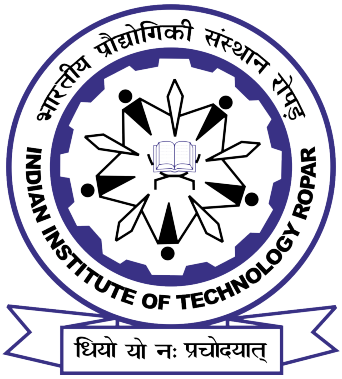

MA515: Foundation of Data Science
Model Discovery for a Damped Oscillator

Project Report

Team - Paach Chawani

Prashant Kumar 2022CSB1202
Moksh Kandpal 2022CSB1093
Pranav Dipesh Bhole 2022CSB1103
Prateek Jagdish Girhepuje 2022EEB1203
Sahil Yadav 2022EEB1210

Indian Institute of Technology, Ropar

Abstract. We use physics-informed neural networks (PINNs) trained per trajectory to obtain smooth continuous estimates of position $x(t)$ and its derivatives. Using those derivatives we apply sparse regression (Elastic-Net/Lasso selection followed by weighted OLS debias) to identify the governing ODE of the damped oscillator in the form

$$\ddot{x} = ax + b\dot{x},$$

map a, b to (ω, ζ) via $a = -\omega^2$ and $b = -2\zeta\omega$, and evaluate parameter recovery error and trajectory fit. The pipeline generates synthetic trajectories (noise optional), learns smooth derivatives using per-trajectory PINNs, performs sparse model selection, debiases coefficients, and validates the discovered ODE by simulation. Results (coefficients, estimated parameters and error metrics) are saved to `dataset/output.json`.

Click here to view full code

Contents

1 Introduction

Model discovery from time-series data is the problem of inferring a compact mathematical model (typically an ODE) that reproduces observed dynamics. For systems where a low-dimensional library of candidate terms is appropriate, combining smooth derivative estimation (here via PINNs) with sparse regression yields interpretable governing equations. This project focuses on the damped harmonic oscillator:

$$\ddot{x} + 2\zeta\omega\dot{x} + \omega^2 x = 0,$$

and demonstrates recovery of the parameters ω (natural frequency) and ζ (damping ratio) from synthetic trajectories.

2 Dataset

2.1 Data-generation process

We generated synthetic, noise-free trajectories of the analytic damped oscillator:

$$x(t) = e^{-\zeta\omega t}\big(A\cos(\omega_d t) + B\sin(\omega_d t)\big), \quad \omega_d = \omega\sqrt{1-\zeta^2},$$

with true parameters:

$$\omega_{\text{true}} = 2\pi \approx 6.283185307179586, \quad \zeta_{\text{true}} = 0.1,$$

and 5 different initial-condition pairs (A, B):

`1: A=25, B=64;  2: A=12, B=-5; 3: A=-20, B=30; 4: A=5, B=15; 5: A=-10, B=-25`

Each trajectory uses $n = 500$ time points on $t \in [0, 10]$ (uniform grid). The pipeline writes ‘dataset/data.csv‘ and ‘dataset/metadata.json‘.

2.2 Why synthetic data

Using analytic trajectories ensures exact ground-truth derivatives are available for debugging. It also isolates errors introduced by derivative estimation and regression from measurement noise.

2.3 Sample of the generated dataset

Table 1 shows trajectory id 1 (with initial condition $A = 25$, $B = 64$). The full CSV contains 5 trajectories with 500 points each.

traj_id	t	x	A	B
1	0.000000	25.00000	25	64
1	0.020040	32.39087	25	64
1	0.040080	39.09146	25	64
1	0.060120	45.01419	25	64
1	0.080160	50.08578	25	64
1	0.100200	54.24810	25	64
1	0.120240	57.45869	25	64
1	0.140281	59.69100	25	64
1	0.160321	60.93446	25	64
1	0.180361	61.19423	25	64

Table 1: sample (first 10 rows) from `data.csv` for trajectory 1.

Only a short preview is shown above for clarity. Use the full `data.csv` file in the project folder for training and experiments.

3 Methodology

The pipeline has three main blocks:

1. **Per-trajectory PINNs:** Train a small MLP per trajectory to learn $x(t)$ as a smooth function, then compute $\dot{x}(t)$ and $\ddot{x}(t)$ with autograd.
2. **Sparse regression:** Build a small library $\Theta = [x,\ \dot{x}]$, target $y = \ddot{x}$, run ElasticNetCV (fallback LassoCV) for feature selection on normalized data.
3. **Debias and map parameters:** After selection, run weighted OLS on selected features (weights $\propto |x|$) to debias coefficients; map a, b to ω, ζ.

3.1 Model form and parameter mapping

We fit the model:

$$\ddot{x} = ax + b\dot{x}.$$

Given a and b, physical parameters are:

$$\omega = \sqrt{-a}, \qquad \zeta = -\frac{b}{2\omega},$$

(valid when $a < 0$).

3.2 Why per-trajectory PINNs

Multiple trajectories share the same time grid but different initial conditions. A single network with input t only cannot represent multiple functions simultaneously (different $x(t)$ at same t), which leads to averaging artifacts and poor derivative estimates. Training one network per trajectory avoids this.

4 Implementation details

All code is in one script ‘discover_oscillator.py‘. Key implementation choices:

- Language / libs: Python 3.8+, PyTorch (float64), scikit-learn, numpy, scipy, pandas.
- PINN architecture (per-trajectory): 3-layer MLP with SiLU activations (hidden=128), output scalar; trained with Adam, LR schedule, typical hyperparameters below.
- Training: epochs $\approx$ 8000 per trajectory, batch size 128, learning rate 1e$-$3 (with scheduled drops).
- Derivatives: computed by PyTorch autograd on evaluation grid (same grid as training).
- Sparse selection: ‘ElasticNetCV(l1_ratio=[0.1,0.5,0.9], n_alphas=60)‘ with 5-fold CV; fallback to ‘LassoCV‘ if ENet fails.
- Debias: weighted OLS (weights $w_i = |x_i| + 10^{-6}$) on selected features.
- Filtering: drop points where $|x| \leq \varepsilon$ (stability); ε chosen small (e.g., 1e$-$6).

4.1 Key hyperparameters

Component	Value
Trajectories	5
Points per trajectory	500
PINN hidden size	128
PINN activations	SiLU
PINN epochs	8000
Batch size	128
Optimizer	Adam (weight decay = 1e$-$8)
Sparse selection	ElasticNetCV (l1_ratio=[0.1,0.5,0.9], n_alphas=60)
Debias	Weighted OLS (weights $\propto |x|$)
Precision	double (torch.float64)

Table 2: Key hyperparameters used in experiments.

5 Algorithm (high-level)

```
1. Generate analytic trajectories for given omega_true, zeta_true, initial A,B.
2. For each trajectory:
   a. Train MLP(t) -> x(t).
   b. Evaluate x(t), compute dx/dt and d2x/dt2 via autograd on dense grid.
3. Stack predictions across trajectories to form Theta = [x, dx/dt] and y = d2x/dt2.
4. Filter points with very small |x| (to avoid ill-conditioning).
5. Normalize Theta columns (mean, std).
```

```
6. Run ElasticNetCV to select terms (fallback Lasso).
7. Debias selected coefficients with weighted OLS.
8. Convert (a,b) to (omega, zeta) and simulate the discovered ODE; compute RMS and NR
9. Save results to dataset/output.json.
```

6 Important code snippets

6.1 Un-normalization and mapping

```
# After fitting on normalized Theta:
# coef_norm = fitted coefficients on normalized Theta
# col_means, col_scales computed from Theta-use
coef_unscaled = coef_norm / col_scales
# intercept adjust (if any)
intercept_unscaled = intercept_norm - sum(col_means * coef_unscaled)

# map to physical parameters:
a = coef_unscaled[0]
b = coef_unscaled[1]
omega_est = sqrt(-a)
zeta_est = -b / (2 * omega_est)
```

6.2 Sparse selection + debias (Python sketch)

```
from sklearn.linear_model import ElasticNetCV, LinearRegression

enet = ElasticNetCV(l1_ratio=[0.1,0.5,0.9], cv=5).fit(Theta_norm, y_use)
coef_norm = enet.coef_
sel_mask = np.abs(coef_norm) > 1e-8
# debias with weighted OLS
lr = LinearRegression(); lr.fit(Theta_use[:,sel_mask], y_use, sample_weight=weights)
coef_debiased = np.zeros(Theta_use.shape[1]); coef_debiased[sel_mask]=lr.coef_
```

7 Results

The script writes ‘dataset/output.json‘. We report the final numbers obtained in a single representative run (values taken from that file).

7.1 Recovered model and parameters

Quantity	True value	Estimated value	Relative error
a (coef of x)	$-39.478\,417\,60$	$-39.256\,910\,89$	0.005 550
b (coef of $\dot{x}$)	$-1.256\,637\,06$	$-1.261\,541\,79$	0.003 917
ω	6.283 185 31	6.265 533 57	0.002 809
ζ	0.100 000 00	0.100 673 13	0.006 731

Table 3: True vs estimated model coefficients and parameters (single-run results from 'dataset/output.json').

Notes:

- The intercept is nonzero in this run. The physical model has zero intercept; small nonzero intercepts often indicate a minor bias introduced by regression/unnormalization or PINN derivative residuals. Enforcing 'fit_intercept=False' in the debias OLS removes this degree of freedom.
- Relative errors for ω and ζ are low (sub-percent), indicating successful recovery.

7.2 Trajectory fit and error metrics

Metric	Value
RMS (discovered vs true trajectories)	0.156 415
NRMSE	0.001 478
Relative error in ω	0.002 809
Relative error in ζ	0.006 731

Table 4: Key error metrics from 'dataset/output.json'.

8 Conclusions

This project demonstrates that combining per-trajectory PINNs with sparsity-promoting regression and debiasing reliably discovers a low-dimensional ODE (damped oscillator) and recovers physical parameters with high accuracy on analytic data. The final pipeline is interpretable and extensible to noisy data and richer libraries.

Appendix: Full JSON output (example)


```
{
  "discovered_ode": "x_ddot = a*x + b*x_dot",
  "coefficients": {
    "a": -39.25691088629143,
    "b": -1.261541794725515,
    "intercept": -4.5494830010151475
```

```
  },
  "estimated_parameters": {
    "omega": 6.265533567565609,
    "zeta": 0.1006731335106126
  },
  "true_parameters": {
    "omega_true": 6.283185307179586,
    "zeta_true": 0.1
  },
  "errors": {
    "rms": 0.15641472566036393,
    "nrmse": 0.001477974598305461,
    "omega_rel_error": 0.00280936161373547,
    "zeta_rel_error": 0.006731335106126002
  },
  "method": "ElasticNetCV"
}
```


Acknowledgements. Code and experiments were implemented by Team - Paach Chawani. The pipeline code is in ‘discover_oscillator.py‘.