Problem 1: Bias-Variance Tradeoff and Regularization

In this problem, we will explore the fundamental concepts of the bias-variance tradeoff and demonstrate how regularization affects model performance.

Part 1: Deriving the Bias-Variance Decomposition [15 points]

Task: Show mathematically that the expected test error can be decomposed as:

$$E[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

where:

- $y = f(x) + \epsilon$ is the true target with noise $\epsilon \sim N(0, \sigma^2)$
- $\hat{f}(x)$ is our estimated function
- $\text{Bias} = E[\hat{f}(x)] - f(x)$
- $\text{Variance} = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$

Derivation

Let x be a fixed test point. Assume the data-generating model

$$y = f(x) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

and $\hat{f}(x)$ is a predictor learned from a random training dataset D. We derive a decomposition of the expected squared test error at x: $\mathbb{E}\big[(y - \hat{f}(x))^2\big]$.

$$\begin{aligned}
\mathbb{E}\big[(y - \hat{f}(x))^2\big] &= \mathbb{E}\Big[\big(f(x) + \varepsilon - \hat{f}(x)\big)^2\Big] \\
&= \mathbb{E}\Big[\big((f(x) - \hat{f}(x)) + \varepsilon\big)^2\Big] \\
&= \mathbb{E}\Big[(f(x) - \hat{f}(x))^2 + 2\varepsilon(f(x) - \hat{f}(x)) + \varepsilon^2\Big] \\
&= \mathbb{E}\big[(f(x) - \hat{f}(x))^2\big] + 2\mathbb{E}\big[\varepsilon(f(x) - \hat{f}(x))\big] + \mathbb{E}[\varepsilon^2].
\end{aligned}$$

We handle the cross term using iterated expectation w.r.t. the training data D (equivalently w.r.t. $\hat{f}(x)$) and the independence of ε from D:

$$\begin{aligned}\mathbb{E}\big[\varepsilon(f(x)-\hat{f}\,(x))\big] &= \mathbb{E}_D\Big[\mathbb{E}_\varepsilon\big[\varepsilon(f(x)-\hat{f}\,(x)) \mid D\big]\Big] \\ &= \mathbb{E}_D\Big[(f(x)-\hat{f}\,(x))\,\mathbb{E}_\varepsilon[\varepsilon \mid D]\Big] \\ &= \mathbb{E}_D\Big[(f(x)-\hat{f}\,(x))\,\mathbb{E}_\varepsilon[\varepsilon]\Big] \\ &= \mathbb{E}_D\Big[(f(x)-\hat{f}\,(x))\cdot 0\Big] \\ &= 0.\end{aligned}$$

Also,

$$\mathbb{E}[\varepsilon^2] = \mathrm{Var}(\varepsilon) + (\mathbb{E}[\varepsilon])^2 = \sigma^2 + 0 = \sigma^2.$$

Hence

$$\mathbb{E}\big[(y-\hat{f}\,(x))^2\big] = \mathbb{E}\big[(f(x)-\hat{f}\,(x))^2\big] + \sigma^2.$$

Now define the mean prediction (over training sets)

$$\mu(x) := \mathbb{E}[\hat{f}\,(x)].$$

Add and subtract $\mu(x)$ inside the square:

$$\begin{aligned}\mathbb{E}\big[(f(x)-\hat{f}\,(x))^2\big] &= \mathbb{E}\Big[\big(f(x)-\mu(x)+\mu(x)-\hat{f}\,(x)\big)^2\Big] \\ &= \mathbb{E}\Big[(f(x)-\mu(x))^2 + 2(f(x)-\mu(x))(\mu(x)-\hat{f}\,(x)) + (\mu(\\ &= (f(x)-\mu(x))^2 + 2(f(x)-\mu(x))\,\mathbb{E}[\mu(x)-\hat{f}\,(x)] + \mathbb{E}[(\mu \\ &= (f(x)-\mu(x))^2 + 2(f(x)-\mu(x))\big(\mu(x)-\mathbb{E}[\hat{f}\,(x)]\big) + \mathbb{E}[\\ &= (f(x)-\mu(x))^2 + 2(f(x)-\mu(x))(\mu(x)-\mu(x)) + \mathbb{E}[(\hat{f}\,(\\ &= (f(x)-\mu(x))^2 + \mathbb{E}[(\hat{f}\,(x)-\mu(x))^2].\end{aligned}$$

Using the given definitions,

$$\mathrm{Bias}(x) = \mathbb{E}[\hat{f}\,(x)] - f(x) = \mu(x) - f(x) \quad \Rightarrow \quad (f(x)-\mu(x))^2 = (\mu(x)-f($$

and

$$\mathrm{Var}(x) = \mathbb{E}\Big[(\hat{f}\,(x)-\mathbb{E}[\hat{f}\,(x)])^2\Big] = \mathbb{E}[(\hat{f}\,(x)-\mu(x))^2].$$

Therefore,

$$\mathbb{E}\big[(f(x)-\hat{f}\,(x))^2\big] = \mathrm{Bias}(x)^2 + \mathrm{Var}(x).$$

Substituting back,

$$\mathbb{E}\big[(y - \hat{f}(x))^2\big] = \Big(\mathrm{Bias}(x)^2 + \mathrm{Var}(x)\Big) + \sigma^2$$
$$= \mathrm{Bias}(x)^2 + \mathrm{Var}(x) + \sigma^2.$$

Thus,

$$\mathbb{E}\big[(y - \hat{f}(x))^2\big] = \mathrm{Bias}(x)^2 + \mathrm{Variance}(x) + \mathrm{Irreducible\ Noise}$$

Part 2: Dataset Creation and Visualization [10 points]

Task: Create a dataset using the following specifications:

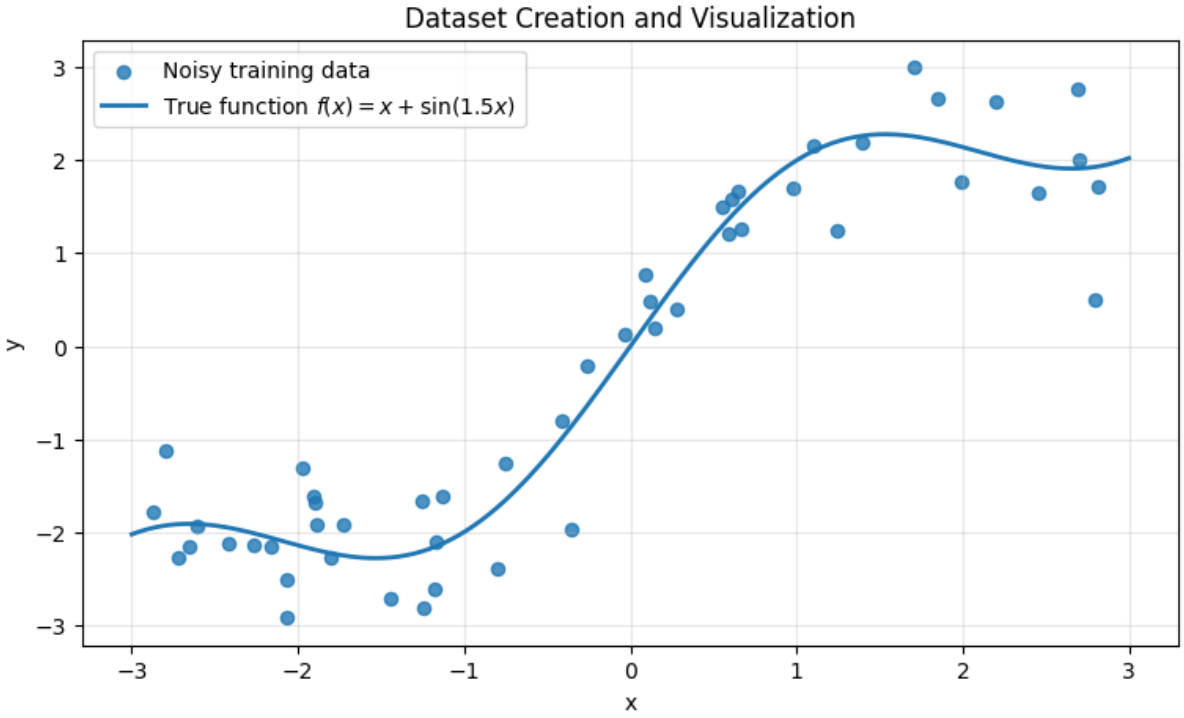
- True function: $f(x) = x + \sin(1.5x)$
- Add Gaussian noise with variance $\sigma^2 = 0.3$
- Generate 50 training samples with $x \in [-3, 3]$

Then create a visualization showing:

1. The noisy training data points
2. The true underlying function $f(x)$

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)

# TODO: Define the true function f(x)
def f(x):
    return x + np.sin(1.5 * x)

# TODO: Define function y(x) that adds noise to f(x)
def y(x, noise_var):
    noise_std = np.sqrt(noise_var)
    return f(x) + np.random.normal(loc=0.0, scale=noise_std, size=x.sh

# TODO: Set parameters
n_samples = 50  # number of training samples
x_range = (-3, 3)    # tuple (min, max) for x values
noise_var = 0.3  # noise variance σ²

# TODO: Generate training data
X_train = np.random.uniform(x_range[0], x_range[1], n_samples)
y_train = y(X_train, noise_var)

# TODO: Create visualization
x_plot = np.linspace(x_range[0], x_range[1], 400)
```

```
y_true = f(x_plot)

plt.figure(figsize=(9, 5))
plt.scatter(X_train, y_train, s=35, alpha=0.8, label="Noisy training d
plt.plot(x_plot, y_true, linewidth=2, label=r"True function $f(x)=x+\s
plt.xlabel("x")
plt.ylabel("y")
plt.title("Dataset Creation and Visualization")
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()
```



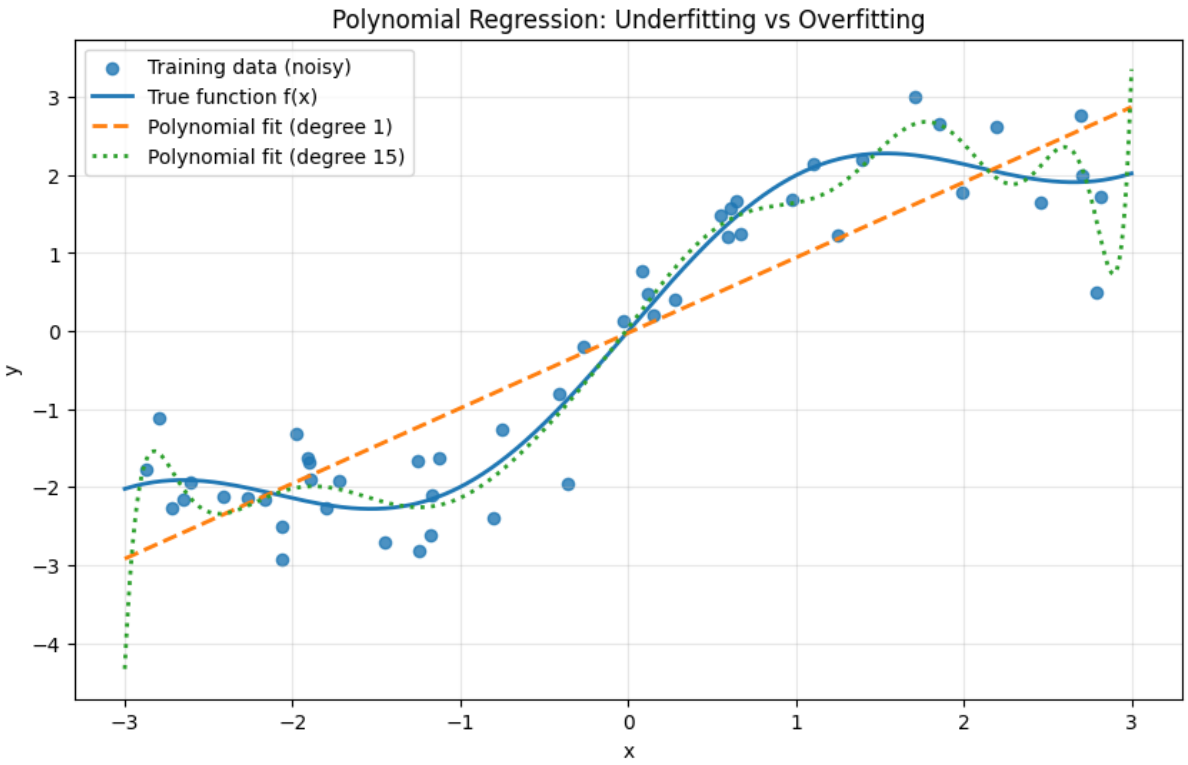
Part 3: Polynomial Fitting - Underfitting vs Overfitting [15 points]

Task: Demonstrate underfitting and overfitting using polynomial regression:

1. Fit a **low-degree polynomial** (e.g., degree 1) to show underfitting
2. Fit a **high-degree polynomial** (e.g., degree 15) to show overfitting
3. Visualize both fits along with the true function and training data

In [11]:
```
# TODO: Fit polynomials of different degrees
low_degree = 1   # underfitting
high_degree = 15  # overfitting

# TODO: Fit the polynomials using np.polyfit
coef_low = np.polyfit(X_train, y_train, deg=low_degree)
```

```
coef_high = np.polyfit(X_train, y_train, deg=high_degree)

poly_low = np.poly1d(coef_low)
poly_high = np.poly1d(coef_high)

x_plot = np.linspace(x_range[0], x_range[1], 600)

# Predictions
y_true = f(x_plot)
y_low = poly_low(x_plot)
y_high = poly_high(x_plot)

# TODO: Create visualization comparing underfitting vs overfitting
plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, s=35, alpha=0.8, label="Training data (n
plt.plot(x_plot, y_true, linewidth=2, label="True function f(x)")
plt.plot(x_plot, y_low, linewidth=2, linestyle="--", label=f"Polynomia
plt.plot(x_plot, y_high, linewidth=2, linestyle=":", label=f"Polynomia

plt.xlabel("x")
plt.ylabel("y")
plt.title("Polynomial Regression: Underfitting vs Overfitting")
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()
```



Part 4: Bias-Variance Tradeoff Analysis [15

points]

Task: Empirically demonstrate the bias-variance tradeoff:

1. For polynomial degrees 1 to 15, fit models on multiple different datasets (at least 100 datasets)
2. Compute the Bias², Variance, and MSE for each degree
3. Create a plot showing how these quantities change with model complexity
4. Identify the optimal model complexity

In [12]:
```
# TODO: Set up parameters for analysis
n_datasets = 100  # number of different datasets to generate
max_degree = 15   # maximum polynomial degree to test
n_test = 200      # number of test points

# TODO: Generate fixed test points
X_test = np.linspace(x_range[0], x_range[1], n_test)
y_test_true = f(X_test)

# Store predictions: shape (max_degree, n_datasets, n_test)
preds = np.zeros((max_degree, n_datasets, n_test))

# TODO: For each dataset, fit polynomials of each degree and store pre
rng = np.random.default_rng(42)

for d in range(n_datasets):
    # Generate a fresh noisy training dataset
    X_train = rng.uniform(x_range[0], x_range[1], n_samples)
    y_train = f(X_train) + rng.normal(0.0, np.sqrt(noise_var), size=X_

    for deg in range(1, max_degree + 1):
        coefs = np.polyfit(X_train, y_train, deg=deg)
        model = np.poly1d(coefs)
        preds[deg - 1, d, :] = model(X_test)
```

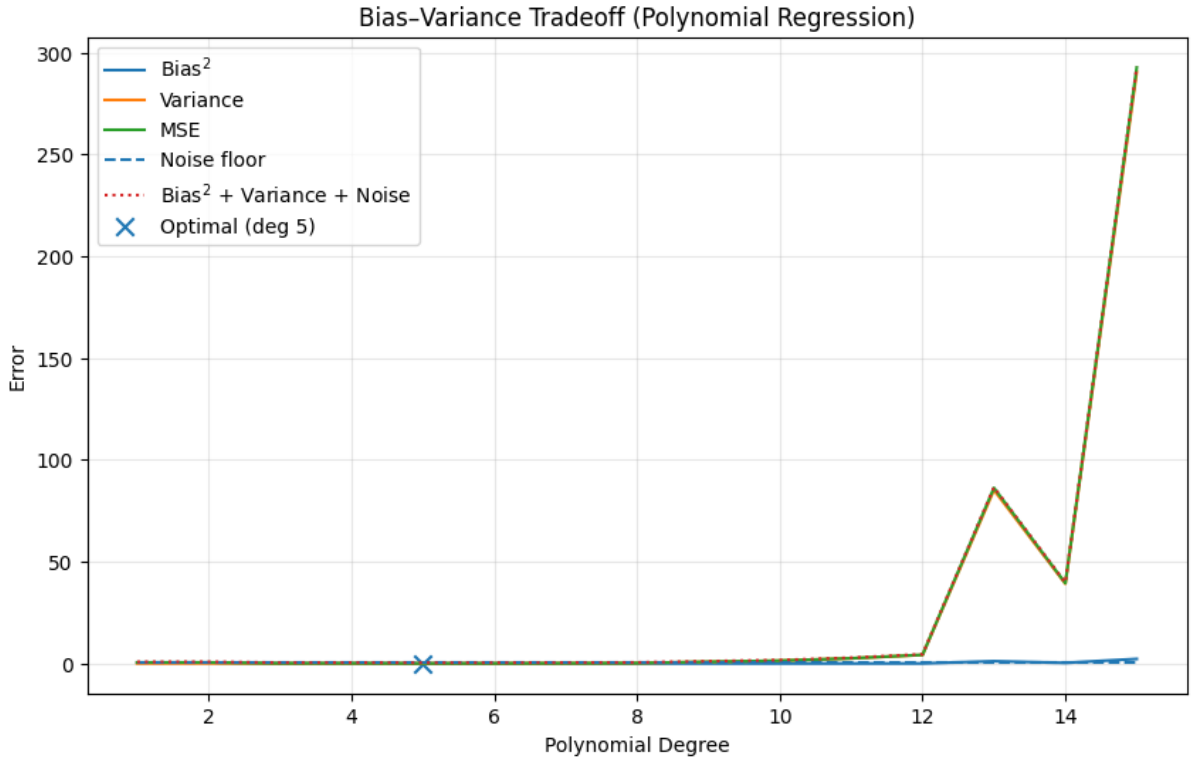
In [13]:
```
# TODO: Compute Bias², Variance, and MSE for each polynomial degree
bias2 = np.zeros(max_degree)
var = np.zeros(max_degree)
mse = np.zeros(max_degree)

for deg in range(1, max_degree + 1):
    Y_hat = preds[deg - 1]                 # (n_datasets, n_test)
    mean_pred = np.mean(Y_hat, axis=0)     # (n_test,)

    bias2[deg - 1] = np.mean((mean_pred - y_test_true) ** 2)
    var[deg - 1] = np.mean(np.var(Y_hat, axis=0, ddof=0))
    mse[deg - 1] = np.mean((Y_hat - y_test_true) ** 2)  # E[(f_hat - f

# Theoretical noise floor (irreducible error)
noise_floor = noise_var
```

```
decomp = bias2 + var + noise_floor

degrees = np.arange(1, max_degree + 1)
opt_deg = degrees[np.argmin(mse)]

# TODO: Print results in a table format
print(f"{'Degree':>6} | {'Bias^2':>10} | {'Variance':>10} | {'MSE':>10
print("-" * 65)
for i, deg in enumerate(degrees):
    print(f"{deg:6d} | {bias2[i]:10.5f} | {var[i]:10.5f} | {mse[i]:10.

print(f"\nOptimal degree (min MSE): {opt_deg}")
```

```
Degree |     Bias^2 |   Variance |        MSE |  Bias^2+Var+Noise
-----------------------------------------------------------------
     1 |    0.47947 |    0.03135 |    0.51081 |            0.81081
     2 |    0.47961 |    0.05172 |    0.53133 |            0.83133
     3 |    0.07068 |    0.03882 |    0.10951 |            0.40951
     4 |    0.07153 |    0.06110 |    0.13263 |            0.43263
     5 |    0.00190 |    0.05442 |    0.05633 |            0.35633
     6 |    0.00197 |    0.07229 |    0.07426 |            0.37426
     7 |    0.00239 |    0.12170 |    0.12409 |            0.42409
     8 |    0.00183 |    0.21219 |    0.21402 |            0.51402
     9 |    0.00182 |    0.80723 |    0.80905 |            1.10905
    10 |    0.01183 |    1.35876 |    1.37058 |            1.67058
    11 |    0.01201 |    2.51435 |    2.52636 |            2.82636
    12 |    0.03115 |    4.21165 |    4.24280 |            4.54280
    13 |    1.13111 |   85.13001 |   86.26112 |           86.56112
    14 |    0.31219 |   39.15747 |   39.46966 |           39.76966
    15 |    2.24087 |  290.45243 |  292.69330 |          292.99330

Optimal degree (min MSE): 5
```

In [14]:

```
# TODO: Create the bias-variance tradeoff plot
# Show: Bias², Variance, MSE, Noise floor, and Bias²+Variance+Noise
# Mark the optimal model (minimum MSE)
plt.figure(figsize=(10, 6))
plt.plot(degrees, bias2, label=r"Bias$^2$")
plt.plot(degrees, var, label="Variance")
plt.plot(degrees, mse, label="MSE")
plt.hlines(noise_floor, xmin=1, xmax=max_degree, linestyles="--", labe
plt.plot(degrees, decomp, linestyle=":", label=r"Bias$^2$ + Variance +

# Mark the optimal model
plt.scatter([opt_deg], [mse[opt_deg - 1]], s=80, marker="x", label=f"O

plt.xlabel("Polynomial Degree")
plt.ylabel("Error")
plt.title("Bias–Variance Tradeoff (Polynomial Regression)")
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()
```



Part 4: Analysis - Best Model Identification

Can you identify the best model?

The best model is the polynomial of degree 5, because it achieves the lowest test MSE in the bias-variance analysis. For degrees lower than 5, the model underfits (high bias), while for degrees higher than 5 the variance grows rapidly (overfitting), which increases the MSE. Degree 5 provides the best balance between bias and variance on this dataset.

Part 5: L2 Regularization (Ridge Regression) [10 points]

Task: Apply L2 regularization to the degree-10 polynomial and compare with the unregularized version.

In [15]:
```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures

# TODO: Set parameters
degree = 10
alpha = 1.0  # Regularization strength (lambda)
```

```
n_datasets = 100
n_test = 200

# Fixed test set
X_test = np.linspace(x_range[0], x_range[1], n_test)
y_test_true = f(X_test)

# Precompute polynomial feature transformer
poly = PolynomialFeatures(degree=degree, include_bias=True)

# Store predictions across datasets
preds_unreg = np.zeros((n_datasets, n_test))
preds_ridge = np.zeros((n_datasets, n_test))

rng = np.random.default_rng(42)

# TODO: Fit unregularized and regularized polynomials on multiple data
for d in range(n_datasets):
    X_train = rng.uniform(x_range[0], x_range[1], n_samples)
    y_train = f(X_train) + rng.normal(0.0, np.sqrt(noise_var), size=X_

    # Shape for sklearn: (n, 1)
    Xtr = X_train.reshape(-1, 1)
    Xte = X_test.reshape(-1, 1)

    Phi_tr = poly.fit_transform(Xtr)
    Phi_te = poly.transform(Xte)

    # Unregularized degree-10 polynomial via least squares on polynomi
    w_unreg, *_ = np.linalg.lstsq(Phi_tr, y_train, rcond=None)
    preds_unreg[d, :] = Phi_te @ w_unreg

    # Ridge-regularized polynomial
    ridge = Ridge(alpha=alpha, fit_intercept=False)  # intercept alrea
    ridge.fit(Phi_tr, y_train)
    preds_ridge[d, :] = ridge.predict(Phi_te)
```

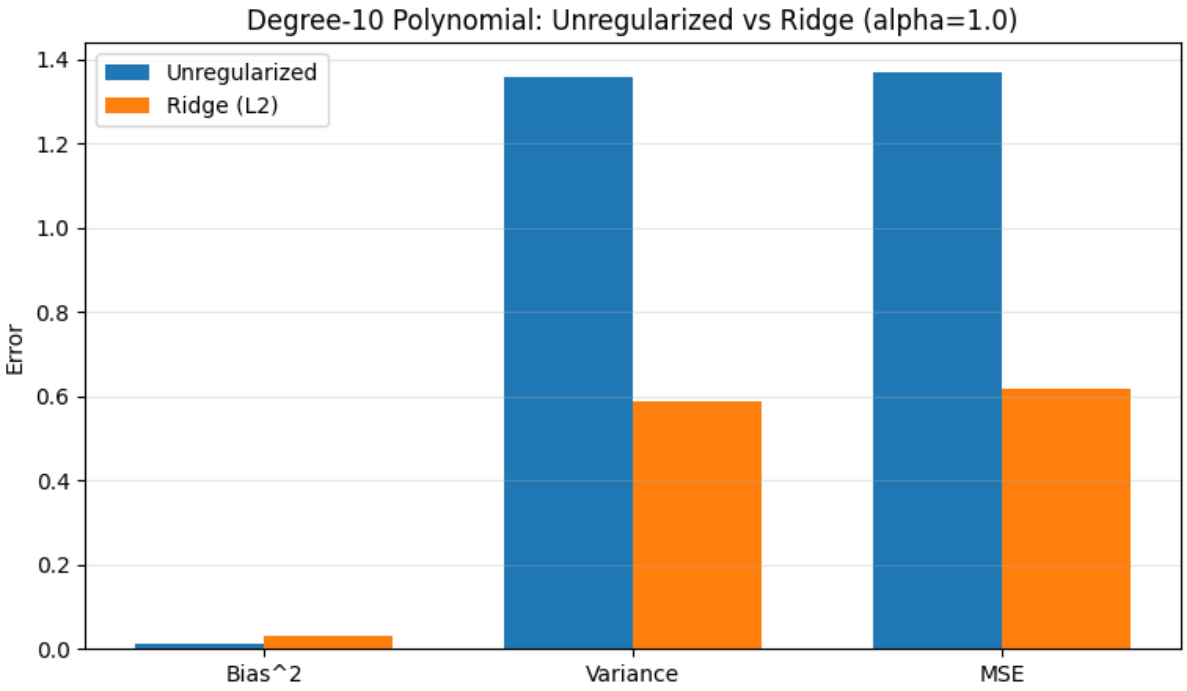
In [16]:

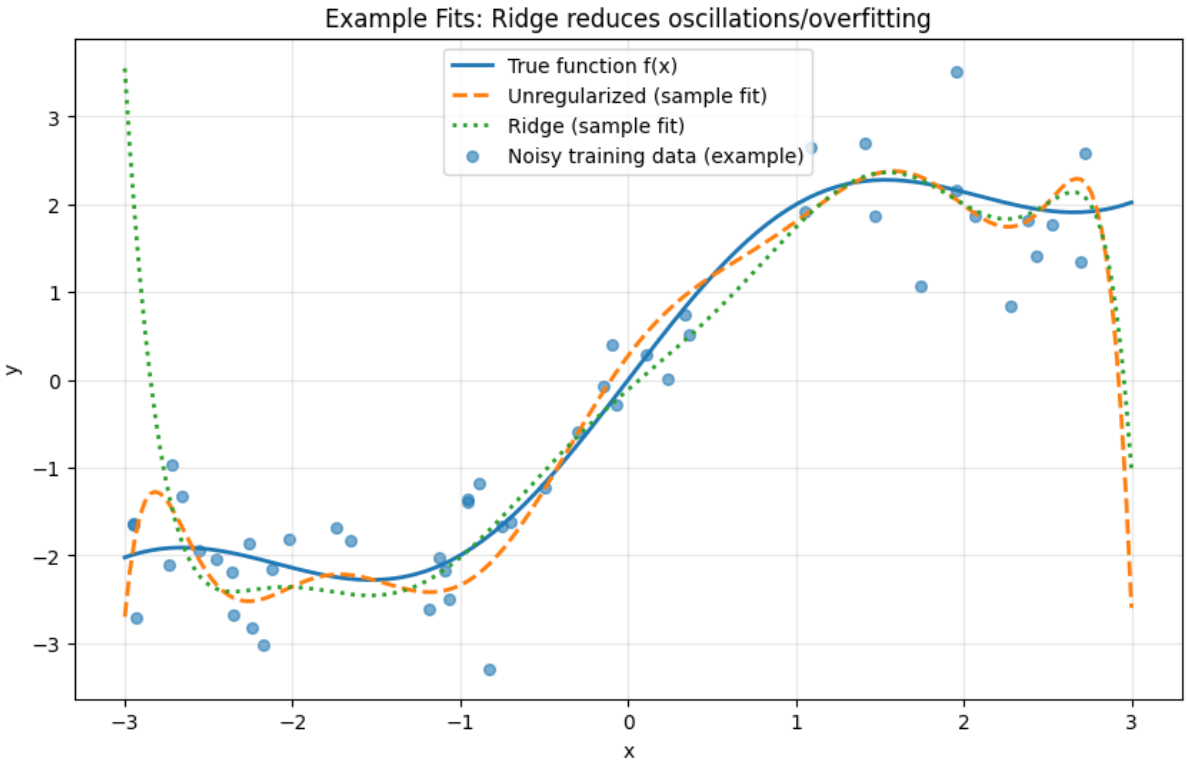
```
# TODO: Compute Bias², Variance, and MSE for both models
def bias_var_mse(preds, y_true):
    mean_pred = np.mean(preds, axis=0)
    bias2 = np.mean((mean_pred - y_true) ** 2)
    var = np.mean(np.var(preds, axis=0, ddof=0))
    mse = np.mean((preds - y_true) ** 2)  # E[(f_hat - f)^2]
    return bias2, var, mse

bias2_u, var_u, mse_u = bias_var_mse(preds_unreg, y_test_true)
bias2_r, var_r, mse_r = bias_var_mse(preds_ridge, y_test_true)

# TODO: Display comparison table
print(f"Degree = {degree}, alpha = {alpha}, datasets = {n_datasets}")
print(f"{'Model':<18} | {'Bias^2':>10} | {'Variance':>10} | {'MSE':>10
print("-" * 56)
```

```
print(f"{'Unregularized':<18} | {bias2_u:10.5f} | {var_u:10.5f} | {mse
print(f"{'Ridge (L2)':<18} | {bias2_r:10.5f} | {var_r:10.5f} | {mse_r:
```

```
Degree = 10, alpha = 1.0, datasets = 100
Model              |     Bias^2 |   Variance |        MSE
------------------------------------------------------------
Unregularized      |    0.01183 |    1.35876 |    1.37058
Ridge (L2)         |    0.02957 |    0.58788 |    0.61745
```

In [17]:
```
# TODO: Create visualization comparing regularized vs unregularized
# Show: bar plot comparing metrics and sample predictions
labels = ["Bias^2", "Variance", "MSE"]
unreg_vals = [bias2_u, var_u, mse_u]
ridge_vals = [bias2_r, var_r, mse_r]

x = np.arange(len(labels))
width = 0.35

plt.figure(figsize=(9, 5))
plt.bar(x - width/2, unreg_vals, width, label="Unregularized")
plt.bar(x + width/2, ridge_vals, width, label="Ridge (L2)")
plt.xticks(x, labels)
plt.ylabel("Error")
plt.title(f"Degree-{degree} Polynomial: Unregularized vs Ridge (alpha=
plt.grid(True, axis="y", alpha=0.3)
plt.legend()
plt.show()

sample_idx = 0
plt.figure(figsize=(10, 6))
plt.plot(X_test, y_test_true, linewidth=2, label="True function f(x)")
plt.plot(X_test, preds_unreg[sample_idx], linestyle="--", linewidth=2,
plt.plot(X_test, preds_ridge[sample_idx], linestyle=":", linewidth=2,

# also show that sample dataset's training points
X_train_s = rng.uniform(x_range[0], x_range[1], n_samples)
y_train_s = f(X_train_s) + rng.normal(0.0, np.sqrt(noise_var), size=X_
plt.scatter(X_train_s, y_train_s, s=30, alpha=0.6, label="Noisy traini

plt.xlabel("x")
plt.ylabel("y")
plt.title("Example Fits: Ridge reduces oscillations/overfitting")
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()
```





Part 5: Analysis - Regularization Effect

Does the regularized model have a higher or lower bias?

The regularized (Ridge) model has higher bias. As visible in the results, Bias² increases from 0.01183 (unregularized) to 0.02957 (Ridge).

What about MSE?

The regularized (Ridge) model has much lower MSE. MSE drops from 1.37058 (unregularized) to 0.61745 (Ridge).

Explain:

Ridge (L2) regularization shrinks the polynomial coefficients, which reduces model flexibility. This typically increases bias slightly (the model can't perfectly chase the true function). But it reduces variance by preventing the degree-10 polynomial from fitting noise and producing wild oscillations, as visible in the plots. Since the unregularized model's error is dominated by very high variance, the variance reduction from Ridge more than compensates for the small bias increase, so the overall MSE becomes much smaller.

Summary

In this problem, you will:

1. Derive the bias-variance decomposition:
   $E[MSE] = \text{Bias}^2 + \text{Variance} + \text{Noise}$
2. Visualize a noisy dataset and the true underlying function
3. Demonstrate underfitting (low-degree) and overfitting (high-degree) with polynomial regression
4. Quantify the bias-variance tradeoff across model complexities
5. Show how L2 regularization trades bias for variance to improve generalization