405
```

```
===Third Degree Polynomial===
Coef: 1 = -15.38636363639926
Coef: 2 = 375.4718614726766
Coef: 3 = -1280.8084415612302
Coef: 4 = 1066.5714285737429
R squared: 0.9993654575390704
```

```
===Fourth Degree Polynomial===
Coef: 1 = -4.864583333346076
Coef: 2 = 72.17613636379089
Coef: 3 = -153.37784090942517
Coef: 4 = -67.44237013001731
Coef: 5 = 240.98214285765388
R squared: 0.9999350482795496
```

```
===Fifth Degree Polynomial===
Coef: 1 = 6.000961538472562
Coef: 2 = -139.8862179490284
Coef: 3 = 1192.3556235457547
Coef: 4 = -4339.048513995636
Coef: 5 = 6744.792016332053
Coef: 6 = -3436.7500000077316
R squared: 0.9999995105686306
```

The following fits and their polynomial coefficients are given. The functions implemented also print out the $r^2$ values of each fit.

b)

Second Degree $r^2$ : 0.989638

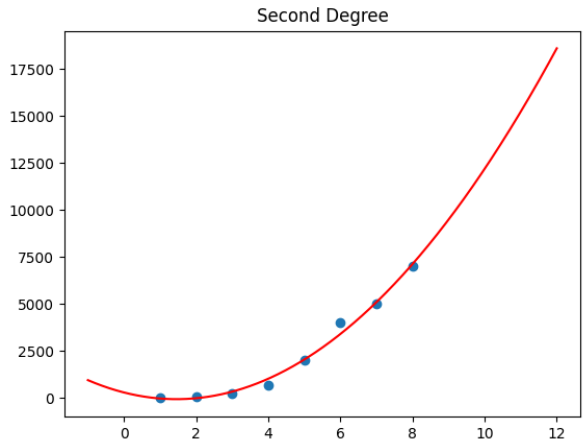
Third Degree $r^2$ : 0.999365

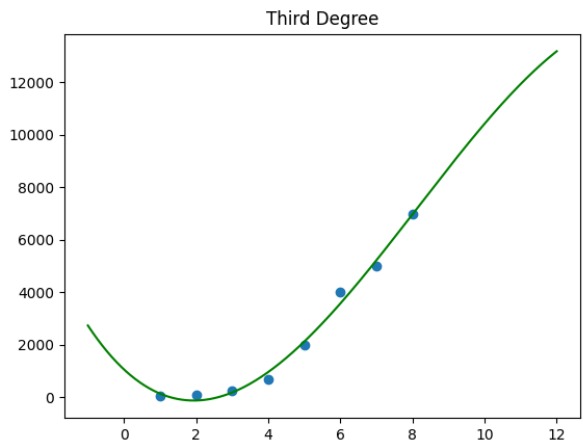
Fourth Degree $r^2$ : 0.999935

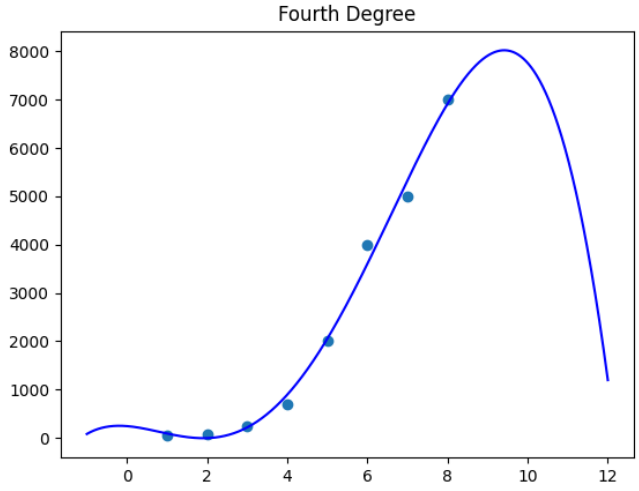
Fifth Degree $r^2$ : 0.999999

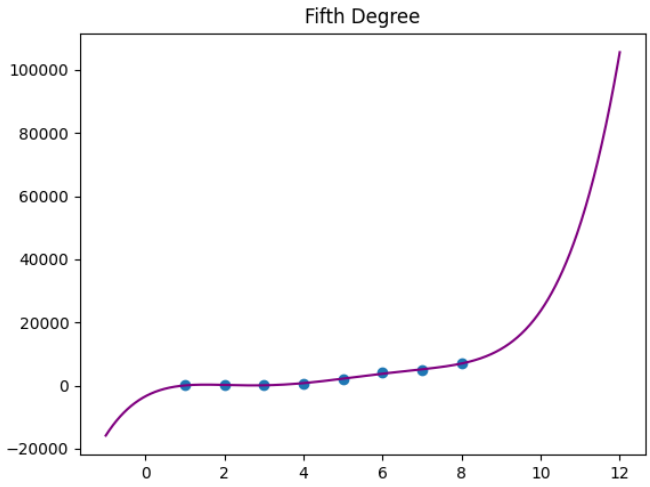
Just from these values, we can see that each progressive polynomial degree increases the accuracy of the model. Just from this metric we would say that the fifth degree polynomial is the best fit.

c)






To determine the best fit for the given model, we can look at the properties of the fits to understand how well they are actually handling the data.

The second degree model is not accurate and misses many points and the shape is also very basic. The third degree model misses just about as many points as the second degree model and its shape is slightly different. This shows that these first models are probably a little too basic. The fourth degree model begins hitting more points while maintaining a shape that looks relevant to the given data. The fifth degree model hits the most points, meaning it is technically the most accurate for the given data in the given range, however it has a suspicious bump between points 1 and 2 and may be 'too' tuned to the data.

When checking for how the models predict, their faults become more obvious. The first 3 models have an issue with values lower than 1 increasing. This would mean that there were more workers at the start then they left at year 1 only to grow again later. However this can be ignored because you wouldn't need to predict the past, and there aren't going to be negative years anyay.

The second degree model overpredicts the growth rate of the factory. Based on the trends, it seems a little high to predict the growth to be over 5000 in 2 years. The third degree model gives a more conservative prediction and begins flattening near the top. This is possible if the factory becomes oversaturated or has a stable workforce. The fourth degree model begins predicting worker numbers to drop which is unlikely based on the given data. Finally, the fifth degree model skyrockets its prediction out of nowhere, showing it was way too fine tuned to the data to be useful.

With these observations, the second degree, fourth degree, and fifth degree could be discarded in favor of the third degree model. It isn't a perfect fit, but it gives the most consistent predictions and isn't too fine tuned on the given data. It also has a R^2 value around 0.999 which is plenty accurate for a model.

3.

Factors to consider for investing in the fast food chain:

- Cost of utilities (water, electricity, garbage)
- Cost of paying workers
- Hours open
- Location
- Deals offered to college students
- Cost of buying the food
- Profits made

Important Variables:

- Cost  of utilities
- Cost of food
- Cost of paying workers
- Profits made

If the only thing you want to track is your profits for each period, you will only need to worry about the big picture and its metrics. So simply comparing the costs of operation to the profits of business will be the main goal. The cost of utilities will most likely be constant considering normal operations will even out from day to day. The cost of food will also most likely be a constant. Food chains learn how much product they will need each day and unless there is a sudden change in business, it won't fluctuate too much.

Tossed Variables:

- Location
- Hours Open
- Deals for college students

These variables can be good ways to change up profits or affect the business but aren't necessary for the big picture on estimating your profits. You will make more money from having longer hours, but you will also need to pay your workers more so they cancel each other out. The location is important but it's not something you can just change easily and it is hard to analyze how it is affecting your profits. And deals come and go, making them difficult to calculate.

4.

```
D and E reliability: 0.997%     F and G reliability: 0.994%
System reliability without component H: 0.905%
The reliability of component H must be 0.884%
```

```
def problem4():
    # components ABC are series
    A = 0.98
    B = 0.97
    C = 0.96
    # components DE are parallel
    D = 0.95
    E = 0.94
    # components FG are parallel
    F = 0.93
    G = 0.92

    # first compute the parallel components
    par_1 = 1 - ((1 - D) * (1 - E))
    par_2 = 1 - ((1 - F) * (1 - G))
    print("D and E reliability: {:.3f}%\tF and G reliability:
{:.3f}%".format(par_1, par_2))

    # compute the whole system with the series components
    complete_system = A * B * C * par_1 * par_2
    print("System reliability without component H:
{:.3f}%".format(complete_system))

    # Now compute the mystery component H
    H = 0.8 / complete_system
    print("The reliability of component H must be {:.3f}%".format(H))
```

In order to solve this task of reliability, I made a very simple python script to calculate the various components and then the mystery component H.

5.

```
def problem5():
    # this is the monty hall problem. The statistics for the problem basically
    # come down to if you switch you have a 2/3 chance of winning and if you
    # stay with your original guess you have a 1/3 chance of winning. We know
    # if contestants 1 - 5 are selected it is a 1/3 chance the prize was chosen.
    # if contestant 6 was chosen, the chance is 2/3.

    # Constant variable for number of simulations so if you want to change the
    # number of times it runs, just change this one variable
    SIM_NUM = 1000
    attempts = []
    results = []
    prize_counter = 0

    # the 2's and 1's being appended to the list act as the ratio of the chance
    # so 2 means 2 out of 3 chance and 1 means 1 out of 3 chance
    for i in range(SIM_NUM):
        if rnd.randint(1, 6) == 6:
            attempts.append(2)
        else:
            attempts.append(1)

    # So we randomly make a number from 1 to 3. This is equivalent to choosing
    # one of the 3 doors to have the prize. We then see if the randomly generated
    # integer is greater than the value at the attempt index. 2 and 3 will be
greater
    # than 1 66% of the time, which is the 1/3 chance. The other option 2 will be
    # greater than the random integer only 33% of the time, which is the 2/3
chance.
    # So finally based on if the correct door was chosen or not, we assign a
    # Binary true or false to a list of results
    for j in attempts:
        if rnd.randint(1, 3) > j:
            results.append(False)
        else:
            results.append(True)

    # count up all the correct guesses
```

```
    for r in results:
        if r == True:
            prize_counter += 1

    final_odds = prize_counter / SIM_NUM
    print("The final odds are {}".format(final_odds))
```

```
C:\Users\Tommy\GitHub\3310
The final odds are 0.394

C:\Users\Tommy\GitHub\3310
The final odds are 0.361

C:\Users\Tommy\GitHub\3310
The final odds are 0.378

C:\Users\Tommy\GitHub\3310
The final odds are 0.371

C:\Users\Tommy\GitHub\3310
The final odds are 0.424
```

I ran the program 5 times because it is super quick and wanted to see how the model was averaging out. Just from 5 runs it looks as though the average is going to be somewhere in the high 30's like 37% or 38%.

I know that the model could be simplified so that the probability itself is just appended to a list and then all probabilities are compounded 1000 times to get the final probability. This would get rid of the need for then creating another random variable that acts as the "prize behind the door", but I feel as though actually choosing a door this way instead of putting the base probability into the calculation makes it more realistic. After all, the prize isn't always behind door number 3.