8.Poynomial Regression From Scratch

October 26, 2021

Polynomial Regression (Non-Linear fitting)

It is not non-linear regression

0.1 Understanding curve fitting

[191]:
```
# Import necessary package
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

0.1.1 1. Degree 2 polynomial (quadratic function for parabola)

$$f(x) = ax^2 \tag{1}$$

Dataset

[192]:
```
# Generate values for variable x between -5 to +5 with 0.1 space
x = np.arange(-5.0, 5.0, 0.1)
x = np.round(x, 2)
x[:5,]
```

[192]:
```
array([-5. , -4.9, -4.8, -4.7, -4.6])
```

[193]:
```
# Calculate y value for each x using quadratic function (I assumed the value of␣
↪parameter a to be 1)
y_init =  1*(x**2)
y_init[:5,]
```

[193]:
```
array([25.  , 24.01, 23.04, 22.09, 21.16])
```

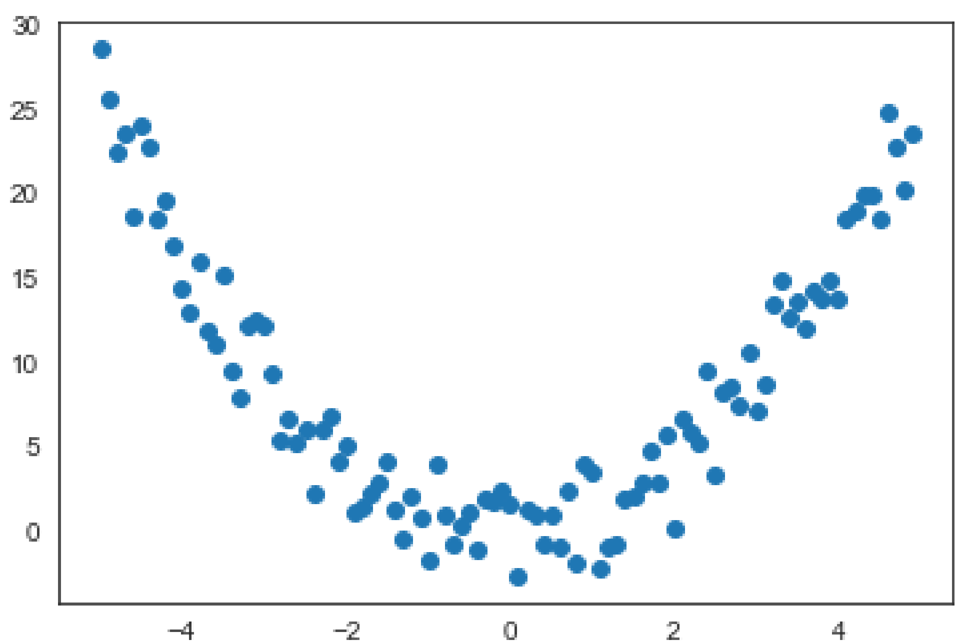
[194]:
```
# Adding some randomness to y value to show different non-linear pattern
y_noise = 2 * np.random.normal(size=x.size)
y = np.round(y_init + y_noise, 2)
```

[195]:
```
# Display the first 5 records
print("  X  ","   Y  ")
print("===============")
count = 0
for i,j in zip(x,y):
    if count == 5:
        break
    else:
        print(i," ",j)
        count = count + 1
```

```
  X      Y
===============
-5.0   28.56
-4.9   25.62
-4.8   22.5
-4.7   23.5
-4.6   18.69
```

Find the relationship

[196]:
```
plt.scatter(x,y)
plt.show()
```



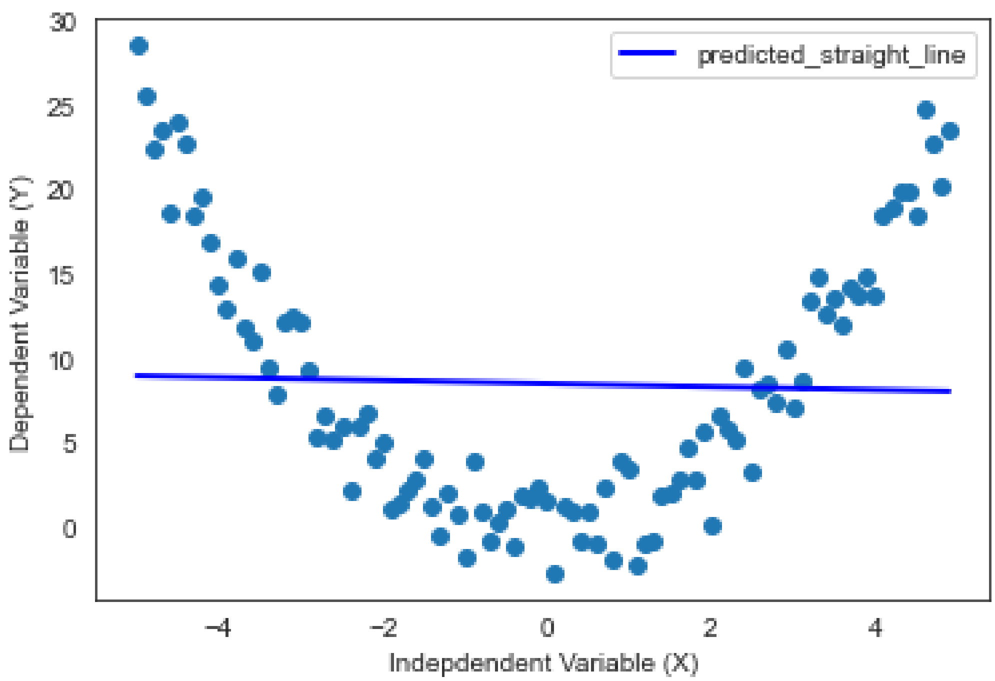
Linear model

[197]:
```
# Fit x and y with straight line
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x.reshape(-1, 1), y)
```

[197]: LinearRegression()

[198]:
```
# Predit value for X
y_pred = model.predict(x.reshape(-1, 1))
```

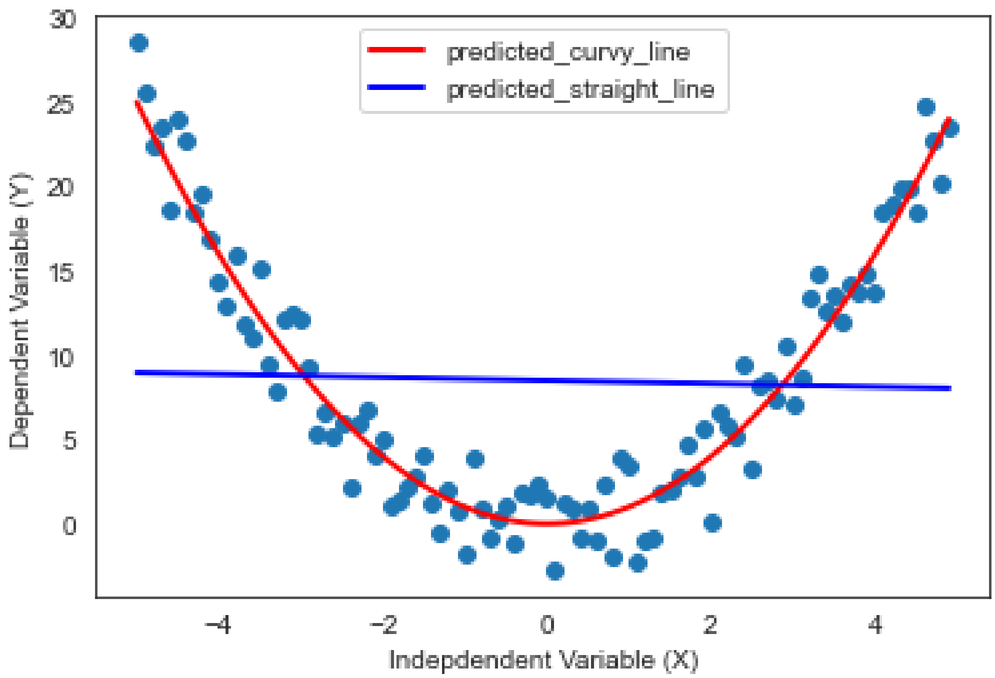
[199]:
```
# Plotting y_observed and y_predicted
plt.scatter(x,y)
plt.plot(x, y_pred, 'b',linewidth = 2, label="predicted_straight_line")
plt.ylabel('Dependent Variable (Y)')
plt.xlabel('Indepdendent Variable (X)')
plt.legend(loc="best")
plt.show()
```



Non-linear model Let us fit a suitable curvy line by estimating the values for parameters a, b, and c

I did not use any algorithm. But, my assumptions are a=1, b=0, c=0

[200]:
```
# We used y_init values as it was generated by using a=1, b=0, c=0
plt.scatter(x,y)
plt.plot(x, y_init, 'r',linewidth = 2,label="predicted_curvy_line")
plt.plot(x, y_pred, 'b',linewidth = 2, label="predicted_straight_line")
plt.ylabel('Dependent Variable (Y)')
plt.xlabel('Indepdendent Variable (X)')
plt.legend()
plt.show()
```



0.1.2 2. Degree 3 polynomial (cubic function)

$$f(x) = ax^3 + bx^2 + cx + d \tag{2}$$

Dataset

[201]:
```
# Generate values for variable x between -5 to +5 with 0.1 space
x = np.arange(-5.0, 5.0, 0.1)
x = np.round(x, 2)
x[:5,]
```

[201]: array([-5. , -4.9, -4.8, -4.7, -4.6])

[202]:
```
# Calculate y value for each x using cubic function
y_init =  1*(x**3) + 1*(x**2) + 1*x + 3
y_init[:5,]
```

[202]:
```
array([-102.    ,  -95.539,  -89.352,  -83.433,  -77.776])
```

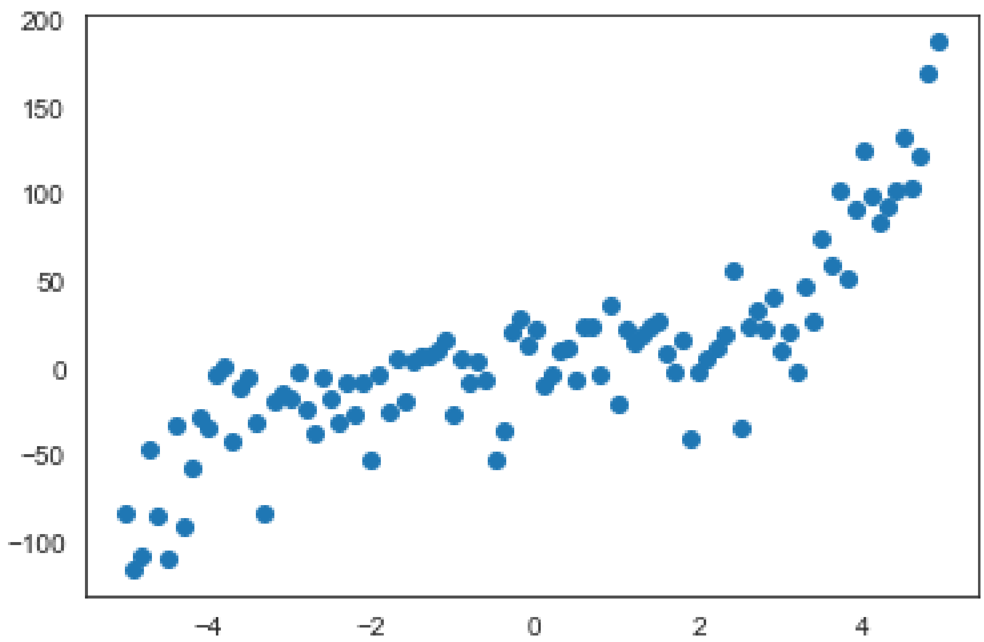
[203]:
```
# Adding some randomness to y value to show different non-linear pattern
y_noise = 20 * np.random.normal(size=x.size)
y = np.round(y_init + y_noise,2)
```

[204]:
```
# Display the first 5 records
print("  X  ","   Y  ")
print("===============")
count = 0
for i,j in zip(x,y):
    if count == 5:
        break
    else:
        print(i," ",j)
        count = count + 1
```

```
  X      Y
===============
-5.0   -83.69
-4.9   -115.39
-4.8   -107.21
-4.7   -46.42
-4.6   -84.99
```

Find the relationship

[205]:
```
plt.scatter(x,y)
plt.show()
```



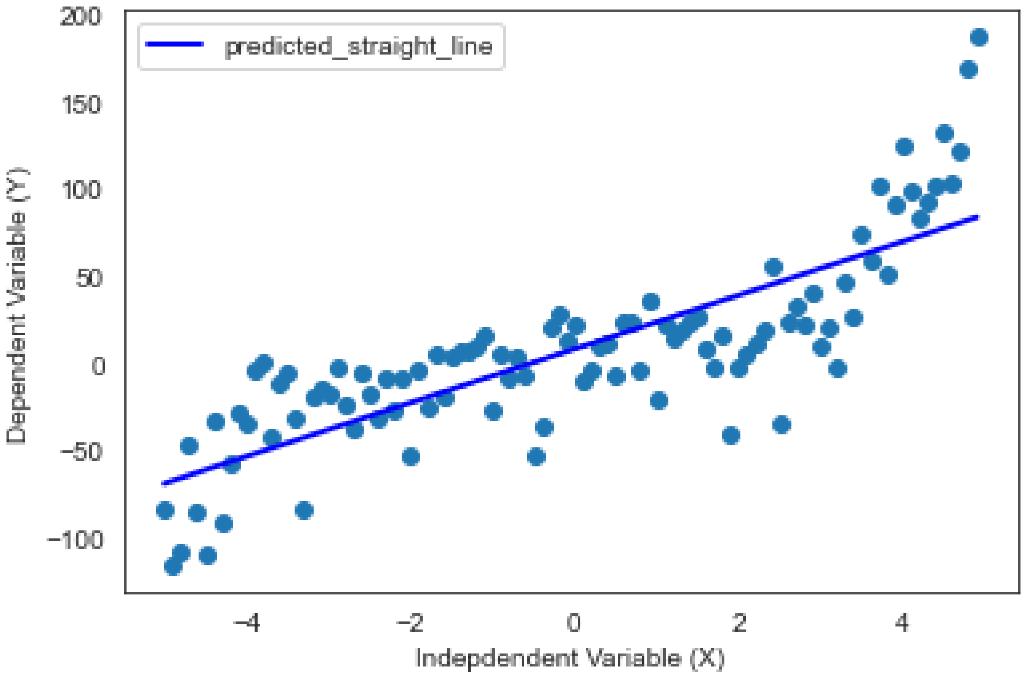
Linear model

[206]:
```
# Fit x and y with straight line model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x.reshape(-1, 1), y)
```

[206]:
```
LinearRegression()
```

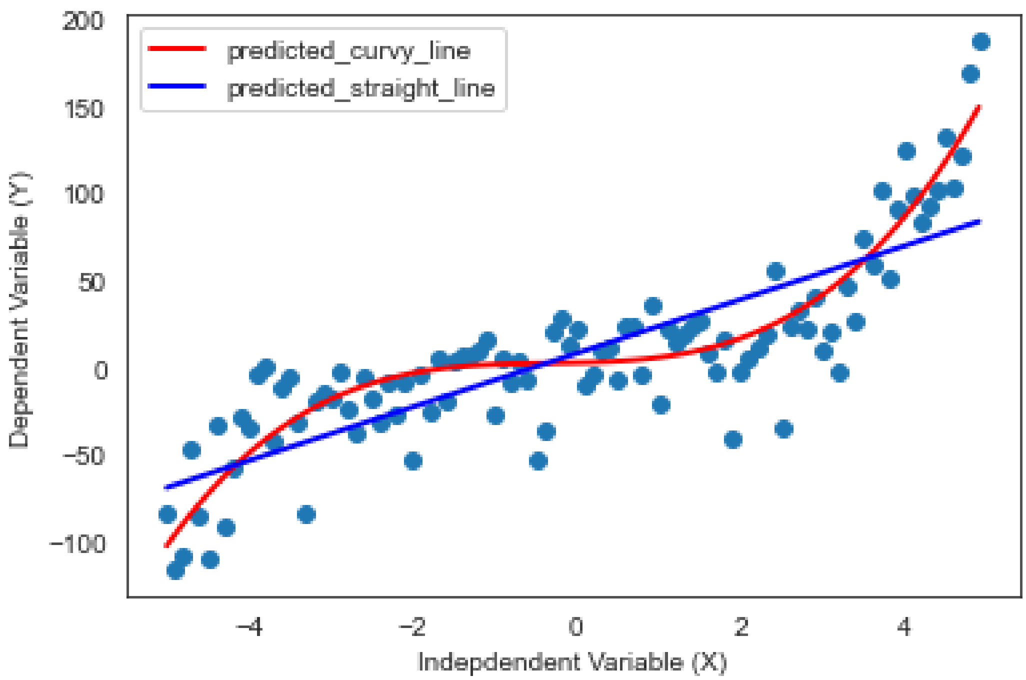
[207]:
```
# Predit value for X
y_pred = model.predict(x.reshape(-1, 1))
```

[208]:
```
# Plotting y_observed and y_predicted
plt.scatter(x,y)
plt.plot(x, y_pred, 'b',linewidth = 2, label="predicted_straight_line")
plt.ylabel('Dependent Variable (Y)')
plt.xlabel('Indepdendent Variable (X)')
plt.legend()
plt.show()
```



Non-linear model I did not use any algorithm. But, my assumptions are a =1, b=1, c=2, d=3

[209]:
```
# We used y_init values as it was generated by using a =1, b=1, c=2, d=3
plt.scatter(x,y)
plt.plot(x, y_init, 'r', linewidth = 2, label="predicted_curvy_line")
plt.plot(x, y_pred, 'b',linewidth = 2, label="predicted_straight_line")
plt.ylabel('Dependent Variable (Y)')
plt.xlabel('Indepdendent Variable (X)')
plt.legend()
plt.show()
```



0.1.3 3. Logistic (Sigmoid) function

$$\frac{1}{1+e^{-ax}} \tag{3}$$

Dataset

[210]:
```
# Generate values for variable x between -5 to +5 with 0.1 space
x = np.arange(-5, 5, 0.1)
x[:5,]
```

[210]:
```
array([-5. , -4.9, -4.8, -4.7, -4.6])
```

[211]:
```
# Calculate y value for each x using logistic function assuming the parameter␣
↪value a=1
y_init =  1/(1+np.exp(-1*x))
```

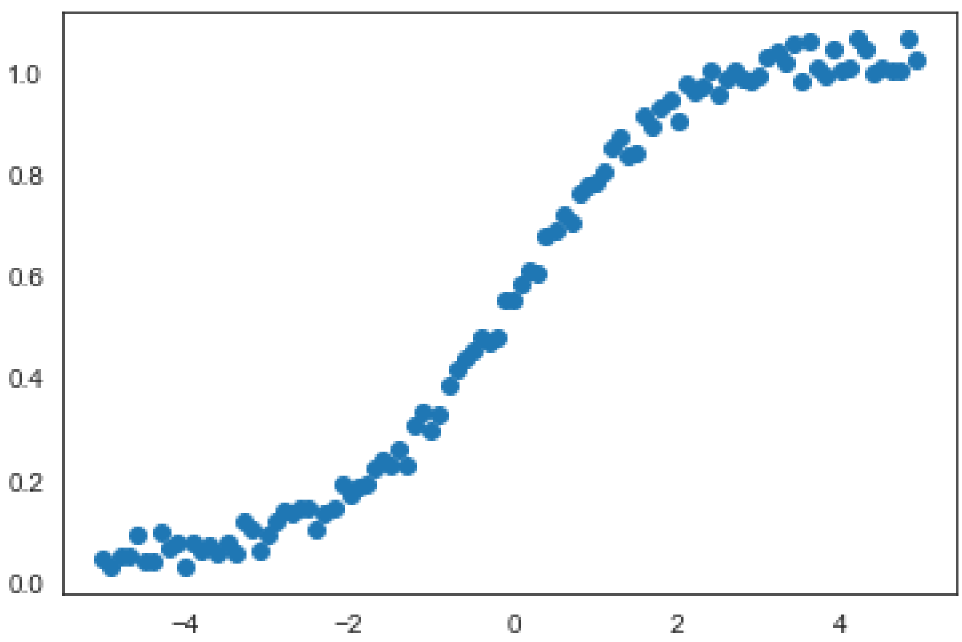
[212]:
```
# Adding some randomness to y value to show different non-linear pattern
y_noise = np.array([0.0001,0.0002,0.0004])
index = np.arange(x.size)
for i in index:
    y[i] = y_init[i] + np.random.uniform(0.01, 0.09)
```

[213]:
```
# Display the first 5 records
print("  X  ","   Y  ")
print("===============")
count = 0
for i,j in zip(x,y):
    if count == 5:
        break
    else:
        print(np.round(i,2)," ",np.round(j,2))
        count = count + 1
```

```
  X      Y
===============
-5.0   0.04
-4.9   0.03
-4.8   0.05
-4.7   0.05
-4.6   0.09
```

Find the relationship

[214]:
```
plt.scatter(x,y)
plt.show()
```



Linear model

[215]:

```
# Fit x and y with straight line model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x.reshape(-1, 1), y)
```

[215]:

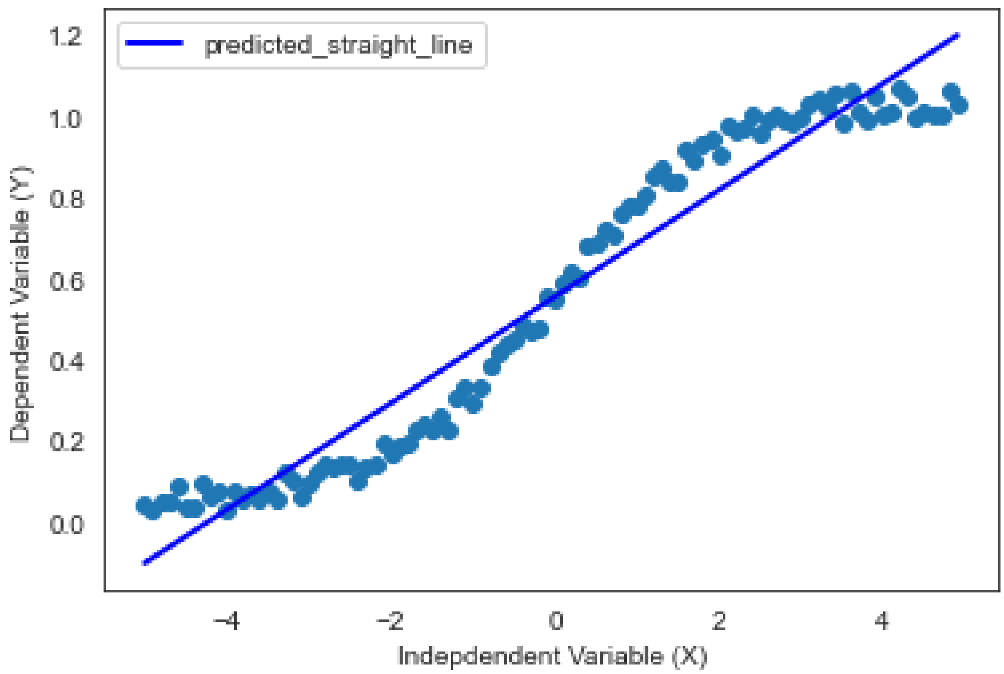
```
LinearRegression()
```

[216]:

```
# Predit value for X
y_pred = model.predict(x.reshape(-1, 1))
```

[217]:

```
# Plotting y_observed and y_predicted
plt.scatter(x,y)
plt.plot(x, y_pred, 'b',linewidth = 2, label="predicted_straight_line")
plt.ylabel('Dependent Variable (Y)')
plt.xlabel('Indepdendent Variable (X)')
plt.legend()
plt.show()
```

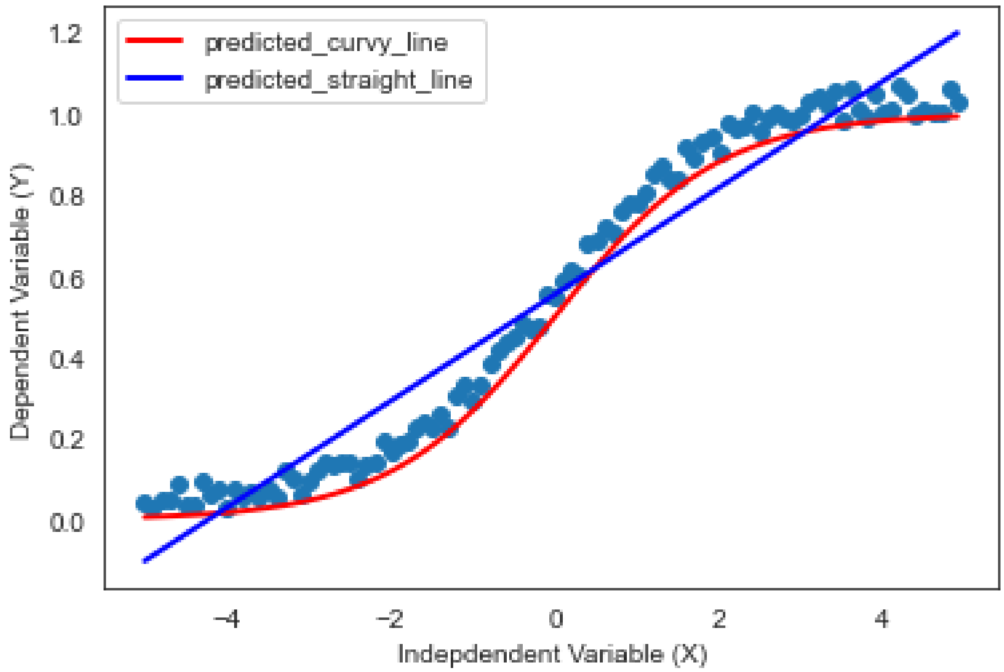


Non-linear model I did not use any algorithm. But, my assumptions are a =1, b=1, c=2, d=3

[218]:

```
# We used y_init values as it was generated by using a =1, b=1, c=2, d=3
plt.scatter(x,y)
plt.plot(x, y_init, 'r', linewidth = 2, label="predicted_curvy_line")
plt.plot(x, y_pred, 'b',linewidth = 2, label="predicted_straight_line")
```

```
plt.ylabel('Dependent Variable (Y)')
plt.xlabel('Indepdendent Variable (X)')
plt.legend()
plt.show()
```



Let us work on a real dataset without Sci-Kit Learn

0.1.4 Step 1: Load the dataset

[219]:
```
# Load the dataset into pandas dataframe
df=pd.read_csv("E:\\MY LECTURES\\DATA SCIENCE\\3.Programs\\dataset\\china_gdp.
↪csv")
# Change this location based on the location of dataset in your machine
```

[220]:
```
# Display the first five records
df.head()
```

[220]:

	Year	GDP
0	1960	5.918412e+10
1	1961	4.955705e+10
2	1962	4.668518e+10
3	1963	5.009730e+10
4	1964	5.906225e+10

Dataset shows Year and GDP (x 1000$).

GDP (output/dependent/target variable).

Year (input/independent/target variable).

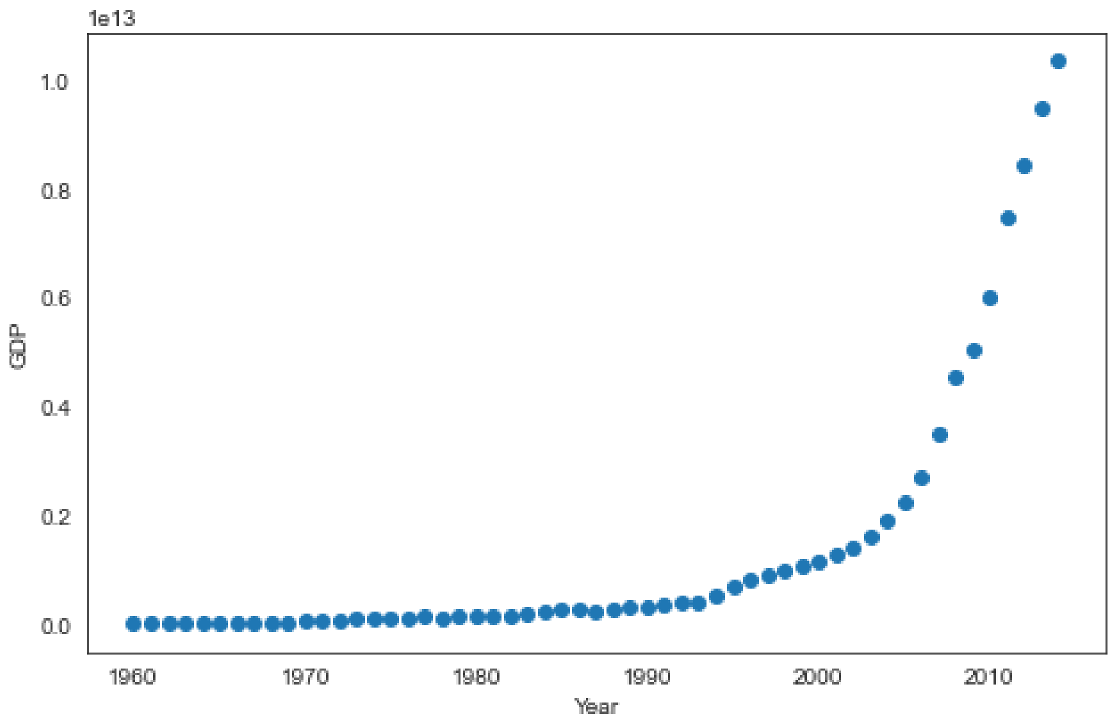
Row <=> record, tuple, instance, sample, observation, object, case, entity Column <=> attribute, variable, field, feature, characteristic, dimension

[221]:
```
df.shape
```

[221]: (55, 2)

0.1.5 Step 2: Apply EDA

[222]:
```
plt.figure(figsize=(8,5))
plt.scatter(df["Year"],df["GDP"])
plt.ylabel('GDP')
plt.xlabel('Year')
plt.show()
```



Seems like exponential and logistic function kind

0.1.6 Step 3. Pre-process and extract the features

if anything required

0.1.7 Step 4. Split the dataset into training and testing set

[223]:
```python
# Splitting dataset into training and testing set
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.2)
```

[224]:
```python
# sorting dataset based on year, else displaying graph is not nice
train = train.sort_values(by = ['Year'])
test = test.sort_values(by = ['Year'])
```

[225]:
```python
# Splitting training and testing set
x_train = train['Year']
y_train = train['GDP']
x_test = test['Year']
y_test = test['GDP']
```

[226]:
```python
print("Training data")
print("==========================")
print("  Year   ","   GDP   ")
print("==========================")
count = 0
for i,j in zip(x_train,y_train):
    if count == 5:
        break
    else:
        print(i," ",j)
        count = count + 1
```

```
Training data
==========================
  Year      GDP
==========================
1962   46685178504.0
1963   50097303271.0
1964   59062254890.0
1967   72057028560.0
1968   69993497892.0
```

[227]:
```python
print("Testing data")
print("==========================")
print("  Year   ","   GDP   ")
print("==========================")
```

```
count = 0
for i,j in zip(x_test,y_test):
    if count == 5:
        break
    else:
        print(i," ",j)
        count = count + 1
```

```
Testing data
==========================
  Year      GDP
==========================
1960   59184116489.0
1961   49557050183.0
1965   69709153115.0
1966   75879434776.0
1982   203550000000.0
```

0.1.8 Step 5: Training phase (bulding the model)

1. Planning to model with Logistic (sigmoid) function with guessed values for parameters From an initial look at the plot shown in EDA, we determine that the logistic (sigmoid) function could be a better approximation, since it has the property of starting with a slow growth, increasing growth in the middle, and then decreasing again at the end. The formula for logistic (sigmoid) is little bit modified to better fit the data points. It is given below.

$$\frac{1}{1+e^{\beta_1(X-\beta_2)}} \tag{4}$$

Logistic (sigmoid) function has two parameters β_1 and β_1 to estimate. For now, let us put some value and then see how it looks like with our dataset.

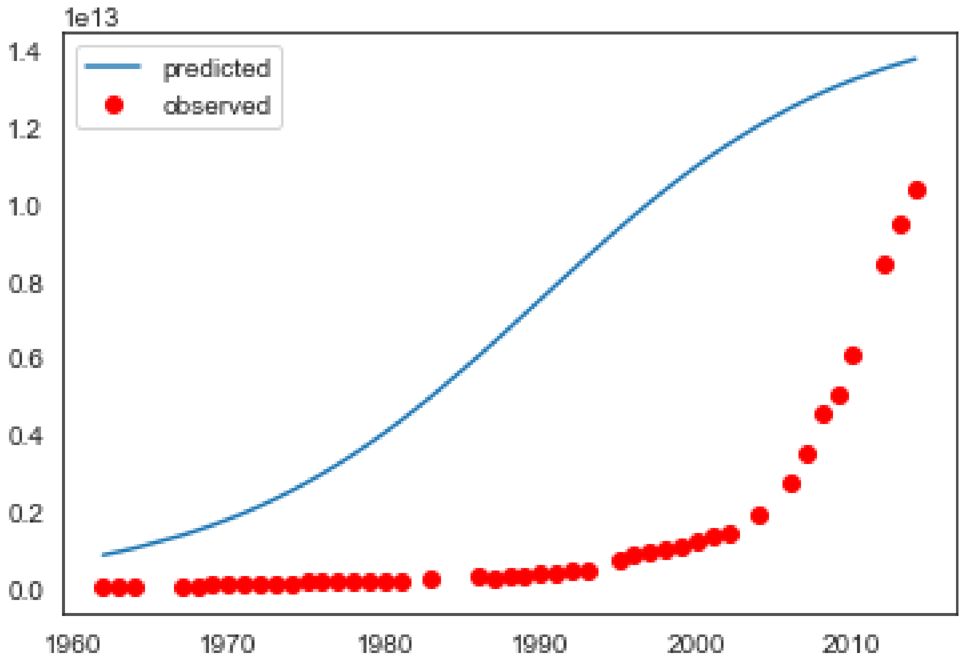
[228]:
```
# Sigmoid/logistic function
def sigmoid(val, beta_1, beta_2):
     output = 1 / (1 + np.exp(-beta_1*(val-beta_2)))
     return output
```

[229]:
```
# Assume (guess) beta_1 and beta_2
beta_1 = 0.10
beta_2 = 1990.0
```

[230]:
```
# Fitting the training data
y_train_pred = sigmoid(x_train, beta_1 , beta_2)
```

[231]:
```
# Plot initial prediction against datapoints
plt.plot(x_train, y_train_pred*15000000000000.,label = "predicted")
plt.plot(x_train, y_train, 'ro', label = "observed")
```

```
plt.legend()
plt.show()
```



2. Finiding the best parameter values with curve fitting function from library Lets first normalize our x_train and x_train to shrink between (0 and 1)

[232]:
```
x_train = x_train/max(x_train)
y_train = y_train/max(y_train)
```

How do we find the best parameters for beta1 and beta2 for logistic function? using curve fitting from scipy library.

[233]:
```
from scipy.optimize import curve_fit
parameters, pcov = curve_fit(sigmoid, x_train, y_train)
```

[234]:
```
parameters
```

[234]:
```
array([691.77941457,   0.99721494])
```

[235]:
```
# beta1 and beta2
beta1, beta2 = parameters
print("beta1 : ",beta1,"\nbeta2 : ",beta2)
```

```
beta1 :  691.7794145664701
beta2 :  0.99721493665803
```

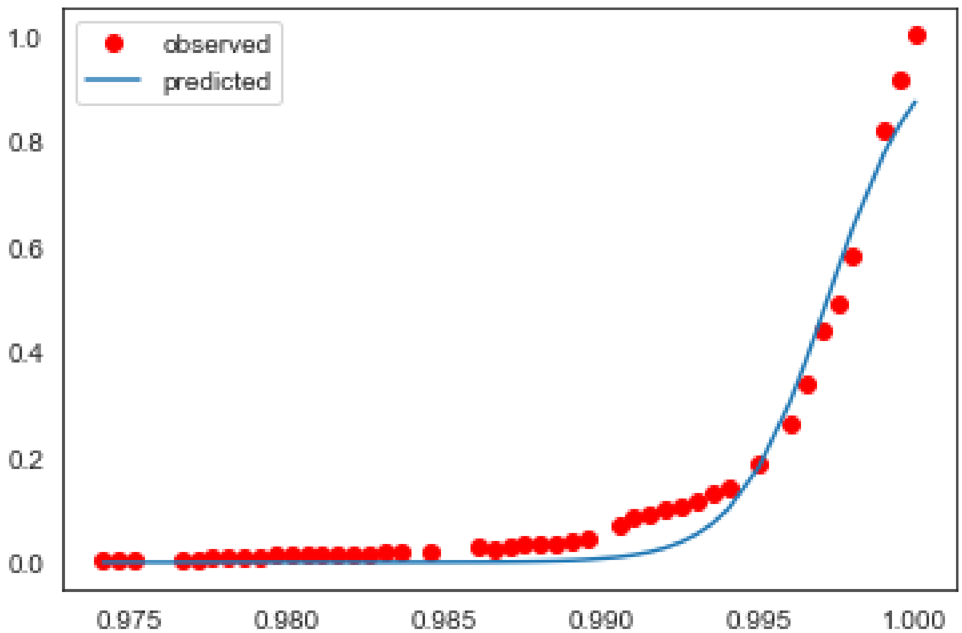
$$\frac{1}{1+e^{\beta_1(X-\beta_2)}} \tag{5}$$

With the estimated parameters values, let us predict the value for x_train

[236]:
```
y_train_pred = sigmoid(x_train, beta1, beta2)
```

Visualizing the model

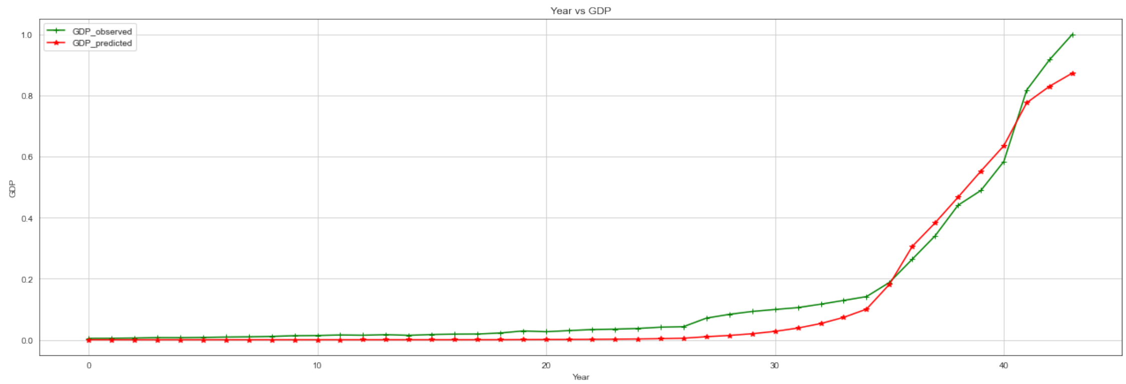
[237]:
```
plt.plot(x_train, y_train, 'ro', label = "observed")
plt.plot(x_train, y_train_pred, label = "predicted")
plt.legend()
plt.show()
```



Plotting observed GDP (x) and predicted GDP (y) for training set

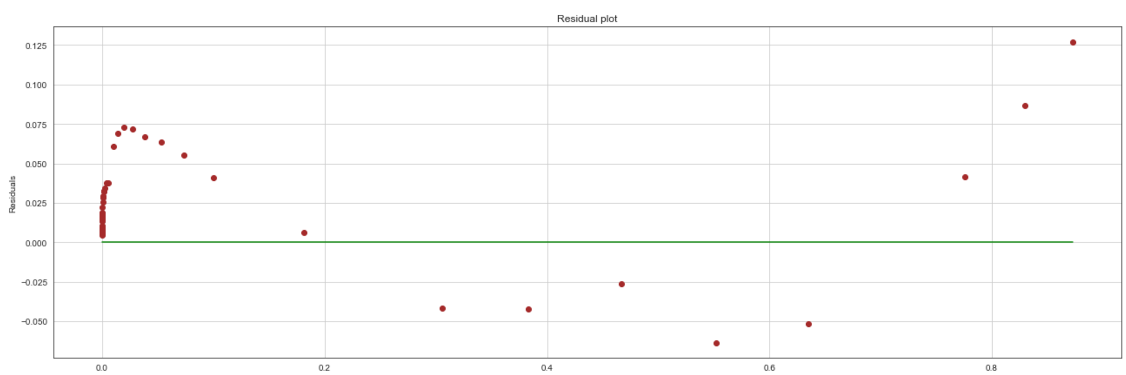
[238]:
```
# Predicting the Test set results
x = np.arange(len(y_train_pred))
fig = plt.figure(figsize=(22,7))
plt.plot(x,y_train,"g-+",label="GDP_observed")
plt.plot(x,y_train_pred,"r-*",label="GDP_predicted")
plt.grid(b=None)
plt.xlabel("Year")
plt.ylabel("GDP")
plt.title("Year vs GDP")
```

```
plt.legend()
plt.show()
```



Residual (Error) plot Unlike regression, where all data points in the residual plot are near around horizontal line, data points in residual plot for polynomial regression must be around the shape of respective polynomial function.

[239]:
```
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
residuals = y_train-y_train_pred
zeros = y_train-y_train
plt.scatter(y_train_pred,residuals,color="brown")
plt.grid(b=None)
plt.plot(y_train_pred,zeros,"g")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual plot")
plt.show()
```



0.1.9 Different error calculations to asses the model for training set

1. Sum of Squared Error (SSE)

$$SSE(m,b) = \sum_{i=1}^{n}(y_i - \hat{y})^2 = \sum_{i=1}^{n}(y_i - (m * x_i + b))^2 \tag{6}$$

[240]:
```
temp1 = np.square(residuals)
Train_SSE = np.round(np.sum(temp1),2)
print("Sum of Squared Error (SSE) :",Train_SSE)
```

```
Sum of Squared Error (SSE) : 0.08
```

2. Mean Squared Error (MSE)

$$MSE(m,b) = \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n} = \frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n} \tag{7}$$

[241]:
```
n = len(x_train)
Train_MSE = np.round(Train_SSE/n,2)
print("Mean Squared Error (MSE) :",Train_MSE)
```

```
Mean Squared Error (MSE) : 0.0
```

3. Root Mean Squared Error (RMSE)

$$RMSE(m,b) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n}} \tag{8}$$

[242]:
```
Train_RMSE = np.round(np.sqrt(Train_MSE),2)
print("Root Mean Squared Error (RMSE) :",Train_RMSE)
```

```
Root Mean Squared Error (RMSE) : 0.0
```

4. Mean Absolute Error (MAE)

$$MAE(m,b) = \frac{\sum_{i=1}^{n}|(y_i - \hat{y})|}{n} \tag{9}$$

[243]:
```
n = len(x_train)
temp1 = np.abs(residuals)
sum = np.sum(residuals)
Train_MAE = np.round(sum/n,2)
print("Mean Absolute Error (MAE) :",Train_MAE)
```

```
Mean Absolute Error (MAE) : 0.02
```

5. Mean Absolute Percentage Error (MAPE)

$$MAPE(m,b) = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - \hat{y})}{y_i}\right| = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - (m * x_i + b))}{y_i}\right| \tag{10}$$

[244]:
```
n = len(x_train)
temp1 = residuals/y_train
temp2 = np.abs(temp1)
sum = np.sum(temp2)
Train_MAPE = np.round(sum/n,2)
print("Mean Absolute Percentage Error (MAPE) :",Train_MAPE)
```

```
Mean Absolute Percentage Error (MAPE) : 0.74
```

0.1.10 Calculating R-Squred value (goodness of model) using SSE

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{11}$$

[245]:
```
from sklearn.metrics import r2_score
out = r2_score(y_train,y_train_pred)
Train_RS = round(out,2)*100
print("R-Squred value (goodness of model) for training set :",Train_RS,"%")
```

```
R-Squred value (goodness of model) for training set : 97.0 %
```

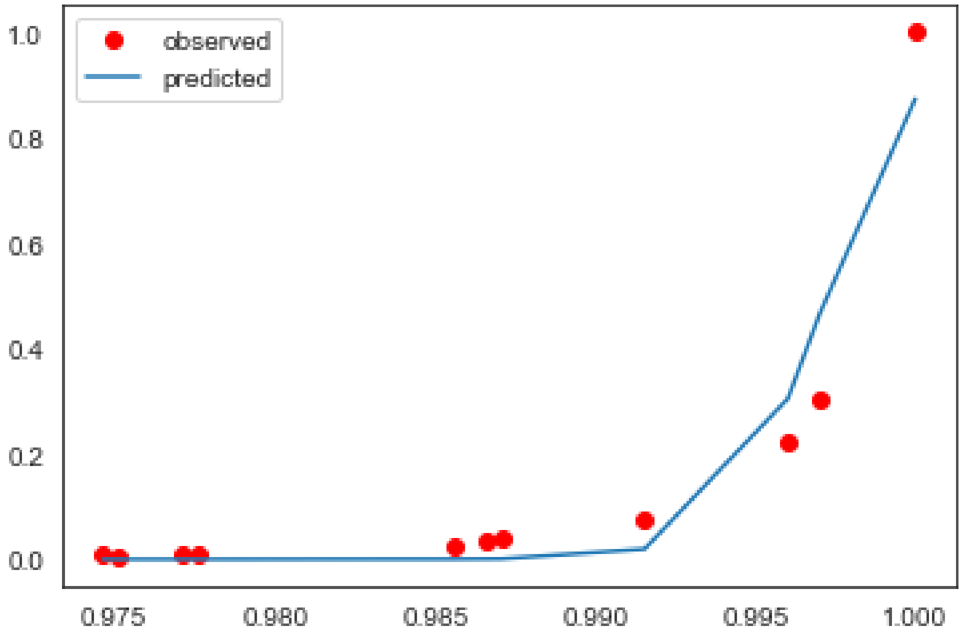
0.1.11 Step 6: Testing phase

[246]:
```
x_test = x_test/max(x_test)
y_test = y_test/max(y_test)
```

[247]:
```
y_test_pred = sigmoid(x_test, beta1, beta2)
```

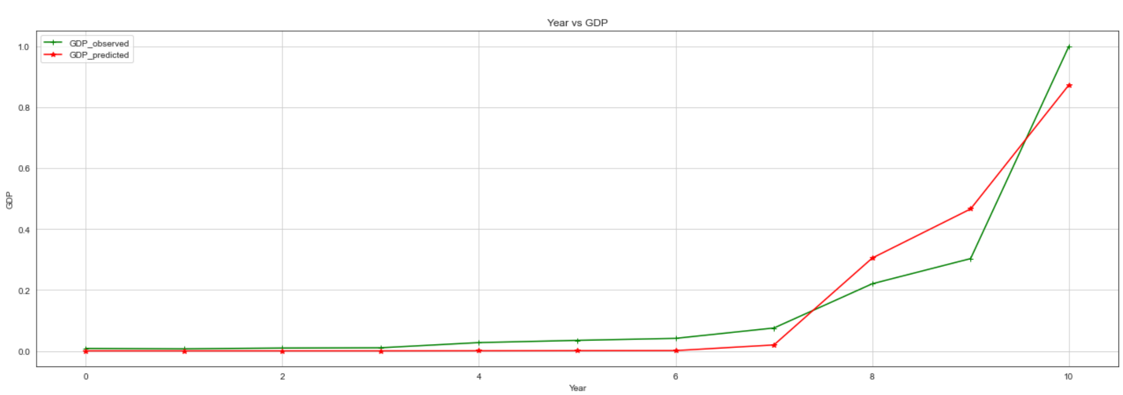
Visualizing the model

[248]:
```
plt.plot(x_test, y_test, 'ro', label = "observed")
plt.plot(x_test, y_test_pred, label = "predicted")
plt.legend()
plt.show()
```



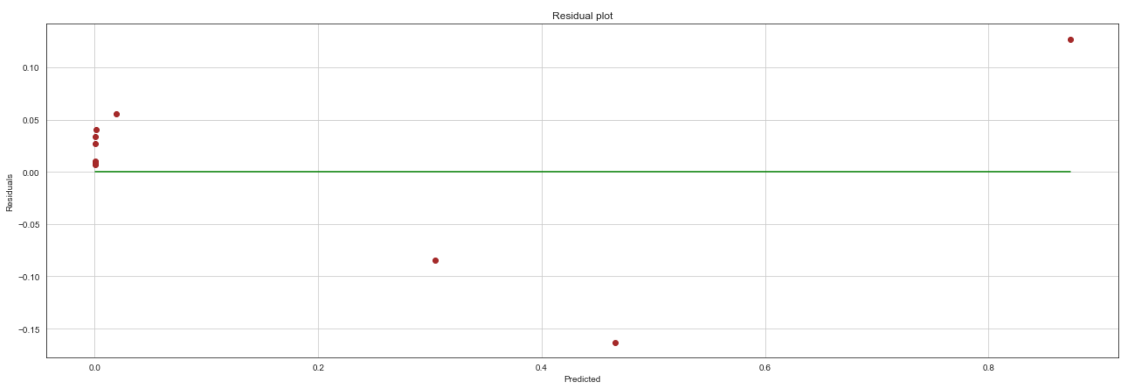
Plotting observed GDP (x) and predicted GDP (y) for training set

[249]:
```
x = np.arange(len(y_test_pred))
fig = plt.figure(figsize=(22,7))
plt.plot(x,y_test,"g-+",label="GDP_observed")
plt.plot(x,y_test_pred,"r-*",label="GDP_predicted")
plt.grid(b=None)
plt.xlabel("Year")
plt.ylabel("GDP")
plt.title("Year vs GDP")
plt.legend()
plt.show()
```



Residual (Error) plot Unlike regression, where all data points in the residual plot are near around horizontal line, data points in residual plot for polynomial regression must be around the shape of respective polynomial function.

[250]:
```
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
residuals = y_test-y_test_pred
zeros = y_test-y_test
plt.scatter(y_test_pred,residuals,color="brown")
plt.grid(b=None)
plt.plot(y_test_pred,zeros,"g")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual plot")
plt.show()
```



0.1.12 Different error calculations to asses the model for the test set

1. Sum of Squared Error (SSE)

$$SSE(m,b) = \sum_{i=1}^{n}(y_i - \hat{y})^2 = \sum_{i=1}^{n}(y_i - (m * x_i + b))^2 \tag{12}$$

[251]:
```
temp1 = np.square(residuals)
Test_SSE = np.round(np.sum(temp1),2)
print("Sum of Squared Error (SSE) :",Test_SSE)
```

```
Sum of Squared Error (SSE) : 0.06
```

2. Mean Squared Error (MSE)

$$MSE(m,b) = \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n} = \frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n} \tag{13}$$

[252]:
```
n = len(x_test)
Test_MSE = np.round(Test_SSE/n,2)
print("Mean Squared Error (MSE) :",Test_MSE)
```

```
Mean Squared Error (MSE) : 0.01
```

3. Root Mean Squared Error (RMSE)

$$RMSE(m,b) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n}} \tag{14}$$

[253]:
```
Test_RMSE = np.round(np.sqrt(Test_MSE),2)
print("Root Mean Squared Error (RMSE) :",Test_RMSE)
```

```
Root Mean Squared Error (RMSE) : 0.1
```

4. Mean Absolute Error (MAE)

$$MAE(m,b) = \frac{\sum_{i=1}^{n}|(y_i - \hat{y})|}{n} \tag{15}$$

[254]:
```
n = len(x_test)
temp1 = np.abs(residuals)
sum = np.sum(residuals)
Test_MAE = np.round(sum/n,2)
print("Mean Absolute Error (MAE) :",Test_MAE)
```

```
Mean Absolute Error (MAE) : 0.01
```

5. Mean Absolute Percentage Error (MAPE)

$$MAPE(m,b) = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - \hat{y})}{y_i}\right| = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - (m * x_i + b))}{y_i}\right| \tag{16}$$

[255]:
```
n = len(x_test)
temp1 = residuals/y_test
temp2 = np.abs(temp1)
sum = np.sum(temp2)
Test_MAPE = np.round(sum/n,2)
print("Mean Absolute Percentage Error (MAPE) :",Test_MAPE)
```

```
Mean Absolute Percentage Error (MAPE) : 0.79
```

0.1.13 Calculating R-Squred value (goodness of model) using SSE

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{17}$$

[256]:

```
from sklearn.metrics import r2_score
out = r2_score(y_test,y_test_pred)
Test_RS = round(out,2)*100
print("R-Squred value (goodness of model) for training set :",Test_RS,"%")
```

```
R-Squred value (goodness of model) for training set : 94.0 %
```

0.1.14 Underfitting and overfitting observation

[257]:

```
print("Error \t From training phase              From testing phase ")
print("==============================================================")
print("SSE   \t",Train_SSE,"\t\t", Test_SSE)
print("MSE   \t",Train_MSE,"\t\t", Test_MSE)
print("RMSE   \t",Train_RMSE,"\t\t", Test_RMSE)
print("MAE   \t",Train_MAE,"\t\t", Test_MAE)
print("RS   \t",Train_RS,"\t\t", Test_RS)
```

```
Error    From training phase              From testing phase
==============================================================
SSE      0.08            0.06
MSE      0.0             0.01
RMSE     0.0             0.1
MAE      0.02            0.01
RS       97.0            94.0
```