Homework 5

Solution 1

(a) Plot of $f(\theta)$ for $-1 \leq \theta \leq 1.2$

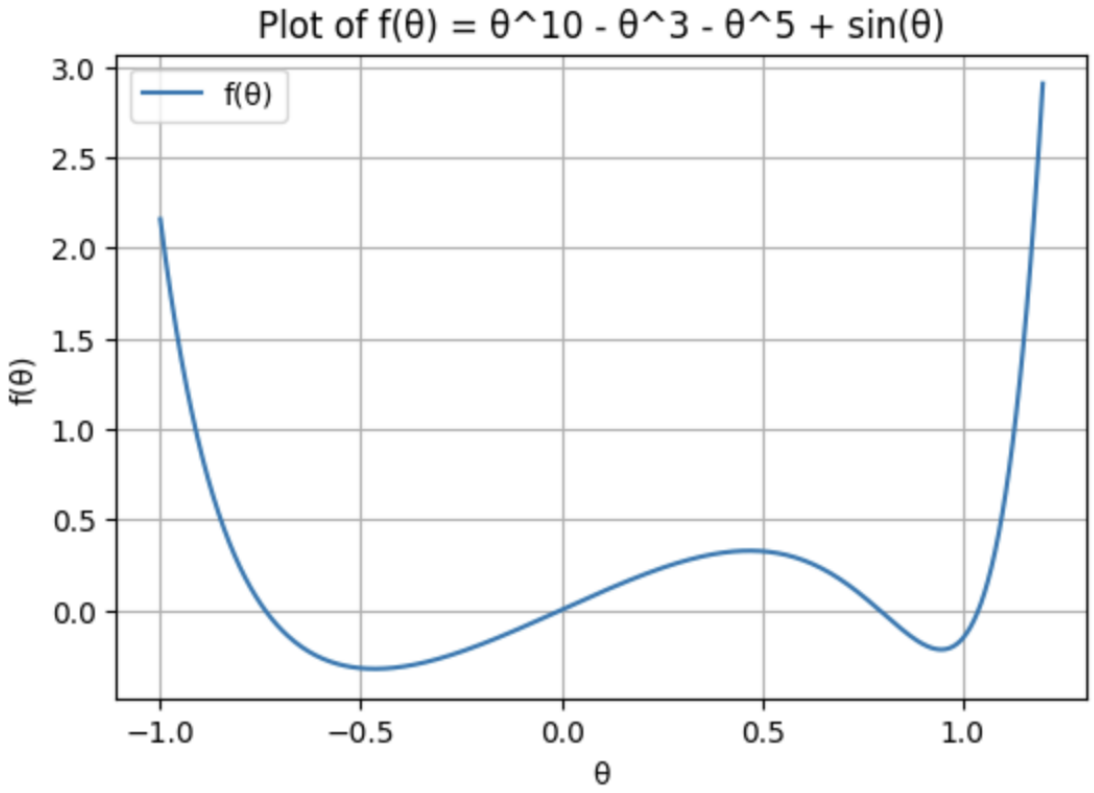


Figure 1: (1a) Estimated piecewise linear function with knot at $X = 1$

(b) Initial $\theta_0 = 0$, learning rate $\eta = 0.01$, and number of iterations $N = 200$.

$$f' = 10\theta^9 - 3\theta^2 - 5\theta^4 + cos(\theta)$$

(i) Final $\theta_N = -0.465201$
Yes, this makes sense as function $f(\theta)$ from part (a) shows a global minimum around $\theta = -0.5$. The gradient descent path starts at 0 and steadily descends to a low point close to that minimum.

(ii) As n increases $f(\theta_n)$ decreases monotonically and plateaus near 50-60 iterations. This pattern suggests that the learning rate is appropriate for gradient descent to work well.

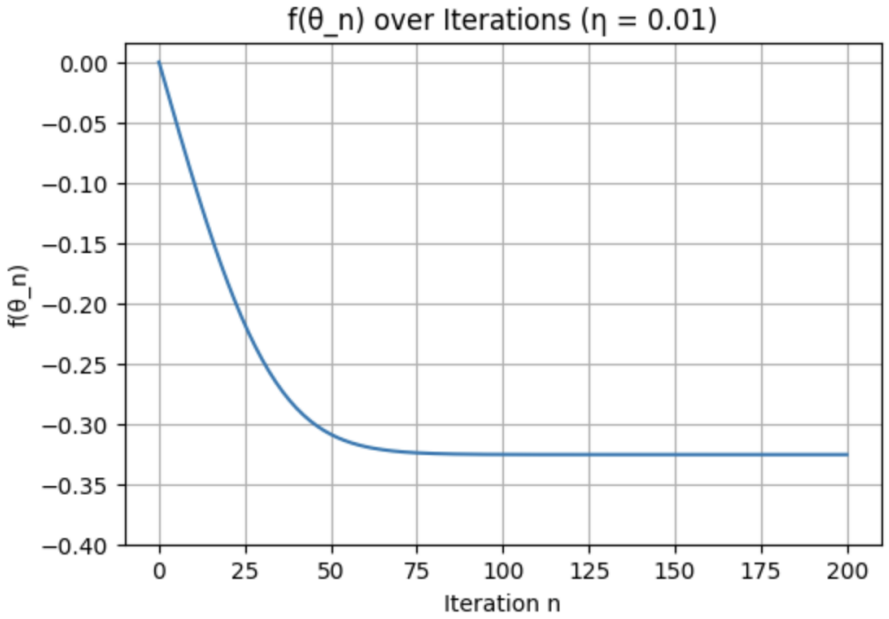


Figure 2: (1b.ii) $f(\theta_n)$ vs number of iterations

(c) $\eta = 10^{-6},\ 10^{-4},\ 10^{-2},\ 0.1,\ 0.5,\ 1$

(i) Plots for different values of η

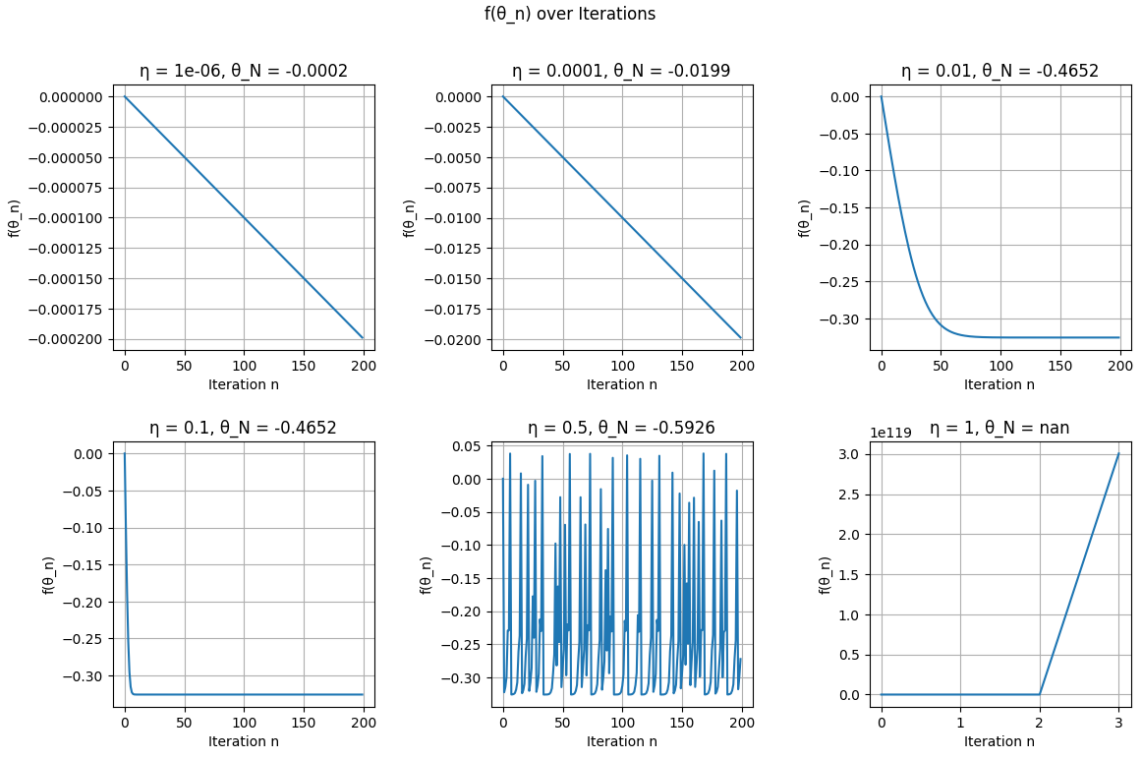


Figure 3: (1c.i) $f(\theta_n)$ vs number of iterations

(ii) Plot of $f(\theta_n)$ v/s θ

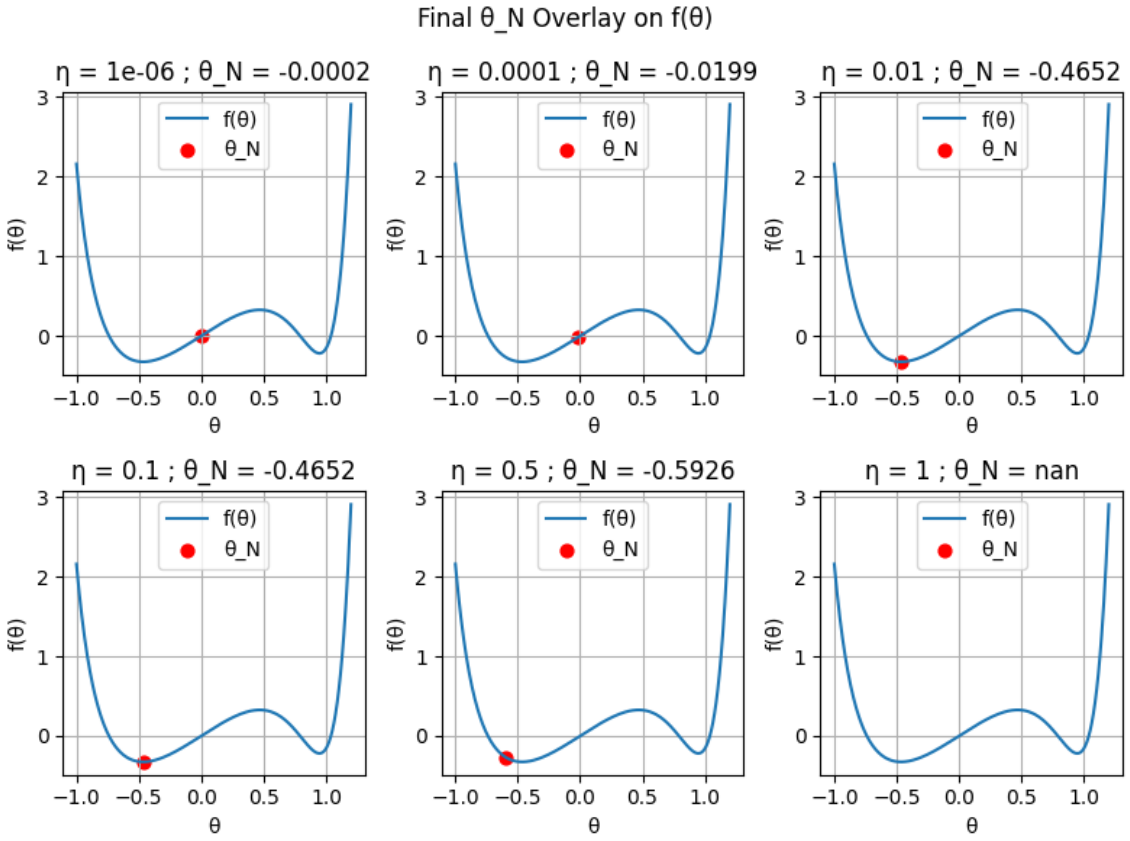


Figure 4: (1c.ii) $f(\theta_n)$ v/s θ

(iii) The graphs show that the learning rate η has a major impact on gradient descent performance. Very small values like 10^{-6} and 10^{-4} are too slow, with almost no progress after 200 iterations. Moderate values like 0.01 and 0.1 work well and lead to fast, stable convergence near the global minimum as seen in both the iteration plots and final overlays. In contrast, larger values like 0.5 cause oscillations, and $\eta = 1$ leads to divergence with exploding values and `NaN`s. Only $\eta = 0.01$ and 0.1 result in sensible minimizers, showing that tuning the learning rate is essential for successful optimization.

(d) Proportion of repetitions of gradient descent results that did not converge to the actual global minimum = 0.24
It is observed that the final θ_N clustered into two main regions: one near the global minimum and another around a local minimum. The histogram shows that approximately 24% of the runs did not converge to the global minimum. This behavior highlights the non-convex nature of the function $f(\theta)$, which contains multiple minima. Because gradient descent is a local optimization method, its outcome is sensitive to the initial starting point. If the initial value lies closer to a local minimum, the algorithm may get trapped there instead of finding the global optimum.

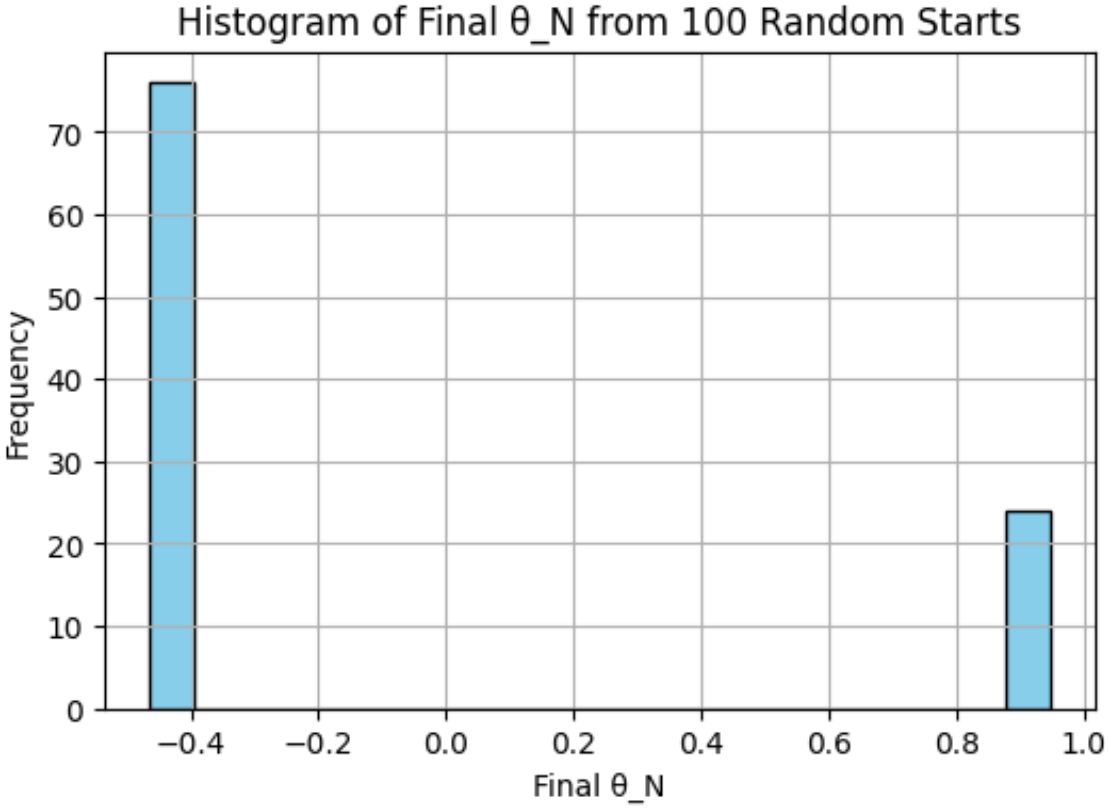
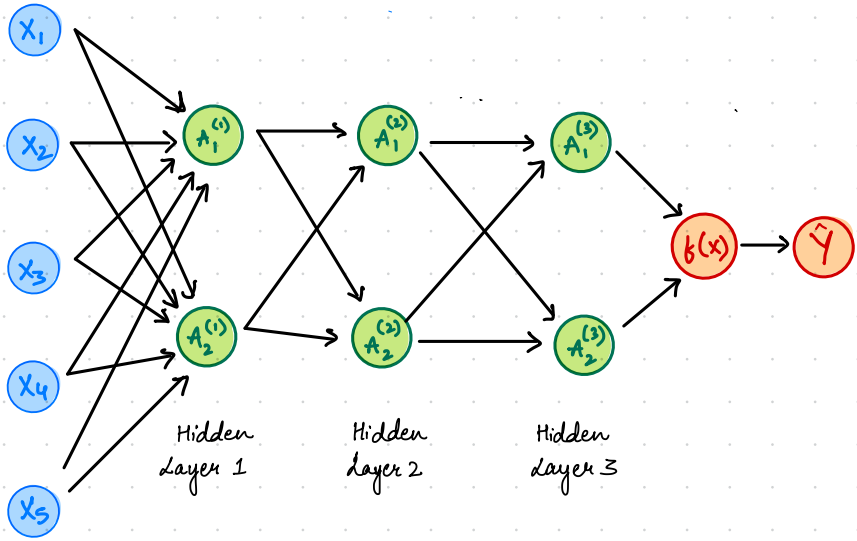


Figure 5: (1d) Histogram of results

Solution 2

(a)



$$A_1^{(1)} = g\left(w_{10}^{(1)} + w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{13}^{(1)} X_3 + w_{14}^{(1)} X_4 + w_{15}^{(1)} X_5\right)$$

$$A_2^{(1)} = g\left(w_{20}^{(1)} + w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{23}^{(1)} X_3 + w_{24}^{(1)} X_4 + w_{25}^{(1)} X_5\right)$$

$$A_1^{(2)} = g\left(w_{10}^{(2)} + w_{11}^{(2)} A_1^{(1)} + w_{12}^{(2)} A_2^{(1)}\right)$$

$$A_2^{(2)} = g\left(w_{20}^{(2)} + w_{21}^{(2)} A_1^{(1)} + w_{22}^{(2)} A_2^{(1)}\right)$$

$$A_1^{(3)} = g\left(w_{10}^{(3)} + w_{11}^{(3)} A_1^{(2)} + w_{12}^{(3)} A_2^{(2)}\right)$$

$$A_2^{(3)} = g\left(w_{20}^{(3)} + w_{21}^{(3)} A_1^{(2)} + w_{22}^{(3)} A_2^{(2)}\right)$$

$$f(X) = \beta_0 + \beta_1 A_1^{(3)} + \beta_2 A_2^{(3)}$$

$$\hat{Y} = f(x)$$

Here g is a non linear function. (ReLU/ Sigmoid)

(b) Minimize RSS $= \sum_{i=1}^{n} (y_i - f(x_i))^2$ where n is no of training samples

$6*2 + 2*3 + 2*3 + 3 = 27$

Number of parameters to estimate = 27.

(c) Now $g(x) = x$

Equations of forward pass will now be

$$A_1^{(1)} = w_{10}^{(1)} + w_{11}^{(1)} X_1 + w_{12}^{(1)} X_2 + w_{13}^{(1)} X_3 + w_{14}^{(1)} X_4 + w_{15}^{(1)} X_5 \quad \text{(i)}$$

$$A_2^{(1)} = w_{20}^{(1)} + w_{21}^{(1)} X_1 + w_{22}^{(1)} X_2 + w_{23}^{(1)} X_3 + w_{24}^{(1)} X_4 + w_{25}^{(1)} X_5 \quad \text{(ii)}$$

$$A_1^{(2)} = w_{10}^{(2)} + w_{11}^{(2)} A_1^{(1)} + w_{12}^{(2)} A_2^{(1)} \quad \text{(iii)}$$

$$A_2^{(2)} = w_{20}^{(2)} + w_{21}^{(2)} A_1^{(1)} + w_{22}^{(2)} A_2^{(1)} \quad \text{(iv)}$$

$$A_1^{(3)} = w_{10}^{(3)} + w_{11}^{(3)} A_1^{(2)} + w_{12}^{(3)} A_2^{(2)} \quad \text{(v)}$$

$$A_2^{(3)} = w_{20}^{(3)} + w_{21}^{(3)} A_1^{(2)} + w_{22}^{(3)} A_2^{(2)} \quad \text{(vi)}$$

$$f(X) = \beta_0 + \beta_1 A_1^{(3)} + \beta_2 A_2^{(3)} \quad \text{(vii)}$$

$$\hat{Y} = f(X)$$

(d) Substituting eq (i) and (ii) in (iii) and (iv)

(iii) $$A_1^{(2)} = w_{10}^{(2)} + w_{11}^{(2)}\left(w_{10}^{(1)} + w_{11}^{(1)}X_1 + w_{12}^{(1)}X_2 + w_{13}^{(1)}X_3 + w_{14}^{(1)}X_4 + w_{15}^{(1)}X_5\right) + w_{12}^{(2)}\left(w_{20}^{(1)} + w_{21}^{(1)}X_1 + w_{22}^{(1)}X_2 + w_{23}^{(1)}X_3 + w_{24}^{(1)}X_4 + w_{25}^{(1)}X_5\right)$$

(iv) $$A_2^{(2)} = w_{20}^{(2)} + w_{21}^{(2)}\left(w_{10}^{(1)} + w_{11}^{(1)}X_1 + w_{12}^{(1)}X_2 + w_{13}^{(1)}X_3 + w_{14}^{(1)}X_4 + w_{15}^{(1)}X_5\right) + w_{22}^{(2)}\left(w_{20}^{(1)} + w_{21}^{(1)}X_1 + w_{22}^{(1)}X_2 + w_{23}^{(1)}X_3 + w_{24}^{(1)}X_4 + w_{25}^{(1)}X_5\right)$$

Rearranging eq (iii)

$$A_1^{(2)} = w_{10}^{(2)} + w_{11}^{(2)}w_{10}^{(1)} + w_{12}^{(2)}w_{20}^{(1)} + \left(w_{11}^{(2)}w_{11}^{(1)} + w_{12}^{(2)}w_{21}^{(1)}\right)X_1 + \left(w_{11}^{(2)}w_{12}^{(1)} + w_{12}^{(2)}w_{22}^{(1)}\right)X_2 + \left(w_{11}^{(2)}w_{13}^{(1)} + w_{12}^{(2)}w_{23}^{(1)}\right)X_3 + \left(w_{11}^{(2)}w_{14}^{(1)} + w_{12}^{(2)}w_{24}^{(1)}\right)X_4 + \left(w_{11}^{(2)}w_{15}^{(1)} + w_{12}^{(2)}w_{25}^{(1)}\right)X_5$$

This is linear in X, can be written as

$$A_1^{(2)} = a_{10} + a_{11}X_1 + a_{12}X_2 + a_{13}X_3 + a_{14}X_4 + a_{15}X_5 \quad \text{(viii)}$$

Similarly eq (iv) will be linear in X. can be written as

$$A_2^{(2)} = a_{20} + a_{21}X_1 + a_{22}X_2 + a_{23}X_3 + a_{24}X_4 + a_{25}X_5 \quad \text{(ix)}$$

Substituting eq (viii) and (ix) in eq (v) and (vi)

(v) $A_1^{(3)} = w_{10}^{(3)} + w_{11}^{(3)}\left(a_{10} + a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + a_{15}x_5\right)$

$$+ w_{12}^{(3)}\left(a_{20} + a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 + a_{25}x_5\right)$$

$$A_1^{(3)} = w_{10}^{(3)} + w_{11}^{(3)}a_{10} + w_{12}^{(3)}a_{20} + \left(w_{11}^{(3)}a_{11} + w_{12}^{(3)}a_{21}\right)x_1 + \left(w_{11}^{(3)}a_{12} + w_{12}^{(3)}a_{22}\right)x_2$$

$$+ \left(w_{11}^{(3)}a_{13} + w_{12}^{(3)}a_{23}\right)x_3 + \left(w_{11}^{(3)}a_{14} + w_{12}^{(3)}a_{24}\right)x_4 + \left(w_{11}^{(3)}a_{15} + w_{12}^{(3)}a_{25}\right)x_5$$

This is linear in x, can be written as

$$A_1^{(3)} = b_{10} + b_{11}x_1 + b_{12}x_2 + b_{13}x_3 + b_{14}x_4 + b_{15}x_5$$

Similarly eq (vi) can be written as

$$A_2^{(3)} = b_{20} + b_{21}x_1 + b_{22}x_2 + b_{23}x_3 + b_{24}x_4 + b_{25}x_5$$

Substituting in eq (vii)

$$f(x) = \beta_0 + \beta_1 A_1^{(3)} + \beta_2 A_2^{(3)}$$

$$f(x) = \beta_0 + \beta_1\left(b_{10} + b_{11}x_1 + b_{12}x_2 + b_{13}x_3 + b_{14}x_4 + b_{15}x_5\right)$$

$$+ \beta_2\left(b_{20} + b_{21}x_1 + b_{22}x_2 + b_{23}x_3 + b_{24}x_4 + b_{25}x_5\right)$$

$$f(x) = \left(\beta_0 + \beta_1 b_{10} + \beta_2 b_{20}\right) + \left(\beta_1 b_{11} + \beta_2 b_{21}\right)x_1 + \left(\beta_1 b_{12} + \beta_2 b_{22}\right)x_2$$

$$+ \left(\beta_1 b_{13} + \beta_2 b_{23}\right)x_3 + \left(\beta_1 b_{14} + \beta_2 b_{24}\right)x_4 + \left(\beta_1 b_{15} + \beta_2 b_{25}\right)x_5$$

This is linear in x can be written as

$$f(x) = \beta_0' + \beta_1' x_1 + \beta_2' x_2 + \beta_3' x_3 + \beta_4' x_4 + \beta_5' x_5$$

where

$$\beta_0' = \beta_0 + \beta_1\left(w_{10}^{(3)} + w_{11}^{(2)} a_{10} + w_{12}^{(3)} a_{20}\right) + \beta_2\left(w_{20}^{(3)} + w_{22}^{(3)} a_{10} + w_{22}^{(3)} a_{20}\right)$$

$$\begin{aligned}\beta_0' = \beta_0 + \beta_1\Big(&w_{10}^{(3)} + w_{11}^{(3)}\left(w_{10}^{(2)} + w_{11}^{(2)} w_{10}^{(1)} + w_{12}^{(2)} w_{20}^{(1)}\right) \\ &+ w_{12}^{(3)}\left(w_{20}^{(2)} + w_{21}^{(2)} w_{10}^{(1)} + w_{22}^{(2)} w_{20}^{(1)}\right)\Big) \\ + \beta_2\Big(&w_{20}^{(3)} + w_{21}^{(3)}\left(w_{10}^{(2)} + w_{11}^{(2)} w_{10}^{(1)} + w_{12}^{(2)} w_{20}^{(1)}\right) \\ &+ w_{22}^{(3)}\left(w_{20}^{(2)} + w_{21}^{(2)} w_{10}^{(1)} + w_{22}^{(2)} w_{20}^{(1)}\right)\Big)\end{aligned}$$

$$\beta_i' = \beta_1\left[w_{11}^{(3)} a_{1i} + w_{12}^{(3)} a_{2i}\right] + \beta_2\left[w_{21}^{(3)} a_{1i} + w_{22}^{(3)} a_{2i}\right]$$

$$\begin{aligned}\beta_i' = \beta_1\Big[&w_{11}^{(3)}\left(w_{11}^{(2)} w_{1i}^{(1)} + w_{12}^{(2)} w_{2i}^{(1)}\right) + w_{12}^{(3)}\left(w_{21}^{(2)} w_{2i}^{(1)} + w_{22}^{(2)} w_{2i}^{(1)}\right)\Big] \\ + \beta_2\Big[&w_{21}^{(3)}\left(w_{11}^{(2)} w_{1i}^{(1)} + w_{12}^{(2)} w_{2i}^{(1)}\right) + w_{22}^{(3)}\left(w_{21}^{(2)} w_{2i}^{(1)} + w_{22}^{(2)} w_{2i}^{(1)}\right)\Big]\end{aligned}$$

for $i \in [1,2,3,4,5]$

The parameters can be written like above to get a linear model in x when $g(x) = x$.

Solution 3

(a) I use the Breast Cancer Wisconsin (Diagnostic) dataset from the UCI Machine Learning Repository to perform a binary classification task. The goal is to predict whether a tumor is malignant or benign based on a set of features.
Each observation in the dataset corresponds to one patient case and includes 30 real-valued input features such as radius, texture, perimeter, area, smoothness, compactness, and other geometric and statistical properties of the cell nuclei present in the image. The response variable is a binary label:
M = malignant (cancerous)
B = benign (non-cancerous)
The supervised learning task is to train a model that can accurately classify new tumors based on these 30 features into either malignant or benign classes. This is a critical and high accuracy is essential for patient safety and treatment planning.

(b) Training three models

i. The **neural network** that was implemented took in 30 input features corresponding to the predictors in the breast cancer dataset. This was followed by two hidden layers, each containing 10 units and using the ReLU activation function to introduce nonlinearity. The final output layer had one neuron with a Sigmoid activation function to produce a probability score between 0 and 1, suitable for binary classification. The network was trained using the mean squared error loss function and the Adam optimizer with a learning rate of 0.001, over 1000 epochs. Before training, input features were standardized using StandardScaler to ensure stable optimization.
The **logistic regression** model was implemented with the liblinear solver selected to ensure convergence for the relatively small dataset. The model was trained with 2000 iterations. This linear model assumes a direct relationship between the input features and the log-odds of the outcome. Feature standardization was applied to match the conditions under which logistic regression typically performs best.
The **random forest classifier** was implemented with 100 decision trees and default settings. This ensemble model constructs multiple decision trees using bootstrap samples of the data and averages their predictions to improve generalization.

ii. Accuracy results

Model	**Test Accuracy**
Neural Network	0.9649
Logistic Regression	0.9357
Random Forest	0.9474

Table 1: Model performance on the Breast Cancer dataset

Among the three models, the neural network achieved the highest test accuracy at 96.49%, followed by random forest at 94.74%, and logistic regression at 93.57%. This indicates that the neural network was most effective at capturing complex patterns in the data. Random forest also performed well with minimal tuning, while logistic regression, though slightly less accurate, remained a strong linear baseline. Overall, all models performed well, but the neural network showed the best predictive performance.

(c) While all models performed well, the neural network captured the most complex patterns in the data, likely due to its depth and nonlinearity.
In terms of training effort, the neural network took the most time to implement and tune. Setting up the architecture, defining the loss and optimizer, and managing the training loop required an understanding of the training process. However, it offered the most flexibility and the best performance. On the other hand, logistic regression was the quickest to implement using scikit-learn. It required minimal tuning but showed slightly lower accuracy, consistent with its linear nature.
The random forest model struck a good balance: it performed well with almost no tuning and was fast to train. It also handled nonlinearities and interactions better than logistic regression.
I found extensive online resources for all three models. While the neural network performed best, it also required more care and effort to set up correctly.

Extra Credit Below are the two plots. Figure 6 displays the training error and test error associated with $f^(X)$, for a range of values of the degrees of freedom d. Figure 7 show the true function, the fitted natural spline with d = 8 and d=20 degrees of freedom and shows the minimum-norm natural spline fits with d = 42 and d = 80 degrees of freedom.

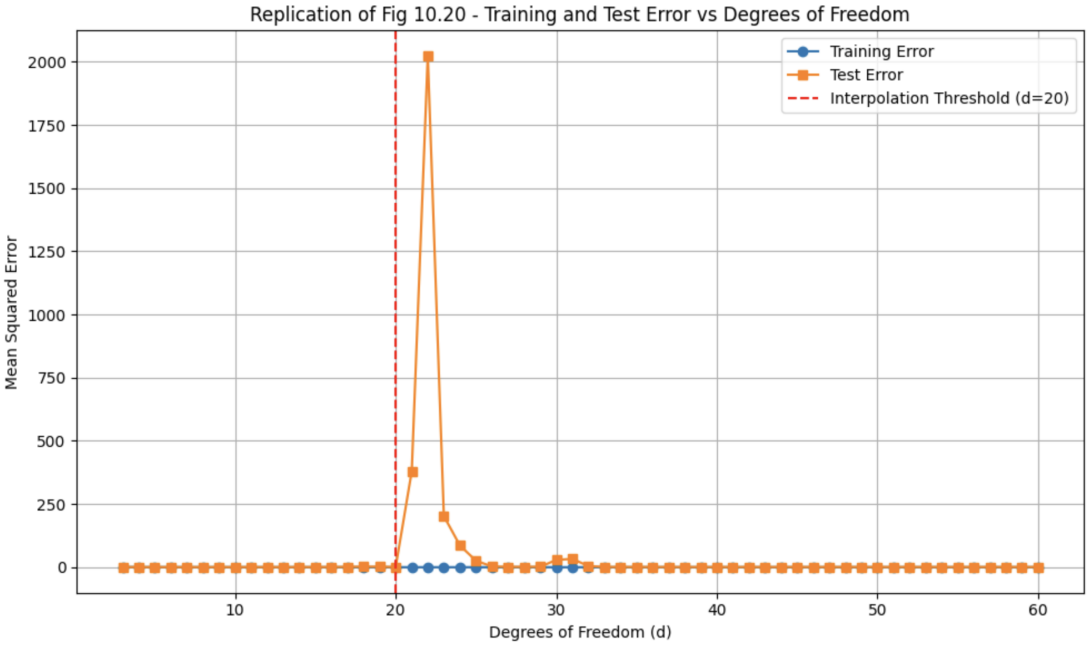


Figure 6: Replication of Figure 10.20 from the book

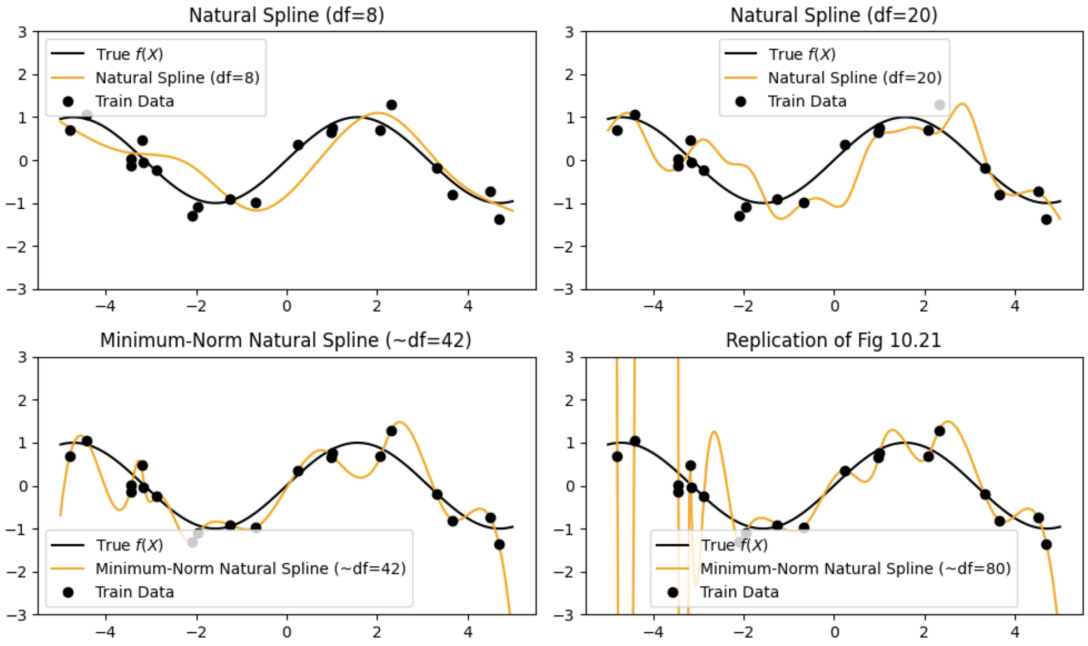


Figure 7: Replication of Figure 10.21 from the book

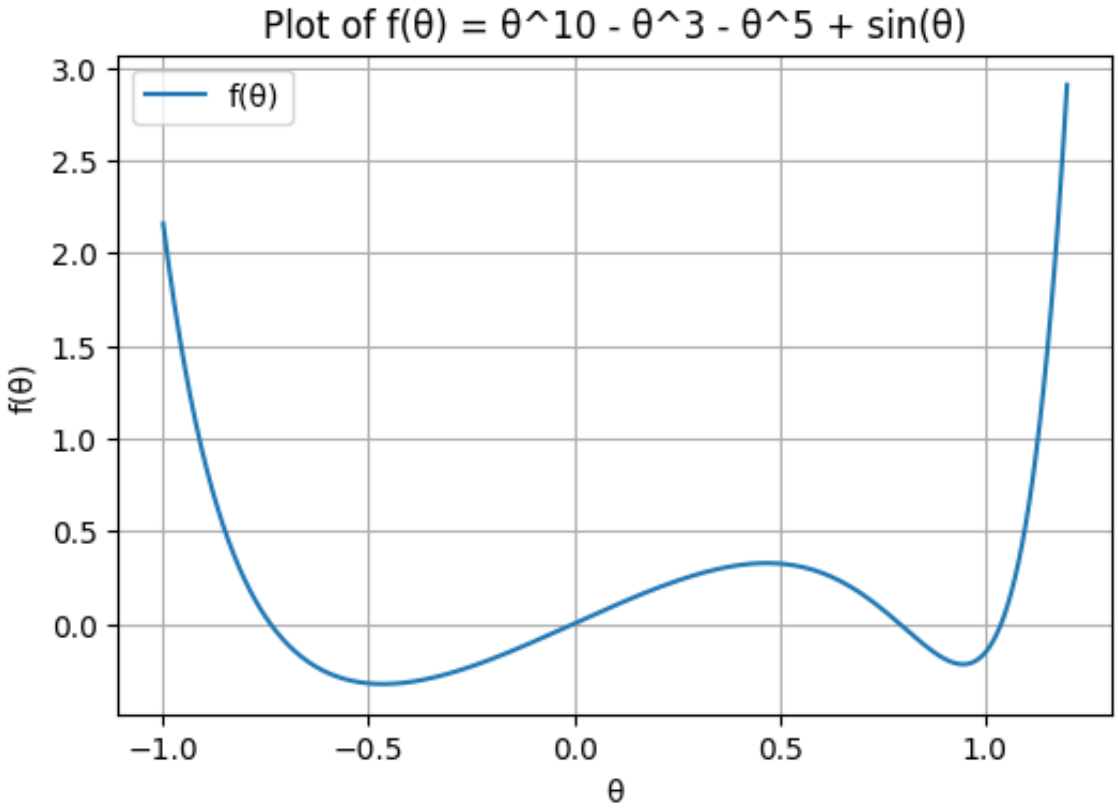
APPENDIX

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from patsy import dmatrix
from sklearn.linear_model import LinearRegression
from scipy.interpolate import UnivariateSpline
```

Question 1

```
def f(theta):
    return theta**10 - theta**3 - theta**5 + np.sin(theta)

def f_prime(theta):
    return 10*theta**9 - 3*theta**2 - 5*theta**4 + np.cos(theta)
```

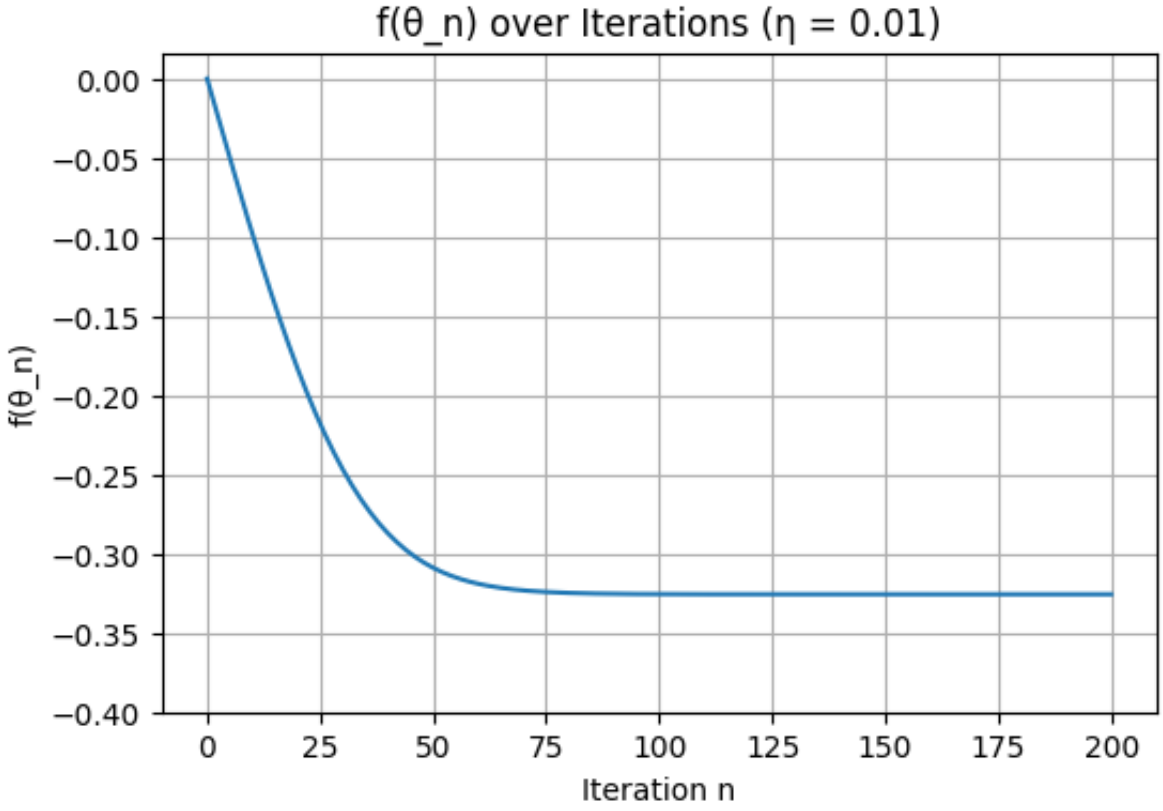
```
thetas = np.linspace(-1, 1.2, 500)
fs = f(thetas)

plt.figure(figsize=(6, 4))
plt.plot(thetas, fs, label='f(θ)')
plt.xlabel('θ')
plt.ylabel('f(θ)')
plt.title('Plot of f(θ) = θ^10 - θ^3 - θ^5 + sin(θ)')
plt.grid(True)
plt.legend()
plt.show()
```



This makes sense as the closest local minimum in near that point only.

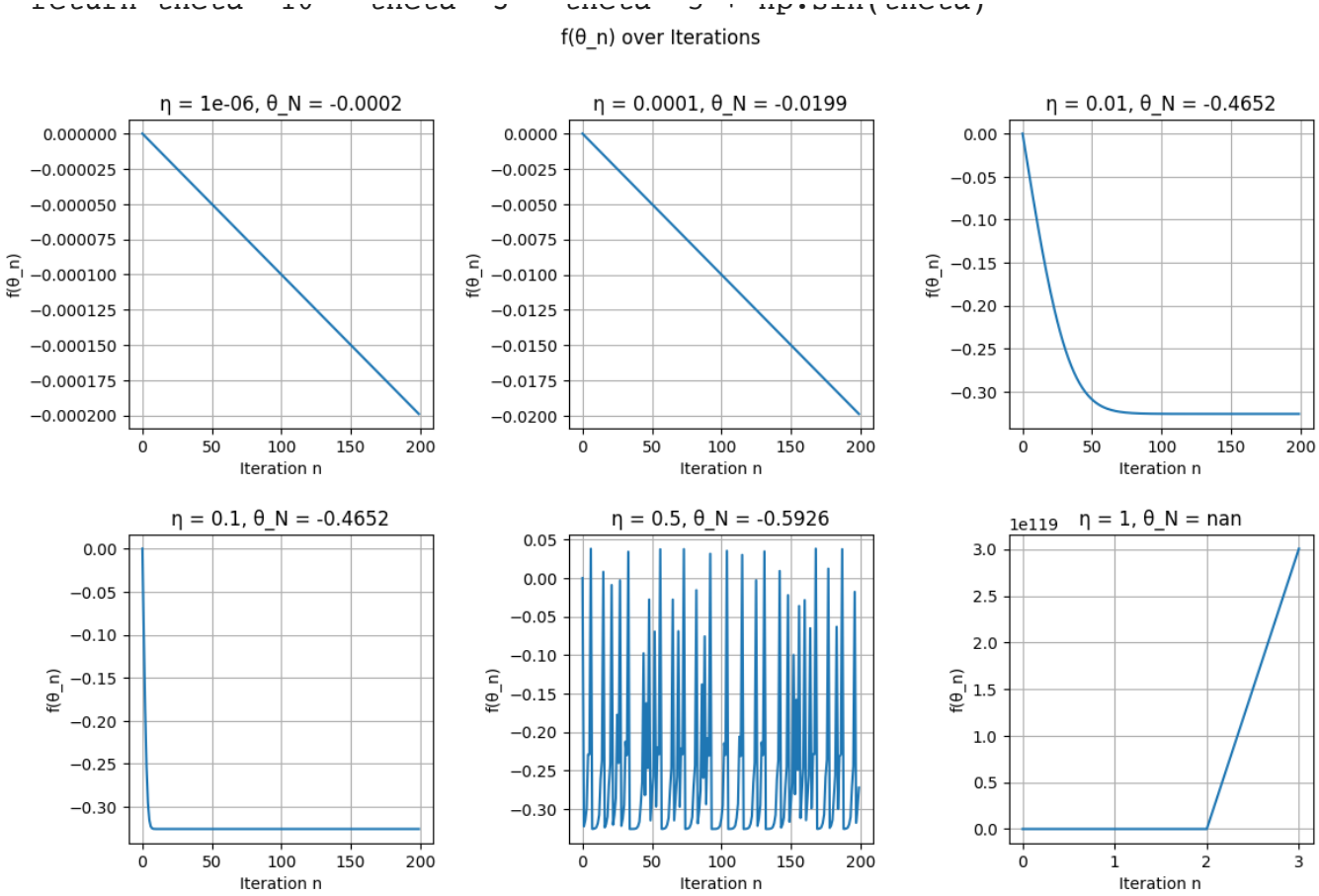
```
intial_theta = 0
eta = 0.01
n = 200

theta_hist = [intial_theta]
for _ in range(n):
    theta_new = theta_hist[-1] - eta * f_prime(theta_hist[-1])
    theta_hist.append(theta_new)

theta_hist = np.array(theta_hist)
f_hist = f(theta_hist)

# (i) Final value
print(f"Final θ_N = {theta_hist[-1]:.6f}")

# (ii) Plot f(θ_n) over iterations
plt.figure(figsize=(6, 4))
plt.plot(range(n+1), f_hist)
plt.xlabel('Iteration n')
plt.ylabel('f(θ_n)')
plt.ylim(bottom=-0.4)
plt.title(f'f(θ_n) over Iterations (η = {eta})')
plt.grid(True)
plt.show()
```

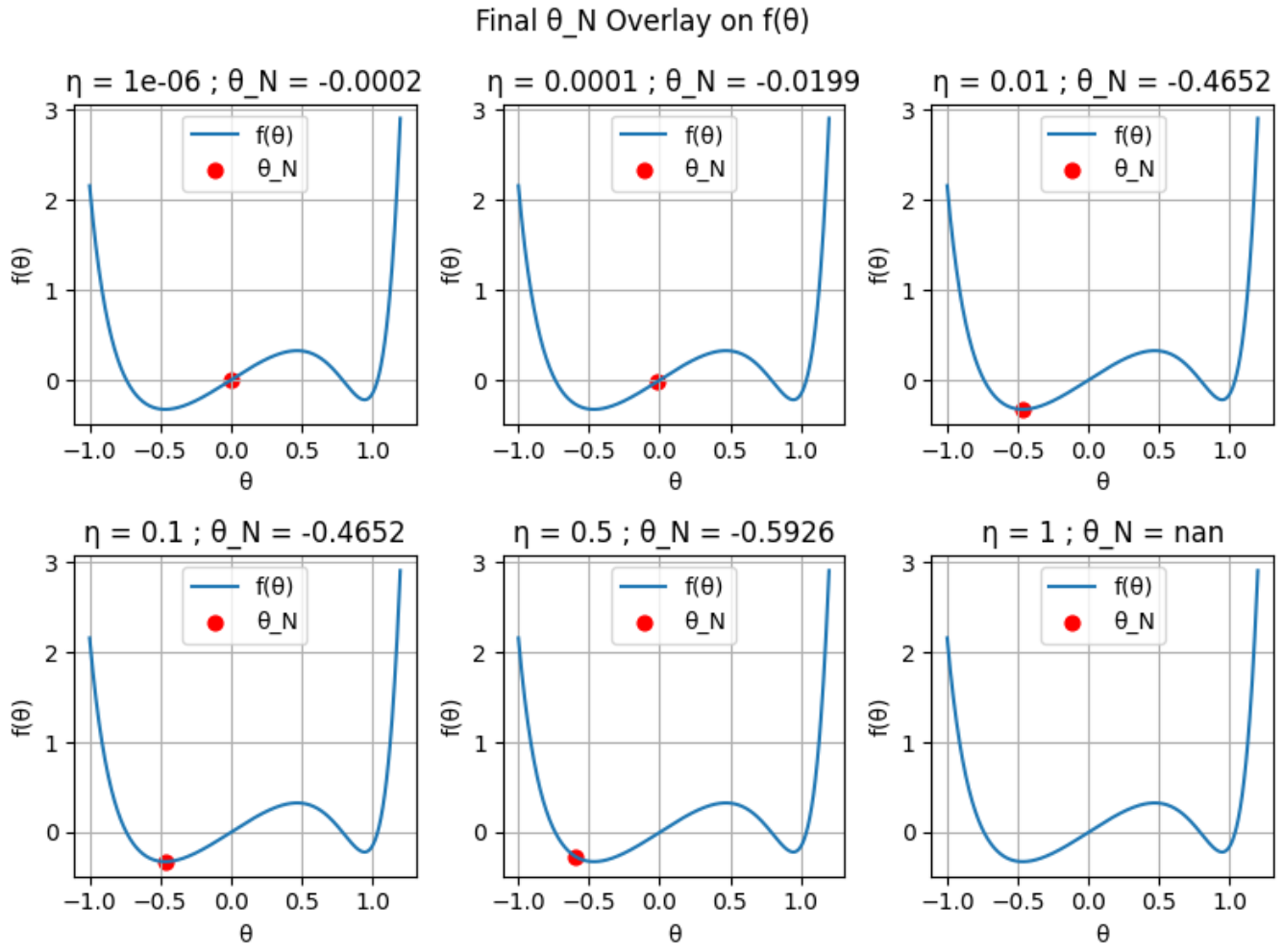
```
Final θ_N = -0.465201
```



```
etas = [1e-6, 1e-4, 1e-2, 0.1, 0.5, 1]

fig1, axes1 = plt.subplots(2, 3, figsize=(12, 8))  # for part (c)(i)
fig2, axes2 = plt.subplots(2, 3, figsize=(8, 6))  # for part (c)(ii)
axes1 = axes1.flatten()
axes2 = axes2.flatten()


for i, e in enumerate(etas):
    theta_hist = [intial_theta]
    for _ in range(n):
        theta_new = theta_hist[-1] - e * f_prime(theta_hist[-1])
        theta_hist.append(theta_new)
    theta_hist = np.array(theta_hist)
    f_hist = f(theta_hist)

    # (c)(i): f(θ_n) over iterations
    axes1[i].plot(range(n+1), f_hist)
    axes1[i].set_title(f'η = {e}, θ_N = {theta_hist[-1]:.4f}')
    axes1[i].set_xlabel('Iteration n')
    axes1[i].set_ylabel('f(θ_n)')
    axes1[i].grid(True)

    # (c)(ii): f(θ) with red dot at final θ_N
    axes2[i].plot(thetas, fs, label='f(θ)')
    axes2[i].scatter(theta_hist[-1], f_hist[-1], color='red', s=40, label='θ_N'
    axes2[i].set_title(f'η = {e} ; θ_N = {theta_hist[-1]:.4f}')
    axes2[i].set_xlabel('θ')
    axes2[i].set_ylabel('f(θ)')
    axes2[i].legend()
    axes2[i].grid(True)

# Final layout adjustments
fig1.suptitle('f(θ_n) over Iterations', fontsize=12)
fig1.tight_layout(pad=2.0)
fig2.suptitle('Final θ_N Overlay on f(θ)', fontsize=12)
fig2.tight_layout(pad=4.0)

plt.tight_layout()
plt.show()
```

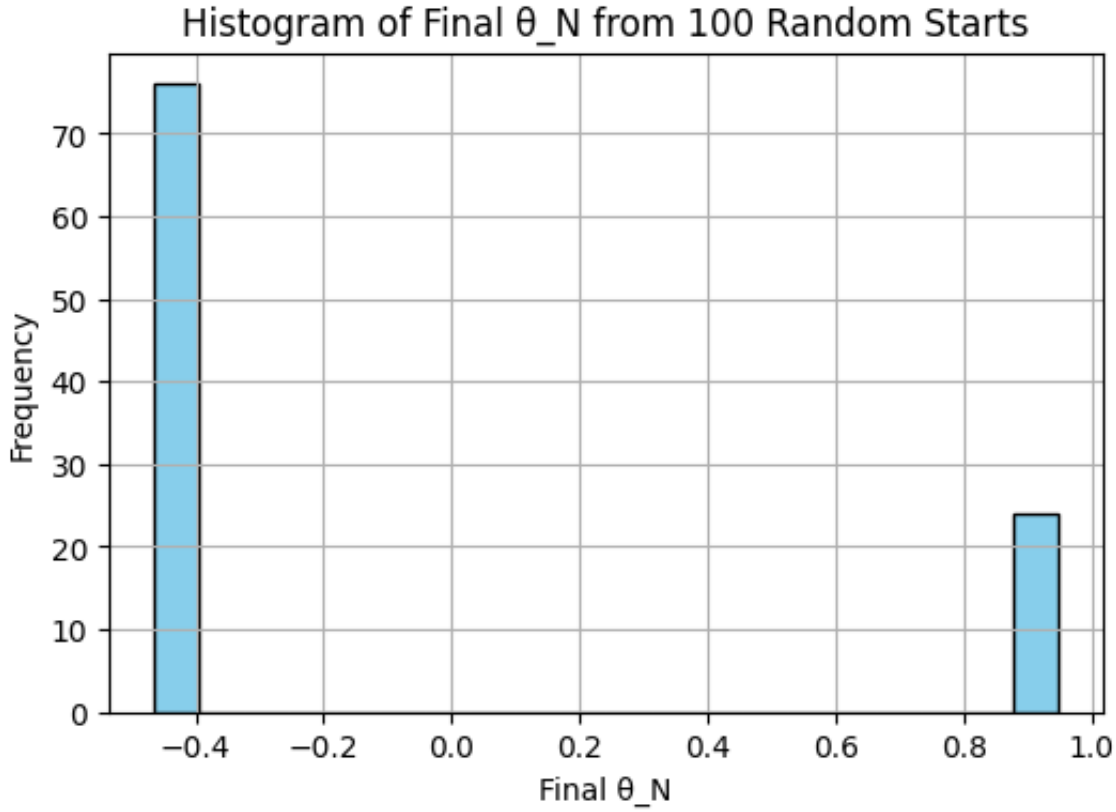
```
<ipython-input-2-b89c2b29c4ad>:5: RuntimeWarning: overflow encountered in s
  return 10*theta**9 - 3*theta**2 - 5*theta**4 + np.cos(theta)
<ipython-input-2-b89c2b29c4ad>:5: RuntimeWarning: invalid value encountered
  return 10*theta**9 - 3*theta**2 - 5*theta**4 + np.cos(theta)
<ipython-input-2-b89c2b29c4ad>:2: RuntimeWarning: overflow encountered in p
  return theta**10 - theta**3 - theta**5 + np.sin(theta)
<ipython-input-2-b89c2b29c4ad>:2: RuntimeWarning: invalid value encountered
  return theta**10 - theta**3 - theta**5 + np.sin(theta)
```


f(θ_n) over Iterations
η = 1e-06, θ_N = -0.0002
η = 0.0001, θ_N = -0.0199
η = 0.01, θ_N = -0.4652
η = 0.1, θ_N = -0.4652
η = 0.5, θ_N = -0.5926
1e119
η = 1, θ_N = nan
f(θ_n)
Iteration n


Final θ_N Overlay on f(θ)
η = 1e-06 ; θ_N = -0.0002
η = 0.0001 ; θ_N = -0.0199
η = 0.01 ; θ_N = -0.4652
η = 0.1 ; θ_N = -0.4652
η = 0.5 ; θ_N = -0.5926
η = 1 ; θ_N = nan
f(θ)
θ_N
θ

```
B = 100
eta = 0.01
theta_final_list = []

np.random.seed(42)  # for reproducibility

for _ in range(B):
    theta = np.random.uniform(-1, 1)
    for _ in range(n):
        theta = theta - eta * f_prime(theta)
    theta_final_list.append(theta)

# Histogram of final θ_N
plt.figure(figsize=(6, 4))
plt.hist(theta_final_list, bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Final θ_N')
plt.ylabel('Frequency')
plt.title(f'Histogram of Final θ_N from {B} Random Starts')
plt.grid(True)
plt.show()

# Estimate convergence to global min
```

```
theta_finals = np.array(theta_final_list)
global_min = thetas[np.argmin(fs)]
converged = np.abs(theta_finals - global_min) < 0.05  # tolerance
proportion_converged = np.mean(converged)
print(f"Proportion of runs not converging near global minimum: {1-proportion_cc
```




```
Proportion of runs not converging near global minimum: 0.24
```

```
!pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.7-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/d
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pyt
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/di
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-p
Downloading ucimlrepo-0.0.7-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.7
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
breast_cancer = fetch_ucirepo(id=17)



X = breast_cancer.data.features
y = breast_cancer.data.targets

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=2, stratify=y)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
y_train_encoded = y_train['Diagnosis'].map({'M': 1, 'B': 0}).values
y_test_encoded = y_test['Diagnosis'].map({'M': 1, 'B': 0}).values

y_train_tensor = torch.tensor(y_train_encoded, dtype=torch.float32).unsqueeze(1
y_test_tensor = torch.tensor(y_test_encoded, dtype=torch.float32).unsqueeze(1)


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(30, 10),
            nn.ReLU(),
            nn.Linear(10, 10),
            nn.ReLU(),
            nn.Linear(10, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)

model = Net()

loss_fn = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 1000
for epoch in range(num_epochs):
```

```
    model.train()
    y_pred = model(X_train_tensor)
    loss = loss_fn(y_pred, y_train_tensor)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 100 == 0:
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {loss.item():.4f}")

model.eval()
with torch.no_grad():
    y_test_probs = model(X_test_tensor)
    y_test_labels = (y_test_probs > 0.5).float()
    nn_acc = accuracy_score(y_test_tensor, y_test_labels)
    print(f"Neural Network Accuracy with MSELoss: {nn_acc:.4f}")
```

```
Epoch 100/1000, Loss: 0.0595
Epoch 200/1000, Loss: 0.0209
Epoch 300/1000, Loss: 0.0122
Epoch 400/1000, Loss: 0.0086
Epoch 500/1000, Loss: 0.0065
Epoch 600/1000, Loss: 0.0052
Epoch 700/1000, Loss: 0.0044
Epoch 800/1000, Loss: 0.0037
Epoch 900/1000, Loss: 0.0033
Epoch 1000/1000, Loss: 0.0030
Neural Network Accuracy with MSELoss: 0.9649
```

```
y_train_1d = y_train_tensor.numpy().ravel()
y_test_1d = y_test_tensor.numpy().ravel()

# Logistic Regression
lr_model = LogisticRegression(max_iter=2000, solver='liblinear')
lr_model.fit(X_train, y_train_1d)
lr_acc = accuracy_score(y_test_1d, lr_model.predict(X_test))

# Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=1)
rf_model.fit(X_train, y_train_1d)
rf_acc = accuracy_score(y_test_1d, rf_model.predict(X_test))
```

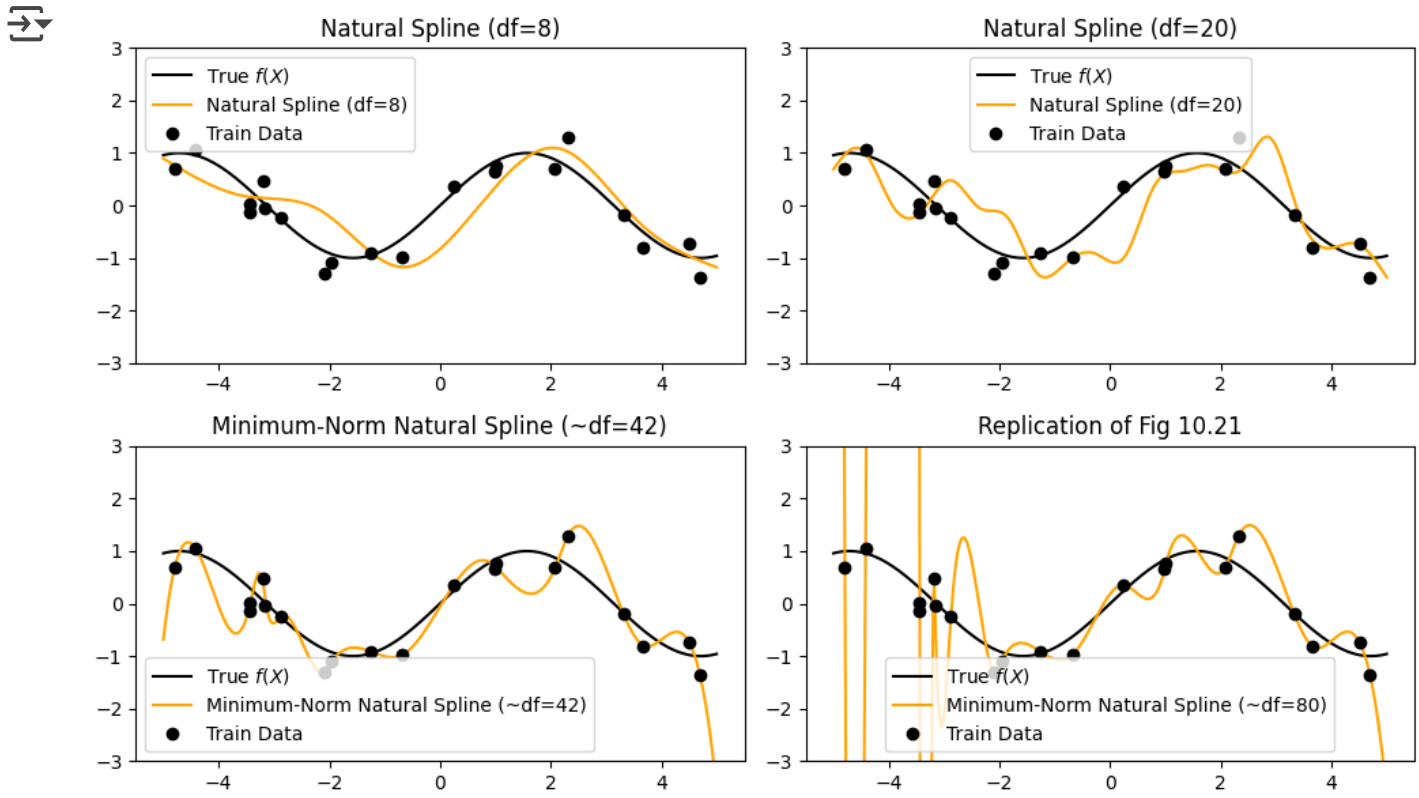
```
print("Model Performance on Test Set:")
print(f"Neural Network         : {nn_acc:.4f}")
print(f"Logistic Regression    : {lr_acc:.4f}")
print(f"Random Forest          : {rf_acc:.4f}")
```

```
Model Performance on Test Set:
Neural Network         : 0.9649
Logistic Regression    : 0.9357
Random Forest          : 0.9474
```

Extra Credit

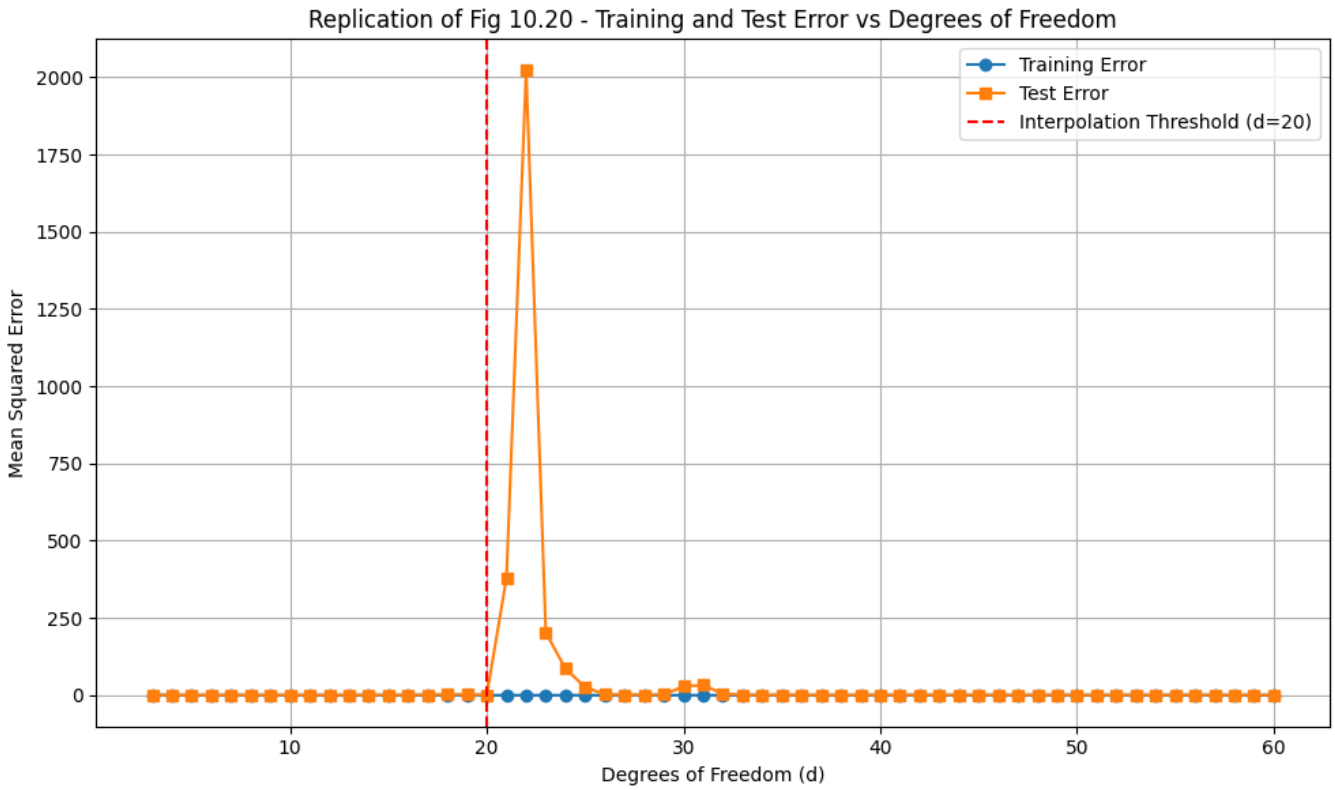
```
np.random.seed(42)
n_train = 20
X_train = np.sort(np.random.uniform(-5, 5, n_train))
y_train = np.sin(X_train) + np.random.normal(0, 0.3, n_train)

def f_true(x):
    return np.sin(x)

x_grid = np.linspace(-5, 5, 500)
y_true = f_true(x_grid)

df_list = [8, 20, 42, 80]
smoothing_dict = {42: 0.1, 80: 1e-8}  # Small smoothing → high flexibility

fig, axes = plt.subplots(2, 2, figsize=(10, 6))
axes = axes.flatten()

for i, df in enumerate(df_list):
    ax = axes[i]

    if df in [8, 20]:
        X_spline = dmatrix(f"cr(x, df={df})", {"x": X_train}, return_type='dataf
        model = LinearRegression().fit(X_spline, y_train)

        X_grid_spline = dmatrix(f"cr(x, df={df})", {"x": x_grid}, return_type='d
        y_pred = model.predict(X_grid_spline)

        method_label = f"Natural Spline (df={df})"
    else:
        s_val = smoothing_dict[df]
        spline = UnivariateSpline(X_train, y_train, s=s_val)
        y_pred = spline(x_grid)

        method_label = f"Minimum-Norm Natural Spline (~df={df})"
```

```
    ax.plot(x_grid, y_true, 'k-', label="True $f(X)$")  # True function
    ax.plot(x_grid, y_pred, color='orange', label=method_label)  # Spline fit
    ax.plot(X_train, y_train, 'ko', label="Train Data")  # Observations
    ax.set_title(method_label)
    ax.set_ylim(-3, 3)
    ax.legend()

plt.tight_layout()
plt.show()
```




```
X_test = np.sort(np.random.uniform(-5, 5, n_test))
y_test = np.sin(X_test) + np.random.normal(0, 0.3, n_test)
```

```
train_errors = []
test_errors = []
df_values = range(3, 61)
interp_threshold = None

for df in df_values:
    X_spline_train = dmatrix(f"cr(x, df={df})", {"x": X_train}, return_type='da
    X_spline_test = dmatrix(f"cr(x, df={df})", {"x": X_test}, return_type='data
    model = LinearRegression().fit(X_spline_train, y_train)
    y_train_pred = model.predict(X_spline_train)
    y_test_pred = model.predict(X_spline_test)
    train_mse = mean_squared_error(y_train, y_train_pred)
    test_mse = mean_squared_error(y_test, y_test_pred)
    train_errors.append(train_mse)
    test_errors.append(test_mse)

    if interp_threshold is None and train_mse < 1e-6:
        interp_threshold = df

plt.figure(figsize=(10, 6))
plt.plot(df_values, train_errors, label="Training Error", marker='o')
plt.plot(df_values, test_errors, label="Test Error", marker='s')
if interp_threshold:
    plt.axvline(x=interp_threshold, color='red', linestyle='--', label=f'Interp

plt.xlabel("Degrees of Freedom (d)")
plt.ylabel("Mean Squared Error")
plt.title("Replication of Fig 10.20 - Training and Test Error vs Degrees of Fre
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.