Polynomial Fitting

This lesson discusses root mean square error and how to fit a polynomial with the information it provides.

WE'LL COVER THE FOLLOWING

In various scientific fields, we often have data that we need to model using a mathematical equation. This is also called curve fitting, and when the mathematical equation of the model is a polynomial, we call it polynomial fitting.

Root mean square error

One way to quantify the fit between data and a model is to compute the root mean square error. This error is defined as the difference between the observed value and the modeled value. The term 'error' is also sometimes known as **residual**. If the error of data point i is written as ε_i, and the total number of observations is N, then the sum of squared errors S is:

$$S = \sum \varepsilon_i^2$$

When the total number of observations is N, the root mean square error E is computed as:

$$E = \sqrt{\frac{1}{N} S} = \sqrt{\frac{1}{N} \sum \varepsilon_i^2}$$

The root mean square error is an estimate of how well the curve fits and can be computed for any model and any dataset.

Fitting a polynomial

We have $x - y$ data to which we would like to fit a curve and we also want to minimize the least square deviation from the fit of the data. NumPy provides the `polyfit()` function. It takes an array containing data for x, the corresponding data for y, and the desired order of the polynomial that will be determined to fit the data in the least-square sense as well as possible.

```
polyfit(x, y, n)
```

The `polyfit()` function returns an array containing the coefficients of the polynomial. The coefficients are placed in order of decreasing power. Suppose we have the following array:

```
array([a, b, c, d])
```

This will correspond to the following polynomial:

$$y = ax^3 + bx^2 + cx + d$$

Let's implement the `polyfit()` function:

```
import numpy as np

x = np.arange(-10, 11)
y = np.random.randn(len(x))     # generating random data for y
z = np.polyfit(x, y, 3)         # computing coeffiecients of polynomial of order 3
print(z)
```

We have the coefficients of the polynomial stored in the variable `z`. It is useful to use the function `poly1d(z)` to create polynomials from the coefficients of the polynomial. This function returns an object which acts like a function and will return a value for a particular `x` argument. It can also be used to generate a set of values by giving an array as the input argument. Plotting the data will give a better picture of how this works, so let's do that in the example below:

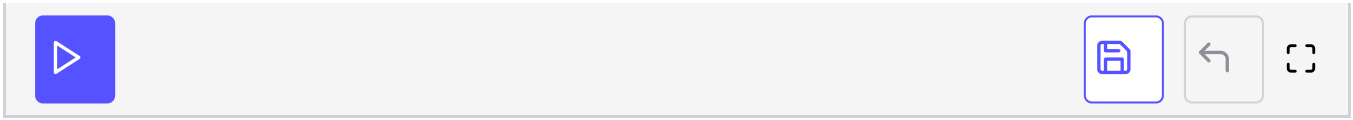
```
import numpy as np
import matplotlib.pyplot as plt
```

```
import matplotlib.pyplot as plt

n = 3           # polynomial order
x = np.linspace(-10, 11, 20)
y = 0.5 * np.random.randn(len(x)) + (0.1 * x**2)   # generating random data for y
z = np.polyfit(x, y, n)         # computing coeffiecients of polynomial of order 3

p = np.poly1d(z)        # generating a polynomial from the coefficients
xp = np.linspace(-10, 11, 100)   # generating xdata for polynomial plotting

fig, ax = plt.subplots()
ax.plot(x, y, 's', marker='.')    # plotting actual data
ax.plot(xp, p(xp))      # plotting polynomial data
```



In line 7, we are computing the coefficients of a third-order polynomial to fit the random data as we create a polynomial from this random data in line 9. This polynomial is plotted alongside the actual data for comparison. Choosing the order of the polynomial is up to the user, as it varies according to data and needs to be fine-tuned to find the best possible fit.

Choosing the right polynomial

The internal algorithm of `polyfit()` computes the polynomial of a specific order that has the least root mean squared error, but what if we want to compute the best possible polynomial that fits the raw data? We need to try different degrees of polynomials to see which one has the least error. We will compute the error by defining a utility function named `RMSE()`. In the example below, we will define `RMSE()` to compute the root mean squared error according to the formula given above.

Do it yourself

The function for creating the random data `create()` has been deliberately hidden from you. Change the order of polynomial `n` and see which order of polynomial fits best. See how the value of `error` varies as you change the value of `n`. The initial value of `n` is set to be 0.

```
import matplotlib.pyplot as plt

# function to compute root mean square error
def RMSE(y1, y2):
    return (np.square(np.subtract(y1, y2))).mean()

n = 0           # polynomial order
```

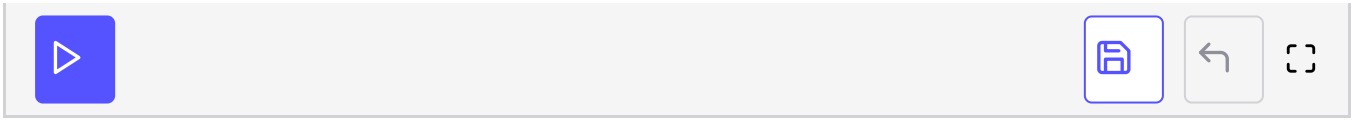
```
n = 0           # polynomial order
x = np.linspace(-5, 5, 40)
y = create(x)       # create random data

z = np.polyfit(x, y, n)         # computing coeffiecients of polynomial of order n

p = np.poly1d(z)            # generating a polynomial from the coefficients
xp = np.linspace(-5, 5, 100)      # generating xdata for polynomial plotting

error = RMSE(y, p(x))       # computing error
print("Root mean square error:", error)

fig, ax = plt.subplots()
ax.plot(x, y, 's', marker='.')      # plotting actual data
ax.plot(xp, p(xp))          # plotting polynomial data
```



As you can see above, as a general trend, RMSE drops when increasing the order of the polynomial, with diminishing returns: for example, increasing the order of the polynomial from `n = 4` to `n = 10` will reduce the RMSE, but will also increase the complexity of the polynomial.

So, the user has to strike a balance between the error and the complexity of the polynomial.

Click below to see the graph for the order of the polynomial against the value of RMSE for the above example:

Show Graph

In the next lesson, we will learn about general curve fitting.