Analyze a Real Dataset with Filtering and FFT

Let’s analyze a real dataset by loading it, applying a low-pass filter, and computing its Fast Fourier Transform (FFT) to identify dominant frequencies—common tasks in scientific signal processing. Since real data varies, I’ll simulate a dataset you can use (or you can grab one online), but I’ll also show how to load real data from a CSV file.

Step 1: Create or Load Data

For this example, I’ll generate a synthetic dataset mimicking experimental measurements (e.g., temperature or voltage over time). If you have a real dataset (CSV or text file), you can load it instead with pandas.read\_csv or numpy.loadtxt.

Code: Signal Analysis in PyCharm

Create a new file (e.g., signal\_analysis.py):

python

import numpy as np

import matplotlib.pyplot as plt

from scipy import fft, signal

# Step 1: Generate synthetic data (or load your own)

t = np.linspace(0, 10, 1000) # Time from 0 to 10 seconds

data = 2 \* np.sin(2 \* np.pi \* 1 \* t) + 0.5 \* np.sin(2 \* np.pi \* 5 \* t) + np.random.randn(len(t)) \* 0.5 # Two frequencies + noise

# If you have a CSV file, uncomment and adjust this instead:

# import pandas as pd

# df = pd.read\_csv('your\_data.csv')

# t = df['time\_column'].values

# data = df['data\_column'].values

# Step 2: Apply a low-pass filter

sos = signal.butter(5, 2, 'low', fs=100, output='sos') # 5th order Butterworth, 2 Hz cutoff

filtered\_data = signal.sosfilt(sos, data)

# Step 3: Compute FFT

sampling\_rate = len(t) / (t[-1] - t[0]) # Hz

frequencies = fft.fftfreq(len(t), 1/sampling\_rate)

fft\_values = fft.fft(data)

positive\_freqs = frequencies[frequencies >= 0]

positive\_magnitudes = np.abs(fft\_values)[frequencies >= 0]

# Step 4: Plot original and filtered data

plt.figure(figsize=(10, 8))

plt.subplot(3, 1, 1)

plt.plot(t, data, label='Original Data', color='blue', alpha=0.5)

plt.plot(t, filtered\_data, label='Filtered Data', color='red')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.title('Original and Filtered Signal')

plt.legend()

plt.grid(True)

# Step 5: Plot FFT

plt.subplot(3, 1, 2)

plt.plot(positive\_freqs, positive\_magnitudes, color='purple')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Magnitude')

plt.title('Frequency Spectrum (FFT)')

plt.grid(True)

plt.xlim(0, 10) # Limit frequency range for clarity

# Step 6: Plot zoomed-in filtered data

plt.subplot(3, 1, 3)

plt.plot(t, filtered\_data, label='Filtered Data', color='red')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.title('Zoomed Filtered Signal')

plt.grid(True)

plt.xlim(0, 2) # Zoom into first 2 seconds

plt.tight\_layout()

plt.show()

What’s Happening Here?

* Data Generation: I created a signal with two frequencies (1 Hz and 5 Hz) plus noise, mimicking a real measurement.
* Filtering: Used scipy.signal.butter (like MATLAB’s butter) to design a low-pass filter, cutting off frequencies above 2 Hz.
* FFT: Computed the frequency spectrum with scipy.fft.fft (like MATLAB’s fft), showing peaks at the signal frequencies.
* Plotting: Three subplots to visualize the original signal, filtered signal, and frequency spectrum.

Your Turn

1. If you have a real dataset (CSV or text), replace the synthetic data with it. Adjust column names in pd.read\_csv as needed.
2. Change the filter cutoff frequency (e.g., 1 Hz or 3 Hz) and see how it affects the filtered signal.
3. Identify dominant frequencies in the FFT plot and compare them to what you expect.

Machine Learning Projects for Beginners

Since you’re new to machine learning (ML) but have a scientific background, I’ll start with simple projects using scikit-learn, a beginner-friendly library for ML in Python. These projects will build on your data handling and plotting skills and introduce ML concepts like regression, classification, and evaluation. We’ll assume no prior ML knowledge but leverage your Python progress.

Step 1: Install scikit-learn

In PyCharm, go to File > Settings > Project > Python Interpreter, click the + button, search for scikit-learn, and install it. Or use the terminal:

bash

pip install scikit-learn

ML Project 1: Linear Regression on a Synthetic Dataset

Let’s start with a regression task: predict a target variable from a feature using linear regression. We’ll generate a simple dataset, fit a model, and visualize the results.

Code: Linear Regression in PyCharm

Create a new file (e.g., ml\_regression.py):

python

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 1: Generate synthetic data

np.random.seed(0)

X = np.linspace(0, 10, 100).reshape(-1, 1) # Feature (e.g., time)

y = 3 \* X.flatten() + 2 + np.random.randn(100) \* 2 # Target (linear with noise)

# Step 2: Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Step 3: Fit linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Step 4: Make predictions

y\_pred = model.predict(X\_test)

# Step 5: Evaluate model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse:.2f}")

print(f"R^2 Score: {r2:.2f}")

# Step 6: Plot results

plt.scatter(X\_train, y\_train, color='blue', label='Training Data', alpha=0.5)

plt.scatter(X\_test, y\_test, color='red', label='Test Data', alpha=0.5)

plt.plot(X\_test, y\_pred, color='green', label='Prediction')

plt.xlabel('Feature (X)')

plt.ylabel('Target (y)')

plt.title('Linear Regression Fit')

plt.legend()

plt.grid(True)

plt.show()

What’s Happening?

* Data: Generated a linear relationship with noise.
* Splitting: Used train\_test\_split to create training and test sets (80% train, 20% test).
* Model: LinearRegression fits a line to the data.
* Evaluation: Mean Squared Error (MSE) and

R2R^2R^2

score measure how well the model fits.

* Plotting: Visualize training data, test data, and the fitted line.

Your Turn

1. Change the noise level in the data generation.
2. Try predicting on new data (e.g., X\_new = np.array([[11], [12]])).
3. Add more features (e.g.,

X2X^2X^2

) to make it a polynomial regression.