EE-325
Programming Assignment-4

October 31, 2024

Name	Roll Number
Harsha Sai Srinivas	23B1202
Yaswanth Ram Kumar	23B1277
Bhavishya Singh Premi	23B1230

Part - 1

Derivation of Marginal Probability Mass Function of Y

Here, we determined the joint probability mass function (PMF) of X and Y, the marginal PMF of Y, and calculated a suitable premium α that ensures a low probability of a negative surplus

$$S = \alpha X - \Omega Y,$$

where Ω is the insured payout amount. Following the derivation, we approximate distributions using Gaussian approximations for large values of λ and small values of p, as specified in the assignment. We then compute the probability $P(S < \alpha)$ for different threshold values α and determine the minimum α that satisfies the requirement $P(S < \alpha) < 0.01$.

Let $X \sim \text{Poisson}(\lambda)$. We are given:

$$P(X = x, Y = y) = P(Y = y|X = x) \cdot P(X = x).$$

Also,

$$P(Y = y|X = x) = \binom{x}{y} p^y (1-p)^{x-y}.$$

Thus, the joint PMF is:

$$P(X = x, Y = y) = \binom{x}{y} p^y (1-p)^{x-y} \cdot \frac{\lambda^x e^{-\lambda}}{x!}.$$

Expanding the binomial coefficient:

$$P(X = x, Y = y) = \frac{x!}{y!(x-y)!} p^y (1-p)^{x-y} \cdot \frac{\lambda^x e^{-\lambda}}{x!}.$$

Simplifying, we get:

$$P(X = x, Y = y) = \frac{\lambda^x e^{-\lambda}}{y!(x-y)!} p^y (1-p)^{x-y}.$$

To find the marginal PMF of Y, denoted $f_Y(y)$, we sum over all $x \geq y$:

$$f_Y(y) = \sum_{x=y}^{\infty} P(X = x, Y = y) = \sum_{x=y}^{\infty} \frac{\lambda^x e^{-\lambda}}{y!(x-y)!} p^y (1-p)^{x-y}.$$

On simplifying, we get:

$$f_Y(y) = \frac{(\lambda p)^y e^{-\lambda p}}{y!}.$$

i.e. $Y \sim \text{Poisson}(\lambda p)$.

Now, Poisson is a limiting Binomial, with $n \to \infty$. Also, the Binomial variable is a sum of i.i.d. Bernoullis. Applying the Central Limit Theorem to the sum of Bernoullis, we can approximate the Poisson (their sum) to be distributed as:

$$X \approx N(\lambda, \lambda) \quad \text{and} \quad Y \approx N(\lambda p, \lambda p(1-p)).$$

The surplus S is defined as:

$$S = \alpha X - \Omega Y$$

where:

- $X \sim \text{Poisson}(\lambda)$, the number of policies sold in a year.
- $Y \sim \text{Poisson}(\lambda p)$, the number of claims in the year.

Expectation of Surplus

The **expected value** $E(S)$ is given by:

$$E(S) = E(\alpha X - \Omega Y) = \alpha E(X) - \Omega E(Y)$$

Since $X \sim \text{Poisson}(\lambda)$, we have $E(X) = \lambda$.
For $Y \sim \text{Poisson}(\lambda p)$, the expected value $E(Y) = \lambda p$.
Thus:

$$E(S) = \alpha\lambda - \Omega\lambda p$$

The **variance of** S, $\text{Var}(S)$, is given by:

$$\text{Var}(S) = \text{Var}(\alpha X - \Omega Y) = \alpha^2 \text{Var}(X) + \Omega^2 \text{Var}(Y) - 2\alpha\Omega \,\text{Cov}(X, Y).$$

Since $X \sim \text{Poisson}(\lambda)$, we have $\text{Var}(X) = \lambda$.
For $Y \sim \text{Poisson}(\lambda p)$, we have $\text{Var}(Y) = \lambda p$.
The covariance between X and Y, $\text{Cov}(X, Y)$, is given by:

$$\text{Cov}(X, Y) = E((X - \lambda)(Y - \lambda p)) = p\lambda.$$

Therefore:

$$\text{Var}(S) = \alpha^2\lambda + \Omega^2\lambda p - 2\alpha\Omega p\lambda.$$

To satisfy $\Pr(S < a) < 0.01$, we standardize S by subtracting $E(S)$ and dividing by $\sqrt{\text{Var}(S)}$, and set it equal to the z-score (inverse of the Gaussian CDF) for a probability of 0.01. Let $Z \sim \mathcal{N}(0, 1)$ be a standard normal variable. Then we want:

$$\Pr\left(\frac{S - E(S)}{\sqrt{\text{Var}(S)}} < \frac{a - E(S)}{\sqrt{\text{Var}(S)}}\right) < 0.01.$$

For a probability of 0.01, the z-score is approximately -2.3263. Thus:

$$\frac{a - (\alpha\lambda - \Omega\lambda p)}{\sqrt{\alpha^2\lambda + \Omega^2\lambda p - 2\alpha\Omega p\lambda}} < -2.3263 = Z.$$

This setup can then be solved numerically to find α, the minimum premium:

$$a - \alpha\lambda + \Omega\lambda p < Z\sqrt{\alpha^2\lambda + \Omega^2\lambda p - 2\alpha\Omega p\lambda}.$$

Premiums Calculated for Different λ, p

λ	p	a	α
10000	0.01	10	0.12415
10000	0.01	0	0.12315
10000	0.01	-10	0.12215
10000	0.01	-20	0.12115
10000	0.01	-30	0.12015
10000	0.01	-100	0.11315

Table 1: Minimum Premium α for $\lambda = 10000$, $p = 0.01$

λ	p	a	α
10000	0.02	10	0.23358
10000	0.02	0	0.23258
10000	0.02	-10	0.23158
10000	0.02	-20	0.23058
10000	0.02	-30	0.22958
10000	0.02	-100	0.22257

Table 2: Minimum Premium α for $\lambda = 10000$, $p = 0.02$

λ	p	a	α
100000	0.01	10	0.10742
100000	0.01	0	0.10732
100000	0.01	-10	0.10722
100000	0.01	-20	0.10712
100000	0.01	-30	0.10702
100000	0.01	-100	0.10632

Table 3: Minimum Premium α for $\lambda = 100000$, $p = 0.01$

λ	p	a	α
100000	0.02	10	0.21040
100000	0.02	0	0.21030
100000	0.02	-10	0.21020
100000	0.02	-20	0.21010
100000	0.02	-30	0.21000
100000	0.02	-100	0.20930

Table 4: Minimum Premium α for $\lambda = 100000$, $p = 0.02$

Part - 2

Finding Minimum Premiums

Our goal is to set the smallest premiums, α_1 and α_2, for two groups, that meet the following conditions:

- $P(S_1 < a_1) < 0.01$ Demographic 1's risk
- $P(S_2 < a_2) < 0.01$ Demographic 2's risk
- $P(S < a) < 0.01$ Total risk across both demographics

The initial approach uses a "grid search" to test different values of α_1 and α_2 and find the minimum premiums that satisfy these conditions. Interestingly, the same values that satisfy each group individually also work for the combined risk, giving the solution:

$$(\alpha_1, \alpha_2) = (0.210210, 2.467467)$$

These values satisfy:

$$\alpha_1 = 0.210210$$
$$\alpha_2 = 2.467467$$

The resulting probabilities are:

$$\text{Prob}(S_1 < a_1) = 0.006138$$
$$\text{Prob}(S_2 < a_2) = 0.009637$$
$$\text{Prob}(S < a) = 0.001555$$

Improving Efficiency with Binary Search

The grid search is accurate but time-consuming. A binary search could improve efficiency by narrowing down the range of possible values for α_1 and α_2 step-by-step. In each step, we take the middle of the range, check if it meets the conditions, and then adjust the range until we find the smallest possible premiums. This method would save computation time.

Choosing the Best (α_1, α_2) Pair

There may be multiple (α_1, α_2) pairs that satisfy the conditions. The best choice is the pair with the smallest values for both α_1 and α_2, keeping premiums low for each group while meeting risk requirements. As there is no other information given, best α would be to pick the smallest possible αs we have computed, but if there is more information provided like the market demand, competition to the Insurance Company, we can pick higher values of $\alpha 1$ and $\alpha 2$ still keeping the pot-breaking risk lower.

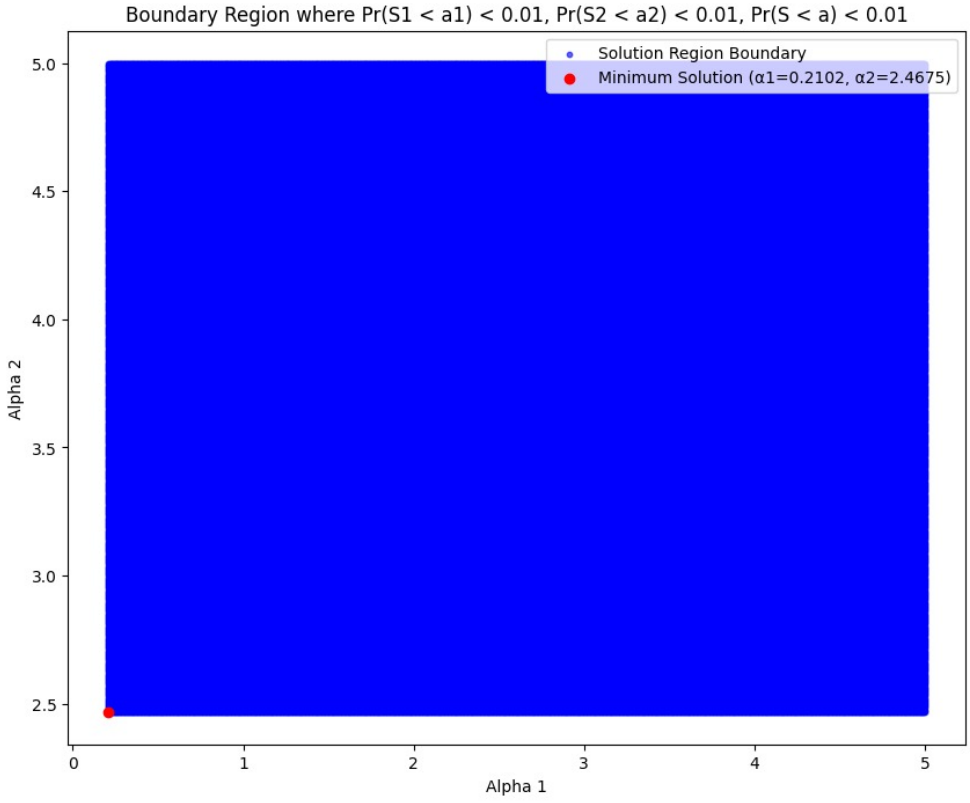


Figure 1: Possible Values of $\alpha 1$ and $\alpha 2$

Cross-Subsidy

Cross-Subsidy Meaning: Cross-subsidy is a strategy where one group (usually high-risk) is financially supported by charging slightly higher premiums to another group (typically low-risk). In insurance, this means

setting premiums so that low-risk customers pay a bit more, which helps cover potential losses from high-risk customers. It can make insurance accessible to high-risk groups but requires careful handling.

How to Achieve Cross-Subsidy

To apply cross-subsidy, we would set premiums so that low-risk customers pay a bit above their actual risk level. This additional premium creates a buffer to help cover claims from high-risk customers. By slightly increasing the premiums for the low-risk group, we meet risk requirements while keeping coverage affordable for high-risk customers.

Arguments in Favor of Cross-Subsidy

1. **More Accessible Insurance:** Cross-subsidy helps make insurance available to high-risk individuals by keeping their premiums affordable. This approach widens our customer base and allows us to include a range of risk levels.
2. **Risk Sharing for Stability:** The method pools premiums from different risk groups, with low-risk customers' extra contributions creating a reserve. This reserve helps cover unexpected high-risk claims, adding stability to the insurance pool.

Arguments Against Cross-Subsidy

1. **Could Feel Unfair:** Low-risk customers might feel they're paying too much to cover high-risk claims, which could seem unfair. If they find the premiums too high, they might leave, reducing our customer base.
2. **Imbalance Risks:** Relying too heavily on low-risk customers to support high-risk ones can backfire. If there's a sudden increase in high-risk customers, the funds may not be sufficient, risking the stability of the system.
3. **Competitive Disadvantage:** Charging low-risk customers too much may lead them to find better rates elsewhere. Losing these customers would weaken the buffer needed to support high-risk claims, potentially putting the cross-subsidy strategy at risk.

Part - 3

Deriving the 95% Confidence Interval

To construct a 95% confidence interval for an unknown population mean μ based on a sample, we use the sample mean $\hat{\mu}$ as our point estimate for μ.

By the Central Limit Theorem, the sampling distribution of $\hat{\mu}$ is approximately normal if the sample size is sufficiently large. This approximation allows us to use properties of the standard normal distribution to determine how much $\hat{\mu}$ is likely to vary around μ. Since we aim to capture 95% of the possible sample means around μ, we focus on the middle 95% of the standard unit normal distribution. For a unit normal variable Z (with mean 0 and variance 1), the probability that it falls within ± 1.96 is approximately 0.95.

$$P(-1.96 < Z < 1.96) \approx 0.95$$

This critical value 1.96 (obtained from the Z-table) represents the distance from the mean that encompasses 95% of the values in a unit normal distribution, with 2.5% of the values in each tail.

Constructing the Confidence Interval

To translate this result to our confidence interval for μ, we can write:

$$\hat{\mu} - 1.96 \cdot \sigma_{\hat{\mu}} \leq \mu \leq \hat{\mu} + 1.96 \cdot \sigma_{\hat{\mu}}$$

where $\sigma_{\hat{\mu}}$ represents the spread of the sample mean around the true mean μ.

Thus, the 95% confidence interval for μ is given by:

$$(\hat{\mu} - 1.96 \cdot \sigma_{\hat{\mu}},\ \hat{\mu} + 1.96 \cdot \sigma_{\hat{\mu}})$$

This interval indicates that if we were to take many random samples and construct intervals in this manner, 95% of them would contain the true population mean μ. Here, μ can be taken as λ and p to find estimates for λ and p.

For the point estimate of p, we can take $\hat{p} = \frac{\sum Y}{\sum X}$.

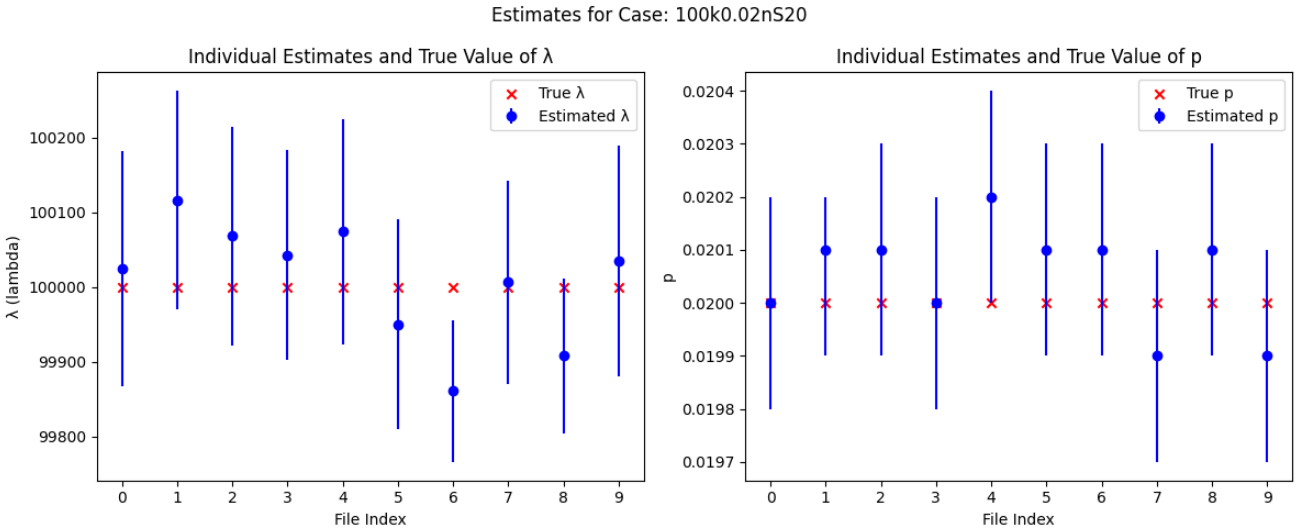


Figure 2: Graph Depicting Point Estimates and CI for N = 20 case, iterated ten times

Threshold a	λ (Estimate)	λ CI Lower	λ CI Upper	p (Estimate)	p CI Lower	p CI Upper	Minimum α
10	100008.73	99929.489651	100038.430349	0.02005	0.019836	0.020009	0.209939
0	100008.73	99929.489651	100038.430349	0.02005	0.019836	0.020009	0.209838
-10	100008.73	99929.489651	100038.430349	0.02005	0.019836	0.020009	0.209738
-20	100008.73	99929.489651	100038.430349	0.02005	0.019836	0.020009	0.209638
-30	100008.73	99929.489651	100038.430349	0.02005	0.019836	0.020009	0.209537
-100	100008.73	99929.489651	100038.430349	0.02005	0.019836	0.020009	0.208836

Table 5: Estimate and αs for N = 100

Threshold a	λ (Estimate)	λ CI Lower	λ CI Upper	p (Estimate)	p CI Lower	p CI Upper	Minimum α
0	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209939
1	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209838
-10	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209738
-20	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209638
-30	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209537
-100	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.208836

Table 6: Estimate and αs for N = 100

Obtaining premium α using the estimated parameters

The programme 1 has been used for obtaining α from the estimates. For both N = 100 and N = 20, for different lower thresholds for Surplus.

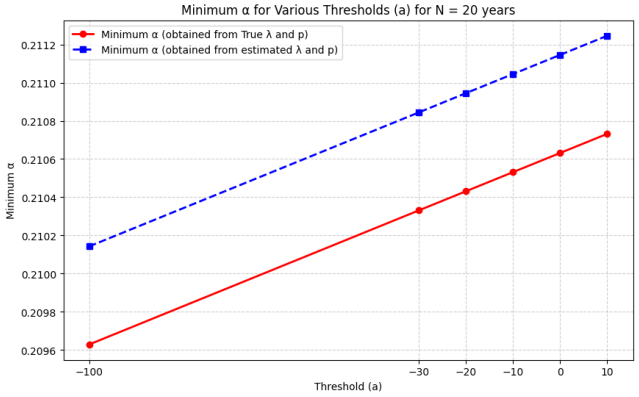


Figure 3: For N = 20 years

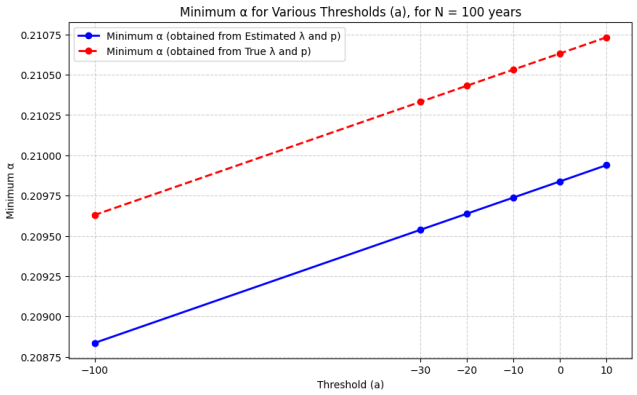


Figure 4: For N = 100 years

Please follow this link for the CSV files we have used for this part : Synthetic Data

All programmes and plots attached below.

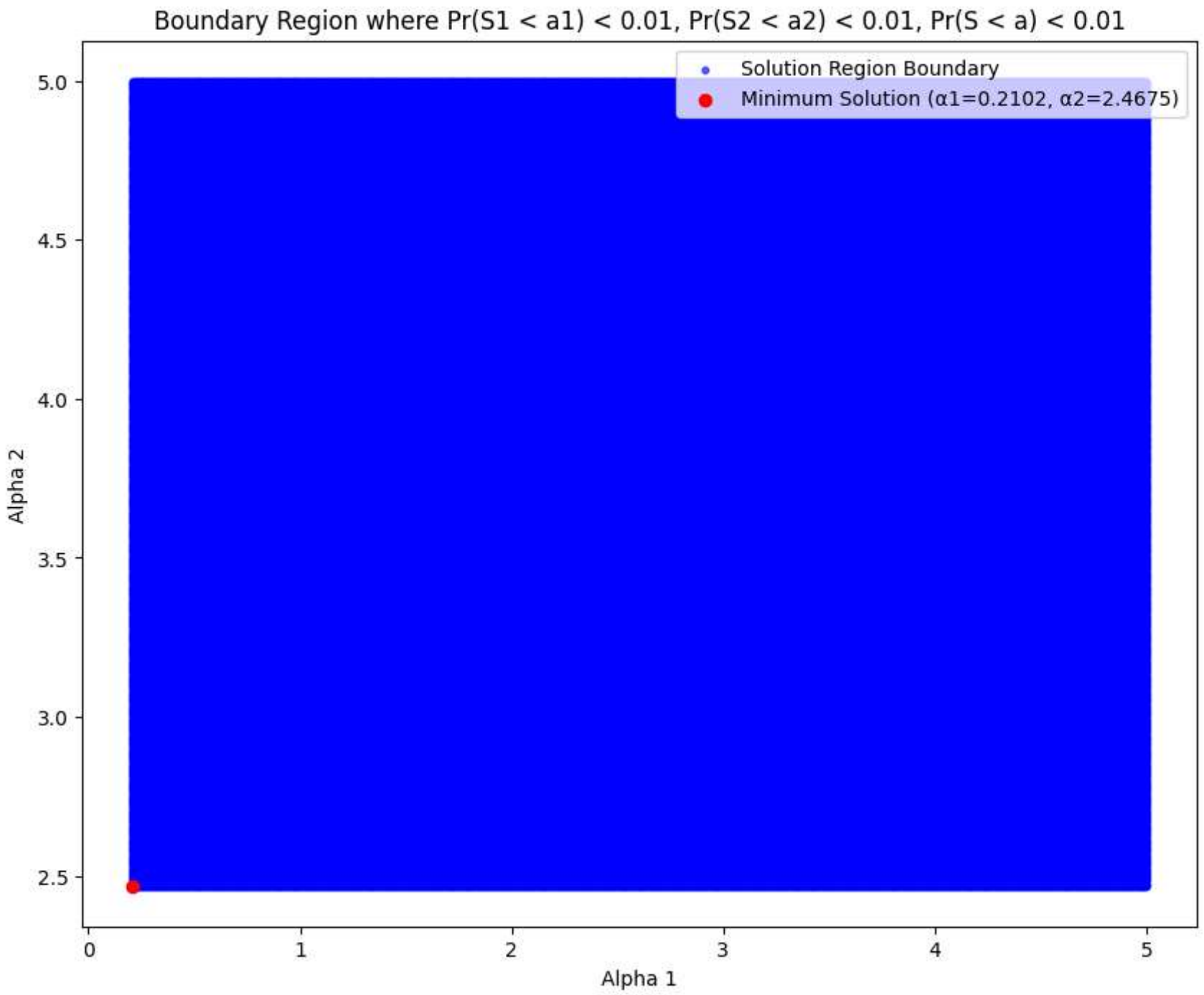
In [ ]:
```python
import numpy as np
from scipy.stats import norm

# Parameters
Omega = 10
a_values = [10, 0, -10, -20, -30, -100]
lambda_values = [10000, 100000]
p_values = [0.01, 0.02]
target_prob = 0.01
tolerance = 1e-6  # Precision for binary search

# Function to calculate the mean and variance of S
def calc_mean_variance_S(alpha, lambda_, p, Omega):
    # Mean of S
    mean_S = alpha * lambda_ - Omega * lambda_ * p

    # Variance of X and Y and Cov(X, Y)
    var_X = lambda_  # Variance of X (Poisson)
    var_Y = lambda_ * p  # Variance of (Y|X) (Binomial)
    cov_XY = lambda_ * p  # Cov(X, Y) = λp (because Y given X is binomial)

    # Variance of S
    var_S = (alpha ** 2) * var_X + (Omega ** 2) * var_Y - 2 * cov_XY * Omega * alpha
    return mean_S, var_S

# Function to find minimum alpha using binary search
def find_min_alpha_binary_search(lambda_, p, Omega, a_values, target_prob, tolerance):
    alphas = []
    for a in a_values:
        low, high = 0, 3  # Search range for alpha
        while high - low > tolerance:
            alpha = (low + high) / 2
            mean_S, var_S = calc_mean_variance_S(alpha, lambda_, p, Omega)
            std_dev_S = np.sqrt(var_S)
            prob = norm.cdf(a, loc=mean_S, scale=std_dev_S)

            if prob < target_prob:
                high = alpha  # Decrease upper bound if condition is met
            else:
                low = alpha  # Increase lower bound if condition is not met

        alphas.append((low + high) / 2)
    return alphas

# Calculate and display the results
results = {}
for lambda_ in lambda_values:
    for p in p_values:
        key = f"lambda={lambda_}, p={p}"
        results[key] = find_min_alpha_binary_search(lambda_, p, Omega, a_values, target_prob, tolerance)

# Displaying the results in a table format
print("Results:")
print("a values:", a_values)
for key, alphas in results.items():
    print(f"{key}: minimum alphas = {alphas}")
```

```
Results:
a values: [10, 0, -10, -20, -30, -100]
lambda=10000, p=0.01: minimum alphas = [0.124153733253479, 0.12315309047698975, 0.12215244770050049, 0.12115180492401
123, 0.12015187740325928, 0.11314880847930908]
lambda=10000, p=0.02: minimum alphas = [0.23357856273651123, 0.23257791996002197, 0.23157727718353271, 0.230576634407
04346, 0.2295759916305542, 0.222572922706604]
lambda=100000, p=0.01: minimum alphas = [0.10741961002349854, 0.10732018947601318, 0.10722005367279053, 0.10711991786
956787, 0.10701978206634521, 0.10631954669952393]
lambda=100000, p=0.02: minimum alphas = [0.21039927005767822, 0.21029913425445557, 0.21019971370697021, 0.21009957790
374756, 0.2099994421005249, 0.2092992067337036]
```

In [ ]:
```python
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

# Parameters for each demographic
params = {
    "demographic_1": {
        "lambda": 100000,
        "Omega": 10,
        "p": 0.02,
        "a": -100,
        "cov_XY": 5  # Covariance between X and Y in demographic 1
    },
    "demographic_2": {
        "lambda": 5000,
        "Omega": 100,
        "p": 0.02,
        "a": -15,
        "cov_XY": 15  # Covariance between X and Y in demographic 2
    },
}
```

```python
# Total surplus target
a_total = 105
target_prob = 0.01

# Function to calculate the mean and variance of S1, S2, and their sum S, incorporating covariance within demographic
def calc_mean_variance_S(alpha1, alpha2, params):
    lambda1, Omega1, p1, cov_XY1 = params["demographic_1"]["lambda"], params["demographic_1"]["Omega"], params["demog
    lambda2, Omega2, p2, cov_XY2 = params["demographic_2"]["lambda"], params["demographic_2"]["Omega"], params["demog

    # Means of S1 and S2
    mean_S1 = alpha1 * lambda1 - Omega1 * lambda1 * p1
    mean_S2 = alpha2 * lambda2 - Omega2 * lambda2 * p2
    mean_S = mean_S1 + mean_S2

    # Variances of S1 and S2, including covariances within demographics
    var_S1 = (alpha1 ** 2) * lambda1 + (Omega1 ** 2) * (lambda1 * p1 * (1 - p1)) - 2 * cov_XY1
    var_S2 = (alpha2 ** 2) * lambda2 + (Omega2 ** 2) * (lambda2 * p2 * (1 - p2)) - 2 * cov_XY2

    # Total variance for S = S1 + S2 (S1 and S2 are independent)
    var_S = var_S1 + var_S2

    return mean_S1, var_S1, mean_S2, var_S2, mean_S, var_S

# Function to find the boundary region and minimum values within that region
def find_boundary_and_min_alpha(target_a1, target_a2, target_a_total, target_prob, params, search_range=5, divisions=
    boundary_pairs = []
    min_alpha1, min_alpha2 = float('inf'), float('inf')
    min_result = None

    # Meshgrid for plotting heatmap
    alpha1_values = np.linspace(0, search_range, divisions)
    alpha2_values = np.linspace(0, search_range, divisions)

    # Perform grid search over the 2D grid
    for alpha1 in alpha1_values:
        for alpha2 in alpha2_values:
            # Calculate means and variances for S1, S2, and S, with covariance within demographics
            mean_S1, var_S1, mean_S2, var_S2, mean_S, var_S = calc_mean_variance_S(alpha1, alpha2, params)

            # Probabilities for each condition
            prob_S1 = norm.cdf(target_a1, loc=mean_S1, scale=np.sqrt(var_S1))
            prob_S2 = norm.cdf(target_a2, loc=mean_S2, scale=np.sqrt(var_S2))
            prob_S = norm.cdf(target_a_total, loc=mean_S, scale=np.sqrt(var_S))

            # Check if all probabilities meet the criteria
            if prob_S1 < target_prob and prob_S2 < target_prob and prob_S < target_prob:
                boundary_pairs.append((alpha1, alpha2))

                # Update minimum alpha1, alpha2 if this pair is smaller
                if alpha1 < min_alpha1 or (alpha1 == min_alpha1 and alpha2 < min_alpha2):
                    min_alpha1, min_alpha2 = alpha1, alpha2
                    min_result = (prob_S1, prob_S2, prob_S)

    # Return minimum alpha1, alpha2, and the boundary pairs for plotting
    return min_alpha1, min_alpha2, min_result, boundary_pairs

# Calculate the boundary and minimum alphas
min_alpha1, min_alpha2, min_result, boundary_pairs = find_boundary_and_min_alpha(
    params["demographic_1"]["a"],
    params["demographic_2"]["a"],
    a_total,
    target_prob,
    params,
    search_range=5,
    divisions=1000
)

# Plotting the boundary region
plt.figure(figsize=(10, 8))
alpha1_boundary, alpha2_boundary = zip(*boundary_pairs)  # Unzip the boundary pairs

plt.scatter(alpha1_boundary, alpha2_boundary, s=10, c='blue', alpha=0.6, label="Solution Region Boundary")
plt.scatter(min_alpha1, min_alpha2, color="red", label=f"Minimum Solution (α1={min_alpha1:.4f}, α2={min_alpha2:.4f})"
plt.xlabel("Alpha 1")
plt.ylabel("Alpha 2")
plt.title("Boundary Region where Pr(S1 < a1) < 0.01, Pr(S2 < a2) < 0.01, Pr(S < a) < 0.01")
plt.legend()
plt.show()

# Display the minimum alpha values
print("Minimum values for alpha1 and alpha2 such that probabilities are close to target probability 0.01:")
print(f"alpha1: {min_alpha1:.6f}, alpha2: {min_alpha2:.6f}")
print(f"Prob(S1 < a1): {min_result[0]:.6f}, Prob(S2 < a2): {min_result[1]:.6f}, Prob(S < a): {min_result[2]:.6f}")
```

```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Creating legend with loc="best"
can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```




```
Minimum values for alpha1 and alpha2 such that probabilities are close to target probability 0.01:
alpha1: 0.210210, alpha2: 2.467467
Prob(S1 < a1): 0.006138, Prob(S2 < a2): 0.009637, Prob(S < a): 0.001555
```

For Part 3: Estimating λ and p from Synthetic Data

Done by:

- 23B1202 - Harsha Sai Srinivas P
- 23B1230 - Bhavishya Singh Premi
- 23B1277 - Yaswanth Ram Kumar B

In [93]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

- Functions for Estimation of λ and p

In [94]:
```
def calculate_point_and_ci(data, confidence=0.95): # Function to calculate point estimate and confidence interval for lambda
    sample_mean = np.mean(data) # Point estimate for λ (probability for claim)
    sample_std_dev = np.std(data, ddof=1)
    spread_sample_mean = sample_std_dev / np.sqrt(len(data))
    z_score = 1.96  # for 95% confidence, 2.5% in each of the tails.
    ci_lower = sample_mean - z_score * spread_sample_mean
    ci_upper = sample_mean + z_score * spread_sample_mean
    return sample_mean, (ci_lower, ci_upper)

def process_single_file(filename): # Just To handle multiple CSVs in a simpler manner.
    data = pd.read_csv(filename)
    X = data['Insurers']
    lambda_estimate, lambda_conf_interval = calculate_point_and_ci(X)

    Y = data['Claims']
    total_claims = Y.sum()
    total_insurers = X.sum()
    p_estimate = total_claims / total_insurers # Point estimate for p (probability for claim)
    p_ci_lower = p_estimate - 1.96 * np.sqrt((p_estimate * (1 - p_estimate)) / total_insurers)
    p_ci_upper = p_estimate + 1.96 * np.sqrt((p_estimate * (1 - p_estimate)) / total_insurers)
    return lambda_estimate, lambda_conf_interval, p_estimate, (p_ci_lower, p_ci_upper)
```

- Pre-processing the data from CSV files

In [95]:
```
file_groups = {
    '10k0.01nS20': [f'10k0.01nS20_{i}.csv' for i in range(1, 11)],
    '100k0.01nS20': [f'100k0.01nS20_{i}.csv' for i in range(1, 11)],
    '100k0.02nS20': [f'100k0.02nS20_{i}.csv' for i in range(1, 11)],
    '10k0.02nS20': [f'10k0.02nS20_{i}.csv' for i in range(1, 11)]
}

true_values = {
    '10k0.01nS20': {'true_lambda': 10000, 'true_p': 0.01},
    '100k0.01nS20': {'true_lambda': 100000, 'true_p': 0.01},
    '100k0.02nS20': {'true_lambda': 100000, 'true_p': 0.02},
    '10k0.02nS20': {'true_lambda': 10000, 'true_p': 0.02}
}
```

- Computing each case and storing results

In [96]:
```
all_results = {}

for case, files in file_groups.items():
    true_lambda = true_values[case]['true_lambda']
    true_p = true_values[case]['true_p']

    case_results = []
    for filename in files:
        lambda_estimate, lambda_conf_interval, p_estimate, p_conf_interval = process_single_file(filename)
        lambda_estimate = np.round(lambda_estimate, 2)
        lambda_conf_interval = (np.round(lambda_conf_interval[0], 2), np.round(lambda_conf_interval[1], 2))
        p_estimate = np.round(p_estimate, 4)
        p_conf_interval = (np.round(p_conf_interval[0], 4), np.round(p_conf_interval[1], 4))

        case_results.append({
            'True λ': true_lambda,
            'Estimated λ': lambda_estimate,
            'λ CI Lower': lambda_conf_interval[0],
            'λ CI Upper': lambda_conf_interval[1],
            'True p': true_p,
            'Estimated p': p_estimate,
            'p CI Lower': p_conf_interval[0],
            'p CI Upper': p_conf_interval[1]
        })

    all_results[case] = pd.DataFrame(case_results)
```

- Displaying the results for each case

In [97]:
```
for case, df in all_results.items():
    print(f"Results for case: {case}")
    display(df)
```

```
Results for case: 10k0.01nS20
```

	True λ	Estimated λ	λ CI Lower	λ CI Upper	True p	Estimated p	p CI Lower	p CI Upper
0	10000	9977.50	9934.54	10020.46	0.01	0.0100	0.0096	0.0105
1	10000	9986.90	9961.30	10012.50	0.01	0.0098	0.0094	0.0103
2	10000	9993.40	9942.23	10044.57	0.01	0.0101	0.0096	0.0105
3	10000	9975.75	9935.21	10016.29	0.01	0.0102	0.0098	0.0107
4	10000	9999.45	9954.10	10044.80	0.01	0.0099	0.0094	0.0103
5	10000	9980.05	9935.53	10024.57	0.01	0.0100	0.0095	0.0104
6	10000	9984.60	9933.96	10035.24	0.01	0.0099	0.0095	0.0104
7	10000	9973.30	9935.84	10010.76	0.01	0.0096	0.0091	0.0100
8	10000	10002.40	9955.51	10049.29	0.01	0.0098	0.0094	0.0103
9	10000	9978.55	9939.32	10017.78	0.01	0.0101	0.0097	0.0105

Results for case: 100k0.01nS20

	True λ	Estimated λ	λ CI Lower	λ CI Upper	True p	Estimated p	p CI Lower	p CI Upper
0	100000	100033.90	99891.56	100176.24	0.01	0.0100	0.0098	0.0101
1	100000	99906.40	99786.55	100026.25	0.01	0.0099	0.0098	0.0101
2	100000	100190.20	100088.44	100291.96	0.01	0.0101	0.0099	0.0102
3	100000	100046.95	99939.18	100154.72	0.01	0.0100	0.0099	0.0101
4	100000	99978.80	99858.32	100099.28	0.01	0.0100	0.0099	0.0102
5	100000	99928.00	99782.04	100073.96	0.01	0.0100	0.0099	0.0101
6	100000	100062.70	99932.80	100192.60	0.01	0.0101	0.0100	0.0102
7	100000	100122.35	99980.48	100264.22	0.01	0.0100	0.0099	0.0102
8	100000	100015.80	99897.23	100134.37	0.01	0.0100	0.0098	0.0101
9	100000	100143.95	100038.57	100249.33	0.01	0.0100	0.0099	0.0102

Results for case: 100k0.02nS20

	True λ	Estimated λ	λ CI Lower	λ CI Upper	True p	Estimated p	p CI Lower	p CI Upper
0	100000	100024.80	99867.91	100181.69	0.02	0.0200	0.0198	0.0202
1	100000	100116.40	99970.34	100262.46	0.02	0.0201	0.0199	0.0202
2	100000	100068.55	99922.40	100214.70	0.02	0.0201	0.0199	0.0203
3	100000	100042.55	99902.13	100182.97	0.02	0.0200	0.0198	0.0202
4	100000	100074.25	99923.54	100224.96	0.02	0.0202	0.0200	0.0204
5	100000	99950.15	99810.00	100090.30	0.02	0.0201	0.0199	0.0203
6	100000	99861.10	99766.39	99955.81	0.02	0.0201	0.0199	0.0203
7	100000	100006.40	99870.86	100141.94	0.02	0.0199	0.0197	0.0201
8	100000	99908.15	99805.01	100011.29	0.02	0.0201	0.0199	0.0203
9	100000	100034.95	99881.03	100188.87	0.02	0.0199	0.0197	0.0201

Results for case: 10k0.02nS20

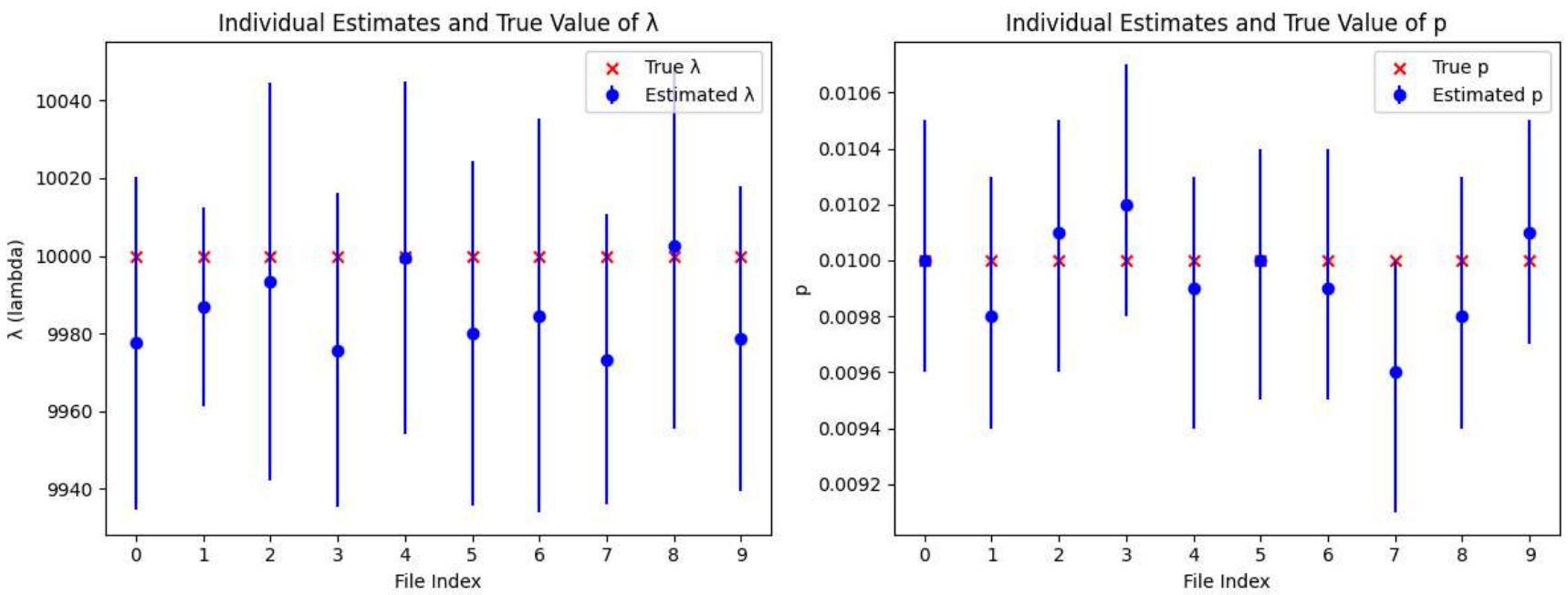
	True λ	Estimated λ	λ CI Lower	λ CI Upper	True p	Estimated p	p CI Lower	p CI Upper
0	10000	9989.95	9949.34	10030.56	0.02	0.0196	0.0190	0.0202
1	10000	10037.55	9995.85	10079.25	0.02	0.0201	0.0195	0.0207
2	10000	9983.70	9945.43	10021.97	0.02	0.0192	0.0186	0.0198
3	10000	9965.75	9917.21	10014.29	0.02	0.0199	0.0193	0.0205
4	10000	9999.65	9949.39	10049.91	0.02	0.0198	0.0192	0.0204
5	10000	9990.20	9950.08	10030.32	0.02	0.0199	0.0193	0.0205
6	10000	10001.50	9961.73	10041.27	0.02	0.0199	0.0192	0.0205
7	10000	10023.40	9980.64	10066.16	0.02	0.0206	0.0200	0.0212
8	10000	9995.05	9952.98	10037.12	0.02	0.0199	0.0193	0.0206
9	10000	9983.00	9949.38	10016.62	0.02	0.0199	0.0193	0.0205

- Plotting λ and p estimates along with true values for each case

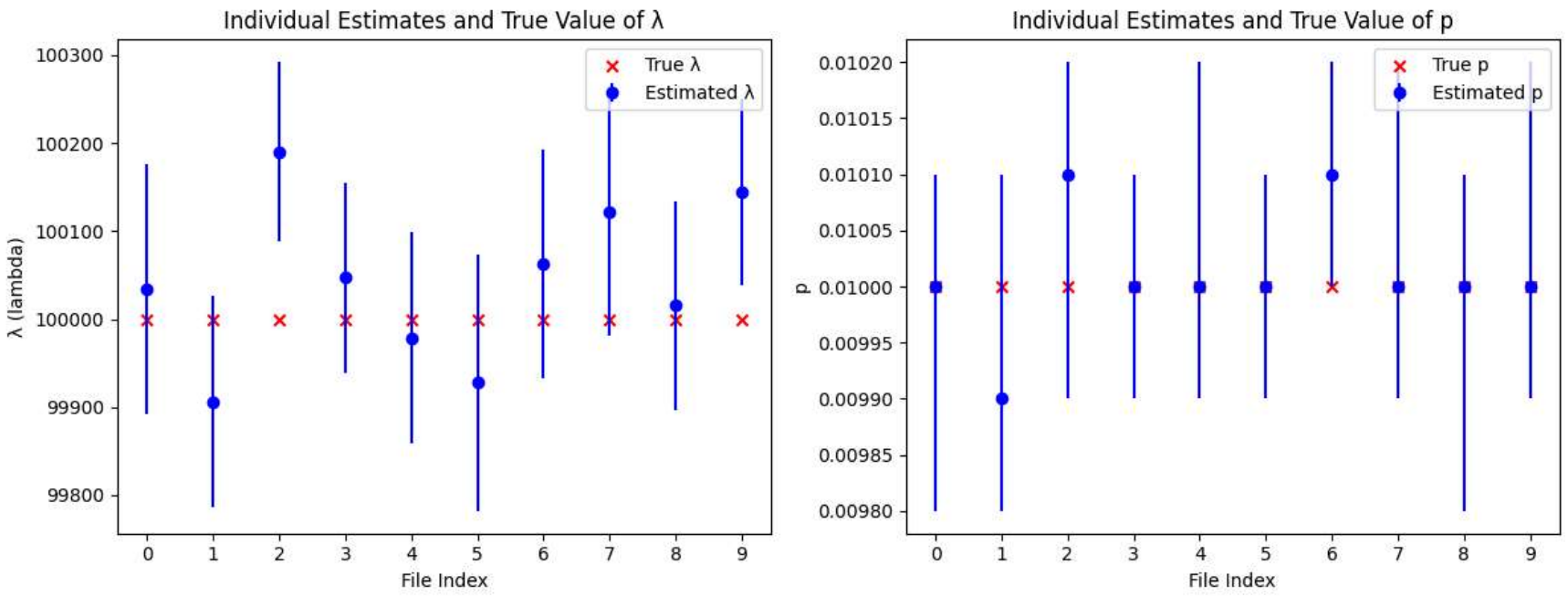
In [98]:
```
for case, df in all_results.items():
    plt.figure(figsize=(12, 5))
    plt.suptitle(f"Estimates for Case: {case}")

    # Plot λ estimates
    plt.subplot(1, 2, 1)
    for idx, row in df.iterrows():
        plt.errorbar(idx, row['Estimated λ'],
                     yerr=[[row['Estimated λ'] - row['λ CI Lower']],
                           [row['λ CI Upper'] - row['Estimated λ']]],
                     fmt='o', color='blue', label='Estimated λ' if idx == 0 else "")
        plt.scatter(idx, row['True λ'], color='red', marker='x', label='True λ' if idx == 0 else "")
```

```
plt.xticks(range(len(df)), df.index.astype(str))
plt.xlabel('File Index')
plt.ylabel('λ (lambda)')
plt.title('Individual Estimates and True Value of λ')
plt.legend()

# Plot p estimates
plt.subplot(1, 2, 2)
for idx, row in df.iterrows():
    plt.errorbar(idx, row['Estimated p'],
                 yerr=[[row['Estimated p'] - row['p CI Lower']],
                       [row['p CI Upper'] - row['Estimated p']]],
                 fmt='o', color='blue', label='Estimated p' if idx == 0 else "")
    plt.scatter(idx, row['True p'], color='red', marker='x', label='True p' if idx == 0 else "")

plt.xticks(range(len(df)), df.index.astype(str))
plt.xlabel('File Index')
plt.ylabel('p')
plt.title('Individual Estimates and True Value of p')
plt.legend()

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

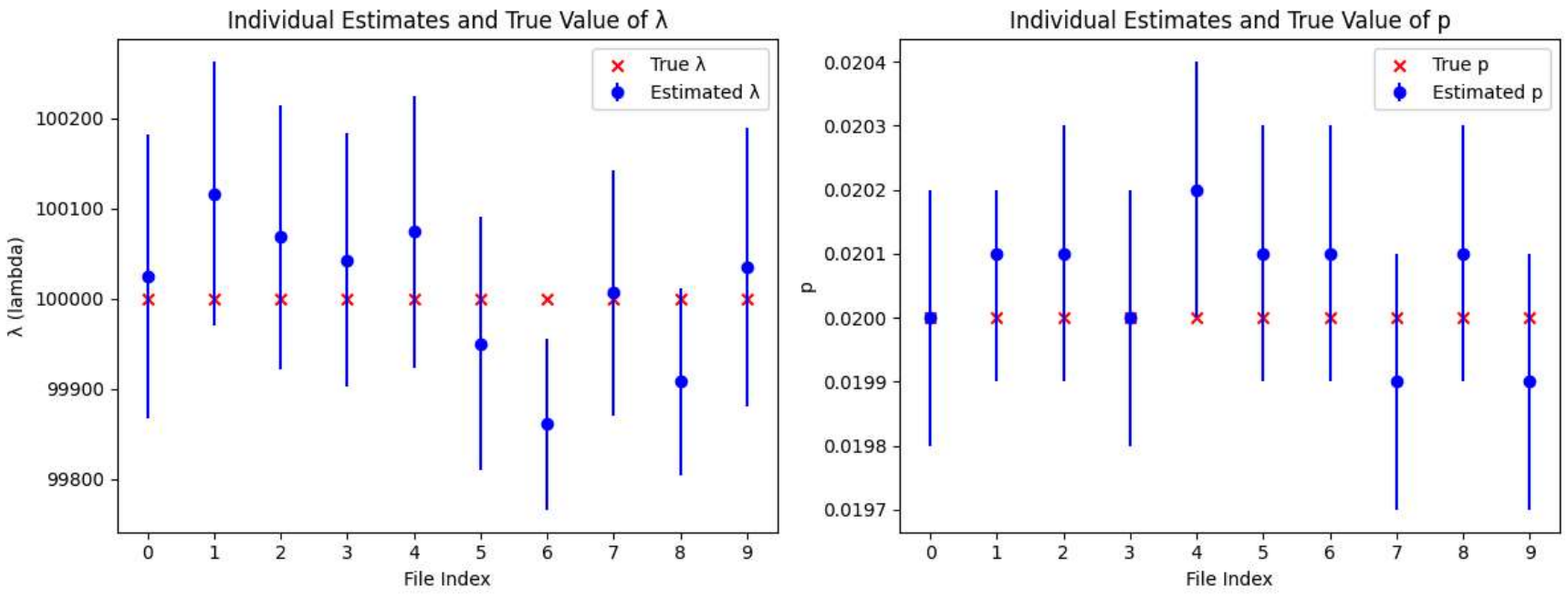
Estimates for Case: 10k0.01nS20

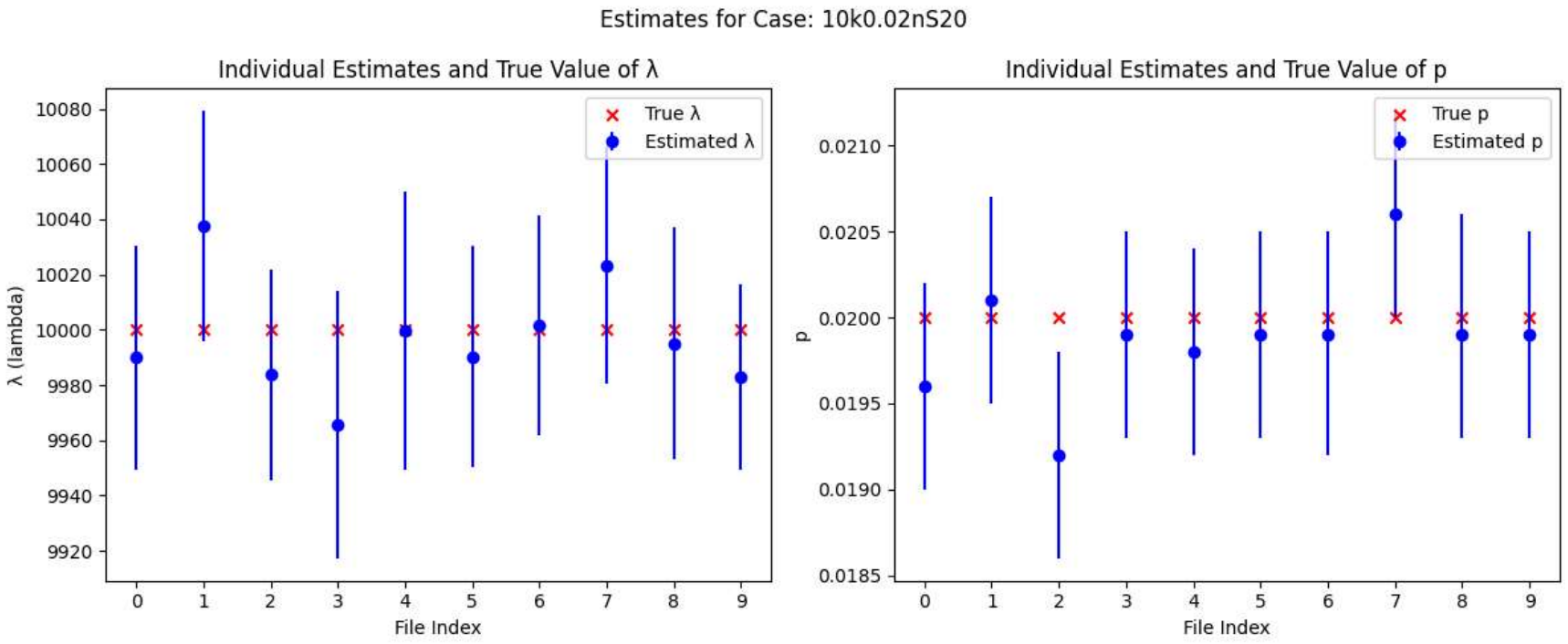


Estimates for Case: 100k0.01nS20



Estimates for Case: 100k0.02nS20





- For N = 20 obtaining α from estimated λ and p values

In [99]:
```
#From Part-1
Omega = 10
a_values = [10, 0, -10, -20, -30, -100]
target_prob = 0.01
tolerance = 1e-6

def calc_mean_variance_S(alpha, lambda_, p, Omega):
    mean_S = alpha * lambda_ - Omega * lambda_ * p
    var_X = lambda_
    var_Y_given_X = lambda_ * p * (1 - p)
    cov_XY = lambda_ * p
    var_S = (alpha ** 2) * var_X + (Omega ** 2) * var_Y_given_X + 2 * cov_XY * Omega * alpha
    return mean_S, var_S

def find_min_alpha_binary_search(lambda_, p, Omega, a_values, target_prob, tolerance):
    alphas = []
    for a in a_values:
        low, high = 0, 3
        while high - low > tolerance:
            alpha = (low + high) / 2
            mean_S, var_S = calc_mean_variance_S(alpha, lambda_, p, Omega)
            std_dev_S = np.sqrt(var_S)
            prob = norm.cdf(a, loc=mean_S, scale=std_dev_S)

            if prob < target_prob:
                high = alpha
            else:
                low = alpha

        alphas.append((low + high) / 2)
    return alphas
```

In [100…
```
# True λ and p values
true_lambda_values = [100000]
true_p_values = [0.02]

# estimated λ and p values from 100k0.02nS20 case
estimated_results = all_results['100k0.02nS20']
estimated_lambda = estimated_results['Estimated λ'].mean()
estimated_p = estimated_results['Estimated p'].mean()

# Calculating the alpha values for both true and estimated values
results_alpha = []

# True λ and p
for lambda_ in true_lambda_values:
    for p in true_p_values:
        alphas = find_min_alpha_binary_search(lambda_, p, Omega, a_values, target_prob, tolerance)
        for a, alpha in zip(a_values, alphas):
            results_alpha.append({
                'λ': lambda_,
                'p': p,
                'a': a,
                'Minimum α (True Values)': alpha,
                'Type': 'True'
            })

# Estimated λ and p
estimated_alphas = find_min_alpha_binary_search(estimated_lambda, estimated_p, Omega, a_values, target_prob, tolerance)
for a, alpha in zip(a_values, estimated_alphas):
    results_alpha.append({
        'λ': estimated_lambda,
        'p': estimated_p,
        'a': a,
        'Minimum α (Estimated Values)': alpha,
        'Type': 'Estimated'
    })

alpha_results_df = pd.DataFrame(results_alpha)
print("Alpha Results Table:")
display(alpha_results_df)
```

```
#NOTE: Here NaN values mean that value is not computed for that case. See the 'Type' for clarity.
```

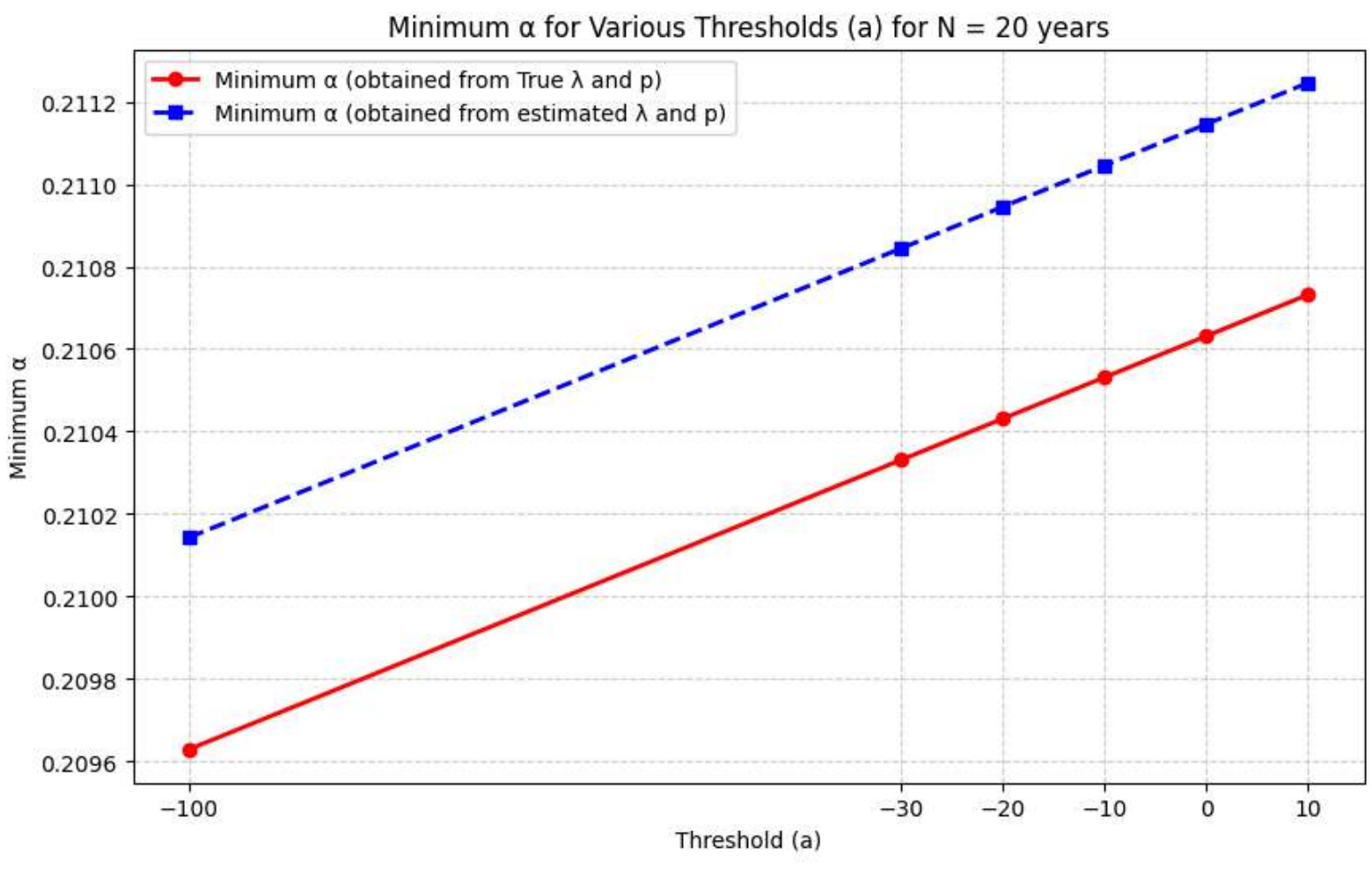
Alpha Results Table:

	λ	p	a	Minimum α (True Values)	Type	Minimum α (Estimated Values)
0	100000.00	0.02000	10	0.210732	True	NaN
1	100000.00	0.02000	0	0.210632	True	NaN
2	100000.00	0.02000	-10	0.210532	True	NaN
3	100000.00	0.02000	-20	0.210431	True	NaN
4	100000.00	0.02000	-30	0.210331	True	NaN
5	100000.00	0.02000	-100	0.209630	True	NaN
6	100008.73	0.02005	10	NaN	Estimated	0.211245
7	100008.73	0.02005	0	NaN	Estimated	0.211145
8	100008.73	0.02005	-10	NaN	Estimated	0.211045
9	100008.73	0.02005	-20	NaN	Estimated	0.210945
10	100008.73	0.02005	-30	NaN	Estimated	0.210844
11	100008.73	0.02005	-100	NaN	Estimated	0.210143

- Plotting alpha results for both true and estimated values, N = 20 years

In [101…

```
plt.figure(figsize=(10, 6))

true_values_df = alpha_results_df[alpha_results_df['Type'] == 'True']
estimated_values_df = alpha_results_df[alpha_results_df['Type'] == 'Estimated']

plt.plot(
    true_values_df['a'],
    true_values_df['Minimum α (True Values)'],
    color='red',
    marker='o',
    linestyle='-',
    linewidth=2,
    markersize=6,
    label='Minimum α (obtained from True λ and p)'
)

plt.plot(
    estimated_values_df['a'],
    estimated_values_df['Minimum α (Estimated Values)'],
    color='blue',
    marker='s',
    linestyle='--',
    linewidth=2,
    markersize=6,
    label='Minimum α (obtained from estimated λ and p)'
)

plt.xlabel('Threshold (a)')
plt.ylabel('Minimum α')
plt.xticks([-100, -30, -20, -10, 0, 10])
plt.title('Minimum α for Various Thresholds (a) for N = 20 years')
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.show()
```



- For N = 100 estimating λ and p values and obtaining α from them

In [102…

```
filename = '100k0.02ns100.csv'
data = pd.read_csv(filename)

# Estimating lambda and p with confidence intervals using previously defined function
X = data['Insurers']
Y = data['Claims']

# Estimating lambda
lambda_estimate, lambda_conf_interval = calculate_point_and_ci(X)

# Estimating p
total_claims = Y.sum()
total_insurers = X.sum()
p_estimate = total_claims / total_insurers
p_ci_lower = p_estimate - 1.96 * np.sqrt((p_estimate * (1 - p_estimate)) / total_insurers)
p_ci_upper = p_estimate + 1.96 * np.sqrt((p_estimate * (1 - p_estimate)) / total_insurers)
p_conf_interval = (p_ci_lower, p_ci_upper)

# Calculating alpha values for estimated lambda and p similarly as done for N = 20 using the code from Part - 1.
alphas_estimated = find_min_alpha_binary_search(lambda_estimate, p_estimate, Omega, a_values, target_prob, tolerance)

# Tabulating results
alpha_results_estimated = pd.DataFrame({
    'Threshold (a)': a_values,
    'Lambda (Estimate)': [lambda_estimate] * len(a_values),
    'Lambda CI Lower': [lambda_conf_interval[0]] * len(a_values),
    'Lambda CI Upper': [lambda_conf_interval[1]] * len(a_values),
    'p (Estimate)': [p_estimate] * len(a_values),
    'p CI Lower': [p_ci_lower] * len(a_values),
    'p CI Upper': [p_ci_upper] * len(a_values),
    'Minimum α (Estimated Values)': alphas_estimated
})


print("Alpha Results Table for 100k0.02ns100.csv (Estimated λ and p):")
display(alpha_results_estimated)
```

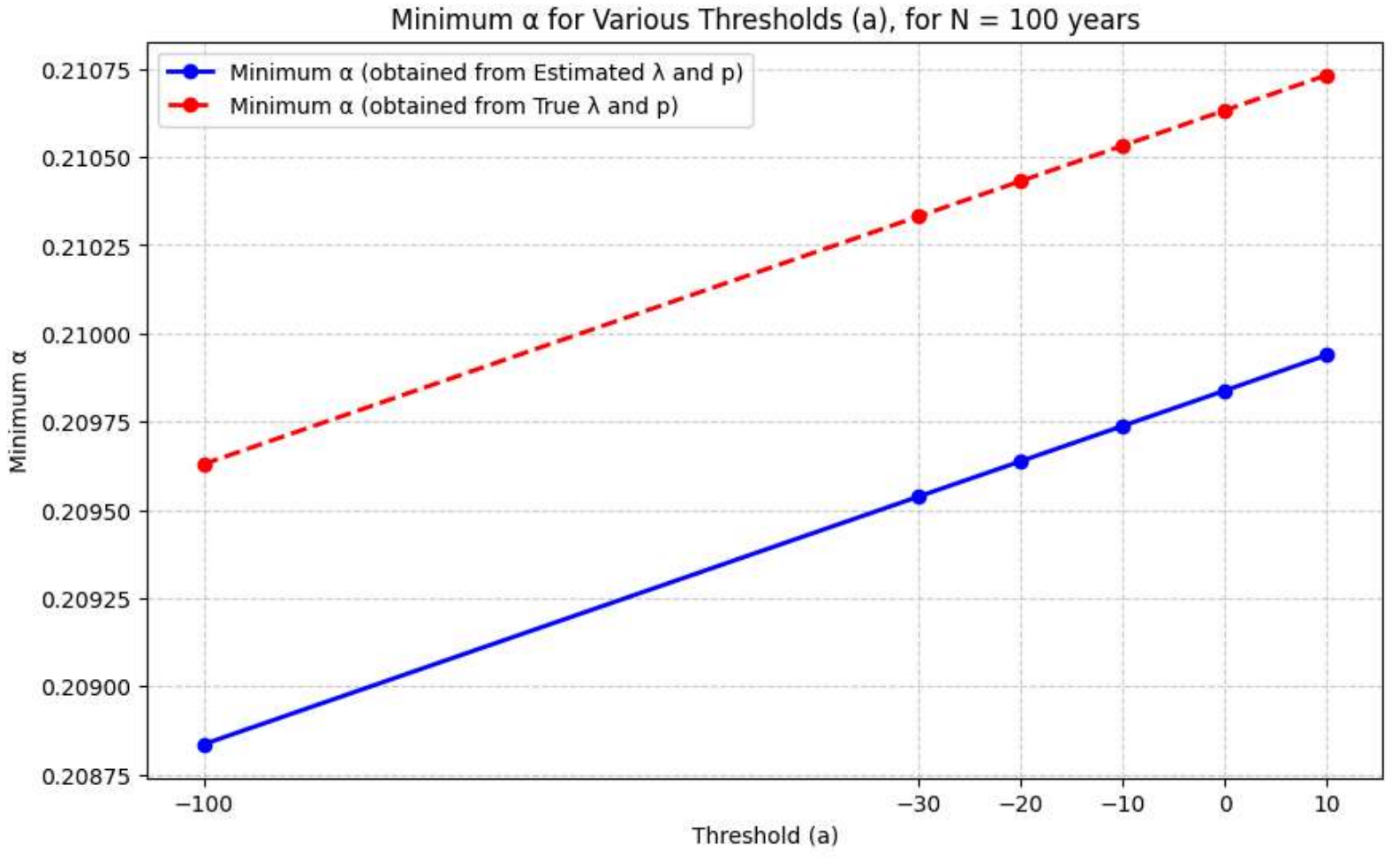
```
Alpha Results Table for 100k0.02ns100.csv (Estimated λ and p):
```

	Threshold (a)	Lambda (Estimate)	Lambda CI Lower	Lambda CI Upper	p (Estimate)	p CI Lower	p CI Upper	Minimum α (Estimated Values)
0	10	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209939
1	0	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209838
2	-10	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209738
3	-20	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209638
4	-30	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.209537
5	-100	99983.96	99929.489651	100038.430349	0.019923	0.019836	0.020009	0.208836

- Plotting alpha results for both true and estimated values, N = 100 years

In [103…

```
plt.figure(figsize=(10, 6))
plt.plot(
    alpha_results_estimated['Threshold (a)'],
    alpha_results_estimated['Minimum α (Estimated Values)'],
    color='blue', marker='o', linestyle='-', linewidth=2, markersize=6,
    label='Minimum α (obtained from Estimated λ and p)'
)
plt.plot(
    true_values_df['a'],
    true_values_df['Minimum α (True Values)'],
    color='red',
    marker='o',
    linestyle='--',
    linewidth=2,
    markersize=6,
    label='Minimum α (obtained from True λ and p)'
)
plt.xlabel('Threshold (a)')
plt.ylabel('Minimum α')
plt.xticks([-100, -30, -20, -10, 0, 10])
plt.title('Minimum α for Various Thresholds (a), for N = 100 years')
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.show()
```


Minimum α for Various Thresholds (a), for N = 100 years
Minimum α (obtained from Estimated λ and p)
Minimum α (obtained from True λ and p)
Minimum α
0.21075
0.21050
0.21025
0.21000
0.20975
0.20950
0.20925
0.20900
0.20875
−100
−30
−20
−10
0
10
Threshold (a)

Thank You