Statistical inference

Assignment 1

Imports

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
```

Part A

A:1

```
# Urn
values = np.asarray([1,2,3])
counts = np.asarray([2, 3, 195])
urn_A1 = np.repeat(values, counts)

# Draw with replacement
N = 100
draw_w_rep = np.random.choice(urn_A1, size = N, p = None, replace = True)
print(draw_w_rep)
```

```
[3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
```

A:2

```
# Urn
values = np.asarray([54, 66, 70, 72, 97])
counts = np.asarray([1, 1, 1, 1, 1])
urn_A2 = np.repeat(values, counts)
```

```
N = 3

casette_times = []
for i in range(100000):
    casette_run = np.random.choice(urn_A2, size=N, p=None, replace=True)
    casette_times.append(sum(casette_run))
```

```
percent_longer_than_200 = len([v for v in casette_times if v > 200]) / len(
    [v for v in casette_times if v <= 200]
)
print(percent_longer_than_200)
```

```
2.3085194375516958
```

A:3

```
# Urn
values = np.asarray(["White", "Red"])
counts = np.asarray([7, 93])
urn_A3 = np.repeat(values, counts)
```

```
N = 96

well_plates = []
for i in range(1000):
    well_plate_96 = np.random.choice(urn_A3, size=N, p=None, replace=True)
    red = [True for r in well_plate_96 if r == "Red"]
    # well_plates.append(sum(red) / (1 - sum(red)))
    well_plates.append(sum(red))
```

```
plt.hist(well_plates, bins=16, edgecolor="black")
plt.title("Pipetting robot accuracy distribution")
plt.xlabel("n correct wells")
plt.ylabel("n plates")
plt.show()
```

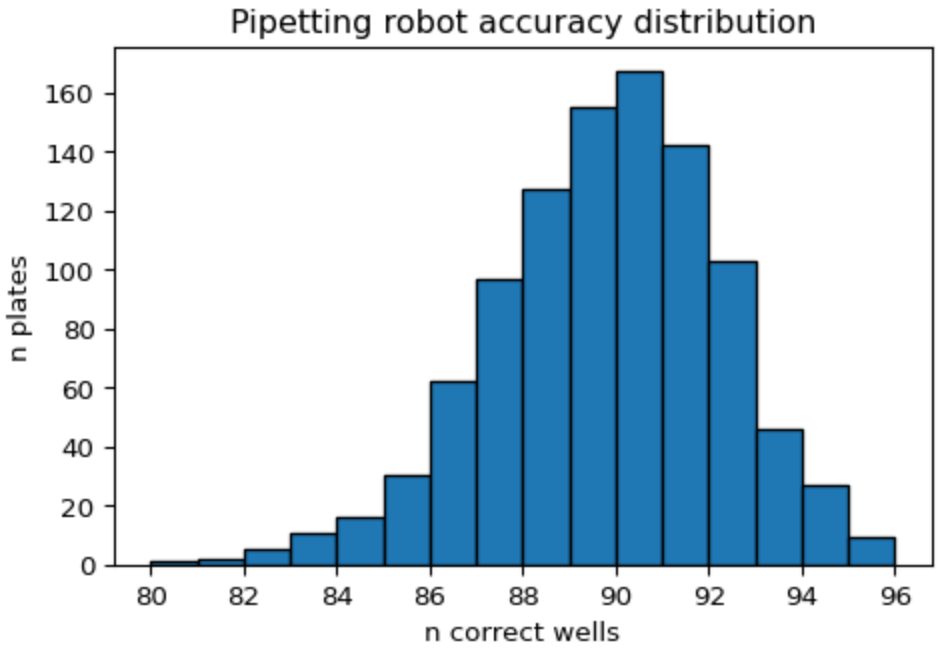

Figure 1: Histogram showing distribution of pipetting accuracy.

Part B

B:1

a)

```
mu = 0
sigma = 2
N = 1000
x_axis = np.linspace(-5, 5, N)
pdf_normal = stats.norm.pdf(x_axis, mu, sigma)
plt.plot(x_axis, pdf_normal)
plt.ylabel("PDF")
plt.show()
```

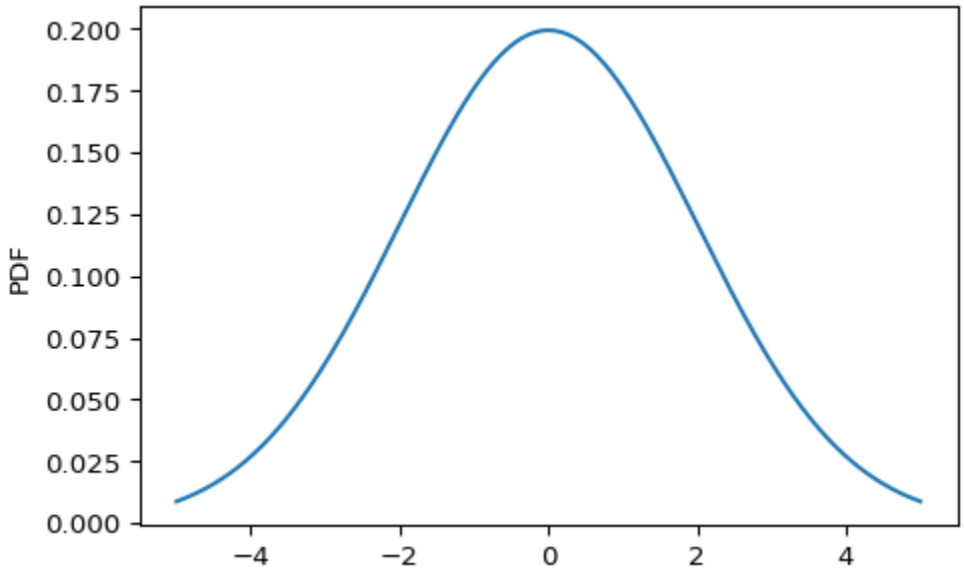

Figure 2: Normal distribution with mu = 0 and sigma = 2.

```
alpha = 2
beta = 5
N = 1000
x_axis = np.linspace(0, 1, N)
pdf_beta = stats.beta.pdf(x_axis, alpha, beta)
plt.plot(x_axis, pdf_beta)
plt.ylabel("PDF")
plt.show()
```

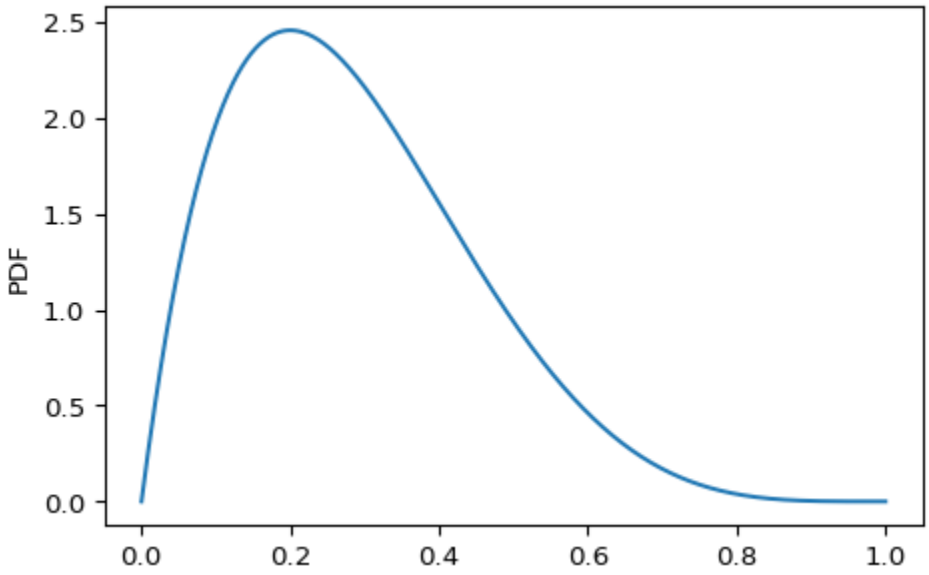

Figure 3: Beta distribution with alpha = 2 and beta = 5.

```
alpha = 3
beta = 3
N = 1000
x_axis = np.linspace(0, 1, N)
pdf_beta = stats.beta.pdf(x_axis, alpha, beta)
plt.plot(x_axis, pdf_beta)
plt.ylabel("PDF")
plt.show()
```

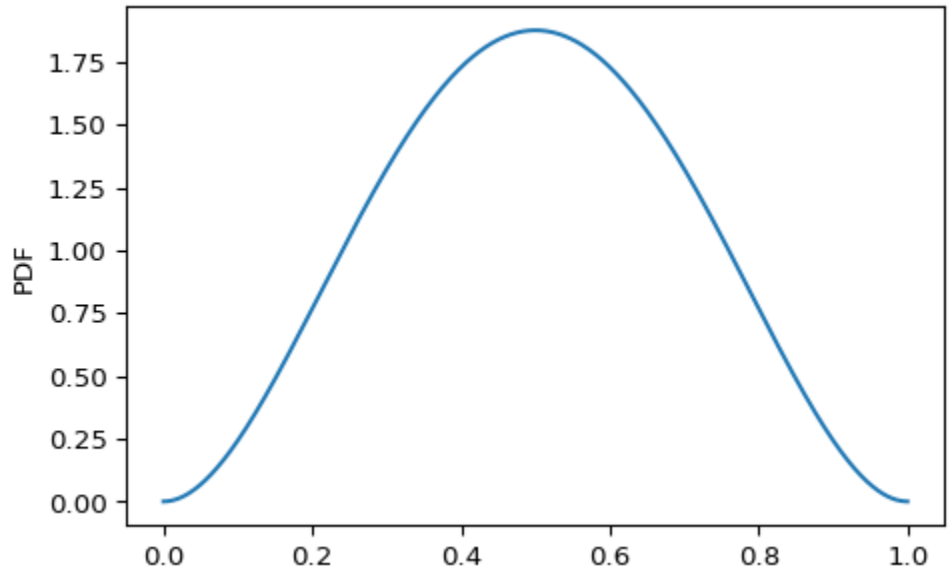

Figure 4: Beta distribution with alpha = 3 and beta = 3.

b)

```
mu = 0
sigma = 2
N = 10
data_rnorm = np.random.normal(mu, sigma, N)
print(data_rnorm)
```

```
[ 0.9165453   2.98895453  0.83684731  0.53096709 -0.00489004 -0.89299133
 -2.08628534 -0.24021313  0.25742229 -0.4612889 ]
```

```
alpha = 2
beta = 5
N = 10
data_rbeta1 = np.random.beta(alpha, beta, N)
print(data_rbeta1)
```

```
[0.0452532  0.10572811 0.03244559 0.19395301 0.50089772 0.06762628
 0.32950317 0.19751325 0.2070941  0.12360379]
```

```
alpha = 3
beta = 3
N = 10
```

```
data_rbeta2 = np.random.beta(alpha, beta, N)
print(data_rbeta2)
```

```
[0.29100309 0.82362741 0.60486317 0.57010691 0.27396461 0.33313635
 0.11113225 0.73473391 0.83265422 0.54613222]
```

B:2

a)

```
PDF_list = np.array([
    [-2, 0.1],
    [2, 0.1],
    [0, 1]
])
```

```
x_axis = np.linspace(-3, 3, 100)
# Plot PDFs
for params in PDF_list:
    pdf_norm = stats.norm.pdf(x_axis, params[0], params[1])
    plt.plot(x_axis, pdf_norm, label=f"N(mu = {params[0]}, sigma =
{params[1]})")
plt.ylabel("Density")
plt.legend()
plt.show()
```

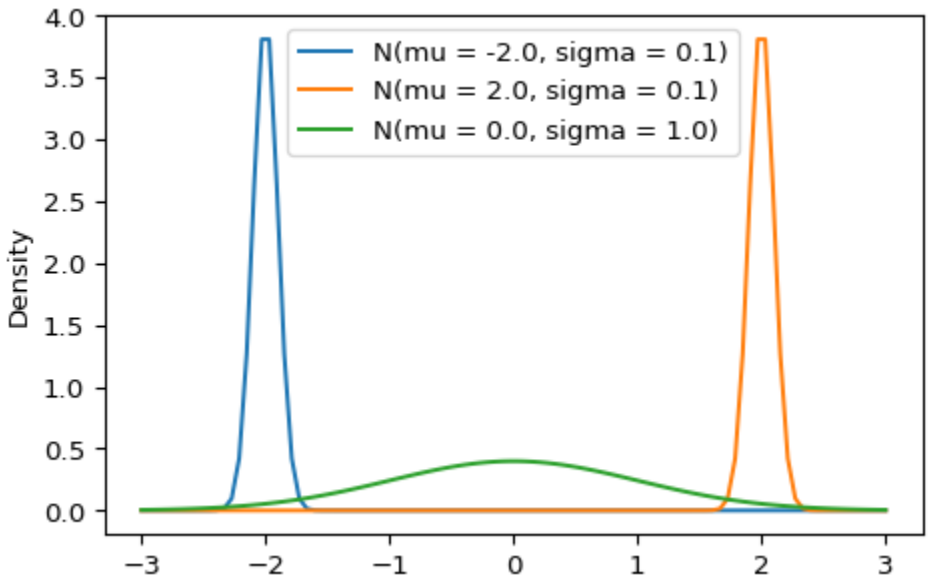

Figure 5: p(x)

b)

```
w1 = 0.05
w2 = w1
w3 = 0.9

pdf_index = np.asarray([0, 1, 2])
weights = np.asarray([int(w1*100), int(w2*100), int(w3*100)])
urn_B2 = np.repeat(pdf_index, weights)
```

```
N = 1000  # n samples total
data = []
for i in range(N):
    rPDF = np.random.choice(urn_B2, size=1, p=None, replace=True)
    mu, sigma = PDF_list[rPDF[0]]
    data.append(np.random.normal(mu, sigma, 1)[0])
```

```
plt.hist(data, bins = 100, edgecolor = "black", density=True)
plt.show()
```

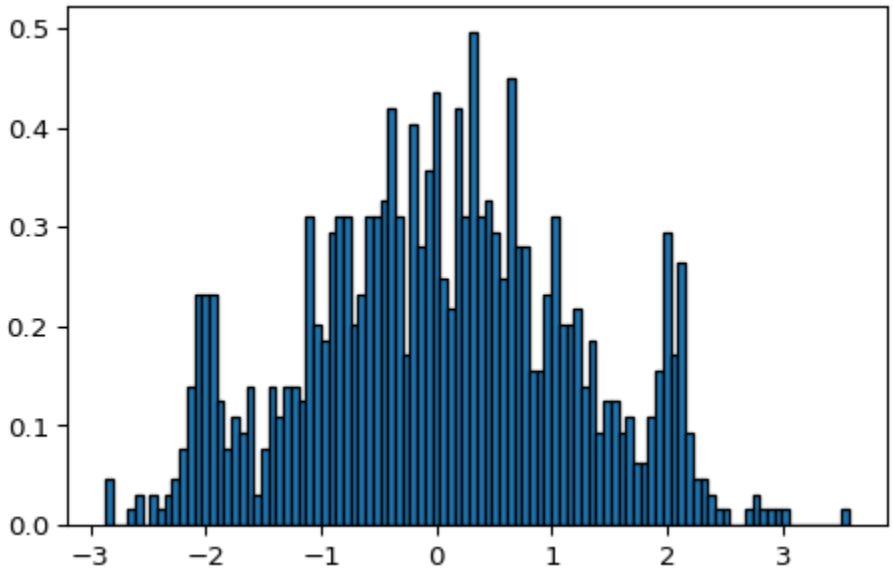

Figure 6: Combined PDFs histogram.

B:3

a)

```
N = 1000
mu_2d = np.array([0, 0])
```

```
sigma_2d = np.array([[1, 0.7], [0.7, 3]])

rnorm_2d = np.random.multivariate_normal(mu_2d, sigma_2d, N)
```

```
plt.plot(rnorm_2d[:, 0], rnorm_2d[:, 1], "bo", markersize=2)
```

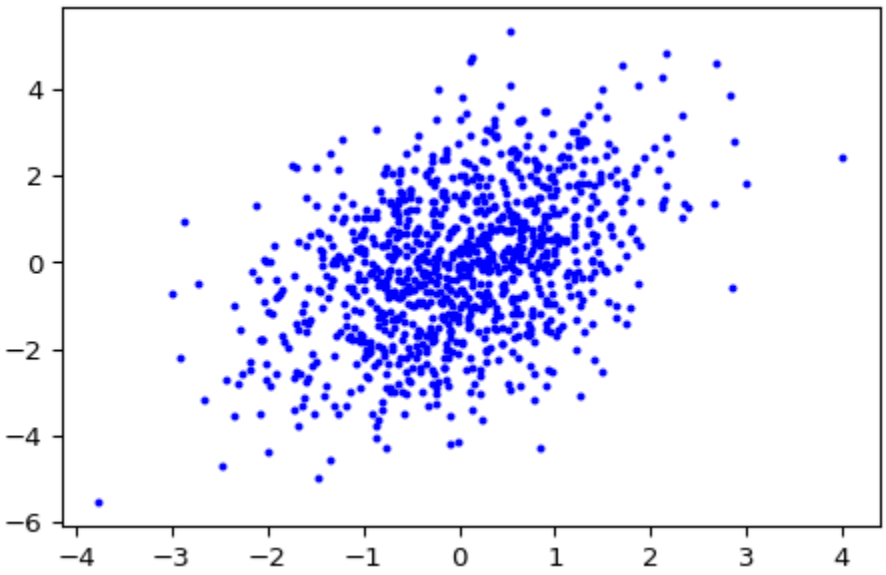

Figure 7: Multivariate normal distribution with mu = [0, 0] and covariance matrix = [[1, 0.7], [0.7, 3]].

b)

```
N = 1000
mu_2d = np.array([0, 0])
sigma_2d = np.array([[1, -0.7], [-0.7, 3]])

rnorm_2d = np.random.multivariate_normal(mu_2d, sigma_2d, N)
```

```
plt.plot(rnorm_2d[:, 0], rnorm_2d[:, 1], "bo", markersize=2)
```

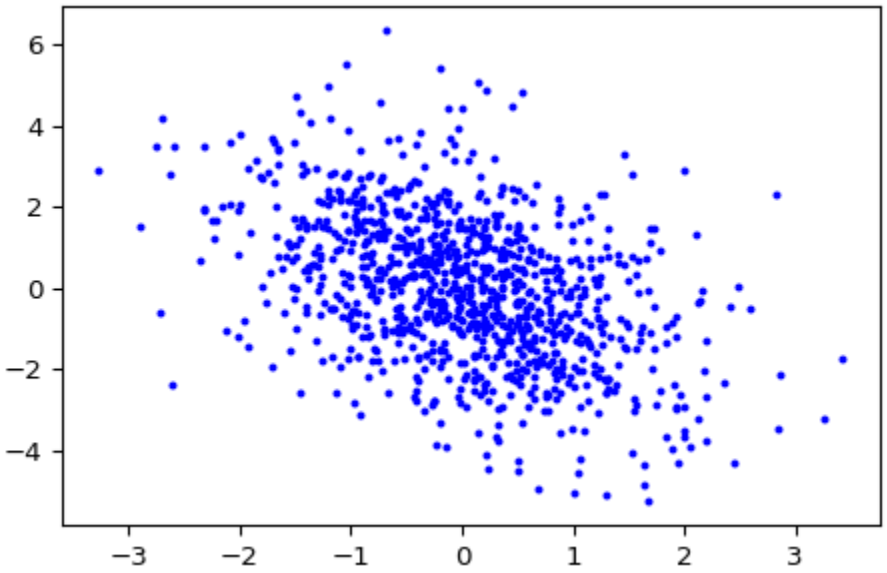

Figure 8: Multivariate normal distribution with mu = [0, 0] and covariance matrix = [[1, −0.7], [−0.7, 3]].

c)

```
N = 1000
mu_2d = np.array([0, 0])
sigma_2d = np.array([[1, 0], [0, 3]])

rnorm_2d = np.random.multivariate_normal(mu_2d, sigma_2d, N)
```

```
plt.plot(rnorm_2d[:, 0], rnorm_2d[:, 1], "bo", markersize=2)
```

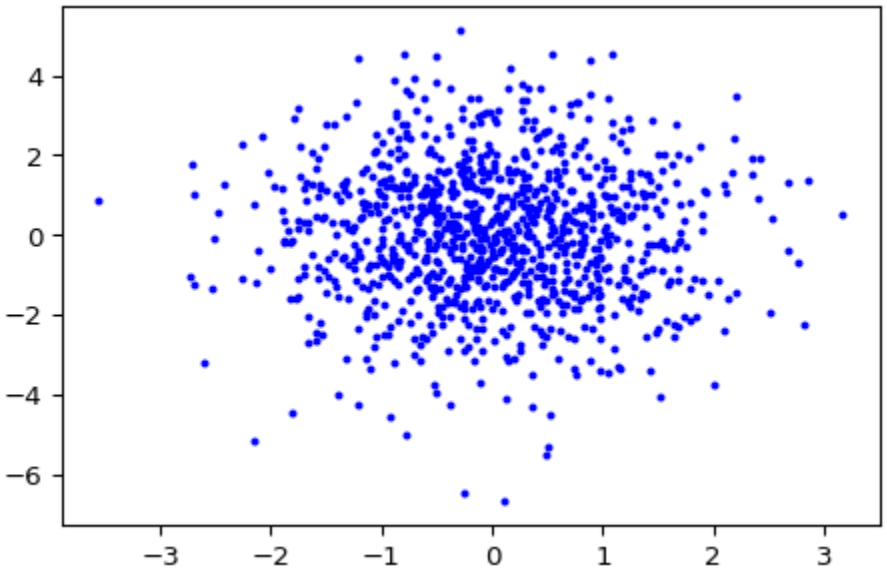

Figure 9: Multivariate normal distribution with mu = [0, 0] and covariance matrix = [[1, 0], [0, 3]].

Part C

C:1

a)

```
transition_matrix = np.array(
    [
        [0.8, 0.2, 0],  # 0 = G
        [0.8, 0, 0.2],  # 1 = M
        [0.1, 0, 0.9],  # 2 = A
    ]
)

def next_state(current, P):
    r = np.random.rand()  # uniform in [0, 1)
    cumulative = np.cumsum(P[current])

    # Find the first index where r <= cumulative probability
    return np.searchsorted(cumulative, r)
```

```
steps = 2000
state_current = 0
states = [state_current]
for i in range(steps):
    state_current = next_state(state_current, transition_matrix)
    states.append(state_current)
```

b)

```
plt.hist(states[1000:], density=True)
plt.xticks([0, 1, 2], ["G", "M", "A"])
plt.show()
```

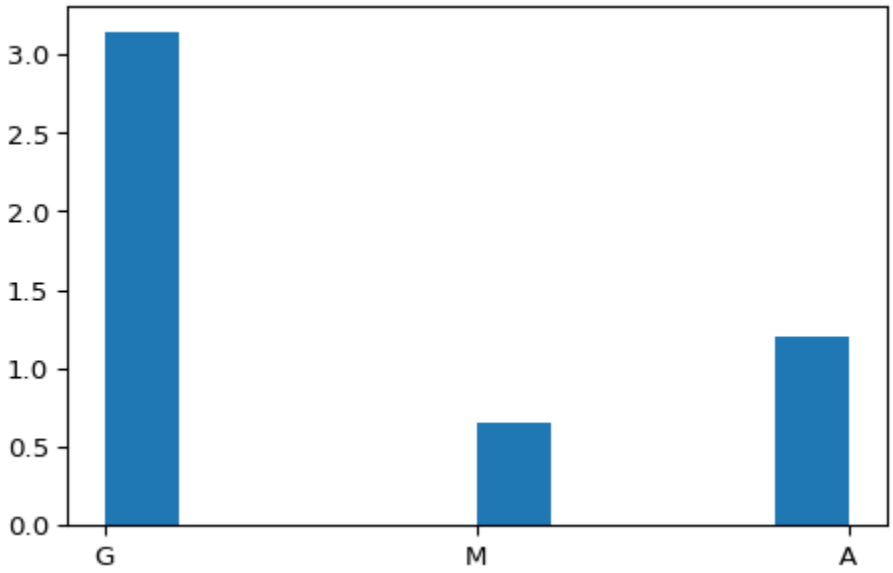

Figure 10: Cell cycle states.

C:2

```
transition_matrix = np.array(
    [
        [0, 1, 0, 0, 0],  # 0 = Start
        [0, 0.9, 0.1, 0, 0],  # 1 = E
        [0, 0, 0, 1, 0],  # 2 = 5
        [0, 0, 0, 0.9, 0.1],  # 3 = I
        [0, 0, 0, 0, 0],  # 4 = End
    ]
)

nucleotides = np.asarray(["A", "C", "G", "T"])

p_nucleotide_E = np.asarray([1, 1, 1, 1])
urn_E = np.repeat(nucleotides, p_nucleotide_E)

p_nucleotide_5 = np.asarray([5, 0, 95, 0])
urn_5 = np.repeat(nucleotides, p_nucleotide_5)

p_nucleotide_I = np.asarray([4, 1, 1, 4])
urn_I = np.repeat(nucleotides, p_nucleotide_I)
```

```
def next_state(current, P):
    r = np.random.rand()  # uniform in [0, 1)
    cumulative = np.cumsum(P[current])

    # Find the first index where r <= cumulative probability
    return np.searchsorted(cumulative, r)
```

```
steps = 20
state_current = 0
states = [state_current]
bases = []
for i in range(steps):
    state_current = next_state(state_current, transition_matrix)
    states.append(state_current)

    if state_current == 1:  # E
        bases.append(np.random.choice(urn_E, size=1, p=None, replace=True))
    elif state_current == 2:  # 5
        bases.append(np.random.choice(urn_5, size=1, p=None, replace=True))
    elif state_current == 3:  # I
        bases.append(np.random.choice(urn_I, size=1, p=None, replace=True))

print(np.array(states).flatten())
print(np.array(bases).flatten())
```

```
[0 1 1 1 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
['T' 'C' 'A' 'G' 'C' 'T' 'A' 'T' 'T' 'A' 'A' 'C' 'T' 'G' 'T' 'A' 'T' 'A'
 'A' 'C']
```

Part D

D:1

a)

```
N = 10
x = np.random.uniform(0, 1, N)
y = [2*n+4 for n in x]
```

b)

```
N = 10
x = np.random.uniform(0, 1, N)
```

```
y = [2 * n + 4 for n in x]

x_obs = []
y_obs = []
for i in range(N):
    rnorm = np.random.normal(0, 0.3, 2)
    x_obs.append(x + rnorm[0])
    y_obs.append(y + rnorm[1])
```