Math 50: Exercise Solutions

Units 2 & 3

Contents

1 Unit 2 Solutions

Unit 2, Exercise 2exercise (Computing conditional averages).
Given data: $\{(1,2),(1,2),(3,1),(1,4),(3,3),(2,2),(1,5)\}$
Solution:

label=() $E[Y_1]$: Average of the first coordinates: $(1+1+3+1+3+2+1)/7 = 12/7 \approx 1.71$

lbbel=() $E[Y_1|Y_2 = 2]$: When $Y_2 = 2$, we have points $(1,2)$, $(1,2)$, $(2,2)$ So $E[Y_1|Y_2 = 2] = (1+1+2)/3 = 4/3 \approx 1.33$

lcbel=() $E[Y_2|Y_1 = 1]$: When $Y_1 = 1$, we have points $(1,2)$, $(1,2)$, $(1,4)$, $(1,5)$ So $E[Y_2|Y_1 = 1] = (2+2+4+5)/4 = 13/4 = 3.25$

ldbel=() $E[Y_2|Y_1 > 1]$: When $Y_1 > 1$, we have points $(3,1)$, $(3,3)$, $(2,2)$ So $E[Y_2|Y_1 > 1] = (1+3+2)/3 = 6/3 = 2$

Unit 2, Exercise 2exercise (Evans textbook exercises). Note: This exercise references external textbook problems from Evans and is not solved here.

Unit 2, Exercise 2exercise (Independence and conditional expectation).
Solution:

label=() **Show that if X and Y are independent, then $E[X|Y = y] = E[X]$ and $E[Y|X = x] = E[Y]$**

By definition of conditional expectation:

$$\begin{aligned} E[X|Y=y] &= \sum_{x\in S_X} x \cdot P(X=x|Y=y) && (1)\\ &= \sum_{x\in S_X} x \cdot \frac{P(X=x, Y=y)}{P(Y=y)} && (2)\end{aligned}$$

Since X and Y are independent, $P(X = x, Y = y) = P(X = x)P(Y = y)$, so:

$$\begin{aligned} E[X|Y=y] &= \sum_{x\in S_X} x \cdot \frac{P(X=x)P(Y=y)}{P(Y=y)} && (3)\\ &= \sum_{x\in S_X} x \cdot P(X=x) && (4)\\ &= E[X] && (5)\end{aligned}$$

By symmetry, $E[Y|X = x] = E[Y]$.

lbbel=() **Prove the tower property of expectation**

$$\begin{aligned}
\sum_{y\in S_Y} E[X|Y=y]P(Y=y) &= \sum_{y\in S_Y}\left(\sum_{x\in S_X} x\cdot P(X=x|Y=y)\right)P(Y=y) &(6)\\
&= \sum_{y\in S_Y}\sum_{x\in S_X} x\cdot P(X=x|Y=y)\cdot P(Y=y) &(7)\\
&= \sum_{x\in S_X}\sum_{y\in S_Y} x\cdot P(X=x|Y=y)\cdot P(Y=y) &(8)\\
&= \sum_{x\in S_X} x\sum_{y\in S_Y} P(X=x|Y=y)\cdot P(Y=y) &(9)\\
&= \sum_{x\in S_X} x\sum_{y\in S_Y} P(X=x,Y=y) &(10)\\
&= \sum_{x\in S_X} x\cdot P(X=x) &(11)\\
&= E[X] &(12)
\end{aligned}$$

lcbel=() **Show that if X and Y are independent, then var$(X+Y)=$ var$(X)+$ var(Y)**

$$\begin{aligned}
\text{var}(X+Y) &= E[(X+Y)^2]-[E[X+Y]]^2 &(13)\\
&= E[X^2+2XY+Y^2]-[E[X]+E[Y]]^2 &(14)\\
&= E[X^2]+2E[XY]+E[Y^2]-E[X]^2-2E[X]E[Y]-E[Y]^2 &(15)
\end{aligned}$$

Since X and Y are independent, $E[XY]=E[X]E[Y]$, so:

$$\begin{aligned}
\text{var}(X+Y) &= E[X^2]+2E[X]E[Y]+E[Y^2]-E[X]^2-2E[X]E[Y]-E[Y]^2 &(16)\\
&= E[X^2]-E[X]^2+E[Y^2]-E[Y]^2 &(17)\\
&= \text{var}(X)+\text{var}(Y) &(18)
\end{aligned}$$

Unit 2, Exercise 2exercise (Verifying variance formula for Bernoulli variable).

For a Bernoulli variable $Y\sim$ Bernoulli(q), we want to verify that Var$(Y)=q(1-q)$.

Theoretical derivation:

$$\begin{aligned}
E[Y] &= 1\cdot q+0\cdot(1-q)=q &(19)\\
E[Y^2] &= 1^2\cdot q+0^2\cdot(1-q)=q &(20)\\
\text{Var}(Y) &= E[Y^2]-[E[Y]]^2=q-q^2=q(1-q) &(21)
\end{aligned}$$

Python verification:

```
import numpy as np
import matplotlib.pyplot as plt

# Generate range of q values to test
```

```
q_values = np.linspace(0.1, 0.9, 20)
empirical_variances = []
theoretical_variances = []

for q in q_values:
    # Generate many samples for this q value
    n_samples = 10000
    samples = np.random.binomial(1, q, n_samples)

    # Compute empirical variance
    emp_var = np.var(samples, ddof=1)  # Use sample variance
    empirical_variances.append(emp_var)

    # Compute theoretical variance
    theo_var = q * (1 - q)
    theoretical_variances.append(theo_var)

# Plot empirical vs theoretical
plt.figure(figsize=(8, 6))
plt.scatter(theoretical_variances, empirical_variances, alpha=0.7)
plt.plot([0, 0.25], [0, 0.25], 'r--', label='y = x (perfect agreement)')
plt.xlabel('Theoretical Variance q(1-q)')
plt.ylabel('Empirical Variance')
plt.title('Verification of Bernoulli Variance Formula')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

# The points should lie close to the red diagonal line
```

Unit 2, Exercise 2exercise (Conditioning with continuous variables).

Given: $Z_1 \sim \text{Normal}(0, 1)$ and $Z_2 \sim \text{Normal}(1, 2)$

Solution using Python simulations:

```
import numpy as np

# Set random seed for reproducibility
np.random.seed(42)

# Parameters for the normal distributions
mu1, sigma1 = 0, 1    # Z1 ~ Normal(0, 1)
mu2, sigma2 = 1, np.sqrt(2)  # Z2 ~ Normal(1, 2)

# Number of simulations
n_sims = 100000

# Generate samples
```

```
Z1 = np.random.normal(mu1, sigma1, n_sims)
Z2 = np.random.normal(mu2, sigma2, n_sims)

# (a) P(Z1 + Z2 > 3)
# Since Z1 + Z2 ~ Normal(1, 3), we can compute this analytically too
sum_Z = Z1 + Z2
prob_a = np.mean(sum_Z > 3)
print(f"P(Z1 + Z2 > 3)  {prob_a:.4f}")

# (b) P(Z1 + Z2 > 3 | Z1 < -1)
# Use conditional simulation: filter samples where Z1 < -1
condition_b = Z1 < -1
conditional_sum = sum_Z[condition_b]  # Only sums where Z1 < -1
prob_b = np.mean(conditional_sum > 3)
print(f"P(Z1 + Z2 > 3 | Z1 < -1)  {prob_b:.4f}")

# (c) P(Z2*Z1 > 0 | Z1 + Z2 < 4)
# Filter samples where Z1 + Z2 < 4
condition_c = sum_Z < 4
conditional_Z1 = Z1[condition_c]
conditional_Z2 = Z2[condition_c]
product = conditional_Z1 * conditional_Z2
prob_c = np.mean(product > 0)
print(f"P(Z2*Z1 > 0 | Z1 + Z2 < 4)  {prob_c:.4f}")
```

Unit 2, Exercise 2exercise (Normal approximation to estimator).

For the model $Y|X \sim \text{Bernoulli}(q_0 + (q_1 - q_0)X)$ with $X \sim \text{Bernoulli}(1/2)$:

Solution:

1. **Python functions:**

```
import numpy as np
import pandas as pd
from scipy.stats import norm

def generate_data(q0, delta, n_samples):
    """
    Generate data from the model Y|X ~ Bernoulli(q0 + delta*X)
    with X ~ Bernoulli(1/2)

    Parameters:
    q0: baseline probability (control group)
    delta: treatment effect (q1 - q0)
    n_samples: total number of samples

    Returns:
    DataFrame with columns X and Y
    """
    # Generate X values (treatment assignment)
    X = np.random.binomial(1, 0.5, n_samples)

    # Generate Y values conditional on X
    # P(Y=1|X) = q0 + delta*X
    probs = q0 + delta * X
    Y = np.random.binomial(1, probs)

    return pd.DataFrame({'X': X, 'Y': Y})

def estimate_delta(df):
    """
    Estimate the treatment effect delta from data

    Parameters:
    df: DataFrame with columns X and Y

    Returns:
    Estimated delta (difference in success rates)
    """
    # Separate treatment and control groups
    control_group = df[df['X'] == 0]
    treatment_group = df[df['X'] == 1]

    # Calculate success rates in each group
    control_rate = control_group['Y'].mean()
    treatment_rate = treatment_group['Y'].mean()

    # Return the difference
    return treatment_rate - control_rate
```

2. **Sample size estimation:**

```
def estimate_sample_size(q0, delta, margin=0.1, confidence=0.95):
    """
    Estimate required sample size for given precision

    Parameters:
    q0: baseline probability
    delta: treatment effect
    margin: desired margin of error
    confidence: confidence level

    Returns:
    Required total sample size
    """
    q1 = q0 + delta
    # Z-score for confidence level
    z = norm.ppf((1 + confidence) / 2)

    # Standard error formula
    # Each group gets N/2 samples
    variance_per_group = q0*(1-q0) + q1*(1-q1)

    # Solve for N: z * sqrt(variance_per_group / (N/2)) = margin
    # => N = 2 * variance_per_group * (z/margin)^2
    N = 2 * variance_per_group * (z / margin)**2

    return int(np.ceil(N))
```

3. **Testing:**

```
def test_sample_size_estimate(q0, delta, n_samples, n_trials=1000):
    """
    Test the sample size estimate by simulation

    Parameters:
    q0, delta: model parameters
    n_samples: sample size to test
    n_trials: number of simulation trials

    Returns:
    Fraction of trials where |estimated_delta - true_delta| < 0.1
    """
    successes = 0

    for _ in range(n_trials):
        # Generate data
        df = generate_data(q0, delta, n_samples)

        # Estimate delta
        estimated_delta = estimate_delta(df)

        # Check if within margin
        if abs(estimated_delta - delta) < 0.1:
            successes += 1

    return successes / n_trials

# Example usage:
# q0, delta = 0.3, 0.2
# n_estimated = estimate_sample_size(q0, delta)
# success_rate = test_sample_size_estimate(q0, delta, n_estimated)
# print(f"Success rate with estimated N={n_estimated}: {success_rate:.3f}")
# Should be approximately 0.95
```

Unit 2, Exercise 2exercise (Implementing a geometric distribution).
Solution:

(a) **Function** `geometric(q,n)`:

```
import numpy as np

def geometric(q, n):
    """
    Generate n samples from a geometric distribution

    Parameters:
    q: probability of success on each trial
    n: number of samples to generate

    Returns:
    Array of n geometric random variables (number of trials until first success)
    """
    samples = []

    for _ in range(n):
        # Count trials until first success
        trials = 0
        while True:
            trials += 1
            # Generate a Bernoulli trial
            success = np.random.binomial(1, q)
            if success:
                break
        samples.append(trials)

    return np.array(samples)

# Alternative vectorized implementation:
def geometric_vectorized(q, n):
    """
    Vectorized version using numpy's geometric distribution
    Note: numpy.random.geometric uses a different parameterization
    """
    # numpy's geometric gives number of failures before first success
    # We want number of trials until first success, so add 1
    return np.random.geometric(q, n)
```

(b) **Testing the variance formula:**

```
import matplotlib.pyplot as plt

def test_geometric_variance():
    """
    Test the variance formula Var(Y) = (1-q)/q^2 for geometric distribution
    """
    # Range of q values to test
    q_values = np.linspace(0.1, 0.9, 9)

    empirical_variances = []
    theoretical_variances = []

    for q in q_values:
        print(f"Testing q = {q:.1f}")

        # Generate many samples
        samples = geometric(q, 10000)

        # Compute empirical variance
        emp_var = np.var(samples, ddof=1)
        empirical_variances.append(emp_var)

        # Compute theoretical variance
        theo_var = (1 - q) / (q**2)
        theoretical_variances.append(theo_var)

        print(f"  Empirical variance: {emp_var:.2f}")
        print(f"  Theoretical variance: {theo_var:.2f}")

    # Plot empirical vs theoretical variance
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.scatter(theo_vars, emp_vars, alpha=0.7, s=60)
    plt.plot([0, max(theo_vars)], [0, max(theo_vars)], 'r--',
             label='y = x (perfect agreement)')
    plt.xlabel('Theoretical Variance (1-q)/q²')
    plt.ylabel('Empirical Variance')
    plt.title('Empirical vs Theoretical Variance')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # Plot both variances vs q
    plt.subplot(1, 2, 2)
    plt.plot(q_vals, theo_vars, 'ro-', label='Theoretical (1-q)/q²', markersize=8)
    plt.plot(q_vals, emp_vars, 'bs-', label='Empirical', markersize=8)
    plt.xlabel('Probability q')
    plt.ylabel('Variance')
    plt.title('Variance vs Probability q')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

test_geometric_variance()
```

2 Unit 3 Solutions

Unit 3, Exercise 3exercise (Bias and consistency).
For $X \sim \text{Bernoulli}(q)$ and samples $X_1, \ldots, X_N$, analyze the following estimators:
Solution:

label=() $\hat{q}_0 = \frac{1}{N}\sum_{i=1}^{N} X_i$

Standard Error: Since $X_i \sim \text{Bernoulli}(q)$, we have $\text{Var}(X_i) = q(1-q)$.

$$\text{SE}(\hat{q}_0) = \sqrt{\frac{q(1-q)}{N}}$$

Bias: $E[\hat{q}_0] = \frac{1}{N}\sum_{i=1}^{N} E[X_i] = \frac{1}{N} \cdot N \cdot q = q$. Therefore, $\hat{q}_0$ is **unbiased**.

Consistency: As $N \to \infty$, $E[\hat{q}_0] = q$ and $\text{SE}(\hat{q}_0) = \sqrt{\frac{q(1-q)}{N}} \to 0$. Therefore, $\hat{q}_0$ is **consistent**.

lbbel=() $\hat{q}_1 = \frac{Y}{N} + \frac{1}{\sqrt{N}}$ where $Y = \sum_{i=1}^{N} X_i$

Standard Error: Since $\frac{Y}{N} = \hat{q}_0$ and $\frac{1}{\sqrt{N}}$ is deterministic:

$$\text{SE}(\hat{q}_1) = \text{SE}\left(\frac{Y}{N}\right) = \sqrt{\frac{q(1-q)}{N}}$$

Bias: $E[\hat{q}_1] = E\left[\frac{Y}{N}\right] + \frac{1}{\sqrt{N}} = q + \frac{1}{\sqrt{N}}$. Therefore, $\hat{q}_1$ is **biased** with bias $\frac{1}{\sqrt{N}}$.

Consistency: As $N \to \infty$, $E[\hat{q}_1] = q + \frac{1}{\sqrt{N}} \to q$ and $\text{SE}(\hat{q}_1) \to 0$. Therefore, $\hat{q}_1$ is **consistent**.

lcbel=() $\hat{q}_2 = \frac{1}{\lfloor N/2 \rfloor}\sum_{i=1}^{\lfloor N/2 \rfloor} X_i$

Standard Error: This uses only the first $\lfloor N/2 \rfloor$ samples:

$$\text{SE}(\hat{q}_2) = \sqrt{\frac{q(1-q)}{\lfloor N/2 \rfloor}}$$

Bias: $E[\hat{q}_2] = \frac{1}{\lfloor N/2 \rfloor}\sum_{i=1}^{\lfloor N/2 \rfloor} E[X_i] = q$. Therefore, $\hat{q}_2$ is **unbiased**.

Consistency: As $N \to \infty$, $\lfloor N/2 \rfloor \to \infty$, so $E[\hat{q}_2] = q$ and $\text{SE}(\hat{q}_2) \to 0$. Therefore, $\hat{q}_2$ is **consistent**.

Unit 3, Exercise 3exercise (Estimator of mean in exponential model).
For $T \sim \text{Exponential}(\lambda)$ with $E[T] = 1/\lambda$, consider $\hat{\lambda} = \frac{1}{T}$.
Solution:

label=() **Simulation to show bias:**

```
import numpy as np
import matplotlib.pyplot as plt

def simulate_exponential_bias():
    """
    Simulate to show that lambda_hat = 1/T_bar is a biased estimator
    """
    # Create array of lambda values to test
    lambda_values = np.linspace(0.2, 2.0, 100)

    n_replicates = 10000  # Number of replicates per lambda
    n_samples = 5         # Sample size per replicate

    bias_values = []

    for lam in lambda_values:
        print(f"Testing lambda = {lam:.2f}")

        # Generate many replicates
        lambda_hat_estimates = []

        for _ in range(n_replicates):
            # Generate n_samples from Exponential(lambda)
            samples = np.random.exponential(1/lam, n_samples)

            # Compute T_bar (sample mean)
            T_bar = np.mean(samples)

            # Compute lambda_hat = 1/T_bar
            lambda_hat = 1 / T_bar
            lambda_hat_estimates.append(lambda_hat)

        # Estimate E[lambda_hat] from the replicates
        expected_lambda_hat = np.mean(lambda_hat_estimates)

        # Compute bias = |E[lambda_hat] - lambda|
        bias = abs(expected_lambda_hat - lam)
        bias_values.append(bias)

    return lambda_values, bias_values

# Run the simulation
lambdas, biases = simulate_exponential_bias()

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(lambdas, biases, 'b-', linewidth=2)
plt.xlabel('True lambda')
plt.ylabel('|E[lambda_hat] - lambda| (Absolute Bias)')
plt.title('Bias of lambda_hat = 1/T_bar as Estimator of lambda')
plt.grid(True, alpha=0.3)
plt.show()

print(f"Maximum bias: {max(biases):.4f}")
print(f"All biases > 0: {all(b > 0.001 for b in biases)}")
```

lbbel=() **Optional proof for $n = 2$:**

For $n = 2$, $\overline{T} = \frac{T_1+T_2}{2}$ where $T_1, T_2 \sim$ Exponential(λ) are independent.

By Jensen's inequality applied to the convex function $f(x) = 1/x$:

$$E\left[\frac{1}{\overline{T}}\right] = E\left[\frac{2}{T_1 + T_2}\right] > \frac{2}{E[T_1 + T_2]} = \frac{2}{2/\lambda} = \lambda$$

Therefore, $E[\hat{\lambda}] \geq \lambda$ with strict inequality unless $\overline{T}$ is constant (which it's not).

Unit 3, Exercise 3exercise (Earnings data).
Solution with Python code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm

# Load the earnings data
url = "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/Earnings/data/earnings.c
df = pd.read_csv(url)

print("Data shape:", df.shape)
print("Columns:", df.columns.tolist())
```

label=() **Expected association:** Based on commonly reported gender wage gaps, we might expect males to earn more than females on average.

lbbel=() **Gender vs. earnings regression:**

```
# Use 'earnk' (earnings in thousands) as response variable
y = df['earnk'].dropna()
X_gender = df.loc[y.index, 'male']

# Add constant for intercept
X_gender_const = sm.add_constant(X_gender)

# Fit the model
model_gender = sm.OLS(y, X_gender_const).fit()
print("=== Gender vs Earnings Regression ===")
print(model_gender.summary())

# Extract key results
gender_coef = model_gender.params['male']
gender_pvalue = model_gender.pvalues['male']

print(f"Gender coefficient: {gender_coef:.2f}")
print(f"P-value: {gender_pvalue:.4f}")
print(f"Statistically significant: {gender_pvalue < 0.05}")
```

lcbel=() **Height vs. earnings regression:**

```
# Regression of earnings on height
X_height = df.loc[y.index, 'height']
X_height_const = sm.add_constant(X_height)

model_height = sm.OLS(y, X_height_const).fit()
print("=== Height vs Earnings Regression ===")
print(model_height.summary())

height_coef = model_height.params['height']
height_pvalue = model_height.pvalues['height']

print(f"Height coefficient: {height_coef:.4f}")
print(f"P-value: {height_pvalue:.4f}")
```

ldbel=() **Separate analysis by gender:**

```
# Split the data
males = df[df['male'] == 1].dropna()
females = df[df['male'] == 0].dropna()

# Males only
y_male = males['earnk']
X_male_height = sm.add_constant(males['height'])
model_male = sm.OLS(y_male, X_male_height).fit()

# Females only
y_female = females['earnk']
X_female_height = sm.add_constant(females['height'])
model_female = sm.OLS(y_female, X_female_height).fit()

male_height_coef = model_male.params['height']
female_height_coef = model_female.params['height']

print(f"Height effect - Males: {male_height_coef:.4f}")
print(f"Height effect - Females: {female_height_coef:.4f}")
```

lebel=() **Interpretation:** Compare the significance of height effects within each gender group to determine if the overall height-earnings association is solely due to gender differences.

Unit 3, Exercise 3exercise (Quiz practice). Note: This exercise references external practice quiz materials.

Unit 3, Exercise 3exercise (Statistical significance - optional challenge).
Solution with simulation:

```
import numpy as np
from scipy import stats

def demonstrate_significance_paradox():
    """
    Demonstrate that two experiments can have different significance
    even when their difference is not significant
    """
    np.random.seed(42)

    # Experiment 1: Small sample, large effect
    n1 = 20  # 10 per group
    true_effect1 = 1.0
    sigma = 2.0

    # Generate data for experiment 1
    control1 = np.random.normal(0, sigma, n1//2)
    treatment1 = np.random.normal(true_effect1, sigma, n1//2)

    # Compute effect estimate and t-test
    effect1_hat = np.mean(treatment1) - np.mean(control1)
    pooled_se1 = sigma * np.sqrt(2 / (n1//2))
    t_stat1 = effect1_hat / pooled_se1
    p_value1 = 2 * (1 - stats.t.cdf(abs(t_stat1), df=n1-2))

    print("=== EXPERIMENT 1 ===")
    print(f"Estimated effect: {effect1_hat:.3f}")
    print(f"P-value: {p_value1:.4f}")
    print(f"Significant: {p_value1 < 0.05}")

    # Experiment 2: Large sample, small effect
    n2 = 200
    true_effect2 = 0.5

    control2 = np.random.normal(0, sigma, n2//2)
    treatment2 = np.random.normal(true_effect2, sigma, n2//2)

    effect2_hat = np.mean(treatment2) - np.mean(control2)
    pooled_se2 = sigma * np.sqrt(2 / (n2//2))
    t_stat2 = effect2_hat / pooled_se2
    p_value2 = 2 * (1 - stats.t.cdf(abs(t_stat2), df=n2-2))

    print("=== EXPERIMENT 2 ===")
    print(f"Estimated effect: {effect2_hat:.3f}")
    print(f"P-value: {p_value2:.4f}")
    print(f"Significant: {p_value2 < 0.05}")

    # Test the difference between the two effect estimates
    effect_diff = effect1_hat - effect2_hat
    se_diff = np.sqrt(pooled_se1**2 + pooled_se2**2)
    t_stat_diff = effect_diff / se_diff
    p_value_diff = 2 * (1 - stats.t.cdf(abs(t_stat_diff), df=n1+n2-4))

    print("=== DIFFERENCE TEST ===")
    print(f"Difference: {effect_diff:.3f}")
    print(f"P-value for difference: {p_value_diff:.4f}")
    print(f"Difference significant: {p_value_diff < 0.05}")

    if (p_value1 < 0.05) != (p_value2 < 0.05) and (p_value_diff >= 0.05):
        print("PARADOX DEMONSTRATED!")

demonstrate_significance_paradox()
```