Project 3: Solve and Visualize a Damped Harmonic Oscillator

A damped harmonic oscillator models systems like a mass on a spring with friction. The governing equation is:

md2xdt2+bdxdt+kx=0m \frac{d^2x}{dt^2} + b \frac{dx}{dt} + kx = 0m \frac{d^2x}{dt^2} + b \frac{dx}{dt} + kx = 0

Where:

* x(t)x(t)x(t)

: Displacement

* mmm

: Mass

* bbb

: Damping coefficient

* kkk

: Spring constant

We’ll rewrite this second-order ODE as a system of first-order ODEs (like you’d do in MATLAB), solve it using Python’s scipy.integrate.odeint, and plot the results in PyCharm. We’ll also compute and plot the energy over time to add a bit of analysis.

Step 1: Setup in PyCharm

You’ve already got numpy, matplotlib, and scipy installed (from the previous projects). If not, install them via PyCharm’s Python Interpreter settings or terminal:

bash

pip install numpy matplotlib scipy

Create a new Python file in PyCharm (e.g., damped\_oscillator.py).

Step 2: Code the Solution

Here’s the complete code to solve the ODE, plot the position and velocity, and analyze the energy:

python

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint

# Step 1: Define the system of ODEs

def damped\_oscillator(state, t, m, b, k):

x, v = state # state = [position, velocity]

dx\_dt = v

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

return [dx\_dt, dv\_dt]

# Step 2: Set parameters and initial conditions

m = 1.0 # Mass (kg)

b = 0.5 # Damping coefficient (kg/s)

k = 2.0 # Spring constant (N/m)

x0 = 1.0 # Initial position (m)

v0 = 0.0 # Initial velocity (m/s)

state0 = [x0, v0] # Initial state [x, v]

# Step 3: Define time array for solution

t = np.linspace(0, 20, 200) # Time from 0 to 20 seconds, 200 points

# Step 4: Solve the ODE

solution = odeint(damped\_oscillator, state0, t, args=(m, b, k))

x = solution[:, 0] # Extract position

v = solution[:, 1] # Extract velocity

# Step 5: Compute energy (kinetic + potential)

kinetic\_energy = 0.5 \* m \* v\*\*2

potential\_energy = 0.5 \* k \* x\*\*2

total\_energy = kinetic\_energy + potential\_energy

# Step 6: Plot position and velocity

plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)

plt.plot(t, x, label='Position (x)', color='blue')

plt.plot(t, v, label='Velocity (v)', color='red')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.title('Damped Harmonic Oscillator: Position and Velocity')

plt.legend()

plt.grid(True)

# Step 7: Plot energy

plt.subplot(2, 1, 2)

plt.plot(t, kinetic\_energy, label='Kinetic Energy', color='green')

plt.plot(t, potential\_energy, label='Potential Energy', color='orange')

plt.plot(t, total\_energy, label='Total Energy', color='purple')

plt.xlabel('Time (s)')

plt.ylabel('Energy (J)')

plt.title('Energy Over Time')

plt.legend()

plt.grid(True)

plt.tight\_layout() # Adjust spacing between subplots

plt.show()

Running in PyCharm

1. Save and Run: Save the file (Ctrl+S) and click the “Run” button (Shift+F10).
2. View Output: You’ll get a plot with two subplots:
   * Top: Position and velocity over time.
   * Bottom: Kinetic, potential, and total energy over time.
3. Interact: Use the plot window’s toolbar to zoom, pan, or save the figure as a PNG.

What’s Happening Here?

* System of ODEs: We convert the second-order ODE into two first-order ODEs:

dxdt=v\frac{dx}{dt} = v\frac{dx}{dt} = v

and

dvdt=−bmv−kmx\frac{dv}{dt} = -\frac{b}{m}v - \frac{k}{m}x\frac{dv}{dt} = -\frac{b}{m}v - \frac{k}{m}x

. This is exactly how you’d prep for MATLAB’s ode45.

* odeint: From scipy.integrate, it’s Python’s equivalent to ode45. It takes the function (damped\_oscillator), initial state, time array, and extra parameters (args).
* Energy Analysis: We compute kinetic (

12mv2\frac{1}{2}mv^2\frac{1}{2}mv^2

) and potential (

12kx2\frac{1}{2}kx^2\frac{1}{2}kx^2

) energy to see how damping dissipates energy over time.

* Plotting: Using subplot to stack two plots vertically, like you might’ve done in MATLAB with subplot(2,1,1).

PyCharm Tips for This Project

* Debugging: If the solution looks off (e.g., oscillations grow instead of decay), set a breakpoint at solution = odeint(...) and debug (Shift+F9). Inspect state0 and parameters to ensure they’re correct.
* Code Navigation: Right-click odeint and select “Go to Definition” to see its documentation in PyCharm—helps you understand its arguments.
* Plot Customization: PyCharm’s autocompletion will suggest arguments for plt.plot (like linewidth or linestyle). Hover and press Ctrl+Q (Cmd+J on Mac) to see docs.

Your Turn

Experiment with the code in PyCharm:

1. Change the damping coefficient b (e.g., set to 0 for no damping, or 2 for overdamping) and observe the effect on the position and energy.
2. Modify the time span or number of points in np.linspace (e.g., np.linspace(0, 30, 300)).
3. Add a phase-space plot (position vs. velocity) as a new figure: plt.plot(x, v).

Reflecting on Your Progress

You’ve now tackled three projects:

* Project 1: Generating and analyzing a noisy signal (arrays, stats, plotting).
* Project 2: Fitting a curve to noisy data (curve fitting with scipy).
* Project 3: Solving and visualizing an ODE (numerical integration, multi-panel plots).

Each builds on the last, mirroring common scientific workflows you might’ve done in MATLAB but with Python’s tools. You’re getting comfy with numpy, matplotlib, and scipy, and PyCharm’s giving you a solid environment to debug and iterate.

What’s Next?

Since you’re learning by coding, here are some options:

1. Extend This Project: Add an external driving force to the oscillator (e.g.,

F(t)=F0cos⁡(ωt)F(t) = F\_0 \cos(\omega t)F(t) = F\_0 \cos(\omega t)

) and solve for a driven damped oscillator.

1. New Challenge: Analyze a real dataset—like loading experimental data from a CSV file and applying filtering or FFT (signal processing).
2. MATLAB Translation: Got a specific MATLAB script from your past you’d like to port over? Share it, and I’ll help you rewrite it in Python.

What sounds good to you? Or if you’ve got questions about anything in these projects (like odeint details or plotting tricks), just ask!

I want to do options 1 and 2. I also want you to give me projects to help me understand machine learning.

Since you’ve already tackled the damped oscillator project and are ready to roll with Options 1 and 2, plus dive into machine learning, I’ll break this down into three sections: extending the oscillator project with a driving force, analyzing a real dataset, and introducing machine learning projects tailored to your scientific background and Python progress. Let’s keep the hands-on coding vibe going in PyCharm, with explanations to tie it back to concepts you might’ve seen in MATLAB.

Extend the Damped Oscillator with an External Driving Force

Let’s add a sinusoidal driving force to the damped harmonic oscillator, turning it into a driven damped oscillator. The equation becomes:

md2xdt2+bdxdt+kx=F0cos⁡(ωt)m \frac{d^2x}{dt^2} + b \frac{dx}{dt} + kx = F\_0 \cos(\omega t)m \frac{d^2x}{dt^2} + b \frac{dx}{dt} + kx = F\_0 \cos(\omega t)

Where

F0F\_0F\_0

is the amplitude of the driving force, and

ω\omega\omega

is its angular frequency. This models real-world systems like a forced spring or electrical circuits. We’ll solve it using scipy.integrate.odeint, plot the results, and look for interesting behaviors like resonance.

Code: Driven Damped Oscillator in PyCharm

Create a new file (e.g., driven\_oscillator.py) or modify your existing damped\_oscillator.py. Here’s the updated code:

python

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint

# Step 1: Define the system of ODEs with a driving force

def driven\_oscillator(state, t, m, b, k, F0, omega):

x, v = state # state = [position, velocity]

dx\_dt = v

dv\_dt = -(b/m) \* v - (k/m) \* x + (F0/m) \* np.cos(omega \* t) # Added driving force

return [dx\_dt, dv\_dt]

# Step 2: Set parameters and initial conditions

m = 1.0 # Mass (kg)

b = 0.3 # Damping coefficient (kg/s)

k = 2.0 # Spring constant (N/m)

F0 = 1.0 # Driving force amplitude (N)

omega = np.sqrt(k/m) # Driving frequency (rad/s), set near natural frequency for resonance

x0 = 1.0 # Initial position (m)

v0 = 0.0 # Initial velocity (m/s)

state0 = [x0, v0] # Initial state [x, v]

# Step 3: Define time array for solution

t = np.linspace(0, 50, 500) # 0 to 50 seconds, 500 points for smoother plots

# Step 4: Solve the ODE

solution = odeint(driven\_oscillator, state0, t, args=(m, b, k, F0, omega))

x = solution[:, 0] # Extract position

v = solution[:, 1] # Extract velocity

# Step 5: Compute energy (kinetic + potential)

kinetic\_energy = 0.5 \* m \* v\*\*2

potential\_energy = 0.5 \* k \* x\*\*2

total\_energy = kinetic\_energy + potential\_energy

# Step 6: Plot position and velocity

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

plt.subplot(3, 1, 1)

plt.plot(t, x, label='Position (x)', color='blue')

plt.plot(t, v, label='Velocity (v)', color='red')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.title('Driven Damped Oscillator: Position and Velocity')

plt.legend()

plt.grid(True)

# Step 7: Plot energy

plt.subplot(3, 1, 2)

plt.plot(t, kinetic\_energy, label='Kinetic Energy', color='green')

plt.plot(t, potential\_energy, label='Potential Energy', color='orange')

plt.plot(t, total\_energy, label='Total Energy', color='purple')

plt.xlabel('Time (s)')

plt.ylabel('Energy (J)')

plt.title('Energy Over Time')

plt.legend()

plt.grid(True)

# Step 8: Plot phase space (position vs velocity)

plt.subplot(3, 1, 3)

plt.plot(x, v, color='purple', alpha=0.5)

plt.xlabel('Position (x)')

plt.ylabel('Velocity (v)')

plt.title('Phase Space')

plt.grid(True)

plt.tight\_layout()

plt.show()

What’s New Here?

* Driving Force: Added

(F0/m)cos⁡(ωt)(F0/m) \cos(\omega t)(F0/m) \cos(\omega t)

to the acceleration term in dv\_dt.

* Resonance: Set

ω\omega\omega

near the natural frequency

k/m\sqrt{k/m}\sqrt{k/m}

to see amplitude growth (resonance effects).

* Phase Space Plot: Added a third subplot showing position vs. velocity to visualize the system’s dynamics.
* Longer Time Span: Extended to 50 seconds to observe steady-state behavior after transients decay.

Your Turn

Run this in PyCharm and experiment:

1. Change omega to be far from

k/m\sqrt{k/m}\sqrt{k/m}

(e.g., 0.5 or 3.0) and compare the amplitude of oscillations.

1. Increase F0 to see how a stronger driving force affects the system.
2. Vary b (damping) to explore underdamped, critically damped, or overdamped regimes.