Project 3 Writing

ECE 8540 REWRITING NORMAL EQUATIONS

September 27, 2022

Bradley Selee
Clemson University
Department of Electrical and Computer Engineering
bselee@clemson.edu

1 Introduction

This project was used to introduce us to basic modeling. In this project, we solved linear models using normal equations from given sets of data. The technique for fitting a line to a set of points can be generalized to any function consisting of a linear combination of terms.

In a very broad sense, these normal equations can be extended to critical problems like detecting a missile with radar. With these linear models, further predictions on new data can be made; but predictions was not a part of this project. For example, we could predict the next location where the missile is headed.

This general equation can be solved for any basis equation, linear or non-linear, as long as it is linear in the unknowns. The residual error e_i is defined and so is the chi-squared metric which is defined as the difference between the best fitting solution and the collective dataset.

This project will solve for this general equation and use differential equations to minimize the chi-squared distance. Furthermore, in project 1 we will use trial and error to form multiple basis equations and decide which one fits the data the best.

2 Methods and Materials

For this project, my implementation was created solely using Python. The only libraries used were NumPy for matrix operations, Pandas to load the data in part 3 into a dataframe, and Matplotlib to plot all the graphs. The code was developed in Windows Subsystem for Linux (WSL) using Visual Studio Code as the text editor. My code implementation is stored on GitHub in a private repository.

2.1 Code Design and Implementation

The main idea behind normal equations is to create 2 matrices, A and b, then use the normal equations to solve for x. This will return a matrix of all the unknown variables. This implementation passes a set a of data into a function, forms matrix A and b, and solves for matrix x. With the unknown variables solved for, they are plugged into the basis equations and plotted.

3 Results

There were three sets of data points given for this project. The first two use a linear function as the basis equation and the third dataset basis equation was created uses

trial and error. Below shows the matrices created, the unknowns solved for, the basis equations and the plots of the final equation.

3.1 Part 1

Part 1 used five data points, matrix A and b are shown below

$$A = \begin{bmatrix} 5 & 1 \\ 6 & 1 \\ 7 & 1 \\ 8 & 1 \\ 9 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 5 \end{bmatrix}$$

The unknowns are a and b and the final equation were found to be

$$a = 1.0$$

$$b = -4.6$$

$$y = x - 4.6$$

The plot of the data points and line are shown below

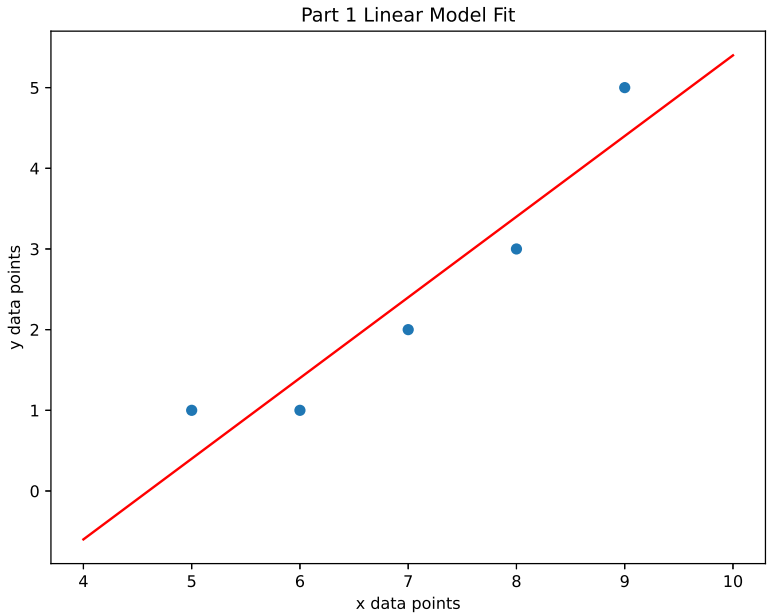

Figure 3.1: Part 1 graph

3.2 Part 2

Part 2 used six data points, the same five points from part 1 and an additional point. This additional point caused the final line to shift slightly to account for this data point. This new data point seems to be an outlier which means it will skew the final model slightly. Matrix A and b are shown below

$$A = \begin{bmatrix} 5 & 1 \\ 6 & 1 \\ 7 & 1 \\ 8 & 1 \\ 9 & 1 \\ 8 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 5 \\ 14 \end{bmatrix}$$

The unknowns are a and b and the final equation were found to be

$$a = 1.815$$

$$b = -8.680$$

$$y = 1.815x - 8.680$$

The plot of the data points and line are shown below

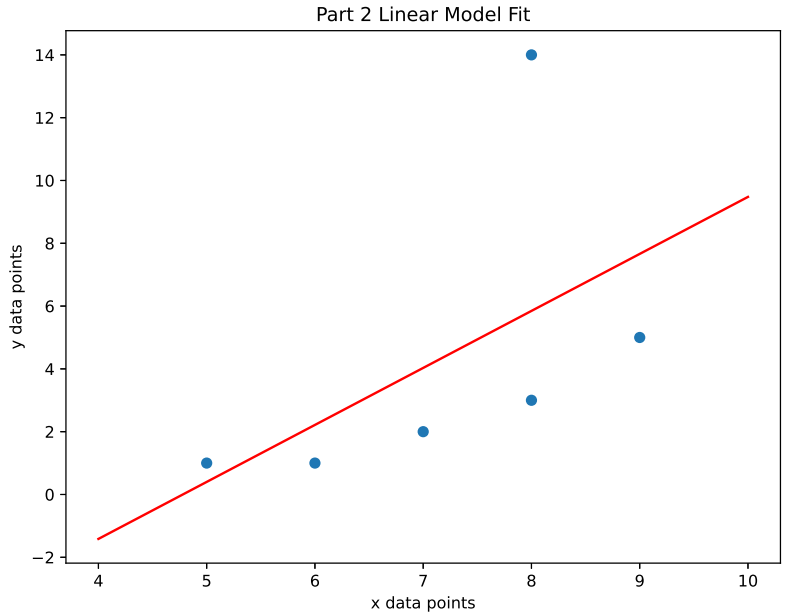

Figure 3.2: Part 2 graph

3.3 Part 3

Part 3 used was large dataset describing the metrics of meals different people ate. The data used for analysis were the number of bites taken and the amount of kilocalories per bite. Initially, the data was plotted as a scatter plot to decide on a basis function. The final basis function used was a modification of the inverse function

$$y = \frac{-1}{ax + 1} + b$$

Because there are a lot of data points, matrix A and b will not be shown. The unknowns a and b and the final equation were found to be

$$a = -0.177$$

$$b = 23.442$$

$$y = \frac{-1}{-0.177x + 1} + 23.442$$

The plot of the data points and line are shown below

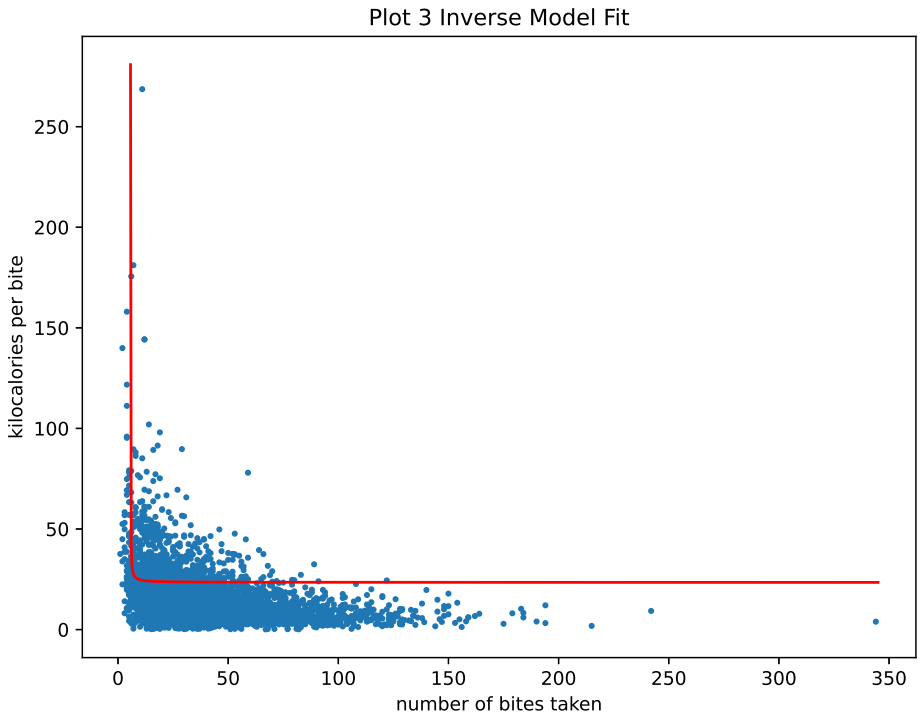

Figure 3.2: Part 3 graph

4 Conclusion

Overall, I think this project was successful. My implementation creates the necessary matrices from the data provided. It solves the normal equations for the desired variables and finally plots the results. This method of normal equations only works if the basis equation is linear in the unknowns, if it is not then a different method will need to be applied. The inverse solution in part 3 will most likely hold if a few additional data points are added, however, if many new data points are added that do not conform to the other data points, this model will probably fail. If I did this project again I would have played around with the basis equation in part 3 a little longer.

5 Appendix

The following appendix is the Python code for this lab.

```
"""lab1.py

Bradley Selee
ECE 8540
Project 1
Due: 09/08/2022
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


def solve_normal_equation(data_points):
    """Function to solve the normal equations of a line from a list of data
        points
    It solves a & b in y=ax+b

    Args:
        data_points: A list of data points where each data point is a tuple
    """
    n = len(data_points)
    m = 2 # equation of a line -> 2 unknowns (y = ax + b)
    A_1 = np.array(data_points)[:,0]
    print(A_1)
    A_1 = np.expand_dims(A_1, axis=1)
    A = np.append(A_1, np.ones((n, 1)), axis=1) # A -> nxm where n = number of
        data points and m = number of unknown variables

    b = np.array(data_points)[:,1]
    b = np.expand_dims(b, axis=1)
```

```
    print(f'Matrix A: \n{A}')
    print(f'Matrix b: \n{b}')
    x = np.dot(np.linalg.inv(np.dot(A.T, A)), np.dot(A.T, b))
    print(f'Solution x: \n{x}')

    # x returns in the form [a, b]
    return x


def plot_solution(data, x, fig_name=None):
    """Plot a scatter plot of the original data and a line plot
    on top of the equation of line using the solution to the normal equation

    Args:
        data: a list of tuples containing the data
        x: the solution to the normal equation
    """
    fig = plt.figure(figsize=(8,6))

    data_x, data_y = zip(*data)
    line_x = np.linspace(min(data_x)-1,max(data_x)+1, 100)
    line_y = x[0][0]*line_x + x[1][0]
    plt.scatter(data_x, data_y)
    plt.plot(line_x, line_y, color='red')
    if fig_name:
        fig.savefig(fig_name)


def plot_inverse(data, x, fig_name=None): #x, show=True, fig_name=None):
    """Plot a scatter plot of the original data and a line plot
    for part 3. This is in the from y = -1/(a*x+1) + b
    """
    fig = plt.figure(figsize=(8,6))

    data_x, data_y = zip(*data)
    a, b = x[0][0], x[1][0]
    line_x = np.linspace(5.67,max(data_x)+1, 1000)
    line_y = (-1) * (1 / (a*line_x+1)) + b
    plt.scatter(data_x, data_y, s=5)
    plt.plot(line_x, line_y, color='red')
    if fig_name:
        fig.savefig(fig_name)


def main():
    data_points = [(5, 1), (6, 1), (7, 2), (8, 3), (9,5)]
    data_points_2 = [(5, 1), (6, 1), (7, 2), (8, 3), (9,5), (8,14)]
```

```
    # Read data for part 3 into a pandas dataframe
    data_file = '83people-all-meals.txt'
    meals_df = pd.read_csv(data_file, sep=' ', header=None)

    print('Part 1:')
    x = solve_normal_equation(data_points)
    plot_solution(data_points, x, fig_name='part1.eps')

    print('\nPart 2:')
    x_2 = solve_normal_equation(data_points_2)
    plot_solution(data_points_2, x_2, fig_name='part2.eps')
    # The line shifts to try and accomodate the outlier point

    # Create 5th column in dataframe that is kilocalories/num_bites
    meals_df[4] = meals_df[3] / meals_df[2]
    # Package data into list of tuples so it is in the form of 'data_points'
        variable
    # This way it can be passed to the function 'solve_normal_equations()'
    meals_tuple = list(zip(meals_df[2], meals_df[4]))
    # Need to formulate equation y = -1/(a*x+1) + b
    print('\nPart 3')
    #plot_part3(meals_df[2], meals_df[4])
    x_3 = solve_normal_equation(meals_tuple, )
    plot_inverse(meals_tuple, x_3, fig_name='part3.eps')

    plt.show()


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

References

No further references were needed for this project.