Ex4: Curve Fit

Objective

1. Understand the significance of the absolute_sigma parameter in curve_fit.
2. Know how to obtain the variance of each optimal/fitted parameter
3. Compare and explain the differences in variance of each optimal/fitted parameter when absolute_sigma=True versus absolute_sigma=False

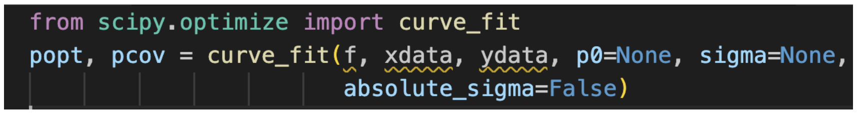
Introduction to Curve Fitting

Curve fitting is finding a mathematical function (a curve) that best matches a series of observed data points. In other words, we choose a function form and adjust its parameters so that it "fits" the data as closely as possible. This best-fit curve captures the underlying trend in the data and can be used to **make predictions or interpolate** values where data might not be available. Unlike exact interpolation, curve fitting does not require the curve to pass through every point; instead, it seeks an approximate function that minimizes the overall error (often using a least-squares criterion).

This is extremely useful in data analysis because it allows us to **summarize relationships** in data with a compact model, smooth out noise, and infer underlying parameters or future behavior of the system being studied. In Python's SciPy library, the **curve_fit** function (in scipy.optimize) is a convenient tool for curve fitting. It uses **non-linear least squares** to estimate the optimal parameters of a given model function so that the function best fits the provided data. Essentially, curve_fit finds the parameter values that minimize the sum of squared differences between the model function and the data (the residuals). Under the hood, curve_fit by default applies the Levenberg-Marquardt algorithm for unconstrained problems (a robust iterative method for least-squares minimization). If you provide parameter bounds (constraints), it will automatically switch to a different algorithm (Trust Region Reflective) to handle those bounds. The function returns the **optimal parameters** of the fit and an estimate of the **covariance matrix**, which can be used to assess the uncertainty of the parameter estimates. This makes curve_fit a powerful one-stop function that fits a wide variety of models to data.

Function Syntax and Parameters

The syntax for curve_fit is:



```
from scipy.optimize import curve_fit
popt, pcov = curve_fit(f, xdata, ydata, p0=None, sigma=None,
                       absolute_sigma=False)
```

Where:

- **f (callable)**: The model function that defines the curve to fit. It must be of the form f(x, a, b, c, ...) where x is the independent variable and a, b, c, ... are the parameters to optimize. In other words, the function should accept x as its first argument, and the subsequent arguments are the parameters that curve_fit will adjust.

- **xdata (array_like)**: The independent variable values where the data is measured. This is typically an array of x-values (of length M) for your data points. For multi-dimensional fitting, xdata can be a tuple or multi-dimensional array representing multiple independent variables, but in most simple cases it's a 1D array of x-values.
- **ydata (array_like)**: The dependent data – an array of y-values (also length M) corresponding to each x in xdata. These are the observed data points we want our function to fit. .
- **p0 (array_like, optional)**: Initial guess for the parameters. This should be an array or list of initial values for each parameter in the model. If not provided, SciPy will try to automatically choose initial values (often all ones). Providing a reasonable guess can help the fitting algorithm converge faster or avoid local minima.
- **sigma (array_like, optional)**: The uncertainties (standard deviations) of ydata. If provided, curve_fit will weight the fit accordingly; points with smaller sigma are treated as more reliable. This is useful if some data points have higher measurement error than others. By default, sigma=None, which is effectively equivalent to all points having equal weight.
- **absolute_sigma (bool, optional)**: If False (default), sigma is used only to weigh relative errors; the overall scaling of pcov is adjusted so that the reduced chi-square is ~1 at the solution. If True, sigma is taken as absolute uncertainties, and thus pcov reflects the true covariance based on those errors. In simpler terms, if you have trustworthy error estimates and want the parameter covariance to directly correspond to those, set absolute_sigma=True. Otherwise, with the default False, the magnitude of pcov may be scaled by a factor (the fit will still be correct, but the reported uncertainties are adjusted)

Return values: The function returns a tuple (popt, pcov). Here, **popt** is an array of the optimized parameters [a, b, c, ...] that best fit the data (in the same order as the parameter arguments in f). **pcov** is the covariance matrix of the parameter estimates – a 2D array where the diagonal entries represent the variance of each optimal/fitted parameter, and off-diagonals represent the covariance between parameters. We'll discuss how to interpret these in detail later. For now, note that a more minor variance (diagonal value) means the fit determines a parameter more precisely.

Interpreting Off-Diagonal Elements: Covariance

A covariance matrix pcov typically looks like this for two parameters (for example, parameters a and b):

$$\mathrm{pcov} = \begin{pmatrix} \sigma_a^2 & \mathrm{cov}(a,b) \\ \mathrm{cov}(b,a) & \sigma_b^2 \end{pmatrix}$$

- **Diagonal elements**:
  σ_a^2, σ_b^2 represent the variances of parameters a, b.

- **Off-diagonal elements** (covariances) measure how two parameters vary together:

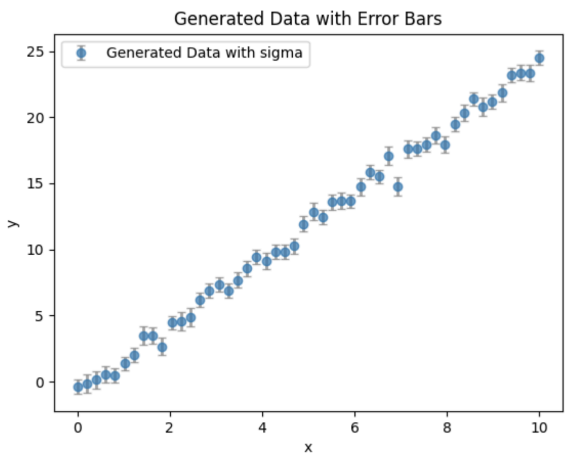
  a. Positive covariance means if one parameter increases, the other tends to increase.

b. Negative covariance means if one parameter increases, the other tends to decrease.
c. Zero covariance indicates that parameters vary independently of each other.

Example

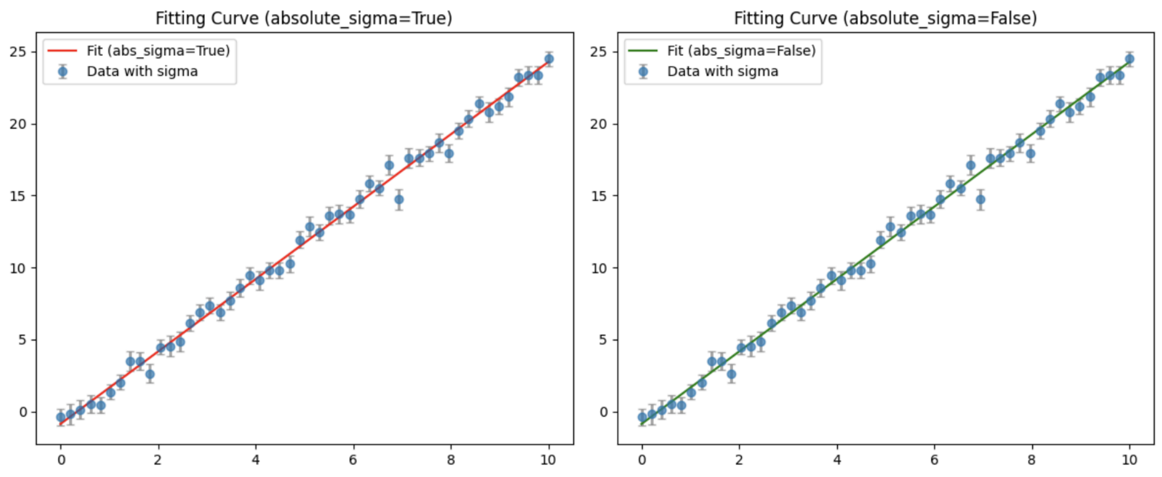
First, we generate data with varying sigma using a first-order linear function.

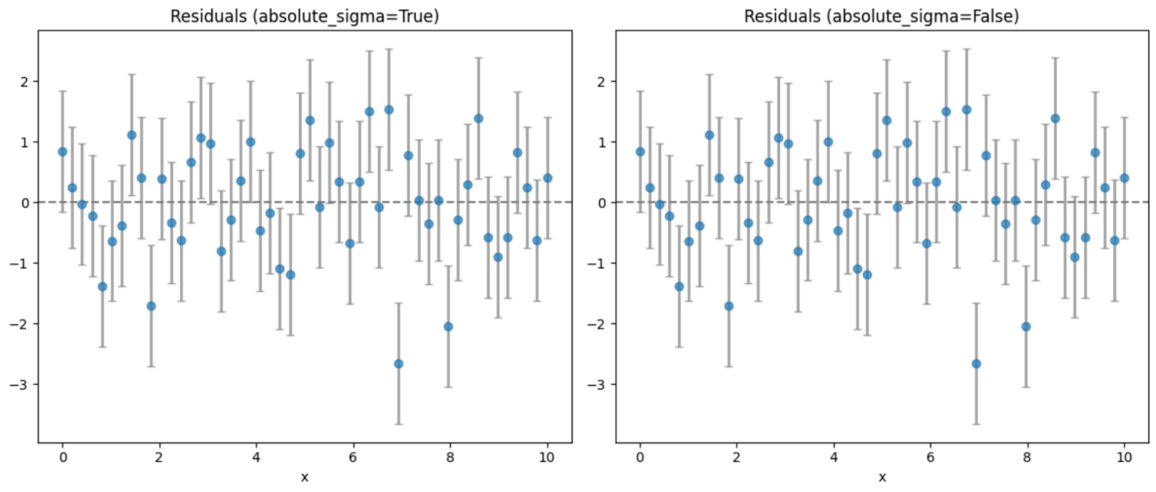
```python
np.random.seed(42)
x = np.linspace(0, 10, 50)
true_a, true_b = 2.5, -0.8

def linear(x, a, b):
    return a * x + b

# define sigma for clearer error bars
sigma = 0.5 + 0.2 * np.random.rand(x.size)
y = linear(x, true_a, true_b) + np.random.normal(0, sigma)
```



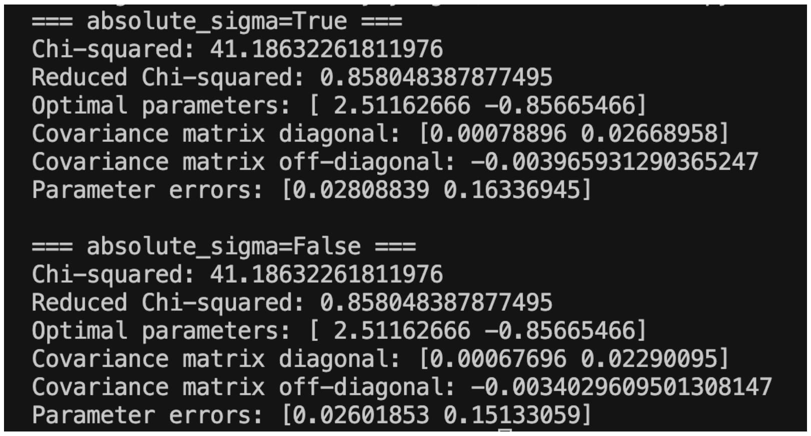
Perform curve fitting separately for absolute_sigma=True and absolute_sigma=Fals

```python
popt_true, pcov_true = curve_fit(linear, x, y, sigma=sigma, absolute_sigma=True)
popt_false, pcov_false = curve_fit(linear, x, y, sigma=sigma, absolute_sigma=False)
```



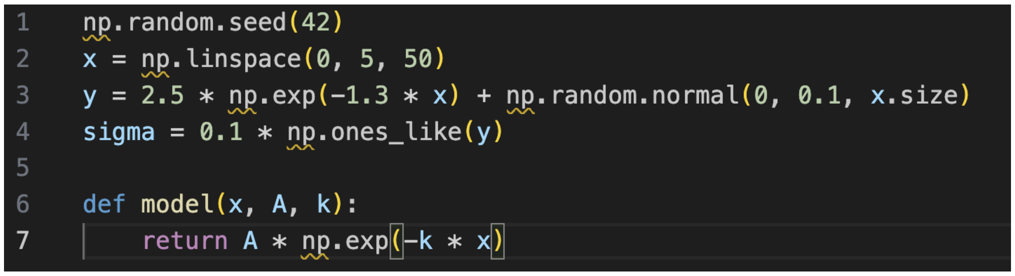


Why are the fit curves and residuals identical in both cases? The reason is that, regardless of whether `absolute_sigma` is set to `True` or `False`, the sigma values used in the formula above (our input values) remain unchanged. Therefore, the numerical results of the calculation must be identical. So, what exactly changes when `absolute_sigma=False`? When `absolute_sigma=False`, the internal mechanism of `curve_fit` adjusts the scale by multiplying the covariance matrix by the computed reduced chi-square value. This effectively forces the reduced chi-square to have a scale of 1. You can check the results listed below.


```
=== absolute_sigma=True ===
Chi-squared: 41.18632261811976
Reduced Chi-squared: 0.858048387877495
Optimal parameters: [ 2.51162666 -0.85665466]
Covariance matrix diagonal: [0.00078896 0.02668958]
Covariance matrix off-diagonal: -0.003965931290365247
Parameter errors: [0.02808839 0.16336945]

=== absolute_sigma=False ===
Chi-squared: 41.18632261811976
Reduced Chi-squared: 0.858048387877495
Optimal parameters: [ 2.51162666 -0.85665466]
Covariance matrix diagonal: [0.00067696 0.02290095]
Covariance matrix off-diagonal: -0.0034029609501308147
Parameter errors: [0.02601853 0.15133059]
```

Excise

Use the data below to perform a nonlinear curve fitting, and compare the residuals and parameter uncertainties between the two cases (absolute sigma = true and absolute sigma = false).


```
np.random.seed(42)
x = np.linspace(0, 5, 50)
y = 2.5 * np.exp(-1.3 * x) + np.random.normal(0, 0.1, x.size)
sigma = 0.1 * np.ones_like(y)

def model(x, A, k):
    return A * np.exp(-k * x)
```

1. Plot the original data with error bars
2. Plot the residuals for each of the two cases separately.
3. List the fitted parameters and their uncertainties for both cases (`absolute_sigma=True` and `absolute_sigma=False`).

Questions

In the previous experimental data (pendulum experiment), there are five different pendulum lengths, and for each length, the period was measured five times (assuming that the only source of error in the period measurement is random error following a Gaussian distribution). Please perform chi-squared fitting in the following two cases:

1. First, average the five period measurements for each pendulum length and obtain the error bar (standard deviation), then perform the fitting.
2. Fit all 25 data points using the errors obtained by the method mentioned above for chi-squared fitting.

Discuss the statistical similarities and differences between the fitting results obtained from these two processing methods.

Reference

1. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.htm