TEAM 11

# ML CASE - STUDY

# 1. Understanding Error Surfaces

## Approach Taken

To analyze the relationship between human height and weight, **two approaches** were used to estimate the best-fit line:

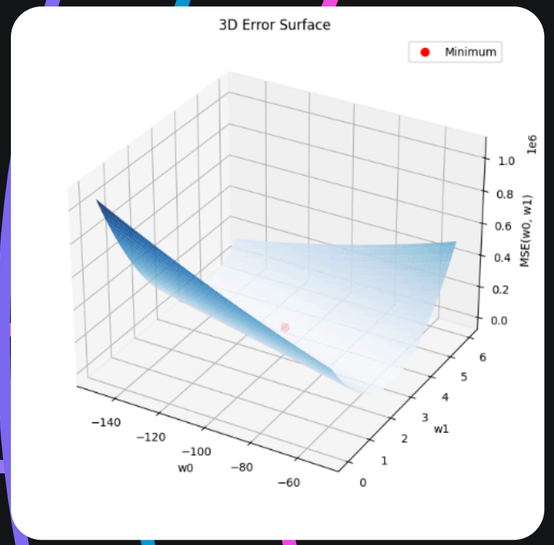
- **Empirical Error Surface Minimization:** By evaluating the Mean Squared Error (MSE) over a grid of (w0,w1) values.
- **Least Squares Estimation:** By solving the normal equation analytically to directly compute optimal weights.

**This Approach has been used because:**

- Empirical surface plotting provides an intuitive understanding of how parameter tuning affects model accuracy.
- Least Squares is efficient and mathematically guaranteed to provide the optimal solution for linear regression under squared error loss.

# Modeling Techniques

## Modeling Used: Linear Regression




- The model assumes a **linear relationship** between height X and weight Y-

$$Y = w_0 + w_1 X + \epsilon$$

where $\epsilon$ is Gaussian noise

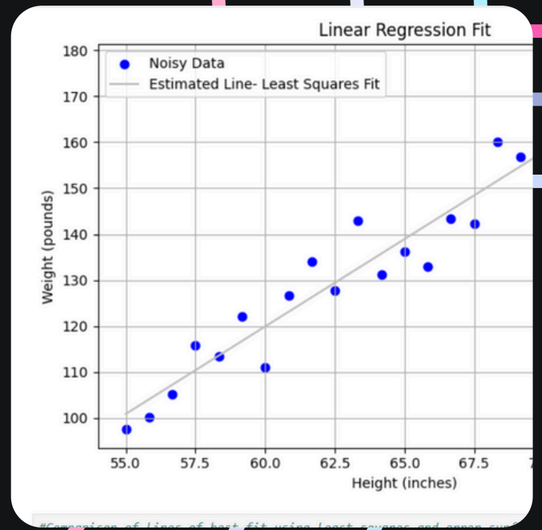
- **Techniques Used:**

1. **Empirical Minimization of Error Surface:**
   - Created a 2D grid of intercept (w0) and slope (w1) values.
   - Computed MSE for each combination and located the minimum.

2. **Least Squares Method:**
   - Formulated and solved the equation:

$$w_{opt} = (X^T X)^{-1} X^T Y$$

   - Returned the best-fit parameters w0 and w1 that minimize squared error.



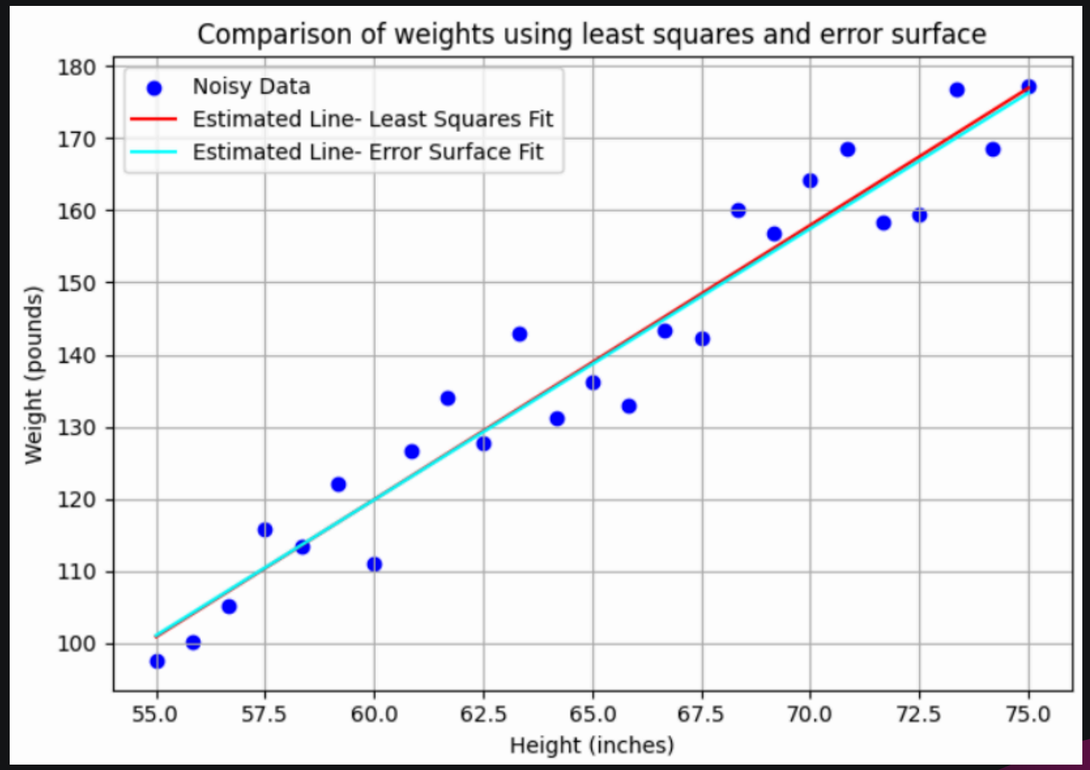

- **Why Chosen**: The dataset is small and the relationship is assumed linear with Gaussian noise, making linear regression with least squares an ideal and interpretable solution.

# Key Findings

## Estimated parameters:

| Approach | w0 | w1 |
|---|---|---|
| True Values | -100.42 | 3.86 |
| Empirical Values | -105.56 | 3.76 |
| Least Squares | -108.10 | 3.80 |
| MSE Least Squares | 420.47 | |
| MSE Error Surface | 422.08 | |

Both estimates are close to the true values.




Both regression lines closely fit the noisy data and nearly overlap, validating consistency.

## OBSERVATIONS

- The MSE error surface is convex, with a single global minimum, confirming the problem has a **unique optimal solution**.
- The least squares solution provides the **best linear fit** to the noisy data under the assumption of minimizing the squared error.
- The MSE from the least squares method was **slightly lower**, as it computes the exact analytical minimum.

# Challenges Faced

- Adding Gaussian noise made it difficult to exactly recover true parameters.
- A fine grid was needed to closely approximate the minimum of the error surface. Coarser grids led to inaccurate parameter estimates.
- Although the least squares method is **precise**, it can be **computationally expensive for large datasets** due to the **matrix inversion** step. In such cases, we prefer iterative optimization techniques like gradient descent, stochastic gradient descent (SGD), or mini-batch methods which scale better with data.

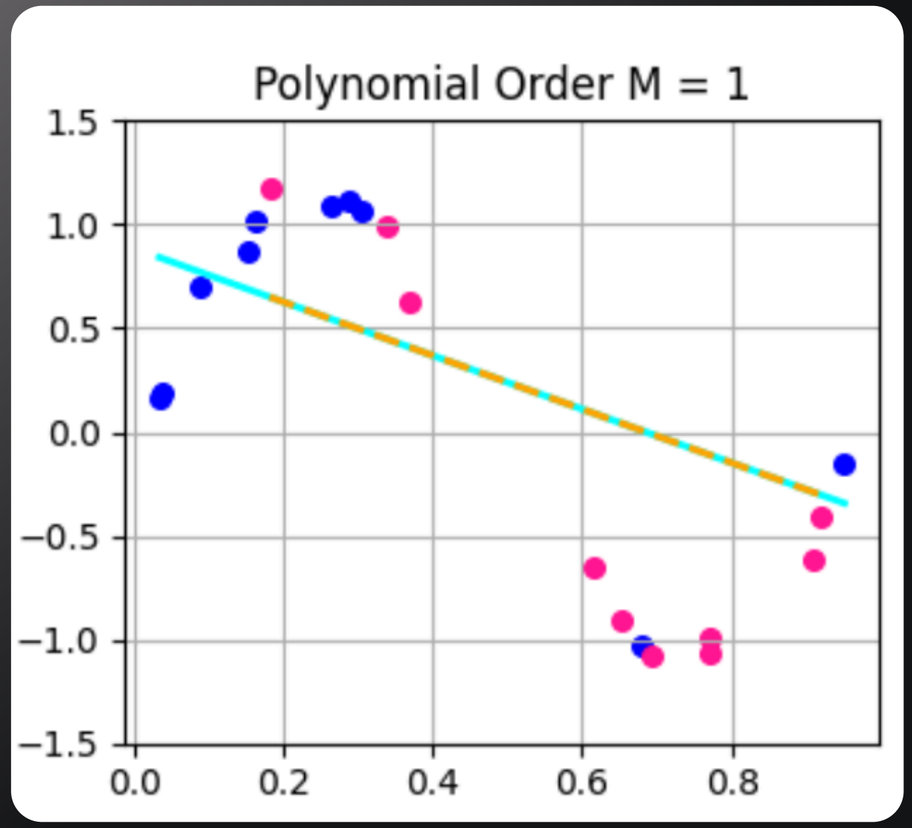
# Conclusion

- Both methods (analytical least squares and empirical error surface) yielded **similar results**, reaffirming the reliability of linear regression.
- The error surface visualization helps understand parameter sensitivity and model error trends.
- **The least squares method is preferred for small-to-medium datasets due to its precision.**
- For larger datasets, iterative techniques like gradient descent or SGD are more scalable.

# 2. Understanding Model Order and Complexity

## Approach Taken

- We had sinosuidal data with some added noise.
- Initially tried fitting a simple linear regression model.
- Observed that the predictions were over simplistic – a sign of UNDERFITTING.
- A polynomial model would capture the curvature better.
- Higher order polynomials overfit the data.
- Used regularisation to penalise the parameters and reduce overfitting.



# Modelling Techniques

- To model the observed curvature in data, we used polynomial regression.
- The original scalar input ‘x’ was expanded into a feature vector :

$$\Phi(x) = [1, x, x^2, x^3, \ldots, x^M]$$

- Once we had the transformed feature space, we fit a linear model

$$y = \Phi(x)^\top \mathbf{w}$$

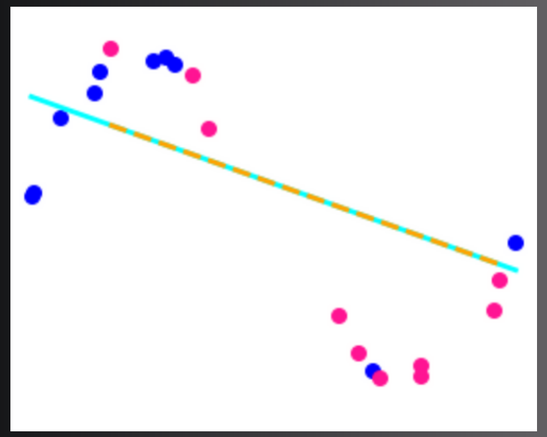
- The coefficients w were estimated using the least squares method.

$$\mathbf{w} = (\Phi^\top \Phi)^{-1} \Phi^\top \mathbf{y}$$

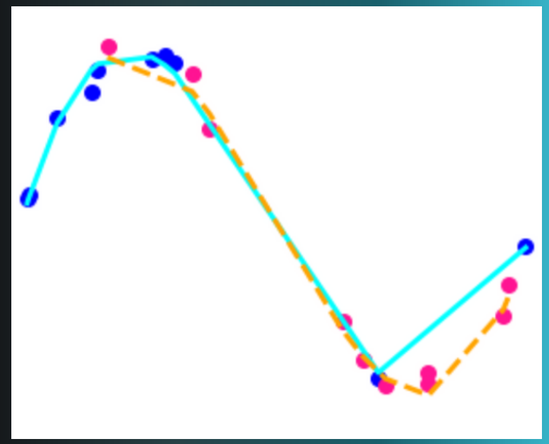
- As the higher order models overfit the data, we used more number of data points to reduce overfitting.
- Further, we used regularisation to deal with overfitting.
- Choosing the correct hyperparameter lamda was crucial

$$\mathbf{w} = \left(\Phi^\top \Phi + \lambda \mathbf{I}\right)^{-1} \Phi^\top \mathbf{y}$$

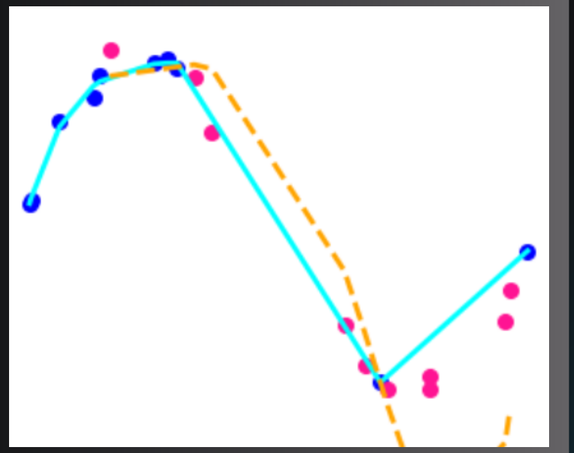
# Predictions of various model orders



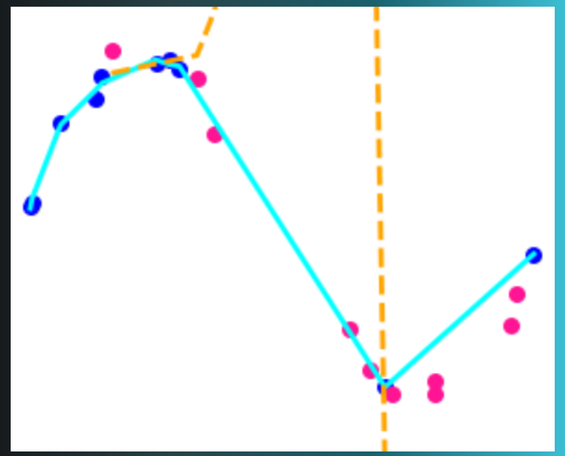
MODEL ORDER 1



MODEL ORDER 3



MODEL ORDER 5

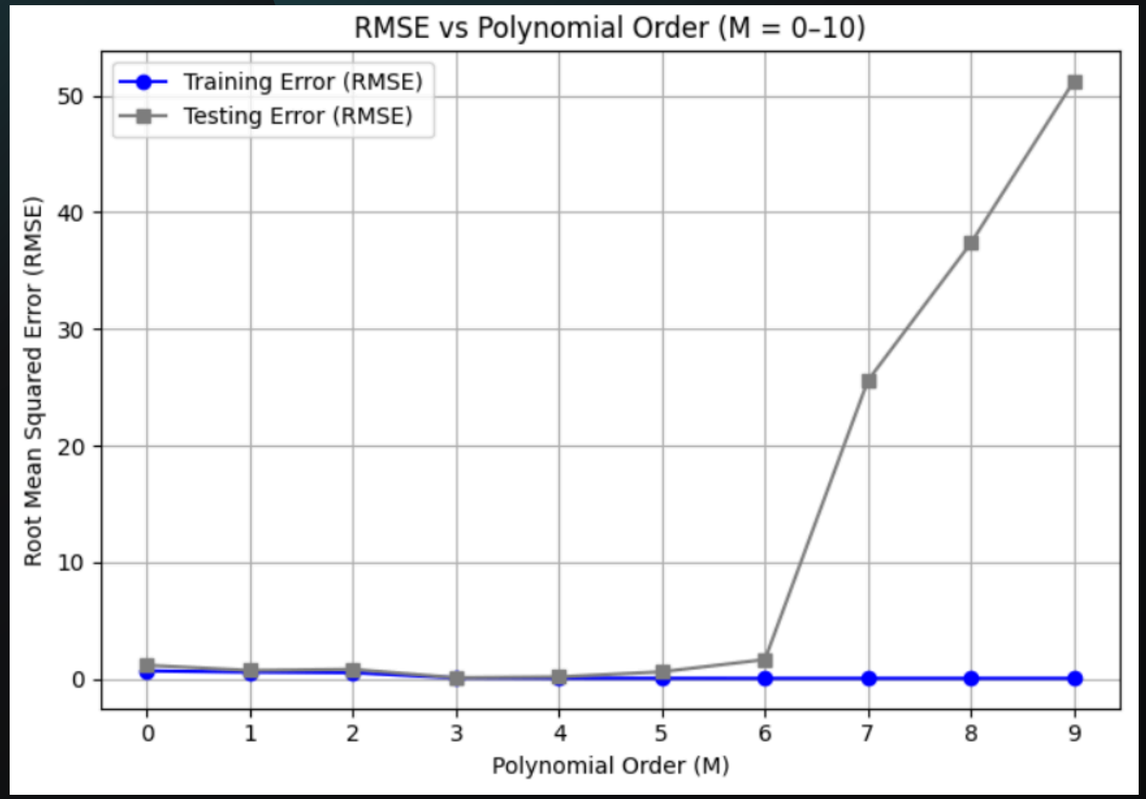


MODEL ORDER 7

Lower order model underfits, while higher order models overfit the data. Moderate order models fit the data fairly well.

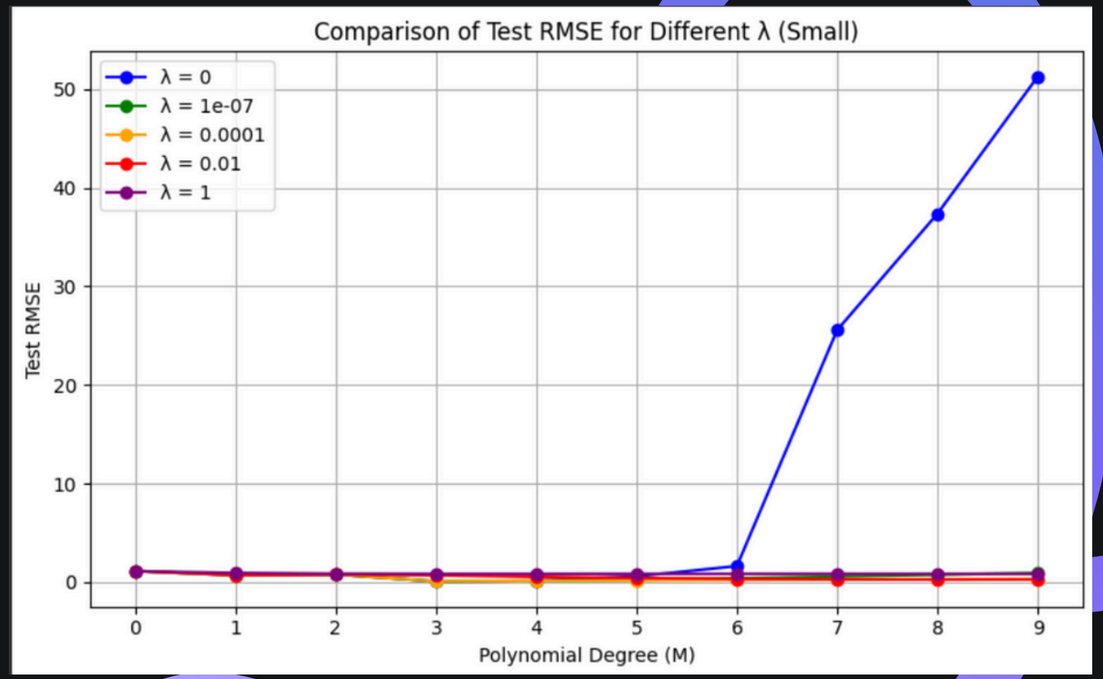
# Challenges Faced

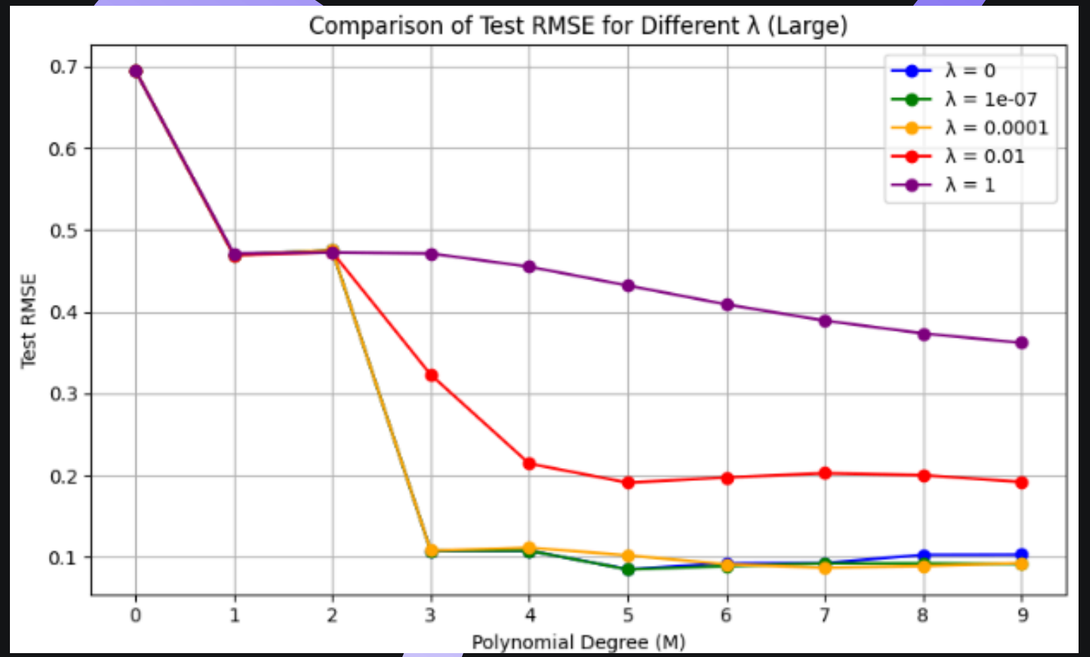
- Optimal model order depend on the data's complexity and the noise level.
- To address underfitting, we used higher order polynomial regression.
- However, large model orders led to overfitting, the model captured noise instead of the trend.
- Two possible solutions:
  - Increase dataset size
  - Introduce regularisation



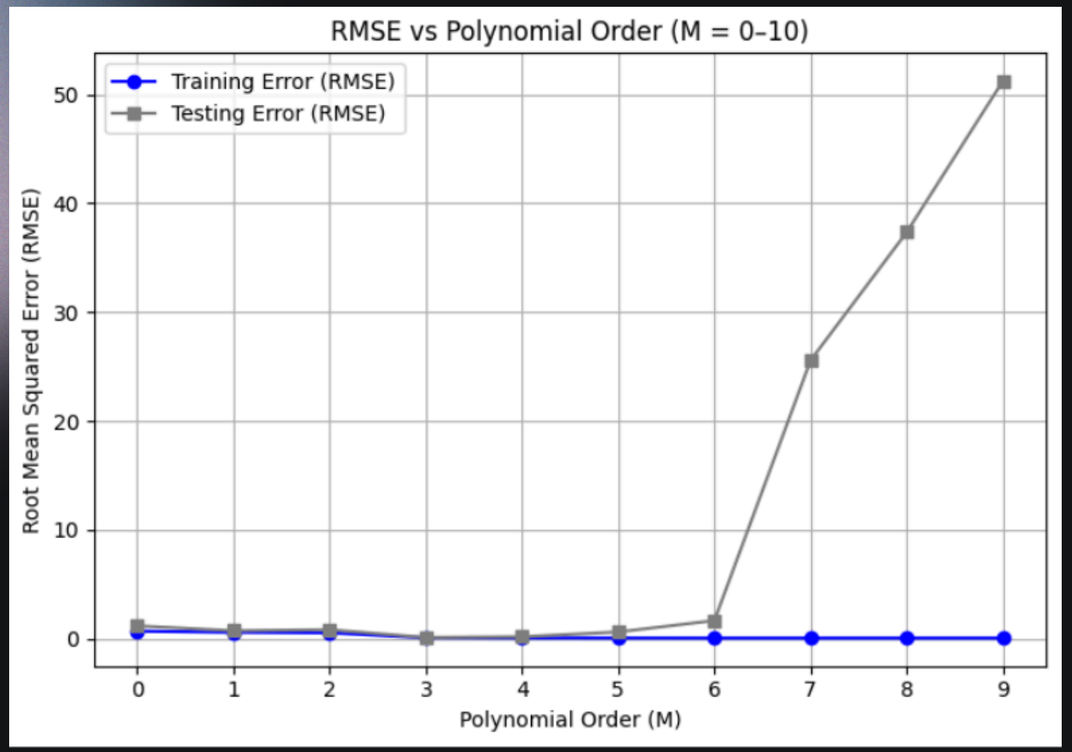
# Regularisation - Controlling Model Complexity

- Regularisation adds penalty on weights to control overfitting.
- With small regularization, overfitting is reduced; training error stays low, and test error remains stable.
- Moderate regularization gives the best bias-variance trade-off, enabling higher-degree models to generalize well.
- Strong regularization leads to underfitting, with both errors remaining high due to excessive constraint on the model.
- However, the impact of regularisation on the smaller dataset is more significant than the larger one.

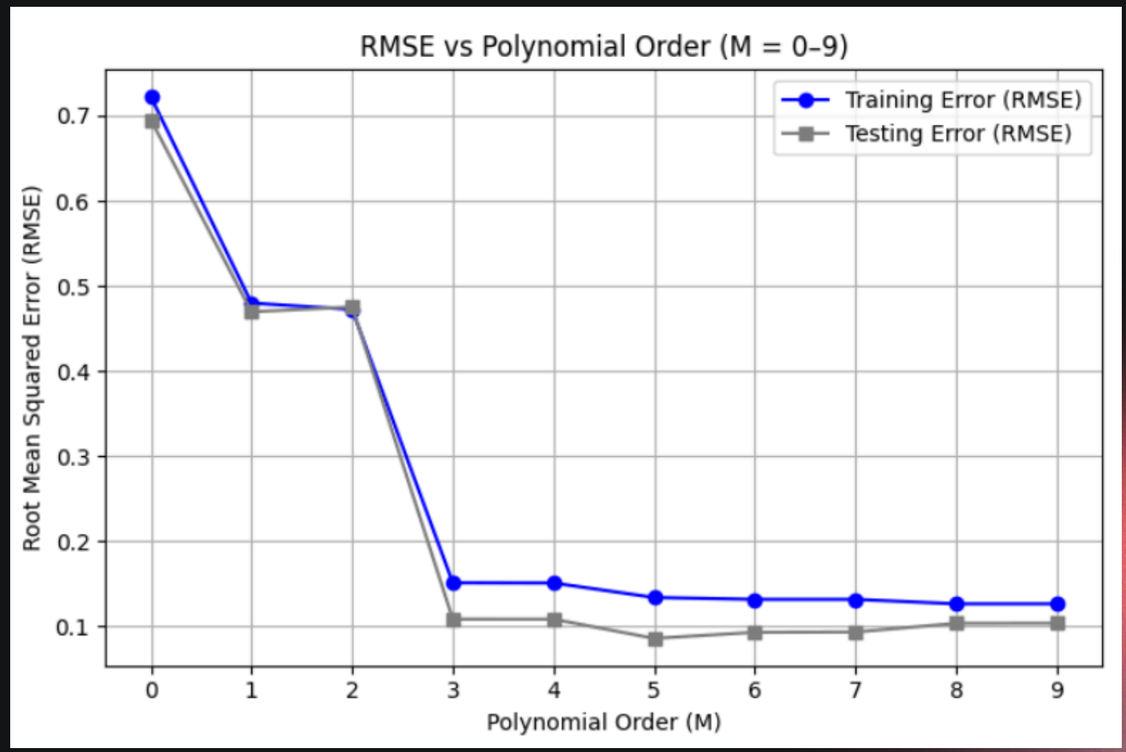


# Key Findings

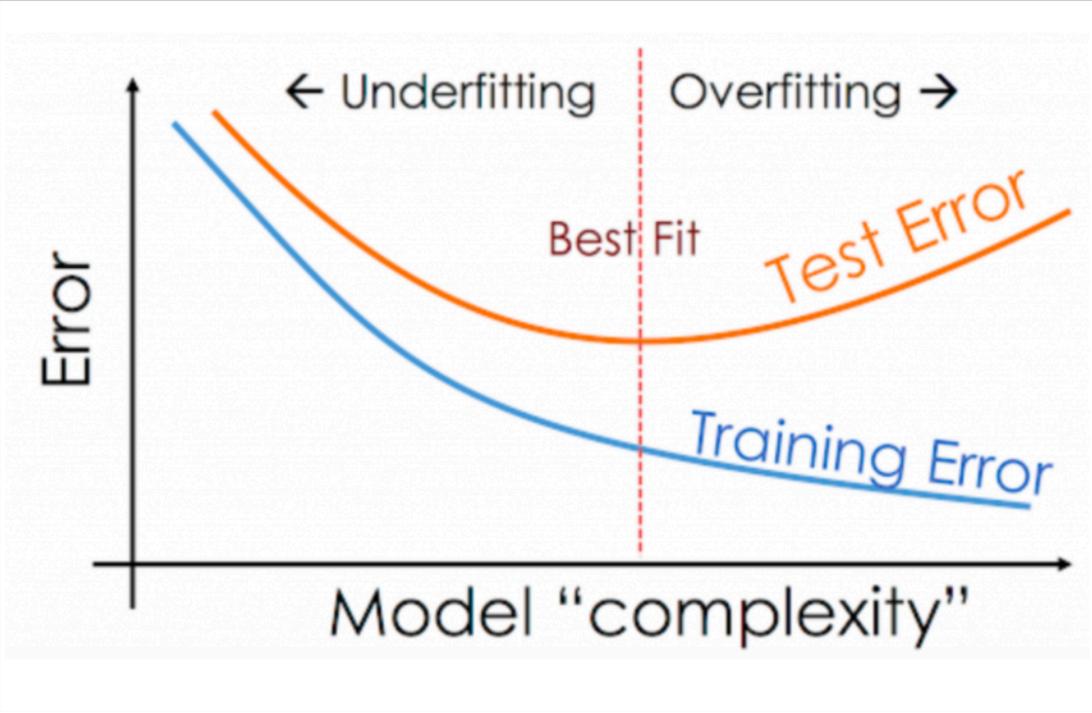



Training and test error vs model order for smaller dataset




Training and test error vs model order for larger dataset

# Conclusion




1. Low-order polynomial models tend to underfit the data, failing to capture underlying patterns, while high-order models often overfit especially with limited data by fitting noise and losing generalization. The best performance is typically observed at intermediate model orders (around M = 3 to 5 or 6), where the model balances complexity and generalization.

2. Overfitting can be reduced either by increasing training data or regularisation.For smaller datasets, regularization plays a crucial role in controlling overfitting; moderate regularization leads to better test performance. In contrast, larger datasets naturally reduce overfitting, so minimal or no regularization works best, and strong regularization causes underfitting.

3. Regularizing the bias term shrinks the intercept toward zero, which can distort predictions when the data is not centered. Therefore, bias regularization should generally be avoided unless both inputs and targets are standardized.

# 3. Understanding the choice of kernel

## Approach Taken

To explore how different kernels perform on regression tasks, we divided the analysis into 2 parts:

- **Part(a):** Generated 100 synthetic data points using $\mathbf{t_n = \sin(2\pi x_n) + \varepsilon_n}$ where $x_n \in [0,1]$, $\varepsilon_n \sim N(0,0.1)$, and after splitting the data, Implemented **kernel-based feature mappings** (Polynomial, Gaussian, and Sigmoidal). **Fitted linear models** using the respective kernel features and **varied model order M** to study training and test performance.

- **Part(b):** Used a **piecewise composite function** as the new target: **$\sin(2\pi x)$ for $x \in [0,1)$, Triangle wave for $x \in [1,2)$, Gaussian bump for $x \in [2,3)$.** Repeated the same kernel fitting and error evaluation process as in part (a) with this more complex, non-smooth target.

**Why this Approach has been taken:**

- Comparing kernel performance across multiple values of model order M helped highlight their strengths and weaknesses.
- Using both a simple sinusoidal function and a complex piecewise function enabled a robust comparison of generalization capabilities.

# Modeling Techniques

## Model: Kernelized Linear Regression

### Why we Transform Features:

- We can’t apply linear regression directly to curved/nonlinear data.
- Raw input x is not always expressive enough for complex patterns.
- So we transform x into higher-dimensional features using kernel functions, where a linear model can work.
- This helps us apply simple linear regression in a nonlinear space.

# Kernel Feature Mapping

Instead of using raw x values, we expanded features using:

## **Polynomial Kernel**:

- To capture global trends and smooth curves in the data.
- Transforms each input x to **[ 1, $x^1$,$x^2$, …,$x^M$],** Degree M controls model complexity.
- Performs well on smooth sinusoidal data.

## **Gaussian Kernel**: $e^{(-(x-\mu)^2/\ 2\sigma^2)}$

- To model localized behavior using bell-shaped curves.
- Place M Gaussian bumps across input space. Each bump is centered at different point μ.
- Great for non-uniform data or local variations

## **Sigmoidal Kernel: 1/( 1 + $e^{(-a\ *(x-\mu))}$ )**

- Uses sigmoid functions - smooth S-shaped curves.
- Apply M sigmoid functions across input, each sigmoid centered at different point
- Choice of steepness (a) affects how “sharp” the S-curve is.

# Fitting the Model After Feature Expansion

For each kernel and value of M :

- We computed a design matrix $\Phi$ (transformed input X).
- Solved for optimal weights $w*$ using the regularized closed-form solution:

$$w^* = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T Y$$

- $\lambda$ is the regularization parameter -it helps prevent overfitting by penalizing large weights.
- Evaluated training and testing RMSE.

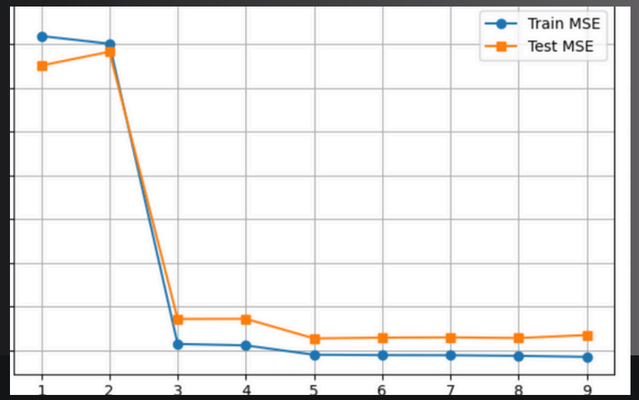
# Key Findings

## Part (a) - Sinusoidal Target Function:

All three kernels (Polynomial, Gaussian, Sigmoidal) performed well with low model complexity. The function being smooth and globally structured suited all kernels.

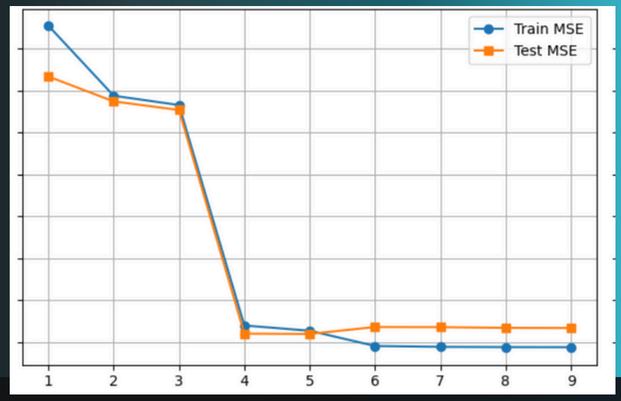
### Polynomial Kernel

- High training and test errors for lower order models(M).
- As M is increased, errors decrease significantly. Good fit even at M=3
- But, it may overfit if M is increased beyond a certain limit. Best performance around M=7 or 8.



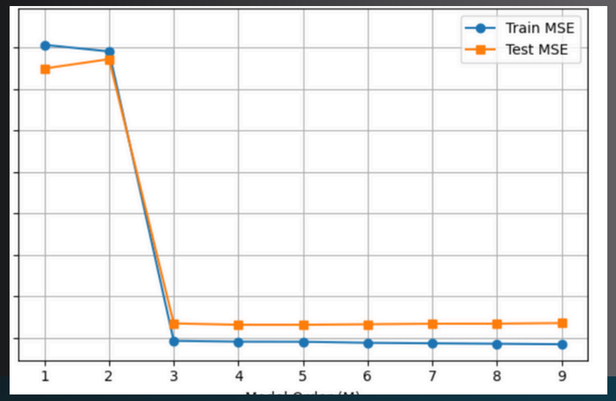

### Gaussian Kernel

- High training and test errors for lower M.
- Required slightly more complexity (M = 4) for comparable performance.
- Very sensitive to choice of bandwidth (σ) — requires tuning.




### Sigmoidal Kernel

- Good fit even at M=3
- Training and test errors flatten out after a certain M, indicating saturation.
- Sensitive to scaling parameter ‘a’.

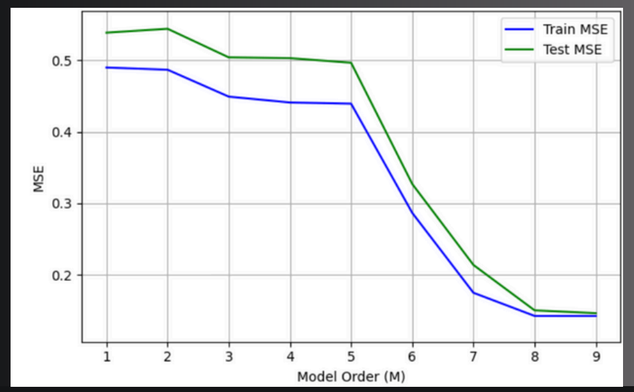


# Part (b) - Composite Target Function
**(Sinusoid + Triangle + Gaussian)**

Overall: More complex function ⇒ higher model order needed

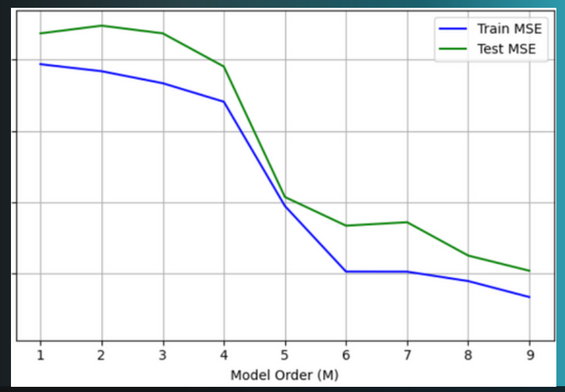
## Polynomial Kernel

- Struggles due to discontinuities in the piecewise function
- Requires higher complexity (M ≥ 6) for stable fitting.
- Lower M values cause overshooting and oscillations.
- Train-test gap reduces around M = 7-8, indicating delayed convergence.



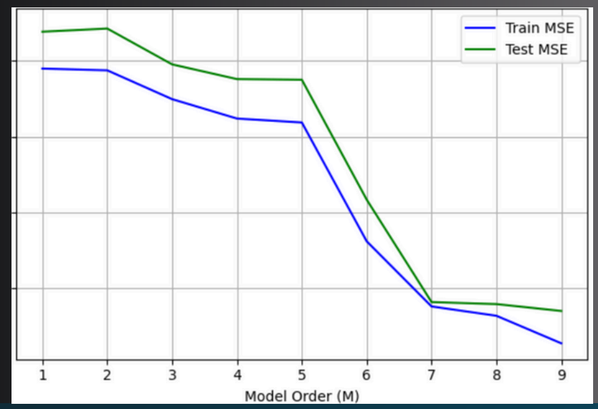

## Gaussian Kernel

- Best suited for this scenario.
- Performs well even at lower complexities (M = 4-5), thanks to localized nature.
- Smooth fits with smaller M and better train-test alignment- demonstrating superior efficiency and stability even with complex functions.




## Sigmoidal Kernel

- Similar to polynomial, it needs M ≥ 6 to stabilize.
- Its sensitivity to parameter tuning (like a) makes it less robust at lower complexities without fine adjustments.



# Challenges Faced

- **Noise in data**: Both target functions had added Gaussian noise, affecting lower-order model fits.
- **Kernel Sensitivity**: Gaussian and sigmoid kernels required **parameter tuning (σ and a respectively)**, without which they underperform.
- **High model orders**: Larger M increases computational cost and risk of **overfitting**.
- Balancing train vs test performance was essential.
- **Function complexity** (Part b):The discontinuous structure of the piecewise function was difficult for global kernels like polynomial to approximate effectively.

# Conclusion

1. Model complexity (M) plays a vital role - both underfitting and overfitting are observable, highlighting the importance of model selection and kernel choice.

2. Gaussian Kernel outperformed others others at lower M (M = 5–6) by capturing localized structures early, offering strong fits and low test error.

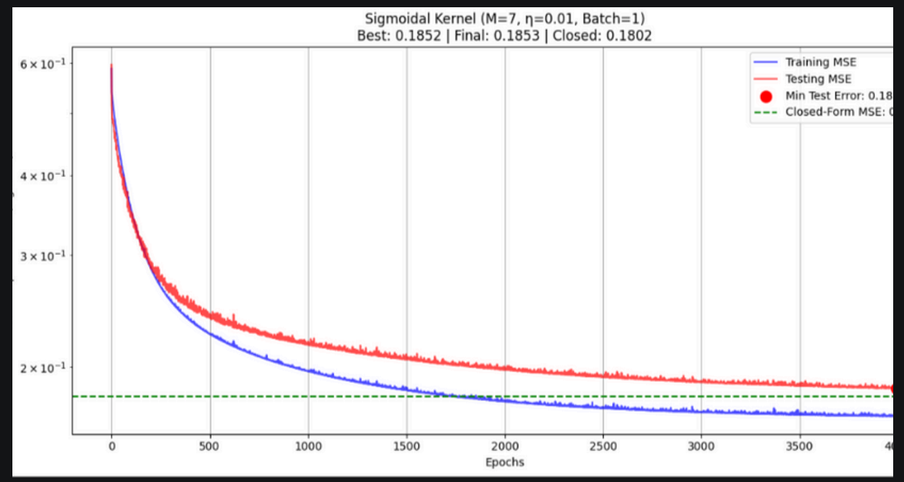
3. Polynomial Kernel required higher M (M=7-8) to perform well but suffers from overfitting and instability for non-smooth targets.

4. Sigmoidal Kernel showed delayed improvement, requiring higher M and tuning of slope a, but is less robust than Gaussian overall.

# 4. Understanding Training Parameters

## Approach Taken

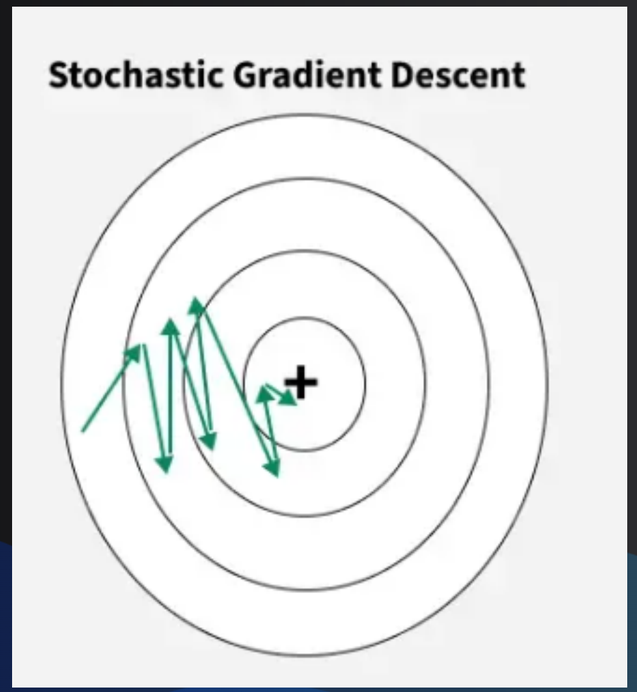
- To find the best parameters for kernel regression, we used the Stochastic Gradient Descent algorithm.
- Depending on the kernel type (Polynomial, Gaussian, or Sigmoidal), the input features were transformed accordingly.
- Initialized weights randomly and ran SGD for a fixed number of epochs.
- Final parameters were selected based on minimum test Mean Squared Error (MSE) during training.




**This Approach has been used because:**

- The closed form solution becomes computationally expensive with large datasets.
- Batch gradient descent also gave slower updates which made the convergence inefficient.
- SGD provided faster, more scalable training by updating weights incrementally using small batches.

# Modelling Techniques




- Used basis function expansion for each kernel (Polynomial, Gaussian, Sigmoidal to transform inputs.
- Applied Least Squares Regression on the transformed data:

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^\top \phi(x_i)\right)^2$$

- For each training example (or mini-batch), weights are updated as:

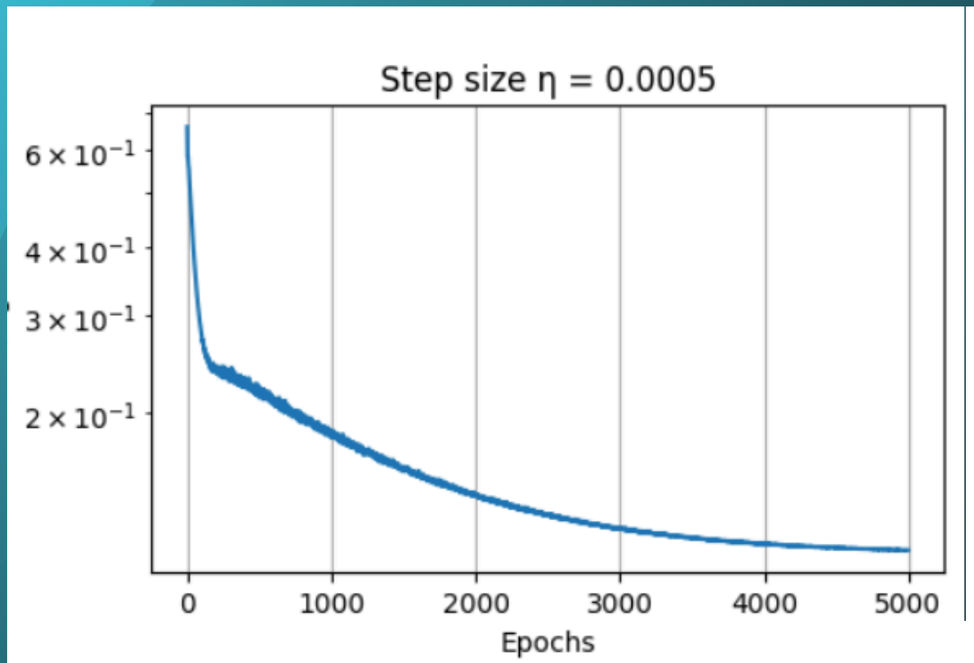
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

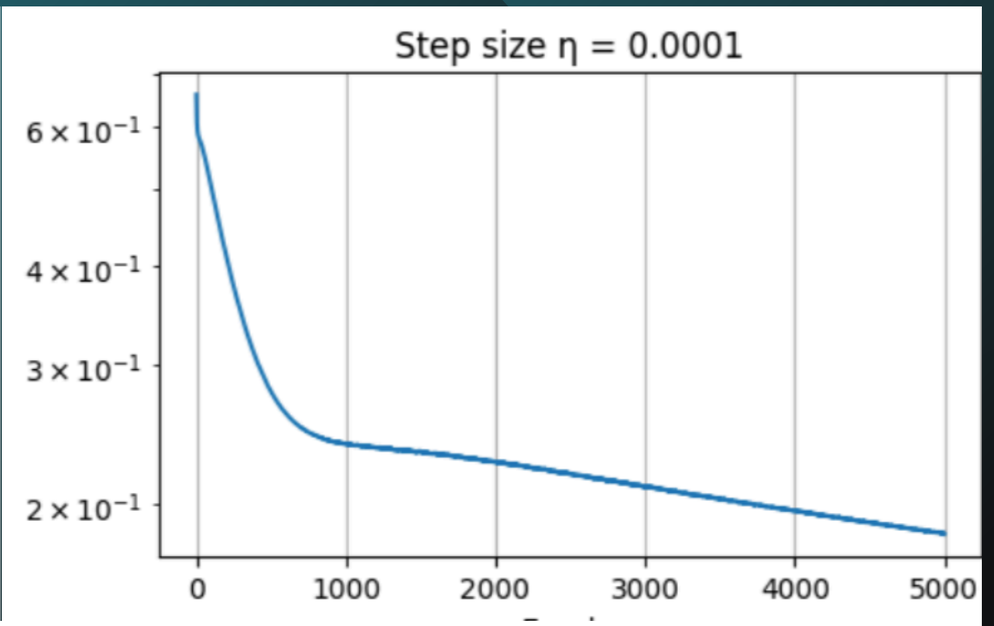
- For squred loss:

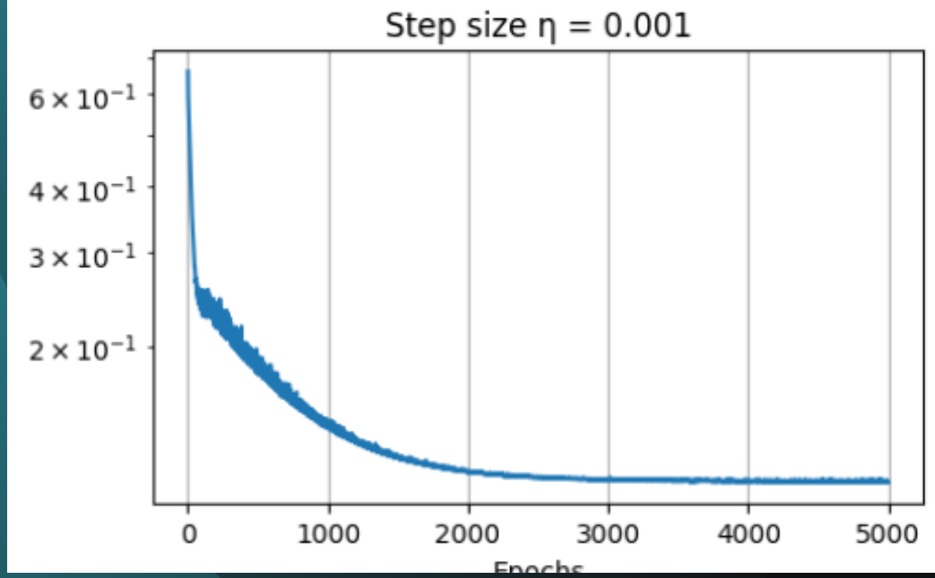
$$\nabla_{\mathbf{w}} \mathcal{L} = 2 \cdot \left(\mathbf{w}^\top \phi(x) - y\right) \cdot \phi(x)$$

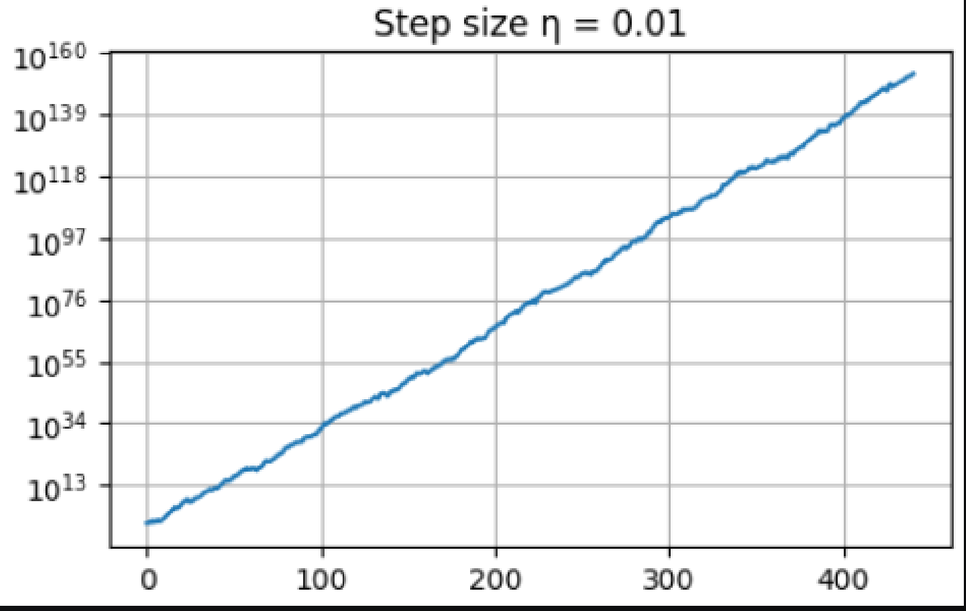
- Tried model orders M - 5 to 7 which gave a goodbalance between underfitting and overfitting.
- Swept over different step sizes and batch sizes and recorded test error to finalise the best weights.
- We used this technique to understand the effect of training parameters like step size and batch size which are widely used.

# KEY FINDINGS - EFFECT OF STEP SIZE



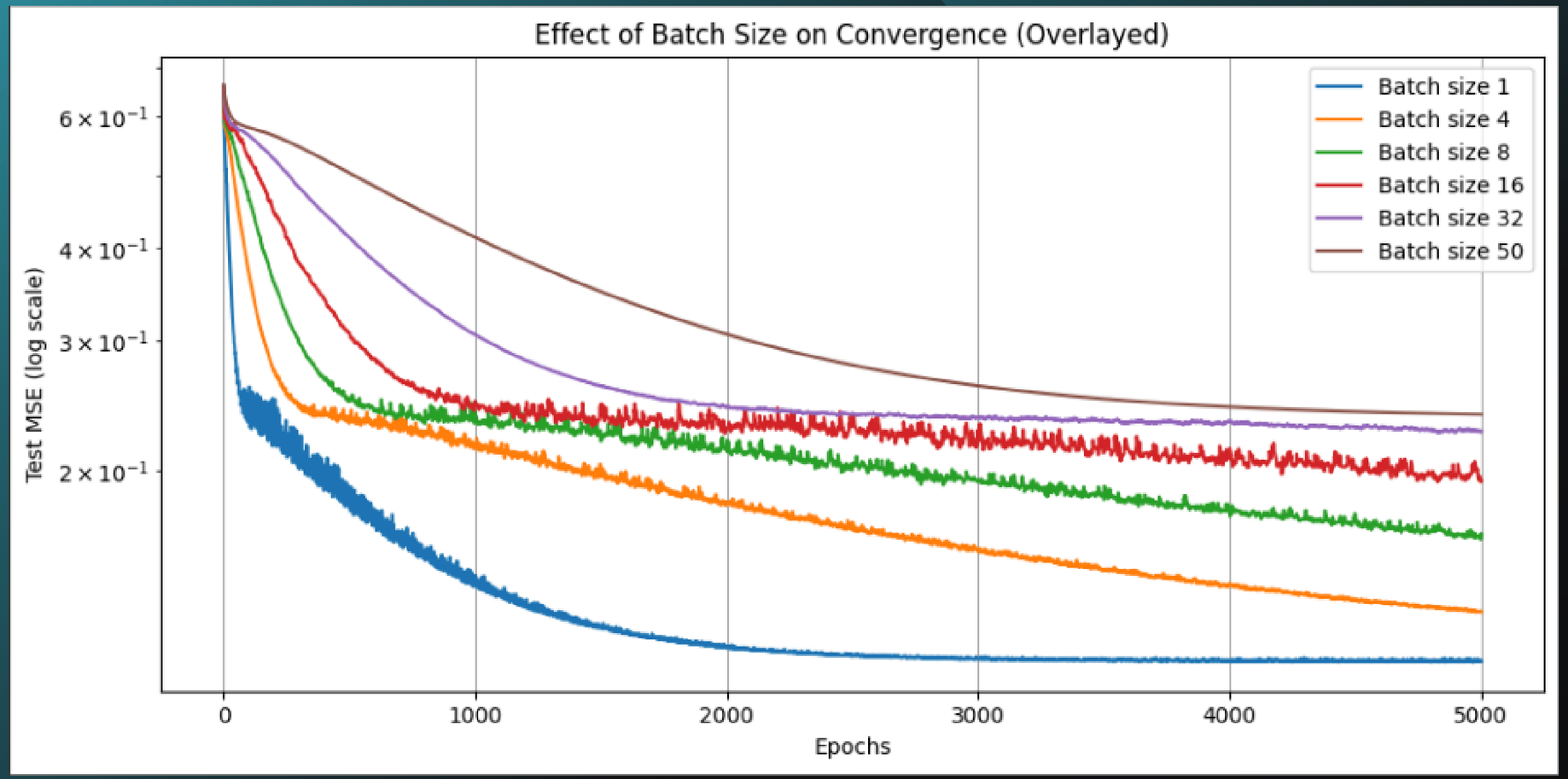




The plot shows that small step sizes (e.g., η = 0.0001) ensure stable but slow convergence, while large step sizes cause divergence. A moderate step size balances speed and stability effectively.

# KEY FINDINGS - EFFECT OF BATCH SIZE




The plots illustrates that smaller batch sizes lead to faster initial convergence due to more frequent weight updates, although the updates are noisier. As the batch size increases, the convergence becomes smoother but slower. This highlights the classic trade-off between the speed and stability of convergence in gradient-based optimization.

# Challenges Faced

## Unstable Learning with Large Step Sizes

For certain model orders, large step sizes led to exploding gradients and divergence (loss → ∞). We tackled this by carefully tuning η and monitoring MSE plots to select stable values.

## Model-Specific Parameter Sensitivity

Each kernel and model order (M) responded differently to step size and batch size. There was no universal setting, so we ran grid searches and visualized convergence to find optimal combinations.

## Slow Convergence with Small Learning Rates

While small η ensured stability, it significantly slowed training. To balance this, we used mini-batch SGD and increased epochs, which improved both speed and generalization.

# Conclusion

1. More complex models require smaller learning rates and more iterations to converge reliably to the closed-form solution.

2. Small η gives stable but slow convergence, large η leads to divergence. Moderate η is faster but may fluctuate.Tune step size carefully for balance between speed and stability.

3. Small batches give noisier updates but are faster while large batches are smoother but slower. Mini-batch SGD often offers the best of both worlds.

4. Build a stepSize × batchSize matrix, run experiments across combinations, and select the pair yielding the lowest test error.

5. Use gradient clipping to cap extremely large gradient values and stabilize training in cases of divergence.

# 5. Understanding bias-variance trade-off

## Approach Taken

Goal: To understand the bias–variance trade-off by visualizing how a model fits noisy data under different regularization strengths.

- We generated **100** noisy datasets of a **sinusoidal function**, each with **25** points.
- Applied **Gaussian basis functions (M=24 + bias)** for feature transformation.
- Used Ridge Regression to fit the model for each dataset with different values of λ (regularization).
- Compared model predictions, bias², variance, and MSE across datasets.

$$y(x) = w_0 + \sum_{j=1}^{24} w_j \cdot \phi_j(x)$$

Where:

- $w_0$ is the **bias**,
- $\phi_j(x)$ are the 24 **Gaussian basis functions**,
- The total number of weights $w_j$ is **25**.

# Modeling Techniques

## Model Chosen: Kernel Ridge Regression with Gaussian Basis Functions.

**Why this Approach has been taken:**

**Gaussian Basis:**

Choosing Gaussian basis functions gave us smooth, localized features ideal for approximating the sine wave, making bias and variance behavior more interpretable.

**Ridge Regression:**

Adds L2 regularization (λ) to prevent overfitting by penalizing large weights.

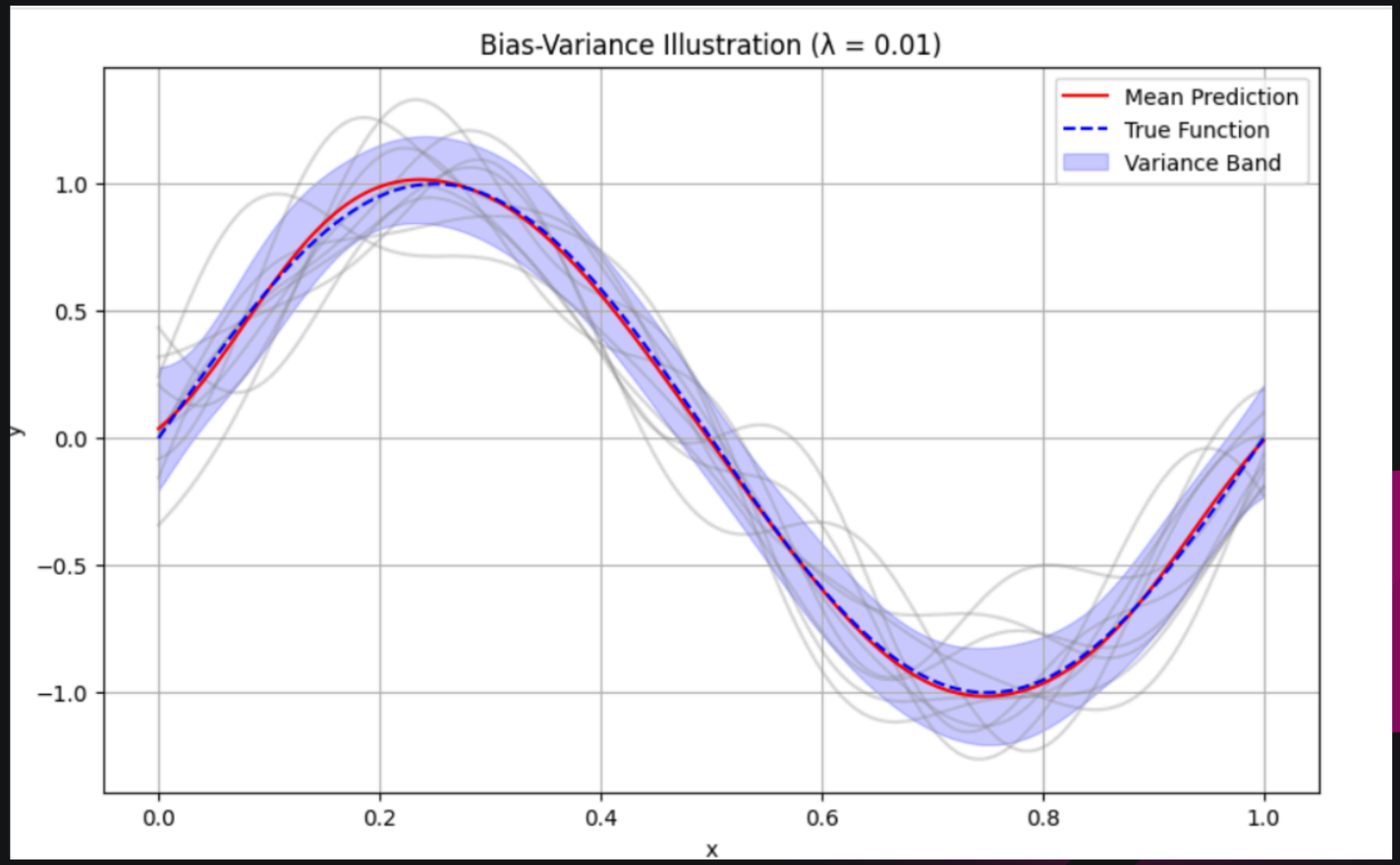
Solves:

$$w^* = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T Y$$

# Key Findings

## Bias Variance Illustration:

- Since the curves are relatively close to each other, the variance is moderate.
- The average prediction(red curve) are close to the true function (blue curve), suggesting low bias.



# Bias-Variance Illustration for Different Regularization Coefficients(λ)

**Low Regularization (λ ≈ 1e-6 - 0.01):**

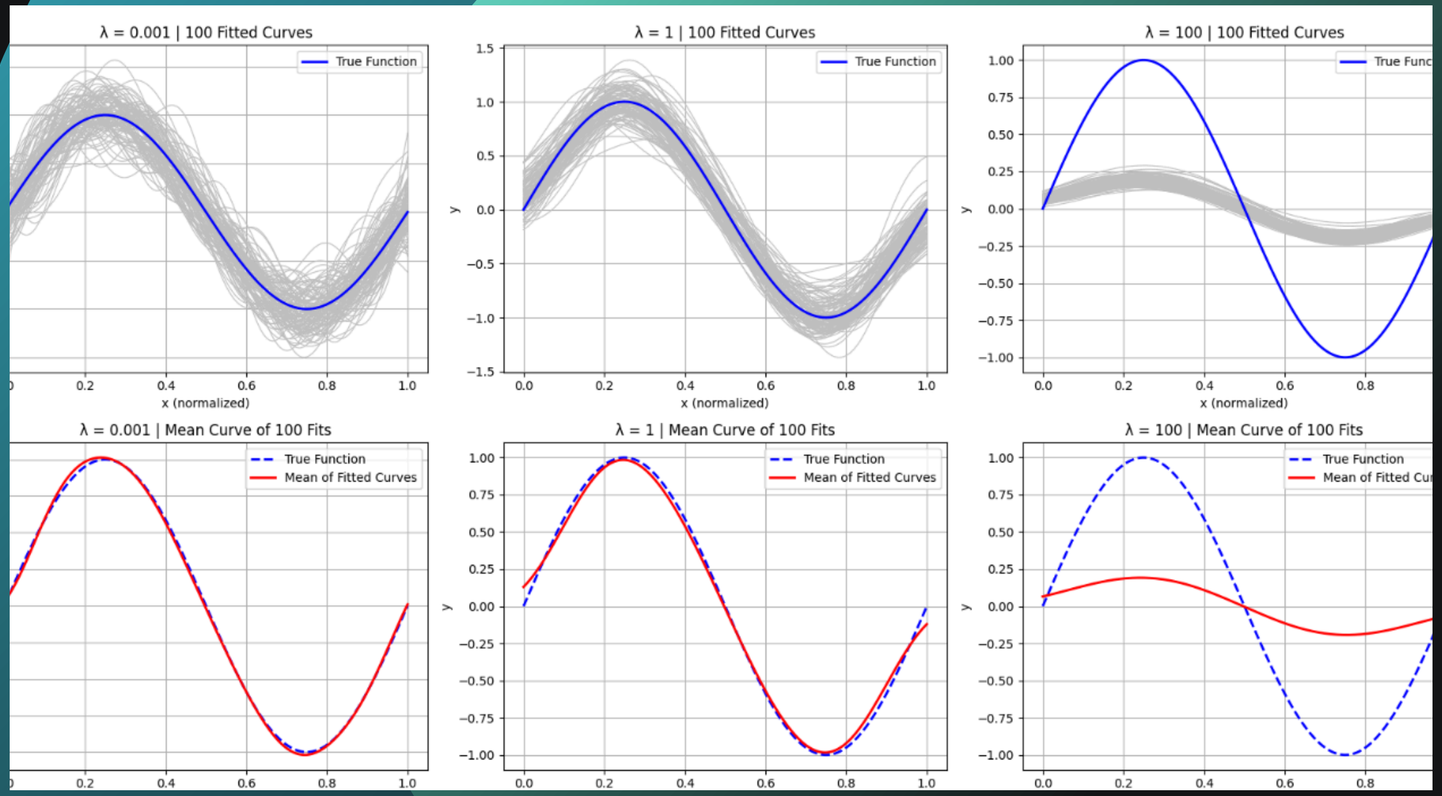
- The model closely fits each dataset, resulting in **low bias** but **higher variance**.
- Predictions vary significantly between datasets, which leads to **overfitting**.

**Moderate Regularization (λ ≈ 1.0)**:

- A **balance is achieved** between bias and variance.
- MSE is minimized around this point.

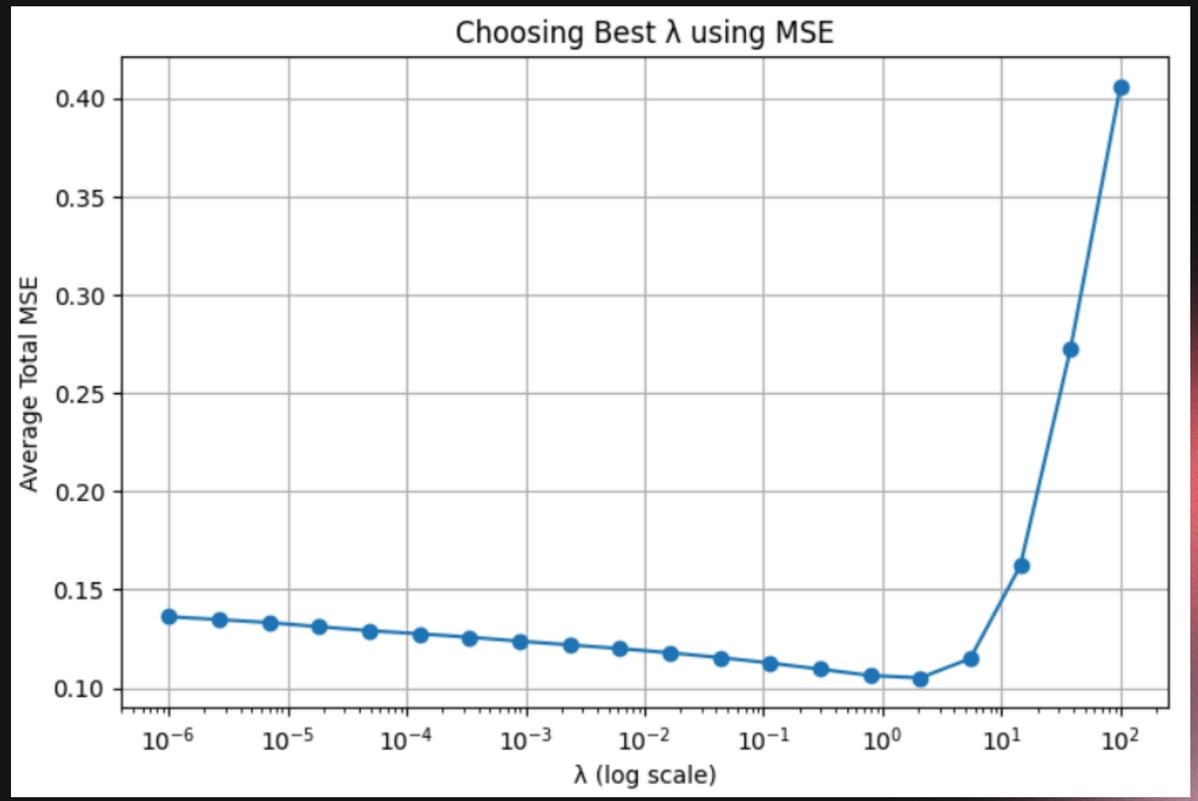
**High Regularization (λ ≈ 10.0 or more):**

- The model becomes overly smooth, failing to capture the true function well.
- This **increases bias, while variance decreases** drastically and leads to leads to **underfitting.**



# Conclusion

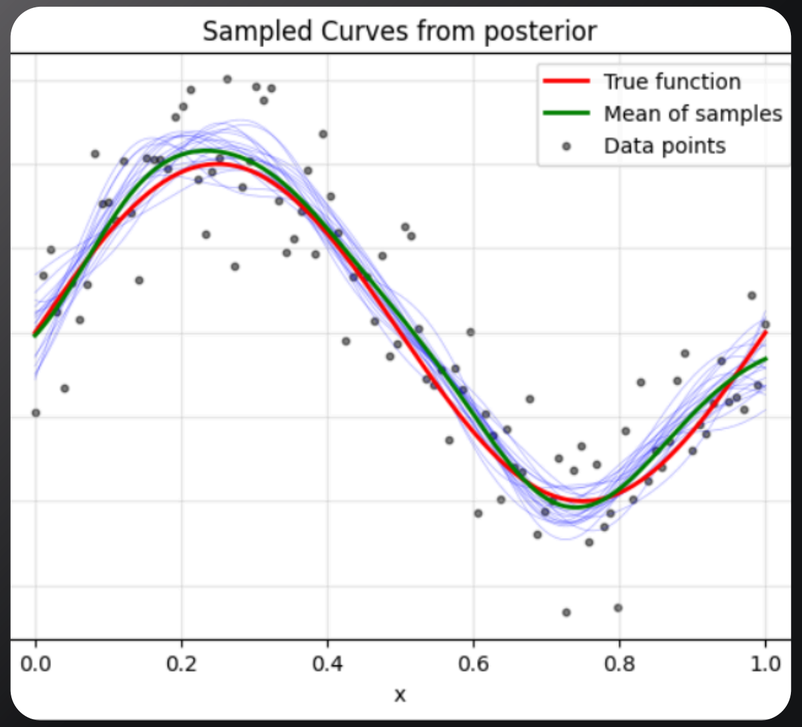
- **Best λ ≈ 2.07** found by minimizing MSE over a wide λ range.
- **Moderate regularization** gives the best generalization.
- **Gaussian basis functions** provided localized flexibility to fit sinusoids.
- Hence Bias–Variance Trade-off is observed clearly using Gaussian kernel regression.



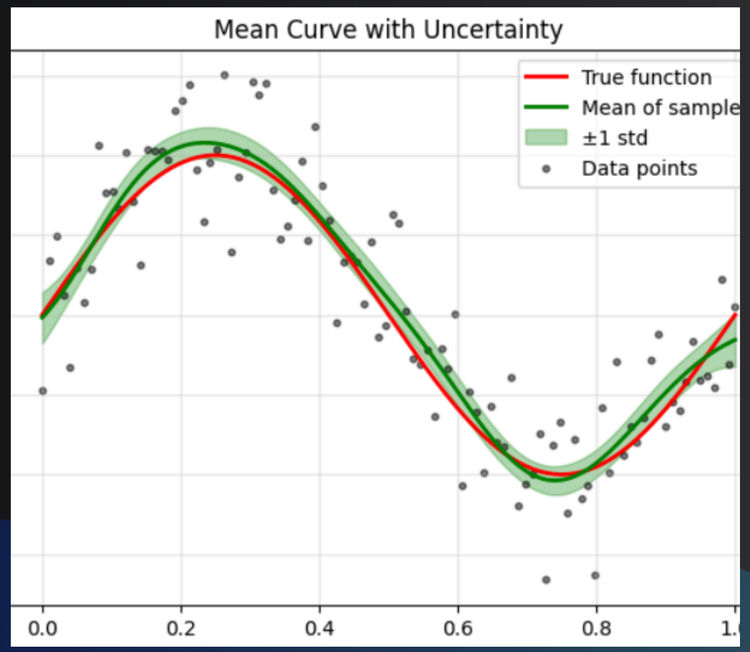
# 6. Exploring Maximum a Posteriori (MAP) Estimation

## Approach Taken

- We adopted a Bayesian approach to model noisy sinusoidal data, rather than using a standard frequentist method like regression.
- The input data was transformed using Gaussian basis functions to form the design matrix.
- A standard normal prior was assumed over the weights, and we performed sequential Bayesian updates - where each new data point updated the posterior, which then became the prior for the next step.
- After processing all data, we sampled multiple weight vectors from the final posterior, generated prediction curves, and took their mean to obtain a smooth, robust fit.



# Modelling Techniques




In Bayesian regression, we treat the weights w as random variables with a probability distribution

- Let each input x be transformed to a feature vector using Gaussian basis functions.
- The output is modeled as

$$t = \mathbf{w}^T \boldsymbol{\phi}(x) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \beta^{-1})$$

- We assume a prior over weights:

p(w) = N (0,1)

- Likelihood for a new point

$$p(t_n \mid \mathbf{w}) = \mathcal{N}(t_n \mid \mathbf{w}^T \phi_n, \beta^{-1})$$

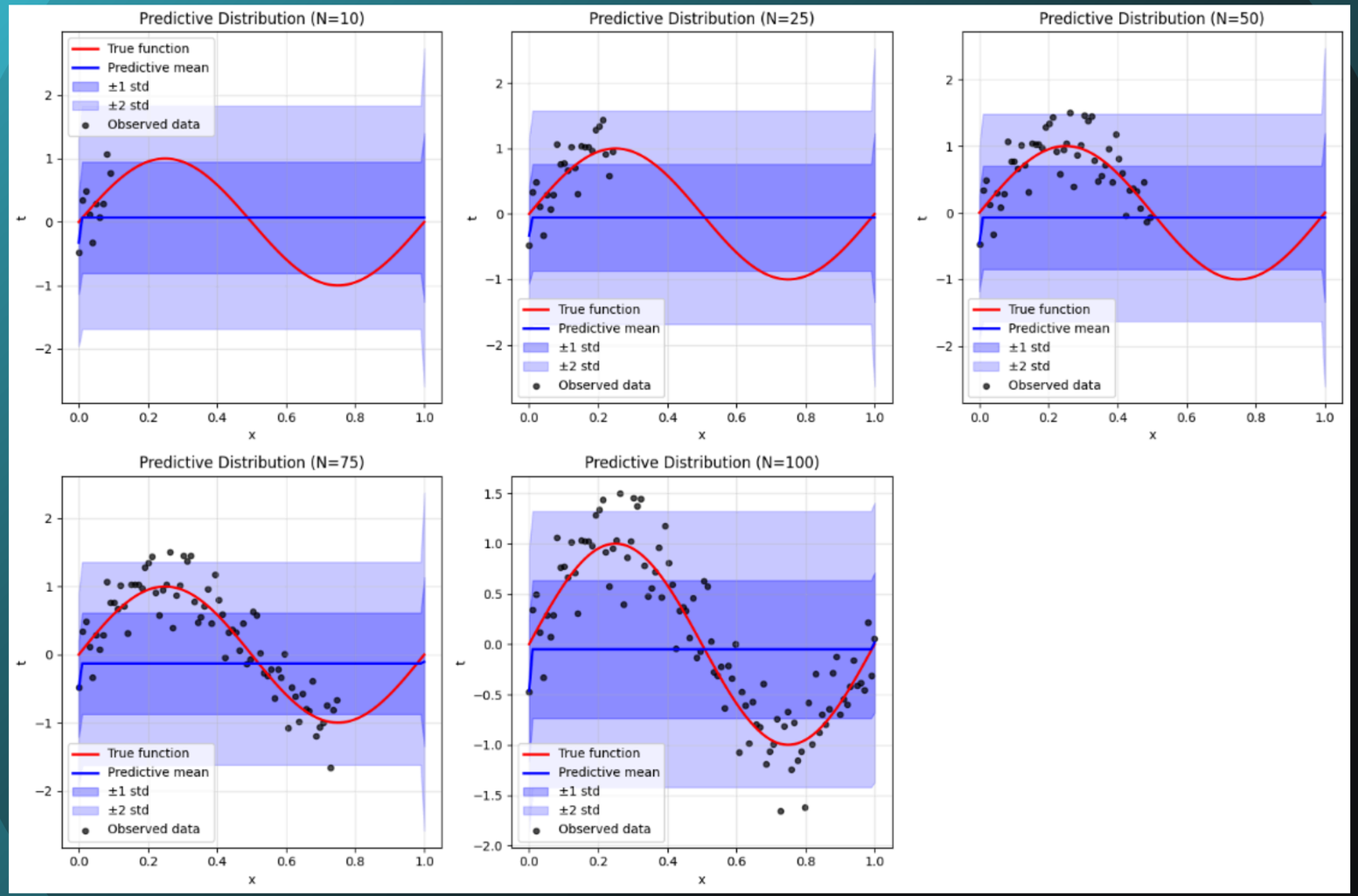
- We compute the posterior after seeing the new data point using Bayes' theorem

$$p(\mathbf{w} \mid t_n) \propto p(t_n \mid \mathbf{w}) \cdot p(\mathbf{w})$$

- As we know, the likelihood and prior are both Gaussian and the product of two Gaussian is also a Gaussian. Hence, the posterior is also Gaussian.

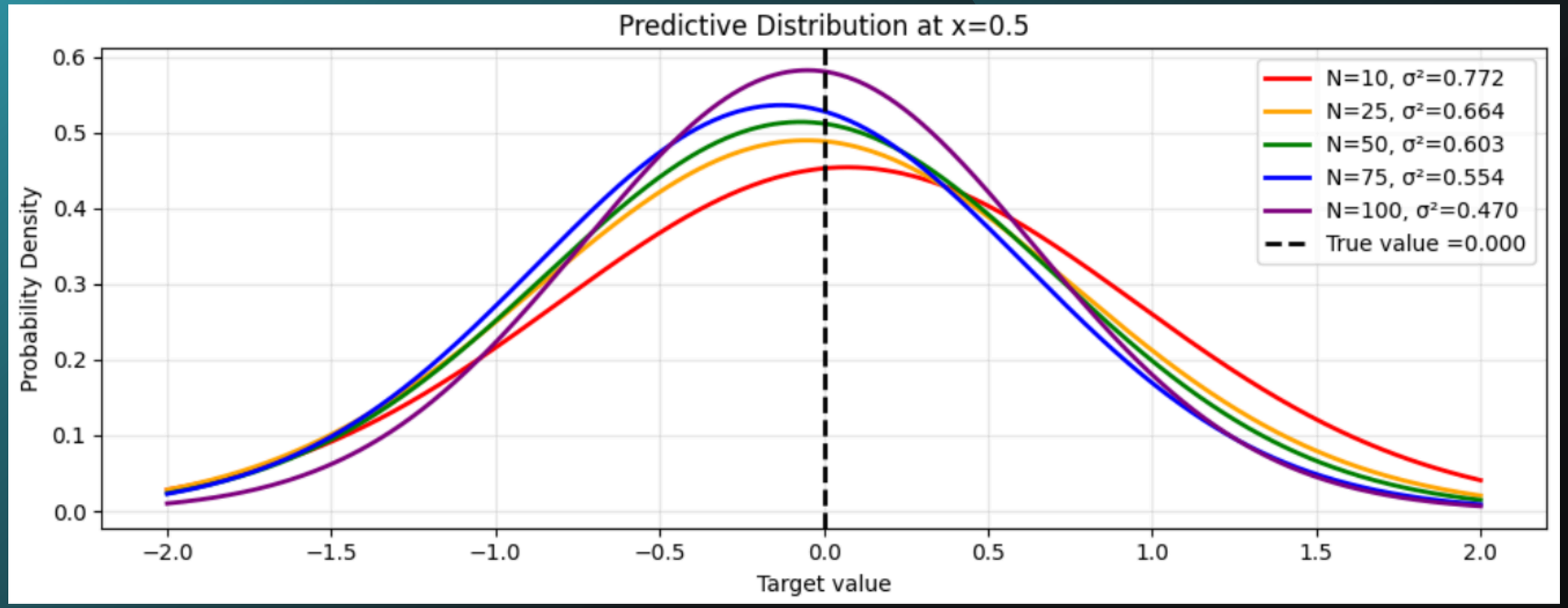
$$S_n^{-1} = S_0^{-1} + \beta \phi_n \phi_n^T \qquad \mathbf{m}_n = S_n \left( S_0^{-1} \mathbf{m}_0 + \beta t_n \phi_n \right)$$

# KEY FINDINGS




The shaded uncertainty bands (±1σ and ±2σ) are wide when only 10–25 points are used but become progressively narrower with 50, 75, and 100 data points — confirming that model confidence increases as more data becomes available.

# KEY FINDINGS




As training size increases, the predictive variance shrinks significantly and the predictive mean aligns more closely to the true function mean.

# Challenges Faced

## Choosing Basis Function width

While the number of Gaussian basis functions was given, tuning the spread/width (σ\sigmaσ) of these functions was challenging. We finally chose 2*range(x)/M

## Interpreting Uncertainty

Bayesian models return distributions over weights. Interpreting predictive uncertainty was initially non-intuitive. Used plots of predictive mean ± standard deviation and posterior sample curves to visualize.

## Understanding the Mathematical Derivations

Deriving and interpreting the Bayesian update equations — especially for sequential MAP updates was initially complex. Plotted graphs to check the accuracy of the approach.

# Conclusion

- The Bayesian MAP framework provides a principled, robust method for regression that naturally incorporates uncertainty and prior knowledge.
- This models learns sequentially from data, avoids overfitting through regularisation via the prior.
- As more data is observed, the MAP estimates of parameters converge, with their mean values stabilizing and uncertainty (standard deviation) reducing — indicating the model is learning more confidently.

# Recommendations

Bayesian regression is ideal when:

- You have small or sequential datasets.
- You need interpretable uncertainty estimates.
- You want to embed prior knowledge into learning.

# THANK YOU