Exercise 1: Linear Regression is Gaussian Inference

Throughout, let $y \in \mathbb{R}^n$ be the response vector, $X \in \mathbb{R}^{n\times p}$ the design matrix (with full column rank), $\beta \in \mathbb{R}^p$ the coefficient vector, and $\varepsilon \sim \mathcal{N}(0, \sigma^2 I_n)$ the noise.

1. Equivalence of the two Gaussian statements

Claim.

$$y = f(x) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \Sigma) \quad \Longleftrightarrow \quad y \,|\, x \;\sim\; \mathcal{N}\big(f(x), \Sigma\big).$$

Answer:
"$\Longrightarrow$" Adding a deterministic vector to a Gaussian shifts its mean, leaving the covariance untouched: $y \mid x \;=\; f(x) + \varepsilon \sim \mathcal{N}\big(f(x), \Sigma\big)$.
"$\Longleftarrow$" Conversely, if $y \mid x$ is Gaussian with mean $f(x)$ and covariance Σ, define $\varepsilon := y - f(x)$. Then $\varepsilon \sim \mathcal{N}(0, \Sigma)$ and trivially $y = f(x) + \varepsilon$. $\square$

2. Log-likelihood versus residual-sum-of-squares

Assume the *classical linear model* $y = X\beta + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma^2 I_n)$. Its likelihood is

$$p(y \mid X, \beta, \sigma^2) = (2\pi\sigma^2)^{-n/2} \exp\Big[-\tfrac{1}{2\sigma^2}(y - X\beta)^\top (y - X\beta)\Big].$$

Taking logs gives

$$\ell(\beta) = \log p(y \mid X, \beta, \sigma^2) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\underbrace{\|y - X\beta\|_2^2}_{\mathrm{RSS}(\beta)}.$$

The first term is β-independent, so

$$\begin{aligned}\arg\max_\beta\ \ell(\beta) &= \arg\min_\beta\ \mathrm{RSS}(\beta) \\ &= \arg\min_\beta\ \tfrac{1}{n}\mathrm{RSS}(\beta) \quad \text{(MSE)}.\end{aligned}$$

Thus, *maximising the Gaussian log-likelihood is identical to ordinary least squares.*

3. Normal equations, MLE, and its covariance

3.1 Point estimator. Define $\mathrm{RSS}(\beta) = \|y - X\beta\|_2^2$. Gradient-setting yields the *normal equations*

$$\nabla_\beta \mathrm{RSS} = -2X^\top(y - X\beta) = 0 \implies X^\top X\,\beta = X^\top y.$$

Because $X^\top X$ is invertible (full rank assumption),

$$\boxed{\hat\beta_{\mathrm{MLE}} = (X^\top X)^{-1} X^\top y}.$$

3.2 Sampling distribution of $\hat{\beta}$. Insert the true model $y = X\beta + \varepsilon$:

$$\hat{\beta} = (X^\top X)^{-1} X^\top (X\beta + \varepsilon) = \beta + (X^\top X)^{-1} X^\top \varepsilon.$$

Since $E[\varepsilon] = 0$ and $\mathrm{Cov}(\varepsilon) = \sigma^2 I_n$,

$$\mathrm{Cov}(\hat{\beta}) = (X^\top X)^{-1} X^\top (\sigma^2 I_n) X \, (X^\top X)^{-1} = \sigma^2 (X^\top X)^{-1}.$$

Hence, each coefficient estimator is unbiased with

$$\boxed{\mathrm{Var}(\hat{\beta}_j) = \sigma^2 \left[(X^\top X)^{-1}\right]_{jj}}.$$

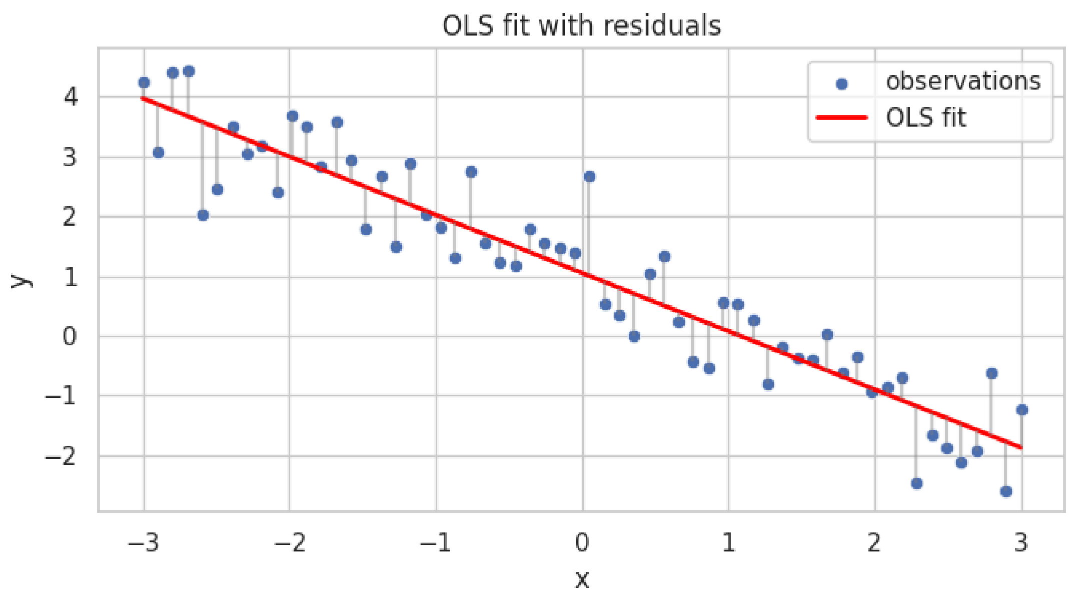
Remark. If σ^2 is unknown, replace it by the usual unbiased estimate $\hat{\sigma}^2 = \mathrm{RSS}(\hat{\beta})/(n-p)$ before computing standard errors—but that lies beyond the scope of this question.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns # For better aesthetics
from sklearn.linear_model import LinearRegression
from scipy.special import binom # Binomial coefficients for polynomial features

sns.set_theme(style="whitegrid")
```

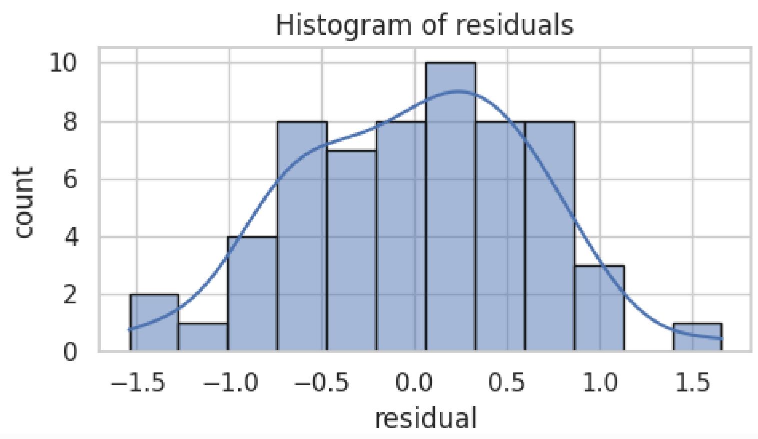
Exercise 1

Task 4

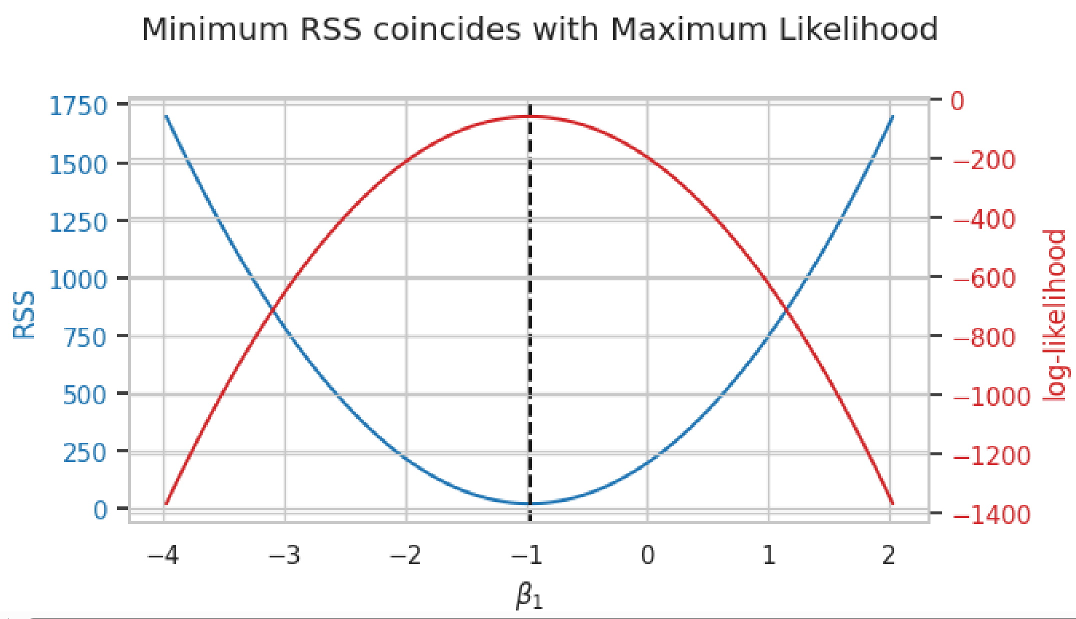
```
# Generate Synthetic data
n = 60
beta0, beta1, sigma = 1.0, -1.0, 0.8
x = np.linspace(-3, 3, n)

rng = np.random.default_rng(42)
eps = rng.normal(0, sigma, n)
y =  beta0 + beta1 * x + eps


# Fit OLS model using sklearn
X = x.reshape(-1, 1)
linreg = LinearRegression().fit(X, y)

y_hat = linreg.predict(X)
beta0_hat = linreg.intercept_
beta1_hat = linreg.coef_[0]


# Plot data, fitted line and vertical error (residual) bars
order = np.argsort(x)

plt.figure(figsize=(7, 4))
sns.scatterplot(x=x, y=y, label="observations")
plt.plot(x[order], y_hat[order], color="red", lw=2, label="OLS fit")

for xi, yi, ypred in zip(x, y, y_hat):
    plt.vlines(xi, min(yi, ypred), max(yi, ypred), color="grey", alpha=0.4)

plt.xlabel("x")
plt.ylabel("y")
plt.title("OLS fit with residuals")
plt.legend()
plt.tight_layout()
plt.show()
```



```
# Do a histogram of the residuals (e.g. sns.histplot)
residuals = y - y_hat

plt.figure(figsize=(5, 3))
sns.histplot(residuals, bins=12, kde=True, edgecolor='k')
plt.xlabel('residual'); plt.ylabel('count')
plt.title('Histogram of residuals')
plt.tight_layout(); plt.show()
```



```
# Plot the log likelihood and the RSS as a function of beta1. Show visually that MLE and OLS are equivalent for linear

beta1_grid = np.linspace(beta1_hat - 3, beta1_hat + 3, 200)
rss_list, ll_list = [], []

for b1 in beta1_grid:
    y_pred = beta0_hat + b1 * x
    res    = y - y_pred
    rss    = np.sum(res**2)
    rss_list.append(rss)
    ll     = -(n/2)*np.log(2*np.pi*sigma**2) - rss/(2*sigma**2)
    ll_list.append(ll)

fig, ax1 = plt.subplots(figsize=(7, 4))

ax1.plot(beta1_grid, rss_list, color='tab:blue', label='RSS')
ax1.set_xlabel(r'$\beta_1$')
ax1.set_ylabel('RSS', color='tab:blue')
ax1.tick_params(axis='y', labelcolor='tab:blue')

ax2 = ax1.twinx()
ax2.plot(beta1_grid, ll_list, color='tab:red', label='log-likelihood')
ax2.set_ylabel('log-likelihood', color='tab:red')
ax2.tick_params(axis='y', labelcolor='tab:red')

ax1.axvline(beta1_hat, ls='--', color='k')
fig.suptitle('Minimum RSS coincides with Maximum Likelihood')
fig.tight_layout(); plt.show()
```



MLE Sheet 8

Aarohi Verma

June 2025

Exercise 2: Bayesian Linear Regression and Basis Function Expansion

Task 1

We are given the following setup for Bayesian Linear Regression:

Prior

$$\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}_p)$$

Likelihood

$$\mathbf{y} \mid \boldsymbol{\beta}, \mathbf{X} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbf{I}_n)$$

Hint: Likelihood as a Gaussian in $\boldsymbol{\beta}$

By viewing the likelihood as a function of $\boldsymbol{\beta}$ (up to proportionality), it is proportional to:

$$p(\mathbf{y} \mid \boldsymbol{\beta}, \mathbf{X}) \propto \mathcal{N}\left(\boldsymbol{\beta}; \boldsymbol{\beta}_{\text{MLE}}, \sigma^2 (\mathbf{X}^\top \mathbf{X})^{-1}\right)$$

The posterior is proportional to the product of the prior and the likelihood:

$$p(\boldsymbol{\beta} \mid \mathbf{y}, \mathbf{X}) \propto \mathcal{N}(\boldsymbol{\beta}; \mathbf{0}, \tau^2 \mathbf{I}_p) \times \mathcal{N}\left(\boldsymbol{\beta}; \boldsymbol{\beta}_{\text{MLE}}, \sigma^2 (\mathbf{X}^\top \mathbf{X})^{-1}\right)$$

Product of Two Gaussians

Recall the result: if

$$\mathcal{N}(\boldsymbol{\beta}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \times \mathcal{N}(\boldsymbol{\beta}; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \propto \mathcal{N}(\boldsymbol{\beta}; \boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}})$$

then:

$$\boldsymbol{\Sigma}_{\text{post}} = \left(\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}\right)^{-1}$$

$$\boldsymbol{\mu}_{\text{post}} = \boldsymbol{\Sigma}_{\text{post}} \left(\boldsymbol{\Sigma}_1^{-1} \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1} \boldsymbol{\mu}_2\right)$$

Let:

$$\boldsymbol{\mu}_1 = \mathbf{0}, \quad \boldsymbol{\Sigma}_1 = \tau^2 \mathbf{I}_p$$
$$\boldsymbol{\mu}_2 = \boldsymbol{\beta}_{\text{MLE}}, \quad \boldsymbol{\Sigma}_2 = \sigma^2 (\mathbf{X}^\top \mathbf{X})^{-1}$$

Then:

Posterior Covariance:

$$\boldsymbol{\Sigma}_{\text{post}} = \left(\frac{1}{\tau^2} \mathbf{I}_p + \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X} \right)^{-1}$$

Posterior Mean:

$$\begin{aligned} \boldsymbol{\mu}_{\text{post}} &= \boldsymbol{\Sigma}_{\text{post}} \left(\frac{1}{\tau^2} \mathbf{0} + \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X} \boldsymbol{\beta}_{\text{MLE}} \right) \\ &= \boldsymbol{\Sigma}_{\text{post}} \cdot \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{y} \end{aligned}$$

Final Posterior Distribution

The posterior distribution of $\boldsymbol{\beta}$ given data is:

$$p(\boldsymbol{\beta} \mid \mathbf{y}, \mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}})$$

with:

$$\boxed{\boldsymbol{\Sigma}_{\text{post}} = \left(\frac{1}{\tau^2} \mathbf{I}_p + \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X} \right)^{-1}} \quad \text{and} \quad \boxed{\boldsymbol{\mu}_{\text{post}} = \boldsymbol{\Sigma}_{\text{post}} \cdot \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{y}}$$

Task 2

In a fully Bayesian model, we want to make predictions by integrating over the posterior distribution of the parameters. For a new input $\mathbf{x}_\star$, the predictive distribution is given by:

$$p(y_\star \mid \mathbf{x}_\star, \mathbf{X}, \mathbf{y}) = \int p(y_\star \mid \mathbf{x}_\star, \boldsymbol{\beta})\, p(\boldsymbol{\beta} \mid \mathbf{X}, \mathbf{y})\, d\boldsymbol{\beta}$$

This integral accounts for:

- **Aleatoric uncertainty** — the inherent noise in observations.
- **Epistemic uncertainty** — uncertainty about the model parameters.

Why is the Predictive Distribution Gaussian?

The likelihood is Gaussian:

$$y_\star \mid \mathbf{x}_\star, \boldsymbol{\beta} \sim \mathcal{N}(\mathbf{x}_\star^\top \boldsymbol{\beta}, \sigma^2)$$

The posterior distribution over the parameters is also Gaussian:

$$\boldsymbol{\beta} \mid \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}})$$

Since the predictive distribution involves a linear transformation of a Gaussian variable (plus independent Gaussian noise), the resulting marginal distribution is also Gaussian.

Predictive Mean

The predictive mean is the expectation of $y_\star$ under the posterior over $\boldsymbol{\beta}$:

$$m(\mathbf{x}_\star) = E_{p(\boldsymbol{\beta}|\mathbf{y})}\left[\mathbf{x}_\star^\top \boldsymbol{\beta}\right] = \mathbf{x}_\star^\top \boldsymbol{\mu}_{\text{post}}$$

Predictive Variance

We use the law of total variance:

$$v(\mathbf{x}_\star) = E_{\boldsymbol{\beta}}\left[\text{Var}(y_\star \mid \mathbf{x}_\star, \boldsymbol{\beta})\right] + \text{Var}_{\boldsymbol{\beta}}\left[E(y_\star \mid \mathbf{x}_\star, \boldsymbol{\beta})\right]$$

Compute the two terms:

$$\begin{aligned} \text{Var}(y_\star \mid \mathbf{x}_\star, \boldsymbol{\beta}) &= \sigma^2 \\ \text{Var}_{\boldsymbol{\beta}}(\mathbf{x}_\star^\top \boldsymbol{\beta}) &= \mathbf{x}_\star^\top \boldsymbol{\Sigma}_{\text{post}} \mathbf{x}_\star \end{aligned}$$

Therefore, the predictive variance is:

$$v(\mathbf{x}_\star) = \sigma^2 + \mathbf{x}_\star^\top \boldsymbol{\Sigma}_{\text{post}} \mathbf{x}_\star$$

Final Predictive Distribution

The predictive distribution is:

$$\boxed{p(y_\star \mid \mathbf{x}_\star, \mathbf{X}, \mathbf{y}) = \mathcal{N}\left(\mathbf{x}_\star^\top \boldsymbol{\mu}_{\text{post}},\ \sigma^2 + \mathbf{x}_\star^\top \boldsymbol{\Sigma}_{\text{post}} \mathbf{x}_\star\right)}$$

Posterior Parameters (from Previous Derivation)

$$\begin{aligned} \boldsymbol{\Sigma}_{\text{post}} &= \left(\frac{1}{\tau^2}\mathbf{I}_p + \frac{1}{\sigma^2}\mathbf{X}^\top \mathbf{X}\right)^{-1} \\ \boldsymbol{\mu}_{\text{post}} &= \boldsymbol{\Sigma}_{\text{post}} \cdot \frac{1}{\sigma^2}\mathbf{X}^\top \mathbf{y} \end{aligned}$$

Task 3

We extend Bayesian linear regression by applying a nonlinear basis function expansion to the input data using a feature map:

$$\boldsymbol{\phi} : R^n \to R^D, \quad x \mapsto \boldsymbol{\phi}(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_D(x) \end{bmatrix}$$

Our model becomes:

$$y = \boldsymbol{\phi}(x)^\top \mathbf{w} + \varepsilon, \quad \mathbf{w} \sim \mathcal{N}(0, \tau^2 \mathbf{I}_D), \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

(a) Why must the model remain linear in parameters?

Even though the model is nonlinear in the input x, it is crucial that the model remains **linear in the parameters w** so that:

- The **conjugacy** between the Gaussian prior and Gaussian likelihood is preserved.
- We retain closed-form expressions for the posterior and predictive distributions.
- The model remains computationally tractable, and Bayesian inference remains analytically solvable.

This strategy allows us to model nonlinear relationships while still using linear algebra tools.

(b) Bayesian Model in Expanded Feature Space

Let the transformed feature matrix be:

$$\Phi = \begin{bmatrix} \boldsymbol{\phi}(x_1)^\top \\ \boldsymbol{\phi}(x_2)^\top \\ \vdots \\ \boldsymbol{\phi}(x_n)^\top \end{bmatrix} \in R^{n \times D}$$

Likelihood:

$$p(\mathbf{y} \mid \Phi, \mathbf{w}) = \mathcal{N}(\Phi \mathbf{w}, \sigma^2 \mathbf{I}_n)$$

Prior:

$$\mathbf{w} \sim \mathcal{N}(0, \tau^2 \mathbf{I}_D)$$

Posterior:

$$\mathbf{w} \mid \Phi, \mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}})$$

where

$$\boldsymbol{\Sigma}_{\text{post}} = \left(\frac{1}{\tau^2}\mathbf{I}_D + \frac{1}{\sigma^2}\Phi^\top\Phi\right)^{-1}, \quad \boldsymbol{\mu}_{\text{post}} = \boldsymbol{\Sigma}_{\text{post}} \cdot \frac{1}{\sigma^2}\Phi^\top\mathbf{y}$$

Predictive Mean for New Input $x_\star$:

$$m(x_\star) = E[y_\star \mid x_\star, \mathbf{y}] = \boldsymbol{\phi}(x_\star)^\top\boldsymbol{\mu}_{\text{post}}$$

(c) Dependence on Inner Products

Using the posterior mean:

$$m(x_\star) = \boldsymbol{\phi}(x_\star)^\top\boldsymbol{\mu}_{\text{post}} = \frac{1}{\sigma^2}\boldsymbol{\phi}(x_\star)^\top\boldsymbol{\Sigma}_{\text{post}}\Phi^\top\mathbf{y}$$

Substitute $\boldsymbol{\Sigma}_{\text{post}}$:

$$m(x_\star) = \frac{1}{\sigma^2}\boldsymbol{\phi}(x_\star)^\top\left(\frac{1}{\tau^2}\mathbf{I}_D + \frac{1}{\sigma^2}\Phi^\top\Phi\right)^{-1}\Phi^\top\mathbf{y}$$

This expression only involves **inner products** of the form $\boldsymbol{\phi}(x_i)^\top\boldsymbol{\phi}(x_j)$ inside $\Phi^\top\Phi$, showing that the prediction depends only on those inner products.

(d) Applying the Kernel Trick

Define the **kernel matrix** $\mathbf{K} = \Phi\Phi^\top \in R^{n\times n}$, with entries:

$$K_{ij} = k(x_i, x_j) = \boldsymbol{\phi}(x_i)^\top\boldsymbol{\phi}(x_j)$$

Define the kernel vector:

$$\mathbf{k}_\star = \begin{bmatrix} k(x_1, x_\star) \\ k(x_2, x_\star) \\ \vdots \\ k(x_n, x_\star) \end{bmatrix} = \Phi\boldsymbol{\phi}(x_\star)$$

It can be shown that:

$$\frac{1}{\sigma^2}\boldsymbol{\phi}(x_\star)^\top\boldsymbol{\Sigma}_{\text{post}}\Phi^\top = \mathbf{k}_\star^\top\left(\mathbf{K} + \frac{\sigma^2}{\tau^2}\mathbf{I}_n\right)^{-1}$$

Therefore, the predictive mean becomes:

$$\boxed{m(x_\star) = \mathbf{k}_\star^\top\left(\mathbf{K} + \frac{\sigma^2}{\tau^2}\mathbf{I}_n\right)^{-1}\mathbf{y}}$$

This expression uses only the kernel $k(x, x')$, eliminating the need to compute $\boldsymbol{\phi}(x)$ explicitly.

Predictive Variance (Sketch): The predictive variance also becomes:

$$v(x_\star) = k(x_\star, x_\star) - \mathbf{k}_\star^\top \left(\mathbf{K} + \frac{\sigma^2}{\tau^2}\mathbf{I}_n\right)^{-1} \mathbf{k}_\star$$

(e) From Kernel Model to Gaussian Processes

The final conceptual step is to interpret the kernel function $k(x, x')$ as a **covariance function** of a stochastic process. That is:

$$f(x) \sim \mathcal{GP}(0, k(x, x'))$$

Key observations:

- Instead of placing a prior on parameters $\mathbf{w}$, we place a prior on functions $f(x)$.
- The kernel function defines the covariance between function values at different inputs.
- As the number of basis functions $D \to \infty$, the Bayesian linear model with basis expansion converges to a **Gaussian Process**.
- This makes GPs a fully non-parametric Bayesian model, where inference and prediction are performed entirely using the kernel function.

Machine Learning Essentials SS25 - Exercise Sheet 8

Instructions

- `TODO` 's indicate where you need to complete the implementations.
- You may use external resources, but **write your own solutions**.
- Provide concise, but comprehensible comments to explain what your code does.
- Code that's unnecessarily extensive and/or not well commented will not be scored.

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns # For better aesthetics
from sklearn.linear_model import LinearRegression
from scipy.special import binom # Binomial coefficients for polynomial featu
from scipy.spatial.distance import cdist

sns.set_theme(style="whitegrid")
```

Exercise 2

Task 4

• ALEATORIC UNCERTAINTY: σ²

- This is the inherent observation noise
- Represents irreducible uncertainty due to measurement noise
- Does NOT shrink as n → ∞ because it's a property of the data generation process
- Even with infinite data, we still have this baseline uncertainty

• EPISTEMIC UNCERTAINTY: ϕ*ᵀ Σ_post ϕ*

- This represents uncertainty about the model parameters
- Captures our lack of knowledge about the true parameter values
- DOES shrink as n → ∞ because more data reduces parameter uncertainty
- With infinite data, we would know the parameters perfectly

Task 5

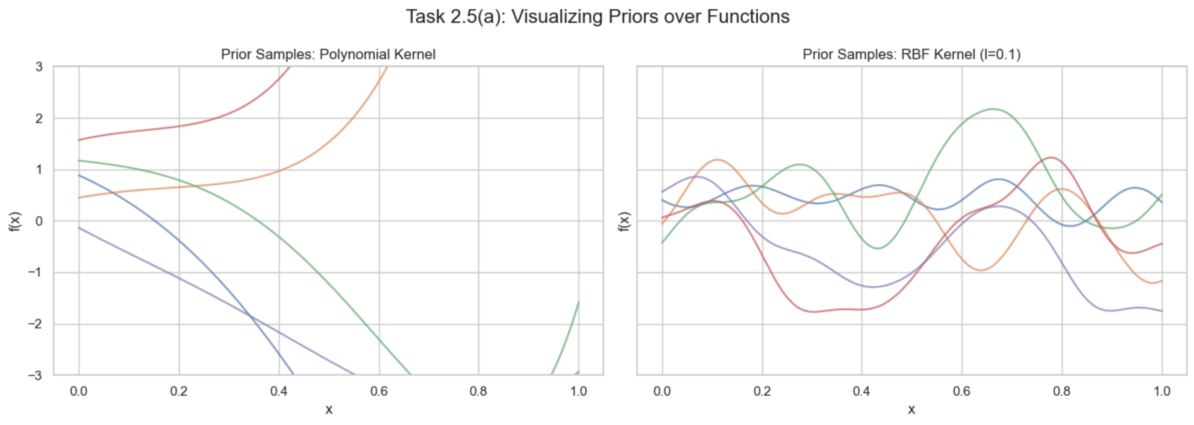
In [5]:
```
# General hyperparameters
TAU_SQ = 1.0       # Prior variance on weights
SIGMA_SQ = 0.1**2  # Observation noise variance

# Kernel-specific hyperparameters
POLY_DEGREE = 9
RBF_LENGTHSCALE = 0.1

# Plotting settings
```

```
NUM_SAMPLES = 5
X_GRID = np.linspace(0, 1, 200).reshape(-1, 1)
```

In [10]:
```
def polynomial_kernel(x1, x2, degree=POLY_DEGREE):
    """Computes the polynomial kernel k(x, x') = (1 + x*x')^degree."""
    # TODO: Implement the polynomial kernel function
    # Convert to 2D arrays if needed
    x1 = np.atleast_2d(x1)
    x2 = np.atleast_2d(x2)

    # If x1 or x2 are column vectors, we need to handle the dot product corr
    if x1.shape[1] == 1 and x2.shape[1] == 1:
        # For 1D case, compute outer product then apply kernel
        return (1 + x1 @ x2.T) ** degree
    else:
        # For higher dimensional case
        return (1 + x1 @ x2.T) ** degree


def rbf_kernel(x1, x2, lengthscale=RBF_LENGTHSCALE):
    """Computes the RBF (squared-exponential) kernel."""
    # TODO: Implement the RBF kernel function
    # Hint: You can use scipy.spatial.distance.cdist(x1, x2, 'sqeuclidean')

    # Use cdist for efficient computation of squared distances
    sq_distances = cdist(x1, x2, 'sqeuclidean')
    return np.exp(-sq_distances / (2 * lengthscale**2))
```

In [12]:
```
def poly_feature_map(x, degree=POLY_DEGREE):
    """Computes the feature map phi(x) for the polynomial kernel."""
    # TODO: Implement the feature map for the polynomial kernel
    # The d-th feature is sqrt(C(degree, d)) * x^d
    # Hint: A loop over the degree d from 0 to 'degree' is a good approach.
    x = np.atleast_2d(x)
    n_samples = x.shape[0]

    # Initialize feature matrix
    features = np.zeros((n_samples, degree + 1))

    # Compute features: sqrt(C(degree, d)) * x^d for d = 0, 1, ..., degree
    for d in range(degree + 1):
        coeff = np.sqrt(binom(degree, d))
        features[:, d] = coeff * (x.flatten() ** d)

    return features

def sample_from_prior(kernel_func, **kwargs):
    """Samples functions from a GP prior defined by a kernel."""
    if kernel_func == polynomial_kernel:
        # Weight-space view for polynomial kernel
        phi = poly_feature_map(X_GRID, POLY_DEGREE)
        samples = []
        for _ in range(NUM_SAMPLES):
            # Sample weights from prior
            w_sample = np.random.multivariate_normal(
                mean=np.zeros(phi.shape[1]),
                cov=TAU_SQ * np.eye(phi.shape[1])
            )
            # Compute function values
            f_sample = phi @ w_sample
            samples.append(f_sample)
        return np.array(samples).T
    else:
        # Function-space view for RBF kernel
        K = kernel_func(X_GRID, X_GRID, **kwargs)
        # Add jitter for numerical stability
        K += 1e-6 * np.eye(K.shape[0])

        # Sample from multivariate normal
        samples = []
        for _ in range(NUM_SAMPLES):
            sample = np.random.multivariate_normal(
                mean=np.zeros(K.shape[0]),
                cov=K
            )
            samples.append(sample)
        return np.array(samples).T
```

In [13]:
```
prior_poly = sample_from_prior(polynomial_kernel)

# Setup the 1x2 plot grid
fig, axes = plt.subplots(1, 2, figsize=(14, 5), sharey=True)
fig.suptitle("Task 2.5(a): Visualizing Priors over Functions", fontsize=16)

# Polynomial Kernel
axes[0].set_title("Prior Samples: Polynomial Kernel")
axes[0].set_xlabel("x")
axes[0].set_ylabel("f(x)")
axes[0].set_ylim(-3, 3) # Set common y-limit for easier comparison

# TODO: Call your `sample_from_prior` function for the polynomial kernel
# and plot the resulting function samples on axes[0].
poly_samples = sample_from_prior(polynomial_kernel)
for i in range(NUM_SAMPLES):
    axes[0].plot(X_GRID.flatten(), poly_samples[:, i], alpha=0.7)

# RBF Kernel
axes[1].set_title(f"Prior Samples: RBF Kernel (l={RBF_LENGTHSCALE})")
axes[1].set_xlabel("x")
axes[1].set_ylabel("f(x)")
axes[1].set_ylim(-3, 3)

# TODO: Call your `sample_from_prior` function for the RBF kernel
# and plot the resulting function samples on axes[1].
rbf_samples = sample_from_prior(rbf_kernel, lengthscale=RBF_LENGTHSCALE)
for i in range(NUM_SAMPLES):
    axes[1].plot(X_GRID.flatten(), rbf_samples[:, i], alpha=0.7)



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



TODO: Briefly comment on qualitative differences between the different kernels.

Qualitative Differences Between Kernels

• POLYNOMIAL KERNEL:

- Generates functions with global, smooth polynomial-like behavior
- High degree (9) leads to very flexible functions with potential oscillations
- Functions tend to have similar global structure across the domain
- Can exhibit extreme values at boundaries due to polynomial nature

• RBF KERNEL:

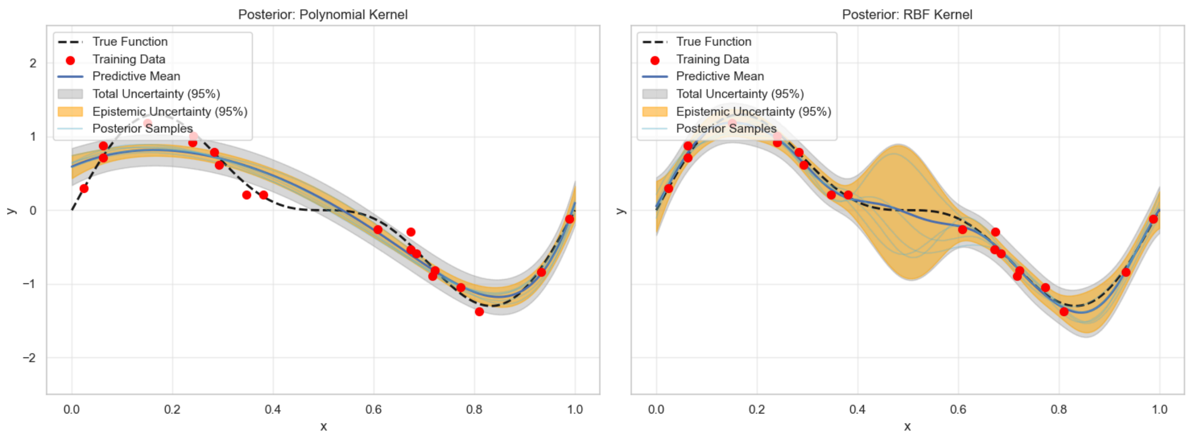
- Generates much smoother, locally varying functions
- Small lengthscale (0.1) creates functions that vary rapidly but smoothly
- Functions are more 'wiggly' but remain bounded and well-behaved
- Local changes don't affect distant parts of the function as much

In [18]:
```python
def compute_posterior_predictive(X_train, y_train, kernel_func, **kwargs):
    """Computes the mean and variance of the posterior predictive distributi
    # TODO: Compute the required kernel matrices:
    K = kernel_func(X_train, X_train, **kwargs)
    K_star = kernel_func(X_GRID, X_train, **kwargs)
    K_star_star = kernel_func(X_GRID, X_GRID, **kwargs)

    # Add noise to K for inversion
    K_noise = K + (SIGMA_SQ / TAU_SQ) * np.eye(K.shape[0])

    # Compute predictive mean using kernel regression formula
    predictive_mean = K_star @ np.linalg.solve(K_noise, y_train)

    # Compute epistemic covariance matrix
    epistemic_cov = K_star_star - K_star @ np.linalg.solve(K_noise, K_star.T

    # Get point-wise epistemic variance from diagonal
    epistemic_var = np.diag(epistemic_cov)
    # Total predictive variance
    total_var = epistemic_var + SIGMA_SQ

    return predictive_mean, total_var, epistemic_var, epistemic_cov

def sample_from_posterior(mean, cov):
    """Samples functions from the posterior predictive distribution."""
    # TODO: Draw NUM_SAMPLES from the multivariate normal distribution
    # defined by the predictive mean and the epistemic covariance matrix
    # Hint: Add a small jitter to 'cov' before sampling to ensure it is posi
    cov_stable = cov + 1e-6 * np.eye(cov.shape[0])

    samples = []
    for _ in range(NUM_SAMPLES):
        sample = np.random.multivariate_normal(mean, cov_stable)
        samples.append(sample)
    return np.array(samples).T
```

In [19]:
```
def true_function(x):
    return np.sin(2 * np.pi * x) + 0.5 * np.sin(4 * np.pi * x)

def generate_data_with_gap(n=20, noise_std=np.sqrt(SIGMA_SQ)):
    """Generates data with a gap in the middle."""
    np.random.seed(42)
    x1 = np.random.uniform(0.0, 0.4, n // 2)
    x2 = np.random.uniform(0.6, 1.0, n // 2)
    X_train = np.concatenate([x1, x2]).reshape(-1, 1)
    y_train = true_function(X_train.flatten()) + np.random.normal(0, noise_s
    return X_train, y_train
```

In [20]:
```
X_train, y_train = generate_data_with_gap()


fig, axes = plt.subplots(1, 2, figsize=(15, 6), sharey=True)
fig.suptitle(" Posterior Distributions", fontsize=16)

# A dict to cleanly loop over the two kernel models
kernels_to_test = {
    "Polynomial": (polynomial_kernel, {'degree': POLY_DEGREE}),
    "RBF": (rbf_kernel, {'lengthscale': RBF_LENGTHSCALE})
}

# 3. Loop through each kernel, compute its posterior, and plot
for ax, (name, (kernel_func, kwargs)) in zip(axes, kernels_to_test.items()):

    # Compute posterior predictive distribution
    mean, total_var, epistemic_var, epistemic_cov = compute_posterior_predic
        X_train, y_train, kernel_func, **kwargs
    )

    # Sample from posterior
    posterior_samples = sample_from_posterior(mean, epistemic_cov)

    # Calculate standard deviations for 95% credible intervals
    total_std = np.sqrt(total_var)
    epistemic_std = np.sqrt(epistemic_var)

    # Plot true function and training data
    ax.plot(X_GRID, true_function(X_GRID.flatten()), 'k--', linewidth=2, lab
    ax.scatter(X_train, y_train, color='red', s=50, zorder=5, label="Trainin

    # Plot predictive mean
    ax.plot(X_GRID, mean, 'b-', lw=2, label="Predictive Mean")

    # Plot uncertainty bands
    ax.fill_between(X_GRID.flatten(),
                    mean - 1.96 * total_std,
                        mean + 1.96 * total_std,
                        color='gray', alpha=0.3, label="Total Uncertainty (95%)'
    ax.fill_between(X_GRID.flatten(),
                        mean - 1.96 * epistemic_std,
                        mean + 1.96 * epistemic_std,
                        color='orange', alpha=0.5, label="Epistemic Uncertainty

    # Plot posterior function samples
    for i in range(NUM_SAMPLES):
        ax.plot(X_GRID, posterior_samples[:, i], 'c-', alpha=0.4, linewidth=

    # Add a dummy line for legend
    ax.plot([], [], 'c-', alpha=0.4, label="Posterior Samples")

    # Final plot settings
    ax.set_title(f"Posterior: {name} Kernel")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.legend(loc='upper left')
    ax.set_ylim(-2.5, 2.5)
    ax.grid(True, alpha=0.3)

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

Posterior Distributions



TODO: Discuss the results.

- Which kernel provides a more reasonable fit to the data and why?

  → The RBF kernel provides a much more reasonable fit because:

  - It captures the smooth, oscillatory nature of the true function better
  - The polynomial kernel shows erratic behavior and poor extrapolation
  - RBF kernel's predictions stay closer to the true function
- Compare the epistemic uncertainty for both models. Where is it largest? How does it behave inside the data gap you created?

  → Epistemic uncertainty is largest:

  - In the DATA GAP (x ∈ [0.4, 0.6]) for both models
  - At the boundaries (x=0, x=1) especially for polynomial kernel

  → In the data gap:

  - RBF: Uncertainty increases smoothly, reflects local nature of RBF

    - Polynomial: Uncertainty can be extreme due to global polynomial behavior
- How do the posterior function samples relate to the uncertainty bands? Explain what the spread of these samples represents.

  → The posterior function samples illustrate the uncertainty bands:

    - Wide spread of samples = high uncertainty (wide bands)
    - Narrow spread of samples = low uncertainty (narrow bands)
    - Samples show the range of plausible functions given the data
    - The variability of samples directly corresponds to epistemic uncertainty

  → This gives intuition: uncertainty bands summarize the spread of all possible functions that could explain the observed data