Project1 - SSY230

Sotiris Koutsoftas

April 16, 2024

Task 1)

Part a: Least Squares Estimation

The estimation of the parameters was performed using a least squares approach as implemented in the custom `LinRegress` function. The linear regression model was tested with a design matrix X that incorporates both a constant term and a varying term according to the parity of the index, as suggested in Appendix A. For the scenario where the true parameters $\boldsymbol{\theta}_{\text{true}}$ are given as $\mathbf{0}$, and the outputs y are solely due to the error term e, we obtain the following results:

- For $N = 10$ samples:
  - Estimated parameters: $\hat{\theta} = \begin{bmatrix} 0.348478 & 0.275804 \end{bmatrix}^T$
  - Estimated variance for the first parameter: 0.685794
  - Theoretical variance for the first parameter: 0.100000
- For $N = 100$ samples:
  - Estimated parameters: $\hat{\theta} = \begin{bmatrix} -0.019553 & -0.006943 \end{bmatrix}^T$
  - Estimated variance for the first parameter: 0.019184
  - Theoretical variance for the first parameter: 0.010000
- For $N = 1000$ samples:
  - Estimated parameters: $\hat{\theta} = \begin{bmatrix} -0.020998 & 0.001539 \end{bmatrix}^T$
  - Estimated variance for the first parameter: 0.002004
  - Theoretical variance for the first parameter: 0.001000

The analysis indicates that as the number of samples N increases, the estimated variance of the first parameter converges towards the theoretical variance derived from the given variance of the error term. This is consistent with the expectations from the least squares theory, where the parameter estimates are asymptotically efficient, meaning that they achieve the lower bound for variance with an increasing number of observations.

Part b)

The evalModel function handles linear reggression by multiplying the regressor matrix X by the parameter vector $m.theta$ to produce the predicted outputs. The `evalModel` function was tested using a dataset created to have a linear trend with added noise. The model's performance on the training data indicates how well the model has captured the underlying data generation process. For each sample size, the plots compare the true outputs with the predicted outputs on the training data:

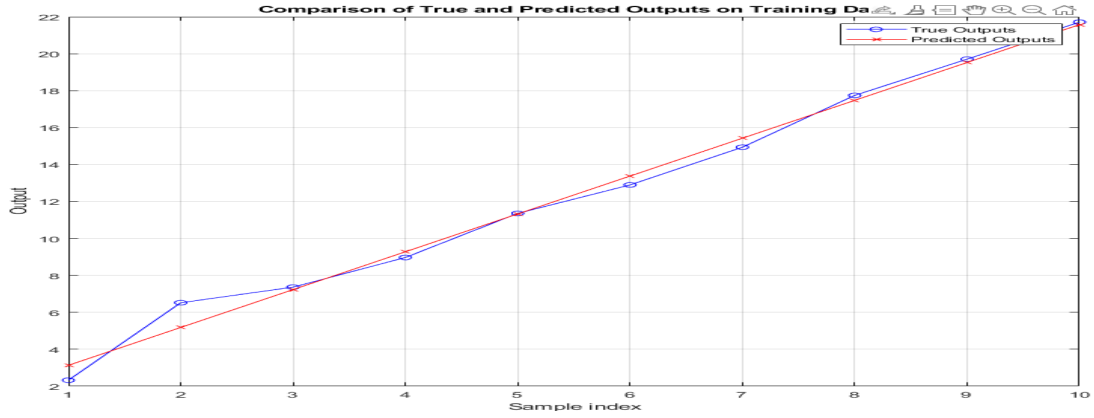

Figure 1: Comparison of true and predicted outputs on training data for $N = 10$.

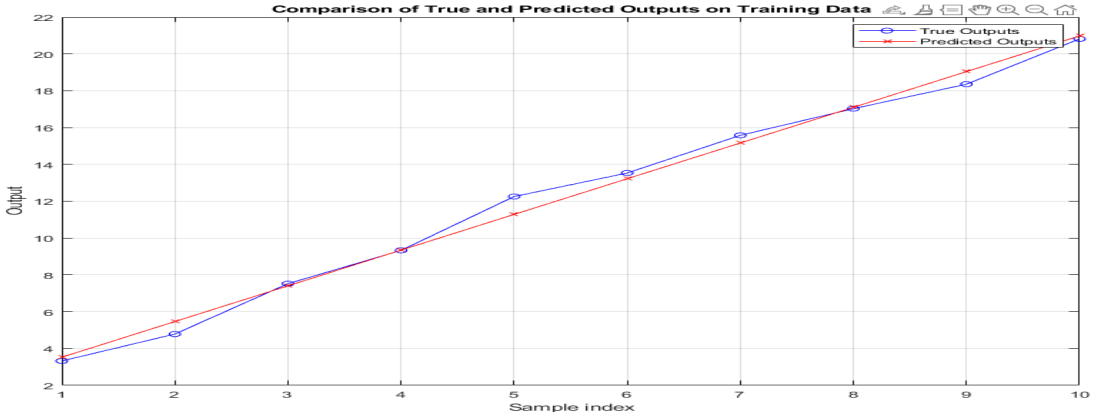

Figure 2: Comparison of true and predicted outputs on training data for $N = 100$.

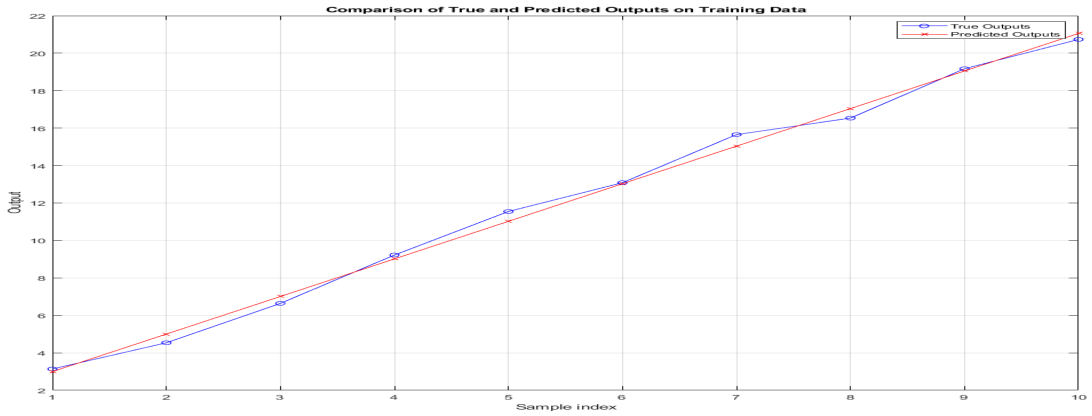

Figure 3: Comparison of true and predicted outputs on training data for $N = 1000$.

Prediction on New Data:

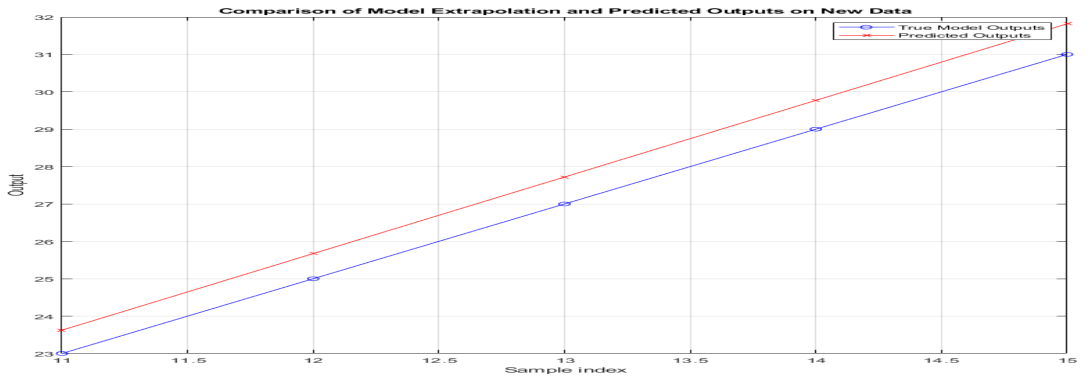

Figure 4: Comparison of model extrapolation and predicted outputs on new data for $N = 10$.

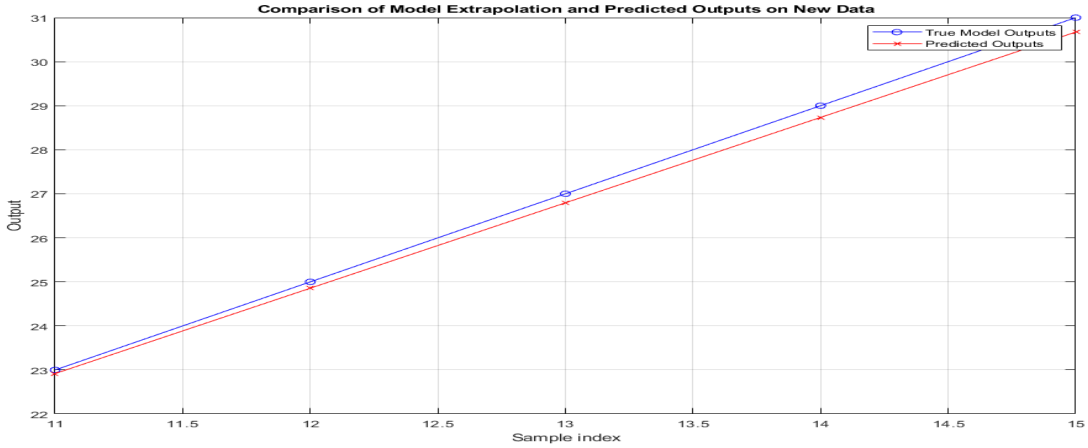

Figure 5: Comparison of model extrapolation and predicted outputs on new data for $N = 100$.

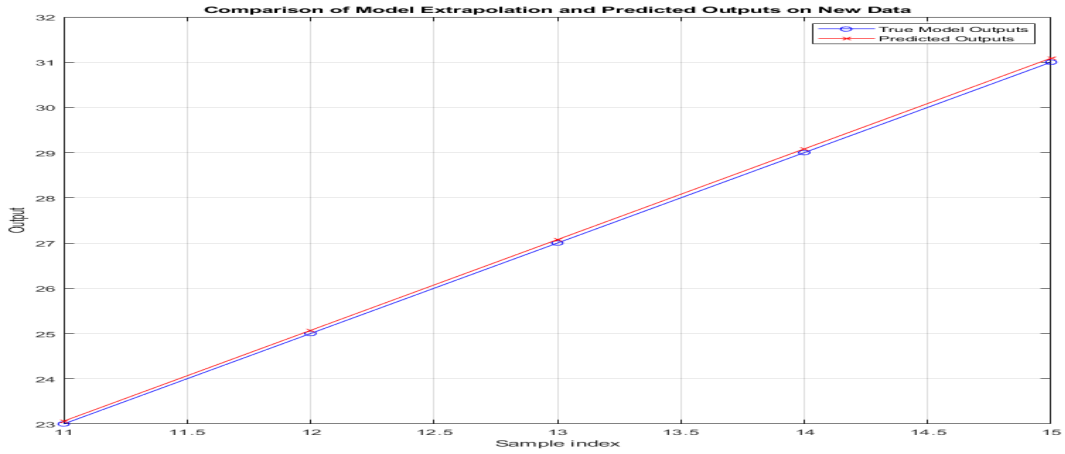

Figure 6: Comparison of model extrapolation and predicted outputs on new data for $N = 1000$.

Part c)

In linear regression, our goal is to find the parameter vector θ that minimizes the cost function, which is the sum of squared residuals. Without regularization, the cost function $J(\theta)$ is defined as:

$$J(\theta) = \sum_{k=1}^{N} \left(y^{(k)} - \theta^T x^{(k)} \right)^2 \tag{1}$$

To incorporate regularization, we add a penalty term to the cost function, leading to the regularized cost function $J_{\text{reg}}(\theta)$:

$$J_{\text{reg}}(\theta) = \sum_{k=1}^{N} \left(y^{(k)} - \theta^T x^{(k)} \right)^2 + \lambda \theta^T \theta \tag{2}$$

Here, λ is the regularization parameter that controls the amount of shrinkage applied to the parameters θ. The derivative of $J_{\text{reg}}(\theta)$ with respect to θ is computed and set to zero to find the minimum, yielding the normal equations with the regularization term:

$$-2X^T(y - X\theta) + 2\lambda\theta = 0 \tag{3}$$

Solving for θ:

$$(X^T X + \lambda I)\theta = X^T y \tag{4}$$

This leads us to the solution for θ that minimizes the regularized cost function. In the MATLAB implementation, we extend the design matrix X and the output vector y with additional rows to incorporate regularization:

$$x2 = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix}, \quad y2 = \begin{bmatrix} y \\ 0 \end{bmatrix} \tag{5}$$

By solving the least squares estimate using $x2$ and $y2$, we are effectively solving the following optimization problem:

$$\min_{\theta} \|x2 \cdot \theta - y2\|^2 = \min_{\theta} \|X\theta - y\|^2 + \lambda \|\theta\|^2 \tag{6}$$

This corresponds to the regularized cost function $J_{\text{reg}}(\theta)$, and thus by augmenting X and y, we use the same least squares approach to solve the regularized problem. As λ increases, it is observed that the estimates of parameters tend to shrink towards zero. This effect is captured in the plot below, where the parameter estimates are plotted against the logarithm of the regularization parameter λ. It is evident that parameter estimates decrease in magnitude as λ increases, demonstrating the trade-off between fitting the training data and penalizing the complexity of the model.

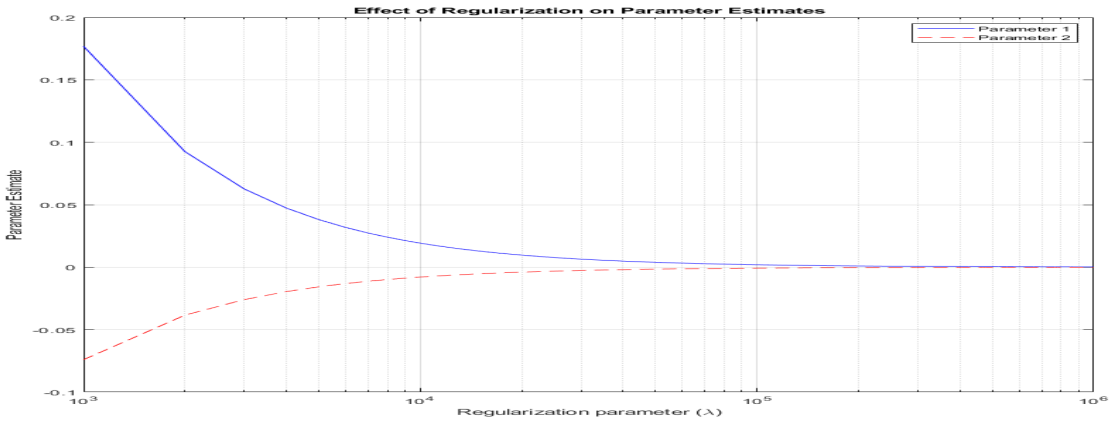

Figure 7: Effect of regularization on the estimates of parameters. Parameter 1 is shown in blue solid line, while Parameter 2 is represented by the red dashed line.

Part d)

Generating Polynomial Features:

To transform our original predictors into a form suitable for polynomial fitting, we generate a new regressor matrix x2. For a polynomial of degree $n = 3$ and a set of regressors $x = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$, the transformation is given by:

$$x2 = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 \cdots x_1^n \cdots x_1x_2^{n-1} & x_2^n \end{bmatrix}$$

$$x2 = \begin{bmatrix} 1 & 2 & 3 & 4 & 6 & 9 & 8 & 12 & 18 & 27 \\ 1 & 4 & 5 & 16 & 20 & 25 & 64 & 80 & 100 & 125 \end{bmatrix}$$

The MATLAB function $poly_{x2}$ constructs this matrix by iteratively calculating all possible combinations of x1 and x2 up to the n-th degree, including interaction terms (as can be seen above) and then using the generated features we apply the `LinRegressRegul` function to estimate the polynomial model parameters. To validate the polynomial fitting, we compare the model's predictions with data generated from known quadratic and cubic functions, confirming the model's ability to capture non-linear relationships.

Part e) Model Visualization and Uncertainty Quantification

In this part of our analysis, we evaluate the performance of three distinct models: linear regression (LR), polynomial regression (POLY), and KNN on a dataset generated from a sine function with additive noise. The models are tested with varying sample sizes to assess their generalization and prediction confidence across different data densities.

Linear Regression Model:

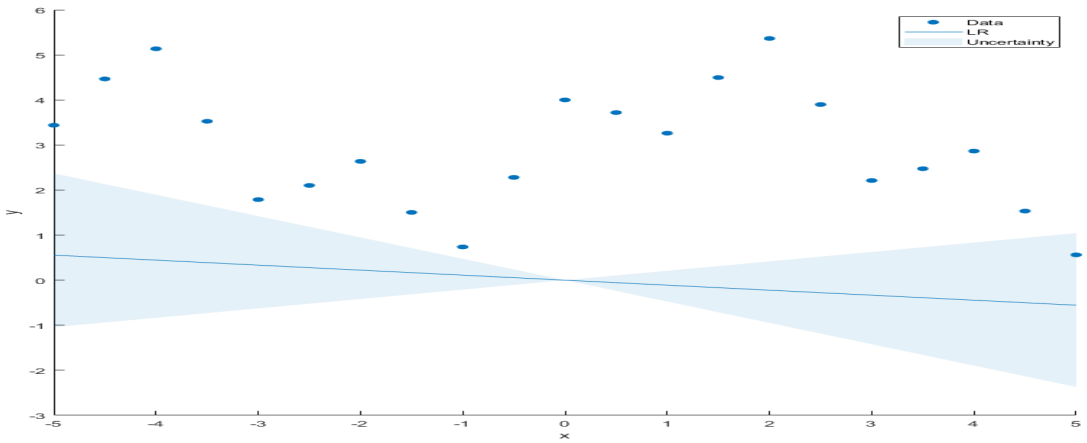

Figure 8: LR model with $n = 5$

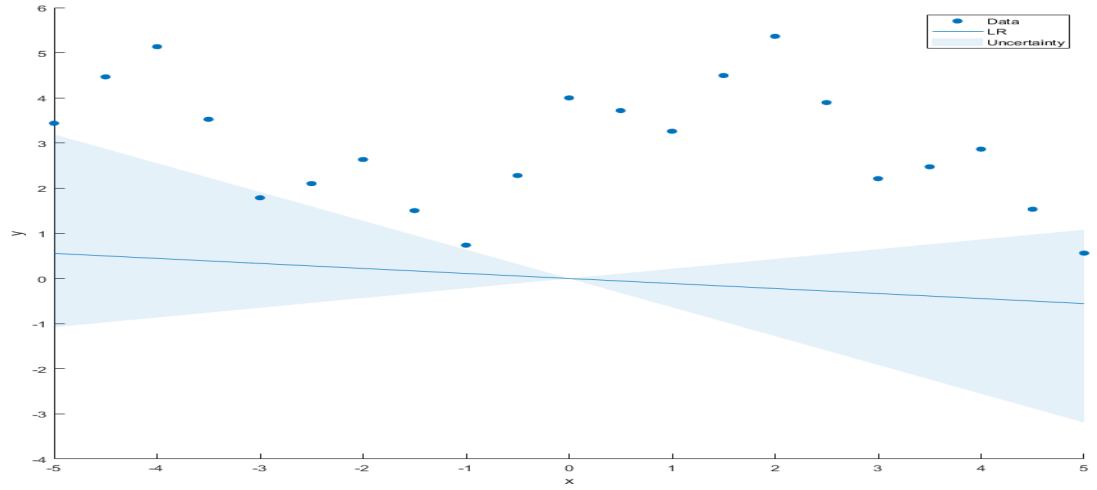

Figure 9: LR model with $n = 10$

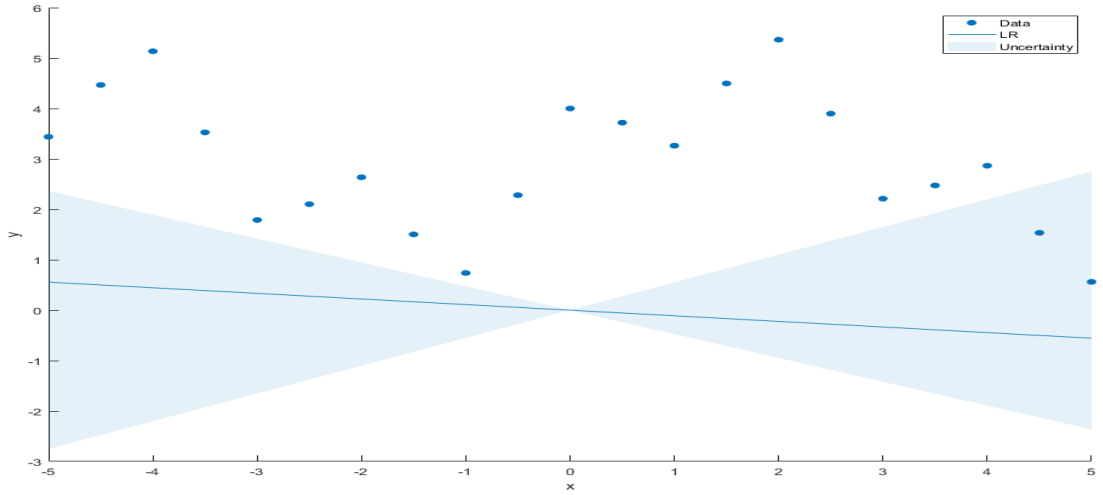

Figure 10: LR model with $n = 20$

Polynomial Regression Model:

The POLY model, fitted with a higher degree polynomial, is observed to encapsulate the non-linear trend of the data more effectively. The shaded uncertainty region indicates confidence in the model's predictions, which narrows near the center of the data distribution where data points are denser.

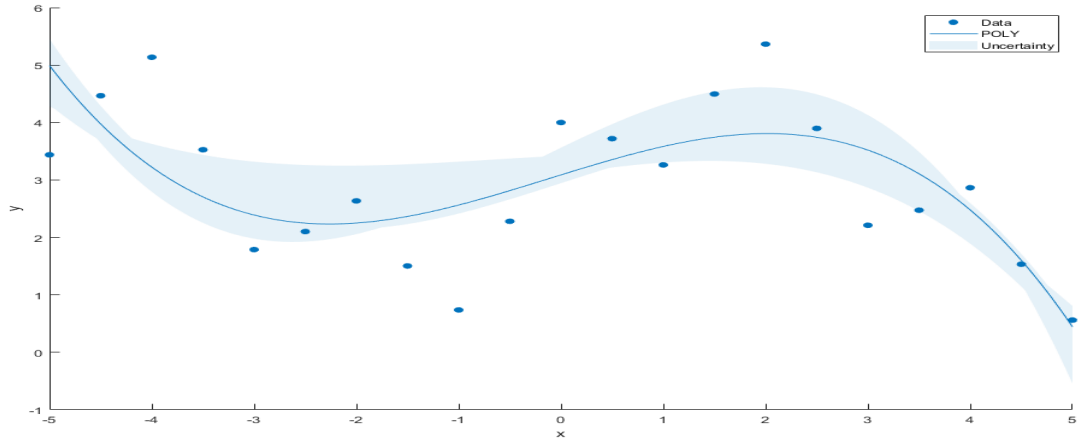

Figure 11: POLY model with $n = 5$ capturing non-linear trends with noticeable uncertainty boundaries.

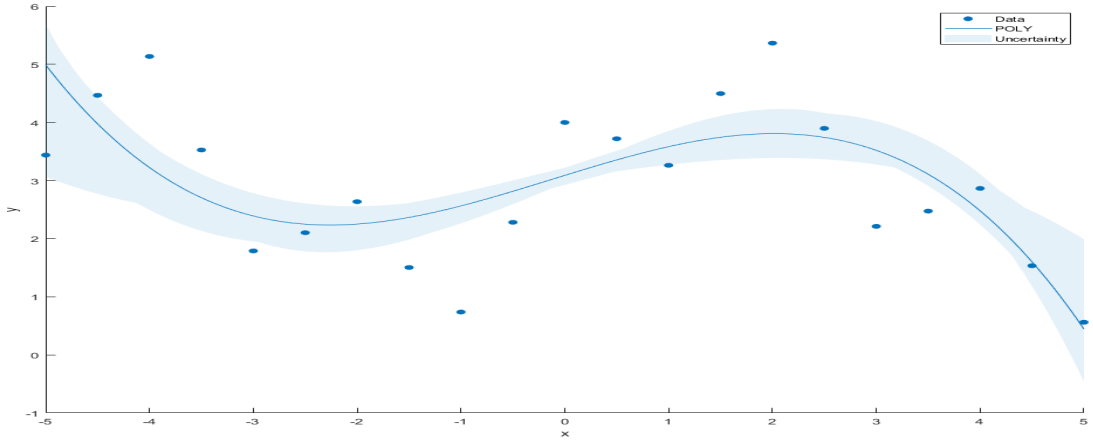

Figure 12: POLY model with $n = 10$ capturing non-linear trends with noticeable uncertainty boundaries.

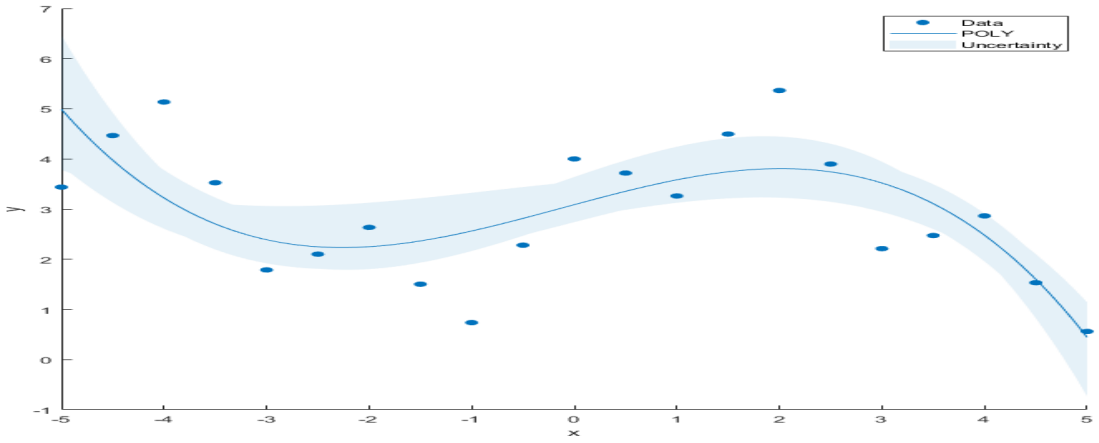

Figure 13: POLY model with $n = 20$ capturing non-linear trends with noticeable uncertainty boundaries.

KNN Regression Model plotted along with LR and POLY:

The KNN model adapts to the local structure of the data. As a non-parametric method, it flexibly approximates the function that generated the data, leading to a staircase-like prediction pattern. This behavior shows the KNN model's sensitivity to local data variations.

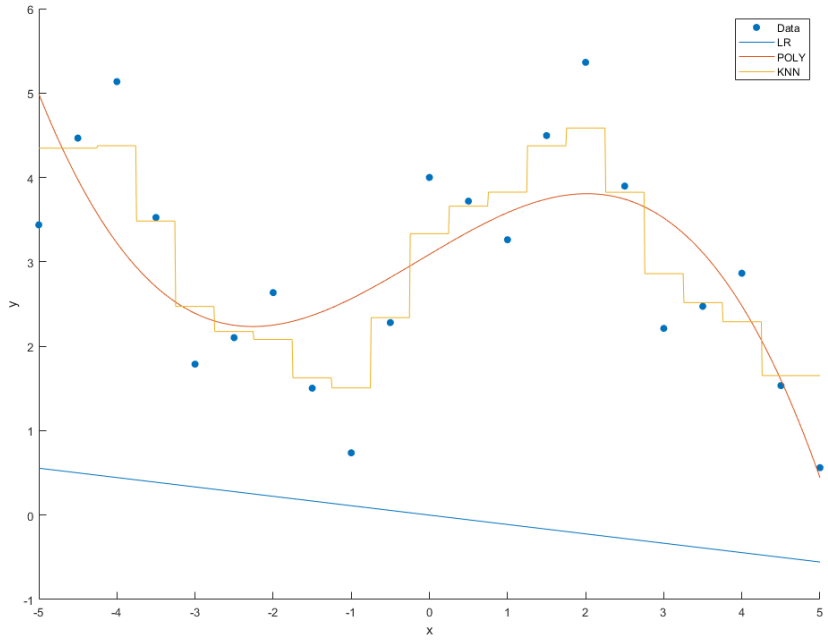

Figure 14: Comparative visualization of LR, POLY, and KNN models, showing the adaptive nature of KNN and the smoother trend captured by POLY.

Comparative Analysis Across Sample Sizes:

The LR model shows a consistent pattern of prediction uncertainty, while the POLY model displays varying degrees of confidence intervals, which narrow with increased sample sizes. The KNN model remains closely tied to the data points, reflecting high flexibility. While the LR model offers simplicity and interpretability, the POLY model provides a better fit to non-linear patterns. The KNN model captures intricate data trends but may require careful tuning to prevent overfitting, especially with small datasets. The plotted uncertainty bounds offer insights into the confidence of model predictions.

Task 2)

The KNN model is realized through the function `knnRegress` that stores the given input and output data along with the number of nearest neighbors specified by k. This func-

tion initializes the KNN model structure, which is used subsequently by the `evalModel` and `plotModel` functions for prediction and visualization, respectively. The `evalModel` function is modified to evaluate the KNN model by finding the k closest points in the input space for a given regressor and computing the mean of their corresponding outputs. This mean value is used as the predicted output. The `plotModel` function is extended to visualize the KNN regression results. The plots show how the model captures the pattern of the data with different values of k. A synthetic dataset was generated with sinusoidal variations and random noise, which it was used to test the KNN regression model for $k = 3$ and $k = 5$. Two different KNN models were trained and their predictions were visualized. The first plot (Figure 15) shows the model performance for $k = 3$, the second plot (Figure 16) for $k = 5$ and the third one shows the model for $k = 5$ plotted along with the corresponding model obtained using LR:

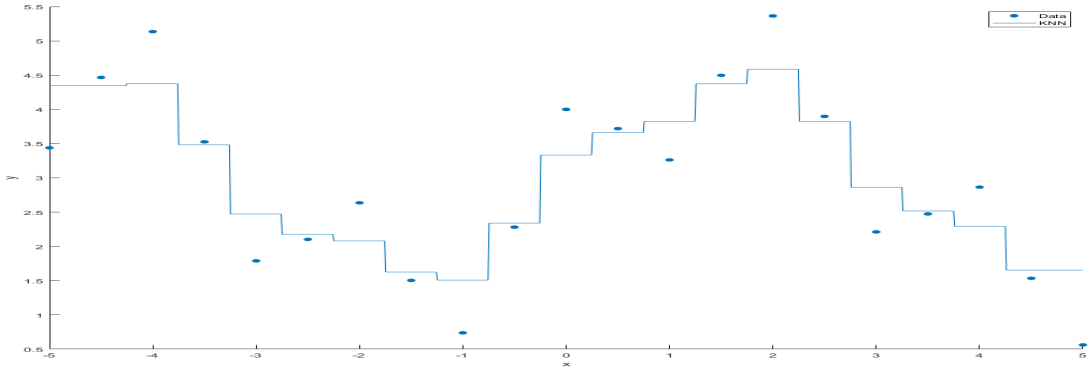

Figure 15: KNN regression with $k = 3$

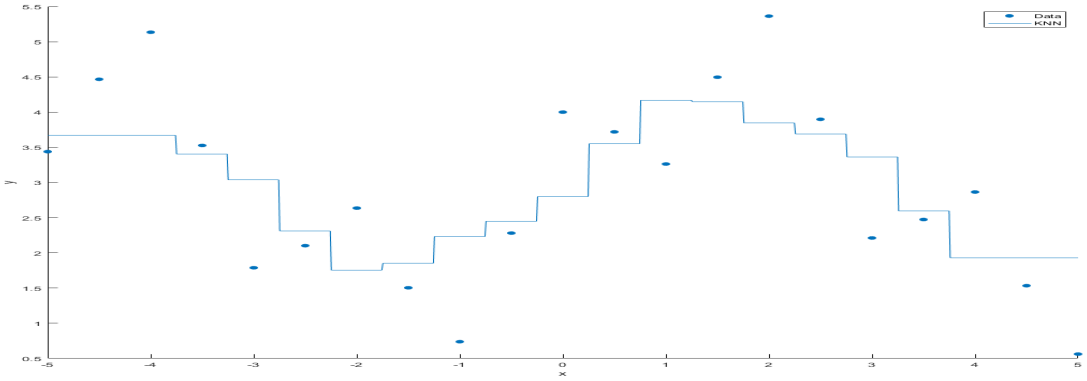

Figure 16: KNN regression with $k = 5$

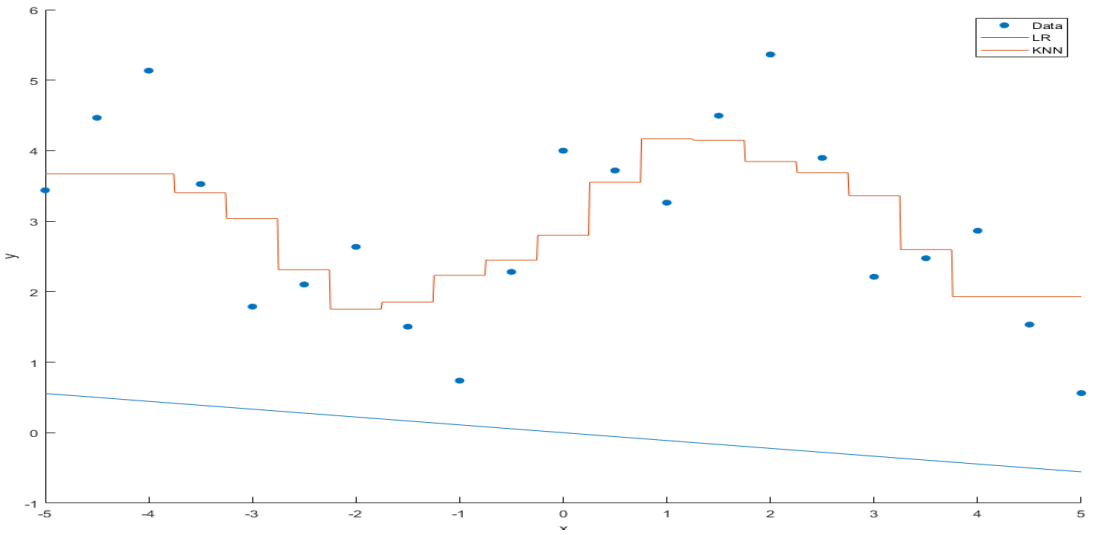

Figure 17: KNN regression with $k = 5$ with the corresponding LR model

The plots indicate that the choice of k significantly influences the model's predictions. With $k = 3$, the model appears to follow the data more closely, suggesting a more flexible approach, whereas $k = 5$ provides a smoother approximation, indicating a trade-off between model complexity and generalization. Both models obtained using KNN regression appear way better compared to the LR model, as can be seen in Figure 17.

Task 3)

3.1.a)

We assess the estimation quality of an LR model by varying the number of data points available. Specifically, the model's MSE and the convergence of estimated parameters toward the true function parameters are evaluated. Three datasets with sample sizes of 10, 100, and 1000 points are generated using the `linearData` function, each with a noise variance of 1. Linear regression models are estimated for each dataset using the `LinRegress` function, and the MSE is computed. The estimated parameters and MSE values for each dataset size are as follows:

$$\text{Theta Estimates (10 data points):} \quad \begin{bmatrix} 0.7290 & 0.7290 \end{bmatrix}$$

$$\text{MSE Values (10 data points):} \quad \begin{bmatrix} 2.6822 \end{bmatrix}$$

Figure 18 illustrates the model fit for the smallest dataset (N=10), showing the regression line and uncertainty bounds. The shaded area represents the 95% confidence interval, signifying substantial estimation uncertainty due to the limited data points.

$$\text{Theta Estimates (100 data points):} \quad \begin{bmatrix} 0.7468 & 0.7468 \end{bmatrix}$$

$$\text{MSE Values (100 data points):} \quad \begin{bmatrix} 1.2167 \end{bmatrix}$$

Increasing the sample size to 100 enhances the model's estimation performance, as evidenced by the reduced MSE. Figure 19 presents a more accurate model fit to the data, with narrower confidence bounds relative to the 10 data points model.

$$\text{Theta Estimates (1000 data points):} \quad \begin{bmatrix} 0.7261 & 0.7261 \end{bmatrix}$$

$$\text{MSE Values (1000 data points):} \quad \begin{bmatrix} 1.4309 \end{bmatrix}$$

With a dataset of 1000 points, the MSE slightly increases compared to 100 samples.

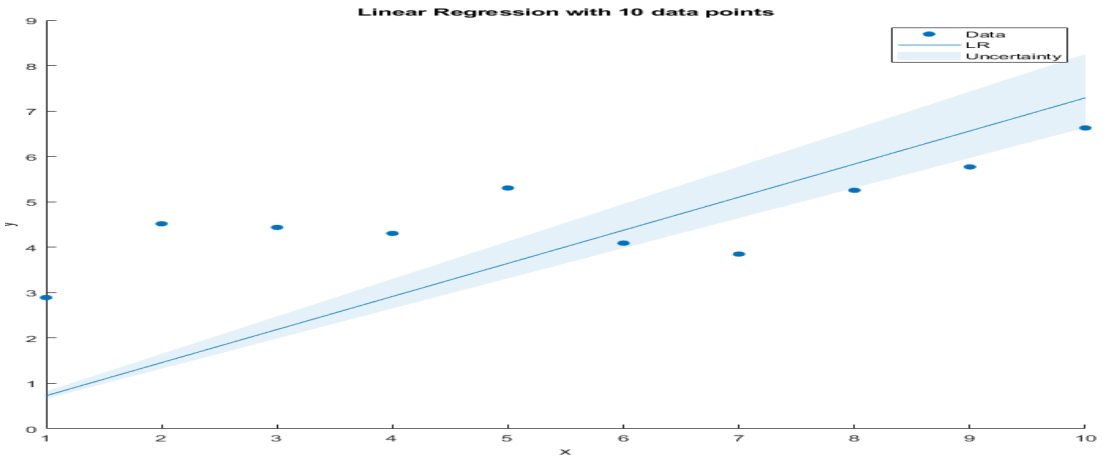

Figure 18: Linear Regression with 10 data points.

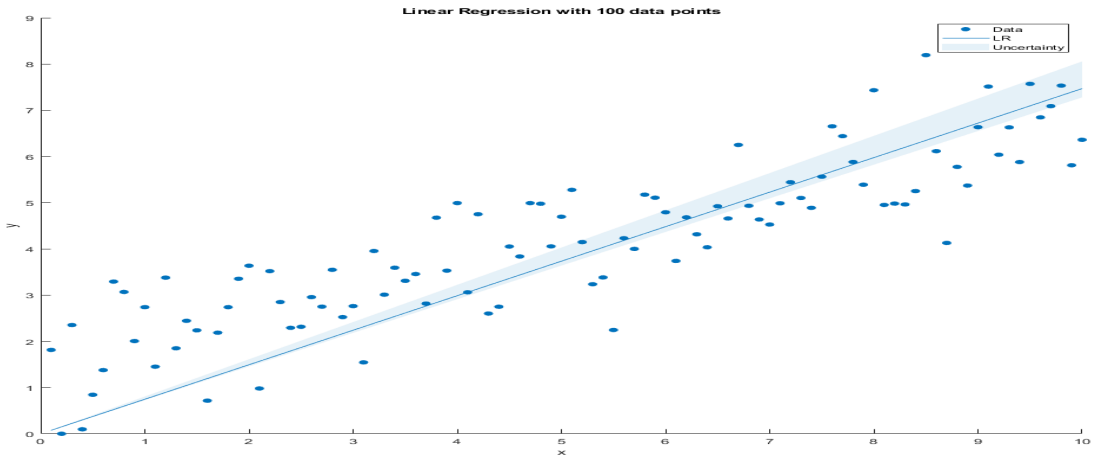

Figure 19: Linear Regression with 100 data points.

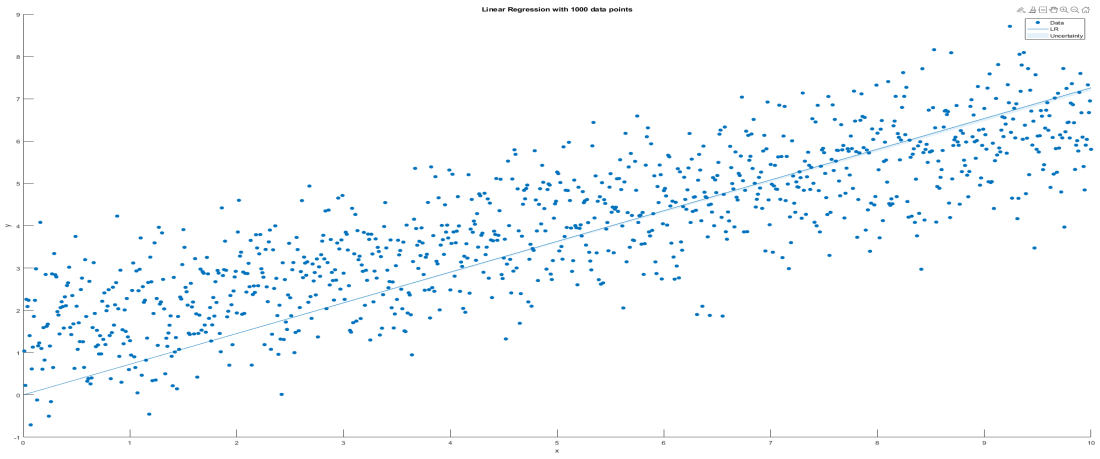

Figure 20: Linear Regression with 1000 data points.

3.1.b)

Here we extend the LR approach to a fifth-order polynomial regression model. This model is expected to capture more complex relationships in the data compared to the simple LR. For each sample size N in $\{10, 100, 1000\}$, a dataset (X, y) was generated using the function `linearData` with a noise variance of 1. For each generated dataset, the fifth-order polynomial model was estimated without regularization ($\lambda = 0$). The polynomial features were constructed by raising the input data X to the powers up to the fifth degree. The MSE values for the different sample sizes are as follows:

- $N = 10$: MSE = 0.2999
- $N = 100$: MSE = 1.0243
- $N = 1000$: MSE = 0.9991

The model and data points for each sample size were plotted to visualize the fit of the polynomial regression model, which shows that as the sample size increases the model's prediction encompasses the uncertainty due to noise, as indicated by the shaded confidence interval around the regression line. The MSE increases with the sample size and as more data is introduced, the model's predictions become more accurate.

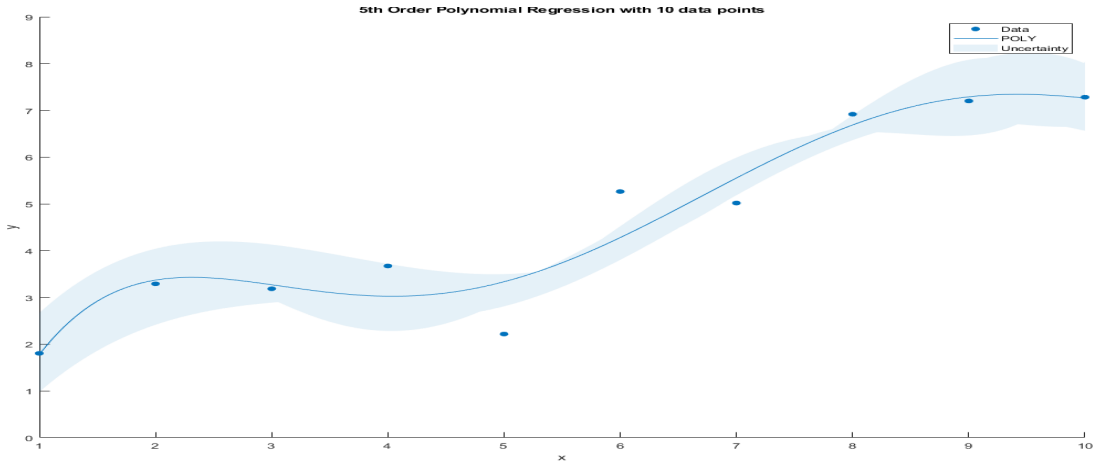

Figure 21: 5th Order Polynomial Regression with 10 data points.

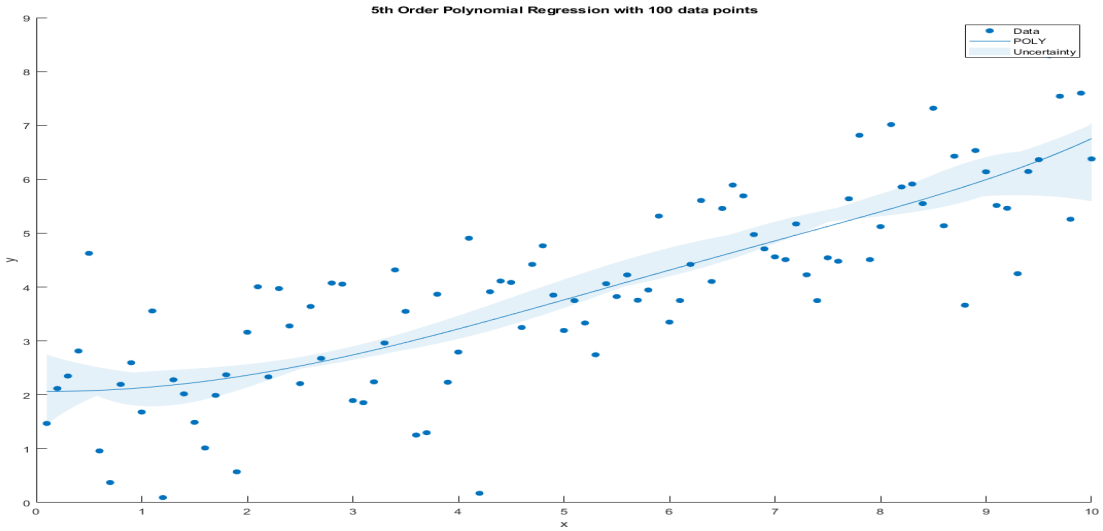

Figure 22: 5th Order Polynomial Regression with 100 data points.

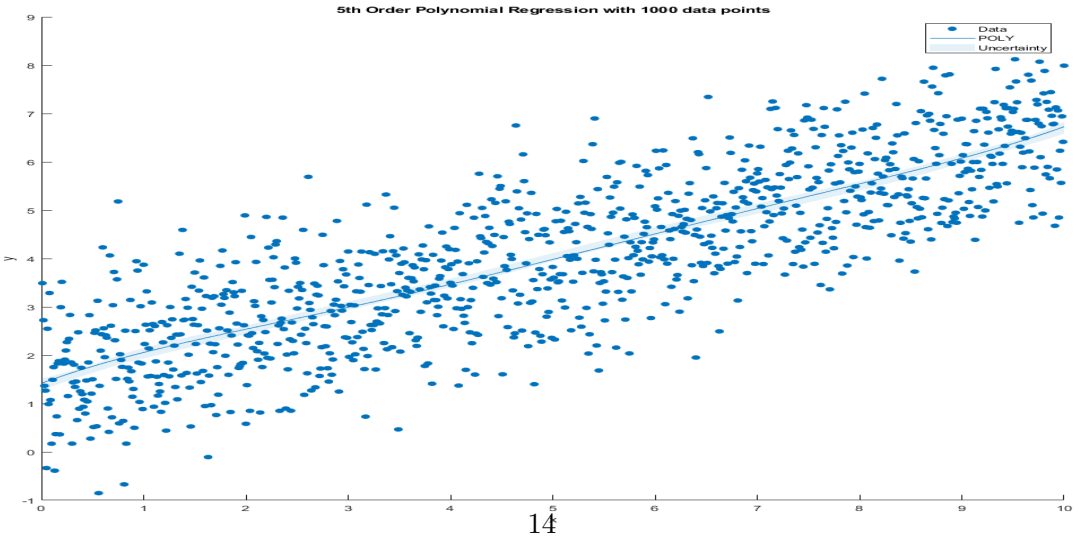

Figure 23: 5th Order Polynomial Regression with 1000 data points.

The plots show that with polynomial regressios we are able to capture the nature of our data with higher accuracy compared to LR.

3.1.c)

In this part of the exercise, we aim to validate the variance of parameter estimates obtained through linear regression by conducting a Monte Carlo simulation. A total of $N = 10$ data points were generated for each dataset, with a set number of simulations, $num\_simulations = 100$, and the noise variance fixed at $noise\_variance = 1$. The true parameter vector was defined as $true\_parameters = \begin{bmatrix} 2 \\ -3 \end{bmatrix}$, representing the intercept and slope respectively.

Monte Carlo Simulation:

Each simulation involved generating a noisy dataset with an underlying true linear relationship. The LR model was estimated using the `LinRegress` function, which yielded a set of estimated parameters for each iteration. This process was repeated for $num\_simulations$ iterations. The variance of the parameter estimates was then computed empirically from the collected samples.

Empirical Variance Calculation:

The empirical variance of the intercept and slope estimates was calculated for each of the simulation sets. The calculated values are as follows:

- For 10 samples:
  - Empirical Variance of Intercept Estimates: 0.46969
  - Empirical Variance of Slope Estimates: 0.012046
- For 100 samples:
  - Empirical Variance of Intercept Estimates: 0.037359
  - Empirical Variance of Slope Estimates: 1.0535×10^{-5}
- For 1000 samples:
  - Empirical Variance of Intercept Estimates: 0.0042868
  - Empirical Variance of Slope Estimates: 1.3818×10^{-8}

The histograms for the estimated intercept and slope across the simulations were plotted to visually represent the distribution of the estimates.

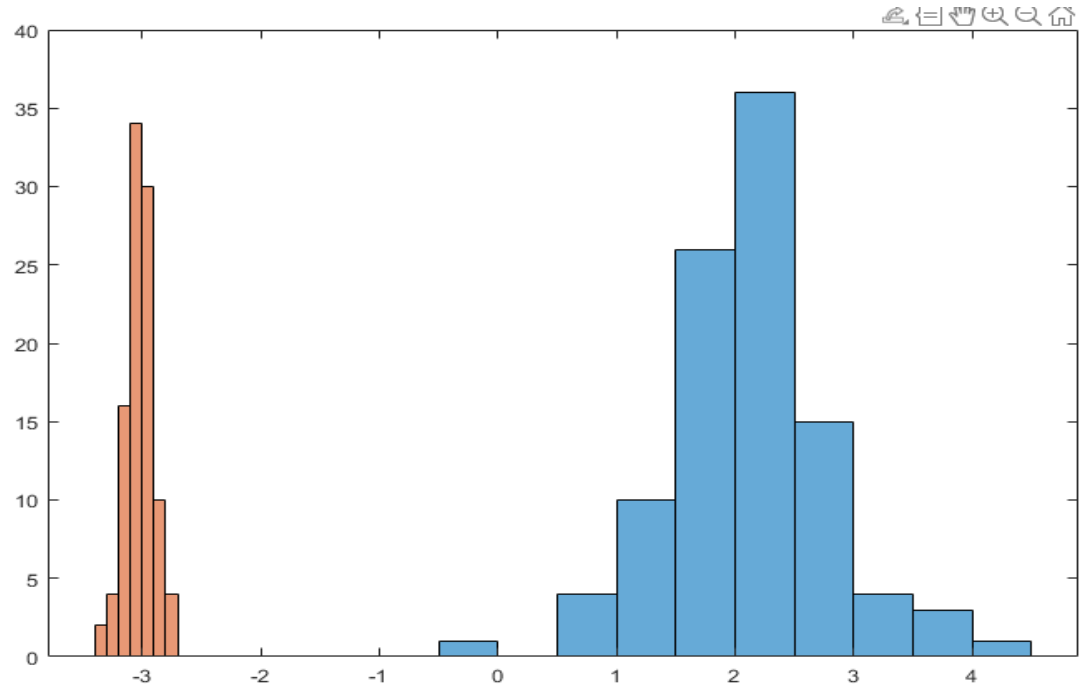

Figure 24: Histogram of estimated intercept and slope from 10 *samples.*

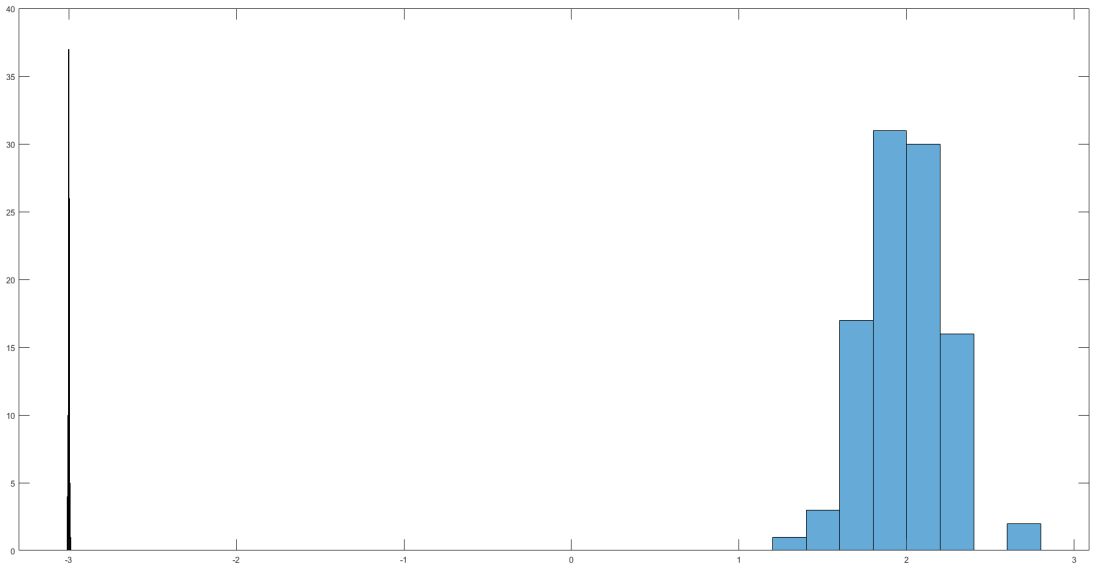

Figure 25: Histogram of estimated intercept and slope from 100 *samples.*

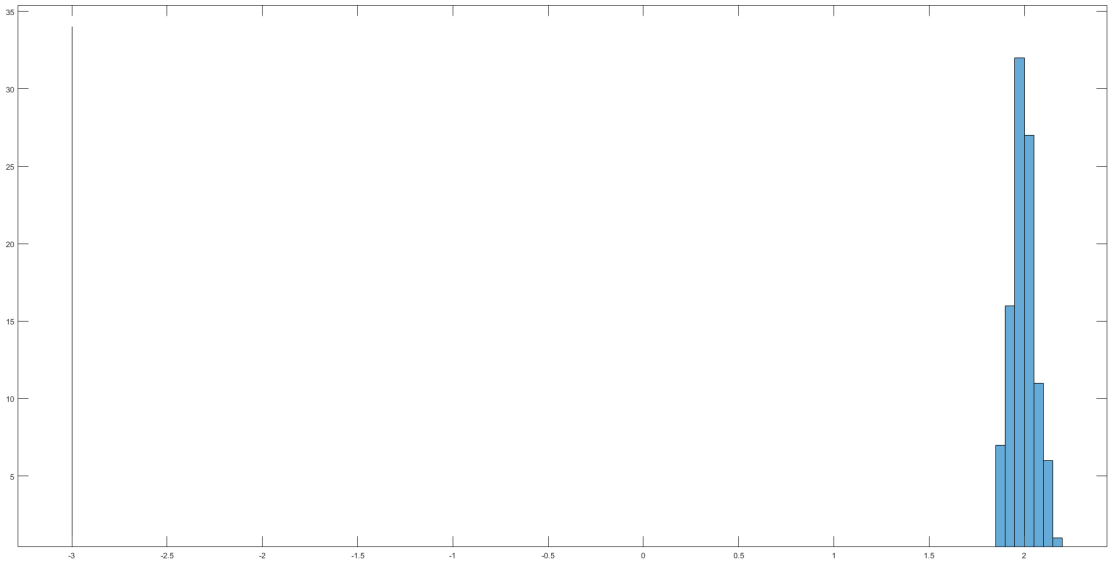

Figure 26: Histogram of estimated intercept and slope from 1000 *samples.*

As expected, the empirical variance of the parameter estimates decreases with an increase in the sample size. This observation aligns with the theoretical expectation that estimates should become more precise as the amount of data increases. The histograms further illustrate this convergence, showcasing a tighter distribution of estimates around the true parameter values for higher sample sizes.

3.1.d)

Here we evaluate the performance of the KNN regression model under varying conditions of sample size and noise level. For each combination of sample size and noise level, a dataset is generated using the `linearData` function. A series of KNN models with k ranging from 1 to 5 are trained on a portion of the dataset, with the remaining data used for validation. The process is repeated for a number of simulations to obtain an average MSE for each k. The results are organized in a series of plots, each representing a different combination of sample size and noise level. The MSE is plotted against the number of neighbors (k) to visualize the model's performance trend.

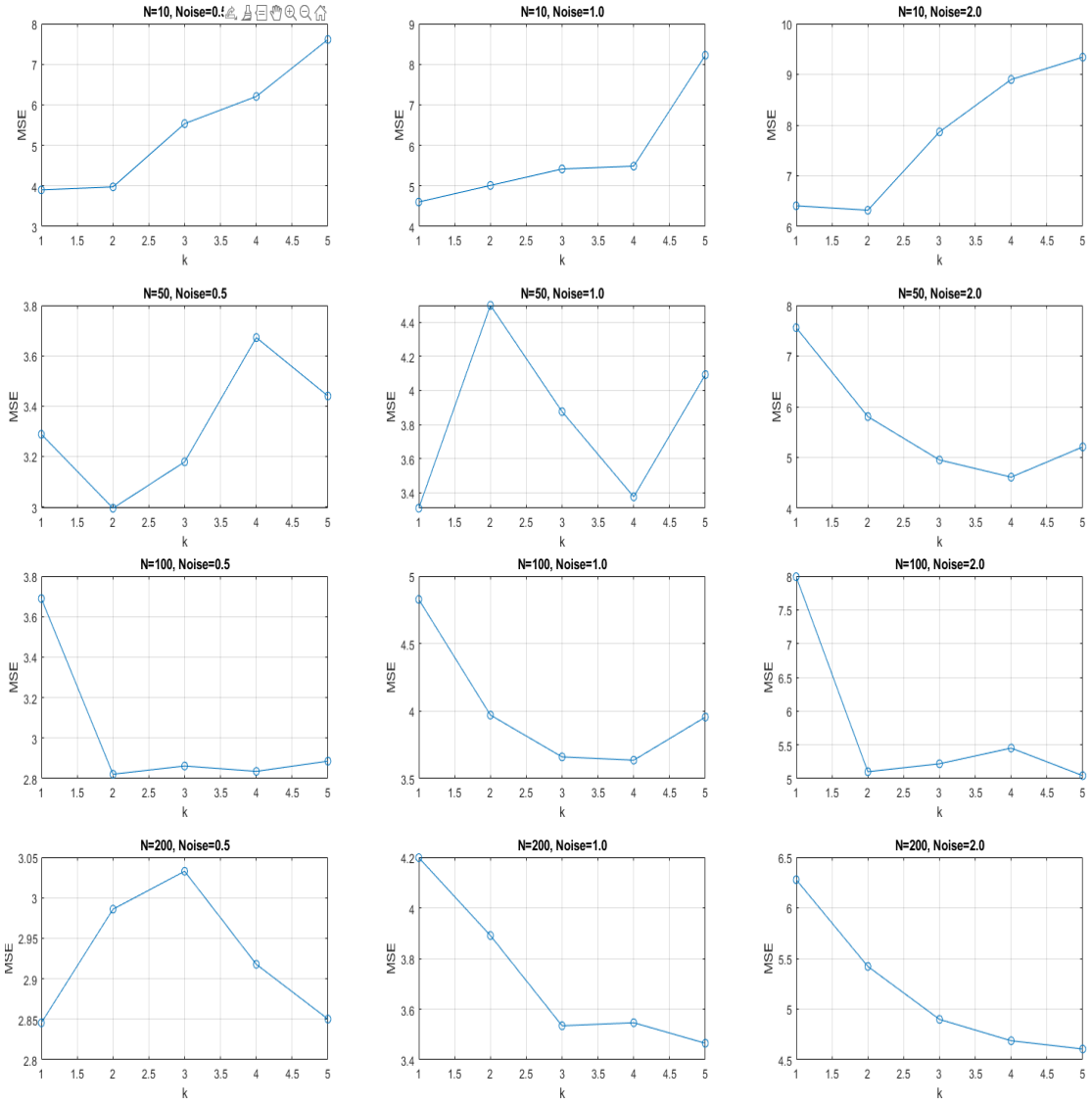

Figure 27: KNN regression MSE for varying k with different sample sizes (10, 50, 100, 200) and noise levels.

As can be observed from Figure 27, the MSE tends to vary with the number of neighbors and exhibits different behaviors depending on the sample size and noise level. Notably, with a lower noise level and a higher number of neighbors, the model tends to perform better, reflecting a lower MSE. As the noise level increases, the MSE generally tends to increase, indicating a decrease in model accuracy. Similarly, the model's performance improves with an increase in sample size, demonstrating the benefits of more data in the learning process.

3.2.a)

Here, we explore the impact of sample size on the accuracy of parameter estimation and the MSE in linear regression. For this, datasets with 10, 100, and 1000 points were generated, each with noise variance set to 1.

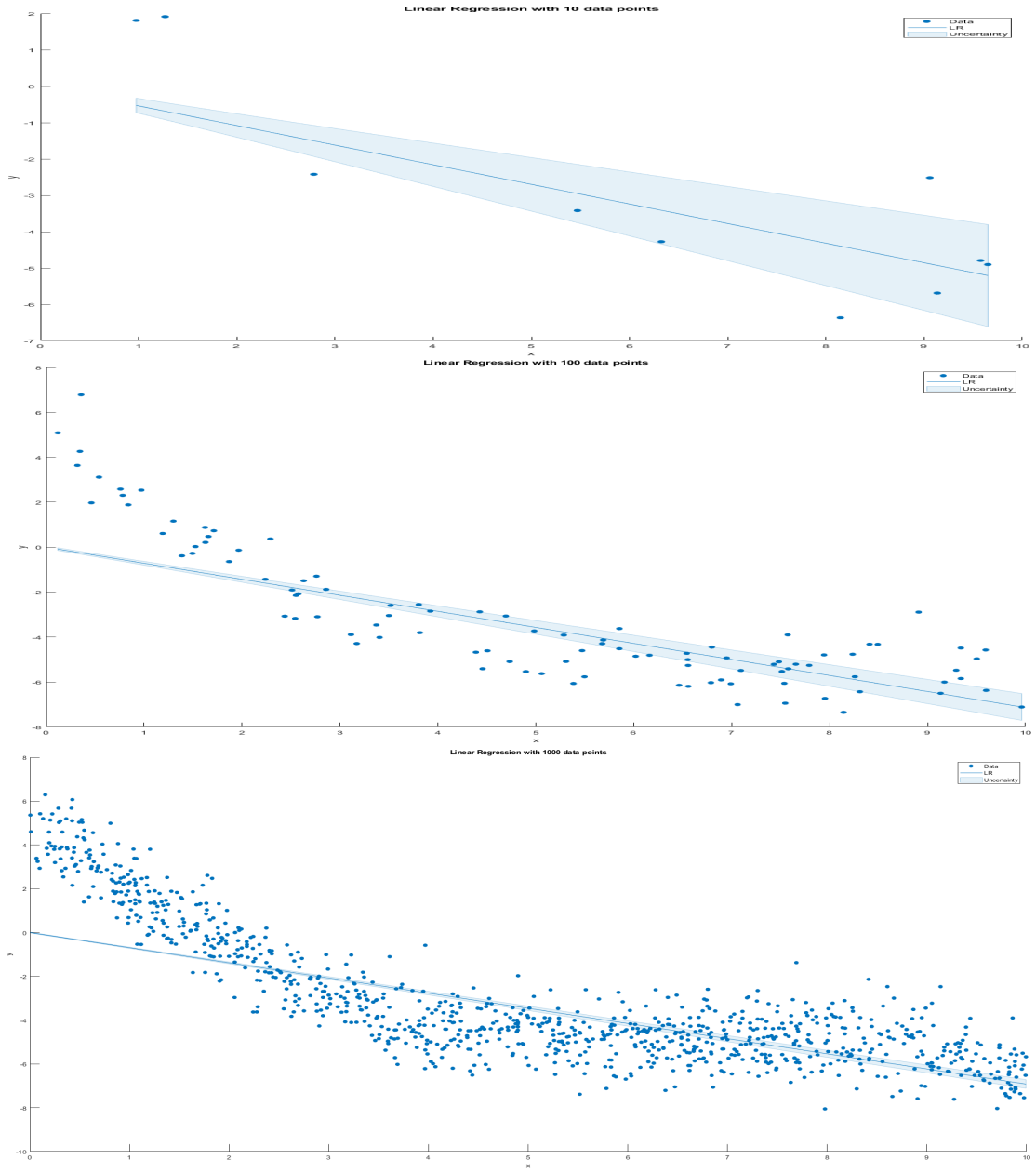

Figure 28: Linear regression on datasets with 10, 100, and 1000 data points. The regression line and the uncertainty in the estimates are shown.

The parameter estimates obtained are:

$$\text{For 10 data points: } \theta = [-0.5396, -0.5396]$$
$$\text{For 100 data points: } \theta = [-0.7133, -0.7133]$$
$$\text{For 1000 data points: } \theta = [-0.6922, -0.6922]$$

And the corresponding MSE values are:

$$\text{For 10 data points: MSE} = 2.4365$$
$$\text{For 100 data points: MSE} = 2.9843$$
$$\text{For 1000 data points: MSE} = 3.2987$$

We observe that as the number of data points increases, the MSE also increases. This could be due to the complexity of the data generation process.

3.2.b)

This experiment examines the effect of polynomial degree on model accuracy with various regularization strengths. We apply linear regression with polynomial terms up to the tenth degree to a dataset of 15 points, introducing noise variance of 1. Five different regularization parameters (λ) are used to mitigate overfitting.

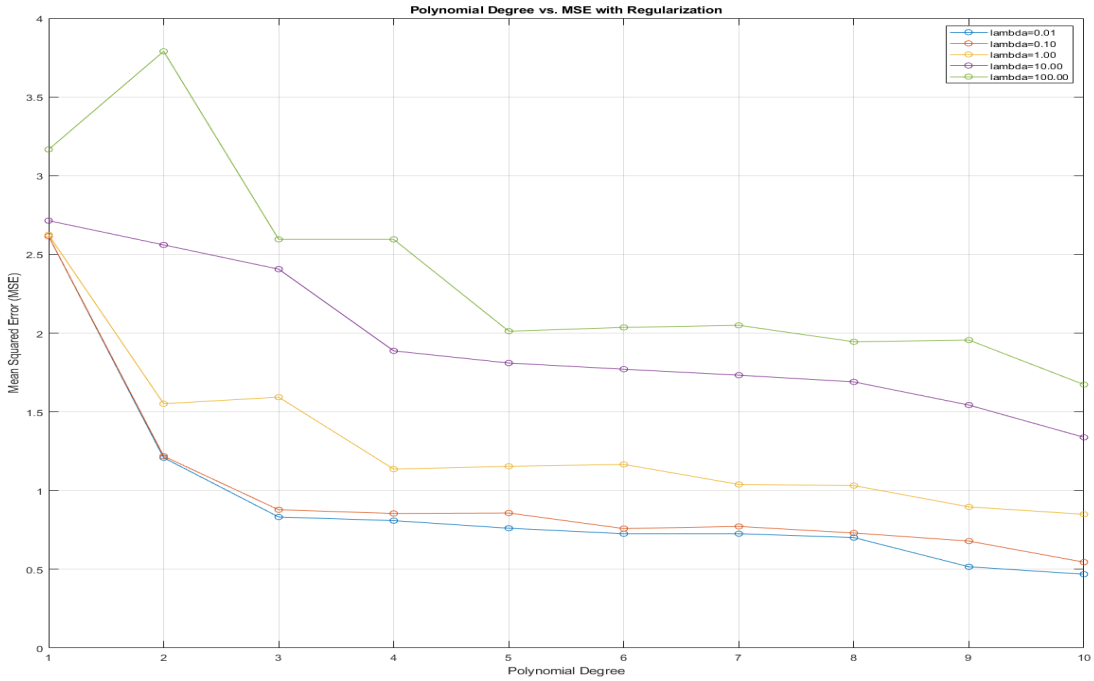

Figure 29: Polynomial Degree vs. Mean Squared Error for various regularization parameters. Each line represents a different λ, showing the trade-off between bias and variance.

The MSE for each polynomial degree and regularization strength is calculated. As expected, higher polynomial degrees without regularization lead to overfitting, which is evidenced by the increasing MSE. However, introducing regularization significantly improves it across various polynomial degrees. The graph shows that as the polynomial degree increases, regularization becomes more critical in controlling the model's complexity. For example, with a regularization parameter of 100 the MSE trends downwards as the degree increases, showing the regularization's ability to smooth out variance due to higher order terms. On the othe hand, lower λ values don't provide sufficient penalization so the MSE increases with the polynomial degree.

3.2.c)

Here we analyze the impact of data asymmetry on linear regression performance. Using a dataset of 10 points with a noise variance of 1, we introduce asymmetry to challenge the regression model further.

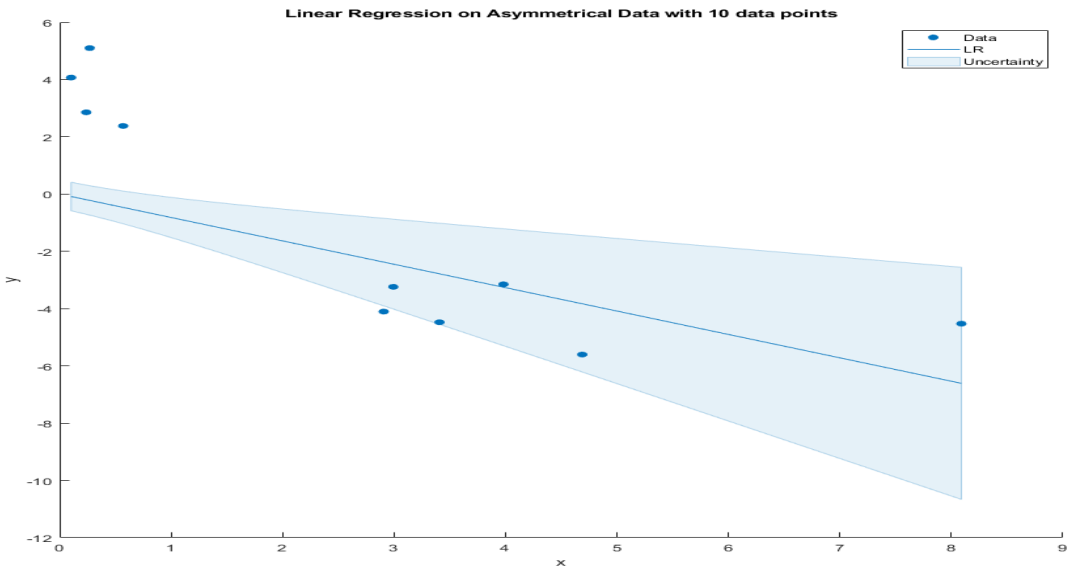

Figure 30: Linear regression fit on asymmetrical data. The shaded area represents the uncertainty in the prediction, indicating increased variance due to the asymmetrical nature of the data.

The MSE is computed to quantify the model's predictive accuracy on this asymmetrical dataset. The results are as follows:

```
Theta Estimates for Asymmetrical Data:
   -0.7873
MSE for Asymmetrical Data:
```

```
3.2586
```

The negative slope of the estimated parameters suggests an inverse relationship in the data, which the LR model attempts to capture. However, the relatively high MSE indicates the model's limited capacity to accurately represent the asymmetrical data distribution. The graph visually supports this, showing a large uncertainty band that does not tightly conform to the data points, reflecting the model's limitations under the given data conditions.

Task 3.2.d

Here we investigated the performance of the KNN regression model across different levels of noise in the data and different choices of k, the number of neighbors to consider. We kept the sample size fixed at 100 and varied the noise variance (σ^2) to be 0.5, 1, and 2. We also experimented with k values ranging from 1 to 10. first we generate synthetic data using the `polyData` function with the specified noise variance, and then split this data into training set (70%) and a testing set (30%). A KNN model was trained on the training set for each value of k and evaluated on the testing set to compute the MSE, where the values were plotted against the number of neighbors k for each noise level. The trends observed are as follows:

- For lower noise variances, the MSE tends to decrease as k increases up to a certain point, after which the improvement plateaus or slightly worsens. This shows that a moderate number of neighbors help to reduce overfitting.
- At higher noise variances the optimal k tends to be smaller, suggesting that fewer neighbors are better when the data are highly noisy.
- There is a trade-off between bias and variance, as too few neighbors (low k) can lead to high variance, where too many neighbors (high k) can lead to high bias.

The optimal choice of k depends on the noise level and the structure of the data. cross-validation can be used to select k in practical scenarios. Following are the plots that show the relationship between the MSE and the number of neighbors k for each noise variance level, where each plot captures the effect of k on the model quality within a particular noise setting:

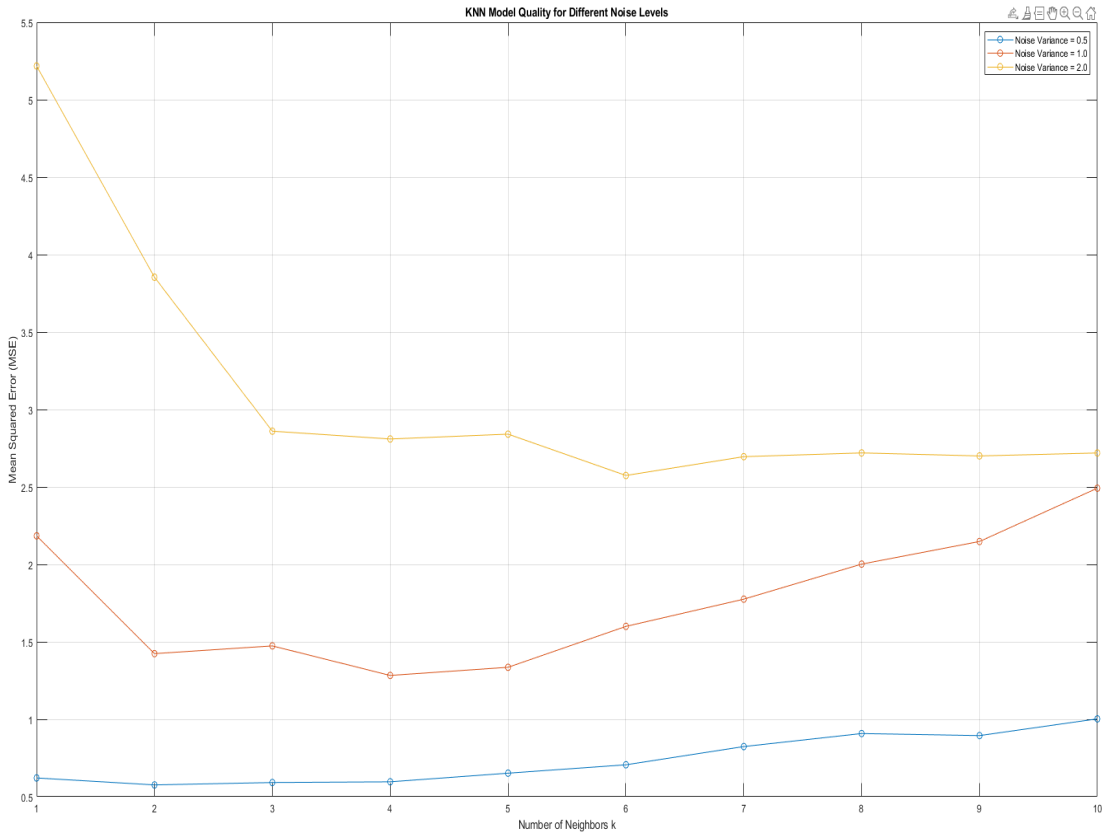

Figure 31: KNN Model Quality with Noise Variances 0.5, 1 and 2

Task 3.3.a

Here we evaluate the effects of model complexity on the generalization ability of polynomial models. Two datasets with noise variance of 0.2 are considered: a small dataset with $N = 50$ points and a larger dataset with $N = 1000$ points. Polynomial models of degrees ranging from 1 to 10 were fitted to both datasets. For the small dataset, the model with the lowest MSE was of degree 10, with an MSE of 0.2614. Similarly, for the large dataset, the model of degree 10 also yielded the lowest MSE, valued at 0.5080. This suggests that higher-degree polynomials may capture the data's structure more effectively, even when the dataset size increases. However, caution is warranted due to the risk of overfitting, especially in the small dataset scenario.

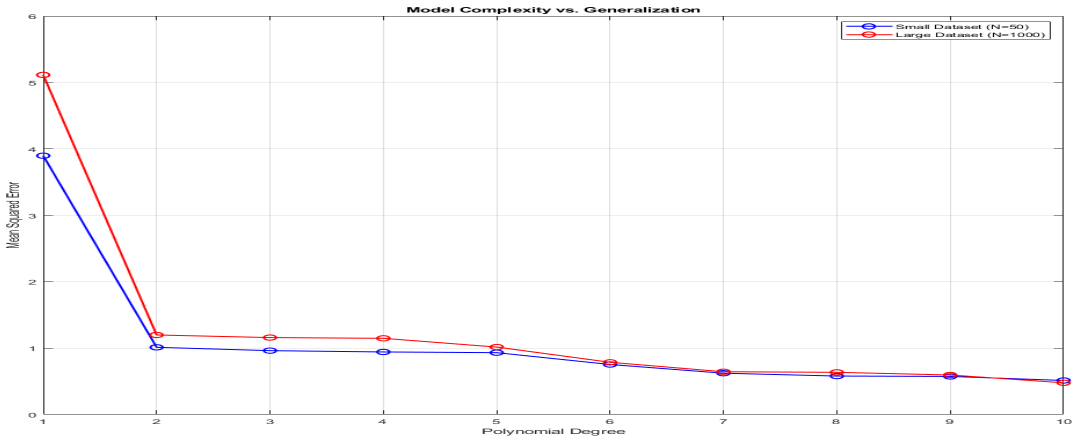

Figure 32: Mean Squared Error (MSE) against polynomial degree for small and large datasets, showing the effect of model complexity on the generalization performance.

Task 3.3.b)

Here we examine the role of regularization in mitigating overfitting in high-degree polynomial regression. A dataset with $N = 50$ data points and a noise variance of 0.1 was used to train a 10th-degree polynomial model. Several regularization strengths (λ) were tested: 0.01, 0.1, 1, 10, and 100. Regularization effectively controls the model's complexity by penalizing the magnitude of the coefficients, thereby reducing overfitting. The regularization strength that achieved the lowest MSE was $\lambda = 0.01$, with an MSE of 0.2362. This implies that a small amount of regularization can improve model performance on the given dataset without leading to underfitting, which can occur with excessive regularization.

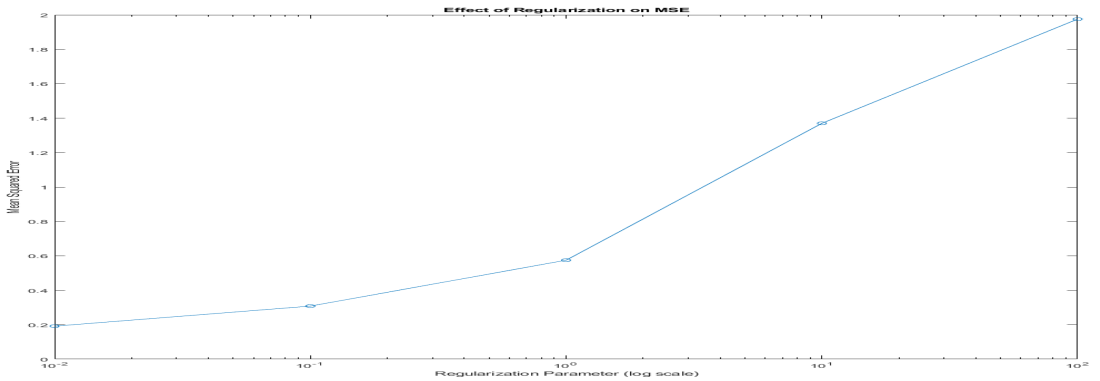

Figure 33: The effect of regularization strength on the mean squared error of a 10th-degree polynomial model. A lower MSE suggests better generalization capability.

Task 3.3.c)

Given a dataset generated from chirp signals with added noise, we aim to investigate the performance of the KNN regression model for different values of k. The dataset consists of $N = 50$ points with a noise variance of 0.2. The dataset is split into training (70%) and validation (30%) sets. We trained KNN models for values of k from 1 to 10.

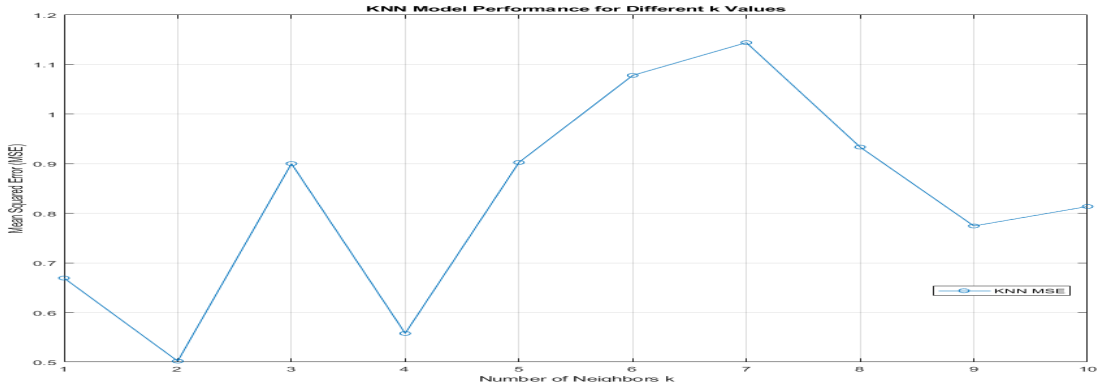

Figure 34: KNN model performance on the validation set for different values of k.

The MSE was found to be lowest for $k = 2$, indicating that this is the optimal number of neighbors for this dataset and noise level. The corresponding MSE for this k value was 2. This suggests that the model achieves the best generalization performance with a moderate level of complexity as determined by the number of neighbors. The relationship between model complexity and generalization is not strictly linear. As k increases, the model complexity decreases, but up to a certain point, after which the performance may plateau or even degrade due to over-smoothing.

Task 4.1.a)

In order to visualize the true function surface for the two-dimensional function estimation problem, we generated a set of noise-free data using the `twoDimData1` function provided. The data consists of 1000 points, which enables a clear and detailed representation of the function's behavior. Using MATLAB's `meshgrid` function we established a grid that covers the input space and allows to interpolate the values of the true function using `griddata` and visualize it as a three-dimensional surface using the `mesh` function. To enhance the visualization, the original noise-free data points were overlaid on the surface plot with markers showing a clear distinction between the interpolated surface and the actual data points. Figure 35 shows the resulting plot where the x-axis represents the first input variable ($X1$), the y-axis represents the second input variable ($X2$), and the z-axis represents the response of the true function.

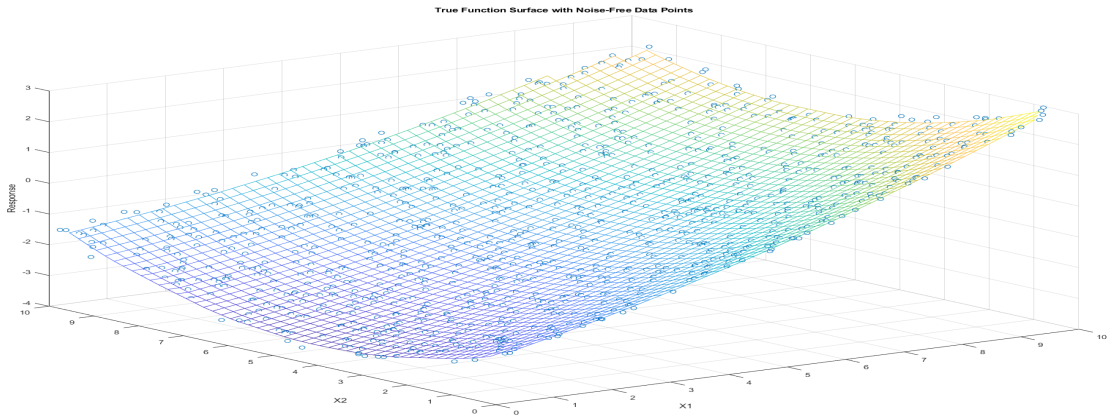

Figure 35: True function surface visualized with noise-free data points.

Task 4.1.b)

The quality of the fit for our models was further analyzed using noisy data sets for both training and validation purposes. The noise variance was set to 1 to simulate realistic conditions where measurements are affected by external factors. The noisy training and validation data points are visualized in Figures 36 and 37, respectively.

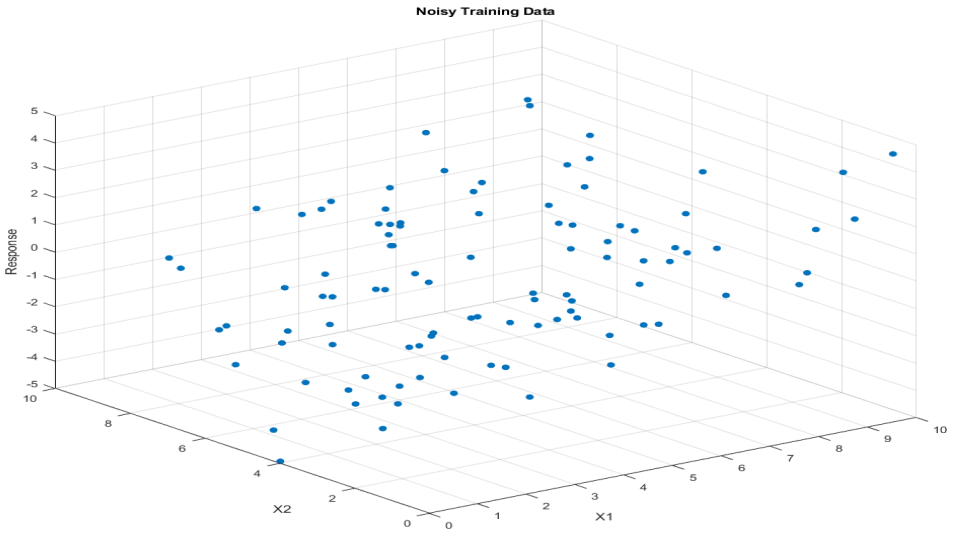

Figure 36: Scatter plot of noisy training data.

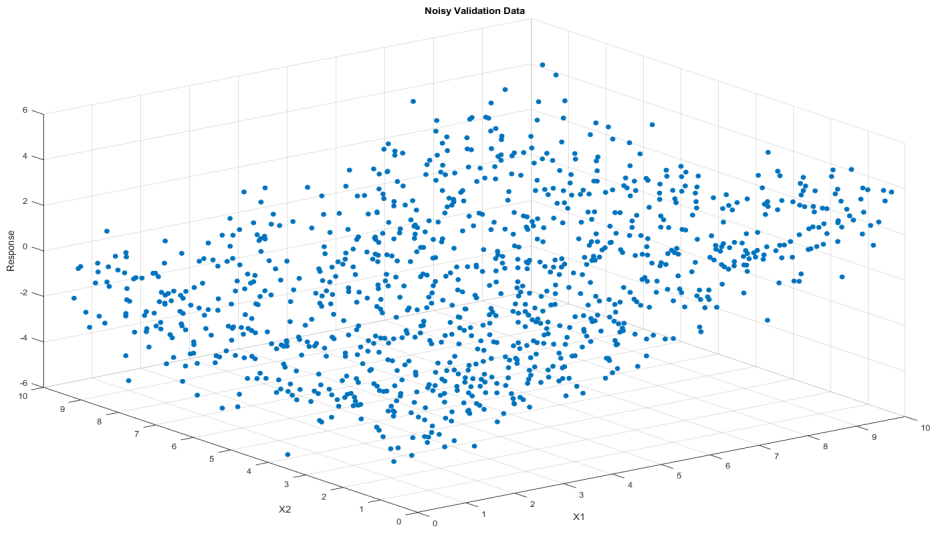

Figure 37: Scatter plot of noisy validation data.

4.1.c)

An LR model was fitted to the noisy training data, which assumes a linear relationship between the input features and the output response. The goal was to evaluate the model's predictive performance on the validation set. The computed MSE for the LR model which provides a quantitative measure of the model's performance on unseen data found to be 2.8800.

4.1.d)

To capture potential nonlinear relationships within the data, polynomial regression models of degrees 2 to 5 were evaluated. Each model's complexity increases with the degree of the polynomial. The MSE values for the polynomial regression models of varying degrees are as follow:

Polynomial Degree	**MSE**
2	1.2019
3	1.2167
4	1.2863
5	2.5722

Table 1: Mean Squared Error for Polynomial Regression Models on Validation Data

The results indicate that as the polynomial degree increases, the MSE initially decreases, reflecting an improved fit. However, beyond a certain point, the MSE starts increasing, which suggests that the model may be overfitting the data. The optimal degree for the polynomial regression model here is with polynomial degree of 2.

4.1.e)

The MSE was calculated for a range of k values and the results were analyzed to find the optimal number of neighbors. The KNN model with the lowest MSE had k = 11, and with the lowest MSE equal to 1.6591 suggesting that this is the optimal number of neighbors for our data. Upon comparison, it was found that the polynomial model performed better with an MSE of 1.2019. In addition to that, the polynomial model appears to always perform better, for a wide range of N values (from 1e1 up to 1e8)

4.2.a)

For an LR model, the number of regressors is equivalent to the number of dimensions plus one (additional regressor is the intercept). Given a dataset with ten dimensions, the total number of regressors R_{linear} is calculated as:

$$R_{linear} = D + 1$$

where D = number of dimensions. For our dataset:

$$R_{linear} = 10 + 1 = 11$$

When extending the model to include polynomial terms up to the third degree, the number of regressors significantly increases. It comprises the linear terms, the squared terms, the cross-product terms for the quadratic component, and the corresponding terms for the cubic component, including all possible combinations of features. The total number of regressors R_{poly} is computed as:

$$\begin{aligned} R_{poly} &= 1 + D + \binom{D}{2} + D + D(D-1) + \binom{D}{3} \\ &= 1 + 10 + \binom{10}{2} + 10 + 10(10-1) + \binom{10}{3} \\ &= 1 + 10 + 45 + 10 + 90 + 120 \\ &= 286 \end{aligned}$$

4.2.b)

As part of the process to assess the performance of regression models on high-dimensional data, we first generate noisy datasets for both training and validation purposes. The parameters were used for the data generation are, the Noise variance: $\sigma^2 = 1$, the number of data points for training: $N_{train} = 1000$ and the number of data points for validation: $N_{val} = 10000$. The MATLAB function `tenDimData` has been used.

4.2.c)

After acquiring the noisy training and validation datasets, we fitted an LR model using the training data. The MSE obtained for the LR model on the validation set is 1.2251.

4.2.d)

For this task, we used the same noisy datasets generated in part 4.2.b for training and validation. The datasets were extended to include polynomial features up to the third degree. The polynomial features of degree 3 were generated for training and validation data. A polynomial regression model was fitted using the training data, and then the model's performance was evaluated on the validation dataset using the MSE. The MSE for the polynomial regression model of degree 3 on the validation dataset is 1.4812.

4.2.e)

Regularization has been added into the polynomial regression model by adding a penalty term proportional to the square of the magnitude of the coefficients. We experimented with various regularization strengths to find the optimal balance between bias and variance. The best regularization strength achieved was $\lambda = 30$. The MSE for the model with this regularization strength was 1.0477. Regularization has led to an improvement in MSE, indicating a better-performing model in comparison to the unregularized one.

4.2.f)

A series of KNN models with varying numbers of neighbors were trained on a dataset with ten dimensions. The KNN regression model was found to perform optimal with $k = 19$ neighbors, resulting in an MSE of 1.2214. This result compared to the best polynomial model with an MSE of 1.2019 doesn't outperforme it. The best polynomial model outperforms the KNN model with 19 neighbors. Despite KNN's flexibility in modeling nonlinear relationships, in this scenario, the polynomial model provided a slightly better fit to the validation data.