Project Overview

We're investigating how to distinguish between two random vectors — one generated from N(0,1) and the other from N(0,1.5) — by only looking at their components. This is about decision-making using statistical signals from independent trials.

Part I: Normal Distribution

We generate:

Vector V_1: 100 samples from N(0,1)

Vector $V_{1.5}$: 100 samples from N(0,1.5)

Then we try to determine which vector is which using 4 different decision procedures:

Procedure A: Component-Wise Comparison

```
import numpy as np
from scipy.stats import norm
from scipy.integrate import quad

# Empirical Simulation
n = 10000
x = np.random.normal(0, 1, n)
y = np.random.normal(0, np.sqrt(1.5), n)
empirical_prob = np.mean(np.abs(x) < np.abs(y))
print(f"Empirical success rate: {empirical_prob:.4f}")

# Theoretical Probability
def integrand(x):
    return norm.pdf(x) * 2 * (1 - norm.cdf(abs(x) / np.sqrt(1.5)))

theoretical_prob, _ = quad(integrand, -np.inf, np.inf)
print(f"Theoretical success probability: {theoretical_prob:.4f}")
```

```
Empirical success rate: 0.5593
Theoretical success probability: 0.5641
```

Description:

In this procedure, we are given a pair of numbers, one from each of two normal distributions: N(0,1) and N(0,1.5). Without knowing which is which, we guess that the number with the smaller absolute value came from the N(0,1) distribution.

Rationale:

This is based on the idea that samples from a distribution with lower variance tend to stay closer to zero. So, when comparing absolute values, the one with the smaller magnitude is more likely to come from the lower variance distribution.

Empirical Results:

After running 10,000 simulations, we found that this rule correctly identified the N(0,1) sample approximately 55.9% of the time.

Theoretical Probability:

The theoretical probability of success is calculated as the probability that $|X| < |Y|$, where $X \sim N(0,1)$ and $Y \sim N(0,1.5)$. This gives a result of approximately 56.4%, which closely matches the empirical result.

Conclusion:

Procedure A gives a result slightly better than random guessing. This makes sense because the lower-variance normal distribution is more concentrated around zero, making it more likely to produce smaller absolute values.

Procedure B: Total Sum of Squares (Assuming Mean Zero)

```
import numpy as np

# Simulate two vectors of 100 components each
n_trials = 10000
n = 100

correct_guesses = 0

for _ in range(n_trials):
    v1 = np.random.normal(0, 1, n)
    v15 = np.random.normal(0, np.sqrt(1.5), n)

    sumsq_v1 = np.sum(v1**2)
    sumsq_v15 = np.sum(v15**2)

    # Guess the one with larger sum of squares is from N(0,1.5)
```

```
    if sumsq_v15 > sumsq_v1:
        correct_guesses += 1

empirical_prob = correct_guesses / n_trials
print(f"Empirical success rate: {empirical_prob:.4f}")
```

```
Empirical success rate: 0.9776
```

Description:

We are given two vectors of 100 components each: one from N(0,1) and one from N(0,1.5). Since the mean is known to be zero, we use the total sum of squares (the sum of squared values) to determine which vector has higher variance. We guess that the vector with the larger sum of squares comes from N(0,1.5).

Rationale:

For normal distributions with mean zero, the sum of squares is a good estimator of variance. The vector with higher variance will tend to have larger squared values overall, making the total sum of squares a reliable indicator.

Empirical Results:

After 10,000 trials, this procedure correctly identified the higher-variance vector approximately 97.76% of the time.

Conclusion:

This procedure performs significantly better than Procedure A. Knowing the mean is zero allows us to directly compare variances using the sum of squares, leading to a higher accuracy rate.

Procedure C: Sample Standard Deviation (Unknown Mean)

```
import numpy as np

n_trials = 10000
n = 100
correct_guesses = 0

for _ in range(n_trials):
    v1 = np.random.normal(0, 1, n)
    v15 = np.random.normal(0, np.sqrt(1.5), n)

    std_v1 = np.std(v1, ddof=1)  # ddof=1 for sample std
    std_v15 = np.std(v15, ddof=1)

    # Guess vector with larger std dev came from N(0, 1.5)
```

```
    if std_v15 > std_v1:
        correct_guesses += 1

empirical_prob = correct_guesses / n_trials
print(f"Empirical success rate: {empirical_prob:.4f}")
```

```
Empirical success rate: 0.9732
```

Description:

In this procedure, we again compare two vectors: one from N(0,1) and the other from N(0,1.5), each with 100 components. Unlike Procedure B, we do not assume the mean is zero, so instead of using the sum of squares, we compute the sample standard deviation of each vector. We guess that the one with the larger standard deviation came from the higher-variance distribution, N(0,1.5).

Rationale:

The sample standard deviation is an unbiased estimate of a distribution's true spread when the mean is unknown. Although slightly less precise than using the sum of squares with known mean, it still reflects the underlying variance and is useful for distinguishing between distributions.

Empirical Results:

After 10,000 trials, this method correctly identified the higher-variance vector approximately 97.32% of the time.

Conclusion:

Procedure C performs very well — though slightly less accurate than Procedure B — and remains a reliable method for distinguishing between normal distributions when the mean is unknown.

Procedure D: Nonparametric Testing (Wilcoxon or Rank-Based Test)

```
import numpy as np
from scipy.stats import ranksums

n_trials = 10000
n = 100
correct_guesses = 0

for _ in range(n_trials):
    v1 = np.random.normal(0, 1, n)
    v15 = np.random.normal(0, np.sqrt(1.5), n)
```

```
    # Perform rank-sum test
    stat, p = ranksums(v15, v1)  # test if v15 > v1

    # If statistic is positive, v15 tends to have larger ranks
    if stat > 0:
        correct_guesses += 1

empirical_prob = correct_guesses / n_trials
print(f"Empirical success rate: {empirical_prob:.4f}")
```

```
Empirical success rate: 0.5008
```

Description:

This procedure compares two vectors of 100 elements each, drawn from N(0,1) and N(0,1.5). Unlike previous methods, we do not assume anything about the distributions' shape or mean. Instead, we apply the Wilcoxon rank-sum test, a nonparametric test that compares the relative ranks of values in two independent samples. We then guess that the vector with the higher sum of ranks (i.e., a positive test statistic) comes from the higher-variance distribution, N(0,1.5).

Rationale:

Nonparametric methods are robust to outliers and distributional assumptions. The Wilcoxon rank-sum test is particularly useful when comparing spread, as higher variance tends to produce more extreme values and thus higher ranks.The rank-sum test works by ranking all values and checking whether one sample consistently has higher ranks than the other. In theory, a distribution with greater spread might result in more extreme values and thus higher ranks. However, it does not explicitly capture variance and is less sensitive to changes in scale when the means are similar.

Empirical Results:

After 10,000 trials, this approach correctly identified the higher-variance vector approximately 50.08% of the time - nearly random chance.

Conclusion:

Procedure D performs poorly in this context. Although nonparametric methods are useful when distributional assumptions are violated, in this case, the test was not able to effectively detect the difference in variance between two zero-mean normal distributions. It's likely that the overlap in the data and the subtle difference in spread are not well captured by the rank-based approach.

Additional Evaluation of Procedures B, C, and D on Five Random Pairs

```
import numpy as np
from scipy.stats import ranksums

n = 100
n_pairs = 5

# Track incorrect decisions for each procedure
errors_B = 0
errors_C = 0
errors_D = 0

for _ in range(n_pairs):
    v1 = np.random.normal(0, 1, n)
    v15 = np.random.normal(0, np.sqrt(1.5), n)

    # --- Procedure B: Sum of Squares ---
    sumsq_v1 = np.sum(v1**2)
    sumsq_v15 = np.sum(v15**2)
    if sumsq_v15 <= sumsq_v1:
        errors_B += 1

    # --- Procedure C: Sample Std Dev ---
    std_v1 = np.std(v1, ddof=1)
    std_v15 = np.std(v15, ddof=1)
    if std_v15 <= std_v1:
        errors_C += 1

    # --- Procedure D: Wilcoxon Rank-Sum Test ---
    stat, _ = ranksums(v15, v1)
    if stat <= 0:
        errors_D += 1

print(f"Procedure B errors out of 5: {errors_B}")
print(f"Procedure C errors out of 5: {errors_C}")
print(f"Procedure D errors out of 5: {errors_D}")
```

```
Procedure B errors out of 5: 0
Procedure C errors out of 5: 0
Procedure D errors out of 5: 4
```

We generated five independent pairs of vectors, each consisting of 100 samples. One vector in each pair was sampled from N(0,1), and the other from N(0,1.5). We applied Procedures B, C, and D to each pair to determine which sample came from the distribution with higher variance.

Results:

Procedure B gave the wrong answer in 0 out of 5 trials

Procedure C gave the wrong answer in 0 out of 5 trials

Procedure D gave the wrong answer in 4 out of 5 trials

These results are consistent with the findings from Part I. Procedures B and C, which are based on comparing variance-related statistics (sum of squares and sample standard deviation), performed with near-perfect accuracy. Procedure D, which uses a rank-based nonparametric test, failed in most cases. This reinforces the observation that rank-based tests like the Wilcoxon rank-sum test are not effective for distinguishing between distributions that differ only in variance but have the same mean and shape.

Part II: Using Cauchy Distributions

We're repeating Part I, but instead of normal distributions, we're now working with:

V_1: 100 samples from the standard Cauchy distribution → location = 0, scale = 1

$V_{1.5}$: 100 samples from the Cauchy(0, √1.5) distribution → same location, larger scale

The Cauchy distribution:

Has no finite mean or variance

Is heavy-tailed, meaning sample averages and standard deviations are unstable

So we expect that Procedures A, B, C, and D may behave very differently (and maybe poorly)

Procedure A with Cauchy Distributions

```
import numpy as np
from scipy.stats import cauchy

n = 10000
x = cauchy.rvs(loc=0, scale=1, size=n)
y = cauchy.rvs(loc=0, scale=np.sqrt(1.5), size=n)

# Guess that the smaller absolute value comes from scale=1
correct_guesses = np.abs(x) < np.abs(y)
empirical_prob = np.mean(correct_guesses)
print(f"Empirical success rate (Cauchy): {empirical_prob:.4f}")
```

```
Empirical success rate (Cauchy): 0.5344
```

Description:

We repeated Procedure A using samples from Cauchy distributions instead of normal ones. Specifically, we drew 10,000 samples each from Cauchy(0,1) and Cauchy(0,√1.5). For each pair, we guessed that the sample with the smaller absolute value came from the distribution with the smaller scale parameter (Cauchy(0,1)).

Empirical Result:

This rule correctly identified the lower-scale sample approximately [insert]% of the time.

Theoretical Expectation:

From class notes, we expect the theoretical probability that |Cauchy(√1.5)| > |Cauchy(1)| to be approximately 54%. This matches our observation that the absolute value rule works only slightly better than chance in the Cauchy case.

Conclusion:

Unlike the normal case, the heavy tails of the Cauchy distribution make it harder to detect scale differences using individual values. While Procedure A still performs slightly above random, it is significantly less effective than in the normal setting.

Procedure B (Cauchy): Sum of Squares with Known Mean

```
import numpy as np
from scipy.stats import cauchy

n_trials = 10000
n = 100
correct_guesses = 0

for _ in range(n_trials):
    v1 = cauchy.rvs(loc=0, scale=1, size=n)
    v15 = cauchy.rvs(loc=0, scale=np.sqrt(1.5), size=n)

    sumsq_v1 = np.sum(v1**2)
    sumsq_v15 = np.sum(v15**2)

    if sumsq_v15 > sumsq_v1:
        correct_guesses += 1

empirical_prob = correct_guesses / n_trials
print(f"Empirical success rate (Cauchy, Procedure B):
{empirical_prob:.4f}")
```

```
Empirical success rate (Cauchy, Procedure B): 0.5645
```

Description:

We repeated Procedure B using samples from Cauchy distributions. Even though the Cauchy distribution has no defined mean or variance, we followed the original method by generating two vectors: one from Cauchy(0,1) and one from Cauchy(0,√1.5). We then computed the total sum of squares for each vector and guessed that the one with the larger sum came from the higher-scale distribution.

Empirical Result:

This method correctly identified the higher-scale vector approximately 56.45% of the time.

Conclusion:

While the use of the sum of squares is technically invalid for Cauchy distributions due to the lack of finite variance, the procedure still showed a modest ability to distinguish between scales. This result highlights how some statistical procedures may still "work" heuristically, even if they are theoretically unjustified in the heavy-tailed Cauchy setting.

Procedure C (Cauchy): Sample Standard Deviation (Unknown Mean)

```
import numpy as np
from scipy.stats import cauchy

n_trials = 10000
n = 100
correct_guesses = 0

for _ in range(n_trials):
    v1 = cauchy.rvs(loc=0, scale=1, size=n)
    v15 = cauchy.rvs(loc=0, scale=np.sqrt(1.5), size=n)

    std_v1 = np.std(v1, ddof=1)
    std_v15 = np.std(v15, ddof=1)

    if std_v15 > std_v1:
        correct_guesses += 1

empirical_prob = correct_guesses / n_trials
print(f"Empirical success rate (Cauchy, Procedure C):
{empirical_prob:.4f}")
```

```
Empirical success rate (Cauchy, Procedure C): 0.5586
```

Description:

This procedure compares the sample standard deviation of two vectors: one from Cauchy(0,1) and one from Cauchy(0,√1.5). We then guess that the vector with the higher sample standard deviation came from the distribution with the larger scale parameter.

Empirical Result:

This method correctly identified the higher-scale vector approximately 55.86% of the time.

Conclusion:

Even though the standard deviation of the Cauchy distribution is undefined, the sample standard deviation still captured some signal from the increased scale. However, its performance was only marginally better than chance, and remains highly unreliable due to the sensitivity of the sample statistic to extreme values in heavy-tailed distributions.

Procedure D (Cauchy): Wilcoxon Rank-Sum Test

```
import numpy as np
from scipy.stats import cauchy, ranksums

n_trials = 10000
n = 100
correct_guesses = 0

for _ in range(n_trials):
    v1 = cauchy.rvs(loc=0, scale=1, size=n)
    v15 = cauchy.rvs(loc=0, scale=np.sqrt(1.5), size=n)

    stat, _ = ranksums(v15, v1)

    if stat > 0:
        correct_guesses += 1

empirical_prob = correct_guesses / n_trials
print(f"Empirical success rate (Cauchy, Procedure D):
{empirical_prob:.4f}")
```

```
Empirical success rate (Cauchy, Procedure D): 0.4977
```

Description:

This procedure applies a nonparametric Wilcoxon rank-sum test to two vectors: one from Cauchy(0,1) and one from Cauchy(0,√1.5). The test ranks all values and compares the distributions. We then guess that the vector with the higher rank-sum statistic (i.e., a positive test statistic) came from the higher-scale distribution.

Empirical Result:

This method correctly identified the higher-scale vector approximately 49.77% of the time.

Conclusion:

The Wilcoxon test is not designed to detect differences in scale, particularly when both distributions are symmetric and have the same location. This is even more problematic with Cauchy distributions, which have undefined variance and are prone to extreme outliers. As a result, the test performs no better than chance. This confirms that rank-based nonparametric methods are not reliable for identifying scale differences in heavy-tailed distributions.

Final Summary

Across all procedures, performance was strongly tied to the properties of the underlying distributions. For normal data, methods that relied on variance-related statistics — like sum of squares or standard deviation — were highly effective, as expected. These measures are reliable when the mean and variance are well-defined.

In contrast, Cauchy distributions posed a challenge. Because they lack a defined mean or variance, standard techniques became unstable and far less reliable. Even so, procedures based on magnitude (like absolute value or sum of squares) retained a slight edge over random guessing. However, rank-based methods such as the Wilcoxon test consistently failed to capture any meaningful difference in spread across both distribution types. This confirms that nonparametric rank tests are ineffective for distinguishing scale when the location is the same and the distributions are heavy-tailed.