Statistical Analysis in Physics

Name: Garima Singh

Course: BSc Hons in Physics

Section: A

College Roll No: 2230119

Examination Roll No: 22036567040

Group: G3

Submitted to:

- *Prof. Rakesh Kumar Pandey*
- *Dr. Pranav Kumar*

College: Kirori Mal College

Problem 1: Distribution function and CLT theorem

a) Generate a random sample space of size N using following probability distribution functions :

   I. Binomial
   II. Poisson
   III. Normal
   IV. Cauchy-Lorentz

   and make histogram plot for at least three different values of their parameters .

b) Verify CLT theorem for the functions mentioned in part (a) for fixed sampling (M : take large value) and vary sample size (N). Show it graphically.

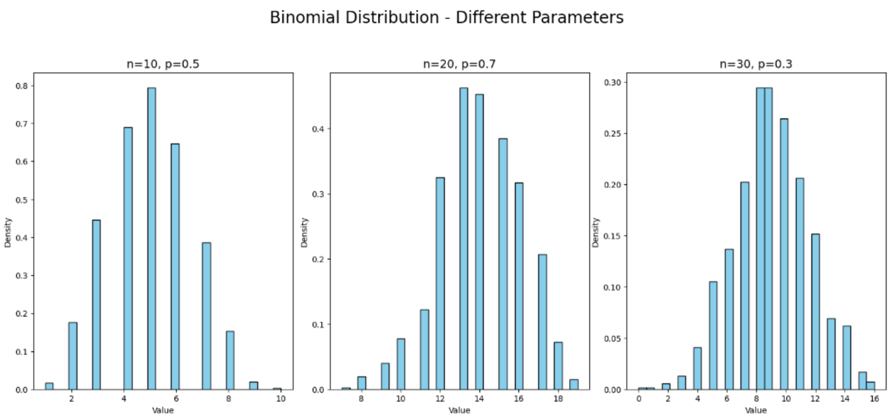
c) With sample size (N) as large as the CLT theorem is verified in part (b), check for minimum sampling (M) required to achieve the normal distribution of sample means. You may take M = 100, 500, 1000, 5000, 10000. Show it graphically.

• *Binomial Distribution Function*

a) Histogram Plot

```
#GARIMA SINGH
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom
np.random.seed(42)
N = 1000
params = [(10, 0.5), (20, 0.7), (30, 0.3)]
fig, axs = plt.subplots(1, 3, figsize=(18, 5))
fig.suptitle('Binomial Distribution - Different Parameters', fontsize=20)
for idx, (n, p) in enumerate(params):
    data = binom.rvs(n=n, p=p, size=N)
    axs[idx].hist(data, bins=30, density=True, edgecolor='black', color='skyblue')
    axs[idx].set_title(f'n={n}, p={p}', fontsize=14)
    axs[idx].set_xlabel('Value')
    axs[idx].set_ylabel('Density')
plt.tight_layout()
plt.show()
```

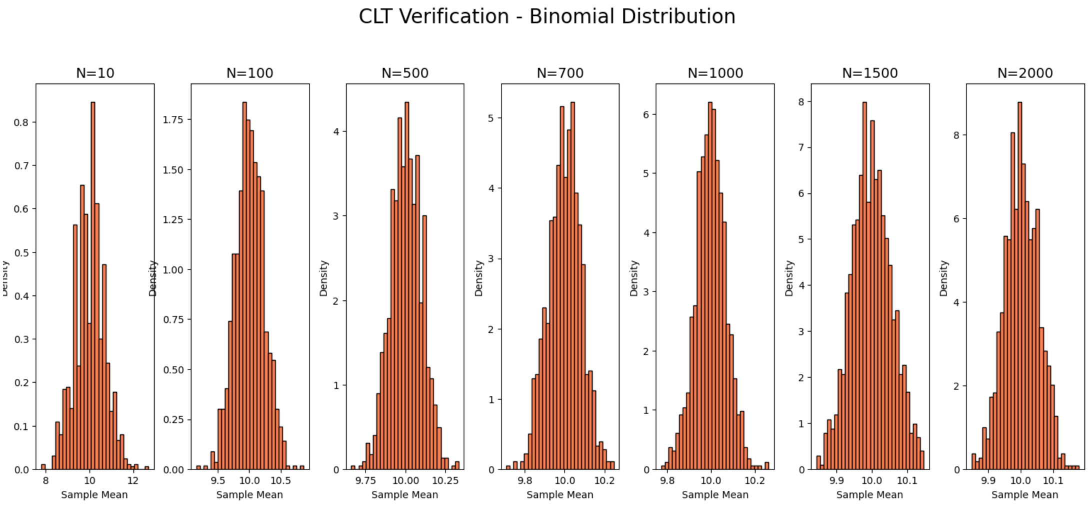
Output



b) CLT Theorem verification by varying sample size N

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom
M = 1000
N_values = [ 10, 100, 500,700,1000,1500,2000]
n, p = 20, 0.5
fig, axs = plt.subplots(1, len(N_values), figsize=(22, 5))
fig.suptitle('CLT Verification - Binomial Distribution', fontsize=20)
for idx, N in enumerate(N_values):
    sample_means = []
    for _ in range(M):
        sample = binom.rvs(n=n, p=p, size=N)
        sample_means.append(np.mean(sample))

    axs[idx].hist(sample_means, bins=30, density=True, edgecolor='black', color='coral')
    axs[idx].set_title(f'N={N}', fontsize=14)
    axs[idx].set_xlabel('Sample Mean')
    axs[idx].set_ylabel('Density')
plt.tight_layout()
plt.show()
```

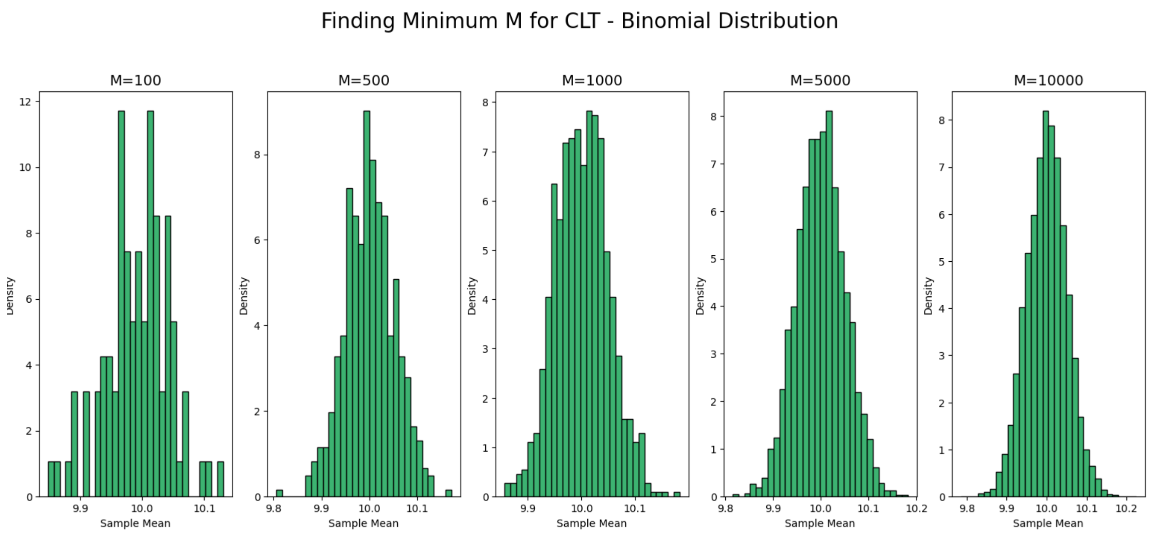
Output



c)Minimum sampling (M) required to achieve normal distribution

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom
N = 2000
M_values = [100, 500, 1000, 5000, 10000]
n, p = 20, 0.5
fig, axs = plt.subplots(1, len(M_values), figsize=(22, 5))
fig.suptitle('Finding Minimum M for CLT - Binomial Distribution', fontsize=20)
for idx, M in enumerate(M_values):
    sample_means = []
    for _ in range(M):
        sample = binom.rvs(n=n, p=p, size=N)
        sample_means.append(np.mean(sample))

    axs[idx].hist(sample_means, bins=30, density=True, edgecolor='black', color='mediumseagreen')
    axs[idx].set_title(f'M={M}', fontsize=14)
    axs[idx].set_xlabel('Sample Mean')
    axs[idx].set_ylabel('Density')
plt.tight_layout()
plt.show()
```

Output

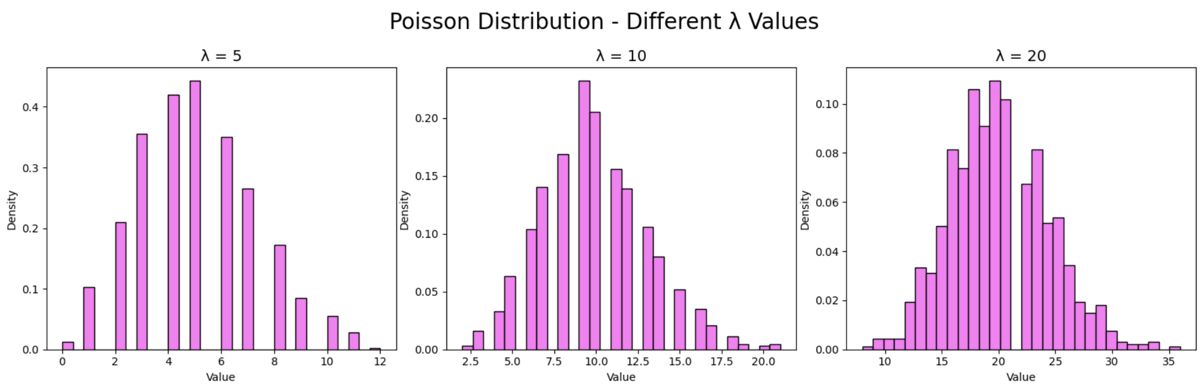


• Poisson Distribution Function

a)Histogram Plot

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson
np.random.seed(42)
N = 1000
lambdas = [ 5, 10,20]
fig, axs = plt.subplots(1, 3, figsize=(18, 5))
fig.suptitle('Poisson Distribution - Different λ Values', fontsize=20)
for idx, lam in enumerate(lambdas):
    data = poisson.rvs(mu=lam, size=N)
    axs[idx].hist(data, bins=30, density=True, edgecolor='black', color='violet')
    axs[idx].set_title(f'λ = {lam}', fontsize=14)
    axs[idx].set_xlabel('Value')
    axs[idx].set_ylabel('Density')
plt.tight_layout()
plt.show()
```

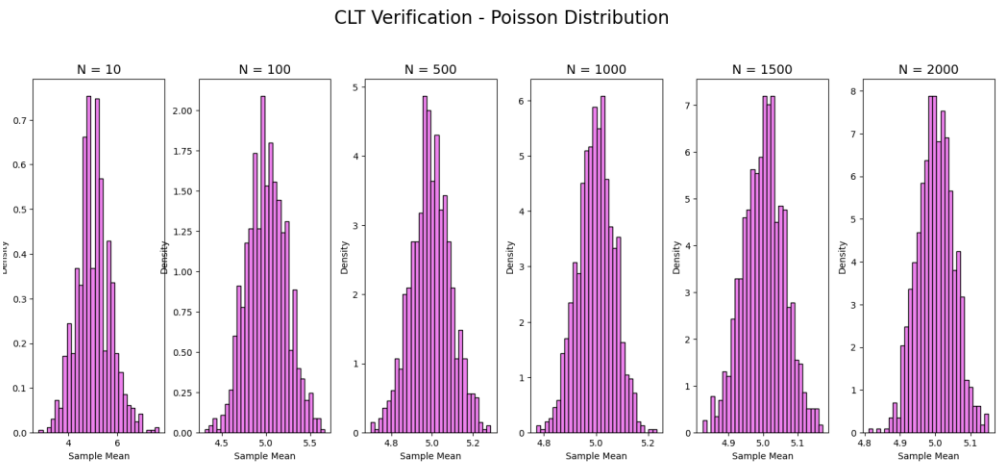
Output



b)CLT Theorem verification by varying sample size N

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson
M = 1000
N_values = [10,100,500,1000,1500,2000]
lam = 5
fig, axs = plt.subplots(1, len(N_values), figsize=(22, 5))
fig.suptitle('CLT Verification - Poisson Distribution', fontsize=20)

for idx, N in enumerate(N_values):
    sample_means = []
    for _ in range(M):
        sample = poisson.rvs(mu=lam, size=N)
        sample_means.append(np.mean(sample))

    axs[idx].hist(sample_means, bins=30, density=True, edgecolor='black', color='violet')
    axs[idx].set_title(f'N = {N}', fontsize=14)
    axs[idx].set_xlabel('Sample Mean')
    axs[idx].set_ylabel('Density')
plt.tight_layout()
plt.show()
```

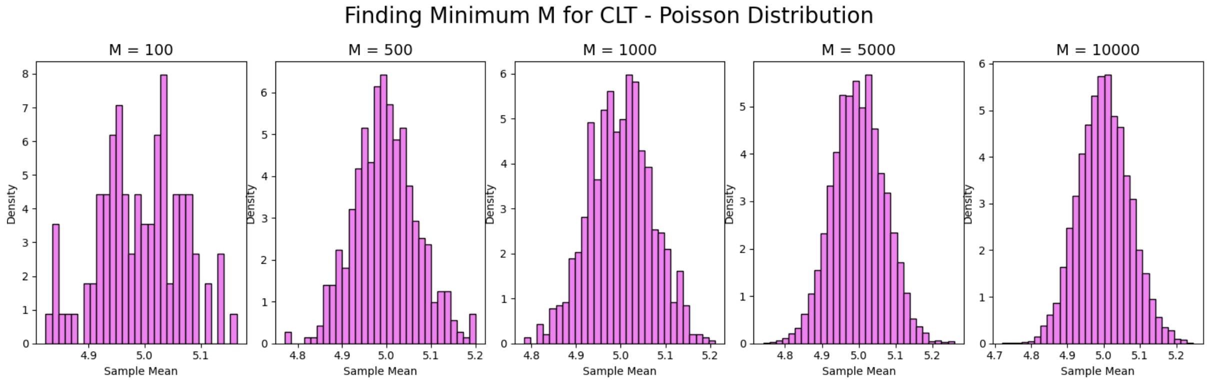
Output



c)Minimum sampling (M) required to achieve normal distribution

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson
N = 1000
M_values = [100, 500, 1000, 5000, 10000]
lam = 5
fig, axs = plt.subplots(1, len(M_values), figsize=(22, 5))
fig.suptitle('Finding Minimum M for CLT - Poisson Distribution', fontsize=20)
for idx, M in enumerate(M_values):
    sample_means = []
    for _ in range(M):
        sample = poisson.rvs(mu=lam, size=N)
        sample_means.append(np.mean(sample))

    axs[idx].hist(sample_means, bins=30, density=True, edgecolor='black', color='violet')
    axs[idx].set_title(f'M = {M}', fontsize=14)
    axs[idx].set_xlabel('Sample Mean')
    axs[idx].set_ylabel('Density')
plt.tight_layout()
plt.show()
```

Output

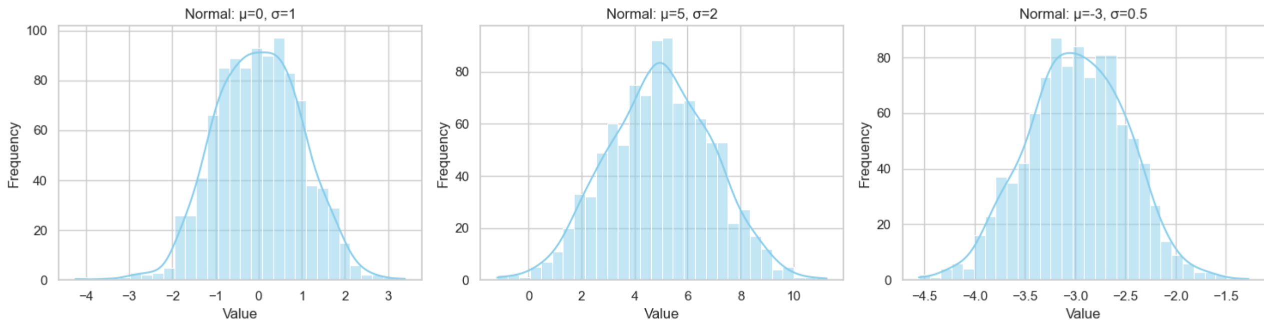


- ## *Normal Distribution Function*

a)Histogram Plot

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")
def part_a_normal_distribution(N, param_sets):
    plt.figure(figsize=(15, 4))
    for i, (mu, sigma) in enumerate(param_sets):
        samples = np.random.normal(loc=mu, scale=sigma, size=N)
        plt.subplot(1, 3, i + 1)
        sns.histplot(samples, bins=30, kde=True, color='skyblue')
        plt.title(f'Normal: μ={mu}, σ={sigma}')
        plt.xlabel('Value')
        plt.ylabel('Frequency')
    plt.tight_layout()
    plt.suptitle(f'Part (a): Normal Distributions with N={N}', y=1.05, fontsize=16)
    plt.show()
N = 1000
param_sets_normal = [(0, 1), (5, 2), (-3, 0.5)]
part_a_normal_distribution(N, param_sets_normal)
```

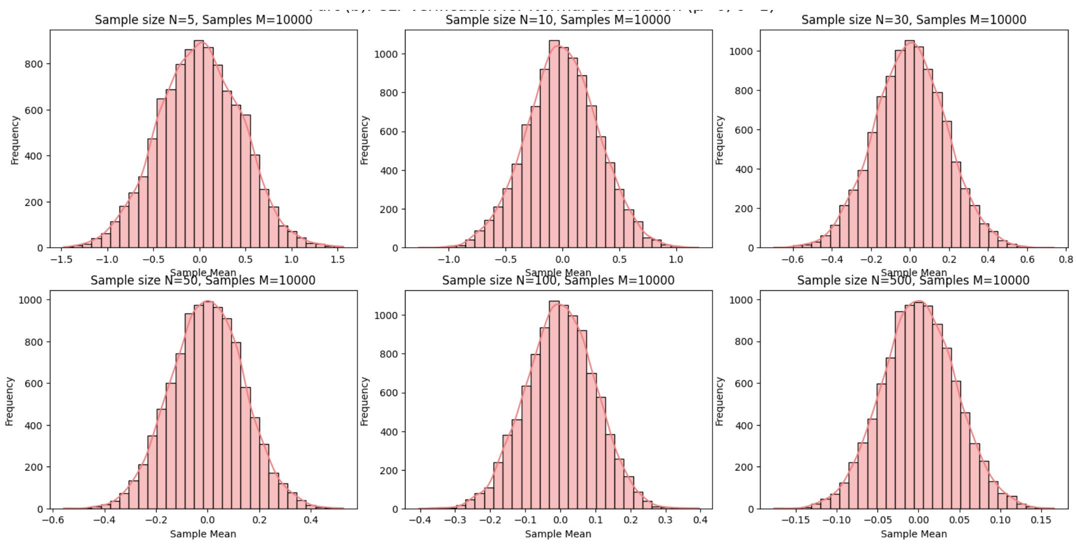
Output



b) CLT Theorem verification by varying sample size N

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
def part_b_verify_clt_normal(M, N_values, mu=0, sigma=1):
    plt.figure(figsize=(18, 10))
    for i, N in enumerate(N_values):
        sample_means = []
        for _ in range(M):
            sample = np.random.normal(loc=mu, scale=sigma, size=N)
            sample_means.append(np.mean(sample))
        plt.subplot(2, len(N_values) // 2, i + 1)
        sns.histplot(sample_means, bins=30, kde=True, color='lightcoral')
        plt.title(f'Sample size N={N}, Samples M={M}')
        plt.xlabel('Sample Mean')
        plt.ylabel('Frequency')
    plt.tight_layout()
    plt.suptitle(f'Part (b): CLT Verification for Normal Distribution (μ={mu}, σ={sigma})', y=1.02, fontsize=16)
    plt.show()
M = 10000
N_values = [5, 10, 30, 50, 100, 500]
part_b_verify_clt_normal(M, N_values)
```

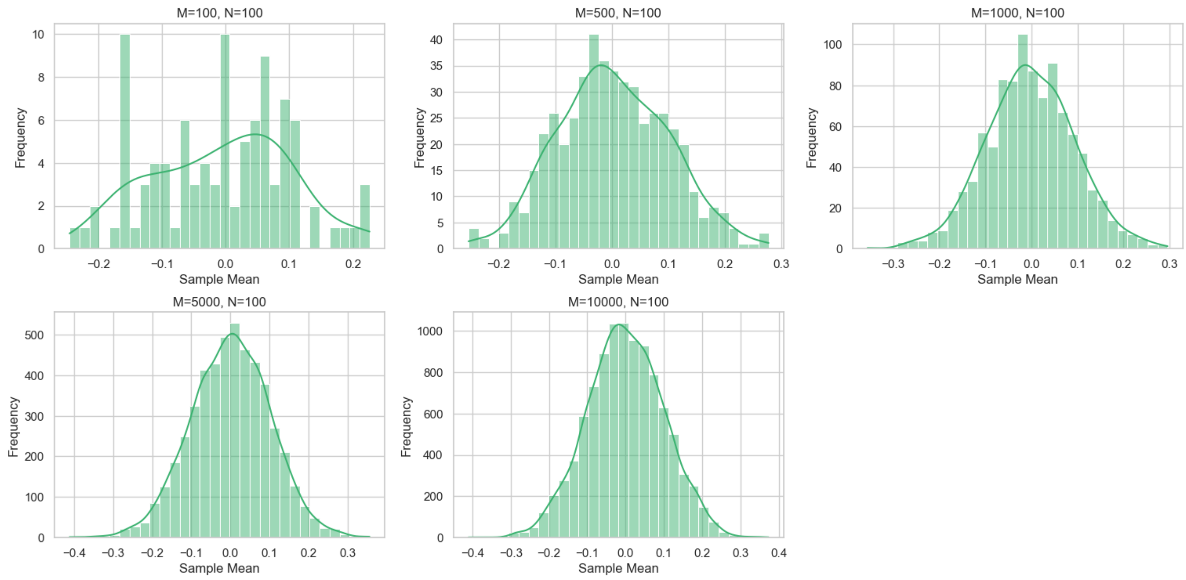
Output



c)Minimum sampling (M) required to achieve normal distribution

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")
def part_c_minimum_m_for_clt(N, M_values, mu=0, sigma=1):
    rows = 2
    cols = int(np.ceil(len(M_values) / rows))
    plt.figure(figsize=(cols * 5, rows * 4))
    for i, M in enumerate(M_values):
        sample_means = []
        for _ in range(M):
            sample = np.random.normal(loc=mu, scale=sigma, size=N)
            sample_means.append(np.mean(sample))

        plt.subplot(rows, cols, i + 1)
        sns.histplot(sample_means, bins=30, kde=True, color='mediumseagreen')
        plt.title(f'M={M}, N={N}')
        plt.xlabel('Sample Mean')
        plt.ylabel('Frequency')

    plt.tight_layout()
    plt.suptitle(f'Part (c): Minimum M Needed for Normality (Fixed N={N})', y=1.05, fontsize=16)
    plt.show()
N_fixed = 100
M_values = [100, 500, 1000, 5000, 10000]
part_c_minimum_m_for_clt(N_fixed, M_values)
```

Output

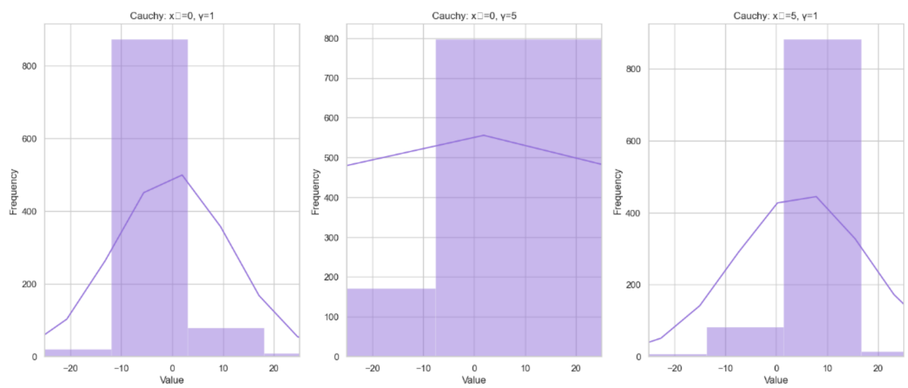


• Cauchy-Lorentz Distribution Function

a)Histogram Plot

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")
def part_a_cauchy_distribution(N, param_sets):
    plt.figure(figsize=(15, 4))
    for i, (x0, gamma) in enumerate(param_sets):
        samples = np.random.standard_cauchy(size=N) * gamma + x0
        plt.subplot(1, 3, i + 1)
        sns.histplot(samples, bins=100, kde=True, color='mediumpurple')
        plt.xlim(-25, 25)  # Cauchy has heavy tails, so limit for better view
        plt.title(f'Cauchy: x₀={x0}, γ={gamma}')
        plt.xlabel('Value')
        plt.ylabel('Frequency')
    plt.tight_layout()
    plt.suptitle(f'Part (a): Cauchy Distributions with N={N}', y=1.05, fontsize=16)
    plt.show()
N = 1000
param_sets_cauchy = [(0, 1), (0, 5), (5, 1)]
part_a_cauchy_distribution(N, param_sets_cauchy)
```

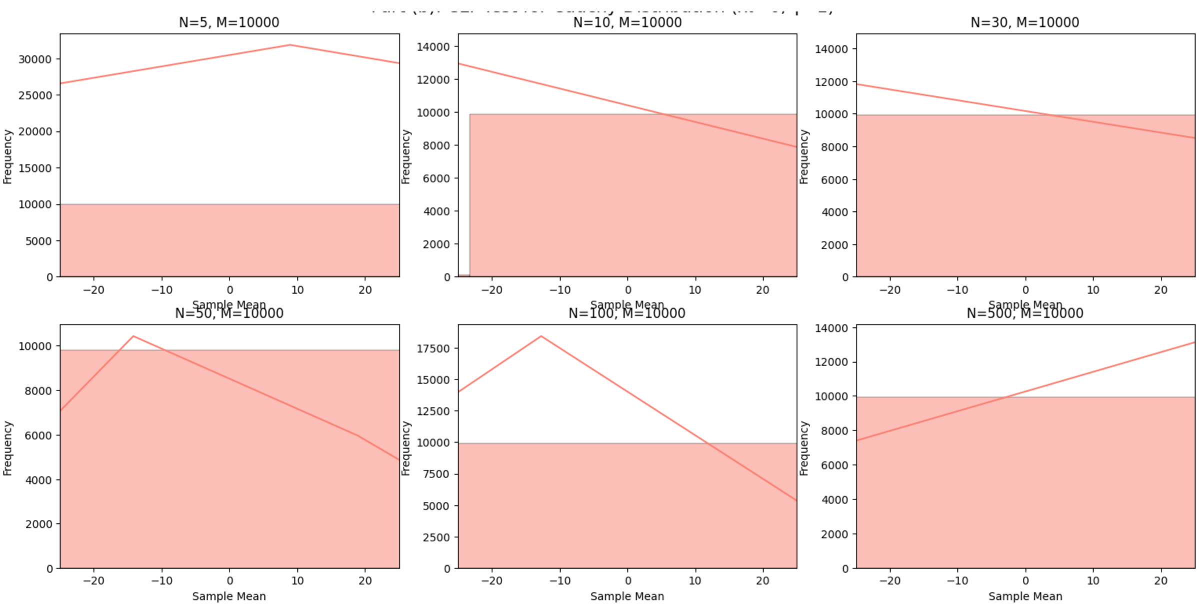
Output



b) CLT Theorem verification by varying sample size N

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
def part_b_clt_cauchy(M, N_values, x0=0, gamma=1):
    plt.figure(figsize=(18, 10))
    for i, N in enumerate(N_values):
        sample_means = []
        for _ in range(M):
            sample = np.random.standard_cauchy(size=N) * gamma + x0
            sample_means.append(np.mean(sample))
        plt.subplot(2, len(N_values) // 2, i + 1)
        sns.histplot(sample_means, bins=100, kde=True, color='salmon')
        plt.xlim(-25, 25)
        plt.title(f'N={N}, M={M}')
        plt.xlabel('Sample Mean')
        plt.ylabel('Frequency')
    plt.tight_layout()
    plt.suptitle(f'Part (b): CLT Test for Cauchy Distribution (x₀={x0}, γ={gamma})', y=1.02, fontsize=16)
    plt.show()
M = 10000
N_values_cauchy = [5, 10, 30, 50, 100, 500]
part_b_clt_cauchy(M, N_values_cauchy)
```

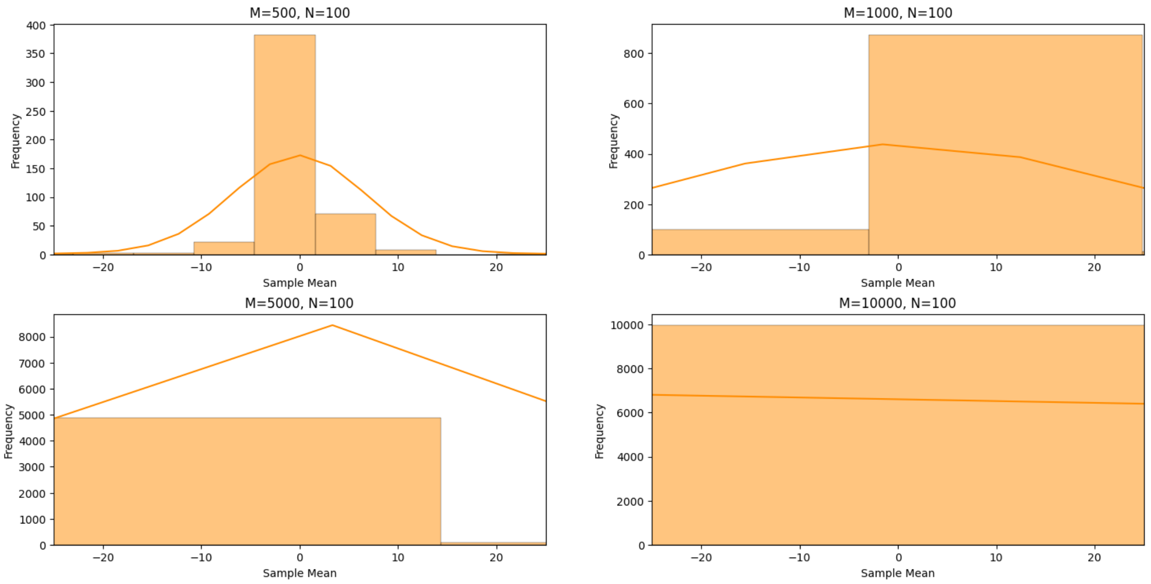
Output



c) Minimum sampling (M) required to achieve normal distribution

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
def part_c_minimum_m_for_cauchy(N, M_values, x0=0, gamma=1):
    rows = 2
    cols = int(np.ceil(len(M_values) / rows))
    plt.figure(figsize=(cols * 5, rows * 4))

    for i, M in enumerate(M_values):
        sample_means = []
        for _ in range(M):
            sample = np.random.standard_cauchy(size=N) * gamma + x0
            sample_means.append(np.mean(sample))

        plt.subplot(rows, cols, i + 1)
        sns.histplot(sample_means, bins=100, kde=True, color='darkorange')
        plt.xlim(-25, 25)
        plt.title(f'M={M}, N={N}')
        plt.xlabel('Sample Mean')
        plt.ylabel('Frequency')

    plt.tight_layout()
    plt.suptitle(f'Part (c): Varying M for CLT in Cauchy (Fixed N={N})', y=1.05, fontsize=16)
    plt.show()
N_fixed_cauchy = 100
M_values_cauchy = [ 500, 1000, 5000, 10000]
part_c_minimum_m_for_cauchy(N_fixed_cauchy, M_values_cauchy)
```

Output



Cauchy distribution *does* ***NOT*** *satisfy the Central Limit Theorem (CLT) well — because it has infinite variance.*

So ***even if we increase N or M****, the sample mean won't stabilize nicely into a normal distribution, unlike Normal or Binomial.*

In graphs, we can see "wild behaviour" (many outliers, fat tails).

Problem 2: Joint distribution

a) Discrete case: Use random number generator (integer random number, Binomial, Poisson) to generate two random variables (X and Y) with sample size more than 50 and perform following:-

i. For the two random variables, make a joint table and show the joint probability graphically. You may use 'heatmap()' or 'imshow()' function for graphical display.

ii. Calculate and display the marginal distribution of the random variables.

iii. Check if two variables are independent or not. [Optional]

b) Continuous case: Generate two random variables (X and Y) of sample size more than 200 using normal distribution and perform following:

i. Make a 3dplot or surface plot of joint probability density of X and Y (probability surface).

ii. On the plot obtained in part (i) mark/show the region representing joint probability for X and Y in the range [a,b] and [c, d] respectively. (a > min(X), b < max(X) and c > min(Y), d < max(Y)).

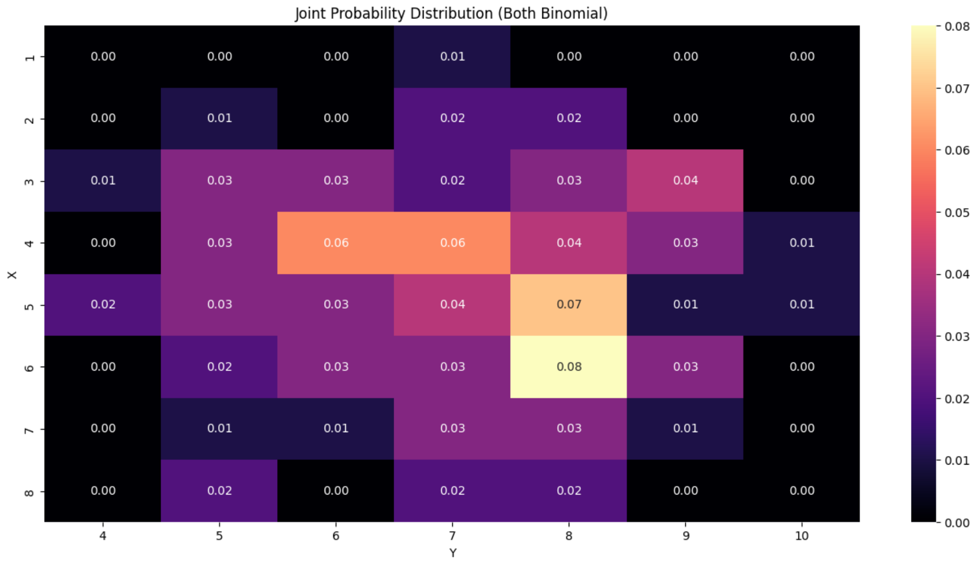
BINOMIAL RANDOM NUMBER GENERATOR

Graphical Display of Joint Probability

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(42)
N = 100
X = np.random.binomial(n=10, p=0.5, size=N)
Y = np.random.binomial(n=10, p=0.7, size=N)
joint_table = pd.crosstab(X, Y, normalize='all')
print("\nJoint Probability Table (X vs Y):\n")
print(joint_table)
# Plot joint probability heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(joint_table, cmap='magma', annot=True, fmt=".2f")
plt.title('Joint Probability Distribution (Both Binomial)')
plt.xlabel('Y')
plt.ylabel('X')
plt.show()
```

Output

```
==================================================== RESTA

Joint Probability Table (X vs Y):

col_0     4     5     6     7     8     9    10
row_0
1      0.00  0.00  0.00  0.01  0.00  0.00  0.00
2      0.00  0.01  0.00  0.02  0.02  0.00  0.00
3      0.01  0.03  0.03  0.02  0.03  0.04  0.00
4      0.00  0.03  0.06  0.06  0.04  0.03  0.01
5      0.02  0.03  0.03  0.04  0.07  0.01  0.01
6      0.00  0.02  0.03  0.03  0.08  0.03  0.00
7      0.00  0.01  0.01  0.03  0.03  0.01  0.00
8      0.00  0.02  0.00  0.02  0.02  0.00  0.00
```

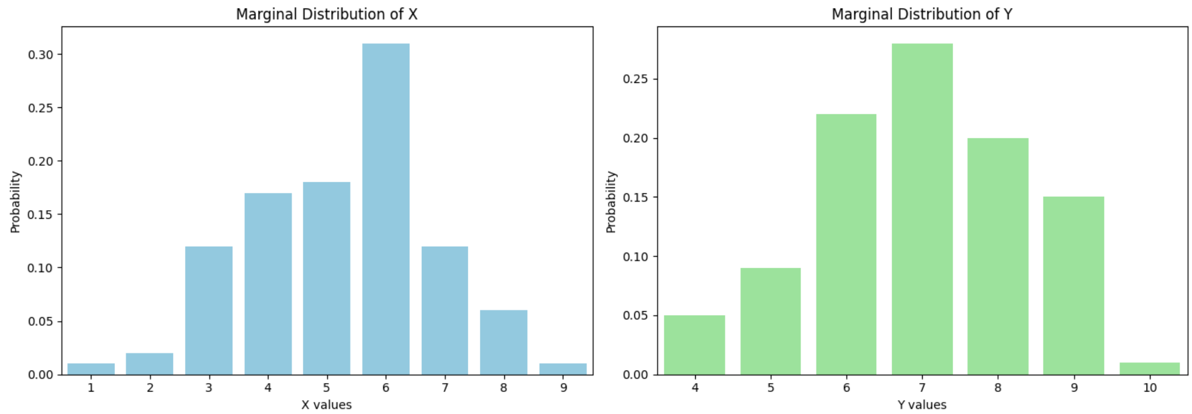


Marginal Distribution of Random Variables

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
N = 100
X = np.random.binomial(n=10, p=0.5, size=N)
Y = np.random.binomial(n=10, p=0.7, size=N)
joint_table = pd.crosstab(X, Y, normalize='all')
marginal_X = joint_table.sum(axis=1)
marginal_Y = joint_table.sum(axis=0)
print("\nMarginal Distribution of X:\n")
print(marginal_X)
print("\nMarginal Distribution of Y:\n")
print(marginal_Y)
fig, axs = plt.subplots(1, 2, figsize=(14, 5))
# Plot Marginal X
sns.barplot(x=marginal_X.index, y=marginal_X.values, ax=axs[0], color='skyblue')
axs[0].set_title('Marginal Distribution of X')
axs[0].set_xlabel('X values')
axs[0].set_ylabel('Probability')
# Plot Marginal Y
sns.barplot(x=marginal_Y.index, y=marginal_Y.values, ax=axs[1], color='lightgreen')
axs[1].set_title('Marginal Distribution of Y')
axs[1].set_xlabel('Y values')
axs[1].set_ylabel('Probability')
plt.tight_layout()
plt.show()
```

Output

```
=====================================

Marginal Distribution of X:

row_0
1    0.01
2    0.02
3    0.12
4    0.17
5    0.18
6    0.31
7    0.12
8    0.06
9    0.01
dtype: float64

Marginal Distribution of Y:

col_0
4     0.05
5     0.09
6     0.22
7     0.28
8     0.20
9     0.15
10    0.01
dtype: float64
```



Independence of Random Variables

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
N = 100
X = np.random.binomial(n=10, p=0.5, size=N)
Y = np.random.binomial(n=10, p=0.7, size=N)
joint_table = pd.crosstab(X, Y, normalize='all')
marginal_X = joint_table.sum(axis=1)
marginal_Y = joint_table.sum(axis=0)
product_marginals = np.outer(marginal_X, marginal_Y)
product_table = pd.DataFrame(product_marginals, index=joint_table.index, columns=joint_table.columns)
difference = np.abs(joint_table - product_table)
print("\nMaximum absolute difference between Joint and Product of Marginals:", difference.values.max())
# Decision based on threshold
threshold = 0.05  # you can set stricter value
if difference.values.max() < threshold:
    print("\n✅ X and Y can be considered approximately independent.")
else:
    print("\n❌ X and Y are NOT independent.")
```

Output

```
=============================================== RESTART: C:/Users/thedy/AppData/Local/Programs/I
Maximum absolute difference between Joint and Product of Marginals: 0.035999999999999
☑ X and Y can be considered approximately independent.
>>
```

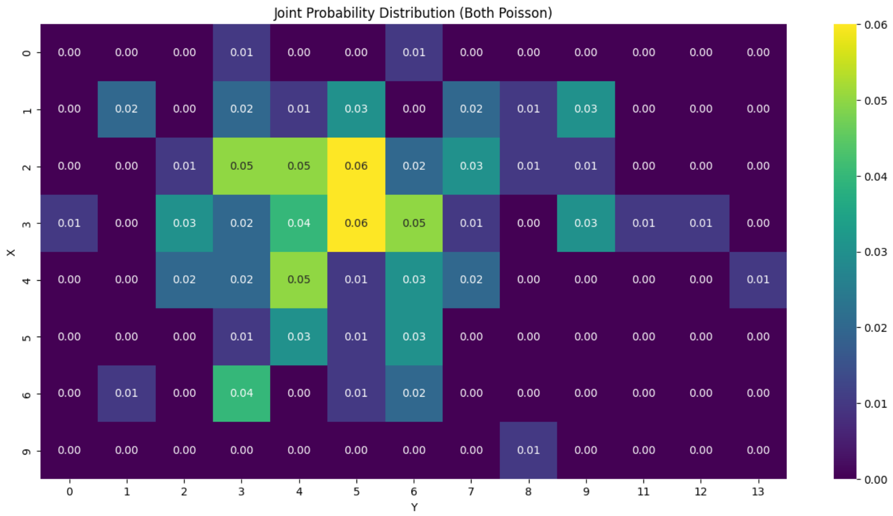
POISSON RANDOM NUMBER GENERATOR

Graphical Display of Joint Probability

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(123)
N = 100
X = np.random.poisson(lam=3, size=N)
Y = np.random.poisson(lam=5, size=N)
joint_table = pd.crosstab(X, Y, normalize='all')
print("\nJoint Probability Table (X vs Y) for Poisson:\n")
print(joint_table)
# Plot joint probability heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(joint_table, cmap='viridis', annot=True, fmt=".2f")
plt.title('Joint Probability Distribution (Both Poisson)')
plt.xlabel('Y')
plt.ylabel('X')
plt.show()
```

Output

```
==================================================== RESTART: C:/Users/thedy/AppData/Local/Progr

Joint Probability Table (X vs Y) for Poisson:

col_0     0     1     2     3     4     5  ...     7     8     9    11    12    13
row_0                                      ...
0      0.00  0.00  0.00  0.01  0.00  0.00  ...  0.00  0.00  0.00  0.00  0.00  0.00
1      0.00  0.02  0.00  0.02  0.01  0.03  ...  0.02  0.01  0.03  0.00  0.00  0.00
2      0.00  0.00  0.01  0.05  0.05  0.06  ...  0.03  0.01  0.01  0.00  0.00  0.00
3      0.01  0.00  0.03  0.02  0.04  0.06  ...  0.01  0.00  0.03  0.01  0.01  0.00
4      0.00  0.00  0.02  0.02  0.05  0.01  ...  0.02  0.00  0.00  0.00  0.00  0.01
5      0.00  0.00  0.00  0.01  0.03  0.01  ...  0.00  0.00  0.00  0.00  0.00  0.00
6      0.00  0.01  0.00  0.04  0.00  0.01  ...  0.00  0.00  0.00  0.00  0.00  0.00
9      0.00  0.00  0.00  0.00  0.00  0.00  ...  0.00  0.01  0.00  0.00  0.00  0.00

[8 rows x 13 columns]
```

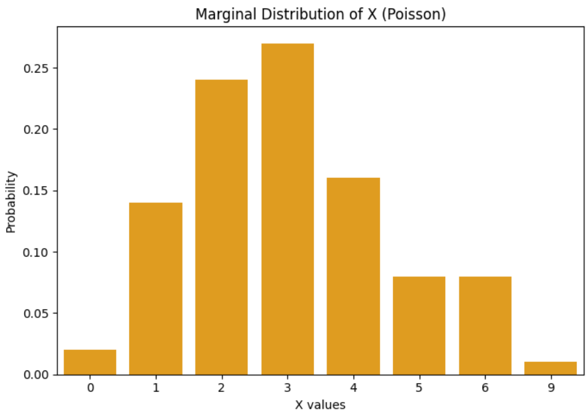


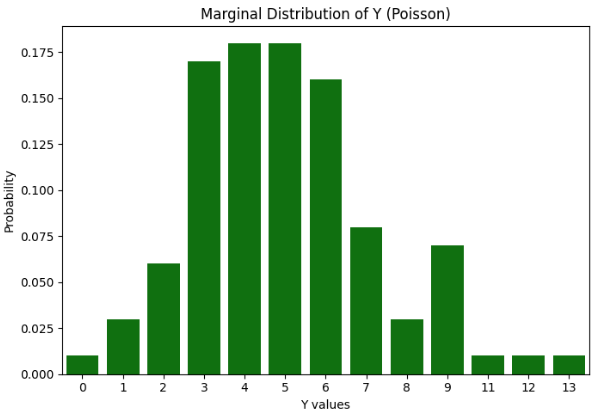
Marginal Distribution of Random Variables

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(123)
N = 100
X = np.random.poisson(lam=3, size=N)
Y = np.random.poisson(lam=5, size=N)
joint_table = pd.crosstab(X, Y, normalize='all')
marginal_X = joint_table.sum(axis=1)
marginal_Y = joint_table.sum(axis=0)
print("\nMarginal Distribution of X:\n")
print(marginal_X)
print("\nMarginal Distribution of Y:\n")
print(marginal_Y)
# Plot Marginal Distributions
fig, axs = plt.subplots(1, 2, figsize=(14, 5))
# Marginal X
sns.barplot(x=marginal_X.index, y=marginal_X.values, ax=axs[0], color='orange')
axs[0].set_title('Marginal Distribution of X (Poisson)')
axs[0].set_xlabel('X values')
axs[0].set_ylabel('Probability')
# Marginal Y
sns.barplot(x=marginal_Y.index, y=marginal_Y.values, ax=axs[1], color='green')
axs[1].set_title('Marginal Distribution of Y (Poisson)')
axs[1].set_xlabel('Y values')
axs[1].set_ylabel('Probability')
plt.tight_layout()
plt.show()
```

Output

```
===================================

Marginal Distribution of X:

row_0
0    0.02
1    0.14
2    0.24
3    0.27
4    0.16
5    0.08
6    0.08
9    0.01
dtype: float64

Marginal Distribution of Y:

col_0
0     0.01
1     0.03
2     0.06
3     0.17
4     0.18
5     0.18
6     0.16
7     0.08
8     0.03
9     0.07
11    0.01
12    0.01
13    0.01
dtype: float64
```





Independence of Random Variables

File Edit Format Run Options Window Help

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(123)
N = 100
X = np.random.poisson(lam=3, size=N)
Y = np.random.poisson(lam=5, size=N)
joint_table = pd.crosstab(X, Y, normalize='all')
marginal_X = joint_table.sum(axis=1)
marginal_Y = joint_table.sum(axis=0)
product_marginals = np.outer(marginal_X, marginal_Y)
product_table = pd.DataFrame(product_marginals, index=joint_table.index, columns=joint_table.columns)
difference = np.abs(joint_table - product_table)
print("\nMaximum absolute difference between Joint and Product of Marginals:", difference.values.max())
threshold = 0.05
if difference.values.max() < threshold:
    print("\n✅ X and Y can be considered approximately independent (Poisson).")
else:
    print("\n❌ X and Y are NOT independent (Poisson).")

joint_flat = joint_table.stack()
product_flat = product_table.stack()
difference_flat = difference.stack()
comparison_df = pd.DataFrame({
    'Joint P(X,Y)': joint_flat,
    'Product P(X)P(Y)': product_flat,
    'Absolute Difference': difference_flat
})
comparison_df = comparison_df.reset_index()
comparison_df.rename(columns={'index': 'X', 'level_1': 'Y'}, inplace=True)
print("\n📋 Clean Comparison Table (Poisson case):\n")
print(comparison_df)
comparison_df_sorted = comparison_df.sort_values('Absolute Difference', ascending=False)
print("\nTop 10 biggest deviations (Poisson case):\n")
print(comparison_df_sorted.head(10))
```

Output

```
========================================== RESTART: C:/Users/thedy/AppData/Local/Pro

Maximum absolute difference between Joint and Product of Marginals: 0.0264

✅ X and Y can be considered approximately independent (Poisson).

📋 Clean Comparison Table (Poisson case):

     row_0  col_0  Joint P(X,Y)  Product P(X)P(Y)  Absolute Difference
0        0      0          0.00            0.0002               0.0002
1        0      1          0.00            0.0006               0.0006
2        0      2          0.00            0.0012               0.0012
3        0      3          0.01            0.0034               0.0066
4        0      4          0.00            0.0036               0.0036
..     ...    ...           ...               ...                  ...
99       9      8          0.01            0.0003               0.0097
100      9      9          0.00            0.0007               0.0007
101      9     11          0.00            0.0001               0.0001
102      9     12          0.00            0.0001               0.0001
103      9     13          0.00            0.0001               0.0001

[104 rows x 5 columns]

Top 10 biggest deviations (Poisson case):

    row_0  col_0  Joint P(X,Y)  Product P(X)P(Y)  Absolute Difference
81      6      3          0.04            0.0136               0.0264
42      3      3          0.02            0.0459               0.0259
19      1      6          0.00            0.0224               0.0224
56      4      4          0.05            0.0288               0.0212
22      1      9          0.03            0.0098               0.0202
57      4      5          0.01            0.0288               0.0188
32      2      6          0.02            0.0384               0.0184
71      5      6          0.03            0.0128               0.0172
31      2      5          0.06            0.0432               0.0168
14      1      1          0.02            0.0042               0.0158
>>>
```

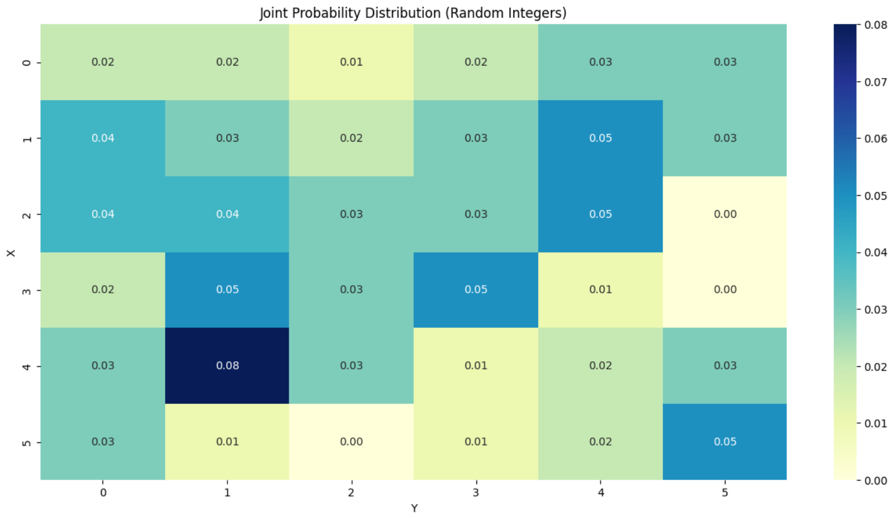
INTEGER RANDOM NUMBER GENERATOR

Graphical Display of Joint Probability

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(1234)
N = 100
X = np.random.randint(low=0, high=6, size=N)
Y = np.random.randint(low=0, high=6, size=N)
joint_table = pd.crosstab(X, Y, normalize='all')
print("\nJoint Probability Table (X vs Y) for Random Integers:\n")
print(joint_table)
# Plot joint probability heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(joint_table, cmap='YlGnBu', annot=True, fmt=".2f")
plt.title('Joint Probability Distribution (Random Integers)')
plt.xlabel('Y')
plt.ylabel('X')
plt.show()
```

Output

```
Joint Probability Table (X vs Y) for Random Integers:

col_0     0     1     2     3     4     5
row_0
0      0.02  0.02  0.01  0.02  0.03  0.03
1      0.04  0.03  0.02  0.03  0.05  0.03
2      0.04  0.04  0.03  0.03  0.05  0.00
3      0.02  0.05  0.03  0.05  0.01  0.00
4      0.03  0.08  0.03  0.01  0.02  0.03
5      0.03  0.01  0.00  0.01  0.02  0.05
```

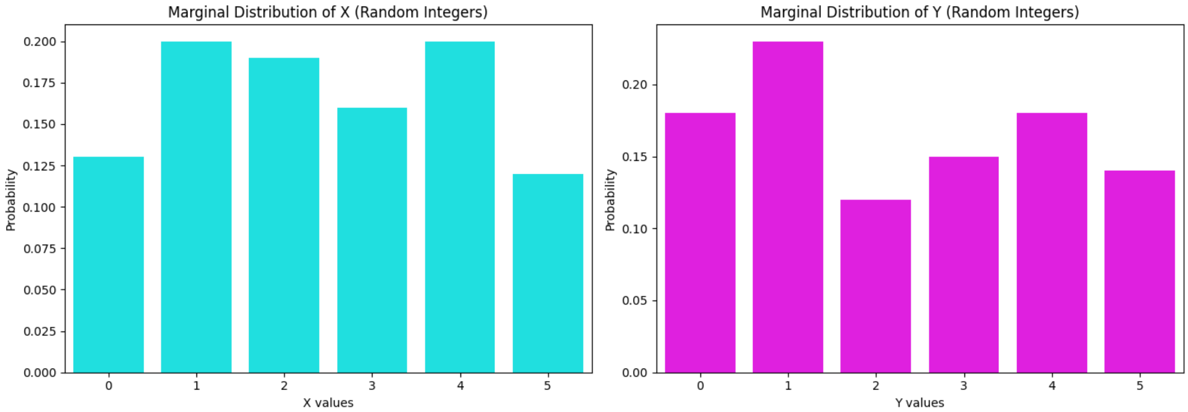


Marginal Distribution of Random Variables

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(1234)
N = 100
X = np.random.randint(low=0, high=6, size=N)
Y = np.random.randint(low=0, high=6, size=N)
joint_table = pd.crosstab(X, Y, normalize='all')
marginal_X = joint_table.sum(axis=1)
marginal_Y = joint_table.sum(axis=0)
print("\nMarginal Distribution of X:\n")
print(marginal_X)
print("\nMarginal Distribution of Y:\n")
print(marginal_Y)
# Plot Marginal Distributions
fig, axs = plt.subplots(1, 2, figsize=(14, 5))
# Marginal X
sns.barplot(x=marginal_X.index, y=marginal_X.values, ax=axs[0], color='cyan')
axs[0].set_title('Marginal Distribution of X (Random Integers)')
axs[0].set_xlabel('X values')
axs[0].set_ylabel('Probability')
# Marginal Y
sns.barplot(x=marginal_Y.index, y=marginal_Y.values, ax=axs[1], color='magenta')
axs[1].set_title('Marginal Distribution of Y (Random Integers)')
axs[1].set_xlabel('Y values')
axs[1].set_ylabel('Probability')
plt.tight_layout()
plt.show()
```

Output

```
Marginal Distribution of X:

row_0
0    0.13
1    0.20
2    0.19
3    0.16
4    0.20
5    0.12
dtype: float64

Marginal Distribution of Y:

col_0
0    0.18
1    0.23
2    0.12
3    0.15
4    0.18
5    0.14
dtype: float64
```



Independence of Random Variables

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
np.random.seed(1234)
N = 100
X = np.random.randint(low=0, high=6, size=N)
Y = np.random.randint(low=0, high=6, size=N)
joint_table = pd.crosstab(X, Y, normalize='all')
marginal_X = joint_table.sum(axis=1)
marginal_Y = joint_table.sum(axis=0)
product_marginals = np.outer(marginal_X, marginal_Y)
product_table = pd.DataFrame(product_marginals, index=joint_table.index, columns=joint_table.columns)
difference = np.abs(joint_table - product_table)
print("\nMaximum absolute difference between Joint and Product of Marginals:", difference.values.max())
threshold = 0.05  # set threshold
if difference.values.max() < threshold:
    print("\n✅ X and Y can be considered approximately independent (Random Integers).")
else:
    print("\n❌ X and Y are NOT independent (Random Integers).")
joint_flat = joint_table.stack()
product_flat = product_table.stack()
difference_flat = difference.stack()
comparison_df = pd.DataFrame({
    'Joint P(X,Y)': joint_flat,
    'Product P(X)P(Y)': product_flat,
    'Absolute Difference': difference_flat
})
comparison_df = comparison_df.reset_index()
comparison_df.rename(columns={'index': 'X', 'level_1': 'Y'}, inplace=True)
print("\n📋 Clean Comparison Table (Random Integer case):\n")
print(comparison_df)
comparison_df_sorted = comparison_df.sort_values('Absolute Difference', ascending=False)
print("\nTop 10 biggest deviations (Random Integer case):\n")
print(comparison_df_sorted.head(10))
```

Output

```
Maximum absolute difference between Joint and Product of Marginals: 0.033999999999999999

☑ X and Y can be considered approximately independent (Random Integers).

📋 Clean Comparison Table (Random Integer case):

    row_0  col_0  Joint P(X,Y)  Product P(X)P(Y)  Absolute Difference
0       0      0          0.02            0.0234         3.400000e-03
1       0      1          0.02            0.0299         9.900000e-03
2       0      2          0.01            0.0156         5.600000e-03
3       0      3          0.02            0.0195         5.000000e-04
4       0      4          0.03            0.0234         6.600000e-03
5       0      5          0.03            0.0182         1.180000e-02
6       1      0          0.04            0.0360         4.000000e-03
7       1      1          0.03            0.0460         1.600000e-02
8       1      2          0.02            0.0240         4.000000e-03
9       1      3          0.03            0.0300         6.938894e-18
10      1      4          0.05            0.0360         1.400000e-02
11      1      5          0.03            0.0280         2.000000e-03
12      2      0          0.04            0.0342         5.800000e-03
13      2      1          0.04            0.0437         3.700000e-03
14      2      2          0.03            0.0228         7.200000e-03
15      2      3          0.03            0.0285         1.500000e-03
16      2      4          0.05            0.0342         1.580000e-02
17      2      5          0.00            0.0266         2.660000e-02
18      3      0          0.02            0.0288         8.800000e-03
19      3      1          0.05            0.0368         1.320000e-02
20      3      2          0.03            0.0192         1.080000e-02
21      3      3          0.05            0.0240         2.600000e-02
22      3      4          0.01            0.0288         1.880000e-02
23      3      5          0.00            0.0224         2.240000e-02
24      4      0          0.03            0.0360         6.000000e-03
25      4      1          0.08            0.0460         3.400000e-02
26      4      2          0.03            0.0240         6.000000e-03
27      4      3          0.01            0.0300         2.000000e-02
28      4      4          0.02            0.0360         1.600000e-02
29      4      5          0.03            0.0280         2.000000e-03
30      5      0          0.03            0.0216         8.400000e-03
31      5      1          0.01            0.0276         1.760000e-02
32      5      2          0.00            0.0144         1.440000e-02
33      5      3          0.01            0.0180         8.000000e-03
34      5      4          0.02            0.0216         1.600000e-03
35      5      5          0.05            0.0168         3.320000e-02
```

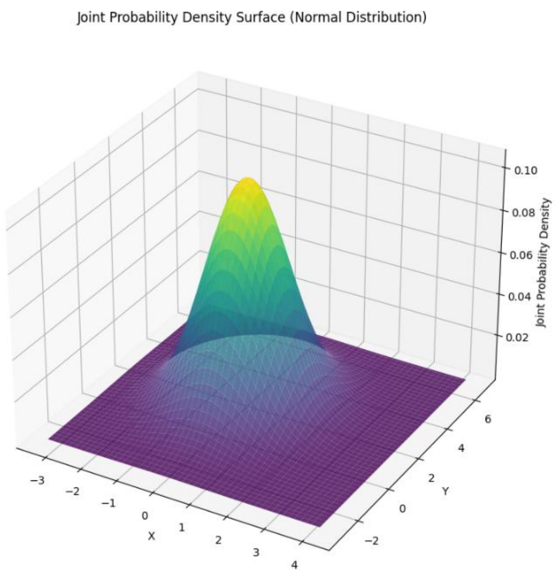
```
27      4      3          0.01            0.0300         2.000000e-02
28      4      4          0.02            0.0360         1.600000e-02
29      4      5          0.03            0.0280         2.000000e-03
30      5      0          0.03            0.0216         8.400000e-03
31      5      1          0.01            0.0276         1.760000e-02
32      5      2          0.00            0.0144         1.440000e-02
33      5      3          0.01            0.0180         8.000000e-03
34      5      4          0.02            0.0216         1.600000e-03
35      5      5          0.05            0.0168         3.320000e-02

Top 10 biggest deviations (Random Integer case):

    row_0  col_0  Joint P(X,Y)  Product P(X)P(Y)  Absolute Difference
25      4      1          0.08            0.0460               0.0340
35      5      5          0.05            0.0168               0.0332
17      2      5          0.00            0.0266               0.0266
21      3      3          0.05            0.0240               0.0260
23      3      5          0.00            0.0224               0.0224
27      4      3          0.01            0.0300               0.0200
22      3      4          0.01            0.0288               0.0188
31      5      1          0.01            0.0276               0.0176
7       1      1          0.03            0.0460               0.0160
28      4      4          0.02            0.0360               0.0160
>>>
```

NORMAL RANDOM VARIABLE GENERATOR

3D PLOT

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from mpl_toolkits.mplot3d import Axes3D
np.random.seed(5678)
# Generate two continuous random variables X and Y (Normal Distribution)
N = 300  # sample size > 200
# Parameters for normal distribution
mu_X, sigma_X = 0, 1
mu_Y, sigma_Y = 2, 1.5
# Generate samples
X = np.random.normal(mu_X, sigma_X, N)
Y = np.random.normal(mu_Y, sigma_Y, N)
# Create grid for evaluating joint PDF
x = np.linspace(min(X)-1, max(X)+1, 100)
y = np.linspace(min(Y)-1, max(Y)+1, 100)
X_grid, Y_grid = np.meshgrid(x, y)
# Calculate joint PDF assuming independence
pdf_X = norm.pdf(X_grid, mu_X, sigma_X)
pdf_Y = norm.pdf(Y_grid, mu_Y, sigma_Y)
joint_pdf = pdf_X * pdf_Y
fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X_grid, Y_grid, joint_pdf, cmap='viridis', edgecolor='none', alpha=0.8)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Joint Probability Density')
ax.set_title('Joint Probability Density Surface (Normal Distribution)')
```

OUTPUT

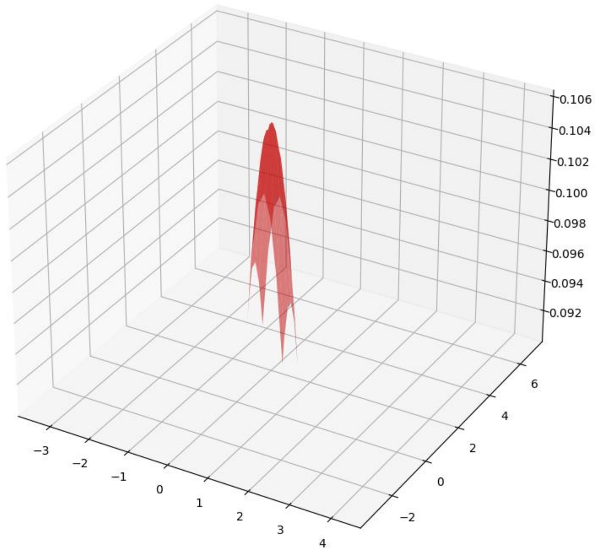


Highlight Region [a, b] and [c, d]

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from mpl_toolkits.mplot3d import Axes3D
np.random.seed(5678)
# Generate two continuous random variables X and Y (Normal Distribution)
N = 300  # sample size > 200
# Parameters for normal distribution
mu_X, sigma_X = 0, 1
mu_Y, sigma_Y = 2, 1.5
# Generate samples
X = np.random.normal(mu_X, sigma_X, N)
Y = np.random.normal(mu_Y, sigma_Y, N)
# Create grid for evaluating joint PDF
x = np.linspace(min(X)-1, max(X)+1, 100)
y = np.linspace(min(Y)-1, max(Y)+1, 100)
X_grid, Y_grid = np.meshgrid(x, y)
# Calculate joint PDF assuming independence
pdf_X = norm.pdf(X_grid, mu_X, sigma_X)
pdf_Y = norm.pdf(Y_grid, mu_Y, sigma_Y)
joint_pdf = pdf_X * pdf_Y
# Define region limits
a = mu_X - 0.5
b = mu_X + 0.5
c = mu_Y - 0.5
d = mu_Y + 0.5
# Mask the region
region_mask = (X_grid >= a) & (X_grid <= b) & (Y_grid >= c) & (Y_grid <= d)
# Print selected limits
print(f"\nSelected Region:\nX in [{a:.2f}, {b:.2f}], Y in [{c:.2f}, {d:.2f}]")
# Plot region
fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(
    X_grid, Y_grid,
    np.where(region_mask, joint_pdf, np.nan),
    color='red', alpha=0.5
)
# Show plot
plt.tight_layout()
plt.show()
```

OUTPUT

```
Selected Region:
X in [-0.50, 0.50], Y in [1.50, 2.50]
```



Problem 3: Hypothesis testing

From an experiment of tossing of a coin n times we observe x number of heads (success) with probability of head being p_0. Perform a hypothesis test with

Null hypothesis $H_0: p = p_0$, and

Alternate hypothesis (i) $H_1: p \neq p_0$ Two-tailed test

(ii)$H_1: p > p_0$ one-tailed test (right tail)

(ii)$H_1: p < p_0$ one-tailed test (left tail)

Find P-value and make a decision with level of significance $\alpha = 0.05$ and 0.01.

Also calculate z-value for the experiment where $z = (\hat{p} - p_0)/\sqrt{p_0(1-p_0)/n}$ with statistical proportion $\hat{p} = x/n$. Compare it with critical value of z (i.e. $z_{\alpha/2}$ or z_α) to make your decision.

$critical\ z$	$\alpha = 0.05$	$\alpha = 0.01$
$z_{\alpha/2}$	1.96	2.575
z_α	1.645	2.325

Reference: Probability & Statistics for Engineers &; Scientists; Ronald E. Walpole, R.H. Myers …

Solution

```python
import math
from scipy.stats import norm
def hypothesis_test_coin_toss(x, n, p0):
    # Step 1: Calculate sample proportion
    p_hat = x / n
    # Step 2: Calculate z-value
    standard_error = math.sqrt(p0 * (1 - p0) / n)
    z = (p_hat - p0) / standard_error
    # Step 3: Calculate P-values
    p_value_two_tailed = 2 * (1 - norm.cdf(abs(z)))
    p_value_right_tailed = 1 - norm.cdf(z)
    p_value_left_tailed = norm.cdf(z)

    # Step 4: Critical values
    critical_values = {
        0.05: {"two_tailed": 1.96, "one_tailed": 1.645},
        0.01: {"two_tailed": 2.575, "one_tailed": 2.325}
    }

    # Step 5: Display clean output
    print("\n" + "="*50)
    print(" HYPOTHESIS TESTING REPORT ".center(50, "="))
    print("="*50)
    print(f"Sample size (n): {n}")
    print(f"Number of successes (x): {x}")
    print(f"Hypothesized proportion (p₀): {p0}")
    print("-"*50)
    print(f"Sample proportion (p̂): {p_hat:.4f}")
    print(f"Standard Error (SE): {standard_error:.4f}")
    print(f"z-value: {z:.4f}")
    print("-"*50)
    print(f"P-value (Two-tailed): {p_value_two_tailed:.4f}")
    print(f"P-value (Right-tailed): {p_value_right_tailed:.4f}")
    print(f"P-value (Left-tailed): {p_value_left_tailed:.4f}")
    print("="*50)

    for alpha in [0.05, 0.01]:
        print(f"\n{'*'*50}")
        print(f"DECISION AT SIGNIFICANCE LEVEL α = {alpha}".center(50))
        print('*'*50)

        # Two-tailed Test
        decision_two_p = "Reject H₀" if p_value_two_tailed <= alpha else "Do not reject H₀"
        decision_two_z = "Reject H₀" if abs(z) > critical_values[alpha]["two_tailed"] else "Do not reject H₀"

        # Right-tailed Test
        decision_right_p = "Reject H₀" if p_value_right_tailed <= alpha else "Do not reject H₀"
```

```python
        # Two-tailed Test
        decision_two_p = "Reject H₀" if p_value_two_tailed <= alpha else "Do not reject H₀"
        decision_two_z = "Reject H₀" if abs(z) > critical_values[alpha]["two_tailed"] else "Do not reject H₀"

        # Right-tailed Test
        decision_right_p = "Reject H₀" if p_value_right_tailed <= alpha else "Do not reject H₀"
        decision_right_z = "Reject H₀" if z > critical_values[alpha]["one_tailed"] else "Do not reject H₀"

        # Left-tailed Test
        decision_left_p = "Reject H₀" if p_value_left_tailed <= alpha else "Do not reject H₀"
        decision_left_z = "Reject H₀" if z < -critical_values[alpha]["one_tailed"] else "Do not reject H₀"

        # Print all decisions
        print(f"Two-tailed Test (P-value method): {decision_two_p}")
        print(f"Two-tailed Test (Critical value method): {decision_two_z}")
        print("-"*50)
        print(f"Right-tailed Test (P-value method): {decision_right_p}")
        print(f"Right-tailed Test (Critical value method): {decision_right_z}")
        print("-"*50)
        print(f"Left-tailed Test (P-value method): {decision_left_p}")
        print(f"Left-tailed Test (Critical value method): {decision_left_z}")
        print("*"*50)

# Example usage
# x = number of heads observed
# n = number of tosses
# p0 = hypothesized probability of head (under H0)

x = 32
n = 50
p0 = 0.5

hypothesis_test_coin_toss(x, n, p0)
```

Output

```
======================================================= RESTART: C:\

==================================================
=========== HYPOTHESIS TESTING REPORT ============
==================================================
Sample size (n): 50
Number of successes (x): 32
Hypothesized proportion (p₀): 0.5
--------------------------------------------------
Sample proportion (p^): 0.6400
Standard Error (SE): 0.0707
z-value: 1.9799
--------------------------------------------------
P-value (Two-tailed): 0.0477
P-value (Right-tailed): 0.0239
P-value (Left-tailed): 0.9761
==================================================

**************************************************
     DECISION AT SIGNIFICANCE LEVEL α = 0.05
**************************************************
Two-tailed Test (P-value method): Reject H₀
Two-tailed Test (Critical value method): Reject H₀
--------------------------------------------------
Right-tailed Test (P-value method): Reject H₀
Right-tailed Test (Critical value method): Reject H₀
--------------------------------------------------
Left-tailed Test (P-value method): Do not reject H₀
Left-tailed Test (Critical value method): Do not reject H₀
**************************************************

**************************************************
     DECISION AT SIGNIFICANCE LEVEL α = 0.01
**************************************************
Two-tailed Test (P-value method): Do not reject H₀
Two-tailed Test (Critical value method): Do not reject H₀
--------------------------------------------------
Right-tailed Test (P-value method): Do not reject H₀
Right-tailed Test (Critical value method): Do not reject H₀
--------------------------------------------------
Left-tailed Test (P-value method): Do not reject H₀
Left-tailed Test (Critical value method): Do not reject H₀
**************************************************
>>>
```

Problem 4: Bayesian Inference

From an experiment of flipping of a coin N times M heads showed up with statistical proportion $\hat{\theta} = M/N$. With the prior distribution $(\pi(\theta))$ given below

(a) Beta distribution $B(\theta; \alpha, \beta)$ with given values of α, β

(b) Gaussian distribution $n(\theta; \mu, \sigma)$ with a given mean μ) and standard deviation σ.

perform following

i) Plot the prior distributions $(\pi(\theta))$ with θ.

ii) Plot the likelihood $l(\theta|x)$ with θ and determine the value of θ that maximizes the probability of the data.

iii) Plot the posterior distribution $\pi(\theta|x) = \pi(\theta)l(\theta|x)/g(x)$, where $g(x) = \sum_\theta \pi(\theta)l(\theta|x)$ is marginal distribution.

iv) Make a single plot for posterior distribution for both the priors with the following values of $(M, N) = (0,0); (1,1); (2,2); (2,3); (2,4); (3,8); (5,16); (10,32); 20,64); (40,128); (80,256); (160,512); (320,1024); (640,2048); (1280,4096)$, discuss the behaviour of posterior distribution by increasing number of trials and number of successes keeping statistical proportion almost constant.

Solution

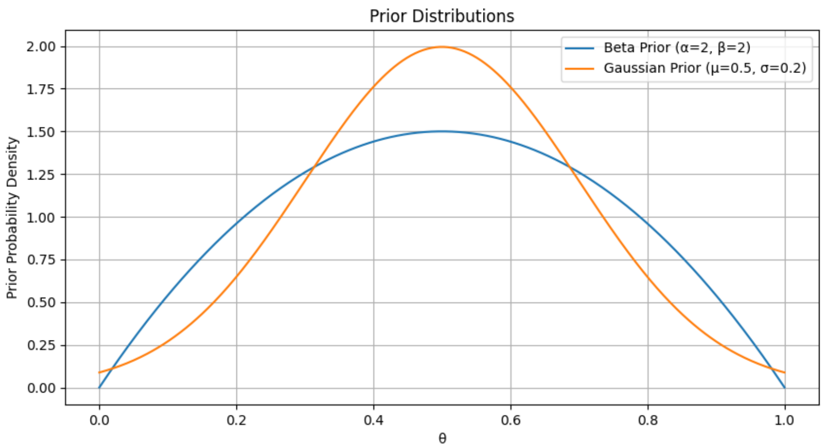
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta, norm
theta = np.linspace(0, 1, 500)  # θ varies between 0 and 1
# Prior parameters
alpha_prior = 2
beta_prior = 2
mu_prior = 0.5
sigma_prior = 0.2
# Function: Likelihood for Binomial trials
def likelihood(theta, M, N):
    return theta**M * (1 - theta)**(N - M)
# Function: Posterior Calculation
def posterior(prior_func, theta, M, N):
    lik = likelihood(theta, M, N)
    numerator = prior_func(theta) * lik
    denominator = np.trapz(numerator, theta)
    return numerator / denominator
# Prior Functions
prior_beta = lambda t: beta.pdf(t, alpha_prior, beta_prior)
prior_gauss = lambda t: norm.pdf(t, mu_prior, sigma_prior)
# Example: Pick M and N
M = 5
N = 10
theta_MLE = M / N
# --------
# (i) Plot Priors
# --------
plt.figure(figsize=(10,5))
plt.plot(theta, prior_beta(theta), label=f'Beta Prior (α={alpha_prior}, β={beta_prior})')
plt.plot(theta, prior_gauss(theta), label=f'Gaussian Prior (μ={mu_prior}, σ={sigma_prior})')
plt.title("Prior Distributions")
plt.xlabel('θ')
plt.ylabel('Prior Probability Density')
plt.legend()
plt.grid()
plt.show()

# --------
# (ii) Plot Likelihood
# --------
plt.figure(figsize=(10,5))
plt.plot(theta, likelihood(theta, M, N), color='purple', label=f'Likelihood (M={M}, N={N})')
plt.axvline(theta_MLE, color='red', linestyle='--', label=f'MLE θ = {theta_MLE:.2f}')
plt.title("Likelihood Function")
plt.xlabel('θ')
plt.ylabel('Likelihood')
```

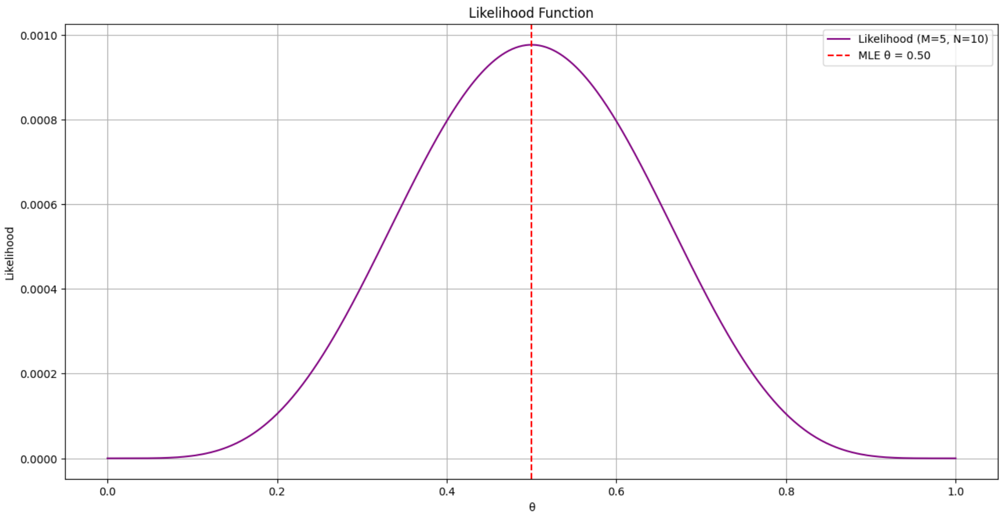
File Edit Format Run Options Window Help

```
plt.plot(theta, prior_gauss(theta), label=f'Gaussian Prior (μ={mu_prior}, σ={sigma_prior})')
plt.title("Prior Distributions")
plt.xlabel('θ')
plt.ylabel('Prior Probability Density')
plt.legend()
plt.grid()
plt.show()

# --------
# (ii) Plot Likelihood
# --------
plt.figure(figsize=(10,5))
plt.plot(theta, likelihood(theta, M, N), color='purple', label=f'Likelihood (M={M}, N={N})')
plt.axvline(theta_MLE, color='red', linestyle='--', label=f'MLE θ = {theta_MLE:.2f}')
plt.title("Likelihood Function")
plt.xlabel('θ')
plt.ylabel('Likelihood')
plt.legend()
plt.grid()
plt.show()

# --------
# (iii) Plot Posterior
# --------
posterior_beta = posterior(prior_beta, theta, M, N)
posterior_gauss = posterior(prior_gauss, theta, M, N)

plt.figure(figsize=(10,5))
plt.plot(theta, posterior_beta, label='Posterior with Beta Prior')
plt.plot(theta, posterior_gauss, label='Posterior with Gaussian Prior')
plt.title(f"Posterior Distributions (M={M}, N={N})")
plt.xlabel('θ')
plt.ylabel('Posterior Probability Density')
plt.legend()
plt.grid()
plt.show()

# --------
# (iv) Single plot for multiple (M,N)
# --------
# Given M, N pairs
MN_pairs = [
    (0,0), (1,1), (2,2), (2,3), (2,4), (3,8), (5,16), (10,32),
    (20,64), (40,128), (80,256), (160,512), (320,1024), (640,2048), (1280,4096)
]

# Plotting
```

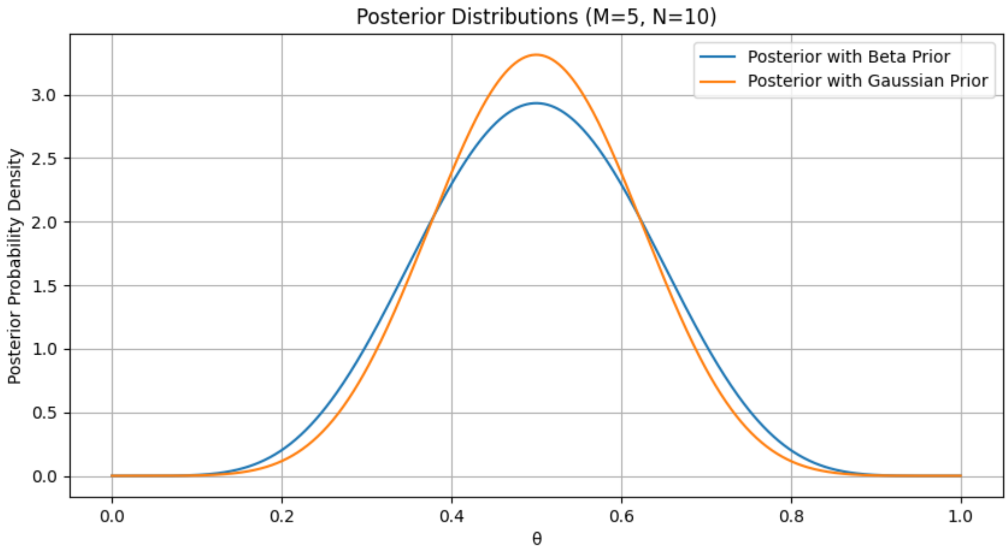
```
# Plotting
fig, axs = plt.subplots(5, 3, figsize=(20, 20))
axs = axs.ravel()

for i, (M, N) in enumerate(MN_pairs):
    if N == 0:
        continue
    post_beta = posterior(prior_beta, theta, M, N)
    post_gauss = posterior(prior_gauss, theta, M, N)

    axs[i].plot(theta, post_beta, label='Beta Prior')
    axs[i].plot(theta, post_gauss, label='Gaussian Prior')
    axs[i].set_title(f"M={M}, N={N}, θ={M/N:.2f}")
    axs[i].set_xlim(0,1)
    axs[i].legend()
    axs[i].grid()

plt.tight_layout()
plt.suptitle("Posterior Distributions for Different (M,N)", y=1.02, fontsize=20)
plt.show()
```

Output

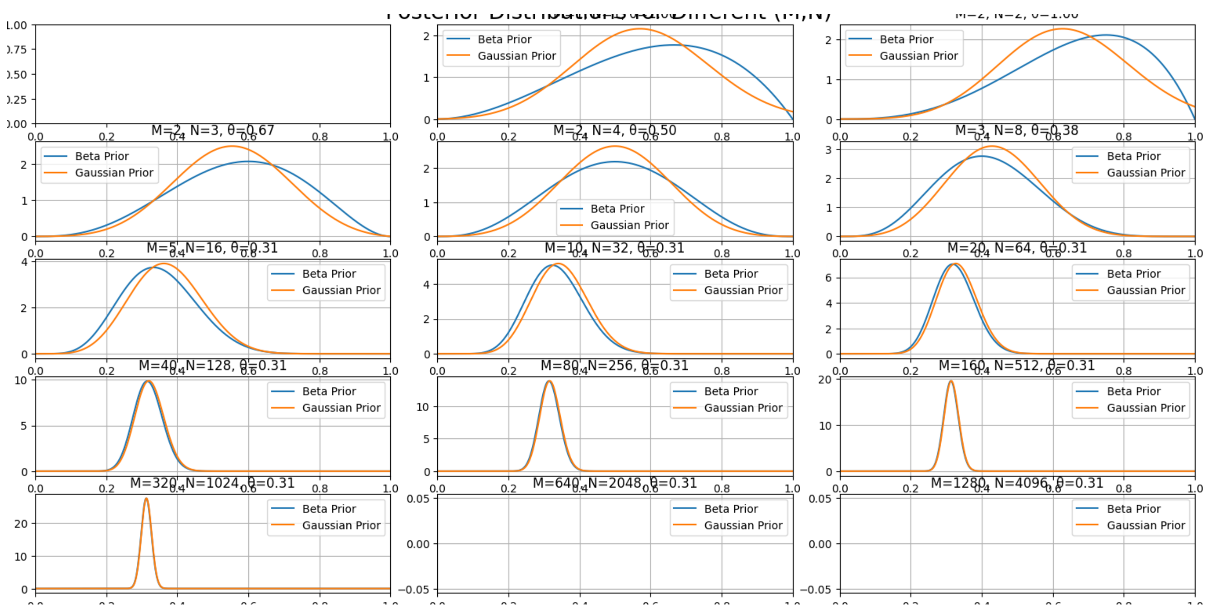




Likelihood Function
Likelihood (M=5, N=10)
MLE θ = 0.50
Likelihood
θ



Posterior Distributions (M=5, N=10)
Posterior with Beta Prior
Posterior with Gaussian Prior
Posterior Probability Density
θ



M=2, N=3, θ=0.67
M=2, N=4, θ=0.50
M=3, N=8, θ=0.38
M=5, N=16, θ=0.31
M=10, N=32, θ=0.31
M=20, N=64, θ=0.31
M=40, N=128, θ=0.31
M=80, N=256, θ=0.31
M=160, N=512, θ=0.31
M=320, N=1024, θ=0.31
M=640, N=2048, θ=0.31
M=1280, N=4096, θ=0.31
Beta Prior
Gaussian Prior

- *As* ***N increases****, the posterior becomes* ***sharper and narrower*** *around the true ϑ = M/N.*
- ***Prior*** *effect* ***decreases*** *as N becomes large (data dominates).*
- *When N is* ***small****, prior shape* ***significantly affects*** *the posterior.*
- *The statistical proportion (M/N) is kept* ***roughly constant****, so posterior centres remain near the same value, but become much* ***tighter****.*