In [ ]:
```python
import numpy as np
from scipy import stats
from scipy.stats import chi2_contingency
from scipy.special import beta, betaln
import matplotlib.pyplot as plt
import pandas as pd
from collections import defaultdict

# パラメータ設定
np.random.seed(42)  # 再現性のため

# 基本パラメータ
pA = 0.1  # 対照群の真の成功率
pB_effect = 0.12  # 介入群の真の成功率（効果あり）
pB_no_effect = 0.1  # 介入群の真の成功率（効果なし）
alpha = 0.05  # 有意水準（片側）
N_max = 3026  # 最大サンプルサイズ
n_simulations = 10000  # 外側のシミュレーション回数
n_mcmc = 10000  # 内側のモンテカルロ回数（ベイズ用）
bf_threshold = 1.05  # ベイズファクターの閾値（$BF_{10} \geq$ bf_threshold）

print(f"パラメータ設定:")
print(f"  対照群成功率: {pA}")
print(f"  介入群成功率（効果あり）: {pB_effect}")
print(f"  介入群成功率（効果なし）: {pB_no_effect}")
print(f"  最大サンプルサイズ: {N_max}")
print(f"  シミュレーション回数: {n_simulations}")
```

```
パラメータ設定:
  対照群成功率: 0.1
  介入群成功率（効果あり）: 0.12
  介入群成功率（効果なし）: 0.1
  最大サンプルサイズ: 3026
  シミュレーション回数: 10000
```

In [ ]:
```python
def generate_data(n, pA, pB):
    """2群のデータを生成"""
    groupA = np.random.binomial(1, pA, n)
    groupB = np.random.binomial(1, pB, n)
    return groupA, groupB

def chi2_test_one_sided(groupA, groupB):
    """片側カイ二乗検定（2×2分割表）"""
    # 成功数と失敗数を計算
    successA = np.sum(groupA)
    failA = len(groupA) - successA
    successB = np.sum(groupB)
    failB = len(groupB) - successB

    # 2×2分割表
    contingency = np.array([[successA, failA], [successB, failB]])

    # カイ二乗検定（Yates補正なし）
    chi2, p_value_two_sided, dof, expected = chi2_contingency(
        contingency, correction=False
    )

    # 片側検定: pB > pA の方向を確認
    if successB / len(groupB) > successA / len(groupA):
        p_value_one_sided = p_value_two_sided / 2
```

```
    else:
        p_value_one_sided = 1 - p_value_two_sided / 2

    return p_value_one_sided < alpha, p_value_one_sided

def bayes_posterior_prob(groupA, groupB, prior_alpha_A, prior_beta_A,
                         prior_alpha_B, prior_beta_B, n_samples=n_mcmc):
    """ベイズ事後分布からP(pB > pA)を計算"""
    # 成功数と失敗数
    successA = np.sum(groupA)
    failA = len(groupA) - successA
    successB = np.sum(groupB)
    failB = len(groupB) - successB

    # 事後分布のパラメータ
    post_alpha_A = prior_alpha_A + successA
    post_beta_A = prior_beta_A + failA
    post_alpha_B = prior_alpha_B + successB
    post_beta_B = prior_beta_B + failB

    # 事後分布からサンプリング
    pA_samples = np.random.beta(post_alpha_A, post_beta_A, n_samples)
    pB_samples = np.random.beta(post_alpha_B, post_beta_B, n_samples)

    # P(pB > pA)を計算
    prob_pB_greater = np.mean(pB_samples > pA_samples)

    return prob_pB_greater

def bayes_factor_10(groupA, groupB, prior_alpha=1, prior_beta=1):
    """BF10を計算 (H1: pA ≠ pB vs H0: pA = pB) """
    # 成功数と失敗数
    successA = np.sum(groupA)
    failA = len(groupA) - successA
    successB = np.sum(groupB)
    failB = len(groupB) - successB

    # H1: pA ≠ pB(独立なベータ事前分布)
    # 周辺尤度 = Beta(successA + alpha, failA + beta) * Beta(successB + alpha, failB + beta)
    log_marginal_H1 = (betaln(successA + prior_alpha, failA + prior_beta) +
                       betaln(successB + prior_alpha, failB + prior_beta) -
                       betaln(prior_alpha, prior_beta) -
                       betaln(prior_alpha, prior_beta))

    # H0: pA = pB = p(共通のベータ事前分布)
    # 周辺尤度 = Beta(successA + successB + alpha, failA + failB + beta)
    log_marginal_H0 = (betaln(successA + successB + prior_alpha,
                              failA + failB + prior_beta) -
                       betaln(prior_alpha, prior_beta))

    # BF10 = P(data | H1) / P(data | H0)
    log_BF10 = log_marginal_H1 - log_marginal_H0
    BF10 = np.exp(log_BF10)

    return BF10

print("ヘルパー関数を定義しました")
```

```
ヘルパー関数を定義しました
```

In [ ]:
```
print("\n=== 実験1: 検出力の確認（効果あり） ===")
```

```
rejections = 0
for i in range(n_simulations):
    groupA, groupB = generate_data(N_max, pA, pB_effect)
    rejected, _ = chi2_test_one_sided(groupA, groupB)
    if rejected:
        rejections += 1

power = rejections / n_simulations
print(f"検出力: {power:.4f} ({power*100:.2f}%)")
print(f"理論値: 約80%")
```

```
=== 実験1: 検出力の確認（効果あり） ===
検出力: 0.8030 (80.30%)
理論値: 約80%
```

In [ ]:

```
print("\n=== 実験2-1: 通常の検定（ピーキングなし、効果なし） ===")

false_positives = 0
for i in range(n_simulations):
    groupA, groupB = generate_data(N_max, pA, pB_no_effect)
    rejected, _ = chi2_test_one_sided(groupA, groupB)
    if rejected:
        false_positives += 1

type1_error = false_positives / n_simulations
print(f"第一種の過誤: {type1_error:.4f} ({type1_error*100:.2f}%)")
print(f"理論値: 5%")
```

```
=== 実験2-1: 通常の検定（ピーキングなし、効果なし） ===
第一種の過誤: 0.0546 (5.46%)
理論値: 5%
```

In [ ]:

```
print("\n=== 実験2-2: ピーキングあり（効果なし） ===")

# 検定パターン
patterns = {
    "2回検定": [0.5, 1.0],
    "3回検定": [1/3, 2/3, 1.0],
    "4回検定": [1/4, 1/2, 3/4, 1.0]
}

results_peeking = {}
for pattern_name, checkpoints in patterns.items():
    false_positives_fwer = 0
    false_positives_by_checkpoint = defaultdict(int)

    for i in range(n_simulations):
        # データを一度に生成（実際の逐次をシミュレート）
        groupA_full, groupB_full = generate_data(N_max, pA, pB_no_effect)

        rejected_at_any = False
        for checkpoint in checkpoints:
            n_current = int(N_max * checkpoint)
            groupA = groupA_full[:n_current]
            groupB = groupB_full[:n_current]

            rejected, _ = chi2_test_one_sided(groupA, groupB)
            if rejected:
                false_positives_by_checkpoint[checkpoint] += 1
                rejected_at_any = True
                break  # 一度でも有意なら打ち切り
```

```
        if rejected_at_any:
            false_positives_fwer += 1

    fwer = false_positives_fwer / n_simulations
    results_peeking[pattern_name] = {
        'FWER': fwer,
        'by_checkpoint': dict(false_positives_by_checkpoint)
    }

    print(f"\n{pattern_name}:")
    print(f"  FWER: {fwer:.4f} ({fwer*100:.2f}%)")
    print(f"  理論上限（独立仮定）: {1 - (1 - alpha)**len(checkpoints):.4f}")
```

```
=== 実験2-2: ピーキングあり（効果なし） ===

2回検定:
  FWER: 0.0837 (8.37%)
  理論上限（独立仮定）: 0.0975

3回検定:
  FWER: 0.1030 (10.30%)
  理論上限（独立仮定）: 0.1426

4回検定:
  FWER: 0.1116 (11.16%)
  理論上限（独立仮定）: 0.1855
```

In [ ]:

```
print("\n=== 実験3-1: 固定nのベイズ検定（効果あり） ===")

wins = 0
for i in range(n_simulations):
    groupA, groupB = generate_data(N_max, pA, pB_effect)
    prob = bayes_posterior_prob(groupA, groupB, 1, 1, 1, 1)
    if prob >= 0.95:
        wins += 1

    # 進捗表示（100回ごと、または最後の1回）
    if (i + 1) % 100 == 0 or (i + 1) == n_simulations:
        progress = (i + 1) / n_simulations * 100
        print(f"進捗: {progress:.1f}% ({i + 1}/{n_simulations})", end='\r')

print()  # 改行を追加
power_bayes_fixed = wins / n_simulations
print(f"検出力（ベイズ固定n）: {power_bayes_fixed:.4f} ({power_bayes_fixed*100:.2f}%)")
print(f"頻度主義の検出力: {power:.4f} ({power*100:.2f}%)")
```

```
=== 実験3-1: 固定nのベイズ検定（効果あり） ===
進捗: 100.0% (10000/10000)
検出力（ベイズ固定n）: 0.8076 (80.76%)
頻度主義の検出力: 0.8030 (80.30%)
```

In [ ]:

```
print("\n=== 実験4-1: 固定nのベイズ検定（効果なし） ===")

false_wins = 0
for i in range(n_simulations):
    groupA, groupB = generate_data(N_max, pA, pB_no_effect)
    prob = bayes_posterior_prob(groupA, groupB, 1, 1, 1, 1)
    if prob >= 0.95:
        false_wins += 1

    # 進捗表示（100回ごと、または最後の1回）
    if (i + 1) % 100 == 0 or (i + 1) == n_simulations:
        progress = (i + 1) / n_simulations * 100
```

```
        print(f"進捗: {progress:.1f}% ({i + 1}/{n_simulations})", end='\r')

print()  # 改行を追加
type1_error_bayes_fixed = false_wins / n_simulations
print(f"判定ミス率（ベイズ固定n）: {type1_error_bayes_fixed:.4f} ({type1_error_bayes_fixed*
print(f"頻度主義の第一種過誤: {type1_error:.4f} ({type1_error*100:.2f}%)")
```

```
=== 実験4-1: 固定nのベイズ検定（効果なし） ===
進捗: 100.0% (10000/10000)
判定ミス率（ベイズ固定n）: 0.0488 (4.88%)
頻度主義の第一種過誤: 0.0546 (5.46%)
```

In [ ]:

```
print("\n=== 実験4-3: 逐次ベイズ検定（効果なし） ===")

checkpoints = [1/4, 1/2, 3/4, 1.0]
false_win_times = []
false_wins_sequential = 0

for i in range(n_simulations):
    groupA_full, groupB_full = generate_data(N_max, pA, pB_no_effect)

    false_won = False
    for checkpoint in checkpoints:
        n_current = int(N_max * checkpoint)
        groupA = groupA_full[:n_current]
        groupB = groupB_full[:n_current]

        prob = bayes_posterior_prob(groupA, groupB, 1, 1, 1, 1)
        if prob >= 0.95:
            false_win_times.append(n_current)
            false_wins_sequential += 1
            false_won = True
            break

    if not false_won:
        false_win_times.append(N_max)  # N_maxに達しても勝てなかった

    # 進捗表示（100回ごと、または最後の1回）
    if (i + 1) % 100 == 0 or (i + 1) == n_simulations:
        progress = (i + 1) / n_simulations * 100
        print(f"進捗: {progress:.1f}% ({i + 1}/{n_simulations})", end='\r')

print()  # 改行を追加
fwer_bayes_sequential = false_wins_sequential / n_simulations
asn_no_effect = np.mean(false_win_times)

print(f"FWER（逐次ベイズ）: {fwer_bayes_sequential:.4f} ({fwer_bayes_sequential*100:.2f}%)"
print(f"ASN（効果なしの場合）: {asn_no_effect:.0f}")
print(f"ASNの中央値: {np.median(false_win_times):.0f}")
```

```
=== 実験4-3: 逐次ベイズ検定（効果なし） ===
進捗: 100.0% (10000/10000)
FWER（逐次ベイズ）: 0.1180 (11.80%)
ASN（効果なしの場合）: 2850
ASNの中央値: 3026
```