Option 3: Machine Learning Project - Classifying Oscillator Behavior

Since you’ve expressed interest in machine learning and completed the driven oscillator (Option 1), let’s tie ML to your physics work. We’ll generate damped oscillator trajectories with varying damping coefficients (( b )), label them as “underdamped” (oscillatory) or “overdamped” (non-oscillatory), and train a classifier to predict the behavior. This introduces binary classification and builds on your Python skills.

Step 1: Setup in PyCharm

You’ll need scikit-learn in addition to numpy, matplotlib, and scipy. Install it if you haven’t:

bash

pip install scikit-learn

Create a new file called ml\_oscillator\_classification.py.

Code: Classifying Oscillator Behavior in PyCharm

Here’s the code:

python

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

# Step 1: Define oscillator function

def oscillator(state, t, m, b, k):

x, v = state

dx\_dt = v

dv\_dt = -(b/m) \* v - (k/m) \* x

return [dx\_dt, dv\_dt]

# Step 2: Generate synthetic dataset with varying b

np.random.seed(0)

t = np.linspace(0, 10, 100) # Time array

m = 1.0 # Fixed mass

k = 2.0 # Fixed spring constant

b\_values = np.concatenate([np.linspace(0.1, 1.9, 50), np.linspace(2.1, 4.0, 50)]) # Underdamped and overdamped

X = [] # Features (trajectories)

y = [] # Labels (0 = underdamped, 1 = overdamped)

critical\_damping = 2 \* np.sqrt(m \* k) # b\_crit = 2\*sqrt(m\*k) = 2.0

for b in b\_values:

state0 = [1.0, 0.0] # Initial position=1, velocity=0

solution = odeint(oscillator, state0, t, args=(m, b, k))

X.append(solution[:, 0]) # Store position trajectory

# Label based on damping: underdamped (b < b\_crit), overdamped (b > b\_crit)

y.append(0 if b < critical\_damping else 1)

X = np.array(X) # Shape: (100 samples, 100 time points)

y = np.array(y)

# Step 3: Feature engineering

X\_features = np.column\_stack([

np.max(np.abs(X), axis=1), # Max amplitude

np.sum(np.diff(X, axis=1) > 0, axis=1), # Number of direction changes (oscillations)

-np.log(np.max(np.abs(X[:, 50:]), axis=1) / np.max(np.abs(X), axis=1)) / t[50] # Decay rate

])

# Step 4: Split data

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

# Step 5: Train logistic regression model

model = LogisticRegression()

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

# Step 6: Predict and evaluate

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

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

print(f"Accuracy: {accuracy:.2f}")

# Step 7: Plot confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Underdamped', 'Overdamped'], yticklabels=['Underdamped', 'Overdamped'])

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

# Step 8: Plot example trajectories

plt.figure(figsize=(10, 4))

for i in [0, 49, 50, 99]: # Plot boundary cases

label = f'b={b\_values[i]:.1f} ({"Under" if y[i] == 0 else "Over"}damped)'

plt.plot(t, X[i], label=label, alpha=0.7)

plt.xlabel('Time (s)')

plt.ylabel('Position (x)')

plt.title('Example Oscillator Trajectories')

plt.legend()

plt.grid(True)

plt.show()

What’s Happening Here?

* Data Generation: Simulated 100 trajectories with ( b ) values below and above the critical damping threshold (

bcrit=2mk=2.0b\_{crit} = 2\sqrt{mk} = 2.0b\_{crit} = 2\sqrt{mk} = 2.0

). Underdamped (

b<2b < 2b < 2

) oscillates; overdamped (

b>2b > 2b > 2

) decays smoothly.

* Feature Engineering: Extracted three features:
  1. Max amplitude.
  2. Number of direction changes (higher for oscillatory underdamped cases).
  3. Decay rate (faster for overdamped).
* Model: LogisticRegression classifies trajectories as underdamped (0) or overdamped (1).
* Evaluation: Accuracy and a confusion matrix assess performance.
* Plots: Confusion matrix and example trajectories visualize the results.

Running in PyCharm

1. Save and Run: Save (Ctrl+S) and hit “Run” (Shift+F10).
2. Output:
   * Console: Accuracy score (likely ~0.9 or higher).
   * Plot 1: Confusion matrix showing true vs. predicted labels.
   * Plot 2: Four example trajectories near the damping boundary.

Your Turn

1. Add noise to the trajectories (X += np.random.randn(\*X.shape) \* 0.05) and check if the classifier still works.
2. Try a different classifier (e.g., from sklearn.svm import SVC; model = SVC()).
3. Add more features (e.g., max velocity: np.max(np.abs(solution[:, 1]))).

Recap and Next Steps

You’ve now completed:

* Option 1: Driven damped oscillator (physics simulation).
* Option 2: Signal analysis with filtering and FFT (signal processing).
* Option 3: Classifying oscillator behavior (ML classification).