

1. Importer les modules

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

2. Préparer vos données

Imaginons que vous avez 10 points (x_i, y_i) et une incertitude de 0,1 sur x et 0,5 sur y :

```
# Exemple de données (vous pouvez remplacer par vos propres valeurs)
x_data = np.array([0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0])
y_data = np.array([1.2, 2.3, 3.1, 4.8, 6.5, 8.0, 9.9, 12.1, 14.2, 16.5])
x_err=0.1
y_err=0.5
```

3. Définir le modèle que vous voulez ajuster

`curve_fit` attend une fonction dont le premier argument est la variable indépendante (x) et les arguments suivants sont les paramètres à estimer.

Exemple : ajuster une fonction quadratique $f(x) = ax^2 + bx + c$

```
def modele_quadratique(x, a, b, c):
    return a * x**2 + b * x + c
```

Vous pouvez bien sûr choisir n'importe quel autre modèle (exponentiel, gaussien, sigmoïde, ...).

4. Appeler `curve_fit`

```
# Estimation des paramètres
params_opt, params_cov = curve_fit(modele_quadratique, x_data, y_data)
# Extraction des paramètres estimés
a_estime, b_estime, c_estime = params_opt
print(f"a = {a_estime:.4f}, b = {b_estime:.4f}, c = {c_estime:.4f}")
```

- `params_opt`: tableau contenant les valeurs optimales des paramètres.
- `params_cov`: matrice de covariance qui permet d'estimer les incertitudes (écarts-types):

```
perr = np.sqrt(np.diag(params_cov))
print(f"Incertitudes : Δa={perr[0]:.4f}, Δb={perr[1]:.4f}, Δc={perr[2]:.4f}")
```

Astuce: si votre fonction possède plusieurs minima locaux ou si les paramètres sont très différents en amplitude, il peut être utile de fournir un guess initial avec l'argument `p0`. Exemple: `curve_fit(..., p0=[1.0, 1.0, 0.0])`.

5. Visualiser le résultat

Points durs pour tracer la courbe ajustée

```
x_fine = np.linspace(min(x_data), max(x_data), 200)
```

```
y_fitted = modele_quadratique(x_fine, *params_opt)
```

```
plt.figure(figsize=(8,5))
```

```
plt.errorbar(x_data, y_data, yerr=y_err, xerr=x_err, fmt='b+', capsize = 5, color='red', label='Points mesurés') #points avec incertitudes
```

```
plt.plot(x_fine, y_fitted, label='Ajustement (quadratique)', linewidth=2)
```

```
plt.title('Ajustement avec curve_fit')
```

```
plt.xlabel('x')
```

```
plt.ylabel('y')
```

```
plt.legend()
```

```
plt.grid(True)
```

```
plt.show()
```

Vous devriez voir vos points originaux et la courbe lisse correspondant au modèle ajusté.

6. Vérifier la qualité de l'ajustement

Quelques indicateurs simples :

Indicateur	Description
R² (coefficient de détermination)	Mesure la proportion de variance expliquée. Vous pouvez le calculer manuellement : $ss_res = np.sum((y_data - modele_quadratique(x_data, *params_opt))^2)$ $ss_tot = np.sum((y_data - np.mean(y_data))^2)$ $r2 = 1 - ss_res/ss_tot$
Résidus	Affichez les différences entre les valeurs observées et prédites : $residus = y_data - modele_quadratique(x_data, *params_opt)$ et tracez-les pour détecter d'éventuels biais.
Inc. des paramètres	Les écarts-types obtenus via $np.sqrt(np.diag(params_cov))$ donnent une idée de la précision de chaque paramètre.

7. Exemple complet (tout-en-un) → disponible sur Pronote

8. Points clés à retenir

1. **Définissez clairement votre fonction modèle** – la première entrée doit toujours être la variable indépendante.
2. **Fournissez un bon point de départ (p0)** si le problème est non linéaire ou si les paramètres sont très différents en ordre de grandeur.
3. **Examinez la matrice de covariance** pour connaître les incertitudes sur chaque paramètre.
4. **Vérifiez les résidus et le R²** afin de vous assurer que le modèle décrit réellement vos données.
5. **Visualisez** – rien ne vaut un bon graphique pour détecter rapidement un mauvais ajustement.