Report for Bias and Variance trade off(q1.py)

• Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. A model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to a high error on training and test data.

Bias2 = (E[f_cap(x)] − f(x))^2

where f(x) represents the true value, f^(x) represents the predicted value

• Variance is the variability of a model prediction for a given data point. Again, imagine you can repeat the entire model building process multiple times. The variance is how much the predictions for a given point vary between different realizations of the model.

Variance = E [(f_cap(x) − E[f_cap(x)])^2]

where f(x) represents the true value, f^(x) represents the predicted value

• Noise is a unwanted distortion in data. Noise is anything that is spurious and extraneous to the original data, that is not intended to be present in the first place, but was introduced due to faulty capturing process.

If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand, if our model has a large number of parameters then it's going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.

Snippets:

import pickle

pickle module is for it serializes objects so they can be saved to a file, and loaded in a program again later on.

Import numpy as np

NumPy is an open source Python package for scientific computing. NumPy supports large, multidimensional arrays and matrices. NumPy is written in Python and C. NumPy arrays are faster compared to Python lists. But NumPy arrays are not flexible like Python lists, you can store only same data type in each column.

---

from sklearn.model_selection import train_test_split as tts, ShuffleSplit

Split arrays or matrices into random train and test subsets

from sklearn.linear_model import LinearRegression as lr

LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

---

from sklearn.preprocessing import PolynomialFeatures

Generate polynomial and interaction features.

Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are [1, a, b, a^2, ab, b^2].

---

import matplotlib.pyplot as plt

matplotlib. **pyplot is** a collection of command style functions that make **matplotlib** work like MATLAB. Each **pyplot** function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the **plot** with labels, etc.

---

import math

It provides access to the mathematical functions defined by the C standard.

---

import statistics

This module provides functions for calculating mathematical *statistics*

```
class linear_regression:
   def __init__(self):
      self.train_data = None
      self.train_data_x = None
      self.train_data_y = None
      self.test_data = None
      self.test_data_x = None
      self.test_data_y = None
      self.split_train_data = []          #split_train_data list
      self.bias = []                      #bias list
      self.variance = []                  #variance list
```

The above code is for storing the empty values and empty lists to variables

---

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

The above is the main code

```
def main():
        ob=linear_regression()
        ob.data_refactoring()
        ob.model_training()
        ob.plot_check(ob.bias,ob.variance)
```

The above is the code snippet for main function.

```python
def bias_variance_calculation(self, x_test, y_test, y_predicted, j):
        bias_total = 0
        variance_total = 0
        E_y_predicted = y_predicted.mean()
        for i in range(500):
            bias = (y_predicted[i]- y_test[i])**2
            bias_total += bias
        variance_total = statistics.variance(y_predicted)
        bias_total /= 500
        # if j == 0: bias_total = None; variance_total=None
        print(j,"degree :-",bias_total," | ",variance_total)
        self.bias.append(bias_total)
        self.variance.append(variance_total)
```

The above function is to calculate the bias and variance:

At first bias_total and varaince_total are intialized to 0.Then mean() is statistics module function that used to calculate average of numbers and list and the value of y_predicted.mean() is stored into E_y_predicted variable(Expected y_predicted value)

In a range of 0-500 we will calculate the (bias)^2 and we add bias value to bias_total

statistics.variance(y_predicted)-This function helps to calculate the variance from a sample of data(here y_predicted can be taken as a list or matrix) and that value is stored into variance_total variable.

print(j,"degree :-",bias_total," | ",variance_total)-this is for printing in the format <jvalue> degree:-<bias_total value> | <variance_total value>

self.bias.append(bias_total)
self.variance.append(variance_total)

The above is for appending the bias_total value to bias list and variance_total value to variance list

---

```python
def chunk_up_split(self,seq,num):
    avg = len(seq) / float(num)
    out = []
    last = 0.0
    while last < len(seq):
        out.append(seq[int(last):int(last + avg)])
        last += avg
    return out

def data_refactoring(self):
    pkl_file = open('./Q1_data/data.pkl', 'rb')
    net_data = pickle.load(pkl_file)
    self.train_data, self.test_data= tts(net_data, test_size=0.1, train_size=0.9, shuffle=True)
    self.train_data_x=self.train_data[:,0]
    self.train_data_y=self.train_data[:,1]
    self.test_data_x=self.test_data[:,0]
    self.test_data_y=self.test_data[:,1]
    self.split_train_data_x = self.chunk_up_split(self.train_data_x, 10)
    self.split_train_data_y = self.chunk_up_split(self.train_data_y, 10)
```

The above code is for data refactoring…

pkl_file = open('./Q1_data/data.pkl', 'rb')
net_data = pickle.load(pkl_file)

The process of loading a pickled file data.pkl which is in Q1_data file into a python program and stored into net_data.

self.train_data, self.test_data= tts(net_data, test_size=0.1, train_size=0.9, shuffle=True)

The data in net_data is split into test data of size 1 and train data of size 9 of num=10 and gets shuffled and stored into train_data and test_data.In the train_data the 0^{th} column consists of train data for x ,so it is stored into variable train_data_x and 1^{st} column consists of train data for y,so it is stored into variable train_data_y.Similarly in the same way for test data, 0^{th} to test_data_x and 1^{st} to test_data_y.

self.split_train_data_x = self.chunk_up_split(self.train_data_x, 10)
self.split_train_data_y = self.chunk_up_split(self.train_data_y, 10)

“chunk_up_split” splits the train data into 10 parts(Here)

The function “chunk_up_split” splits the train data into 10 parts

---

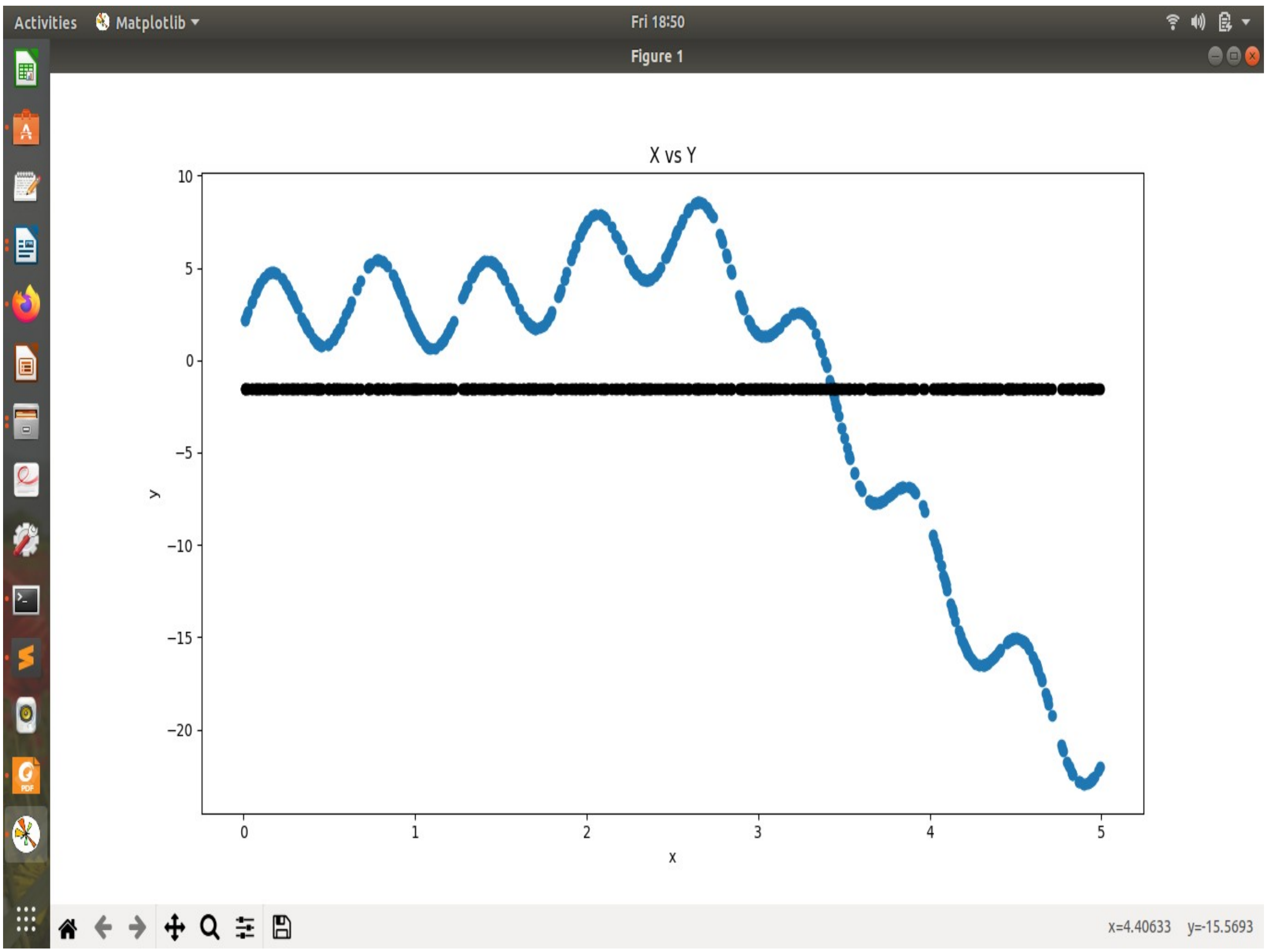
```
def plot_check(self, x, y):
        plt.plot(range(10),x)
        plt.title('number of parameters vs Bias')
        plt.xlabel('Number of parameters')
        plt.ylabel('Bias^2')
        plt.show()
        plt.plot(range(10),y)
        plt.title('number of parameters vs Variance')
        plt.xlabel('Number of parameters')
        plt.ylabel('Variance')
        plt.show()

  def model_training(self):
    for i in range(10):
        model = lr()
        poly=PolynomialFeatures(degree=i)
        x=self.split_train_data_x[i][...,np.newaxis]
        y=self.split_train_data_y[i][..., np.newaxis]
        x_=poly.fit_transform(x)
        x_test=poly.fit_transform(self.test_data_x[...,np.newaxis])
        model.fit(x_, y)
        predicted_y=model.predict(x_test)
        plt.plot(self.test_data_x[...,np.newaxis],self.test_data_y[...,np.newaxis],'o')
        plt.title('X vs Y')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.plot(self.test_data_x,predicted_y.flatten(), 'o', color='black')
```

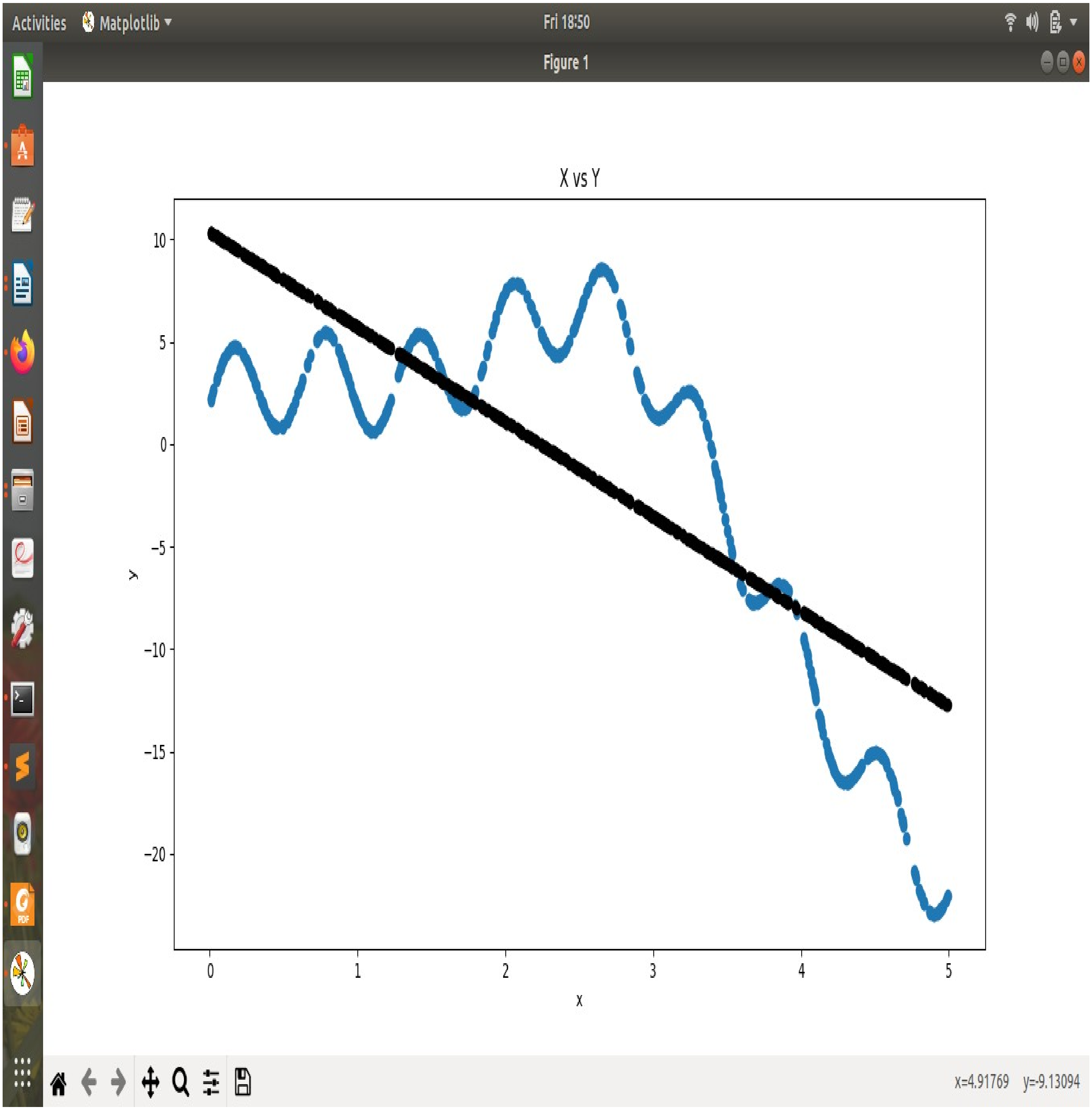
plt.show()
self.bias_variance_calculation(self.test_data_x, self.test_data_y,predicted_y.flatten(),i)

I train each of my subsets for 1 degree. Like subset 1 degree 0,2 for degree 1 etc.

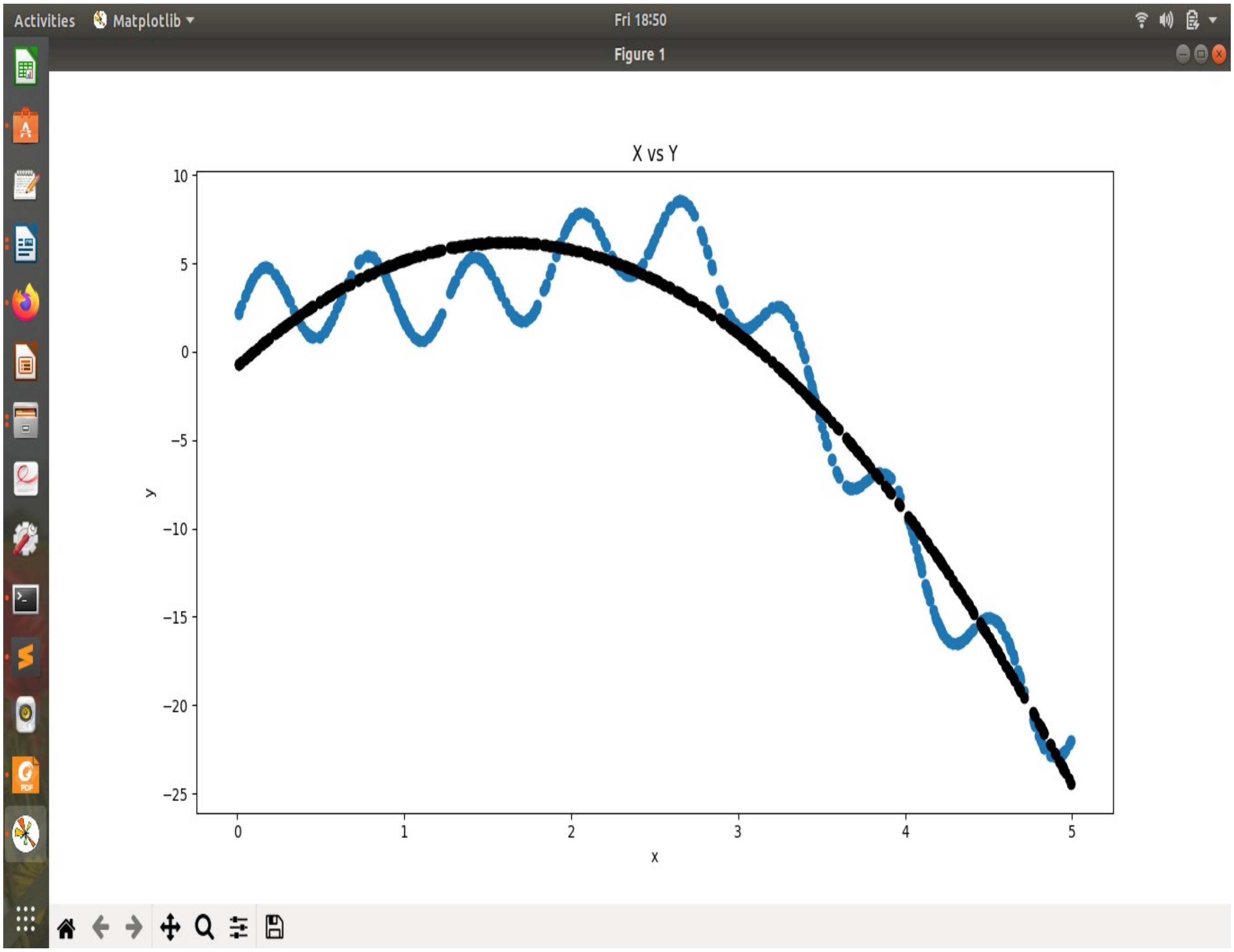
<u>The below are the images of a plottings of graph:</u>



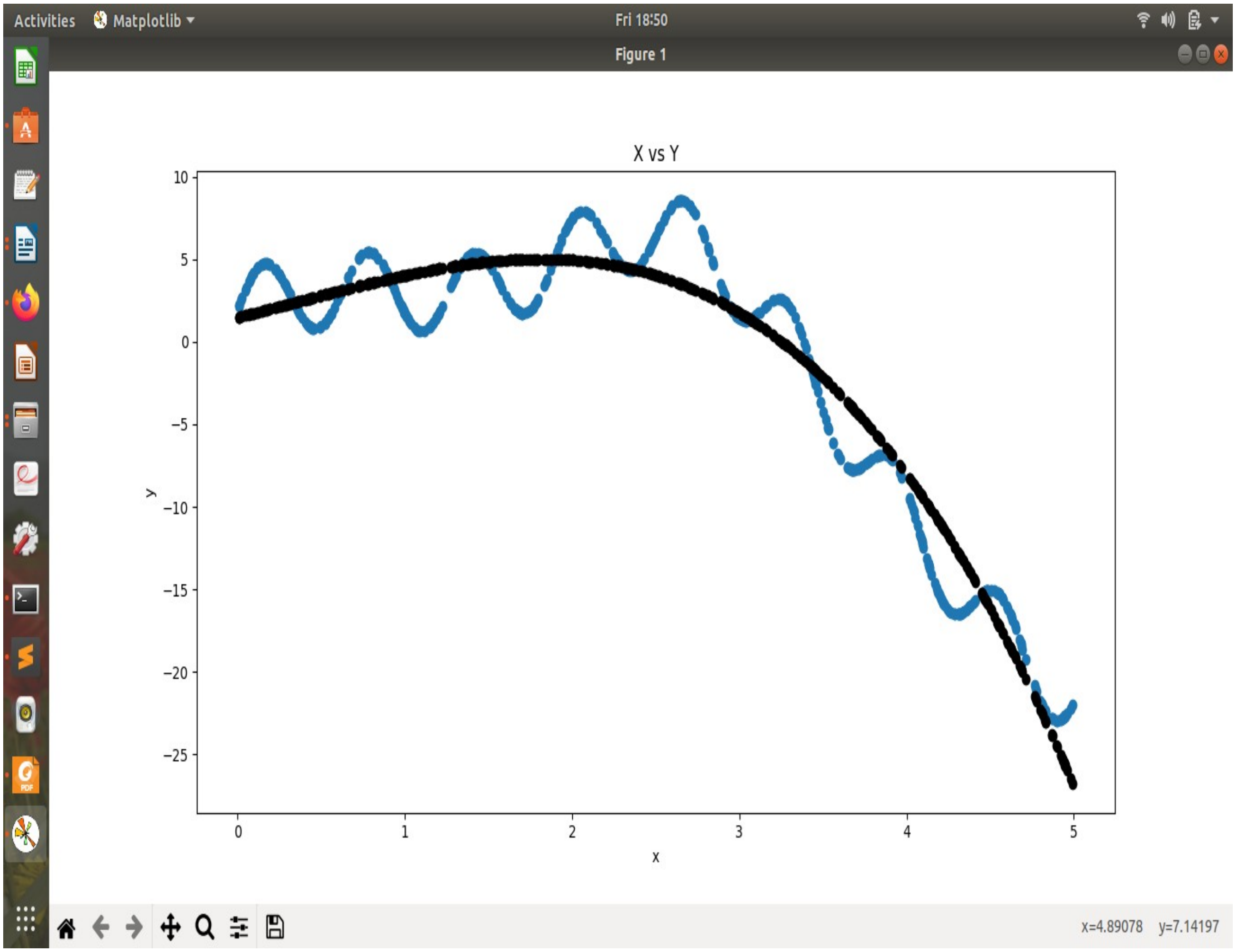
0 degree :- 81.07112267774856 | 0.0



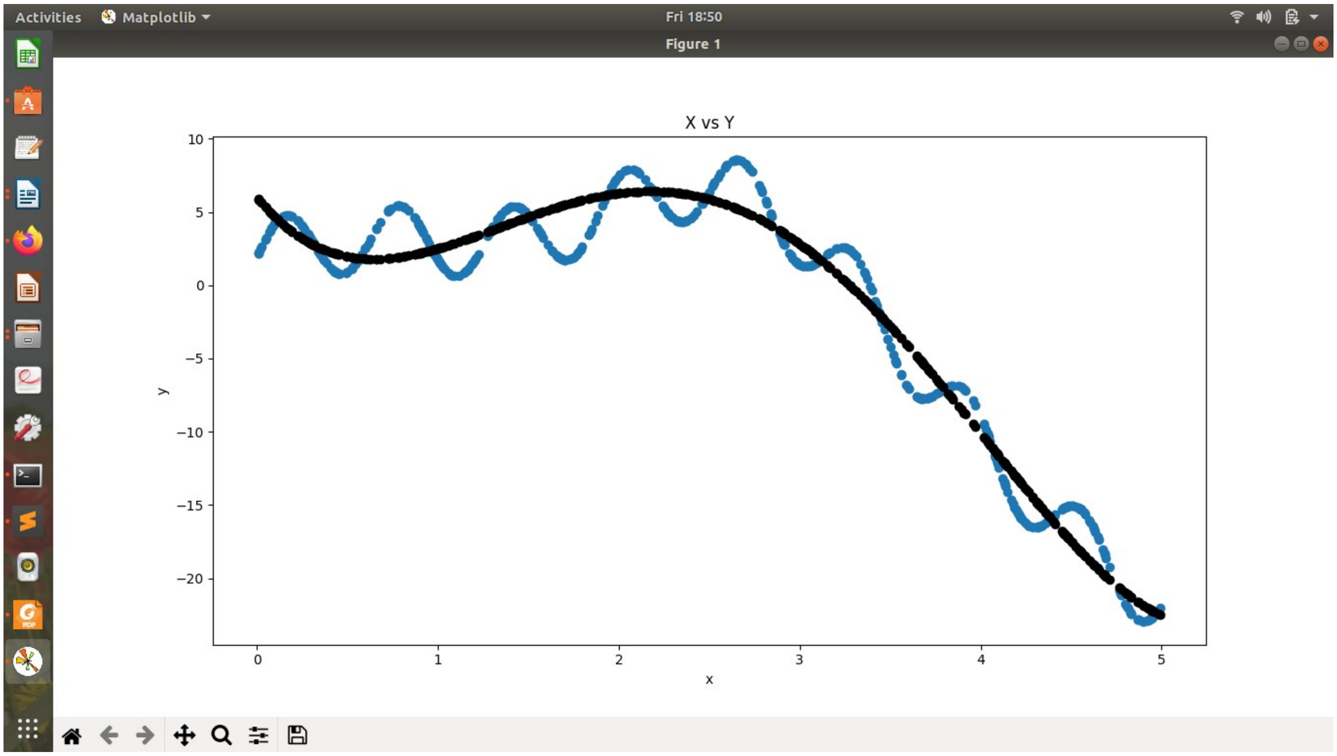
1 degree :- 30.424748508761652 | 47.298113504585984



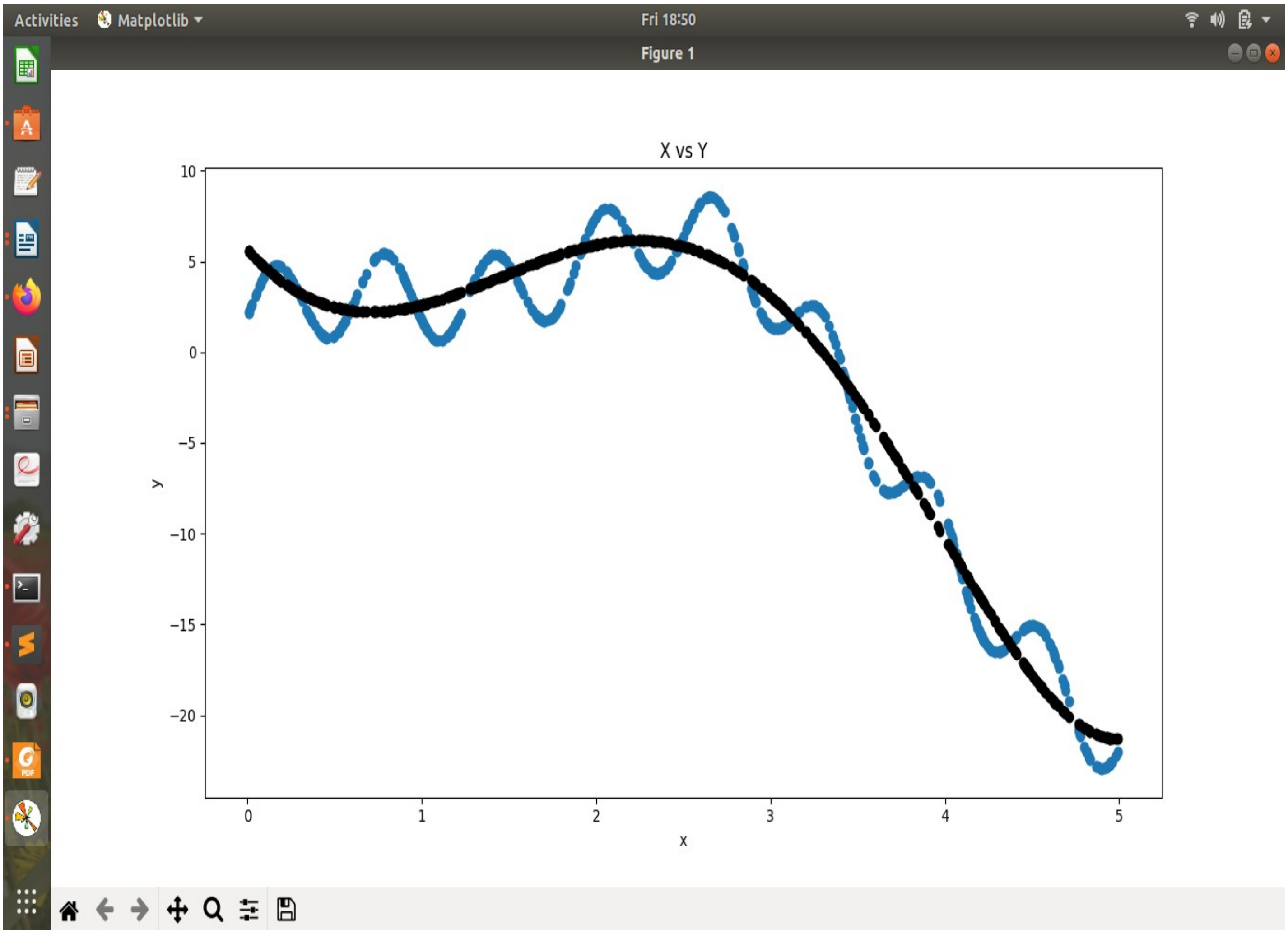
2 degree :- 5.916220296309145 | 75.80261754456109



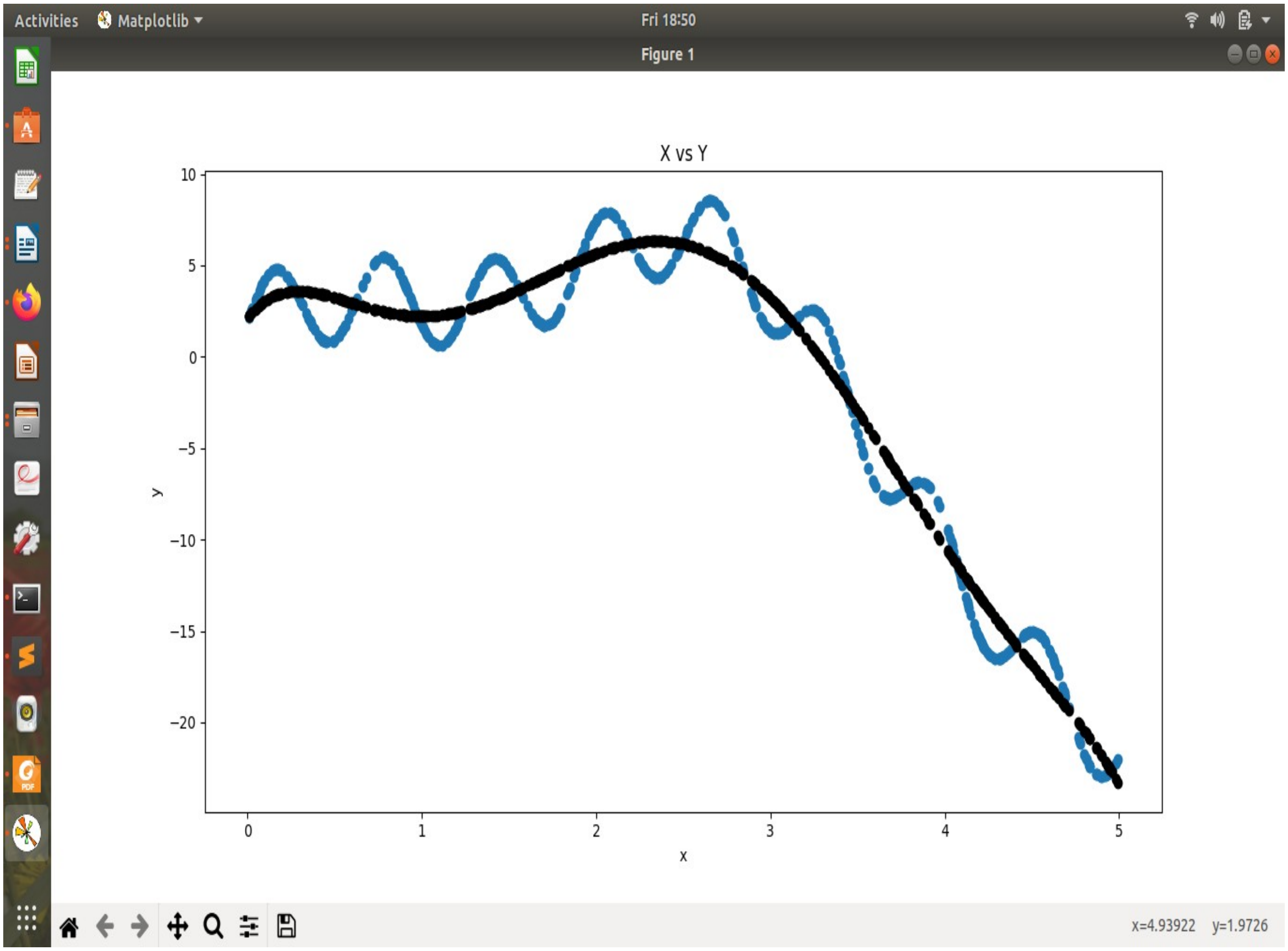
3 degree :- 5.245759710131 83 | 74.731704223917 46



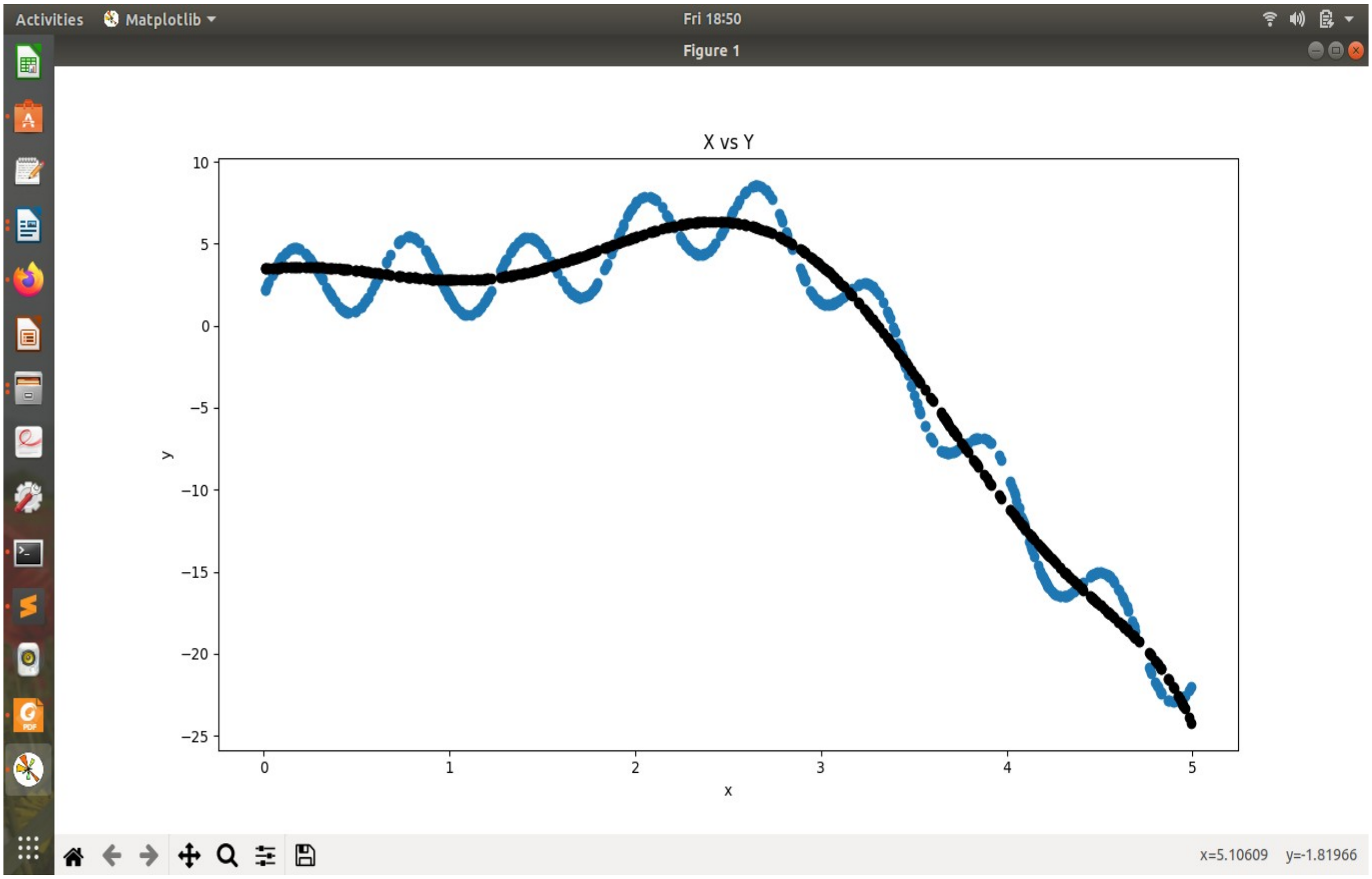
4 degree :- 3.158882058881106 | 80.1899895741217



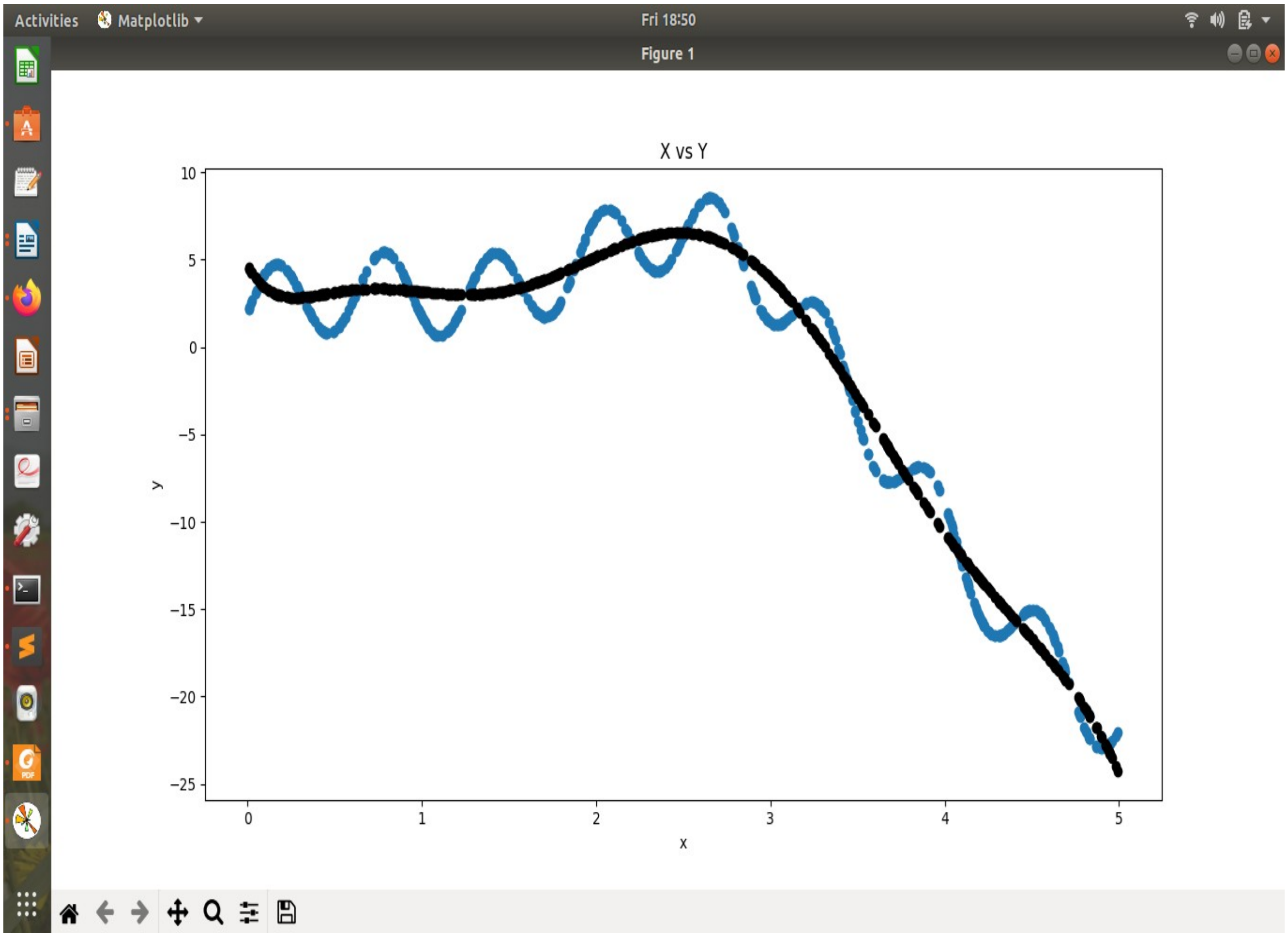
5 degree :- 3.0747203080630903 | 79.86709955971142



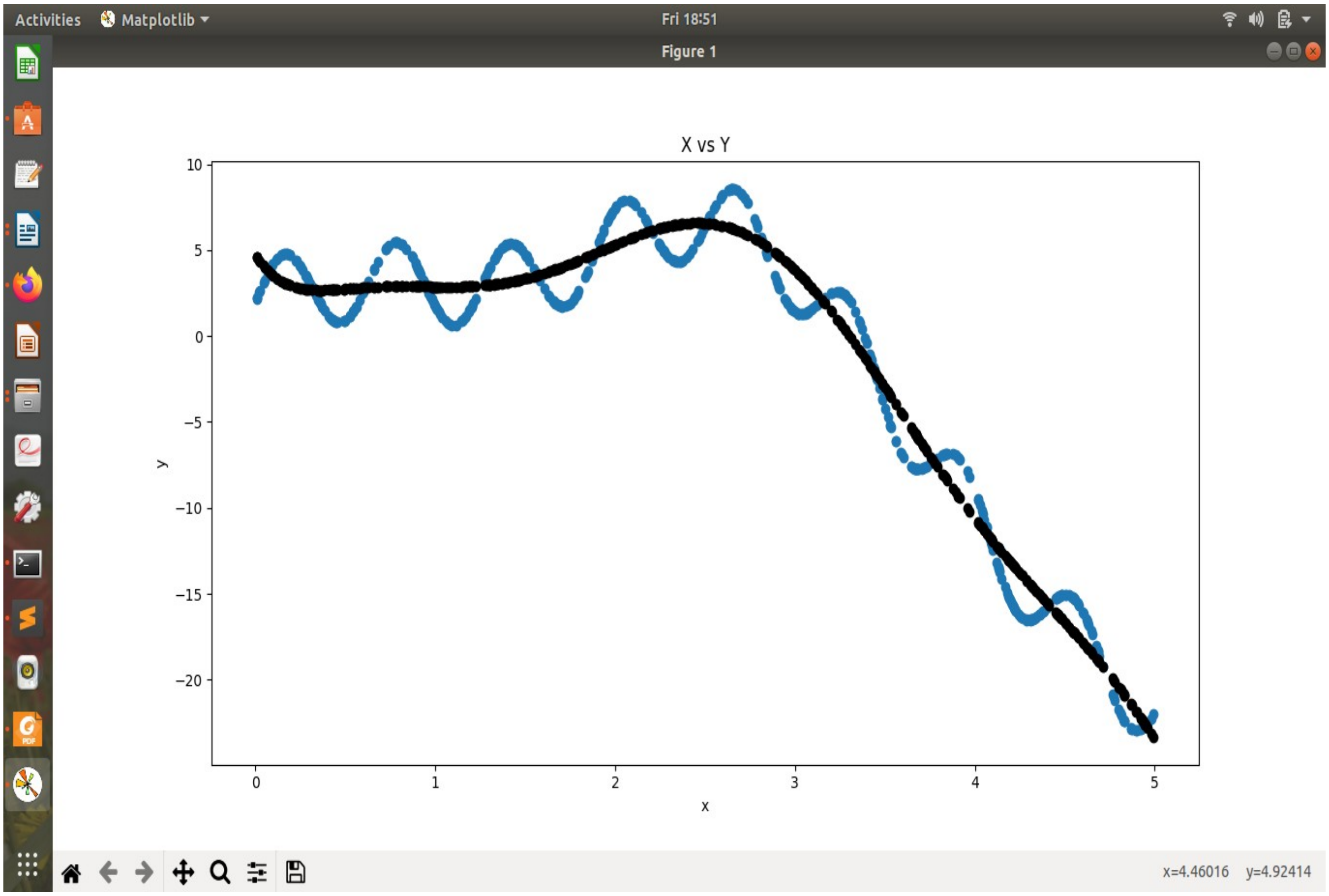
6 degree :- 2.67573171701843334 | 76.665188102558979



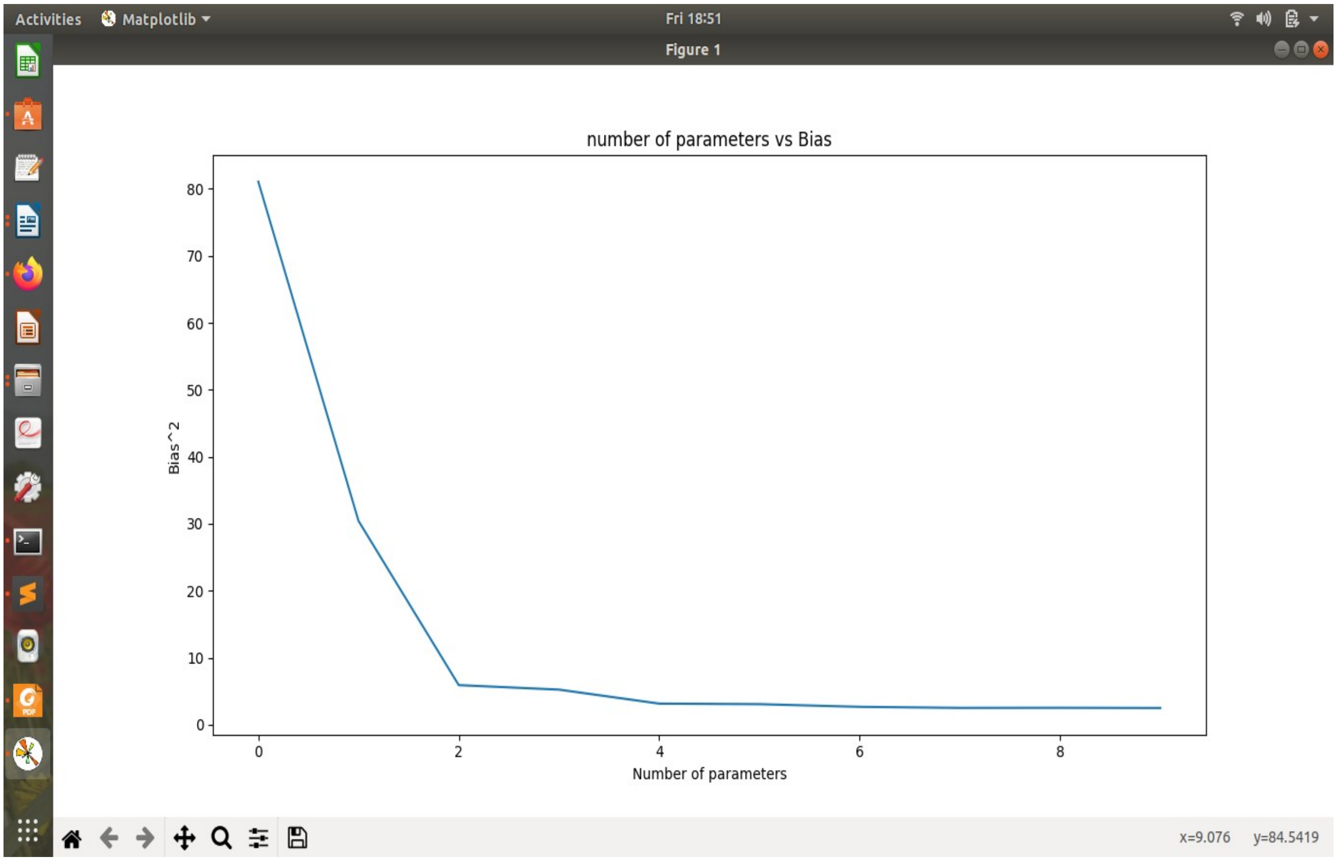
7 degree :- 2.51778451757 43014 | 80.564127947 63528



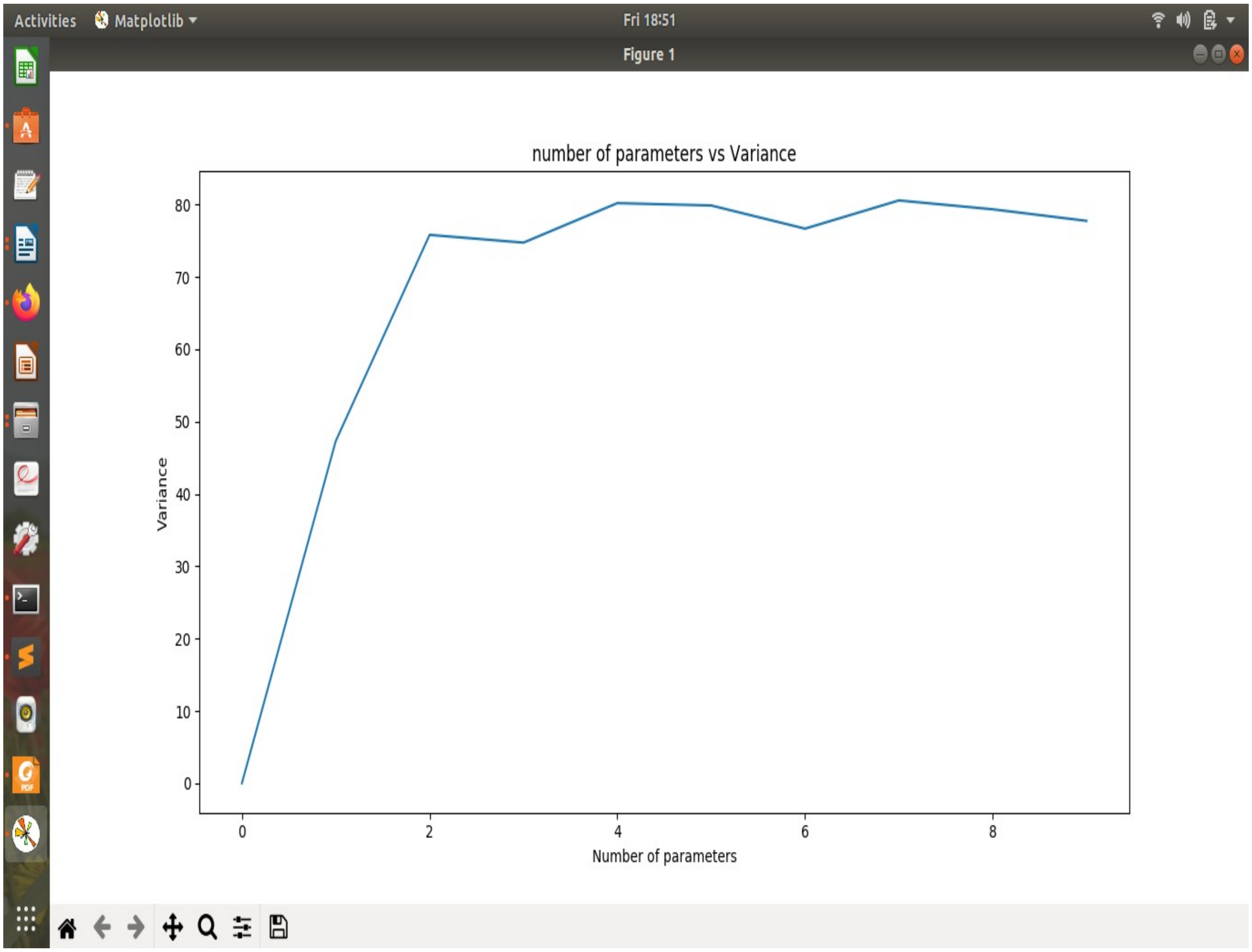
8 degree :- 2.5339920251200576 | 79.33842297654262



9 degree :- 2.5050813704947723 | 77.74044733596841



Number of parameters versus Bias



Number of parameters versus Variance

```python
def plot_check(self, x, y):
	plt.plot(range(10),x)
	plt.title('number of parameters vs Bias')
	plt.xlabel('Number of parameters')
	plt.ylabel('Bias^2')
	plt.show()
	plt.plot(range(10),y)
	plt.title('number of parameters vs Variance')
	plt.xlabel('Number of parameters')
	plt.ylabel('Variance')
	plt.show()
```

The above code snippet is for plotting the graph (***number of parameters vs Bias and number of parameters vs Variance)***

plt.title('number of parameters vs Bias')

For keeping the title of the graph as ' ***number of parameters vs Bias'***

plt.xlabel('Number of parameters')

For labeling the x-axis as ' ***Number of parameters'***

plt.ylabel('Bias^2')

*For labeling the y-axis as '****Bias^2'***

plt.show()

For executing the graph

Similarly for the graph 'number of parameters vs Variance'