Colab Link: https://colab.research.google.com/drive/1i_gbcvK8vAbeZbMqI94lxCaXpEMtZW1-?usp=sharing

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

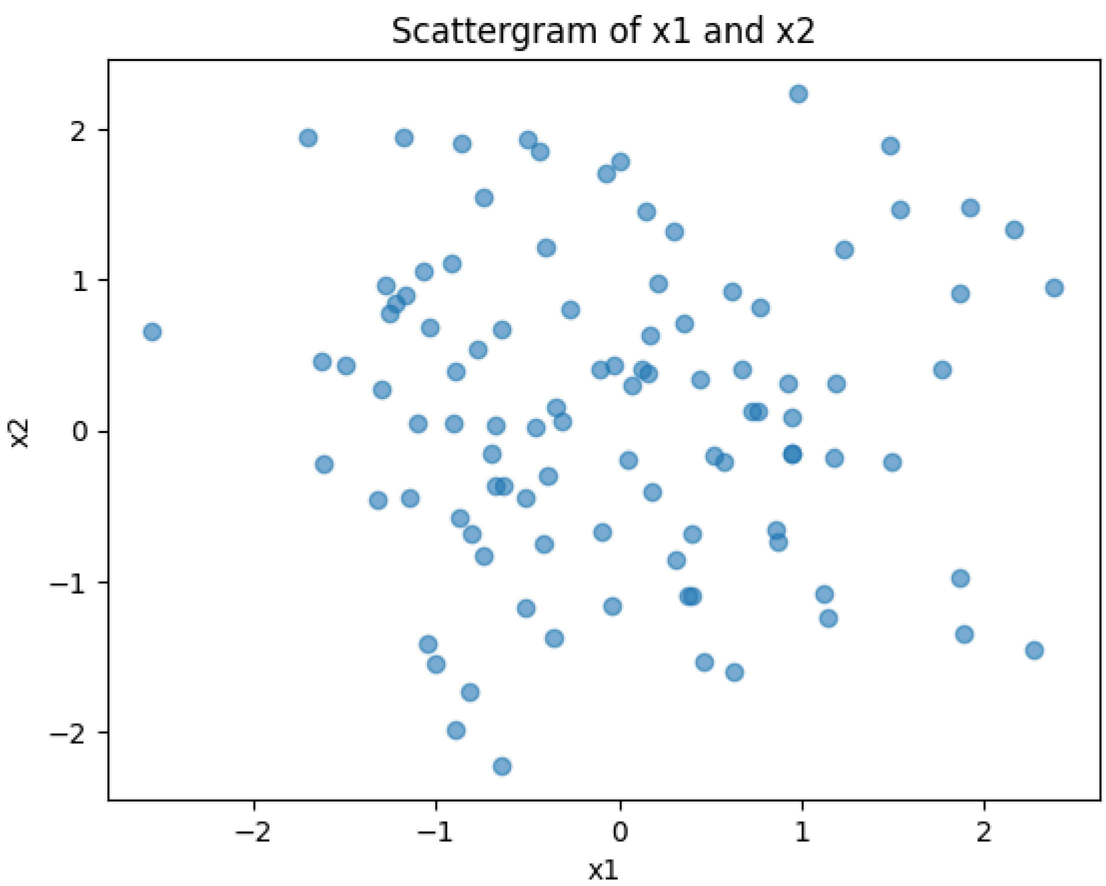
Q1

In [ ]:
```
#a
np.random.seed(0)

mean = [0,0]
cov_matrix = [[1, 0],
              [0, 1]]
x = np.random.multivariate_normal(mean, cov_matrix, size=100)
```

In [ ]:
```
x1 = x[:,0]
x2 = x[:,1]
print(x.shape)
print(x1.shape)
print(x2.shape)
```

```
(100, 2)
(100,)
(100,)
```

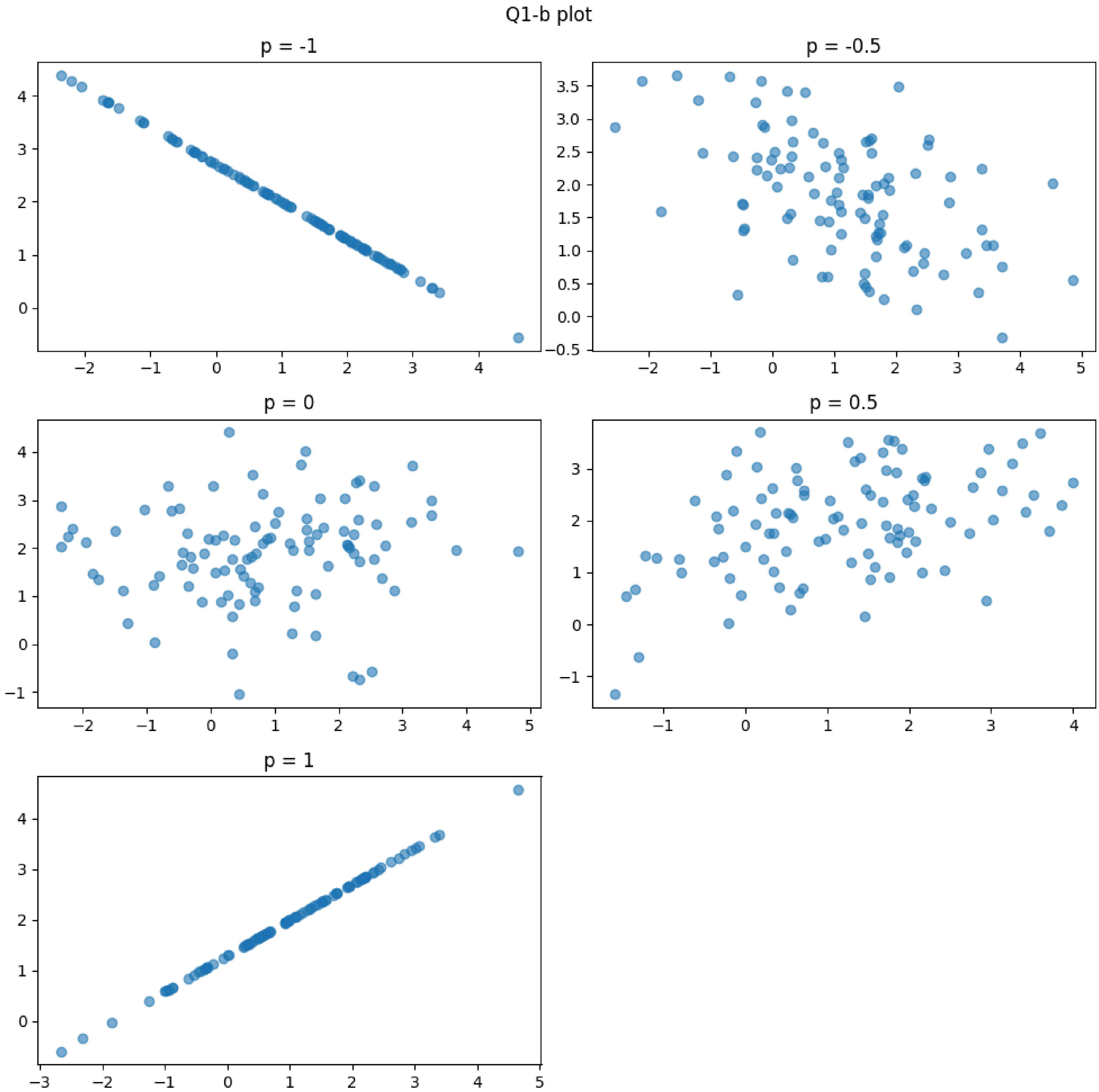
In [ ]:
```
plt.scatter(x1, x2, alpha=0.6)

plt.title('Scattergram of x1 and x2')
plt.xlabel('x1')
plt.ylabel('x2')

plt.show()
```



Explanation: The dots are scattered randomly, which is expected as the two variables are uncorrelated. To clarify, an increase in x1 does not affect x2.

In [ ]:
```
#b
np.random.seed(0)

def generate_data(p):
    cov_x1_x2 = p*(math.sqrt(2))*1
    mean = [1,2]
    cov_matrix = [[2, cov_x1_x2],
                  [cov_x1_x2, 1]]
    X = np.random.multivariate_normal(mean, cov_matrix, size=100)
    x1 = X[:,0]
    x2 = X[:,1]
    return x1,x2
p = [-1,-0.5,0,0.5,1]

fig, axes = plt.subplots(3, 2, figsize=(10, 10))
axes = axes.flatten()

for i,p_value in enumerate(p):
    x1,x2 = generate_data(p_value)
    axes[i].scatter(x1,x2,alpha=0.6)
```

```
    axes[i].set_title(f"p = {p_value}")

fig.delaxes(axes[-1])

fig.suptitle("Q1-b plot")
plt.tight_layout()
plt.show()
```



Explanation: A negative p shows the trend of x2 decreasing as x1 increases.

A positive p shows the trend of x2 increasing as x1 increases.

The further p is from 0, the easier the trends can be seen.

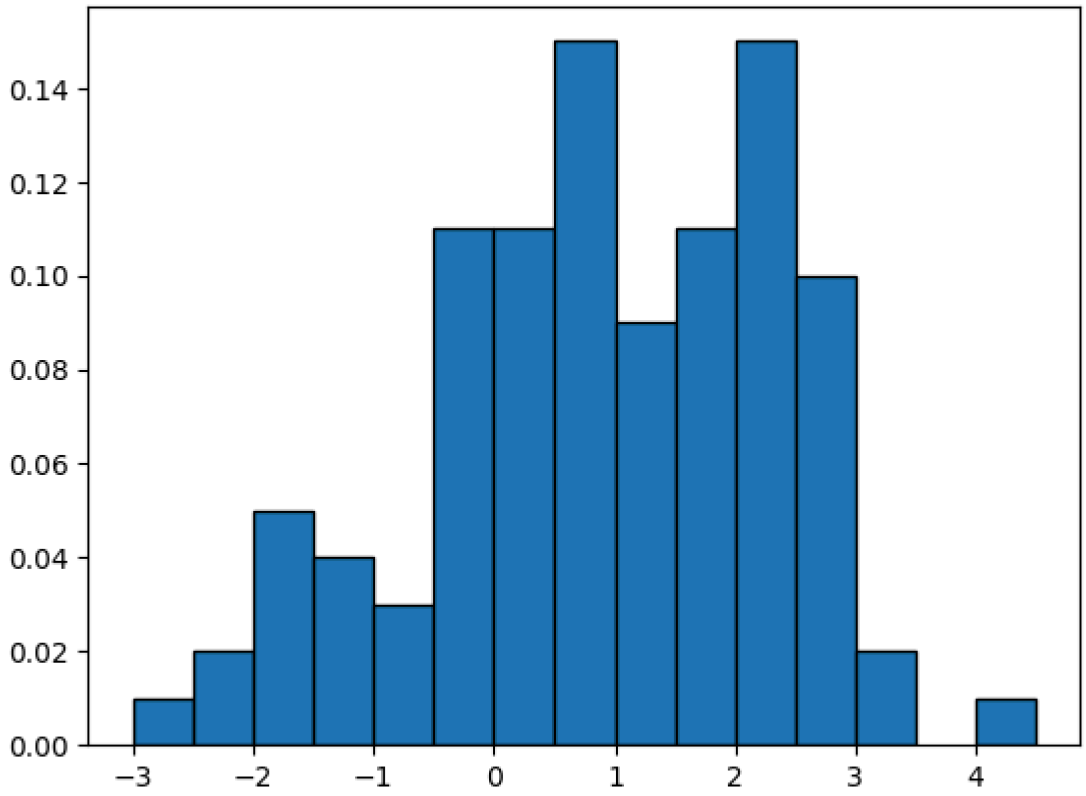
In [ ]:
```
#c
np.random.seed(0)

#I will construct histogram with interval of 0.5
x1,x2 = generate_data(0.5)
x1 = (x1//0.5)*0.5
x2 = (x2//0.5)*0.5
#0 represents 0-0.5 / 0.5 represents 0.5-1
```

In [ ]:
```
unique_values, counts = np.unique(x1, return_counts=True)
total_count = 0
for count in counts:
    total_count += count
probabilities = counts/total_count

pdf_x1 = list(zip(unique_values, probabilities))
pdf_x1
```

Out[ ]:
```
[(-3.0, 0.01),
 (-2.5, 0.02),
 (-2.0, 0.05),
 (-1.5, 0.04),
 (-1.0, 0.03),
 (-0.5, 0.11),
 (0.0, 0.11),
 (0.5, 0.15),
 (1.0, 0.09),
 (1.5, 0.11),
 (2.0, 0.15),
 (2.5, 0.1),
 (3.0, 0.02),
 (4.0, 0.01)]
```

In [ ]:
```
bin = [x[0] for x in pdf_x1]
frequencies = [x[1] for x in pdf_x1]

# Plot the histogram as a bar plot
plt.bar(bin, frequencies, width=0.5, align='edge', edgecolor='black')
```

Out[ ]: <BarContainer object of 14 artists>



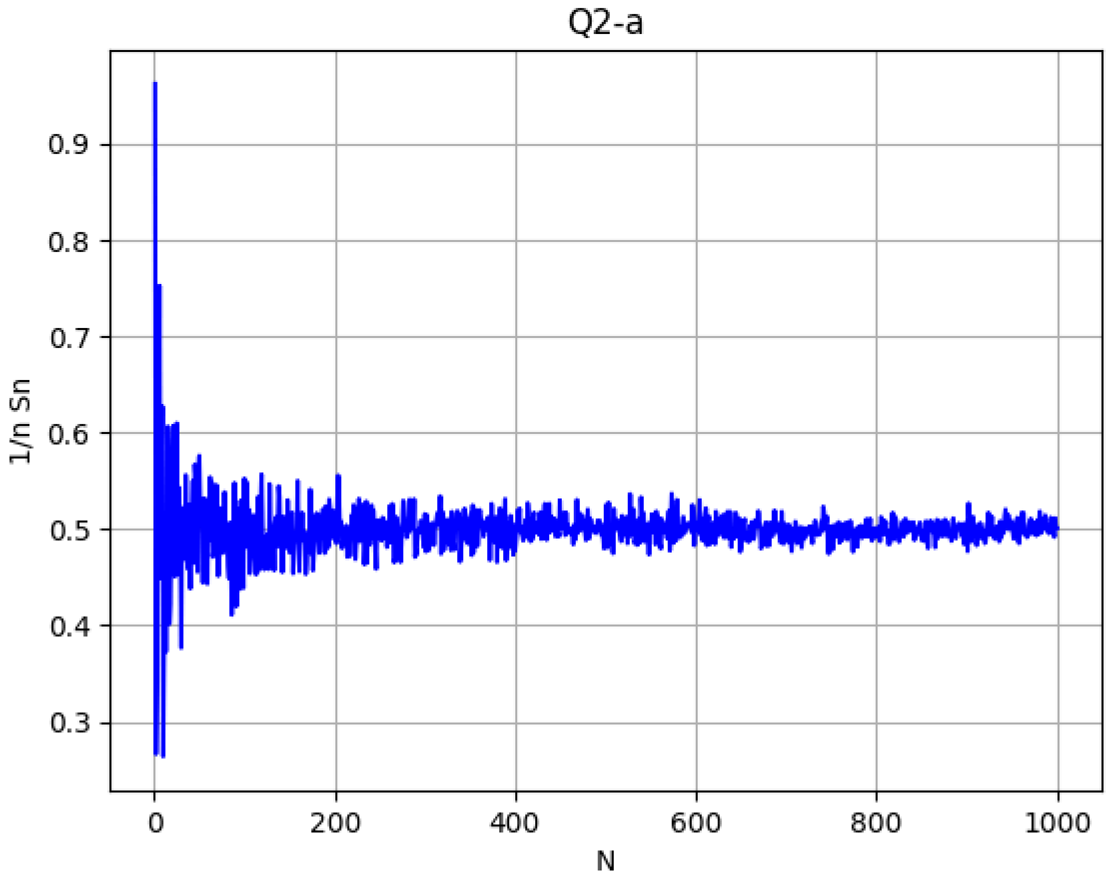
In [ ]:
```
#expected value
temp = [(x[0]+0.25)*x[1] for x in pdf_x1]
E_x1 = np.sum(temp)
print(f"Expected Value : {E_x1}")
#Variance
temp2 = [((x[0]+0.25)**2)*x[1] for x in pdf_x1]
E_x1_squared = np.sum(temp2)
Var_x1 = E_x1_squared - (E_x1**2)
print(f"Variance : {Var_x1}")
```

```
Expected Value : 0.905
Variance : 2.0734750000000006
```

Explanation: From the histogram, the shape fairly reflects Gaussian distribution of X1. The calculated expected value of 0.905 and variance of 2.07 are also very close to the real expected value and variance of X1.

Q2

In [ ]:
```
#a
def generate_s(n):
    s = np.random.rand(n)
    s = np.sum(s)
    return s/n
N = range(1,1001)
S = []
for i in N:
    S.append(generate_s(i))


plt.plot(N, S, color='b')

plt.xlabel('N')
plt.ylabel('1/n Sn')
plt.title("Q2-a")
plt.grid(True)

plt.show()
```



In [ ]:
```
S[-5:]
```

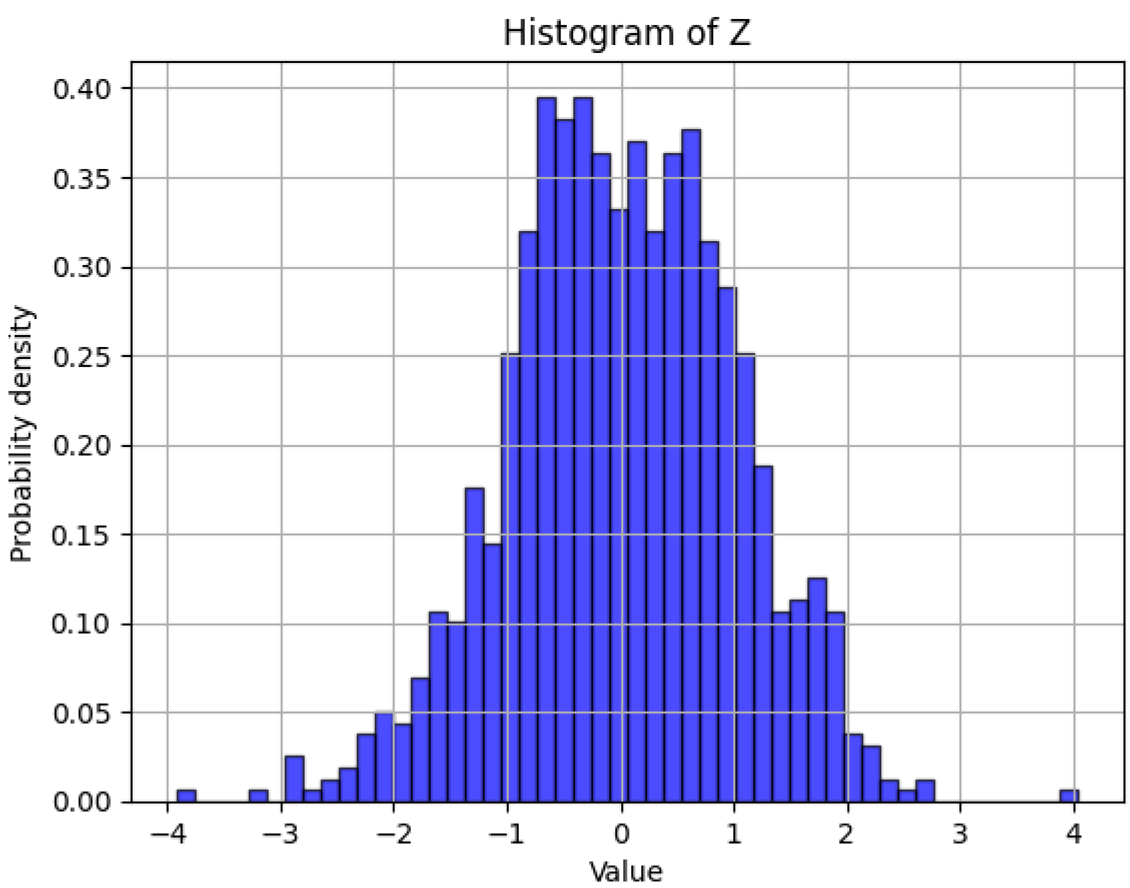
Out[ ]:
```
[0.5010805593882334,
 0.49184026151878674,
 0.511451814034396,
 0.5006343749384916,
 0.5007174085217688]
```

Explanation: It converges to 0.5

In [ ]:
```
#b
S = []
for i in range(1000):
    S.append((generate_s(100)*100))
S = np.array(S)
Z = (S-50)/(math.sqrt(100/12))
Z.shape
```

Out[ ]:
```
(1000,)
```

In [ ]:
```
plt.hist(Z, bins=50, color='b', alpha=0.7, edgecolor='black', density=True)
plt.xlabel('Value')
plt.ylabel('Probability density')
plt.title('Histogram of Z')

plt.grid(True)

plt.show()
```



In [ ]:
```
#c
range_Z = max(Z)- min(Z)
```

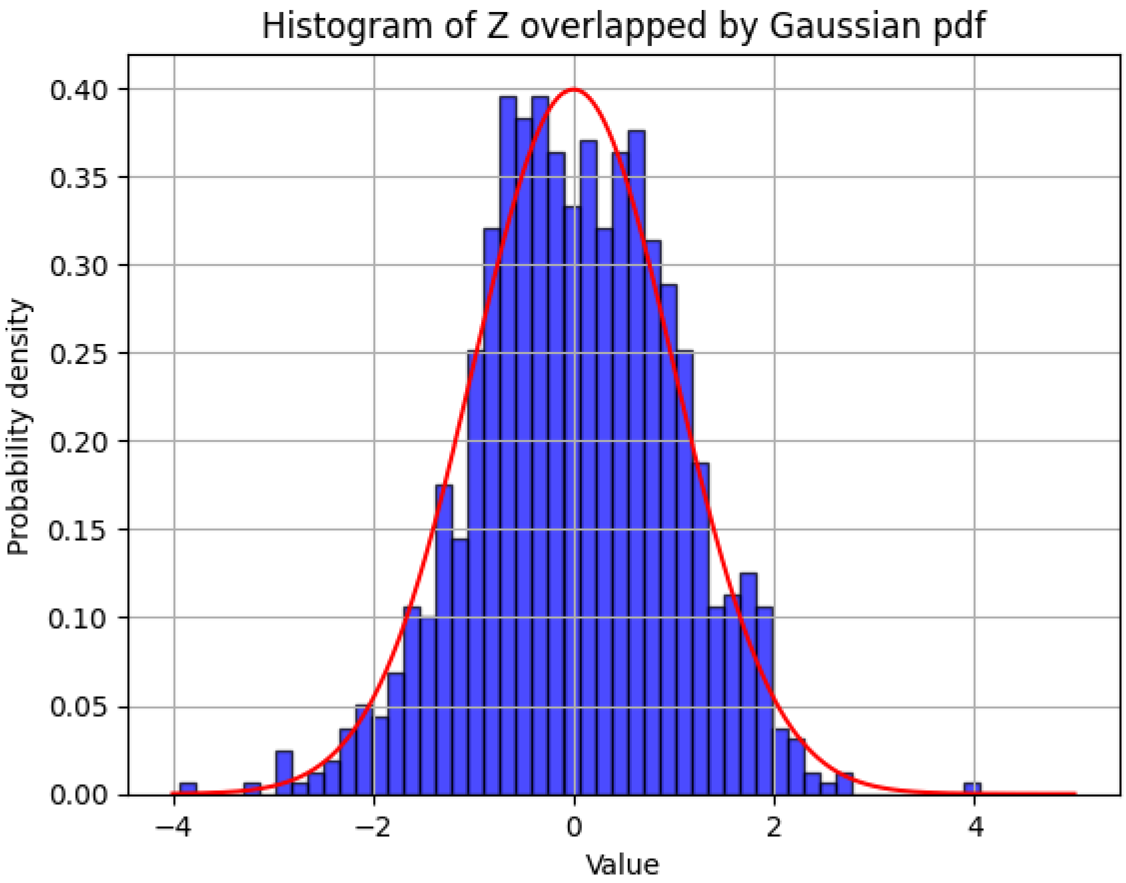
```
range_Z/50
```

Out[ ]: 0.1593972775731772

In [ ]:
```
z = np.linspace(math.floor(min(Z)), math.ceil(max(Z)), 1000)
numerator = math.e**(-0.5*(z**2))
gaussian_pdf = numerator/math.sqrt(2*math.pi)
gaussian_pdf[:5]
```

Out[ ]: array([0.00013383, 0.00013874, 0.00014381, 0.00014905, 0.00015448])

In [ ]:
```
plt.hist(Z, bins=50, color='b', alpha=0.7, edgecolor='black', density=True)
plt.plot(z, gaussian_pdf, color='r', label='Gaussian PDF')

plt.xlabel('Value')
plt.ylabel('Probability density')
plt.title('Histogram of Z overlapped by Gaussian pdf')

plt.grid(True)

plt.show()
```



Explanation: The distribution of Z closely reflects Gaussian distribution with 0 mean and variance = 1

In [ ]:
```
#d

#I will separate Z with range of 0.2
Z = (Z//0.2)*0.2
Z[:5]

#0 represents 0-0.2 / 0.2 represents 0.2-0.4
```

Out[ ]: array([0.8, -0.8, -1.6, 0.2, -0.6])

In [ ]:
```
unique_values, counts = np.unique(Z, return_counts=True)
total_count = 0
for count in counts:
    total_count += count

probabilities = counts/total_count

pdf_Z = list(zip(unique_values, probabilities))
for i in range(len(pdf_Z)):
    pdf_Z[i] = list(pdf_Z[i])
    pdf_Z[i][0] = round(pdf_Z[i][0],2)
pdf_Z
```

Out[ ]:
```
[[-4.0, 0.001],
 [-3.4, 0.001],
 [-3.0, 0.004],
 [-2.8, 0.001],
 [-2.6, 0.002],
 [-2.4, 0.007],
 [-2.2, 0.01],
 [-2.0, 0.009],
 [-1.8, 0.015],
 [-1.6, 0.026],
 [-1.4, 0.03],
 [-1.2, 0.035],
 [-1.0, 0.054],
 [-0.8, 0.076],
 [-0.6, 0.078],
 [-0.4, 0.084],
 [-0.2, 0.058],
 [0.0, 0.074],
 [0.2, 0.067],
 [0.4, 0.072],
 [0.6, 0.073],
 [0.8, 0.055],
 [1.0, 0.053],
 [1.2, 0.034],
 [1.4, 0.024],
 [1.6, 0.022],
 [1.8, 0.021],
 [2.0, 0.004],
 [2.2, 0.005],
 [2.4, 0.002],
 [2.6, 0.002],
 [4.0, 0.001]]
```

In [ ]:
```python
#expected value
temp = [(z[0]+0.1)*z[1] for z in pdf_Z]
E_z = np.sum(temp)
print(f"Expected Value : {E_z}")
#Variance
temp2 = [((z[0]+0.1)**2)*z[1] for z in pdf_Z]
E_z_squared = np.sum(temp2)
Var_z = E_z_squared - (E_z**2)
print(f"Variance : {Var_z}")
```

```
Expected Value : 0.019800000000000026
Variance : 1.0043279600000001
```

Explanation:

Estimated Expected Value = 0.0198

Theoretical Expected Value = 0

Estimated Variance = 1.004

Theoretical Variance = 1

In [ ]: