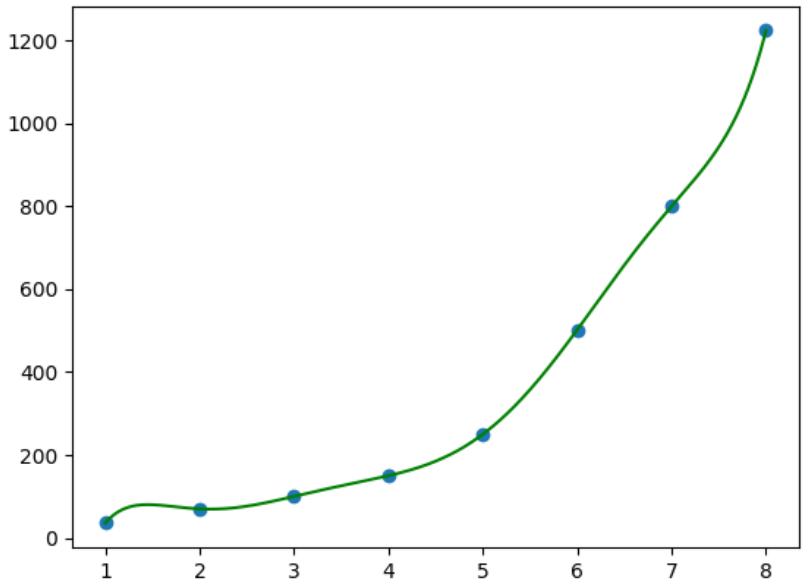
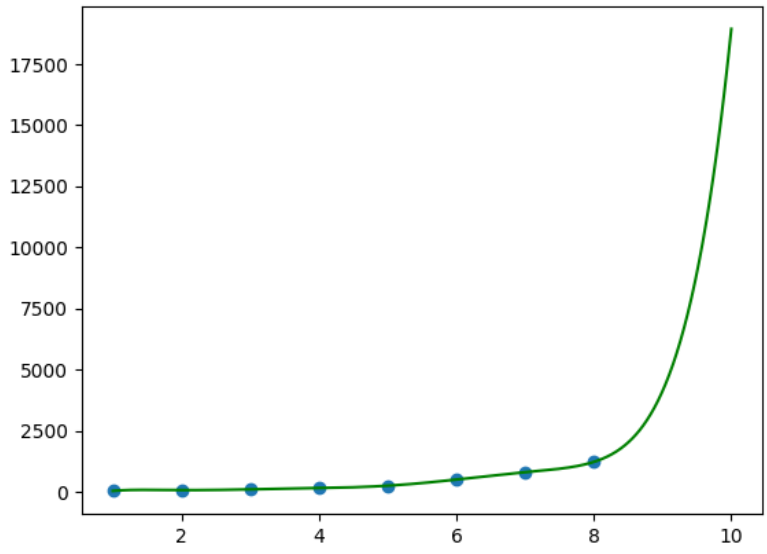
Thomas Neyman
MATH 3310
February 15 2022
Homework 2

Question 1. )  (Driver code last page)
Scatter Plots:



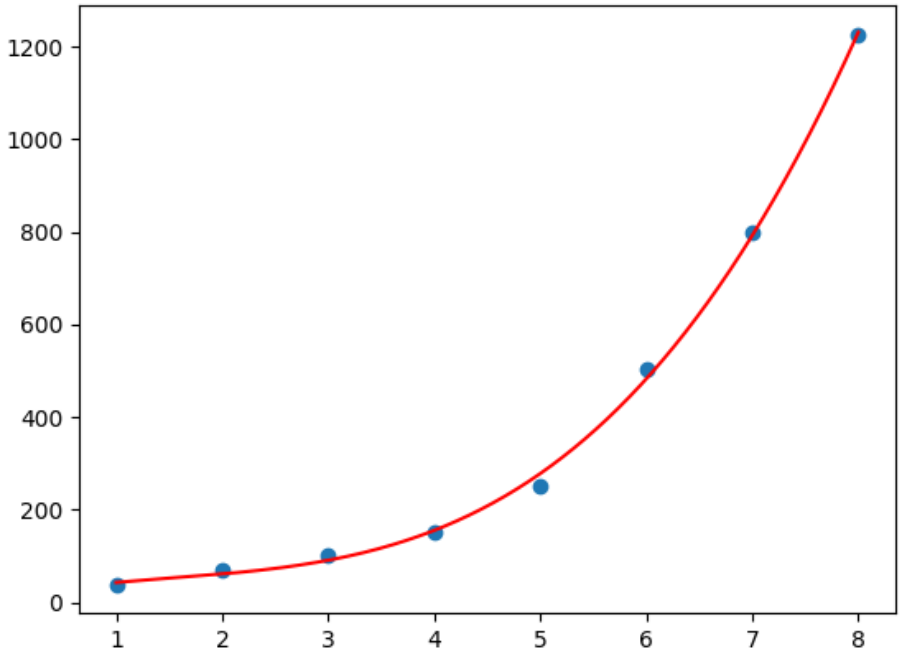
Code:

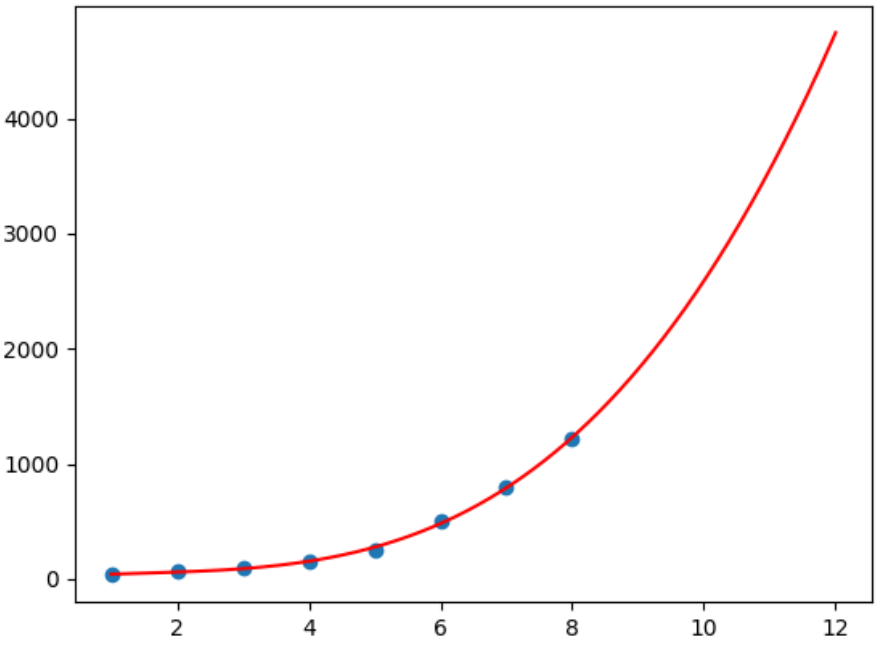
```
# Code generously provided by professor
# This code is simply used to get the y value in a form that can be plotted
def PolyCoefficients(x, coeffs):
    o = len(coeffs)
    y = 0
    for i in range(o):
        # This line is changed so that it sorts in descending order instead of
ascending
        y += coeffs[o - i - 1]*x**i
    return y

def problem_1():
    c = np.array([1, 2, 3, 4, 5, 6, 7, 8])
    y = np.array([36, 70, 100, 150, 250, 502, 800, 1224])
    w = lagrange(c, y)
    plt.scatter(c, y)
    # use linspace to add 1000 extra points, showing more of a curve
    c = np.linspace(1, 8, 1000)
    # for extended results use code below instead
    #c = np.linspace(1, 10, 1000)
    plt.plot(c, PolyCoefficients(c, ply(w).coef), 'g-')
    plt.show()
```

The main scatter plot shows results that are consistent with what is expected. The values continuously rise with only tiny variations of bumps between x = 1 and x = 8. This is a good indication that the fit is good and represents the data well. The main trend of increasing values is maintained throughout the graph's representation.

However, only increasing the estimation by 2 more points shows the values increase by a very substantial margin, going from y = 1200 to y = 17500. This could be a sign that the model is overpredicting too high after the value 8, or it could be an example where values start smaller then suddenly compound into something much larger exponentially. A model that could be similar to this is tracking a growth in population. But we will assume this is too drastic of a change for a model like this.

Question 2. )
Scatter Plots:



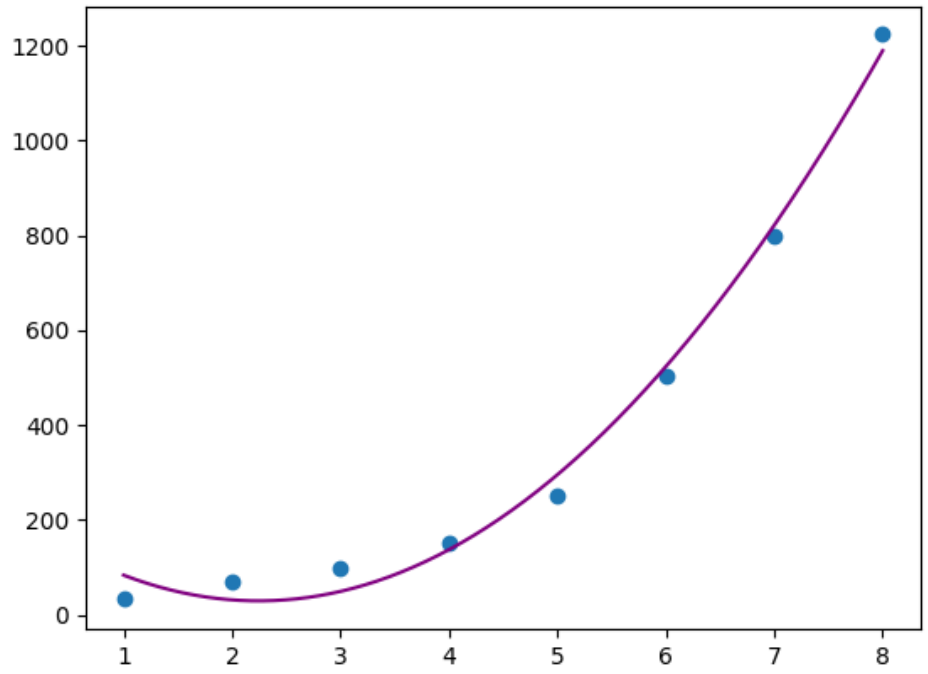
Code:

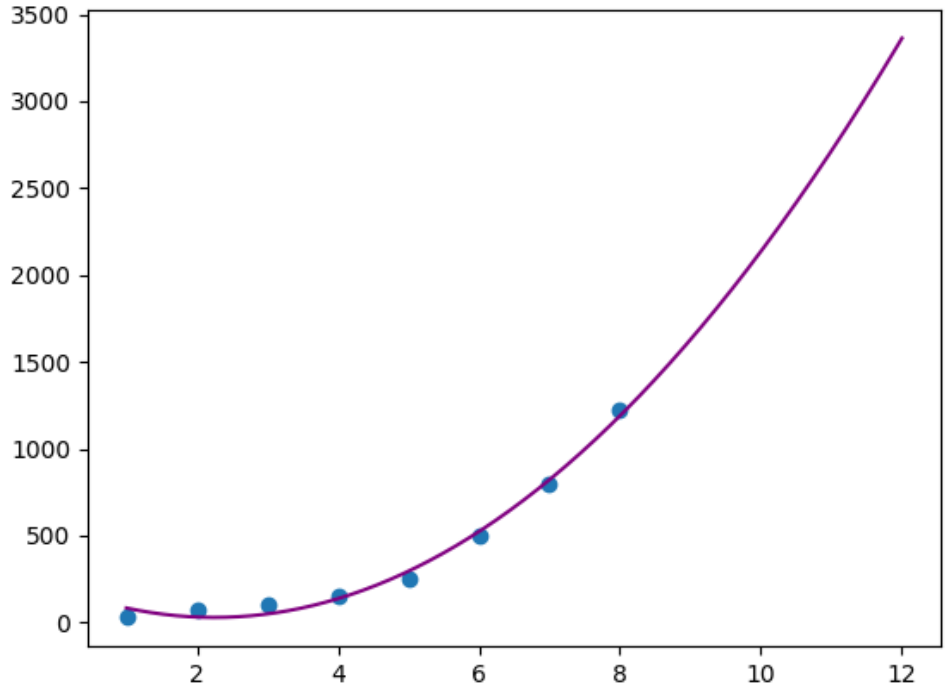
```
def third_degree_poly(x, a, b, c, d):
    return a*x**3+b*x**2+c*x+d

def problem_2():
    c = np.array([1, 2, 3, 4, 5, 6, 7, 8])
    y = np.array([36, 70, 100, 150, 250, 502, 800, 1224])
    plt.scatter(c, y)
    popt, pcov = curve_fit(third_degree_poly, c, y)
    c = np.linspace(1, 8, 1000)
    plt.plot(c, third_degree_poly(c, *popt), 'r-')
    plt.show()
    plt.figure()
    c = np.array([1, 2, 3, 4, 5, 6, 7, 8])
    # This prints the values of our coefficients a, b, c, and d
    print(popt)
    print(sum((y - third_degree_poly(c, *popt))**3))
```

Just by using common sense, we can see that a polynomial model to the third degree works much nicer than the previous Lagrange model. The first model had values that went through every point and bended and curved in order to fit the model as nicely as possible, but this caused it to become almost too fit to its specific data points and became inaccurate when predicting higher values.

This new model shows a much more realistic and probable estimation of values after the value 8. Instead of jumping up to around 17500 at x = 10, the third degree polynomial model only goes to around 2500 which is much more reasonable when the value at x = 8 was only 1200. Because of this, the second model is a much better fit for the set of data given. It isn't as precise in specific values, but it is much better overall.

Question 3. )
Scatter Plots:



Code:

```
def second_degree_poly(x, a, b, c):
    return a*x**2+b*x+c

def problem_3():
    c = np.array([1, 2, 3, 4, 5, 6, 7, 8])
    y = np.array([36, 70, 100, 150, 250, 502, 800, 1224])
    plt.scatter(c, y)
    popt, pcov = curve_fit(second_degree_poly, c, y)
    c = np.linspace(1, 12, 1000)
    plt.plot(c, second_degree_poly(c, *popt), 'purple')
    plt.show()
    plt.figure()
    c = np.array([1, 2, 3, 4, 5, 6, 7, 8])
    # This prints the values of our coefficients a, b, and c
    print(popt)
    print(sum((y - second_degree_poly(c, *popt))**2))
```

Here we see the model actually start to become worse. This shows that the degree used is a little too basic and not concise or complex enough to properly and accurately represent the data given. The values plotted are off by minor to significant amounts on most of the points and estimations for values beyond x = 8 seem to be under-estimated. So the last model is underguessing and not very accurate with the points it plots with the given data.

So the best model ends up being the second model for a few reasons. The first model was too specific. It was extremely accurate with points that were given in data but was not good at estimating future values which would be necessary for a model. The third model was too basic and not complex enough to accurately plot and estimate values for the model. This means that when the model was somewhere in between too complex and too basic, it worked the best.

Driver Code:

```
from turtle import right
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from numpy.polynomial.polynomial import Polynomial as ply
from scipy.interpolate import lagrange

def main():
    problem_1()
    problem_2()
    problem_3()

# All the code here

if __name__ == '__main__':
    main()
```