Polynomial Regression and Model Selection Project Report

Lecture: CS454/554 - Introduction to Machine Learning and Artificial Neural Networks

Instructor: Prof. Ethem Alpaydın

Tuna Tuncer - S018474

1. Introduction

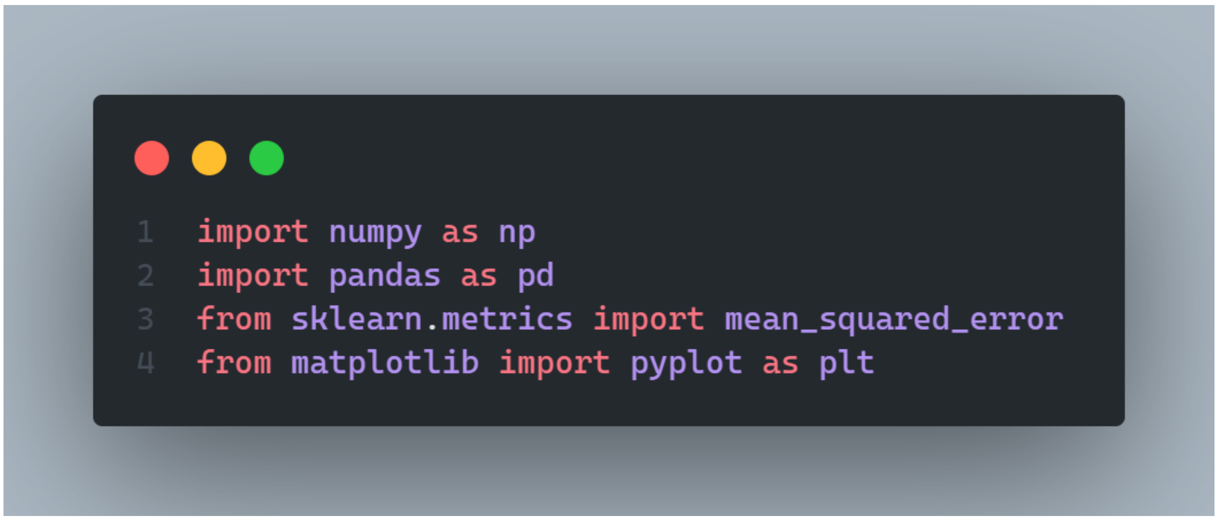
I was given one test data set with 100 instances and ten training data sets, each having 25 instances, for this assignment. I was expected to create polynomial regression models using the training data sets, and then measure and compare the mean squared errors of the models using the test data set. It was desired that the polynomial regression models I would develop had degrees ranging from 1 to 6. The data was generated using a hidden f(x) function and made more random by adding gaussian noise to it.

2. Methodology

For each x degree, 10 different models were created by using each of the 10 training data sets one by one. The mean squared errors of each of these 10 models were calculated using the test data set. Afterwards, a single value was obtained by averaging 10 different mean squared error values. When this process was repeated for each degree, a total of 6 different average mean square error values were obtained. Thus, it has been empirically found that the models with which degree explain the data more successfully.

3. Implementation Details

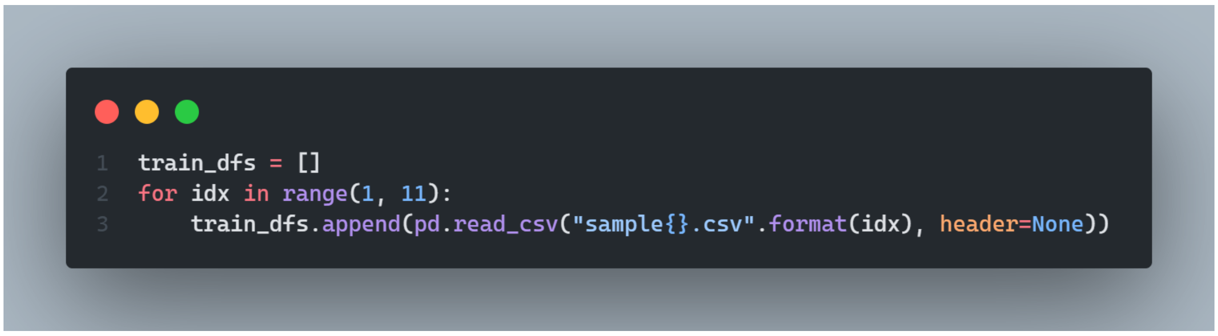
Python is used as the programming language for this assignment. In addition, Jupyter Notebook, which is widely used for machine learning related tasks, is also used as an editor.



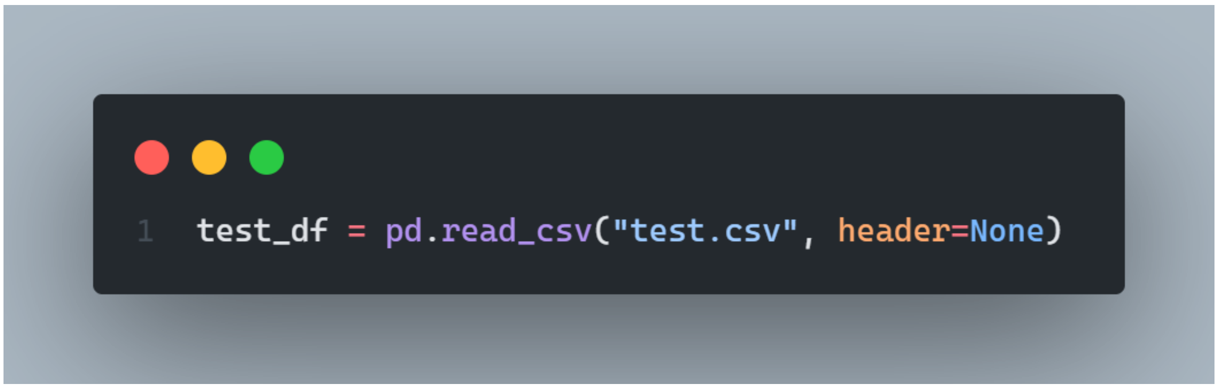
```python
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot as plt
```

Figure [1]: Imported Libraries

As seen in Figure 1, necessary libraries are imported. Pandas is used to read data from CSV files and save it in data frames. Numpy is used to create polynomial regression models. Sklearn is used to calculate the mean squared error of the models. Matplotlib is used to visualize the whole process.



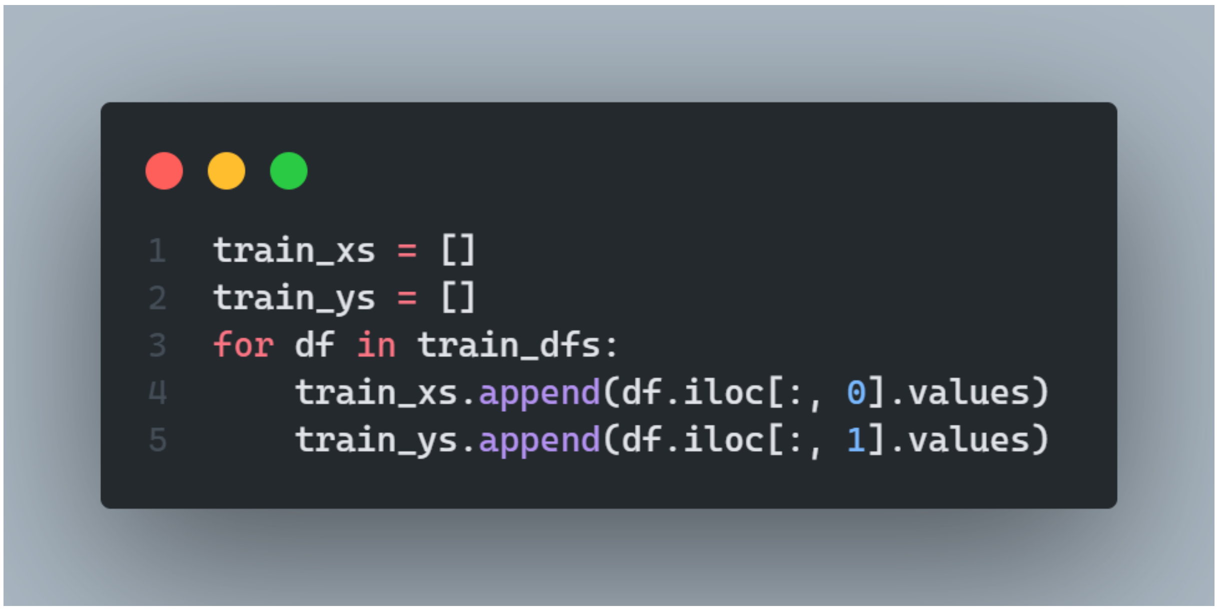
```python
train_dfs = []
for idx in range(1, 11):
    train_dfs.append(pd.read_csv("sample{}.csv".format(idx), header=None))
```



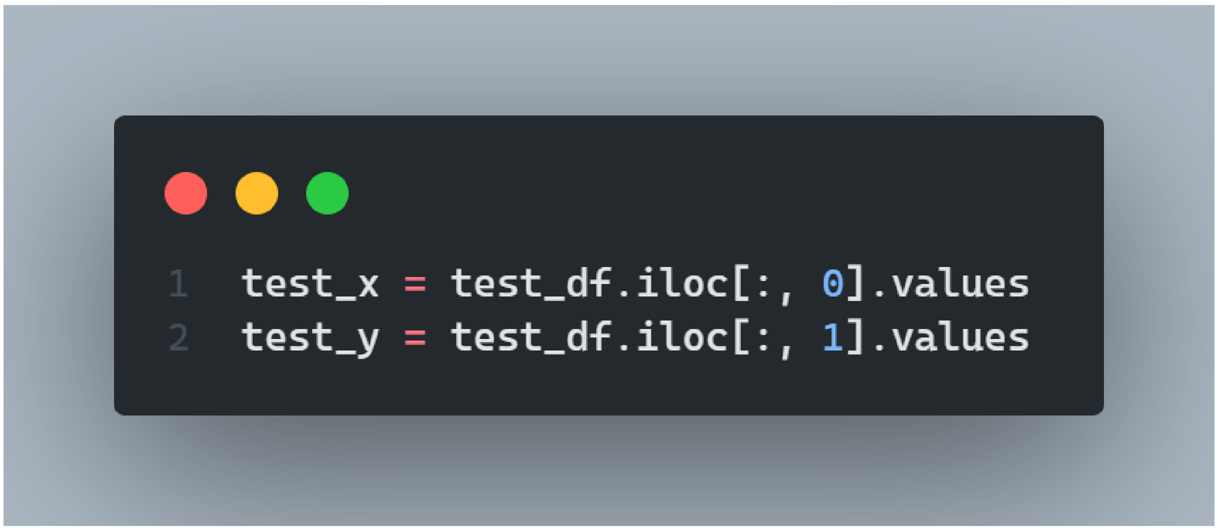
```python
test_df = pd.read_csv("test.csv", header=None)
```

Figure [2, 3]: Reading CSV Files and Saving into Data Frames

As seen in Figures 2 and 3, the data were read from the 10 training and 1 test data set CSV files and saved in data frames.



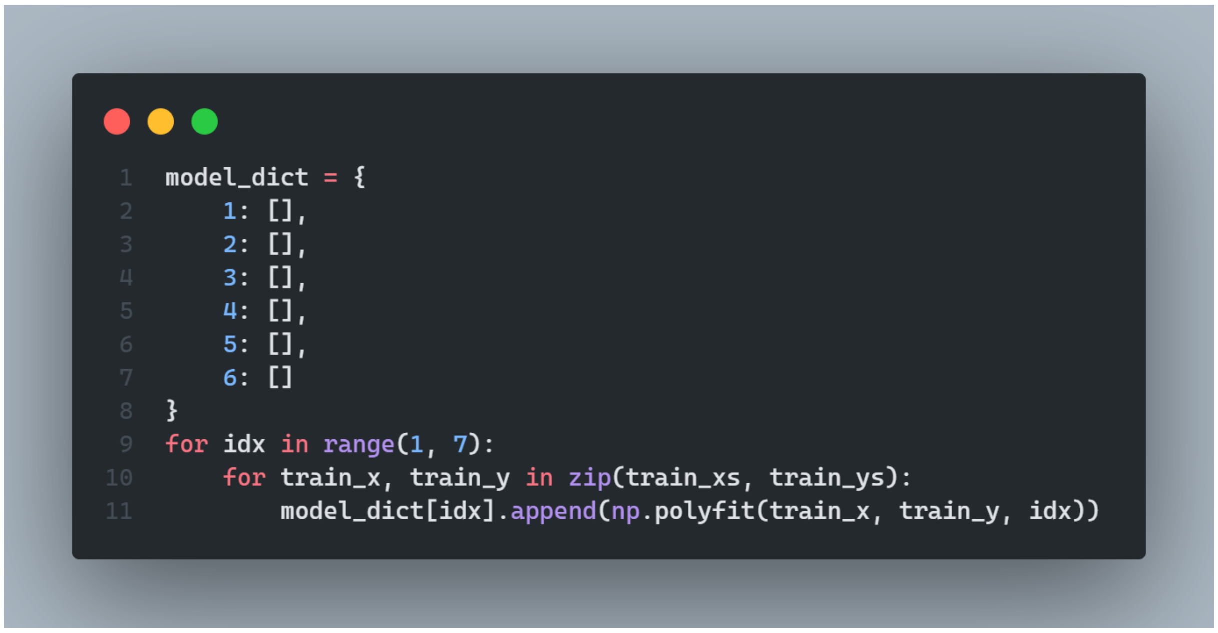
```python
train_xs = []
train_ys = []
for df in train_dfs:
    train_xs.append(df.iloc[:, 0].values)
    train_ys.append(df.iloc[:, 1].values)
```



```python
test_x = test_df.iloc[:, 0].values
test_y = test_df.iloc[:, 1].values
```

Figure [4, 5]: Splitting Input and Output Data

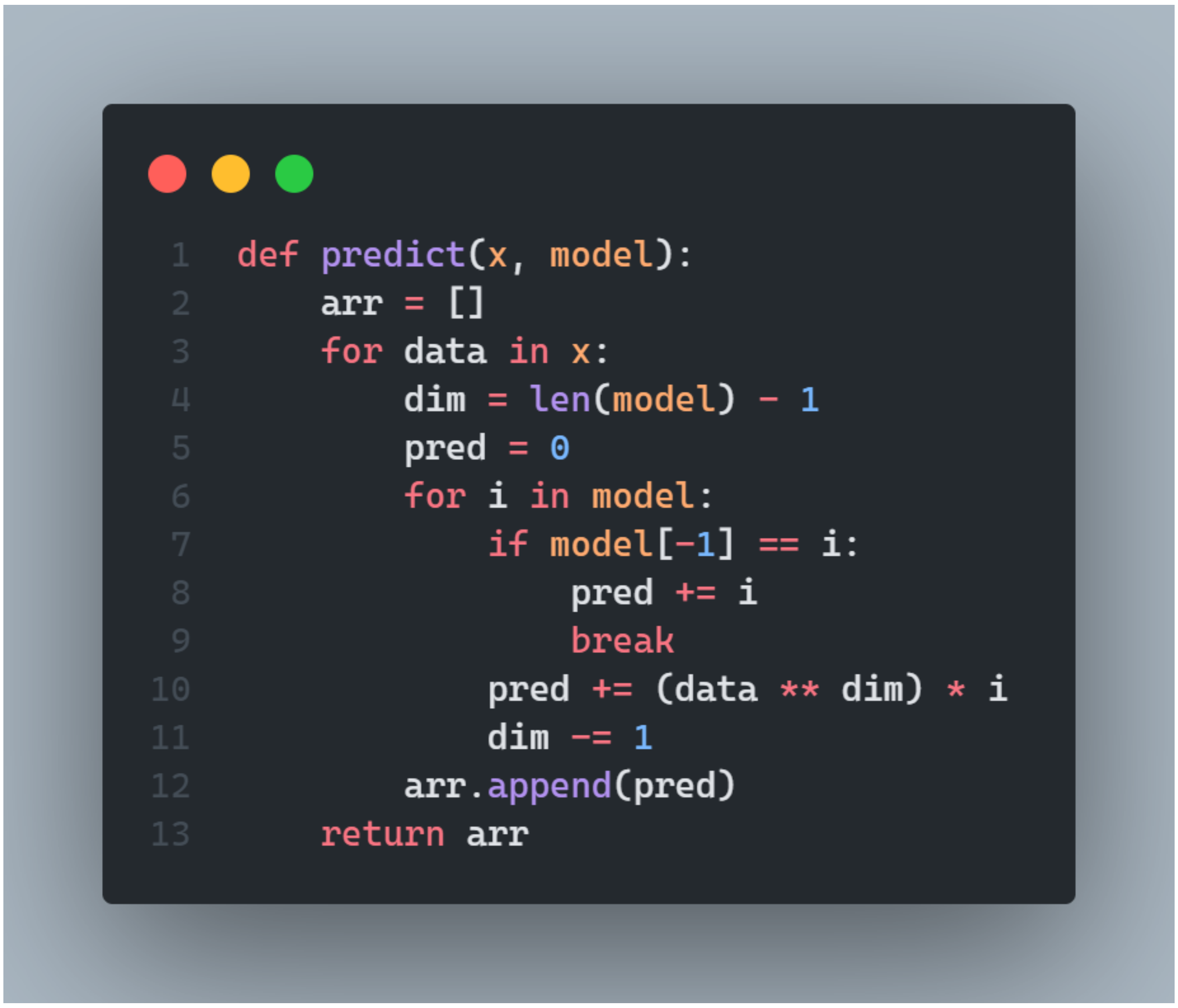
As seen in Figures 4 and 5, the input and output columns in the given training and test data sets are divided into different data frames.



```python
model_dict = {
    1: [],
    2: [],
    3: [],
    4: [],
    5: [],
    6: []
}
for idx in range(1, 7):
    for train_x, train_y in zip(train_xs, train_ys):
        model_dict[idx].append(np.polyfit(train_x, train_y, idx))
```

Figure [6]: Creating the Models

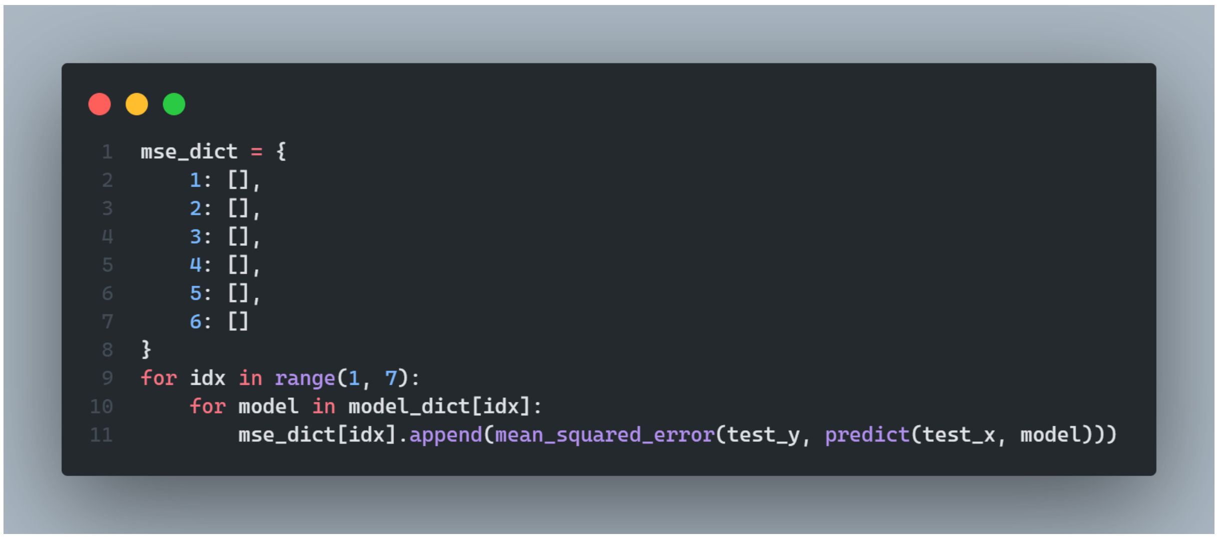
In Figure 6, it can be seen how models are created for each degree with the help of the ***np.polyfit*** function.



```python
def predict(x, model):
    arr = []
    for data in x:
        dim = len(model) - 1
        pred = 0
        for i in model:
            if model[-1] == i:
                pred += i
                break
            pred += (data ** dim) * i
            dim -= 1
        arr.append(pred)
    return arr
```

Figure [7]: Defining a Predict Function

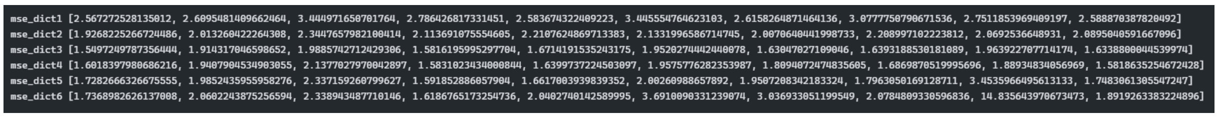
In Figure 7, the predict method to be used when calculating the mean squared error is created. This method takes a list of inputs and one of the models created above in its parameters. Then, using this model, the output value mapped by each input is calculated. Finally, a list of the output values is returned.



```python
mse_dict = {
    1: [],
    2: [],
    3: [],
    4: [],
    5: [],
    6: []
}
for idx in range(1, 7):
    for model in model_dict[idx]:
        mse_dict[idx].append(mean_squared_error(test_y, predict(test_x, model)))
```

Figure [8]: Calculating Mean Squared Error Values

In Figure 8, a mean squared error value was calculated using the test data set for each model created.



```
mse_dict1 [2.567272528135012, 2.6095481409662464, 3.444971650701764, 2.786426817331451, 2.583674322409223, 3.445554764623103, 2.6158264871464136, 3.0777750790671536, 2.7511853969409197, 2.588870387820492]
mse_dict2 [1.926822526672448 6, 2.013260422264308, 2.3447657982100414, 2.113691075554605, 2.2107624869713383, 2.1331996586714745, 2.0070640441998733, 2.208997102223812, 2.0692536648931, 2.0895040591667096]
mse_dict3 [1.5497249787356444, 1.914317046598652, 1.9885742712429306, 1.5816195995297704, 1.6714191535243175, 1.9520274442440078, 1.63047027109046, 1.6393188530181089, 1.963922707714174, 1.6338800044539974]
mse_dict4 [1.6018397980686216, 1.9407904534903055, 2.1377027970042897, 1.5831023434000844, 1.6399737224503097, 1.9575776282353987, 1.8094072474835605, 1.6869870519995696, 1.88934834056969, 1.5818635254672428]
mse_dict5 [1.7282666326675555, 1.9852435955958276, 2.337159260799627, 1.591852886057904, 1.6617003939839352, 2.00260988657892, 1.9507208342183324, 1.7963050169128711, 3.4535966495613133, 1.7483061305547247]
mse_dict6 [1.736898262613700 8, 2.0602243875256594, 2.338943487710146, 1.6186765173254736, 2.0402740142589995, 3.6910090331239074, 3.036933051199549, 2.0784809330596836, 14.835643970673473, 1.8919263383224896]
```

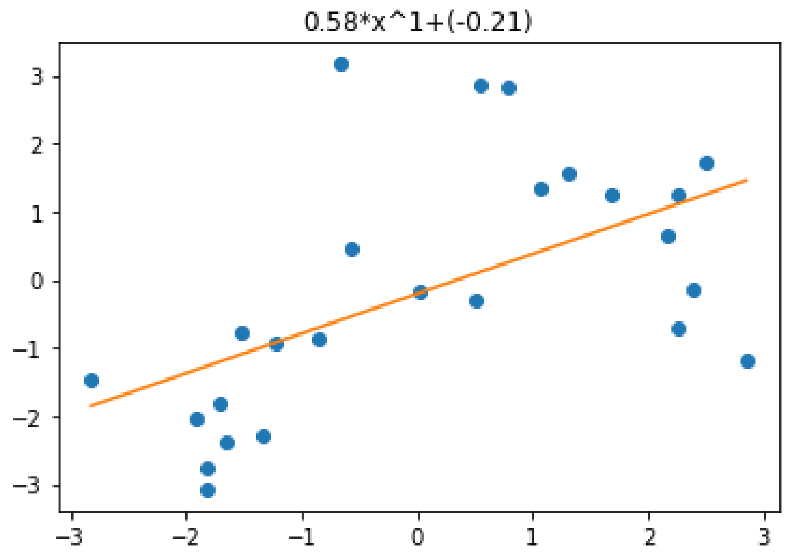
Figure [9]: Mean Squared Errors for All Models of All Degrees

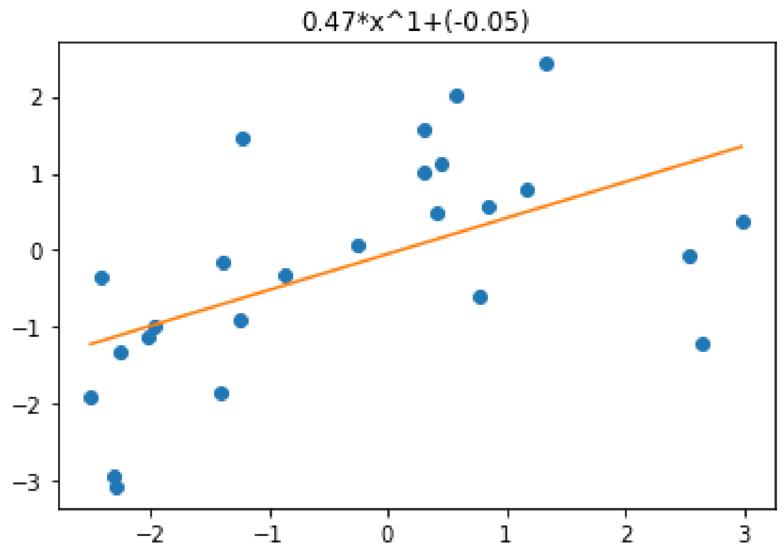
In Figure 9, the calculated mean squared errors of each model created for each degree can be seen.

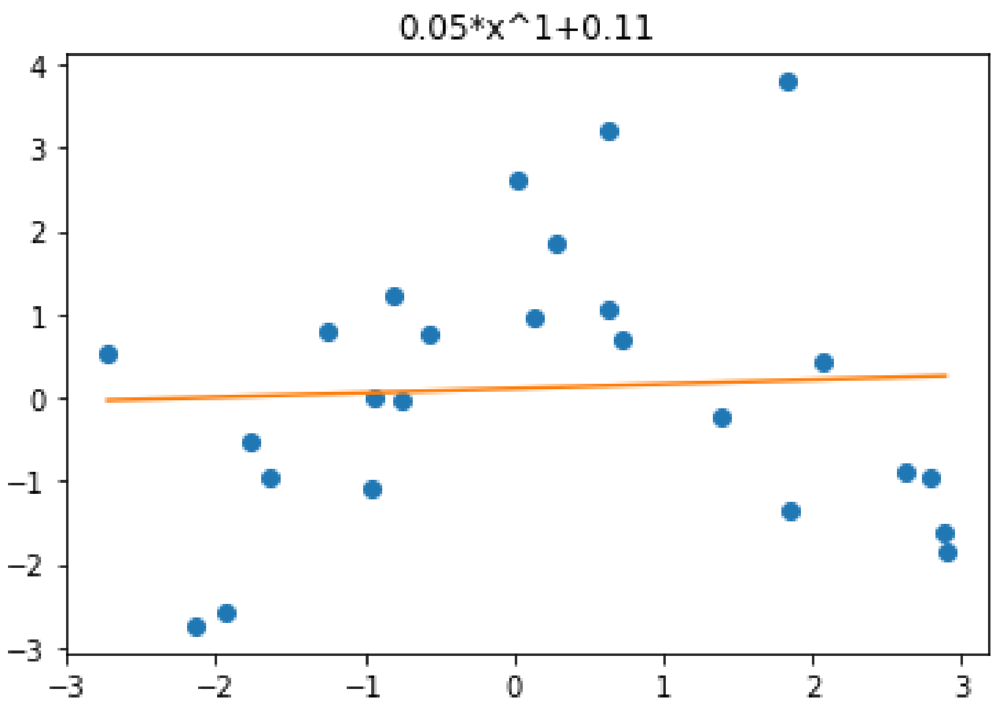
4. Results

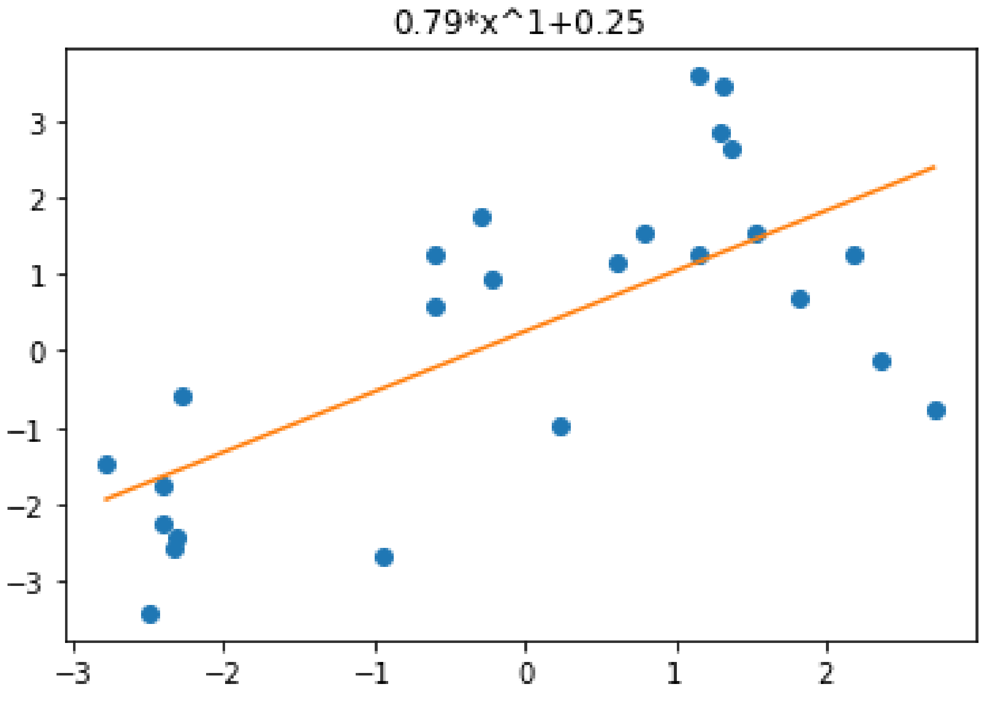
Below you can see the plots of total 60 different models created for each degree.

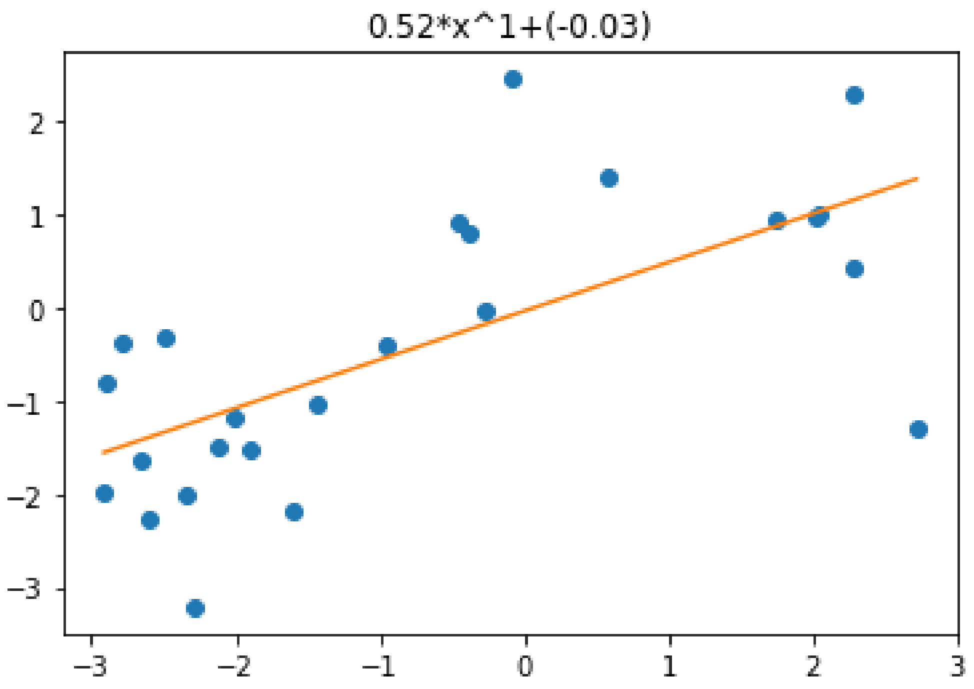
- Plots for Degree 1:



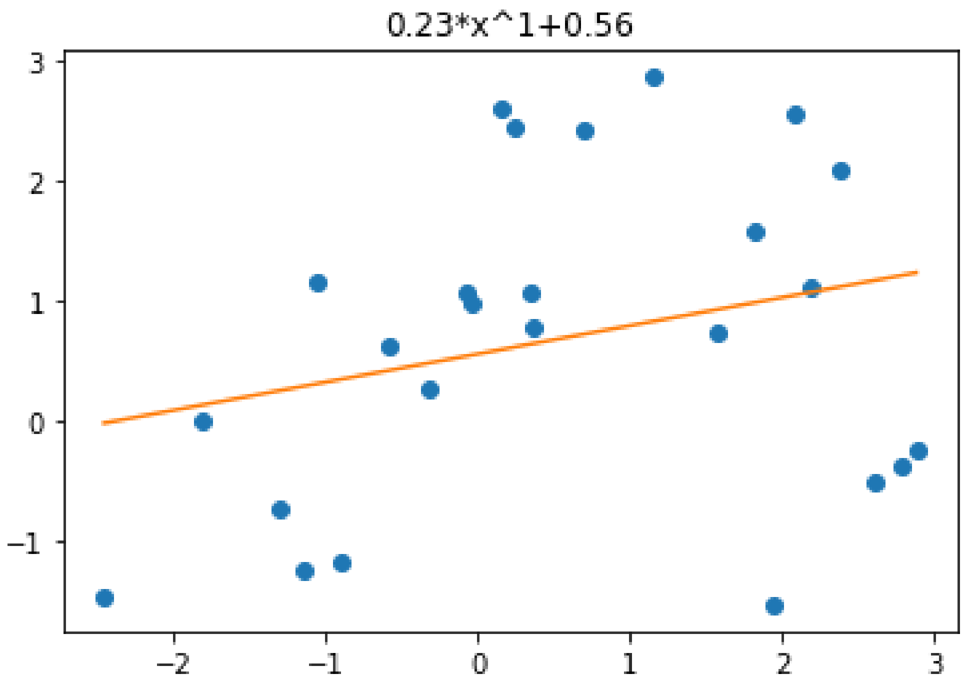



0.05*x^1+0.11
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

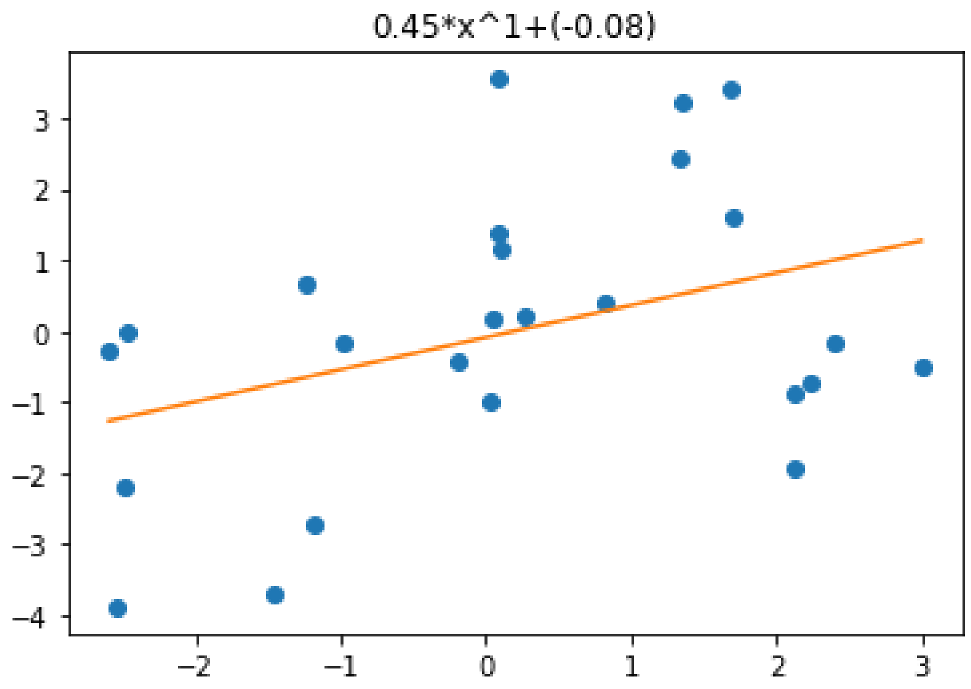

0.79*x^1+0.25
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2

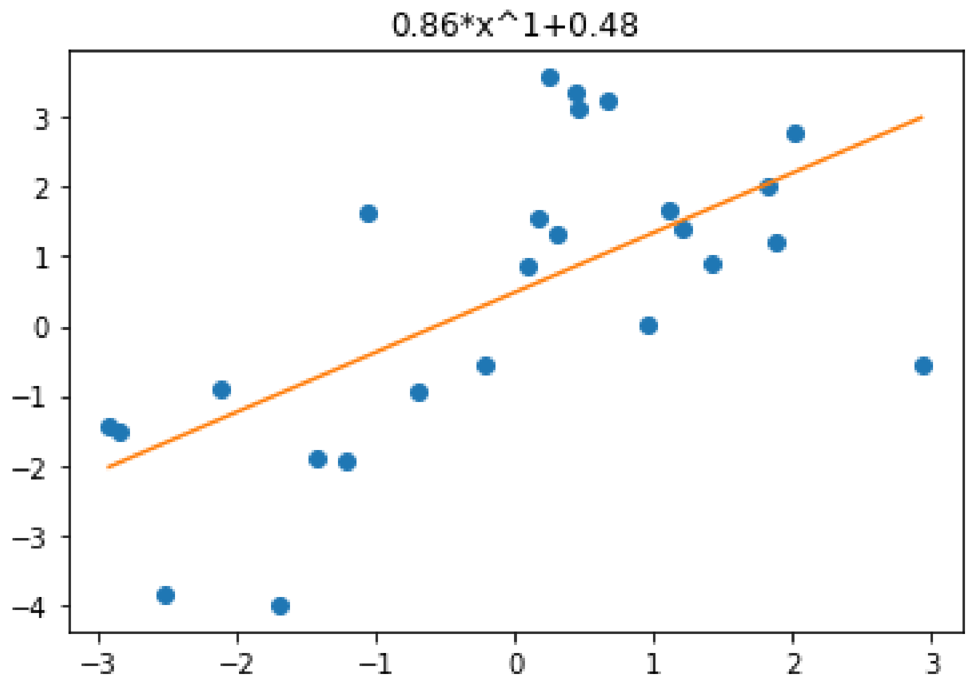


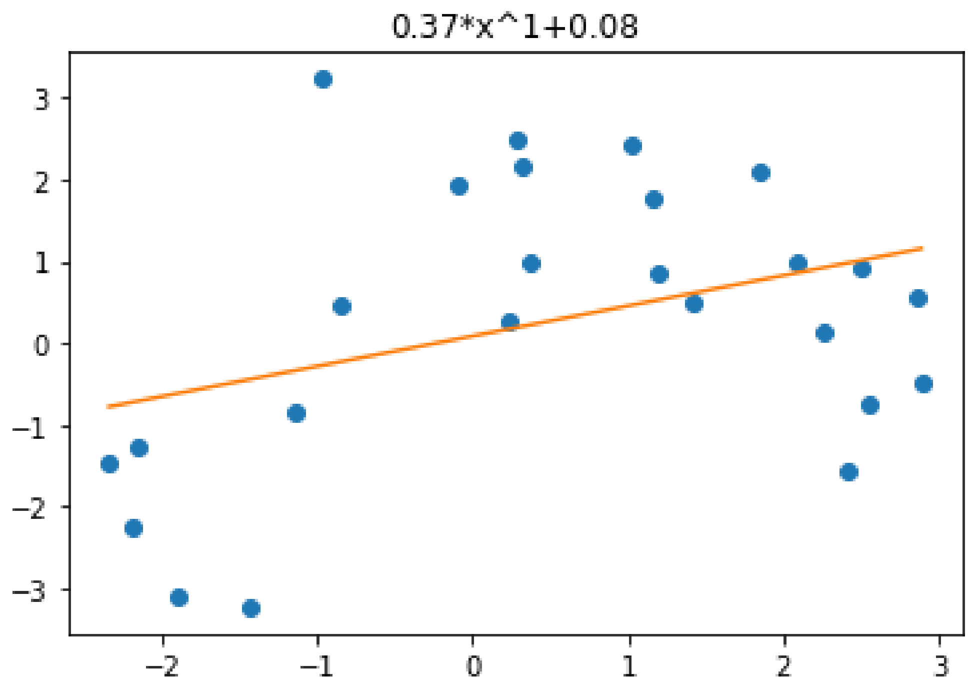
0.52*x^1+(-0.03)
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

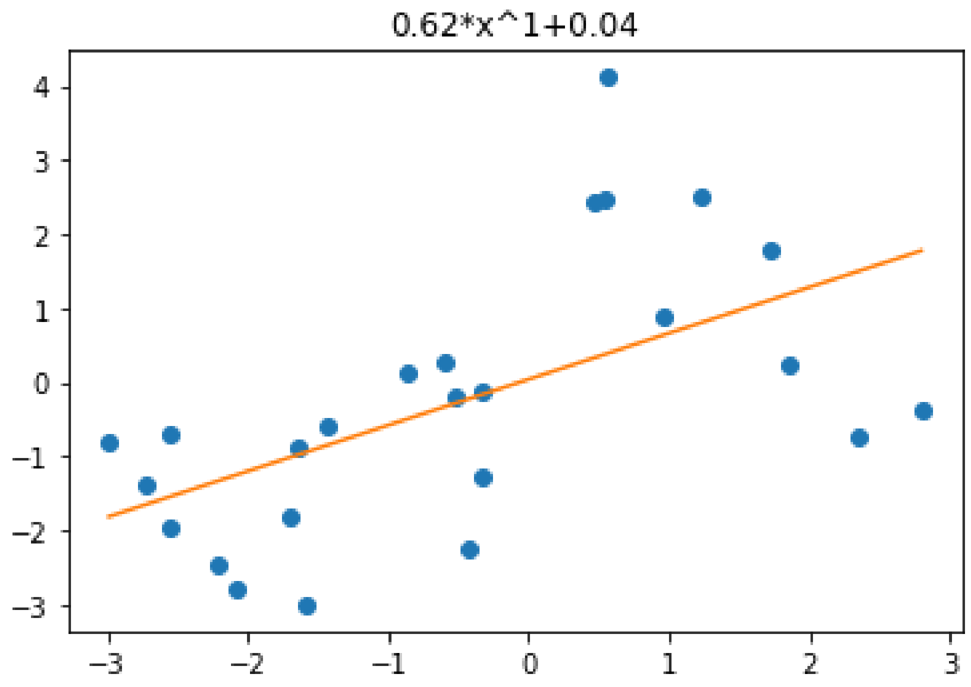


0.23*x^1+0.56
3
2
1
0
−1
−2
−1
0
1
2
3

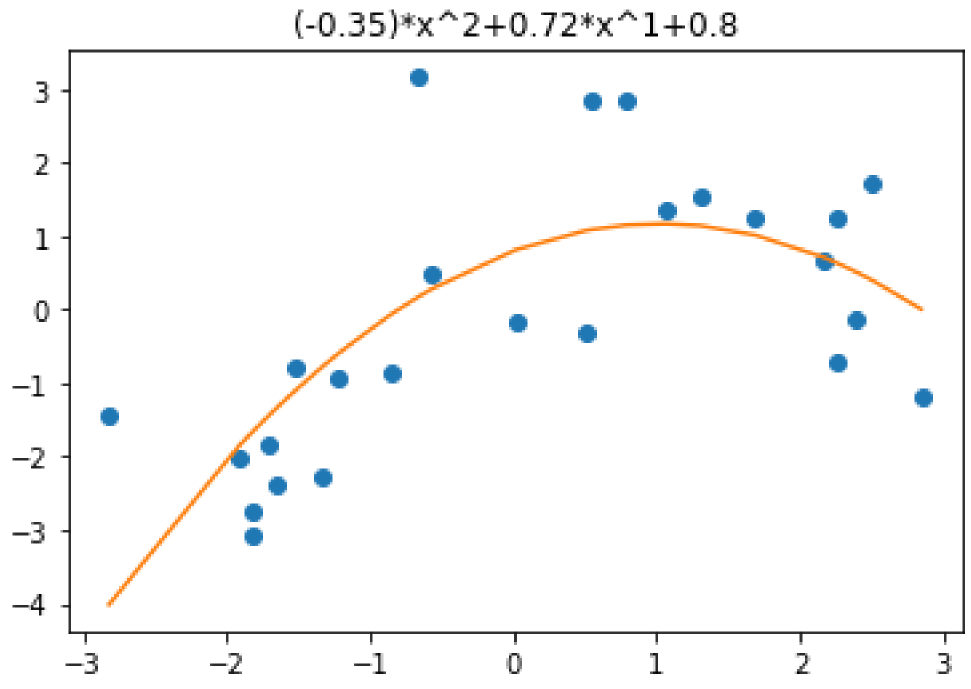

0.45*x^1+(-0.08)

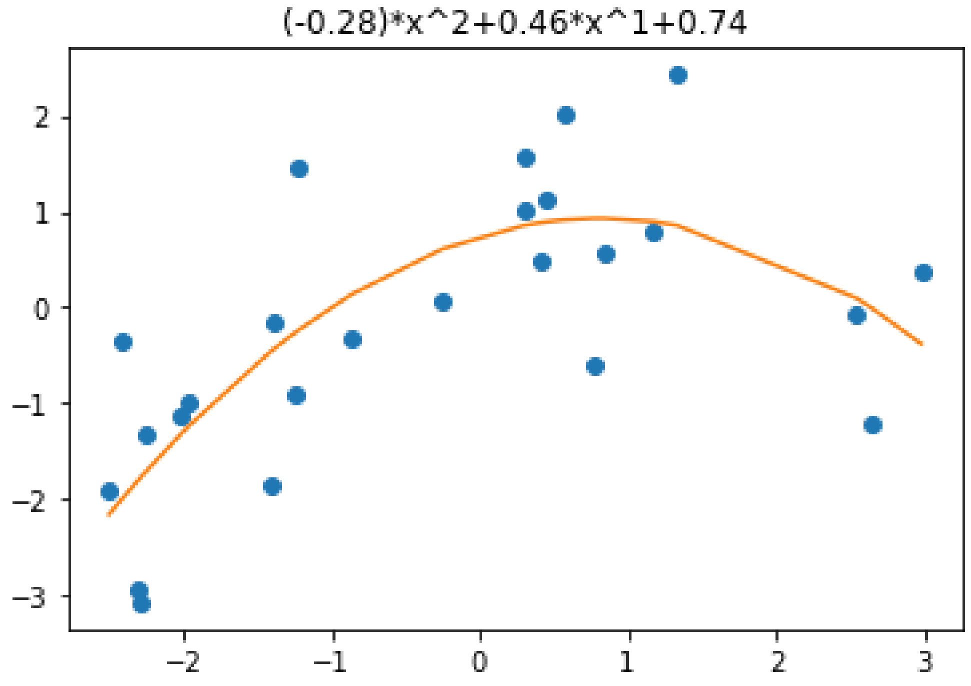

0.86*x^1+0.48

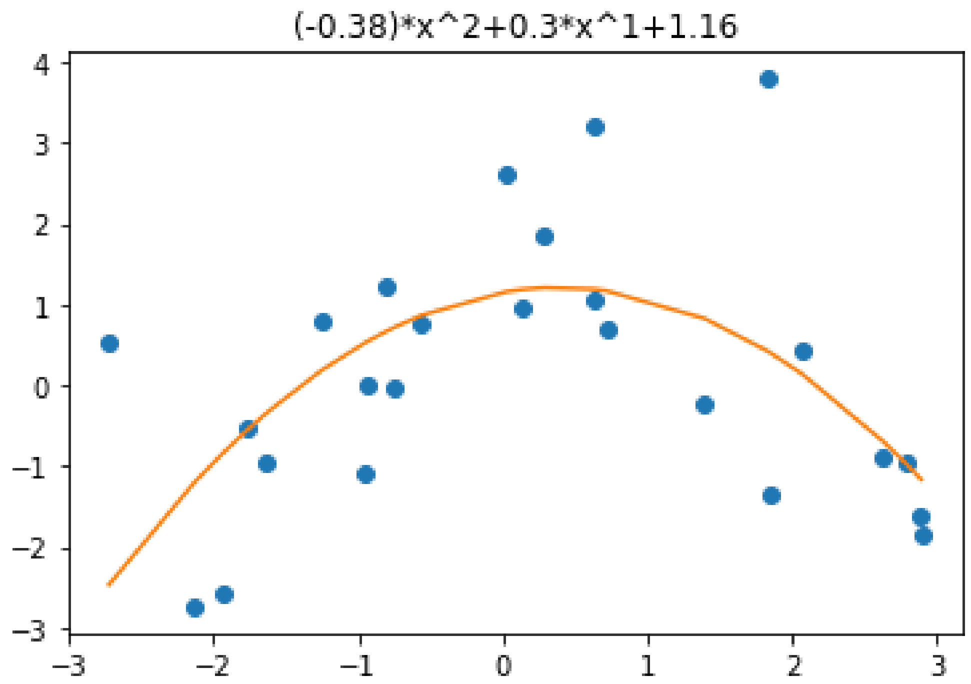

0.37*x^1+0.08
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

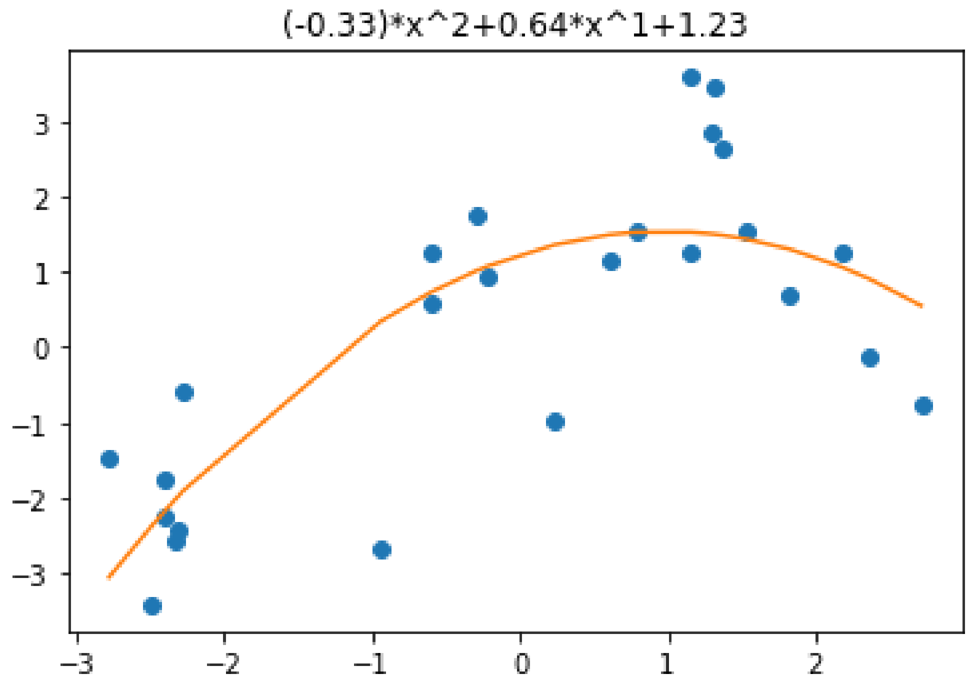

0.62*x^1+0.04
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

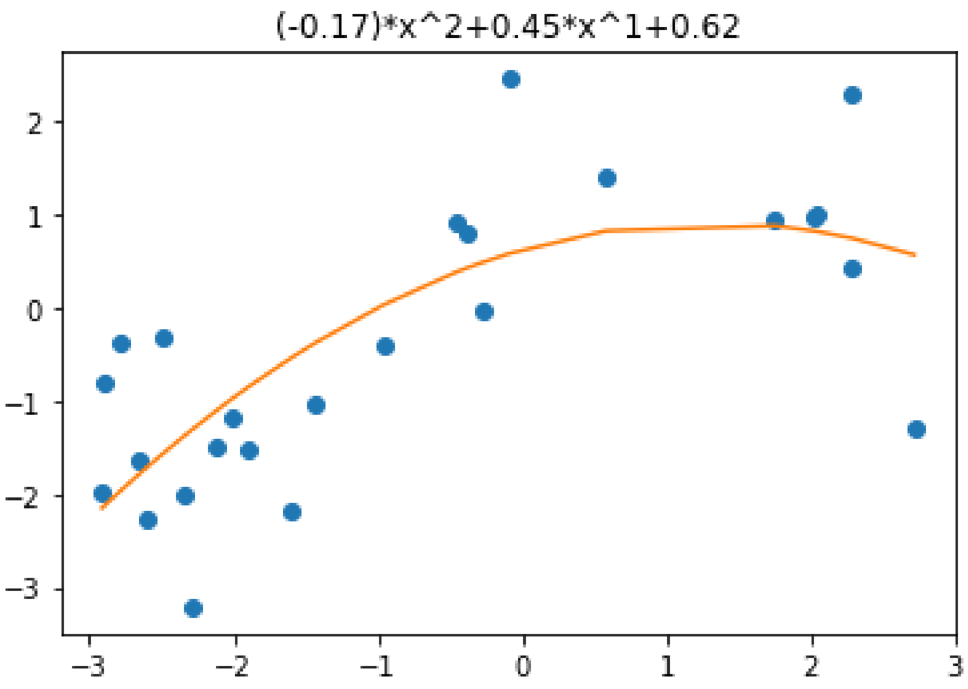
- Plots for Degree 2:

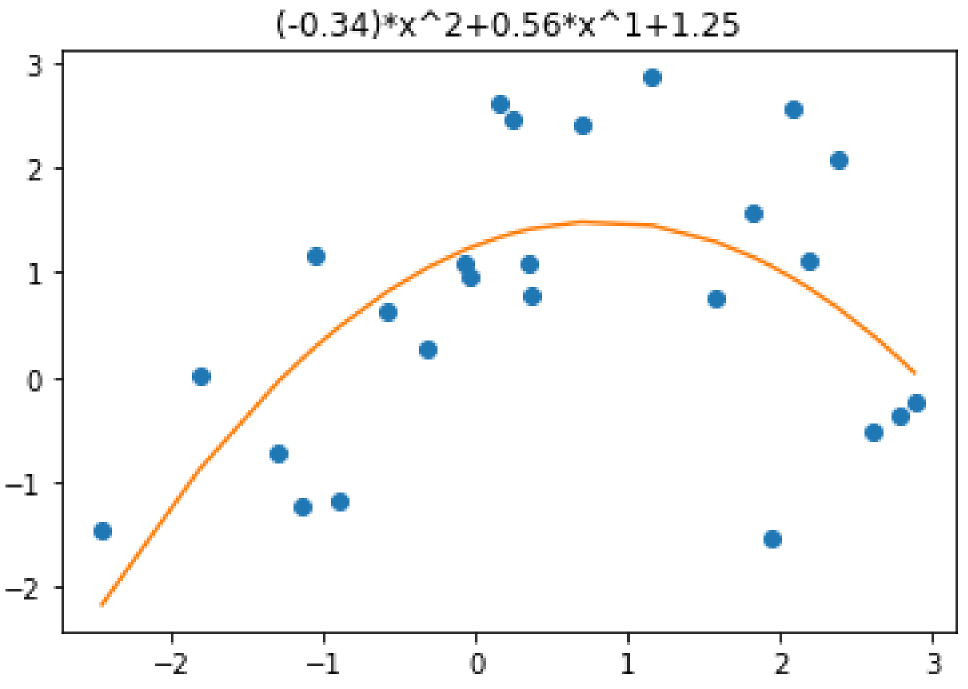


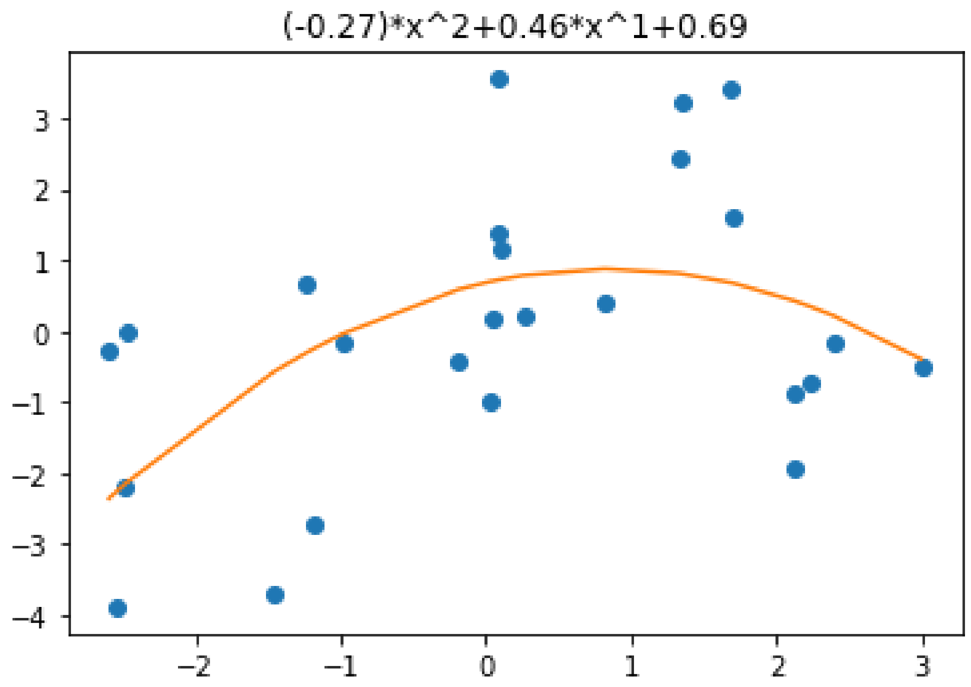


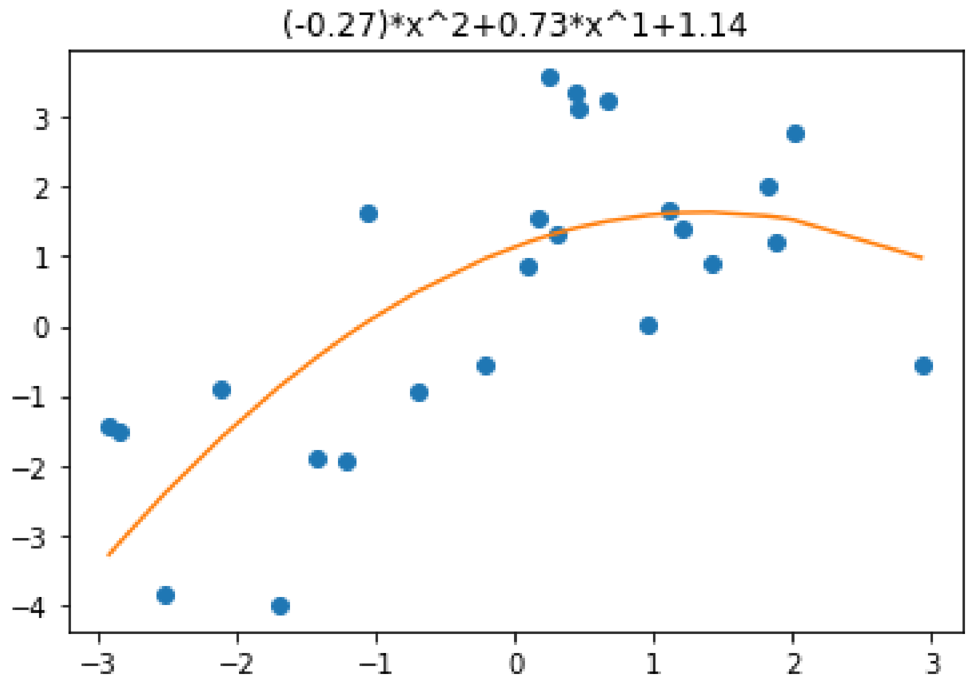

(-0.38)*x^2+0.3*x^1+1.16
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

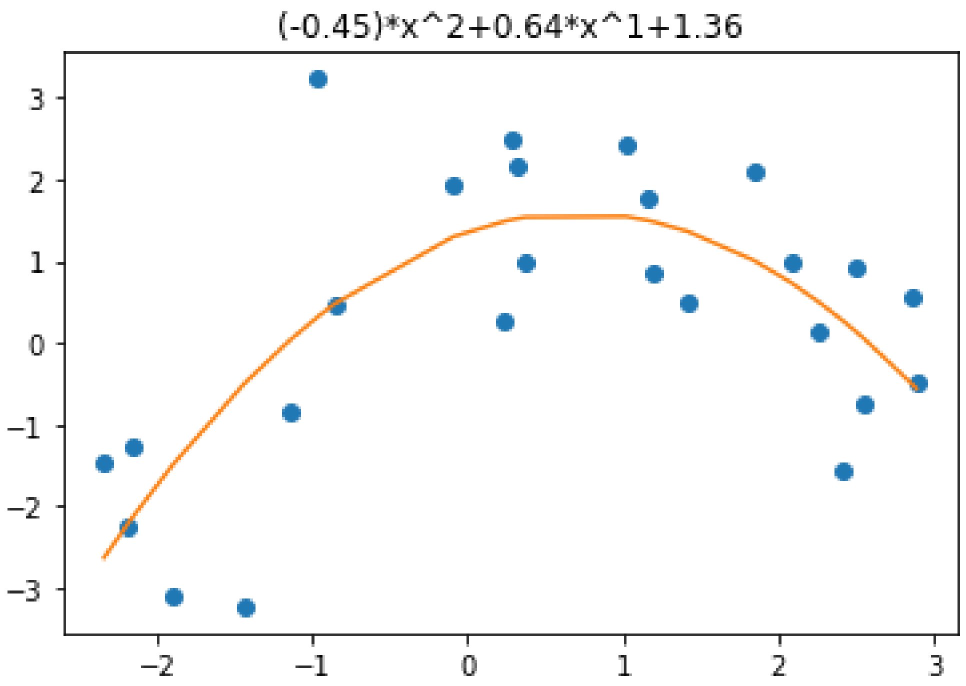

(-0.33)*x^2+0.64*x^1+1.23
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2

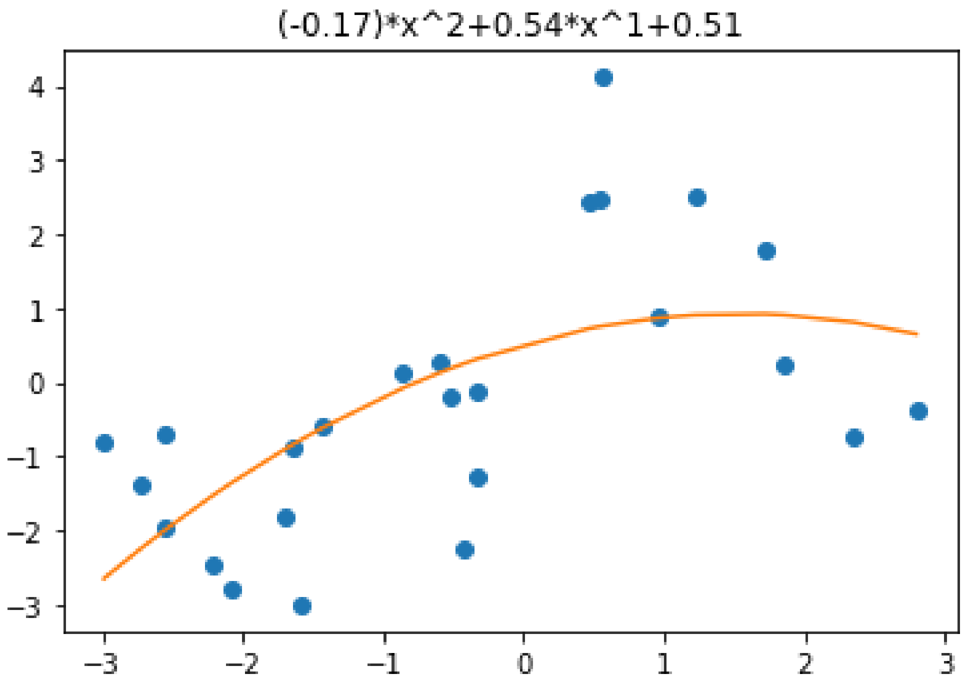

(-0.17)*x^2+0.45*x^1+0.62
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


(-0.34)*x^2+0.56*x^1+1.25
3
2
1
0
−1
−2
−2
−1
0
1
2
3

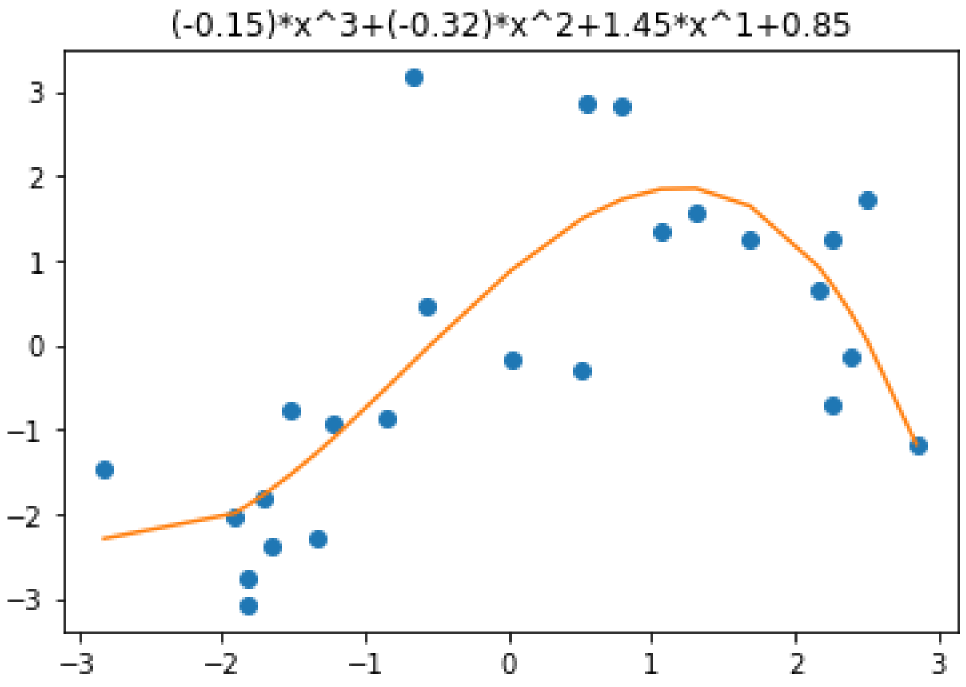

(-0.27)*x^2+0.46*x^1+0.69
3
2
1
0
−1
−2
−3
−4
−2
−1
0
1
2
3

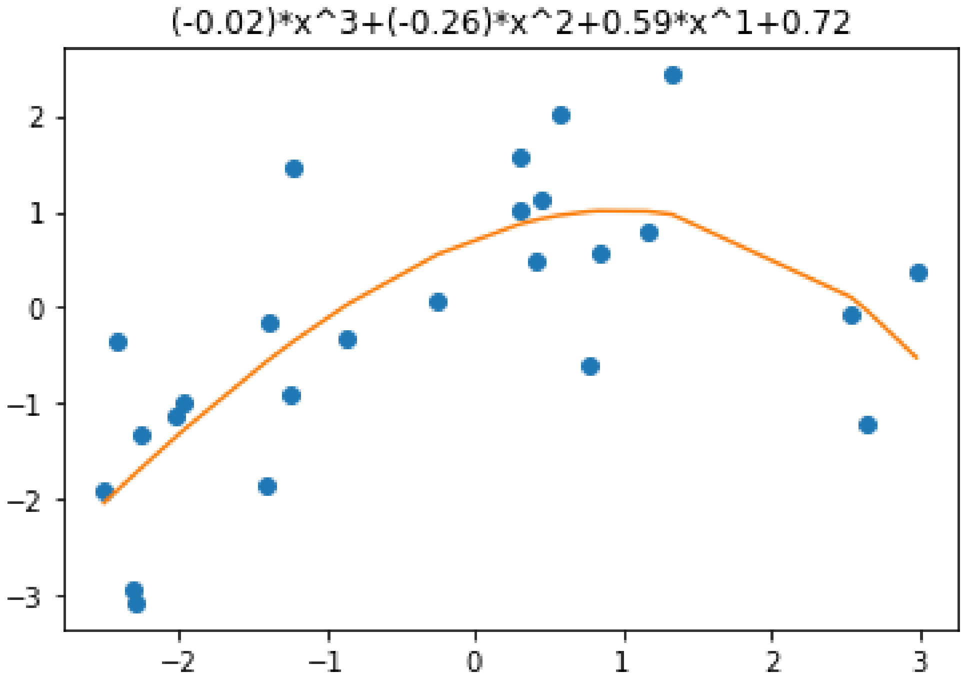

(-0.27)*x^2+0.73*x^1+1.14
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3

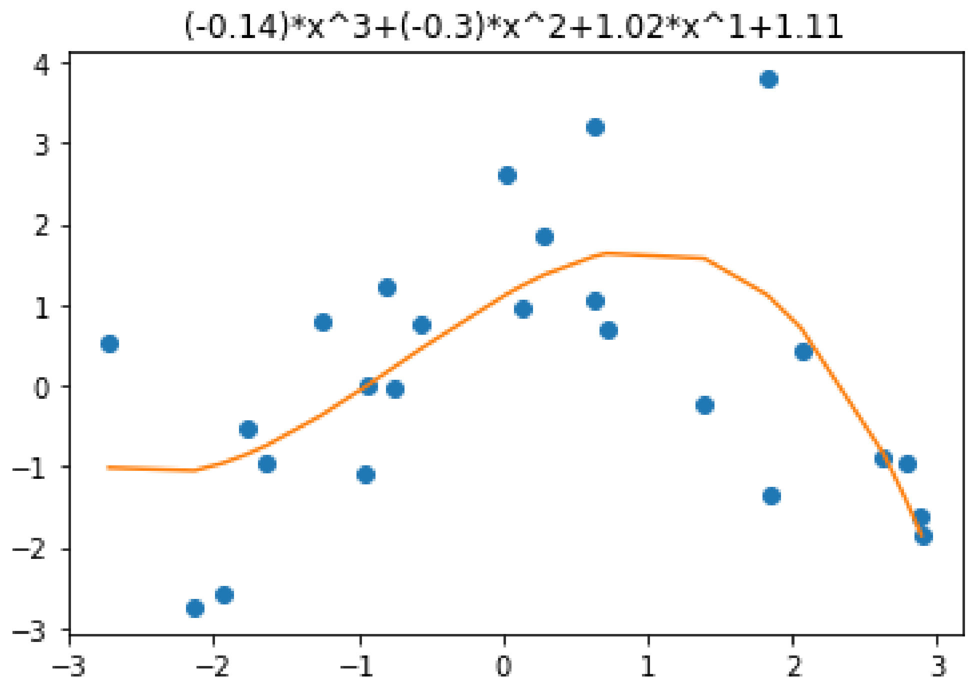

(-0.45)*x^2+0.64*x^1+1.36
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

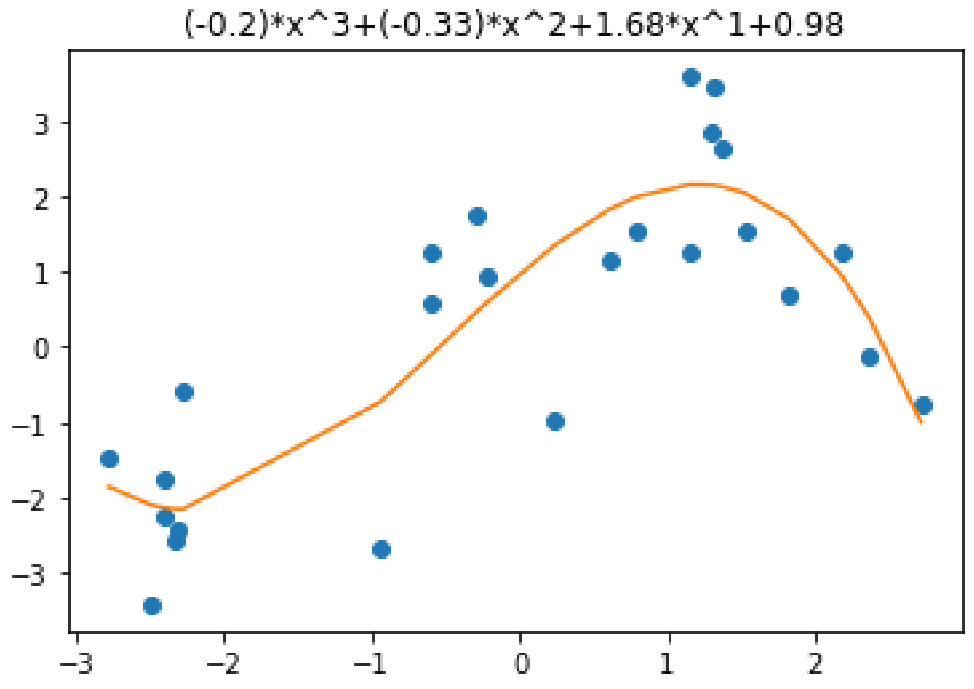

(-0.17)*x^2+0.54*x^1+0.51
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

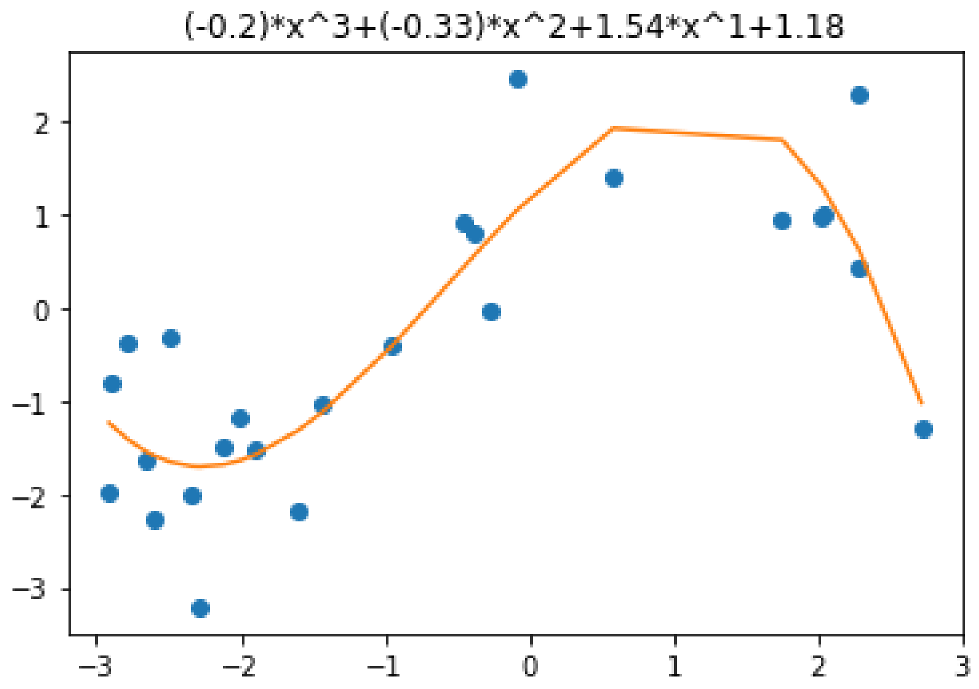
- Plots for Degree 3:

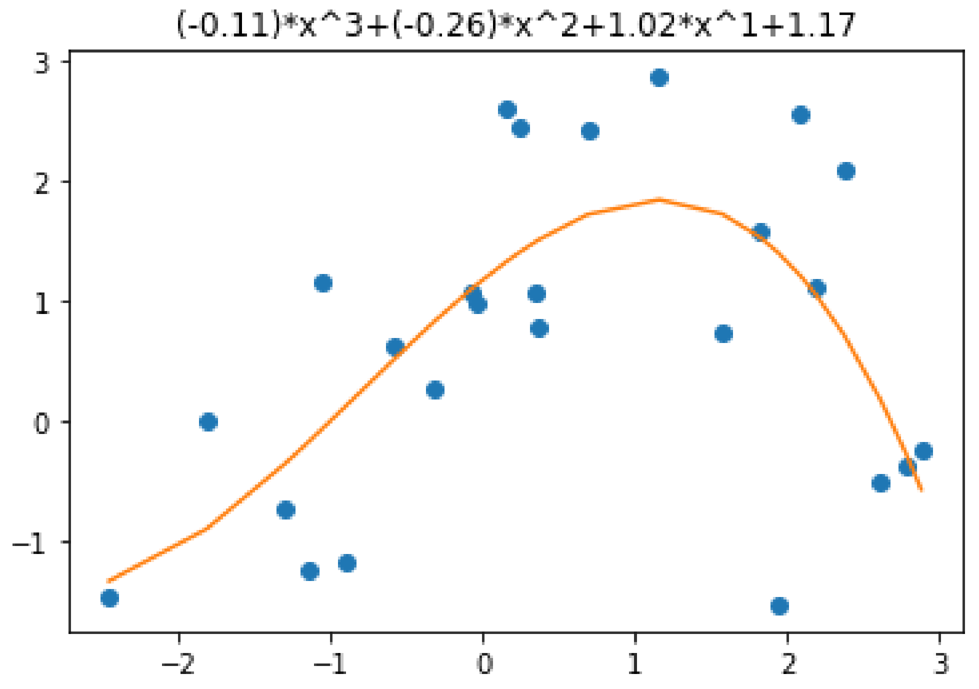


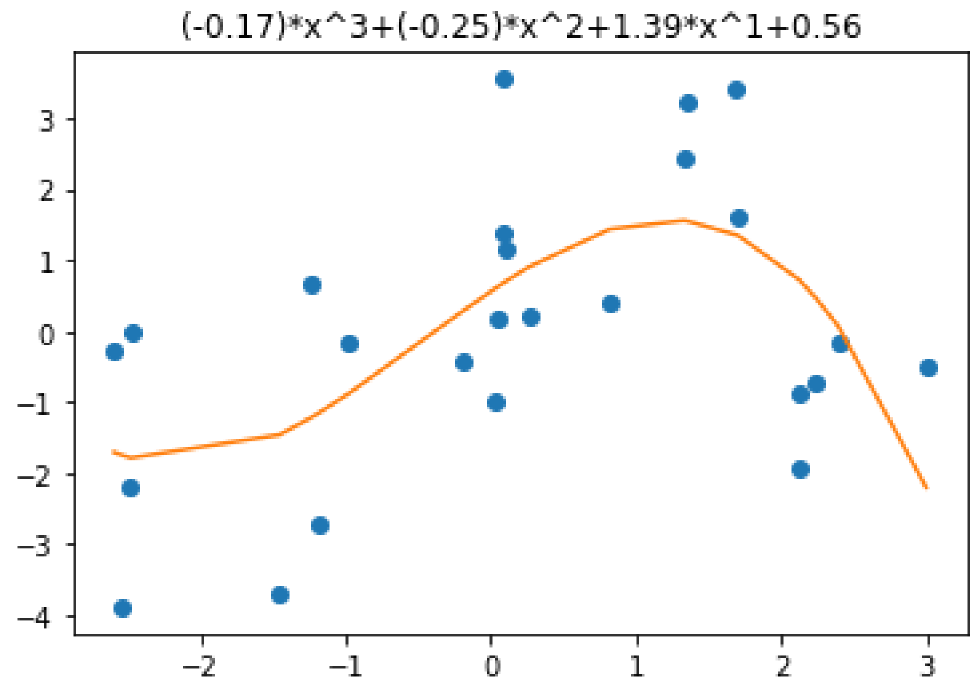
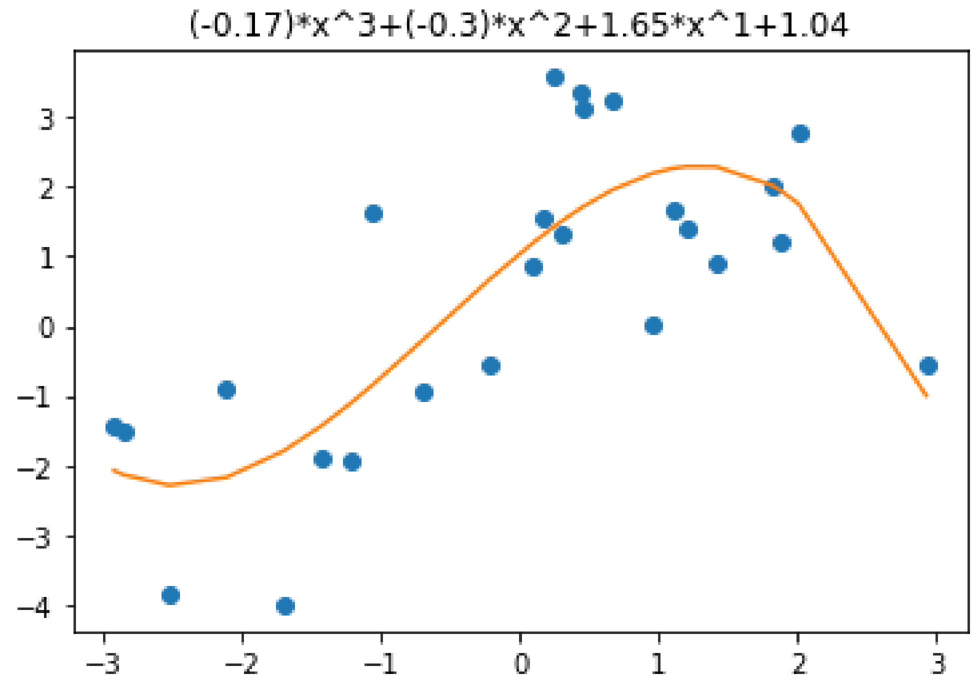


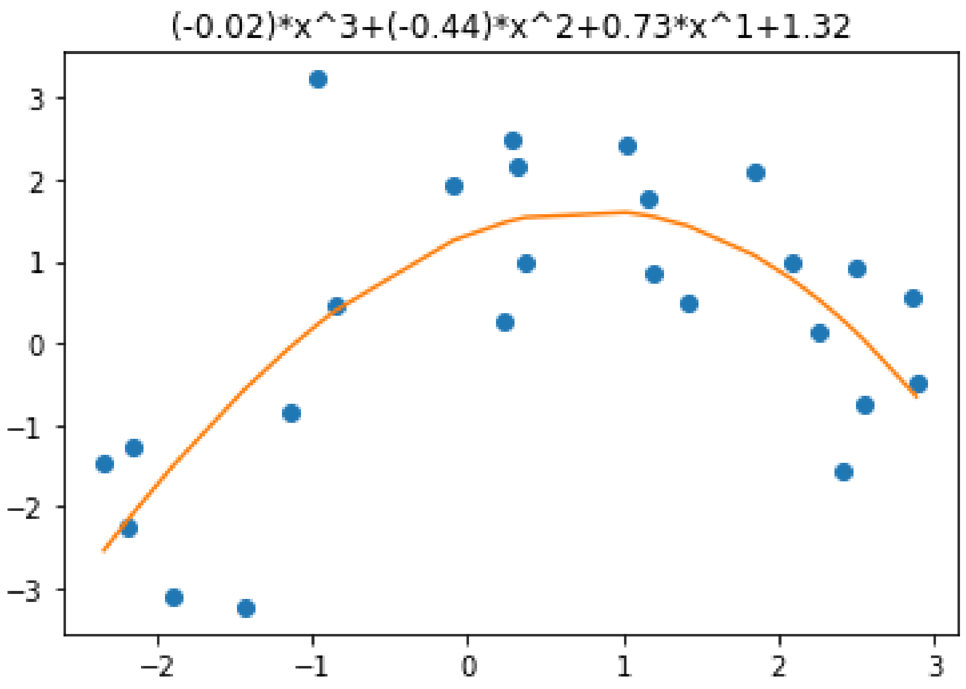

(-0.14)*x^3+(-0.3)*x^2+1.02*x^1+1.11
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

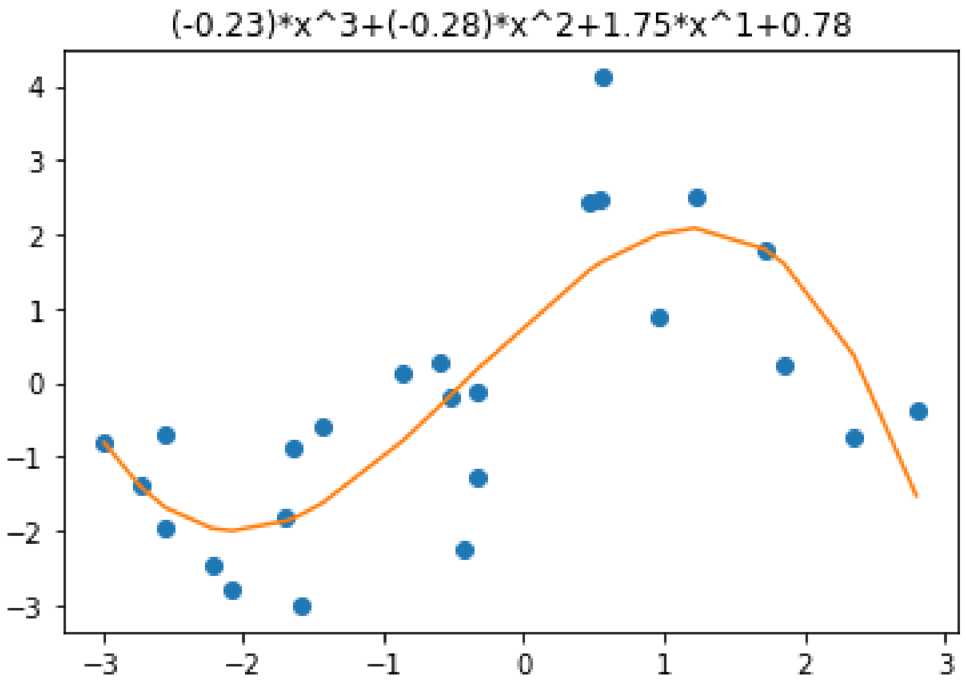

(-0.2)*x^3+(-0.33)*x^2+1.68*x^1+0.98
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2


(-0.2)*x^3+(-0.33)*x^2+1.54*x^1+1.18
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

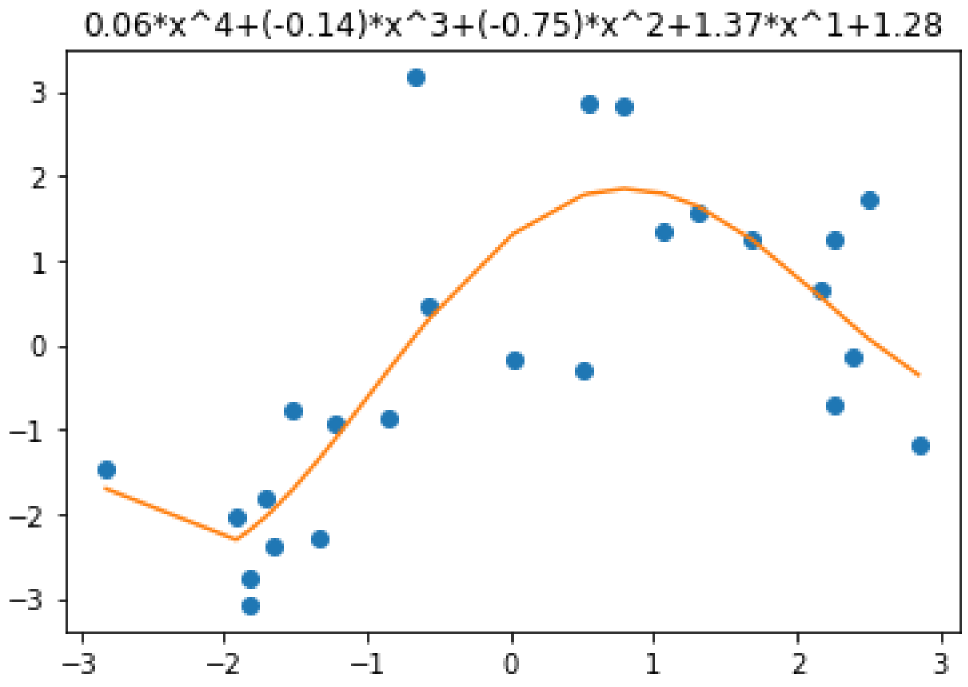

(-0.11)*x^3+(-0.26)*x^2+1.02*x^1+1.17
3
2
1
0
−1
−2
−1
0
1
2
3

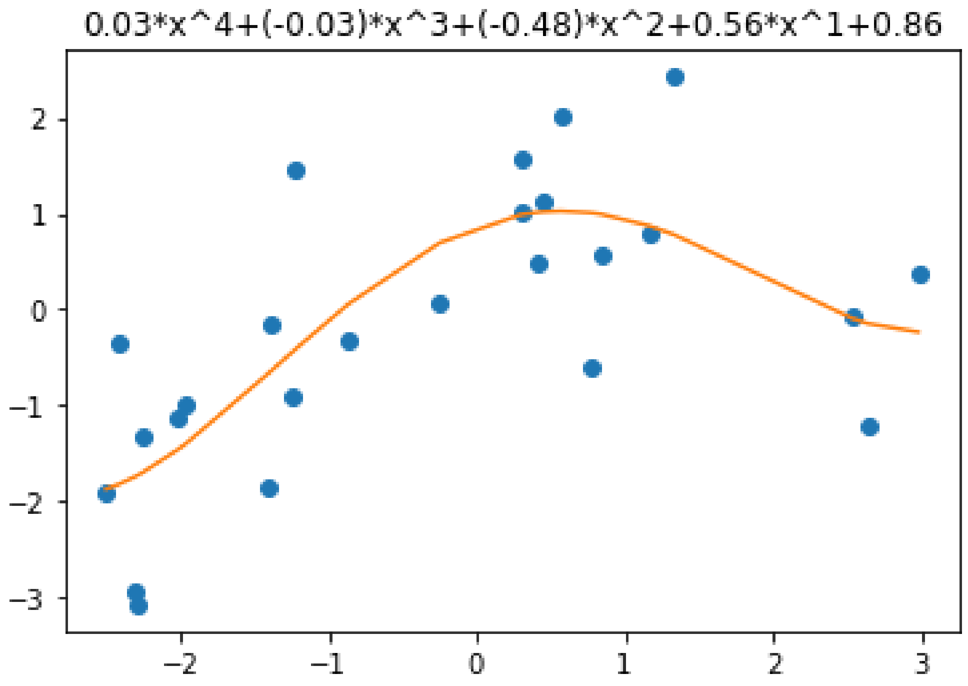

(-0.17)*x^3+(-0.25)*x^2+1.39*x^1+0.56
3
2
1
0
−1
−2
−3
−4
−2
−1
0
1
2
3

(-0.17)*x^3+(-0.3)*x^2+1.65*x^1+1.04
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3

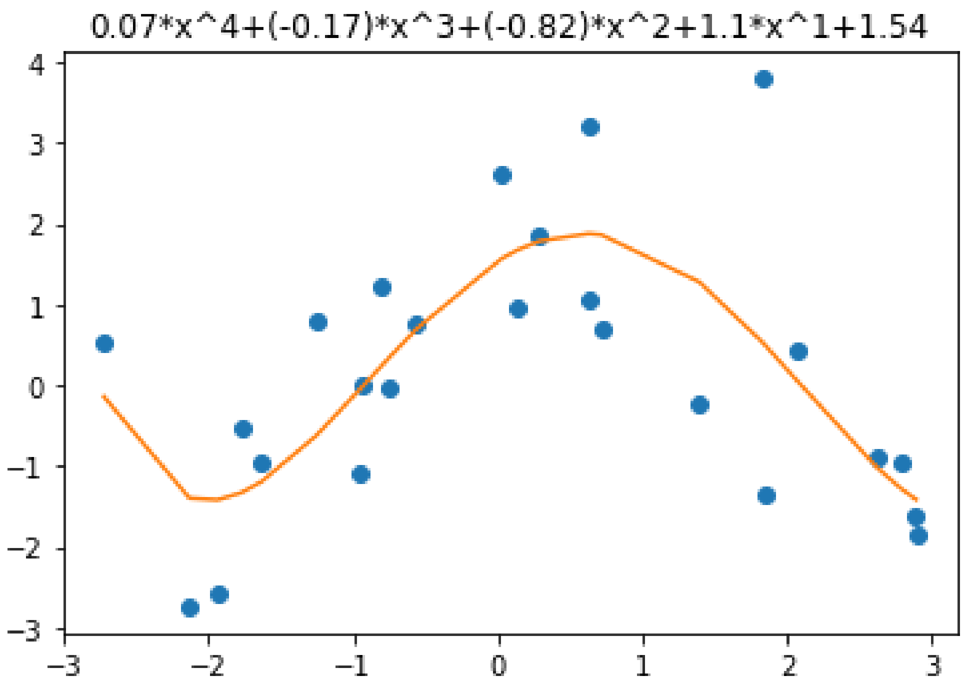

(-0.02)*x^3+(-0.44)*x^2+0.73*x^1+1.32
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

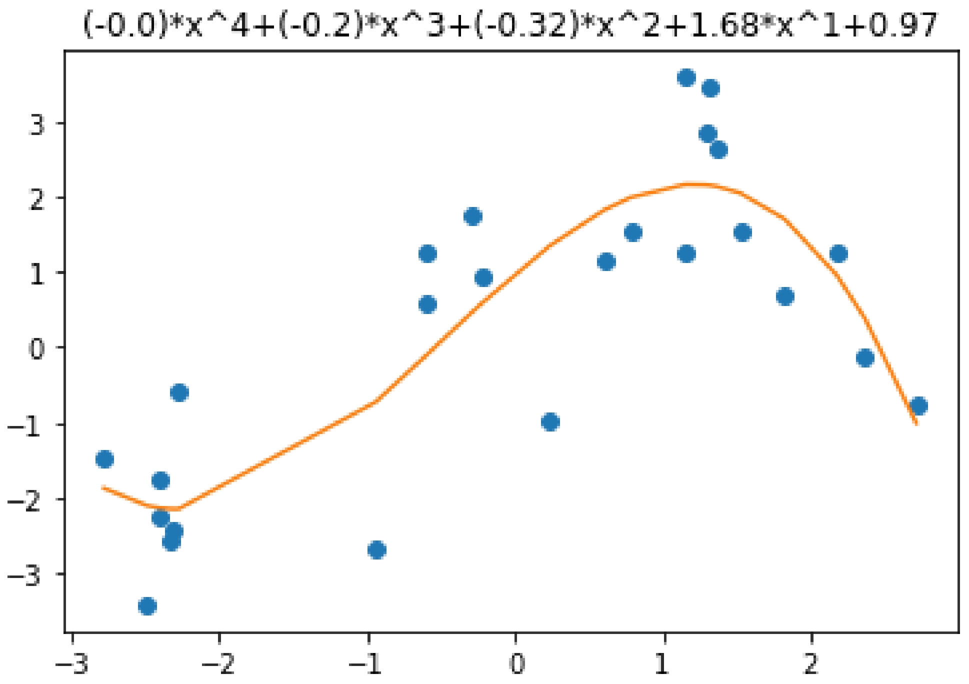

(-0.23)*x^3+(-0.28)*x^2+1.75*x^1+0.78
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

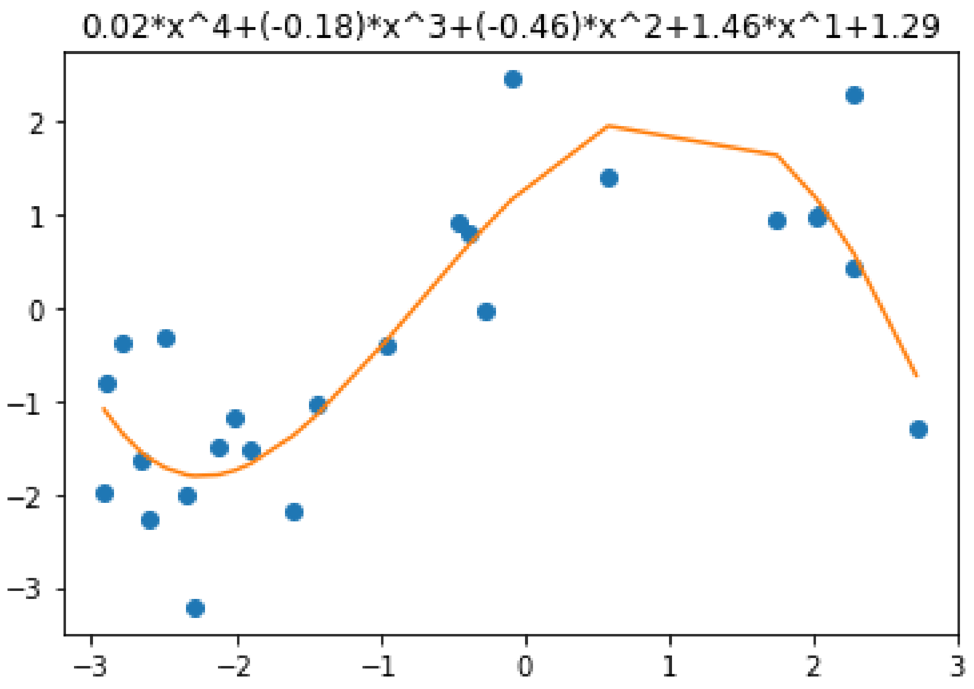
- Plots for Degree 4:

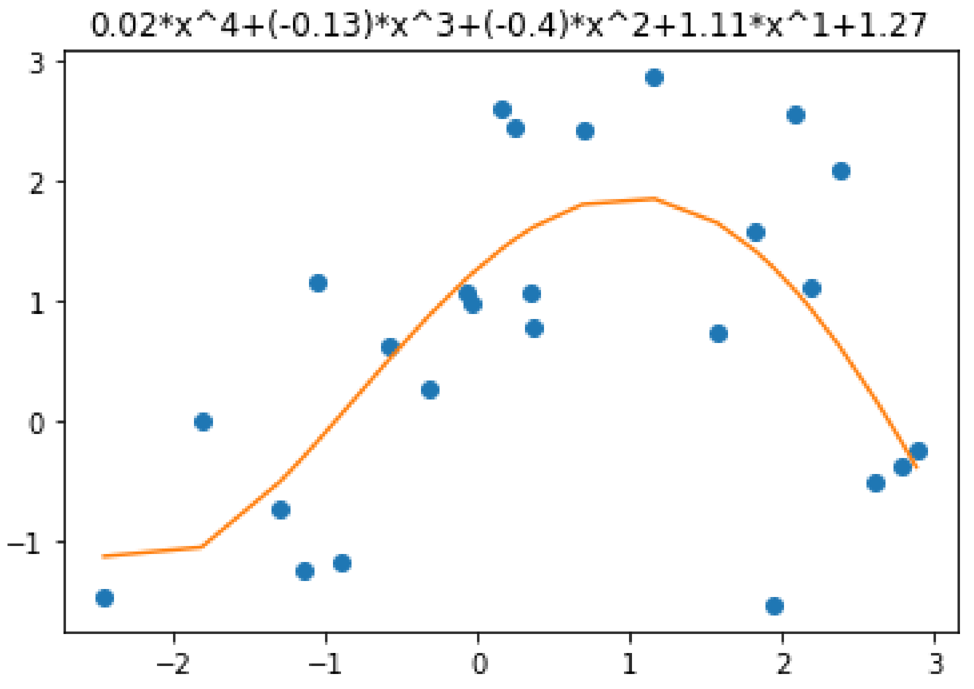


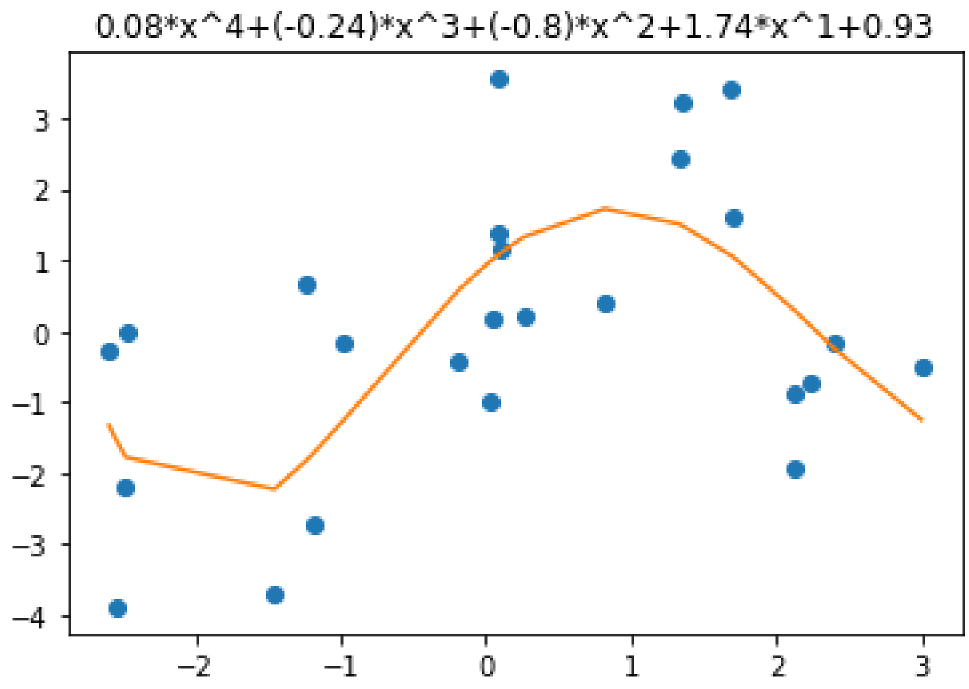


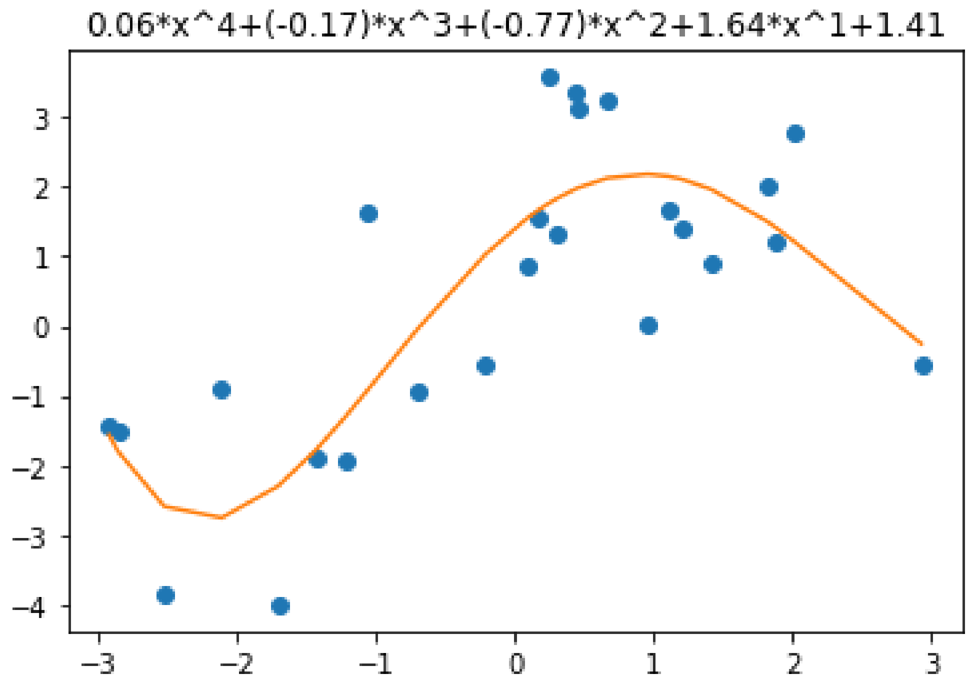

0.07*x^4+(-0.17)*x^3+(-0.82)*x^2+1.1*x^1+1.54
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

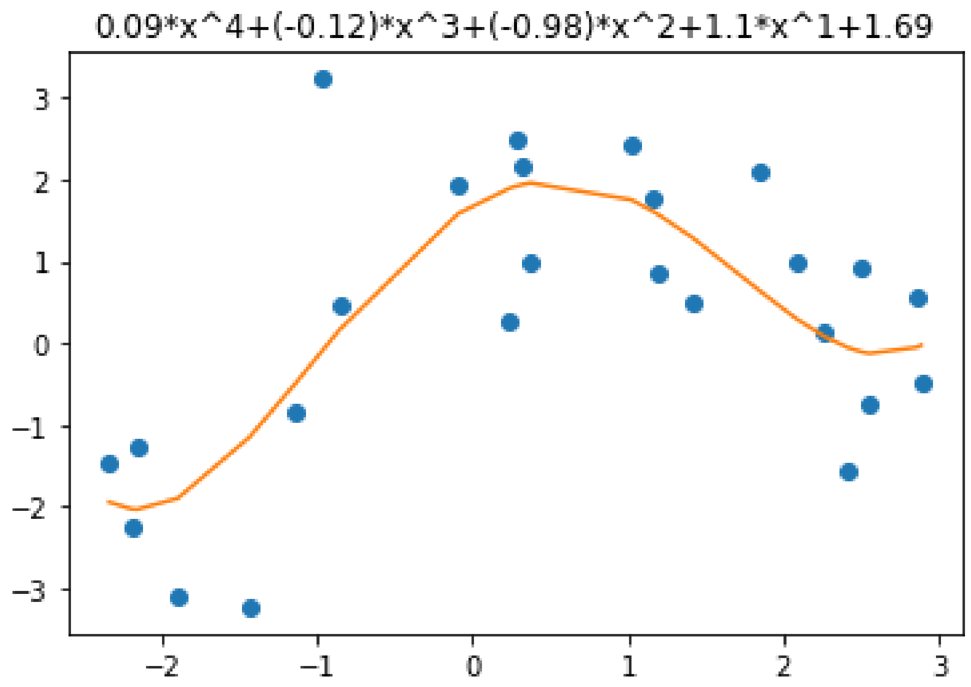

(-0.0)*x^4+(-0.2)*x^3+(-0.32)*x^2+1.68*x^1+0.97
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2

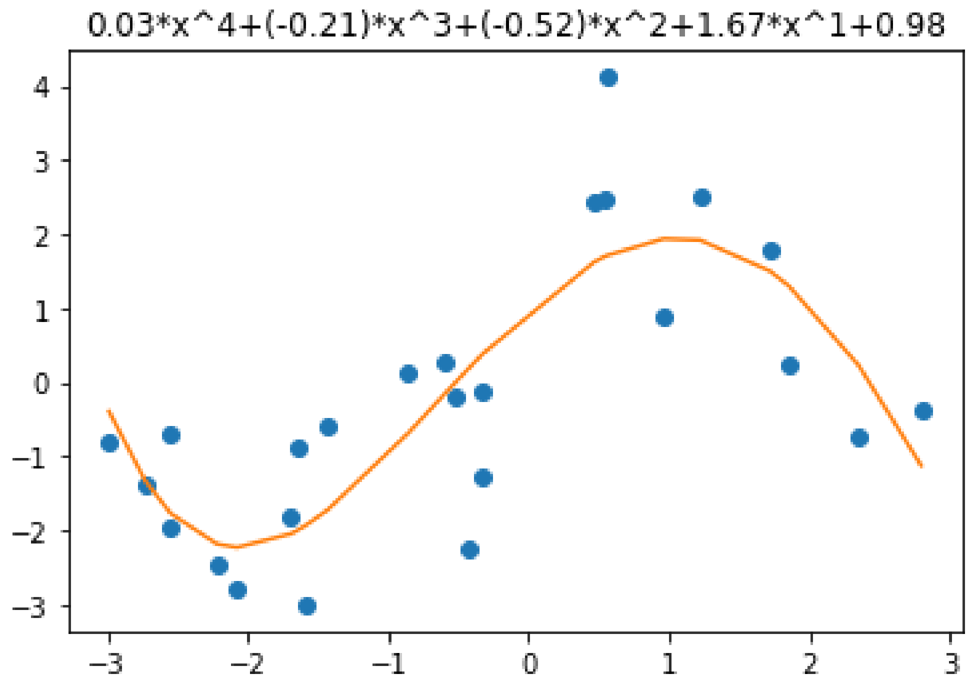

0.02*x^4+(-0.18)*x^3+(-0.46)*x^2+1.46*x^1+1.29
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


0.02*x^4+(-0.13)*x^3+(-0.4)*x^2+1.11*x^1+1.27
3
2
1
0
−1
−2
−1
0
1
2
3


0.08*x^4+(-0.24)*x^3+(-0.8)*x^2+1.74*x^1+0.93
3
2
1
0
−1
−2
−3
−4
−2
−1
0
1
2
3

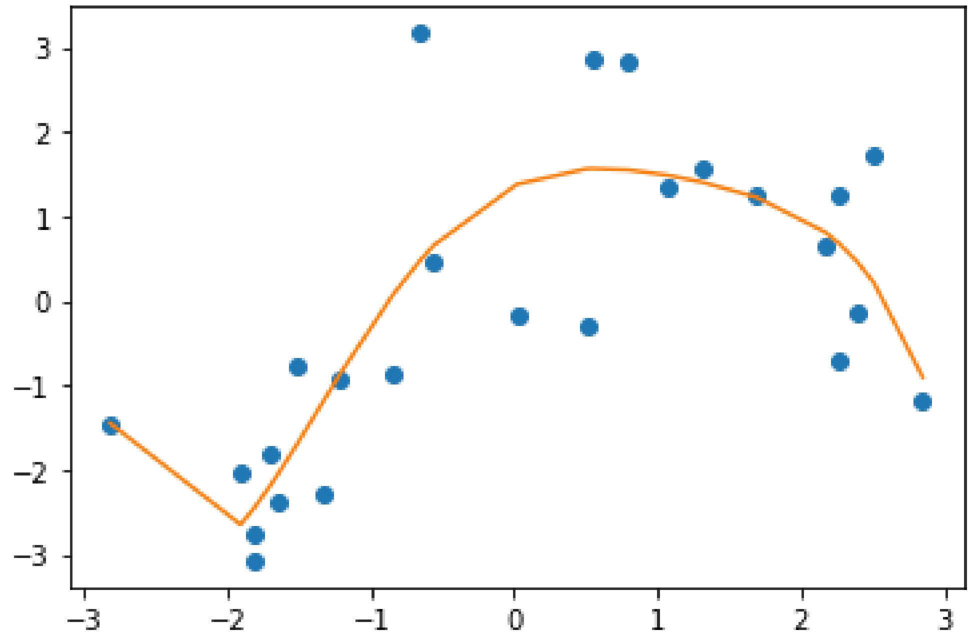

0.06*x^4+(-0.17)*x^3+(-0.77)*x^2+1.64*x^1+1.41
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3


0.09*x^4+(-0.12)*x^3+(-0.98)*x^2+1.1*x^1+1.69
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

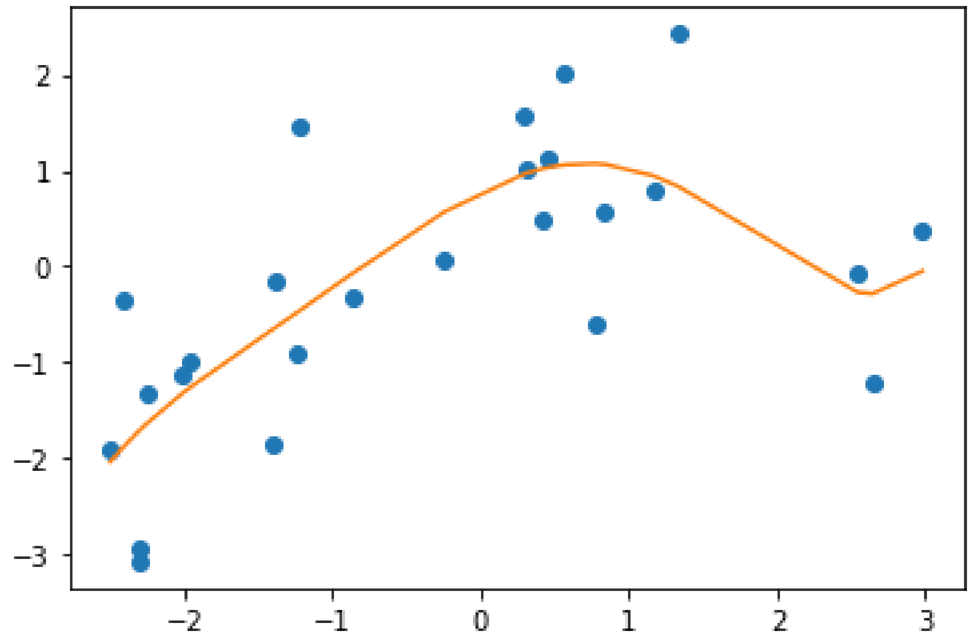

0.03*x^4+(-0.21)*x^3+(-0.52)*x^2+1.67*x^1+0.98
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

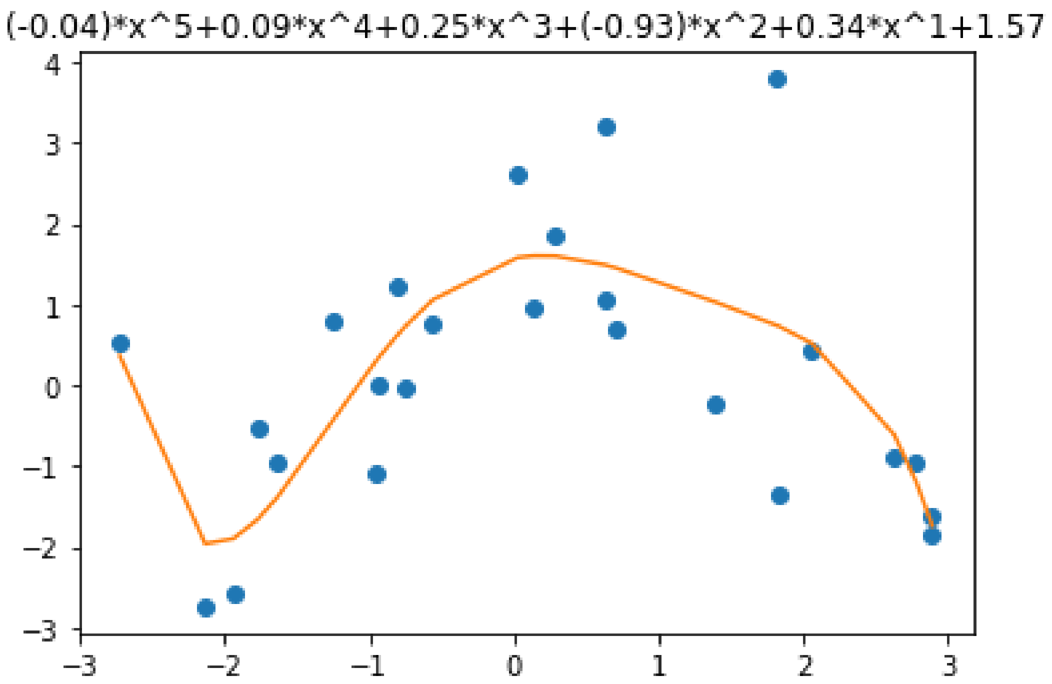
- Plots for Degree 5:

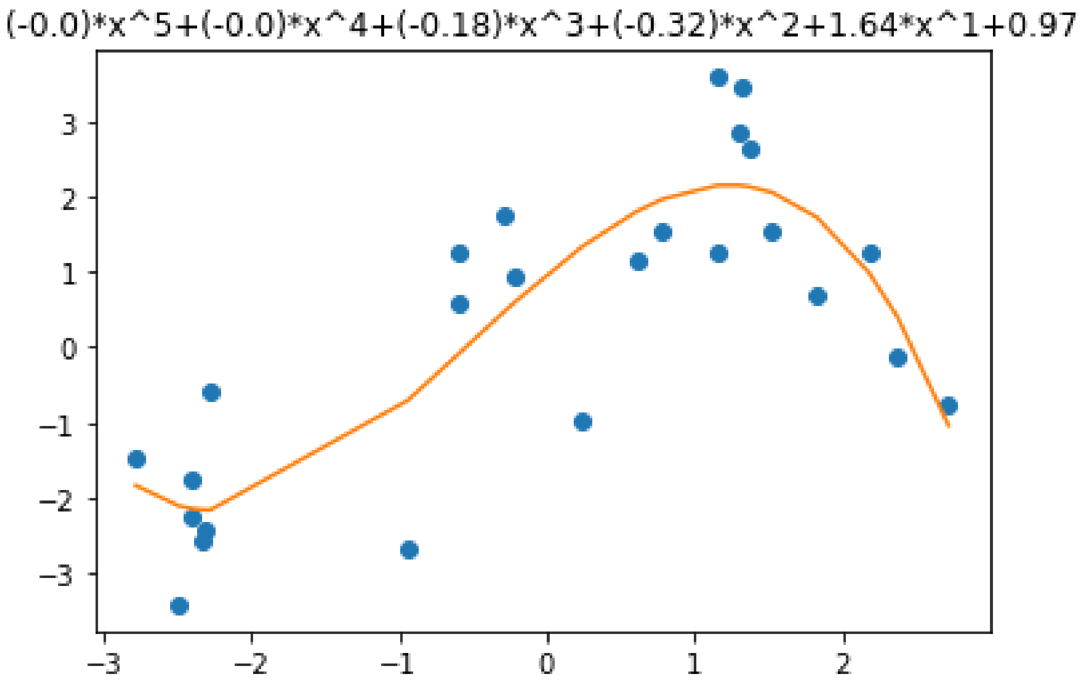
(-0.03)*x^5+0.06*x^4+0.18*x^3+(-0.82)*x^2+0.75*x^1+1.37

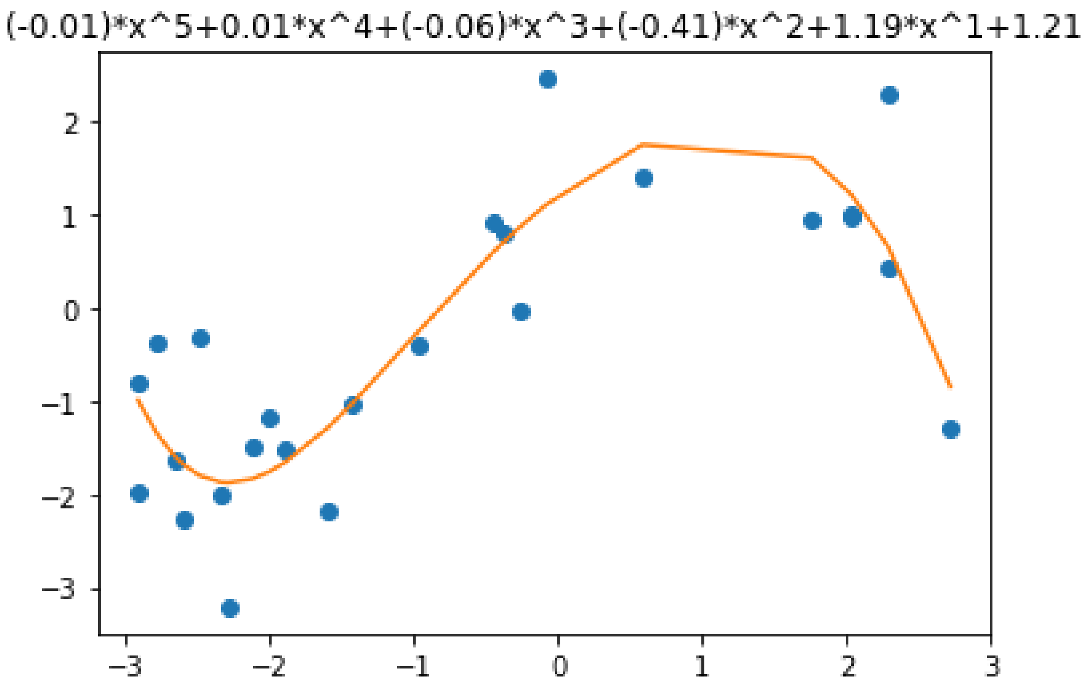


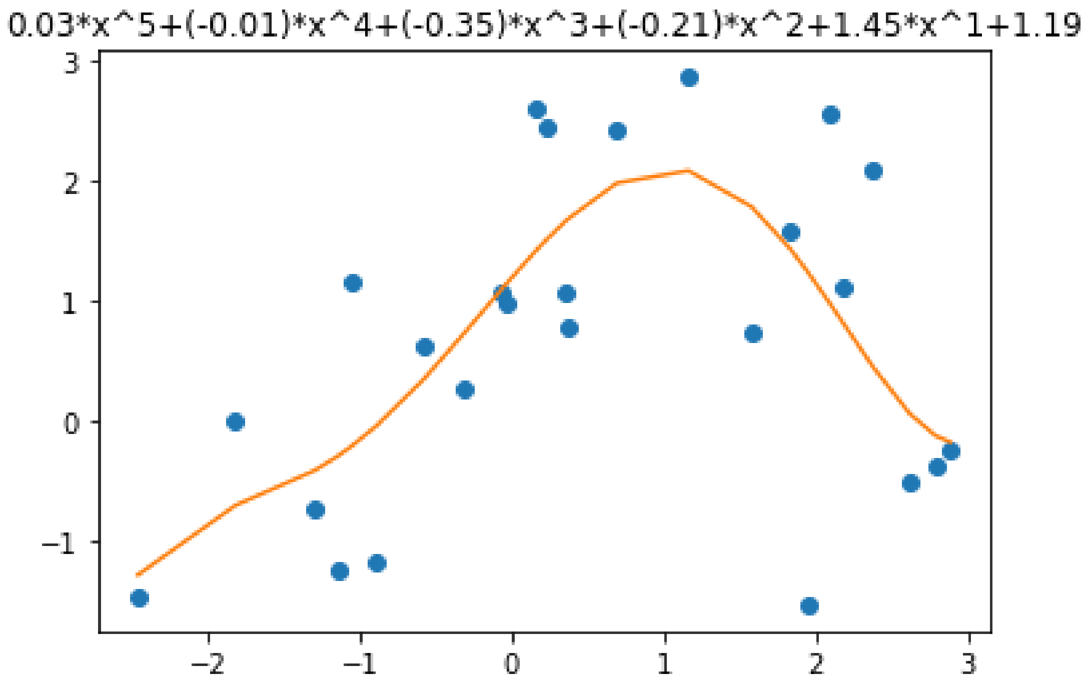
0.02*x^5+0.01*x^4+(-0.16)*x^3+(-0.39)*x^2+0.77*x^1+0.78

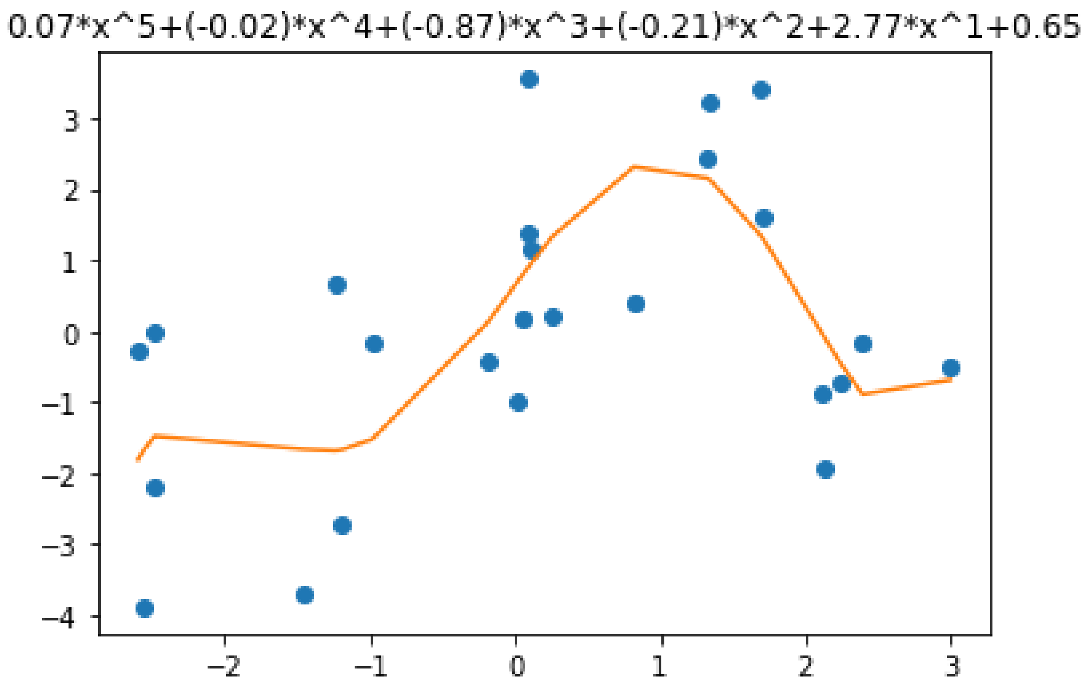


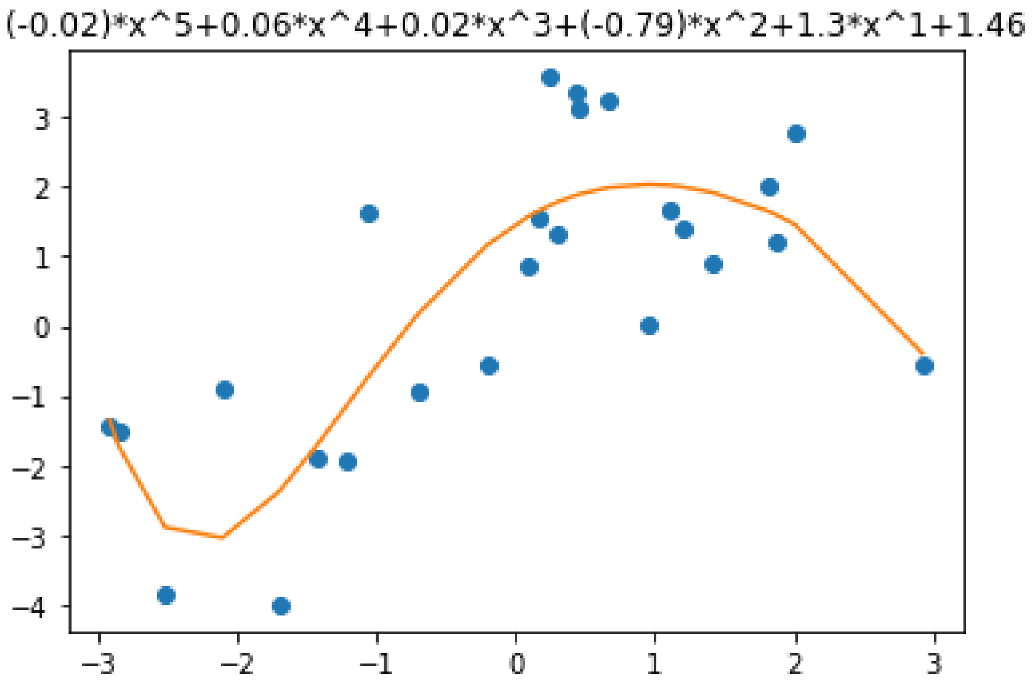

(-0.04)*x^5+0.09*x^4+0.25*x^3+(-0.93)*x^2+0.34*x^1+1.57
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


(-0.0)*x^5+(-0.0)*x^4+(-0.18)*x^3+(-0.32)*x^2+1.64*x^1+0.97
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2

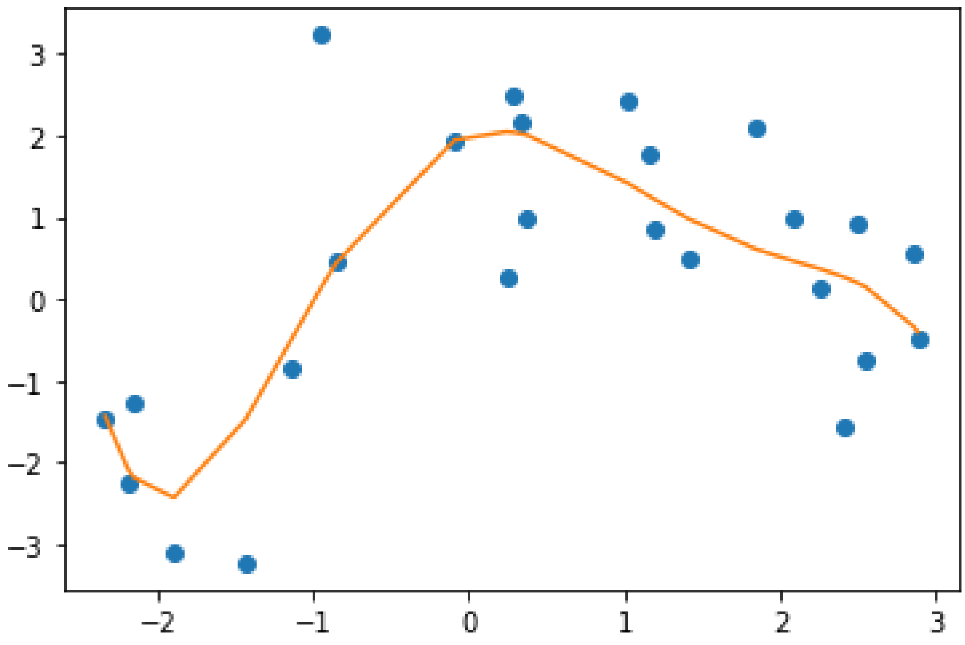

(-0.01)*x^5+0.01*x^4+(-0.06)*x^3+(-0.41)*x^2+1.19*x^1+1.21
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


0.03*x^5+(-0.01)*x^4+(-0.35)*x^3+(-0.21)*x^2+1.45*x^1+1.19
3
2
1
0
−1
−2
−1
0
1
2
3

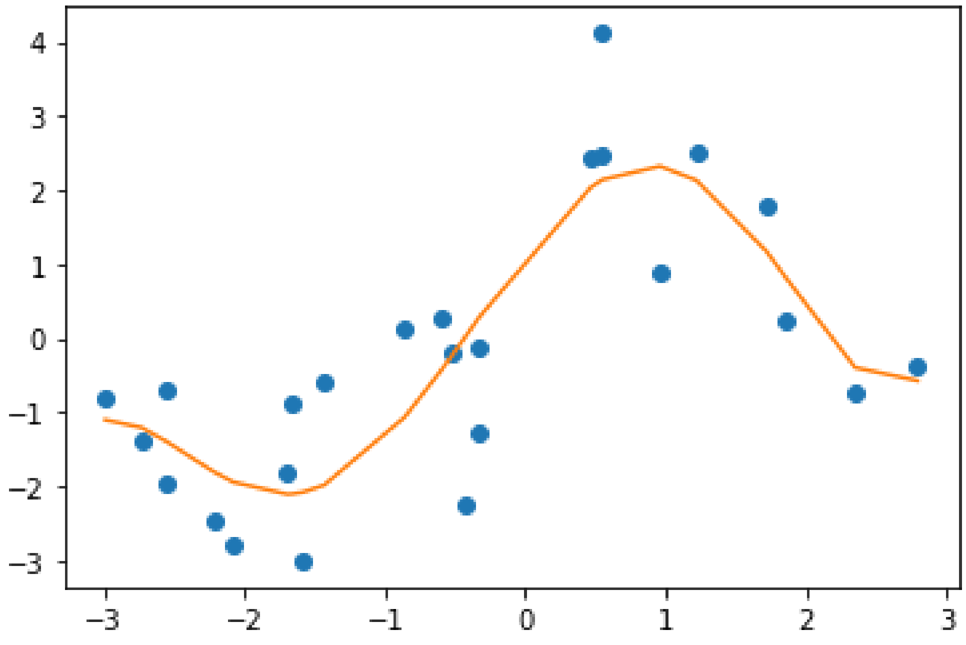

0.07*x^5+(-0.02)*x^4+(-0.87)*x^3+(-0.21)*x^2+2.77*x^1+0.65
3
2
1
0
−1
−2
−3
−4
−2
−1
0
1
2
3


(-0.02)*x^5+0.06*x^4+0.02*x^3+(-0.79)*x^2+1.3*x^1+1.46
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3

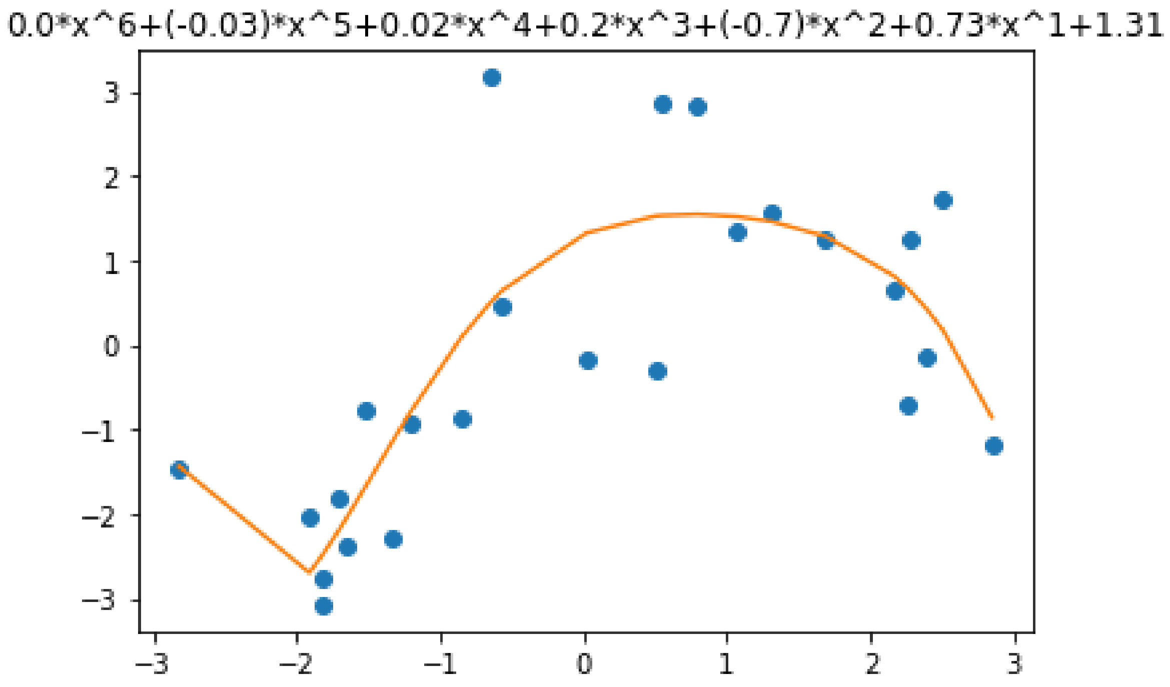
(-0.06)*x^5+0.18*x^4+0.28*x^3+(-1.48)*x^2+0.51*x^1+2.0

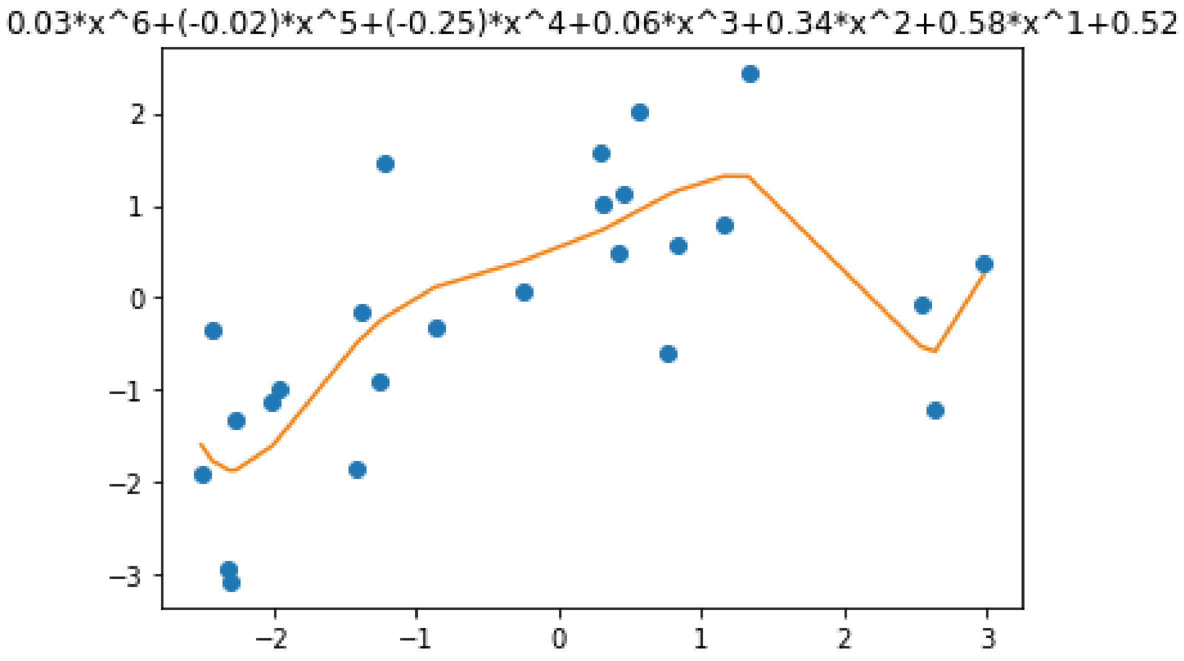


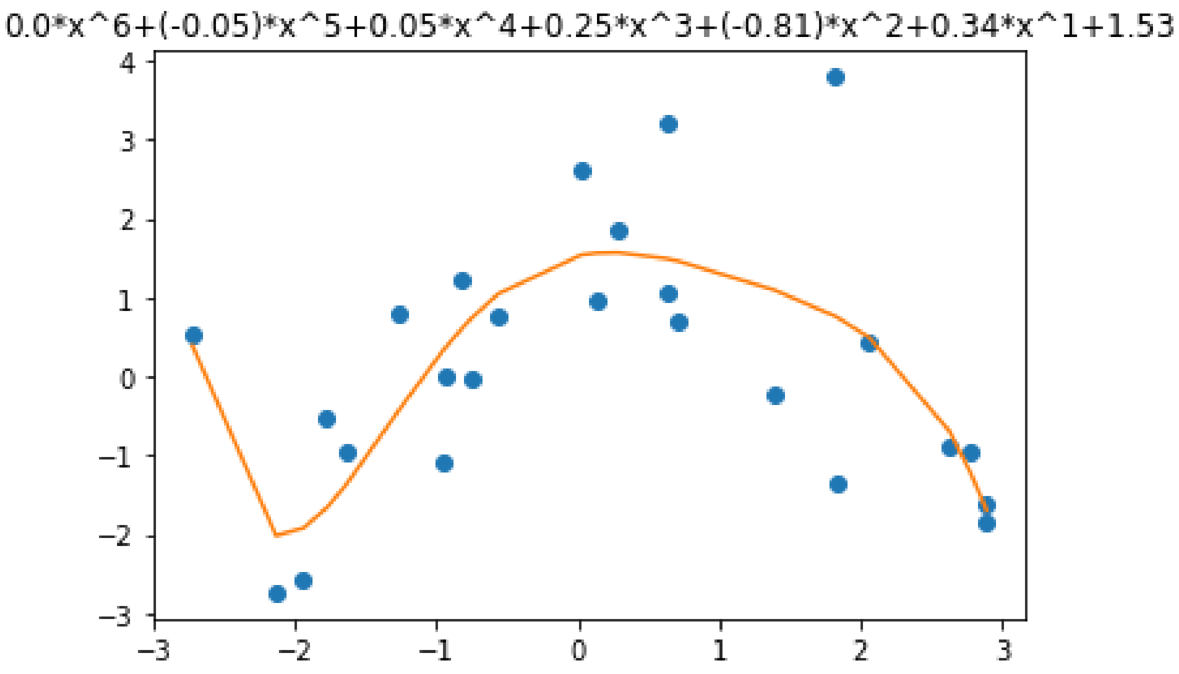
0.04*x^5+0.06*x^4+(-0.61)*x^3+(-0.71)*x^2+2.41*x^1+1.13

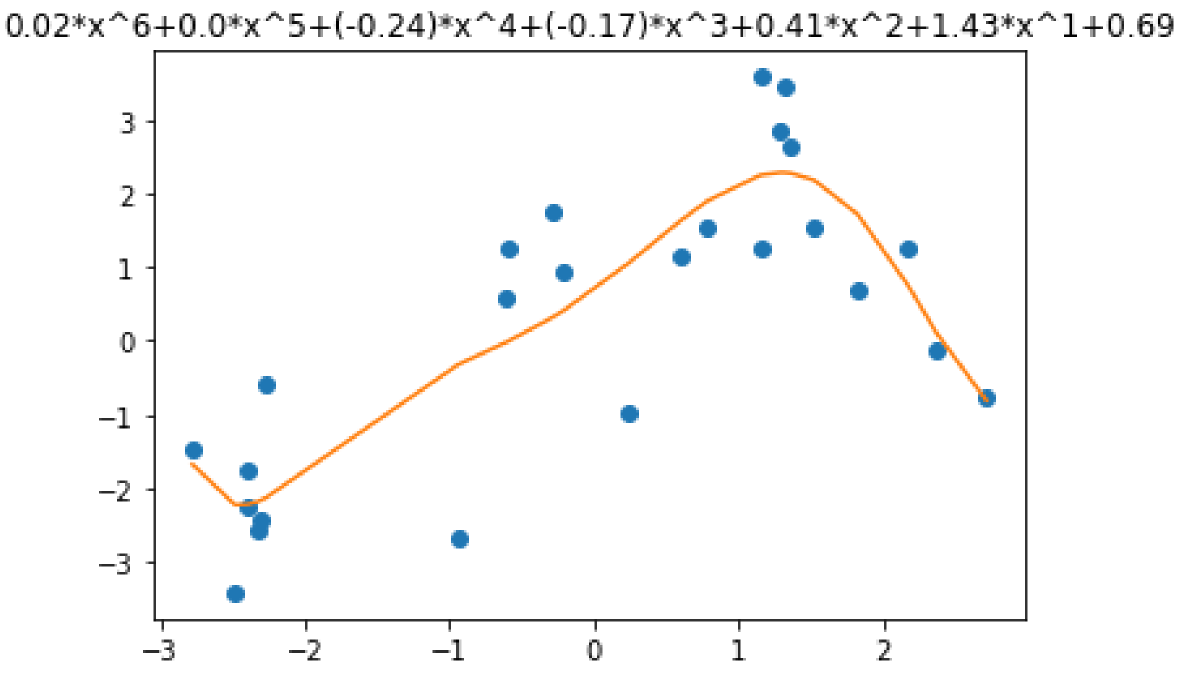


- Plots for Degree 6:

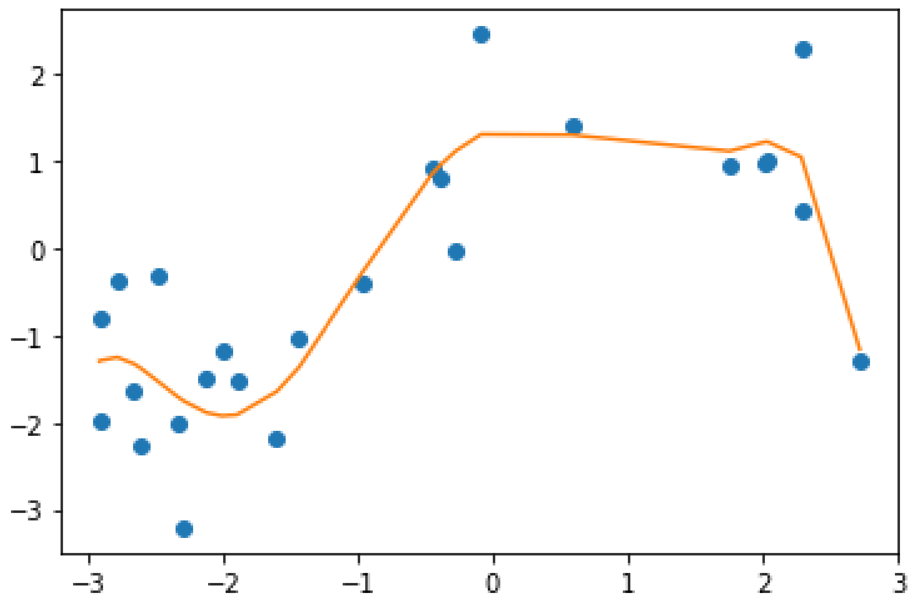




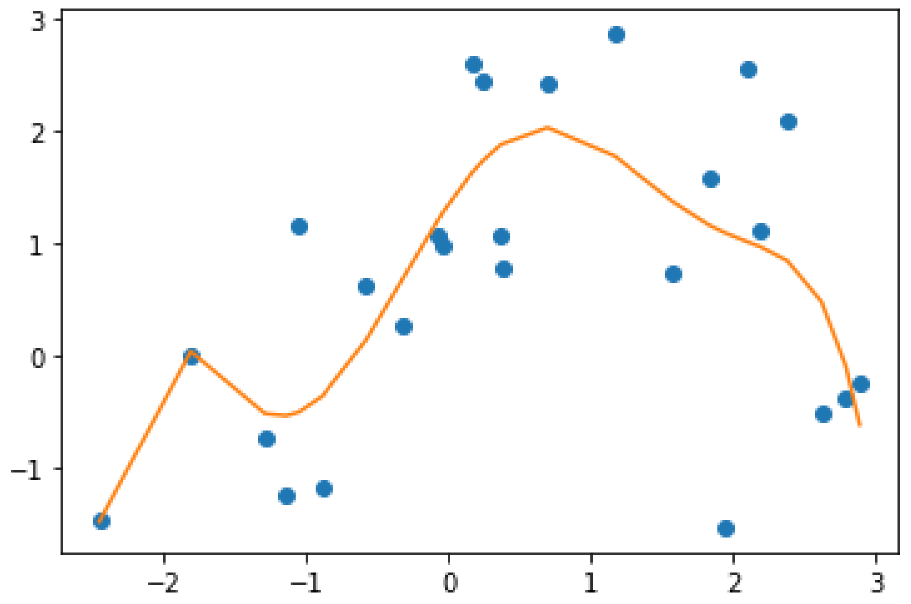

0.0*x^6+(-0.05)*x^5+0.05*x^4+0.25*x^3+(-0.81)*x^2+0.34*x^1+1.53
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


0.02*x^6+0.0*x^5+(-0.24)*x^4+(-0.17)*x^3+0.41*x^2+1.43*x^1+0.69
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2

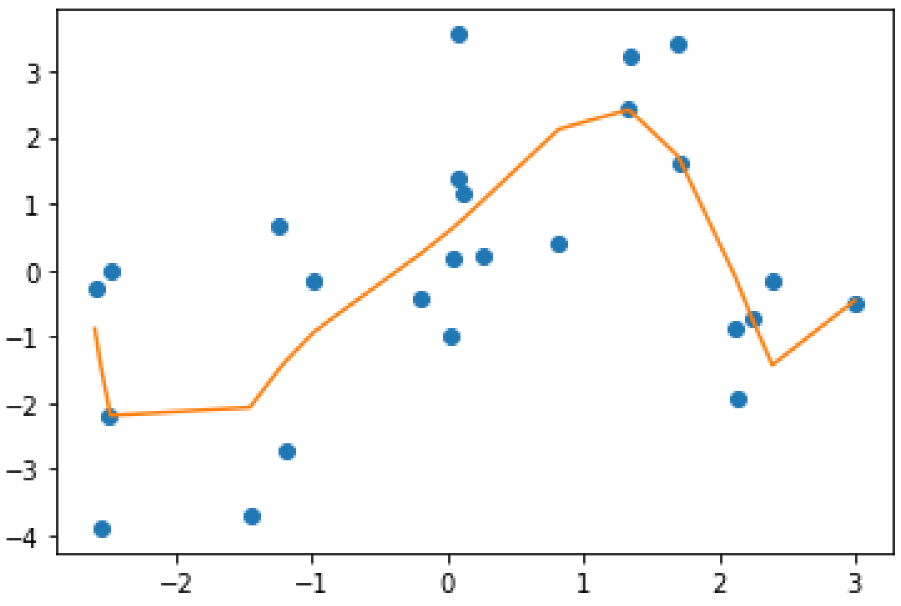
(-0.03)*x^6+(-0.04)*x^5+0.33*x^4+0.22*x^3+(-1.32)*x^2+0.53*x^1+1.36



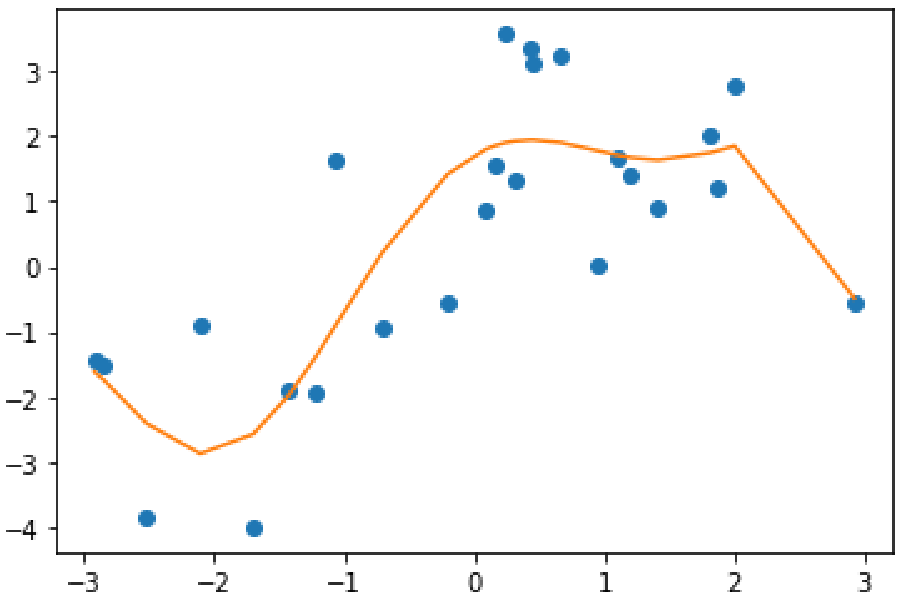
(-0.04)*x^6+0.09*x^5+0.34*x^4+(-0.75)*x^3+(-0.93)*x^2+1.85*x^1+1.35

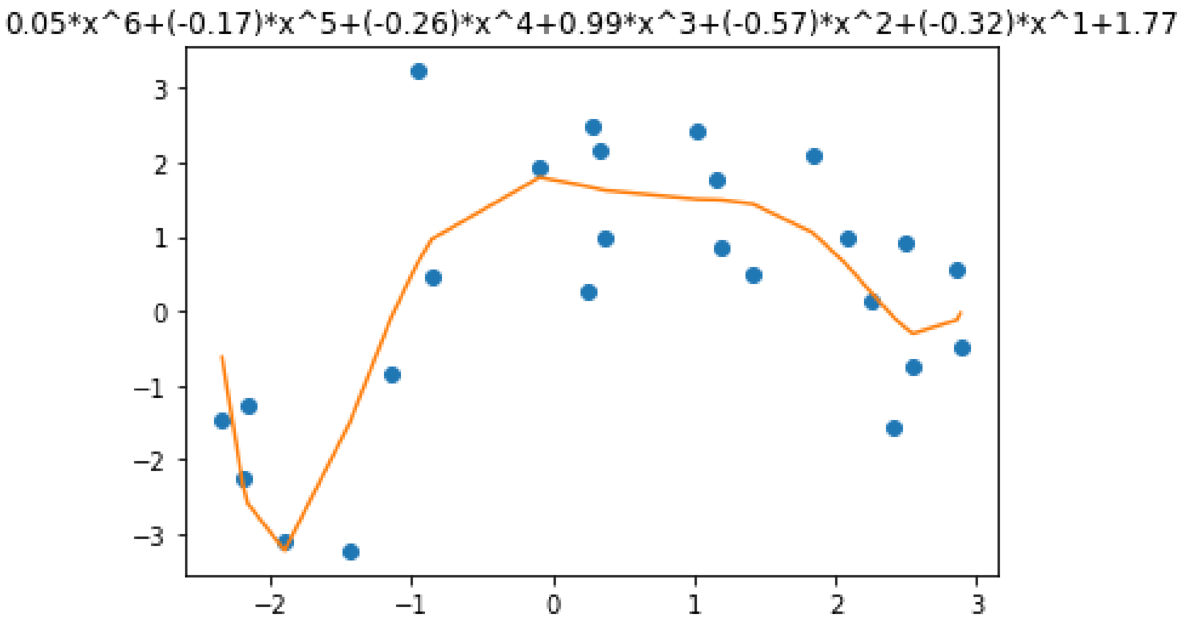


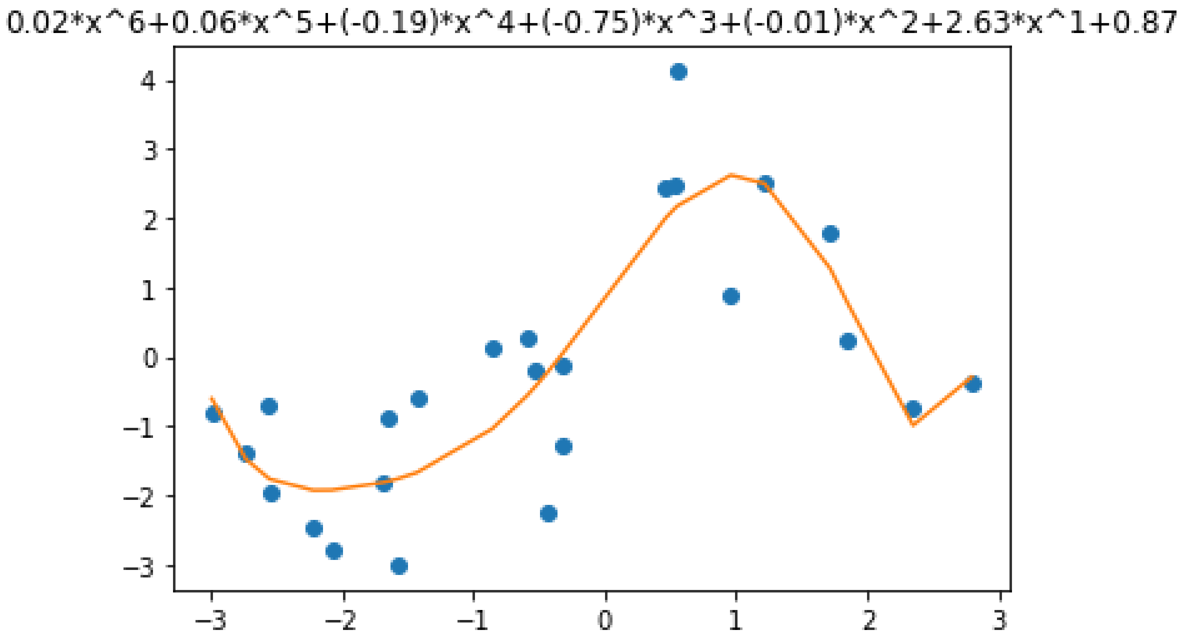
0.05*x^6+(-0.04)*x^5+(-0.47)*x^4+(-0.03)*x^3+0.55*x^2+1.72*x^1+0.57



(-0.02)*x^6+(-0.03)*x^5+0.31*x^4+0.12*x^3+(-1.47)*x^2+1.13*x^1+1.71







Finally, below you can find the average mean squared error values for each degree.

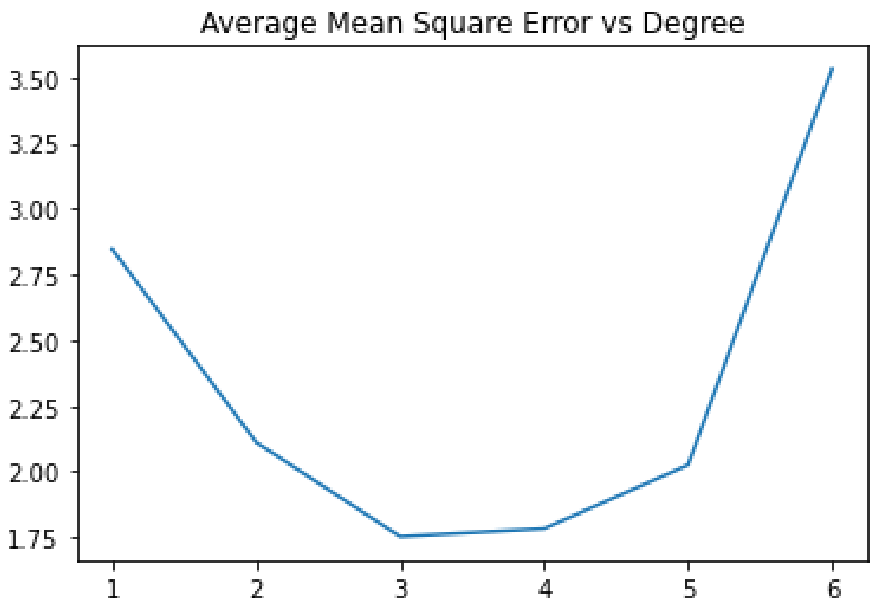


Figure [10]: Average Mean Square Error of Models with Different Degrees

As seen in Figure 10, the lowest error rate was found in the 3rd degree. This shows that 3rd degree models should be used in model selection. In addition, the hidden f(x) function that we mentioned at the beginning is probably a 3rd degree function.