PyCharm Setup Recap

Before we dive into the projects, let’s make sure your PyCharm environment is ready:

1. Python Interpreter: Ensure PyCharm is pointing to a Python installation (e.g., 3.11 or 3.12). Go to File > Settings > Project > Python Interpreter. If it’s not set, click the gear icon, select “Add Interpreter,” and point to your Python install.
2. Install Libraries: We’ll need numpy, matplotlib, and scipy. In the same Python Interpreter settings, click the + button, search for each library, and install them. Or use PyCharm’s terminal (bottom of the window) to run:

bash

pip install numpy matplotlib scipy

1. Create a File: In PyCharm’s Project pane (left side), right-click your project folder, select New > Python File, name it (e.g., project1.py), and you’re ready to code.
2. Running Code: PyCharm has a green “Run” button (or Shift+F10) to execute your script. Plots will pop up in a separate window thanks to matplotlib.

PyCharm Tip: If plots don’t appear, go to File > Settings > Tools > Python Scientific and uncheck “Show plots in tool window” to have them pop up like MATLAB’s plots. Alternatively, keep it enabled to view plots in PyCharm’s SciView pane.

Grok: https://x.com/i/grok/share/JJG8eycupCq0bdkvtonwz4dUT

Project 1: Analyze and Plot a Noisy Signal

This project generates a noisy sine wave, computes basic stats (mean, standard deviation), and plots it with a mean line—perfect for practicing array operations, stats, and plotting in Python.

Code for PyCharm

Create a new file in PyCharm (e.g., noisy\_signal.py) and paste this code:

python

import numpy as np

import matplotlib.pyplot as plt

# Step 1: Generate time array (like MATLAB's 0:0.01:10)

t = np.arange(0, 10, 0.01) # 0 to 10 seconds, step of 0.01

# Step 2: Generate a noisy sine wave

frequency = 0.5 # Hz

amplitude = 3 # Arbitrary units

signal = amplitude \* np.sin(2 \* np.pi \* frequency \* t) # Sine wave

noise = np.random.randn(len(t)) # Random noise (like MATLAB's randn)

noisy\_signal = signal + noise # Combine

# Step 3: Compute basic stats

mean\_signal = np.mean(noisy\_signal)

std\_signal = np.std(noisy\_signal)

print(f"Mean of the noisy signal: {mean\_signal:.2f}")

print(f"Standard deviation: {std\_signal:.2f}")

# Step 4: Plot the signal and the mean

plt.plot(t, noisy\_signal, label='Noisy Signal', color='blue', alpha=0.5) # Signal

plt.axhline(y=mean\_signal, color='red', linestyle='--', label=f'Mean = {mean\_signal:.2f}') # Mean line

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.title('Noisy Sine Wave with Mean')

plt.legend()

plt.grid(True)

plt.show()

Running in PyCharm

1. Save the File: Save your file (Ctrl+S or Cmd+S on Mac).
2. Run the Code: Click the green “Run” button in the top-right corner (or press Shift+F10). Alternatively, right-click the file in the Project pane and select “Run ‘noisy\_signal’.”
3. View Output: The console at the bottom will show the printed stats (mean and standard deviation). A plot window will pop up showing the noisy signal and the mean line.

PyCharm Tip: If the plot doesn’t appear, ensure plt.show() is at the end of your script. You can also enable interactive plotting by adding plt.ion() after the imports, though plt.show() works fine for most cases.

What’s Happening Here?

* np.arange: Mimics MATLAB’s 0:0.01:10 for time steps.
* Noise and Signal: np.random.randn and np.sin are like MATLAB’s randn and sin.
* Stats: np.mean and np.std replace MATLAB’s equivalents.
* Plotting: matplotlib gives you control similar to MATLAB, but you need plt.show() to display the plot.

Your Turn

Experiment with the code in PyCharm:

1. Change the frequency or amplitude and rerun (Shift+F10).
2. Add more noise by scaling it (e.g., noise = np.random.randn(len(t)) \* 2).
3. Plot the original signal (signal) on the same plot in a different color (e.g., plt.plot(t, signal, label='Original Signal', color='green')).

PyCharm Debugging Tip: If something goes wrong, PyCharm will highlight errors in red. Hover over them for suggestions, or click the red lightbulb for quick fixes (like adding missing imports). You can also set breakpoints (click the left gutter next to a line number) and use the debugger (Shift+F9) to step through your code.