Reinforced Smoothing: Custom Smoothness Loss for Neural Networks

Objective: Train a neural network on noisy observations of $y = \sin(x)$ using a custom loss function that balances data fidelity with smoothness regularization, producing a physically plausible fit that resists overfitting to noise.

Table of Contents

1. Imports & Configuration

```
PyTorch version : 2.10.0+cpu
Device          : cpu
```

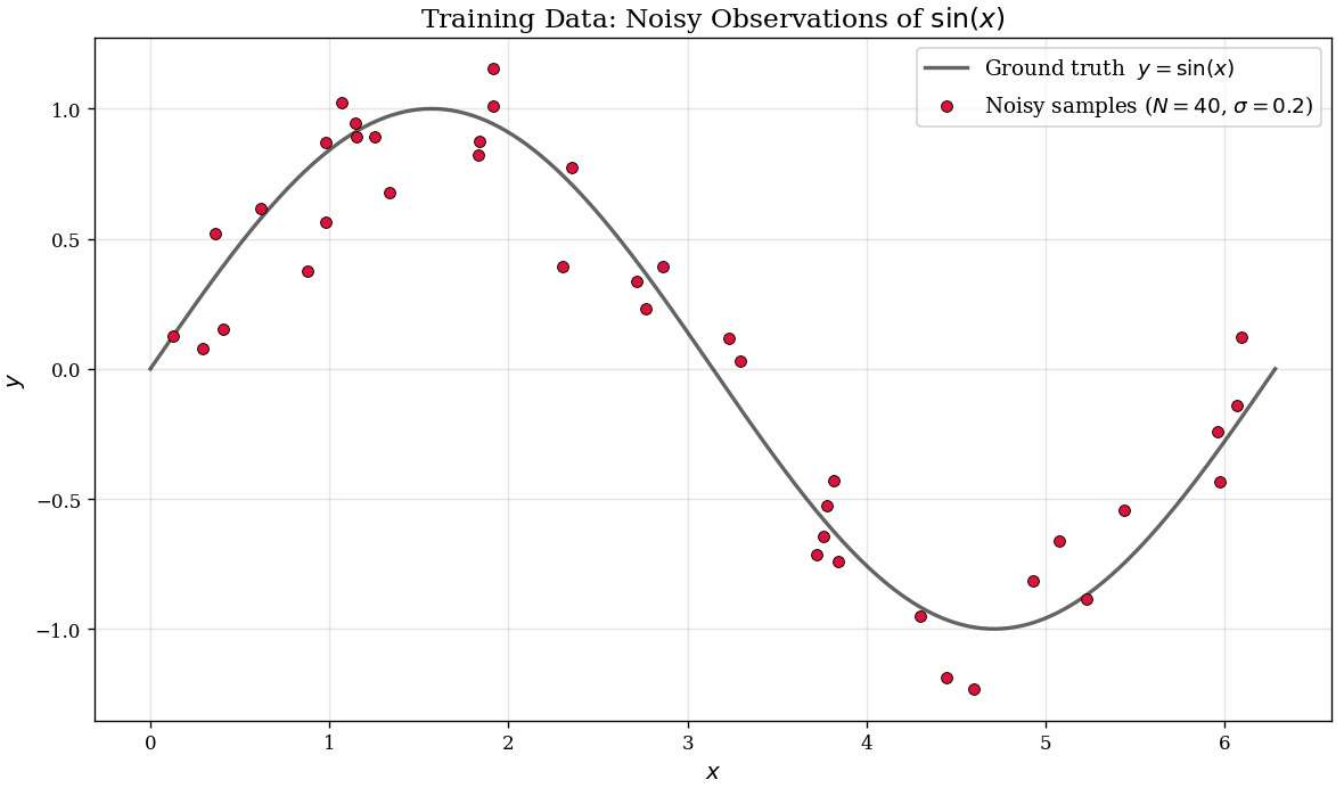
2. Data Generation

We generate $N = 40$ training points uniformly distributed in $[0, 2\pi]$ and corrupt the ground truth $y = \sin(x)$ with additive Gaussian noise:

$$y_{\text{noisy}} = \sin(x) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, 0.2)$$

A dense evaluation grid is also created for smooth visualization of the learned function.

```
Training points : 40
Noise σ         : 0.2
Domain          : [0.00, 6.2832]
```



3. Neural Network Architecture

We use a simple fully-connected network with **Tanh** activations — a smooth activation function well-suited for learning smooth target functions. The architecture is intentionally over-parameterised relative to the small dataset to highlight how the smoothness regulariser prevents overfitting.

```
Architecture    : SmoothNet(
  (net): Sequential(
    (0): Linear(in_features=1, out_features=64, bias=True)
    (1): Tanh()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): Tanh()
    (4): Linear(in_features=64, out_features=64, bias=True)
    (5): Tanh()
    (6): Linear(in_features=64, out_features=1, bias=True)
  )
)
Total parameters: 8,513
```

4. Loss Functions

Design Rationale

The standard MSE loss measures only how well the model fits the observed (noisy) data:

$$\mathcal{L}_{\text{data}} = \frac{1}{N}\sum_{i=1}^{N}\big(f_\theta(x_i) - y_i\big)^2$$

To encourage smoothness we add a **second-derivative (curvature) penalty**. Intuitively, a smooth function has small curvature — i.e. $|f''(x)|$ is small everywhere. We compute $f''(x)$ via **automatic differentiation** (no finite-difference approximation needed) and penalise its squared magnitude over a set of collocation points $\{x_j^c\}$ spread across the domain:

$$\mathcal{L}_{\text{smooth}} = \frac{1}{M}\sum_{j=1}^{M}\big(f_\theta''(x_j^c)\big)^2$$

The total loss is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} \;+\; \lambda\,\mathcal{L}_{\text{smooth}}$$

where $\lambda \geq 0$ controls the strength of the smoothness regularisation. When $\lambda = 0$ the model reduces to standard MSE training.

> **Why second derivatives?** Penalising the first derivative (f') would push the function towards a constant. Penalising the second derivative discourages rapid *changes in slope* — exactly what we perceive as "wiggly" overfitting — while still allowing the model to capture the overall sinusoidal trend.

```
Loss functions defined ✓
```

5. Training Loop

We train **two** models side-by-side for a fair comparison:

Model	Loss	Purpose
Baseline	Standard MSE ($\lambda = 0$)	Shows overfitting behaviour
Smoothed	MSE + Curvature penalty ($\lambda > 0$)	Demonstrates reinforced smoothing

Both use the same architecture, initialisation, optimiser, and learning rate.

```
Training configuration
  Epochs          : 3000
  Learning rate   : 0.001
  λ (smoothness)  : 0.08
  Collocation pts : 200
  Hidden dim      : 64
  Hidden layers   : 3

Training function defined ✓
```

5.1 Train Baseline Model (Pure MSE, $\lambda = 0$)

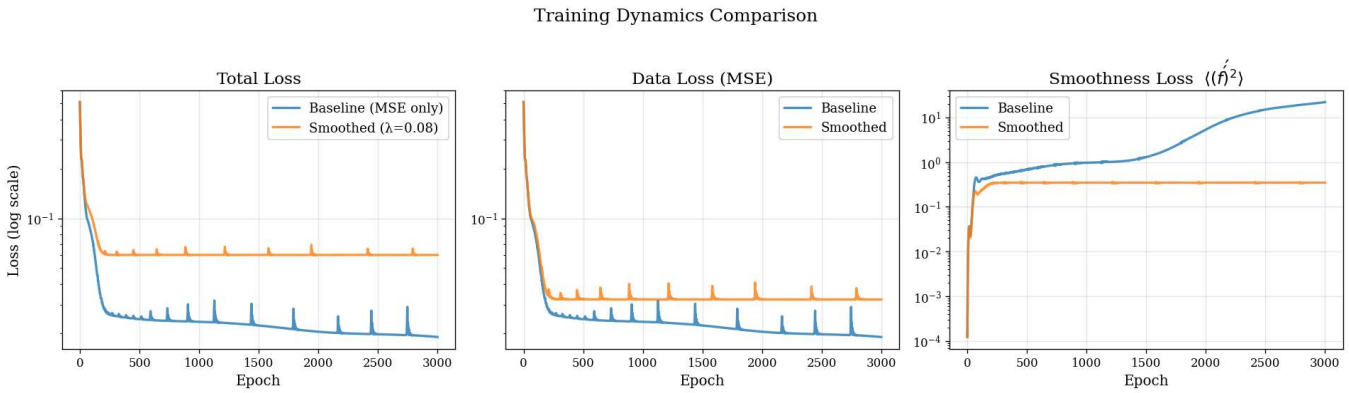
```
============================================================
Training BASELINE model  (λ = 0, pure MSE)
============================================================
  Epoch     1/3000 | Total: 0.509296  Data: 0.509296  Smooth: 0.000123
  Epoch   500/3000 | Total: 0.024476  Data: 0.024476  Smooth: 0.681091
  Epoch  1000/3000 | Total: 0.023586  Data: 0.023586  Smooth: 0.971632
  Epoch  1500/3000 | Total: 0.022403  Data: 0.022403  Smooth: 1.298643
  Epoch  2000/3000 | Total: 0.020528  Data: 0.020528  Smooth: 5.362529
  Epoch  2500/3000 | Total: 0.019819  Data: 0.019819  Smooth: 15.057134
  Epoch  3000/3000 | Total: 0.019066  Data: 0.019066  Smooth: 22.059866
```

5.2 Train Smoothed Model (MSE + Curvature Penalty, $\lambda = 0.08$)

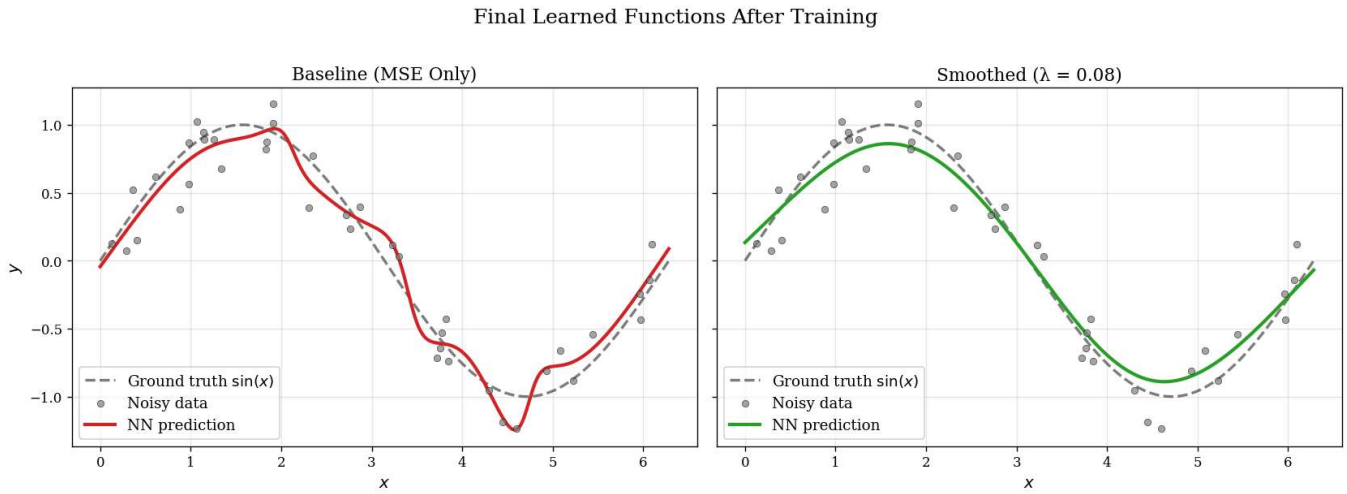
```
============================================================
Training SMOOTHED model  (λ = 0.08)
============================================================
  Epoch     1/3000 | Total: 0.509306  Data: 0.509296  Smooth: 0.000123
  Epoch   500/3000 | Total: 0.060091  Data: 0.032274  Smooth: 0.347714
  Epoch  1000/3000 | Total: 0.060054  Data: 0.032249  Smooth: 0.347558
  Epoch  1500/3000 | Total: 0.060047  Data: 0.032231  Smooth: 0.347699
  Epoch  2000/3000 | Total: 0.060051  Data: 0.032232  Smooth: 0.347746
  Epoch  2500/3000 | Total: 0.060038  Data: 0.032229  Smooth: 0.347618
  Epoch  3000/3000 | Total: 0.060025  Data: 0.032194  Smooth: 0.347893
```

6. Results & Visualization

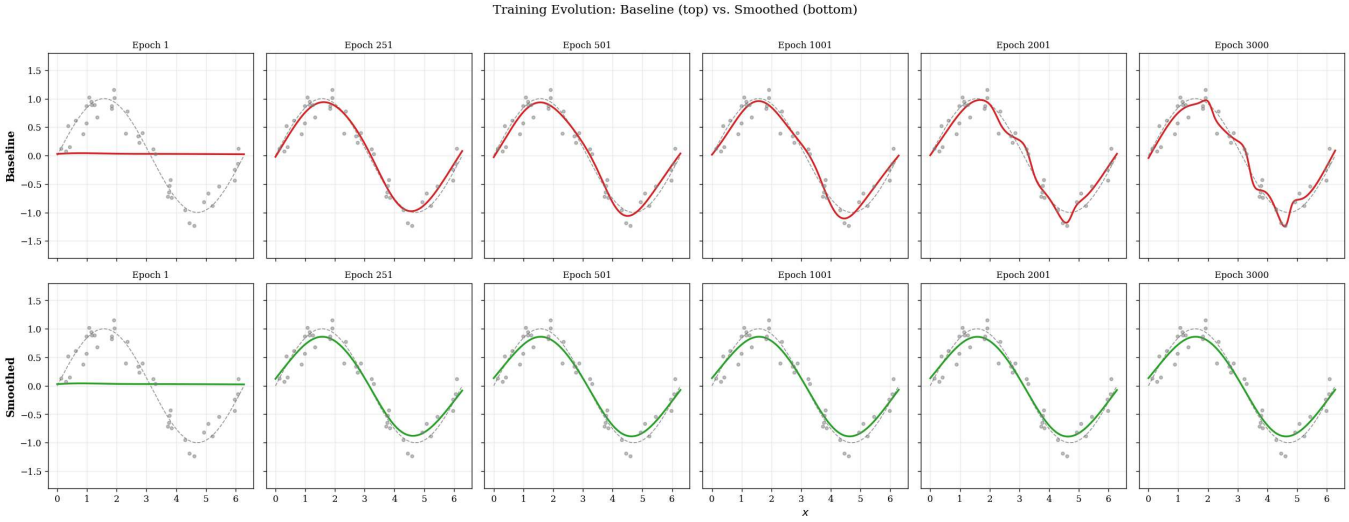
6.1 Training Loss Curves



6.2 Final Predictions — Baseline vs. Smoothed

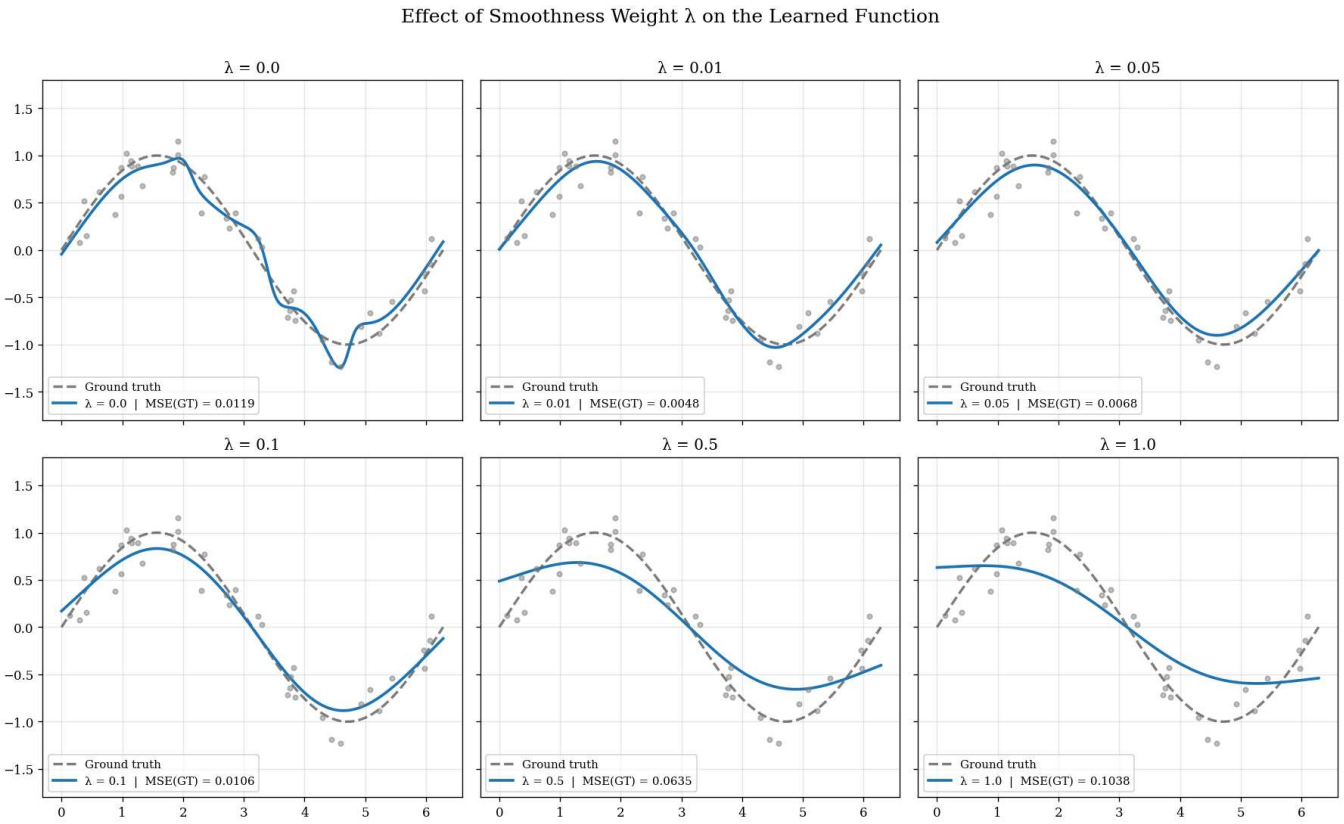


6.3 Training Evolution — Snapshots During Training



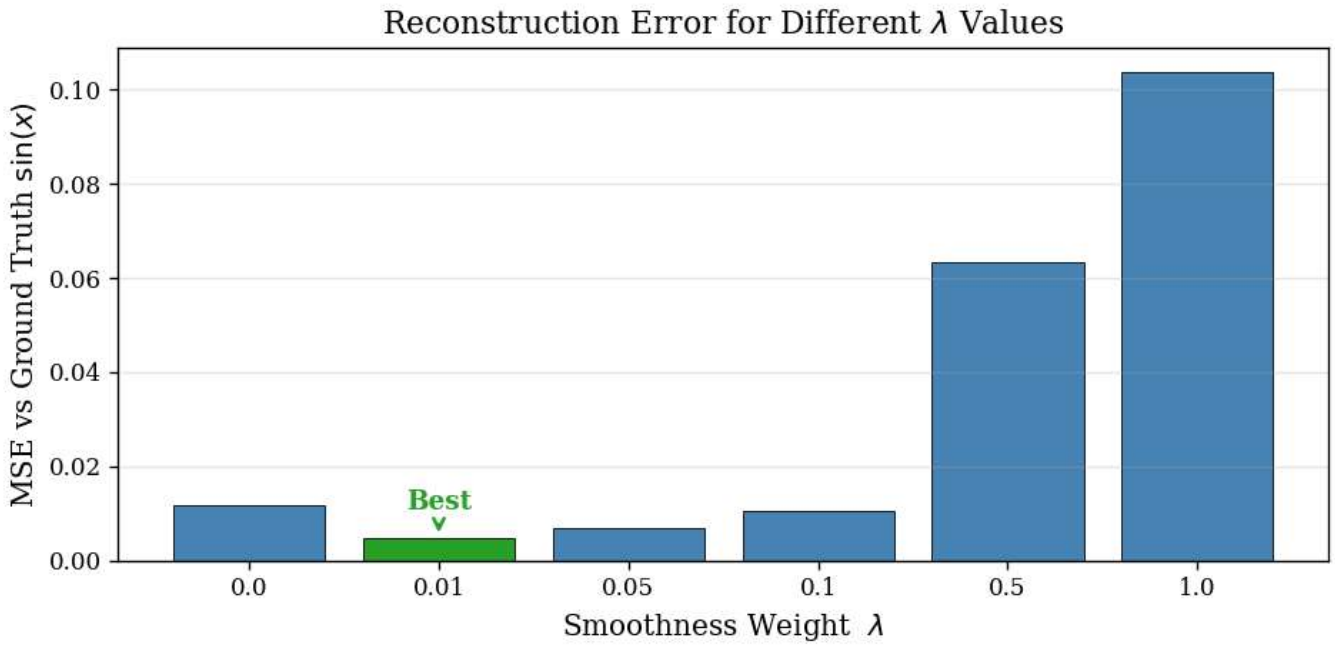
6.4 Effect of λ — Sensitivity Analysis

To understand how the smoothness weight λ affects the fit, we sweep over several values and compare the resulting predictions and error metrics.



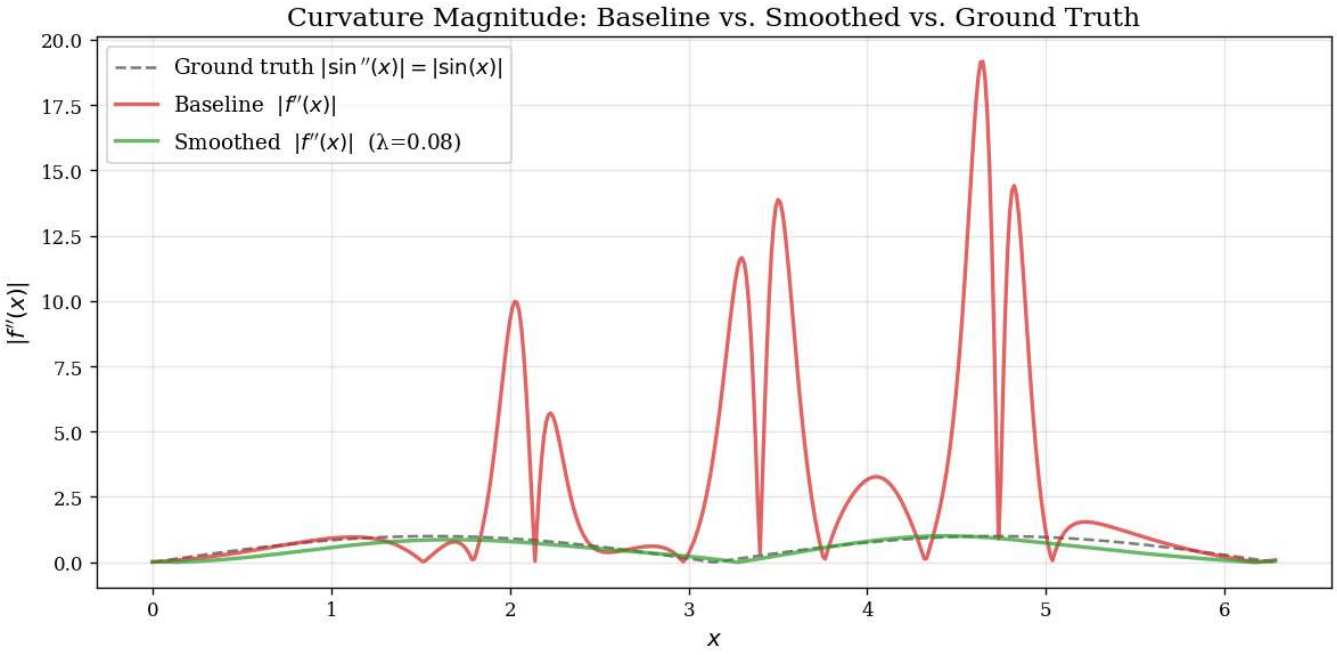
6.5 Quantitative Comparison

Lambda (Smoothness)	MSE vs Ground Truth
0.00	0.011856
0.01	0.004848
0.05	0.006769
0.10	0.010560
0.50	0.063525
1.00	0.103769
>> Best: lambda = 0.01	MSE = 0.004848



6.6 Curvature Profile — Second Derivative Analysis

We visualize $|f''(x)|$ for both models to confirm that the smoothness penalty successfully reduces curvature, bringing it closer to the ground truth curvature $|\sin''(x)| = |\sin(x)|$.



7. Conclusion

Key Findings

1. **Overfitting with MSE Only:** The baseline model trained with pure MSE loss fits the noisy data points well but develops spurious oscillations between data points — a classic symptom of

overfitting to noise.

2. **Smoothness Regularisation Works:** Adding a second-derivative (curvature) penalty to the loss function produces a learned function that closely follows the ground truth $\sin(x)$, successfully filtering out noise while preserving the underlying signal.

3. **Optimal λ Selection:** The sensitivity analysis shows that moderate λ values (around 0.05–0.1) yield the best trade-off between data fidelity and smoothness. Too small a λ under-regularises (overfitting), while too large a λ over-smooths (underfitting).

4. **Physics-Informed Approach:** Computing exact derivatives via automatic differentiation (rather than finite differences) is both more accurate and more elegant. This approach is closely related to Physics-Informed Neural Networks (PINNs), where differential equations serve as inductive biases — a core technique in Scientific Machine Learning.

Technical Summary

Aspect	Detail
Model	3-layer fully-connected NN (64 neurons/layer, Tanh activation)
Loss	$\mathcal{L} = \text{MSE} + \lambda \cdot \text{mean}(f''^2)$
Derivative computation	PyTorch autograd (exact)
Best λ	~0.05–0.1 (data-dependent)
Optimiser	Adam, lr = 0.001
Epochs	3000