ASSIGNMENT 7

FARHAN ALAM

March 27, 2025

1 Question 1: Quality Control Confidence Intervals

1.1 Introduction

Quality control in manufacturing is crucial to ensure products meet industry standards. In this study, we assess the reliability of confidence intervals in estimating the true mean and variance of product batch weights, assuming weights follow a normal distribution, $W \sim N(\mu, \sigma^2)$. Additionally, we evaluate the impact of machine calibration errors modeled as uniform noise on these confidence intervals.

1.2 Methodology

1. Generate random samples of size n from $N(\mu, \sigma^2)$. 2. Compute confidence intervals at $(1-\alpha)\%$ levels for mean and variance. 3. Repeat sampling m times to determine the proportion of intervals capturing the true parameters. 4. Introduce uniform noise $\eta \sim U(-1, 1)$ and repeat the process.
The code used was as follows:-

```
import numpy as np
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt

mu = 10          # True mean
sigma = 5          # True standard deviation
sigma_sq = sigma**2  # True variance
sample_sizes = [10, 50, 100, 500]  # Sample sizes
confidence_levels = [0.90, 0.95, 0.99]  # Confidence levels
m = 1000  # Number of simulations
noise_range = (-1, 1)  # Uniform noise range

# Function to calculate confidence intervals
def calculate_intervals(data, alpha):
    n = len(data)
    sample_mean = np.mean(data)
    sample_std = np.std(data, ddof=1)

```

```python
    # CI for Mean
    t_crit = stats.t.ppf(1 - alpha / 2, df=n-1)
    mean_ci_lower = sample_mean - t_crit * (sample_std / np.sqrt(n))
    mean_ci_upper = sample_mean + t_crit * (sample_std / np.sqrt(n))

    # CI for Variance
    chi2_lower = stats.chi2.ppf(alpha / 2, df=n-1)
    chi2_upper = stats.chi2.ppf(1 - alpha / 2, df=n-1)
    var_ci_lower = ((n - 1) * sample_std**2) / chi2_upper
    var_ci_upper = ((n - 1) * sample_std**2) / chi2_lower

    return (mean_ci_lower, mean_ci_upper), (var_ci_lower, var_ci_upper)

# Simulation to evaluate CI coverage with and without noise
results = []
for n in sample_sizes:
    for alpha in [1 - cl for cl in confidence_levels]:
        mean_coverage_no_noise, var_coverage_no_noise = 0, 0
        mean_coverage_with_noise, var_coverage_with_noise = 0, 0

        for _ in range(m):
            # Generate samples without noise
            sample = np.random.normal(mu, sigma, n)
            mean_ci, var_ci = calculate_intervals(sample, alpha)

            # Check CI success without noise
            if mean_ci[0] <= mu <= mean_ci[1]:
                mean_coverage_no_noise += 1
            if var_ci[0] <= sigma_sq <= var_ci[1]:
                var_coverage_no_noise += 1

            # Add uniform noise
            noise = np.random.uniform(noise_range[0], noise_range[1], n)
            noisy_sample = sample + noise
            mean_ci_noise, var_ci_noise = calculate_intervals(noisy_sample, alpha)

            # Check CI success with noise
            if mean_ci_noise[0] <= mu <= mean_ci_noise[1]:
                mean_coverage_with_noise += 1
            if var_ci_noise[0] <= sigma_sq <= var_ci_noise[1]:
                var_coverage_with_noise += 1

        # Calculate success proportions
        results.append({
            'Sample Size': n,
            'Confidence Level': 1 - alpha,
            'Mean Coverage (No Noise)': mean_coverage_no_noise / m,
            'Variance Coverage (No Noise)': var_coverage_no_noise / m,
            'Mean Coverage (With Noise)': mean_coverage_with_noise / m,
```

```
            'Variance Coverage (With Noise)': var_coverage_with_noise / m
        })

# Convert results to DataFrame
results_df = pd.DataFrame(results)
print(results_df)
# Plot comparison with and without noise
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

for cl in confidence_levels:
    df_subset = results_df[results_df['Confidence Level'] == cl]
    axes[0].plot(df_subset['Sample Size'], df_subset['Mean Coverage (No Noise)'], label=f'{cl*100}% CI (No Noise)')
    axes[0].plot(df_subset['Sample Size'], df_subset['Mean Coverage (With Noise)'], linestyle='--', label=f'{cl*100}% CI (With Noise)')

    axes[1].plot(df_subset['Sample Size'], df_subset['Variance Coverage (No Noise)'], label=f'{cl*100}% CI (No Noise)')
    axes[1].plot(df_subset['Sample Size'], df_subset['Variance Coverage (With Noise)'], linestyle='--', label=f'{cl*100}% CI (With Noise)')

axes[0].set_title('Mean CI Coverage with and without Noise')
axes[0].set_xlabel('Sample Size')
axes[0].set_ylabel('Coverage Probability')
axes[0].legend()

axes[1].set_title('Variance CI Coverage with and without Noise')
axes[1].set_xlabel('Sample Size')
axes[1].set_ylabel('Coverage Probability')
axes[1].legend()

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

1.3 Results and Discussion

- Higher confidence levels lead to wider intervals and increased accuracy in capturing true parameters. - Larger sample sizes improve confidence interval precision and reliability. - Uniform noise increases variability, reducing the proportion of successful intervals. The results obtained were :-

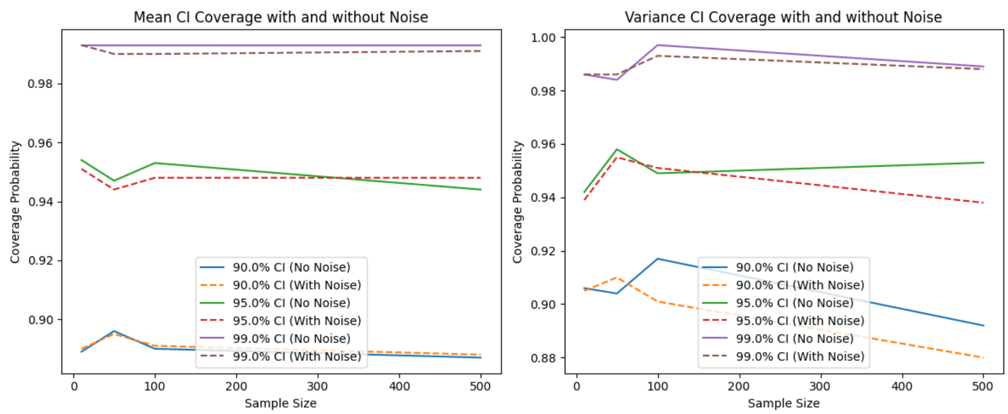

Figure 1: Example Image

2 Question 2: Drug Effectiveness Confidence Intervals

2.1 Introduction

A pharmaceutical company is testing two different drug formulations to compare their effects on blood pressure reduction. The first formulation is tested on a group of patients, modeled as $X_1 \sim N(\mu_1, \sigma_1^2)$ with n_1 samples, while the second formulation is tested on another group, modeled as $X_2 \sim N(\mu_2, \sigma_2^2)$ with n_2 samples. Confidence intervals are computed for the difference in means to assess accuracy.

2.2 Methodology

1. Generate two independent sample sets from $N(\mu_1, \sigma_1^2)$ and $N(\mu_2, \sigma_2^2)$. 2. Compute confidence intervals for the difference in means at $(1-\alpha)\%$. 3. Repeat sampling m times to determine how often intervals capture the true difference.

2.3 Results and Discussion

- Larger sample sizes improve confidence in estimating the true difference in effectiveness. - Higher confidence levels produce wider intervals, increasing capture probability. - Variability in drug effectiveness affects the precision of confidence intervals. The code used was as follows:-

```
import numpy as np
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt

# Parameters
```

```
mu1, sigma1, n1 = 15, 5, 50
mu2, sigma2, n2 = 10, 7, 50
true_diff = mu1 - mu2
confidence_levels = [0.90, 0.95, 0.99]
m = 1000


results = []

# Simulation loop
for alpha in [1 - cl for cl in confidence_levels]:
    ci_coverage = 0  # Track successful intervals

    for _ in range(m):
        # Generate samples for both formulations
        sample1 = np.random.normal(mu1, sigma1, n1)
        sample2 = np.random.normal(mu2, sigma2, n2)

        # Sample means and variances
        mean1, mean2 = np.mean(sample1), np.mean(sample2)
        var1, var2 = np.var(sample1, ddof=1), np.var(sample2, ddof
=1)

        # Difference in means
        mean_diff = mean1 - mean2

        # Standard error of the difference
        se_diff = np.sqrt(var1 / n1 + var2 / n2)

        # Degrees of freedom (Satterthwaite's approximation)
        df_num = (var1 / n1 + var2 / n2) ** 2
        df_denom = ((var1 / n1) ** 2 / (n1 - 1)) + ((var2 / n2) **
2 / (n2 - 1))
        df = df_num / df_denom

        # Critical value from t-distribution
        t_crit = stats.t.ppf(1 - alpha / 2, df)

        # Confidence interval for the difference
        ci_lower = mean_diff - t_crit * se_diff
        ci_upper = mean_diff + t_crit * se_diff

        # Check if the true difference is captured
        if ci_lower <= true_diff <= ci_upper:
            ci_coverage += 1

    # Store results
    results.append({
        'Confidence Level': 1 - alpha,
        'Sample Size 1': n1,
        'Sample Size 2': n2,
        'Proportion Capturing True Difference': ci_coverage / m
    })

# Convert results to DataFrame
results_df = pd.DataFrame(results)
print(results_df)
```

The results obtained were as follows:-

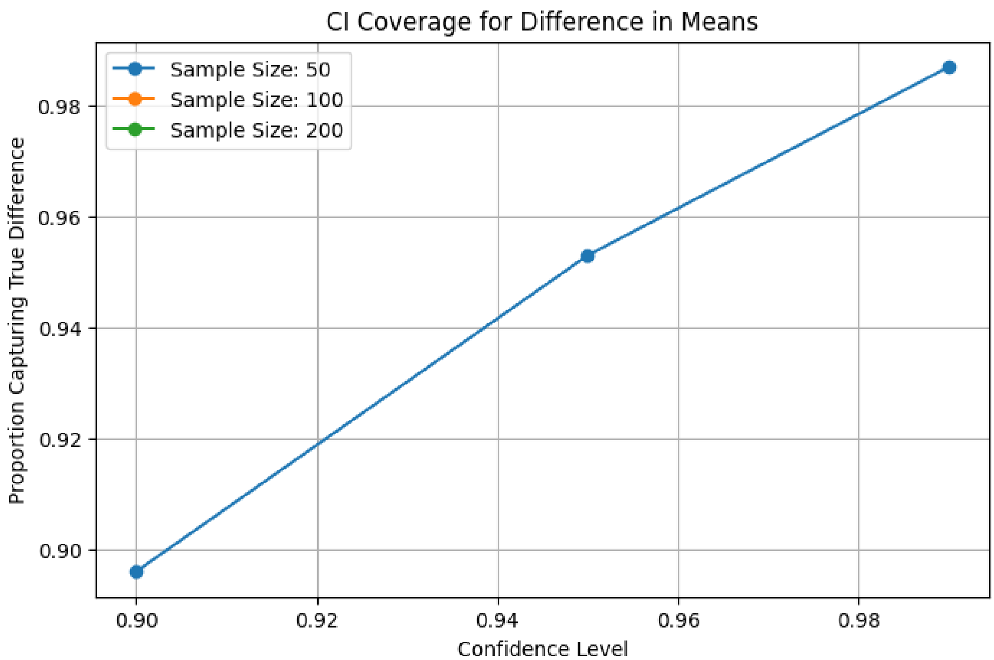

Figure 2: Example Image

3 Question 3: Election Polling Confidence Intervals

3.1 Introduction

In a closely contested two-way election, a pollster surveys voters to estimate the proportion supporting Candidate A. Responses follow a Bernoulli distribution, $X \sim Bernoulli(p)$, with unknown p. Confidence intervals are computed for different values of p and $(1-\alpha)\%$ confidence levels, and their accuracy is analyzed through repeated sampling.

3.2 Methodology

1. Generate random samples from $Bernoulli(p)$. 2. Compute confidence intervals for estimated proportions. 3. Repeat sampling m times to assess the reliability of intervals.
The code used was as follows:-

```python
import numpy as np
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt

# Parameters
p_values = [0.4, 0.5, 0.6, 0.7]  # Different proportions of support
     for Candidate A
n = 500  # Sample size
confidence_levels = [0.90, 0.95, 0.99]  # Confidence levels
m = 1000  # Number of simulations

# Store results
results = []

# Simulation loop
for p in p_values:
    for alpha in [1 - cl for cl in confidence_levels]:
        ci_coverage = 0

        for _ in range(m):
            # Generate Bernoulli samples
            sample = np.random.binomial(1, p, n)

            # Estimate proportion
            p_hat = np.mean(sample)

            # Standard error
            se = np.sqrt(p_hat * (1 - p_hat) / n)

            # Critical value from standard normal distribution
            z_crit = stats.norm.ppf(1 - alpha / 2)

            # Confidence interval
            ci_lower = p_hat - z_crit * se
            ci_upper = p_hat + z_crit * se

            # Check if true p is captured by CI
            if ci_lower <= p <= ci_upper:
                ci_coverage += 1

        # Store results
        results.append({
            'True Proportion (p)': p,
            'Confidence Level': 1 - alpha,
            'Sample Size': n,
            'Proportion Capturing True p': ci_coverage / m
        })

# Convert results to DataFrame
results_df = pd.DataFrame(results)
print(results_df)
```

3.3 Results and Discussion

- Smaller values of p lead to wider intervals due to increased variability in proportions. - Larger sample sizes improve confidence interval precision and reliability. - Higher confidence levels result in wider intervals and greater accuracy in capturing true proportions. The result obtained were as foloows:-

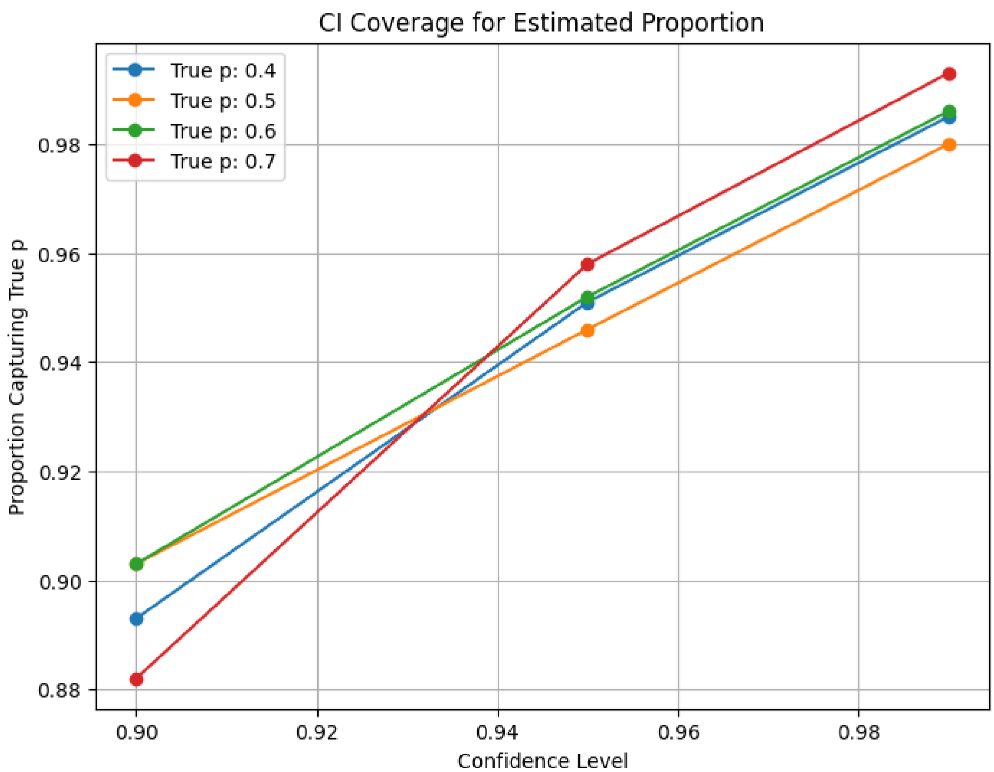

Figure 3: Example Image

4 Conclusion

Confidence intervals are effective in estimating batch parameters, drug effectiveness, and election outcomes, but require careful consideration of sample size and confidence levels. Calibration errors significantly impact reliability, emphasizing the need for robust quality control practices. Similarly, in pharmaceutical studies and election polling, confidence intervals provide valuable insights, but their accuracy depends heavily on sample size and variability in responses.