Project 2: Fit a Curve to Noisy Data

This project generates noisy quadratic data and fits a polynomial to it, introducing you to curve fitting in Python with scipy. It’s a step up in complexity but still approachable.

Code for PyCharm

Create another file in PyCharm (e.g., curve\_fit.py) and paste this code:

python

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import curve\_fit

# Step 1: Define a quadratic function

def quadratic(x, a, b, c):

return a \* x\*\*2 + b \* x + c

# Step 2: Generate noisy data

x = np.linspace(-5, 5, 100) # Like MATLAB's linspace(-5, 5, 100)

true\_a, true\_b, true\_c = 1, -2, 3 # True parameters

y = quadratic(x, true\_a, true\_b, true\_c) # True quadratic

noise = np.random.randn(len(x)) \* 2 # Noise

y\_noisy = y + noise # Noisy data

# Step 3: Fit a curve

popt, pcov = curve\_fit(quadratic, x, y\_noisy) # Fit the quadratic

a\_fit, b\_fit, c\_fit = popt # Extract fitted parameters

y\_fit = quadratic(x, a\_fit, b\_fit, c\_fit) # Fitted curve

# Step 4: Plot

plt.scatter(x, y\_noisy, label='Noisy Data', color='blue', alpha=0.5)

plt.plot(x, y, label='True Quadratic', color='green', linestyle='--')

plt.plot(x, y\_fit, label='Fitted Quadratic', color='red')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Quadratic Fit to Noisy Data')

plt.legend()

plt.grid(True)

plt.show()

# Step 5: Print results

print(f"True parameters: a={true\_a}, b={true\_b}, c={true\_c}")

print(f"Fitted parameters: a={a\_fit:.2f}, b={b\_fit:.2f}, c={c\_fit:.2f}")

Running in PyCharm

1. Save and Run: Save the file (Ctrl+S) and hit the “Run” button (Shift+F10).
2. View Output: The console will print the true and fitted parameters. A plot window will pop up showing the noisy data, true curve, and fitted curve.
3. Interact with Plots: You can zoom, pan, and save the plot from the pop-up window’s toolbar, just like in MATLAB.

PyCharm Tip: Use the “Run” button’s dropdown to create a “Run Configuration” if you plan to tweak scripts often. This lets you set specific arguments or environments for running your script.

What’s Happening Here?

* curve\_fit: From scipy.optimize, it’s like MATLAB’s polyfit but more flexible. It minimizes the error between your data and model.
* np.linspace: Like MATLAB’s linspace, it creates evenly spaced points.
* Plotting: Combines scatter plots (plt.scatter) and line plots (plt.plot) for data and fits.

Your Turn

Play around with the code in PyCharm:

1. Change the true parameters (true\_a, true\_b, true\_c) and rerun.
2. Increase the noise level (e.g., multiply noise by 5).
3. Try fitting a different function, like a linear one:

python

def linear(x, m, c):

return m \* x + c

popt, pcov = curve\_fit(linear, x, y\_noisy)

PyCharm Refactoring Tip: If you want to rename a variable (e.g., y\_noisy to data), right-click it, select Refactor > Rename, and PyCharm will update all references automatically. Super handy for keeping code clean!

PyCharm Features to Leverage While Coding

1. Autocomplete: Start typing (e.g., np.), and PyCharm will suggest functions like arange or mean. Press Tab or Enter to accept.
2. Documentation: Hover over a function and press Ctrl+Q (or Cmd+J on Mac) to see its docs—great for learning what curve\_fit expects.
3. Debugging: If your fit doesn’t work as expected, set a breakpoint (click the gutter next to a line number), then run in Debug mode (Shift+F9). Step through your code to see variable values.
4. Version Control: If you mess up and want to undo, PyCharm’s Git integration (or Local History) can save you. Go to VCS > Enable Version Control Integration to set up Git.

Next Steps with These Projects

Both projects give you a solid foundation for scientific coding in Python with PyCharm. Here’s how to build on them:

* Project 1: Add error bars to the plot using plt.errorbar (pass yerr=std\_signal). Or filter the signal (e.g., with a moving average using np.convolve).
* Project 2: Load real data from a file (e.g., a CSV with np.loadtxt or pandas.read\_csv) and fit a curve to it. Or try fitting more complex models like exponentials or Gaussians.

Since you mentioned learning by coding, I’m happy to keep throwing you projects or help tweak these further. Got any specific MATLAB tasks from your science days you want to recreate in Python? Or want to dive into something like solving ODEs or signal processing next? Let me know where you want to go!