Curve Fitting

In this lesson, we will discuss curve fitting by finding the optimized parameters of an equation.

WE'LL COVER THE FOLLOWING

In some applications, polynomials might not be the best choice to fit a model to data. Instead, functions involving exponentials or sinusoids might be needed. While `polyfit()` fits a polynomial to data, there are many other functions that you may want to use to fit your data. For example, the function `curve_fit()` can be used to fit an arbitrary function that you define; `curve_fit` is part of the `scipy.optimize` package.

Syntax

The `curve_fit` function requires you to write a function that takes the independent variable as its first argument followed by the parameter(s) that you want to fit and returns the value of the function at all values of the independent variable for the supplied parameters. For example, to fit a straight line, you would need to write a function:

```
def func(x, a, b):
    y = a * x + b
    return y
```

where `x` is the independent variable, `y` is the dependent variable and `a` and `b` are the function parameters for fitting the data.

There are three necessary arguments for the `curve_fit` function:

1. The function that you want to fit.

2. The values of the independent variable.

3. The values of the dependent variable.

This is a good place to start, however, the final parameters might, in some cases, include a *fourth* argument.

> 💡 The `curve_fit` function is based on the Levenburg-Marquardt algorithm. For the curve fitting to work, this algorithm needs to start the fitting process with initial guesses for the final parameters. If these are not specified, the value 1.0 is used for the initial guess. If the algorithm fails to fit a function to data, we need to give the algorithm better estimates for the initial parameters.

The `curve_fit` function then returns an array with the optimal parameters (in a least-squares sense) and a second array containing the covariance of the optimal parameters. The diagonals provide the variance of the parameter estimations.

Fitting parameters

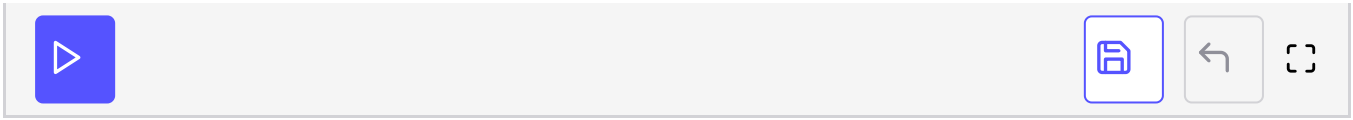
A nuclear physicist is working on finding the half-life of an unknown isotope. The data from the geiger counter has a lot of noise but the data closely resembles this function:

$$f(x) = Ae^{ax} + b$$

Let's help the nuclear physicist find the parameters of this equation using the `curve_fit` function of `scipy.optimize`:

```
from scipy.optimize import curve_fit
import numpy as np
import matplotlib.pyplot as plt

def RMSE(y1, y2):
    return (np.square(np.subtract(y1, y2))).mean()

def func(x, A, a, b):
    y = A * np.exp(a * x) + b
    return y

x = np.linspace(0, 4, 50)
```

```
y = 0.2 * np.random.randn(len(x)) + func(x, 4.5, -1.3, 3) # adding noise to signal

popt, pcov = curve_fit(func, x, y, p0=(3, -1, 2.9)) # p0 are initial estimates

print("A = ", popt[0], "a = ", popt[1], "b = ", popt[2] )
yfit = func(x, *popt) # equivalent to popt[0], popt[1], popt[2]
plt.figure()
plt.plot(x, y, 'bo', label='observed')
plt.plot(x, yfit, 'r', label='fit')
plt.legend(loc='best')

rmse = RMSE(y, yfit)
plt.title('RMSE: '+str(rmse));
```



In lines 8 - 10, we have defined the function of interest with `x` being the independent variable and `A`, `a`, and `b` being the parameters we want to find. In line 13, we have defined the function with some initial arguments and added random noise to the output of the function `func`.

In line 15, we are calling the function `curve_fit` with a tuple `p0` as the fourth input argument. `p0` has initial guesses of the parameters we want to find. The initial guesses are $A = 3$, $a = -1.3$ and $b = 2.9$. Once we provide the algorithm with roughly the right parameters, the fitting works. Try changing these parameters and see what happens.

We have stored the optimal parameters in `popt` and the variance of these parameters in `pcov`. As you can see in the output as well, the optimal parameters are close to the actual parameters.

In line 23, we calculate the root mean square error using the function `RMSE()`.

In the next lesson, we will discuss optimization in SciPy.