# 1 Chapter 10 Same-Different Judgment

**Problem 10.5** *Refer back to section 10.2.3 on the response distribution in the binary same-different task. Choose $p_{same} = 0.4$, $\mu = 1$, and $\sigma = 1$. We will use two numerical techniques for calculating the probability that the observer will report "same" in a given stimulus condition, $p(\hat{C} = 1 \mid s_1, s_2)$. Note that we need to calculate four numbers, since there are four possible combinations of $s_1$ and $s_2$.*

(a) Use Riemann integration. For both $x_1$ and $x_2$, use a grid from $-5$ to 5 in steps of 0.01. Calculate the four numbers we are looking for.

(b) Use Monte Carlo simulation. Compare. The results should be very similar.

a)
The response probability is given by:

$$p(\hat{C} = 1|s_1, s_2) = \sum_{x_1} \sum_{x_2} p(\hat{C} = 1|x_1, x_2) \cdot p(x_1|s_1) \cdot p(x_2|s_2) \cdot (\Delta x)^2$$

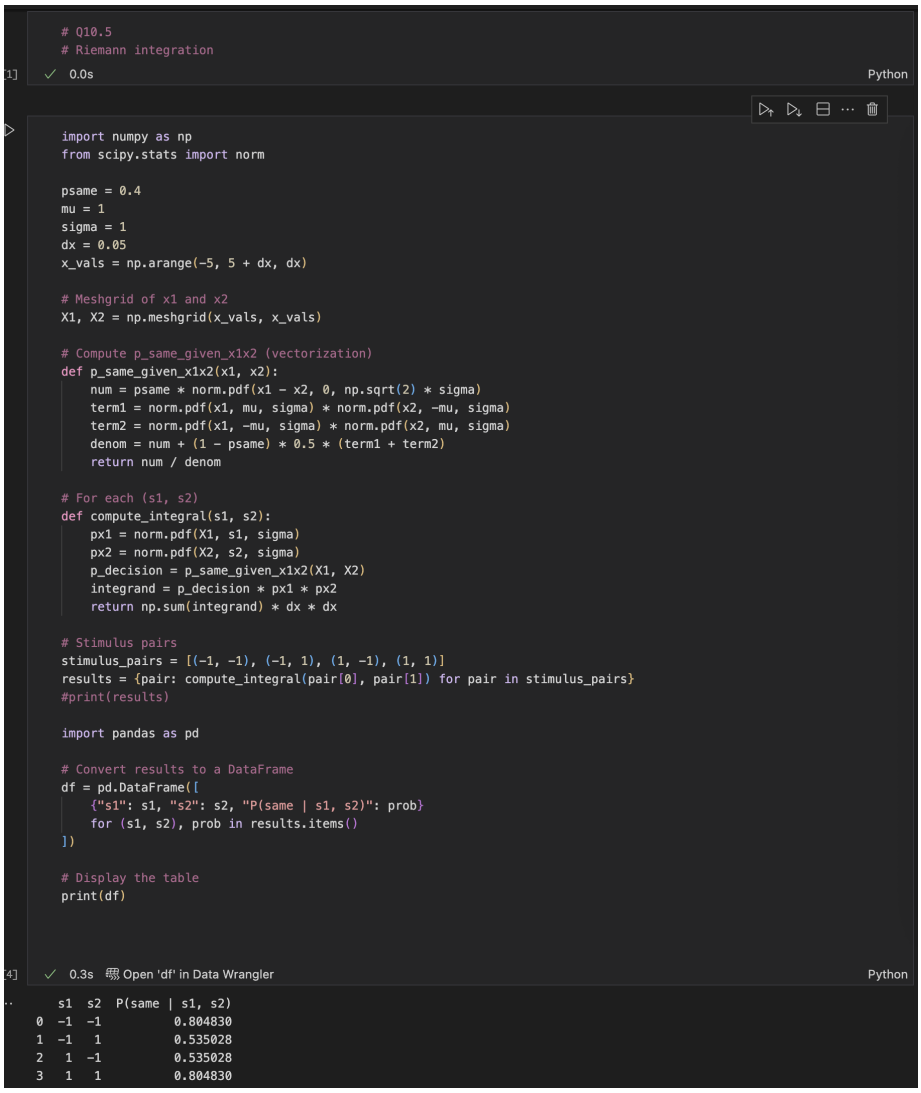
Where,

$$p(\hat{C} = 1|x_1, x_2) = \frac{p_{\text{same}} \cdot \mathcal{N}(x_1 - x_2; 0, 2\sigma^2)}{p_{\text{same}} \cdot \mathcal{N}(x_1 - x_2; 0, 2\sigma^2) + (1 - p_{\text{same}}) \cdot \left[\frac{1}{2}\mathcal{N}(x_1; \mu, \sigma^2)\mathcal{N}(x_2; -\mu, \sigma^2) + \frac{1}{2}\mathcal{N}(x_1; -\mu, \sigma^2)\mathcal{N}(x_2; \mu, \sigma^2)\right]}$$

Here are the pairs:

| $s_1$ | $s_2$ | $P(\text{same} \mid s_1, s_2)$ |
|---|---|---|
| $-1$ | $-1$ | 0.8048 |
| $-1$ | 1 | 0.5302 |
| 1 | $-1$ | 0.5302 |
| 1 | 1 | 0.8048 |

Table 1: Estimated probability of "same" response for different stimulus pairs.



```
# Q10.5
# Riemann integration
```

```
[1]  ✓ 0.0s                                                    Python
```

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

psame = 0.4
mu = 1
sigma = 1
dx = 0.05
x_vals = np.arange(-5, 5 + dx, dx)

# Meshgrid of x1 and x2
X1, X2 = np.meshgrid(x_vals, x_vals)

# Compute p_same_given_x1x2 (vectorization)
def p_same_given_x1x2(x1, x2):
    num = psame * norm.pdf(x1 - x2, 0, np.sqrt(2) * sigma)
    term1 = norm.pdf(x1, mu, sigma) * norm.pdf(x2, -mu, sigma)
    term2 = norm.pdf(x1, -mu, sigma) * norm.pdf(x2, mu, sigma)
    denom = num + (1 - psame) * 0.5 * (term1 + term2)
    return num / denom

# For each (s1, s2)
def compute_integral(s1, s2):
    px1 = norm.pdf(X1, s1, sigma)
    px2 = norm.pdf(X2, s2, sigma)
    p_decision = p_same_given_x1x2(X1, X2)
    integrand = p_decision * px1 * px2
    return np.sum(integrand) * dx * dx

# Stimulus pairs
stimulus_pairs = [(-1, -1), (-1, 1), (1, -1), (1, 1)]
results = {pair: compute_integral(pair[0], pair[1]) for pair in stimulus_pairs}
#print(results)

import pandas as pd

# Convert results to a DataFrame
df = pd.DataFrame([
    {"s1": s1, "s2": s2, "P(same | s1, s2)": prob}
    for (s1, s2), prob in results.items()
])

# Display the table
print(df)
```

```
[4]  ✓ 0.3s  Open 'df' in Data Wrangler                          Python
```

```
    s1  s2  P(same | s1, s2)
0  -1  -1          0.804830
1  -1   1          0.535028
2   1  -1          0.535028
3   1   1          0.804830
```

Figure 1: Python code and answer

b)
Using Monte Carlo simulation, here is the idea:

1. For each stimulus pair $(s_1, s_2)$, generate $N$ samples from:

$$x_1 \sim \mathcal{N}(s_1, \sigma^2), \quad x_2 \sim \mathcal{N}(s_2, \sigma^2)$$

2. For each pair $(x_1^{(i)}, x_2^{(i)})$, compute:

$$p(\hat{C} = 1 \mid x_1, x_2) = \frac{p_{\text{same}} \cdot \mathcal{N}(x_1 - x_2; 0, 2\sigma^2)}{p_{\text{same}} \cdot \mathcal{N}(x_1 - x_2; 0, 2\sigma^2) + \frac{1-p_{\text{same}}}{2} [\mathcal{N}(x_1; \mu, \sigma^2) \cdot \mathcal{N}(x_2; -\mu, \sigma^2) + \mathcal{N}(x_1; -\mu, \sigma^2) \cdot \mathcal{N}(x_2; \mu, \sigma^2)}$$

3. Estimate the overall probability of "same" by averaging over samples:

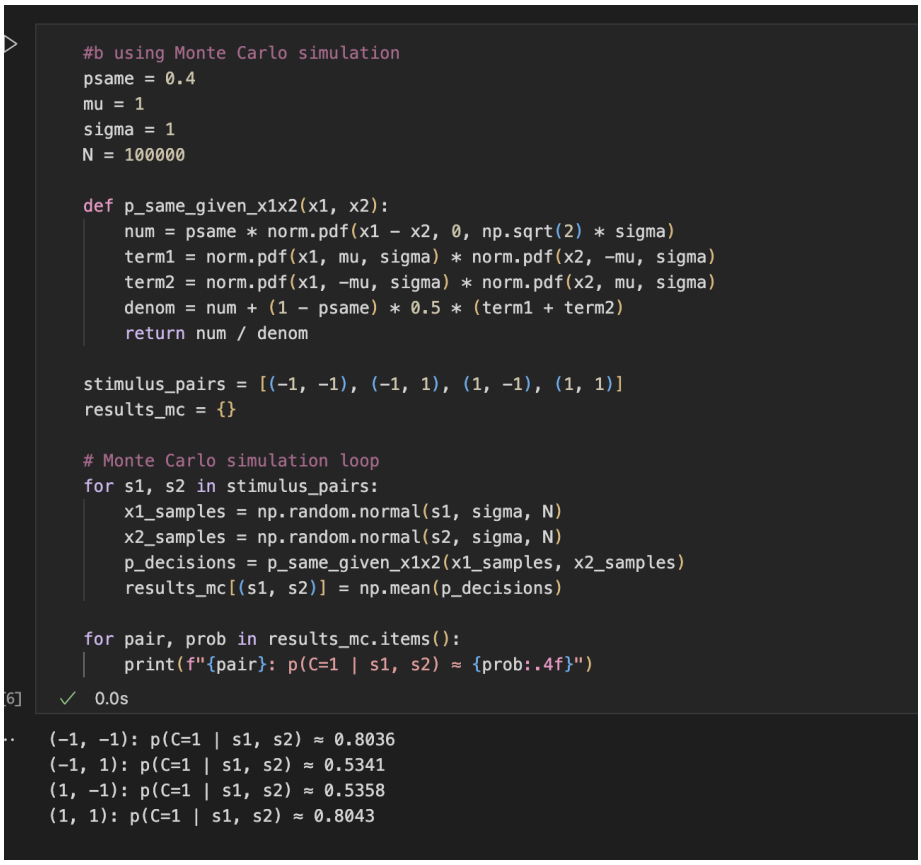
$$p(\hat{C} = 1 \mid s_1, s_2) \approx \frac{1}{N} \sum_{i=1}^{N} p(\hat{C} = 1 \mid x_1^{(i)}, x_2^{(i)})$$

Here are the pairs

| $s_1$ | $s_2$ | $P(\text{same} \mid s_1, s_2)$ |
|---|---|---|
| $-1$ | $-1$ | 0.8036 |
| $-1$ | 1 | 0.5341 |
| 1 | $-1$ | 0.5358 |
| 1 | 1 | 0.8043 |

Table 2: Estimated probabilities from Monte Carlo simulation.

:



```python
#b using Monte Carlo simulation
psame = 0.4
mu = 1
sigma = 1
N = 100000

def p_same_given_x1x2(x1, x2):
    num = psame * norm.pdf(x1 - x2, 0, np.sqrt(2) * sigma)
    term1 = norm.pdf(x1, mu, sigma) * norm.pdf(x2, -mu, sigma)
    term2 = norm.pdf(x1, -mu, sigma) * norm.pdf(x2, mu, sigma)
    denom = num + (1 - psame) * 0.5 * (term1 + term2)
    return num / denom

stimulus_pairs = [(-1, -1), (-1, 1), (1, -1), (1, 1)]
results_mc = {}

# Monte Carlo simulation loop
for s1, s2 in stimulus_pairs:
    x1_samples = np.random.normal(s1, sigma, N)
    x2_samples = np.random.normal(s2, sigma, N)
    p_decisions = p_same_given_x1x2(x1_samples, x2_samples)
    results_mc[(s1, s2)] = np.mean(p_decisions)

for pair, prob in results_mc.items():
    print(f"{pair}: p(C=1 | s1, s2) ≈ {prob:.4f}")
```

[6] ✓ 0.0s

```
(-1, -1): p(C=1 | s1, s2) ≈ 0.8036
(-1, 1): p(C=1 | s1, s2) ≈ 0.5341
(1, -1): p(C=1 | s1, s2) ≈ 0.5358
(1, 1): p(C=1 | s1, s2) ≈ 0.8043
```

Figure 2: Python code and answer

Since Riemann integration uses a long summation over a fine grid of $x_1$ and $x_2$, while Monte Carlo simulation uses randomly drawn samples, they should both converge to the same true value, provided we have enough resolution and enough samples.