Technical University of Denmark

02443 - Stochastic Simulation

Project 1: Simulation and Estimation in a Markov Model of Breast Cancer

Gunnhildur Katrín Erlendsdóttir - s212510

Helga Dís Halldórsdóttir - s212457

David Miles-Skov - s204755

Patrick Jensen Martins - s204748

Professor:

Bo Friis Nielsen

Contents

Table 1: Contributions

Part 1	David Miles-Skov & Patrick Jensen Martins
Part 2	Gunnhildur K. Erlendsdóttir & Helga Dís Halldórsdóttir
Part 3	Everyone

1 Introduction

In this project a a Markov model is simulated and analyzed for breast cancer, describing possible complications following a surgery for breast cancer.

We consider a sequence of random variables X_t, where $t = 0, 1, \ldots$ represents time and X_t can take values from 1 to N. These variables exhibit the Markov property, implying that the future values are conditionally independent of the past values given the present value. This property allows us to determine the next value X_{t+1} solely based on the current value X_t. The transition probabilities p_{ij} represent the likelihood of transitioning from state i to state j. These probabilities are organized in an $N \times N$ probability matrix P. The validity of the probabilities is ensured by the condition $\sum_{j=1}^{N} p_{ij} = 1$ for all $i = 1, 2, \ldots, N$.

2 Part 1: A Discrete-Time Model

This project utilizes a Markov model to track the potential health complications in women following the removal of their breast tumors. The model defines states in terms of the "health" of the patient, as it relates to potential redevelopment of breast cancer after surgery. For example, state 1 corresponds to a healthy patient, where no tumors have been found, state 2 corresponds to the local recurrence of cancer (i.e., new tumors have been found near the site of the original tumor) and so on. The model accounts for the possibility of distant metastasis, where the cancer reappears in another location. Both local recurrence and metastasis at varying distances can occur simultaneously. The occurrence of death is possible from any state, and results in the simulation terminating.

2.1 Task 1

A Markov Chain Monte Carlo method was used to generate a sequence of states for 1000 women. The sequence initial began in state 1, making random transitions to other states based on the following probability matrix that was given:

$$P = \begin{bmatrix} 0.9915 & 0.005 & 0.0025 & 0 & 0.001 \\ 0 & 0.986 & 0.005 & 0.004 & 0.005 \\ 0 & 0 & 0.992 & 0.003 & 0.005 \\ 0 & 0 & 0 & 0.991 & 0.009 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Each transition corresponds to a period of one month allowing for a simple calculation of lifetime after surgery/original tumor removal.

After simulating the sequence of states for 1000 women, and calculating their respective lifetimes, the overall distribution of lifetimes was found in figure 1.

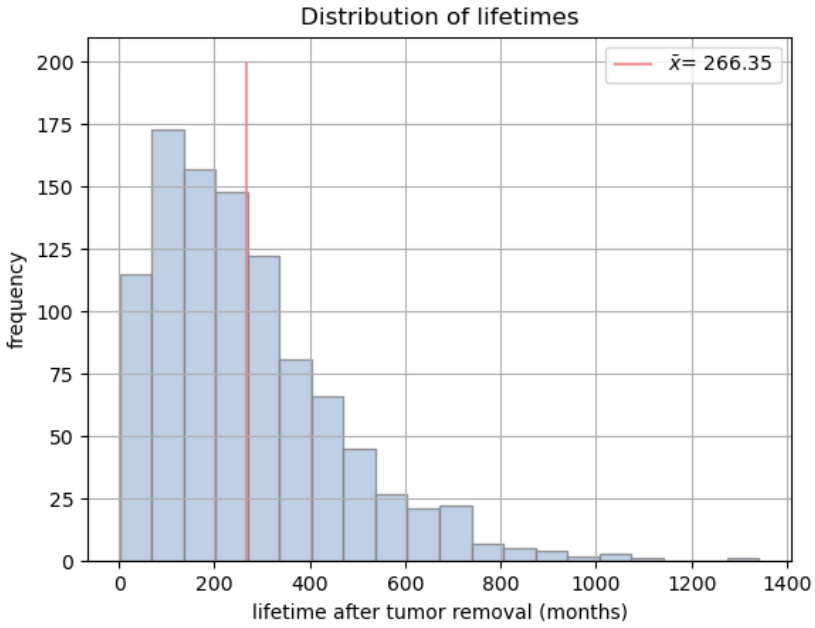

Figure 1: Histogram showing the distribution of lifetime in months after breast cancer surgery

The mean lifetime was also found, with a value of $\bar{x} = 266.35$ *months* $= 22.20$ *years* , as shown in figure 1. The proportion of women that experience local recurrence of cancer is estimated to be 59.5%.

2.2 Task 2

In order to find the probability distribution over the states after 120 months ($\mathbf{p}_{120}$) produced by our simulations, the sequence of states for 10^4 women was simulated via Markov Chain Monte Carlo (just as in task 1). The probability distribution over states was calculated for each sequence. Finally, the mean distribution was found.

This simulated probability distribution was compared with that of the formula:

$$\mathbf{p}_t = \mathbf{p}_0(\mathbf{P}^t) \tag{1}$$

Where $\mathbf{P}$ is given as (refer) and $\mathbf{p}_0$ is determined to be $[1, 0, 0, 0, 0]$, as the initial state is always 1. The results are represented below.

Table 2: Comparison between simulated and analytical state probability distributions at $t = 120$

State (i)	1	2	3	4	5
Simulated $P(X = i)$	0.622	0.129	0.106	0.0306	0.113
Analytical $P(X = i)$	0.359	0.159	0.166	0.0677	0.248

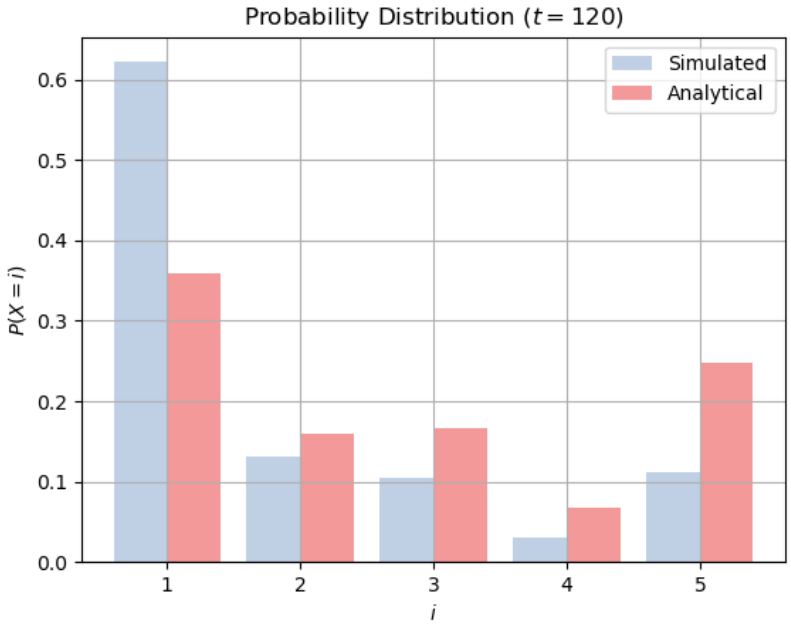

Figure 2: Comparison between simulated and analytical state probability distributions at $t = 120$

In order to evaluate the similarity of distributions, a χ^2 test was conducted, yielding the following results:

Table 3: Results of χ^2 test

Statistical Quantity	**Value**
χ^2 test statistic	0.328
P-value	0.988

A high p-value obtained from the χ^2 test suggests that there is no significant difference between the two distributions. Therefore, we can conclude that the probability distribution of states of our simulated sample and the analytical probability distribution are similar.

2.3 Task 3

The theoretical *discrete phase-type* distribution of lifetimes

$$P(T = t) = \pi(\mathbf{P_s}^t)\mathbf{p_s}$$

and mean lifetime

$$E(T) = \pi(\mathbf{I} - \mathbf{P_s})^{-1} \cdot \mathbf{1}$$

were calculated for $t \in 1, ...1200$. After sampling 2000 lifetimes and plotting both the theoretical distribution and simulated distribution, it is clear that our simulated lifetimes adhere to a distribution very similar to that described by the theoretical *phase-type* distribution.

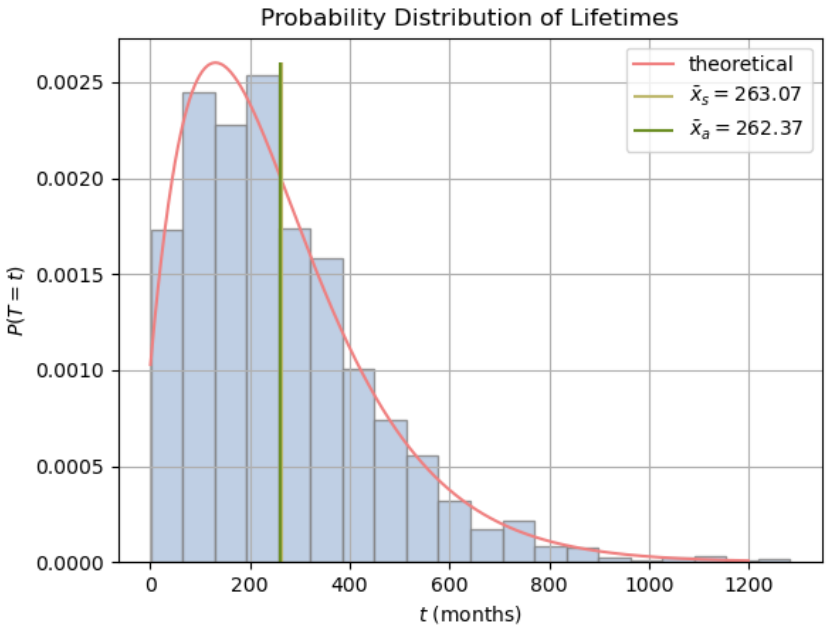

Figure 3: Comparing the theoretical and simulated distributions of states and means

2.4 Task 4

The expected lifetime, after surgery, for a woman who survives the first 12 months following surgery, but also experiences some form of cancer recurrence in the first 12 months is: 176.83 months

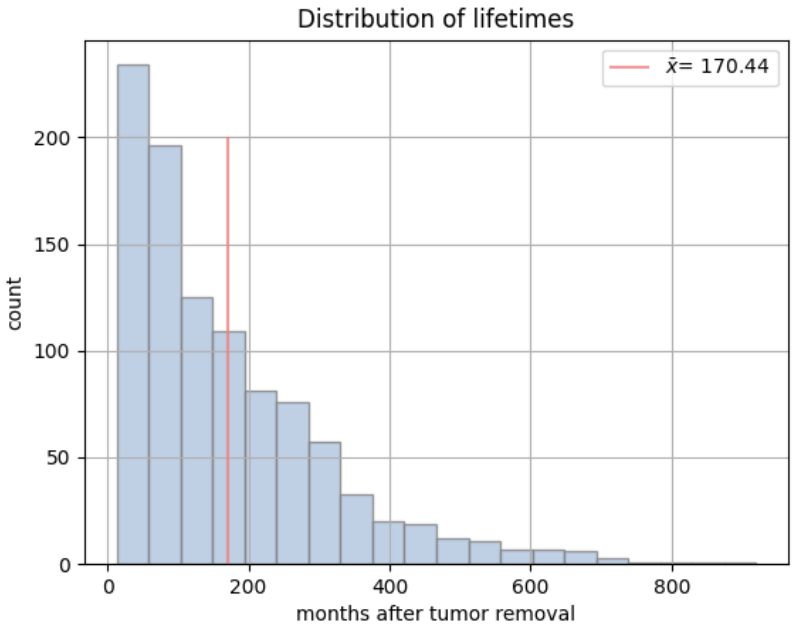

Figure 4: Mean lifetimes within the first 12 months after removal, with local recurrence.

A simulation of 10 samples was performed to investigate the variance of the estimate of the mean lifetime.

The estimate of the mean was calculated to be equal to 173.28 with a variance of 11.68, which is a relatively high variance.

2.5 Task 5

The fraction of women who live within the first 350 months after surgery was computed for a simulated sample of 200 women, 100 times. The fractions were plotted in histogram 5 below.

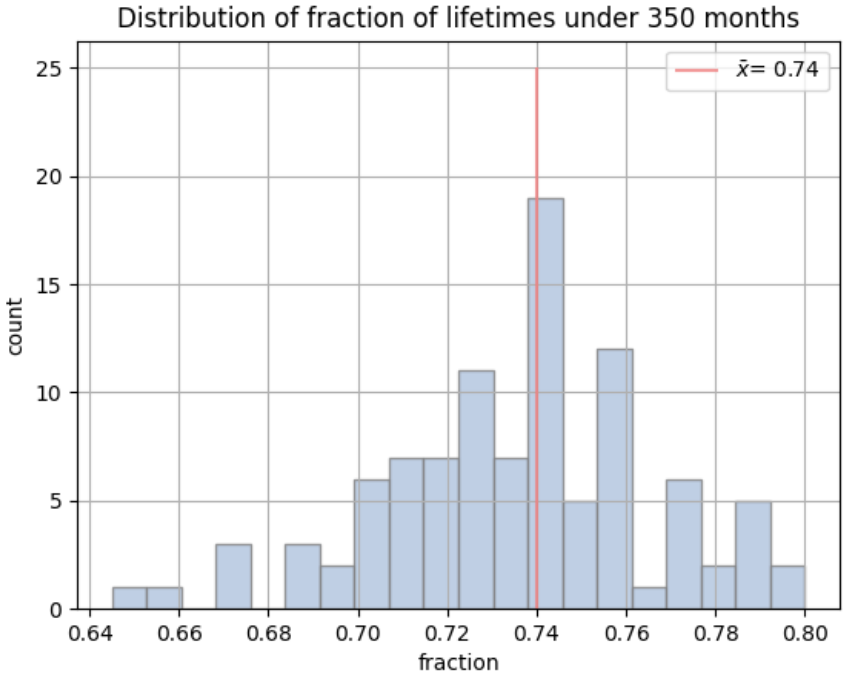

Figure 5: Distribution of proportion of women who died within the first 350 months following surgery

In order to reduce the variance of the estimate of the fraction, the control variate method was used. The mean lifetime after surgery was used as the control variate, such that: $Z = X + c(Y - \mu_Y)$, where Y is the sample with the mean lifetimes and X is the sample with the fractions of women who live within the first 350 months after surgery.

Table 4: Estimates and variances using crude MC and control variates methods

	Estimate	Variance
Crude MC	$7.4025 \cdot 10^{-1}$	$7.86 \cdot 10^{-4}$
Control Variates	$7.4025 \cdot 10^{-1}$	$3.46 \cdot 10^{-4}$

The variance was reduced by approximately 40%.

2.6 Task 6

The discrete time Markov chain model assumes that there is a time space of exactly one month between each state transition. This means that for any randomly generated observation, the interval between the time at which a woman has undergone surgery and the time at which she passes away is always an integer number. These assumptions are clearly not realistic, since in reality events don't always occur in regular intervals and are quite unpredictable in terms of the timing at which they occur. In order to relax those assumptions, one could, for each state reached, compute an integer between some range that would determine the number of months a woman remains in the current state, and then set the probability of remaining at the same state as 0 and randomly transition to the next state according to the transition probabilities of the current state.

3 Part 2: A continuous-time model

In the previous part, we assumed transitions from one state to another only happened once a month. In this part, we make the assumption that transitions between states can happen at any time, known as Continuous-Time Markov Chains (CTMC). A CTMC is specified by a transition-rate matrix Q, which indicates the rate at which you can transition from state i to state j, given that you are in state i. Additionally, an important characteristic of Continuous-Time Markov Chains is the sojourn time, which refers to the duration that the system remains in a specific state (denoted as X(t)), follows an exponential distribution with a rate of $-q_{ii}$.

3.1 Task 7

The following transition-rate matrix was given:

$$Q = \begin{bmatrix} -0.0085 & 0.005 & 0.0025 & 0 & 0.001 \\ 0 & -0.014 & 0.005 & 0.004 & 0.005 \\ 0 & 0 & -0.008 & 0.003 & 0.005 \\ 0 & 0 & 0 & -0.009 & 0.009 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Simulation of 1000 women was done, that all start at state 1, until state 5. The results are displayed in a histogram below. It shows the lifetime distribution of the women after surgery.

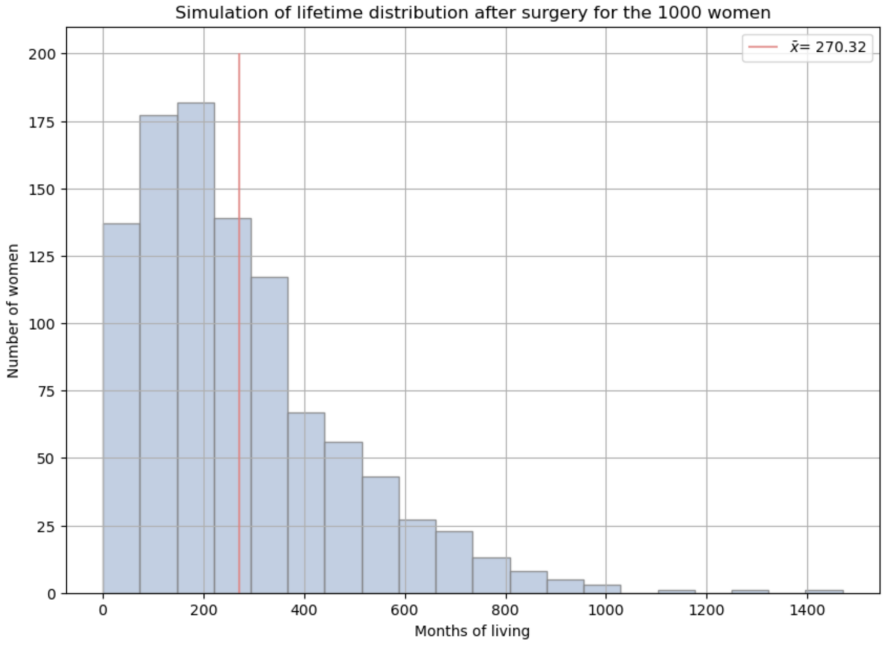

Figure 6: Lifetime distribution of 1000 simulation

The lifetime mean and the lifetime standard deviation with the corresponding confidence interval is shown in table 5 below.

Table 5: Lifetime mean and lifetime standard deviation and the corresponding confidence intervals

Statistical Quantity	**Value**	**95% Confidence Interval**
Mean	22.526	[21.479, 23.574]
Standard Deviation	5.963	[5.682, 6.243]

When looking at the lifetime distribution, it is clear that a few women live longer than 11000 months or approximately 92 years after surgery. This is not very realistic as women of different ages could be having the surgery and women do usually not have breast cancer before the age of 40. The proportion of women where the cancer reappeared distantly after 30.5 months was calculated to be 39.3%. This was calculated by identifying the number of women who were in a state higher than 2 and had zero duration in states 3 and 4. This was then divided by the total number of women simulated.

3.2 Task 8

The lifetime distribution follows a continuous time phase-type distribution:

$$F_T(t) = 1 - \mathbf{p_o} exp(\mathbf{Q_s} t)\mathbf{1}$$

where Q_s is a sub-matrix of Q, where last row and column are removed:

$$Q_s = \begin{bmatrix} -0.0085 & 0.005 & 0.0025 & 0 \\ 0 & -0.014 & 0.005 & 0.004 \\ 0 & 0 & -0.008 & 0.003 \\ 0 & 0 & 0 & -0.009 \end{bmatrix}$$

The empirical lifetime distribution done in task 7 is now compared with the theoretical lifetime distribution. Figure 7 shows both distributions on the same graph, and it is clear that both of them are very similar.

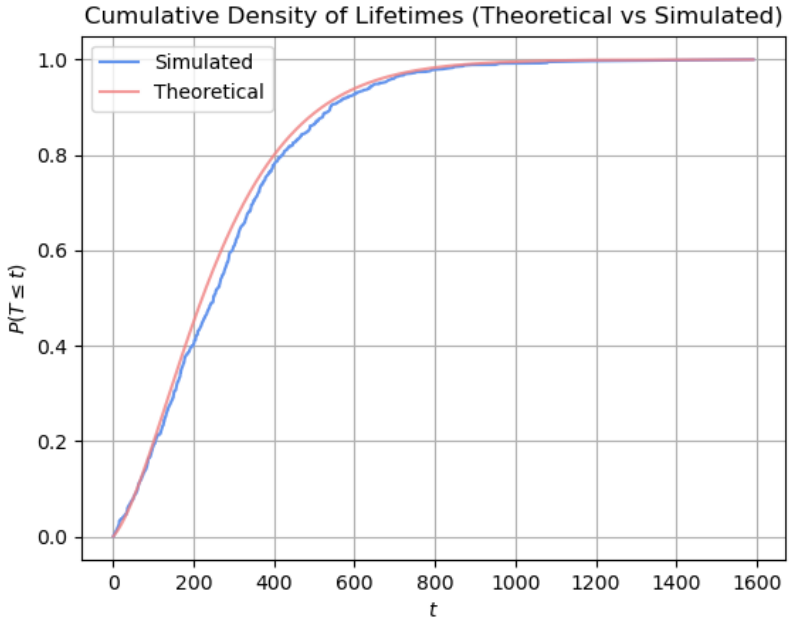

Figure 7: Comparing the empirical lifetime distribution to the theoretical lifetime distribution

To determine whether the empirical distribution comes from the same distribution as the theoretical one, the Kolmogorov-Smirnov test was also carried out. The results from that are shown in table

6. The results show a high p-value so there is not enough evidence to reject the null hypothesis which indicate that there is no significant difference between the two distributions. Therefore, it is concluded that the empirical distribution comes from the same distribution as the theoretical one.

Table 6: Results from the Kolmogorov-Smirnov test

D	**p-value**
0.0235	0.8779

3.3 Task 9

Now the effect of a certain treatment is considered and two treatments are compared visually. To do that the survival functions, $S(t)$, are examined, which is defined as the proportion of women alive at time t:

$$S(t) = P(T > t)$$

An unbiased estimator of the survival functions, $\hat{S}(t)$, is the Kaplan-Meier estimator, where N is the total number of women and $d(t)$ is the number of women who have died at time t:

$$\hat{S}(t) = \frac{N - d(t)}{N}$$

The transition-rate matrix is now as follows:

$$Q_t = \begin{bmatrix} * & 0.0025 & 0.00125 & 0 & 0.001 \\ 0 & * & 0 & 0.002 & 0.005 \\ 0 & 0 & * & 0.003 & 0.005 \\ 0 & 0 & 0 & * & 0.009 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

1000 women who have received the treatment were simulated. Both the lifetime distributions and the Kaplan-Meier estimate of the survival function for both the 1000 women that received the treatment and the 1000 women that did not received the treatment were plotted. The results are shown on figures 8 and 9 below. It is clear from the figure that the women who received a treatment are living longer then the women who did not receive a treatment. This is further confirmed by the difference is means.

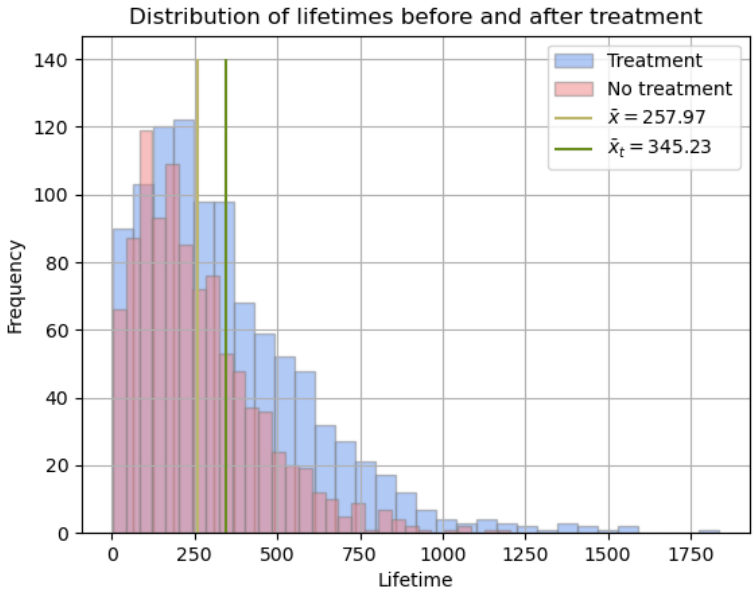

Figure 8: Comparison of lifetime distributions before and after treatment

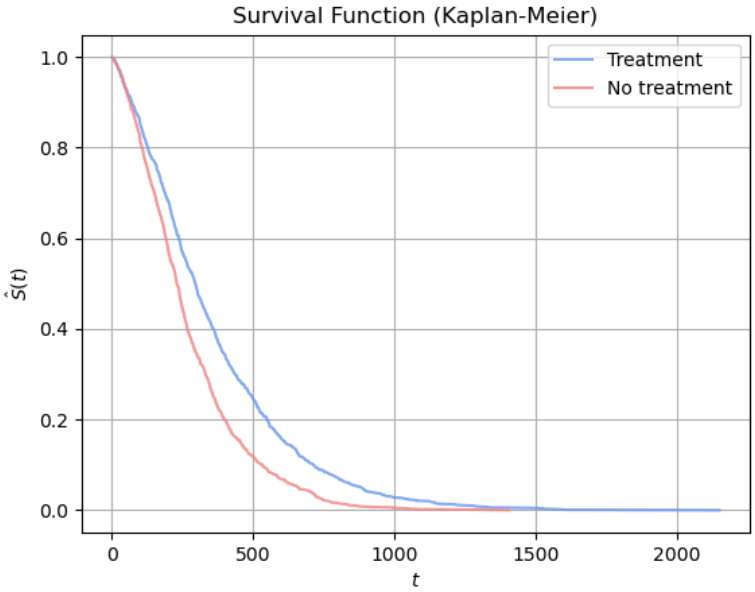

Figure 9: Kaplan-Meier estimation of the survival function for women who receive a treatment and women who do not receive a treatment

3.4 Task 11

In transitioning from a discrete-time model to a continuous-time model, the assumption that women move from one state to another only once a month is eliminated. Instead, the continuous-time model allows for transitions to occur at any time, providing a more realistic representation of upstaging in breast cancer.

Additionally, the model assumes that the time a woman remains in a particular state follows an exponential distribution. However, to further enhance the model, the exponential distribution assumption can be replaced with an Erlang distribution. The Erlang distribution represents the sum of k independent and identically distributed random variables, each following an exponential distribution.

By introducing a k value of, for example, 1000, the model can be extended to have Erlang-distributed sojourn times. This allows for a broader range of duration distributions in each state, capturing a more diverse set of potential upstaging scenarios.

4 Part 3: Estimation

In the following part of the project, the objective is to estimate the unknown parameter Q. In practical situations, the transitional stages of patients are not directly known, they are observed periodically during screenings conducted every few years. For this part we assume the state is observed every 4'th year (48 months).

4.1 Task 12

Here the same Q as in part 2 of the project is used. 1000 women were simulated, starting at state 1, until state 5. A vector was created that observes the state each women is at every 48 months:

$$Y^{(i)} = (X(0), X(48), X(96), ...,)$$

First, the maximum possible lifetime for the simulated values was determined, as well as a vector representing each 48th month, which served as the observation points. At each observation point (48 months), the cumulative sojourn times from the previous simulation were examined. If the sum of the sojourn times for the states was greater than or equal to 48 months, the corresponding state for each woman was recorded. By following this approach, an approximate indication of the state of

each woman at every 48th month was obtained, resulting in a vector or time series that represented their transition through the different states. Figure 10 shows the time series that were generated of 1000 women from state 1 to state 5 (showed as state 0-4 on the figure). From the figure, is is observed that all the women start in state 1 and after around 700 months, most of the women have reached state 5.

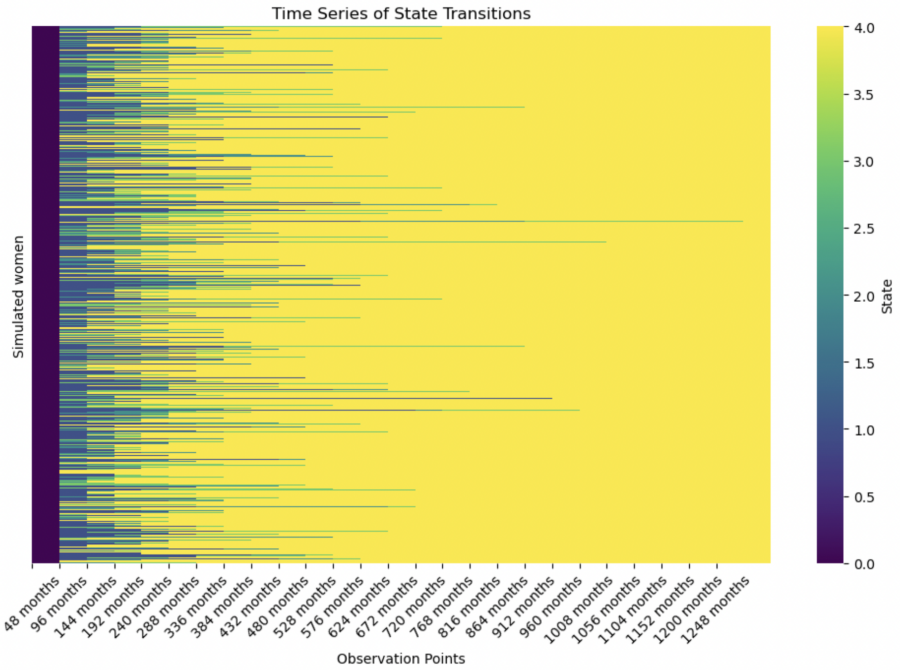

Figure 10: Time series of state transitions for all 1000 simulated women every 48 months

4.2 Task 13

Now the assumption is made that those 1000 time series is all that is observed. The Monte Carlo Expectation Maximization algorithm

We generate a random sample from the true matrix Q in order to obtain our "real" observations.

For each time series in the observed sample, we generate another random sequence from a proposed Q_0. Afterwards, we check whether the final state of the generated sequence is equal to the target state of the time series interval from the observed sample. If it is, then we accept the sequence and move on to the next time series interval. Otherwise we reject the simulated sample and repeat. This process is repeated for every woman observed.

After that, we update the proposed matrix Q_0 based on the following equation, by traversing through

all the collected trajectories:

$$q_{i,j} = \frac{N_{i,j}}{S_i} \tag{2}$$

The diagonal elements of Q can be computed by the following expression:

$$q_{ii} = -(q_{i,1} + ... + q_{i,(i-1)} + q_{i,(i+1)} + ...q_{i,N}), \text{ for } i \in 1, ..., N \tag{3}$$

This step is repeated every time a new set of trajectories is collected until the infinite norm of the difference between Q^k and Q^{k+1} is less than 10^{-3}.

After having implemented the algorithm, we plotted the error evolution after k iterations:

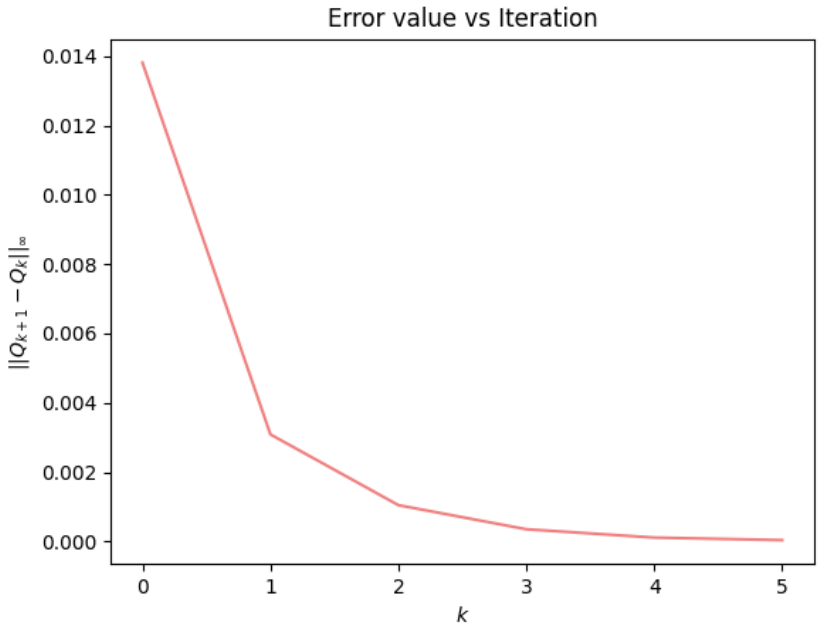

Figure 11: Error evolution after k iterations

As we can observe from the plot, the the infinite norm of the difference between Q^k and Q^{k+1} (error) gets reduced until falling bellow the value of 10^{-3}.

4.2.1 Pseudocode

Initialise Q_0, obtain Y

$Q_k \leftarrow Q_0$

Initialise N_k as a 5×5 matrix of zeros

Initialise S_k as a 1×5 array of zeros

$error \leftarrow \infty$

$\epsilon \leftarrow 10^{-3}$

while $error > \epsilon$ **do**

$i \leftarrow 0$

while $i \leq 1000$ **do**

`observed` $\leftarrow Y_i$

Generate satisfactory time series according to `observed` - call `generated`

Update total number of jumps N_k and total *sojourn times* S_k using `generated`

end while

Create updated Q_{k+1} using N_k and S_k using equations 2 and 3.

$error \leftarrow ||Q_k - Q_{k+1}||_\infty$

end while

5 Appendix

Part 1: A discrete-time model

Task 1

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

```
def MCMC():
    P = np.array([[0.9915, 0.005, 0.0025, 0, 0.001],
                  [0, 0.986, 0.005, 0.004, 0.005],
                  [0, 0, 0.992, 0.003, 0.005],
                  [0, 0, 0, 0.991, 0.009],
                  [0, 0, 0, 0, 1]])
    N = len(P[0])
    # initial state = 0
    states = [0]
    local_reappearance = False # Boolean recording if cancer
    reappears locally

    while states[-1] != N-1: # Death occurs in final column
        currrent_state = states[-1] # Current i state (as
   defined by previous transition)

        # performing transition
        transition_probabilities = P[currrent_state]
        new_state = np.random.choice(list(range(N)), p =
   transition_probabilities) # New column

        # Updating
        if new_state == 1: # If transitioning into state 2,
   local reappearance has occured
            local_reappearance = True
```

```
        states.append(new_state)
    lifetime = len(states)-1

    return lifetime, local_reappearance
```

```
def plot_distribution_lifetimes(lifetimes):
    mean = round(np.mean(lifetimes), 2)
    plt.hist(lifetimes, 20, color="lightsteelblue", alpha
  =0.8, edgecolor='gray')
    plt.vlines(mean,ymin=0, ymax=200, color="lightcoral",
  alpha=0.8, label=r"$\bar{x}$"+f"= {mean}")
    plt.grid()
    plt.legend()
    plt.ylabel("frequency")
    plt.xlabel("lifetime after tumor removal (months)")
    plt.title("Distribution of lifetimes")
    plt.show()
```

```
def gen_analyse_samples(n):
    lifetimes, local_reappearances = [], 0
    # Random sampling for n times

    for i in range(n):
        if i%100==0:
            print(f"simulated {i} samples")
        # Perform MCMC
        lifetime, local_reappearance = MCMC()
        lifetimes.append(lifetime)
        if local_reappearance: local_reappearances += 1

    print("Finished generating samples")
    plot_distribution_lifetimes(lifetimes)
    print(f"Mean lifetime after tumor removal: {round(np.
  mean(lifetimes), 2)}")
```

```
    print(f"Proportion of women experiencing local
   reappearance of cancer: {local_reappearances/1000}")
```

Task 2

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from collections import Counter
```

```
def make_dist(p):
    counts = [p.count(i)/len(p) for i in range(5)]
    return counts
```

```
def doChi2(observed, expected):
    return stats.chisquare(f_obs=observed, f_exp=expected)
```

```
def get_state_vector(P, t):
    # Making use of Markov Property
    P_t = np.linalg.matrix_power(P, t)
    p0 = np.array([1,0,0,0,0])
    return np.dot(p0, P_t)
```

```
# Markov Chain Monte Carlo
def MCMC():
    P = np.array([[0.9915, 0.005, 0.0025, 0, 0.001],
                  [0, 0.986, 0.005, 0.004, 0.005],
                  [0, 0, 0.992, 0.003, 0.005],
                  [0, 0, 0, 0.991, 0.009],
                  [0, 0, 0, 0, 1]])
    N = len(P[0])
    # initial state
    states = [0]

    for _ in range(120): # t = 0 -> 120
```

```
        currrent_state = states[-1] # Current state (as
    defined by previous transition)
        # performing transition
        transition_probabilities = P[currrent_state]
        new_state = np.random.choice(list(range(N)), p =
    transition_probabilities) # New column
        # Updating
        states.append(new_state)
    return states
```

```
def get_samples(n):
    observed = np.zeros(5)
    t = 120
    P = np.array([[0.9915, 0.005, 0.0025, 0, 0.001],
                  [0, 0.986, 0.005, 0.004, 0.005],
                  [0, 0, 0.992, 0.003, 0.005],
                  [0, 0, 0, 0.991, 0.009],
                  [0, 0, 0, 0, 1]])
    expected = get_state_vector(P, t)

    for i in range(n):
        if i%100==0:
            print(f"simulated {i} samples")
        sequence_of_states = MCMC()
        state_vector = np.array(make_dist(sequence_of_states
    ))
        observed += state_vector

    observed = observed/n # Mean state vector
    print(f"Observed: {observed}")
    print(f"Expected: {expected}")

    x = np.array(list(range(1, len(observed)+1)))
```

```
    plt.bar(x-0.2, observed, color="lightsteelblue", label="
   Simulated",alpha=0.8, width=0.4, align='center')
    plt.bar(x+0.2, expected,  color="lightcoral", label="
   Analytical", alpha=0.8, width=0.4, align='center')
    plt.xticks(x)
    plt.xlabel(r"$i$")
    plt.ylabel(r"$P(X = i)$")
    plt.title("Probability Distribution "+r"$(t=120)$")
    plt.legend()
    plt.grid()
    plt.show()

    t, p = doChi2(observed, expected)
    print("chi-2 test:")
    print(f"Test statistic: {t}")
    print(f"P-value: {p}")
```

Task 3

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from task1 import MCMC
```

```
def gen_ecdf(numbers):
    y = np.cumsum(numbers)/sum(numbers)
    return y
```

```
def calc_P_T_t(t):
    P = np.array([[0.9915, 0.005, 0.0025, 0, 0.001],
                  [0, 0.986, 0.005, 0.004, 0.005],
                  [0, 0, 0.992, 0.003, 0.005],
                  [0, 0, 0, 0.991, 0.009],
                  [0, 0, 0, 0, 1]])
```

```
    pi = np.array([1,0,0,0])
    P_s = P[:4, :4]
    p_s = P[:4, -1]
    return np.dot(pi, np.dot(np.linalg.matrix_power(P_s, t),
    p_s))
```

```
def calc_mean():
    P = np.array([[0.9915, 0.005, 0.0025, 0, 0.001],
                  [0, 0.986, 0.005, 0.004, 0.005],
                  [0, 0, 0.992, 0.003, 0.005],
                  [0, 0, 0, 0.991, 0.009],
                  [0, 0, 0, 0, 1]])

    pi = np.array([1,0,0,0])
    P_s = P[:4, :4]
    a = np.linalg.inv(np.identity(4)-P_s)
    E_T = sum(np.dot(pi, a))
    return E_T
```

```
def compare_simulated_analytic():
    lifetimes_sim = []
    # Random sampling 5000 times
    for i in range(2000):
        if i%500==0:
            print(f"simulated {i} samples")

        # Perform MCMC
        lifetime, _ = MCMC()
        lifetimes_sim.append(lifetime)

    # Generating probability distribution:
    mean_sim = np.mean(lifetimes_sim)
    mean_a = calc_mean()
```

```
    x = np.arange(1, 1200)
    lifetimes_a = [calc_P_T_t(t) for t in x]

    plt.hist(lifetimes_sim, 20, color="lightsteelblue",
   alpha=0.8, edgecolor='gray',density=True)
    plt.plot(x, lifetimes_a, color="lightcoral", label="
   theoretical")
    plt.vlines(mean_sim, ymin=0, ymax=0.0026, color="
   darkkhaki", label=r"$\bar{x}_s=$"+f"{round(mean_sim, 2)}"
   )
    plt.vlines(mean_a, ymin=0, ymax=0.0026, color="olivedrab
   ", label=r"$\bar{x}_a=$"+f"{round(mean_a, 2)}")
    plt.grid()
    plt.legend()
    plt.ylabel(r"$P(T=t$)")
    plt.xlabel(r"$t$"+" (months)")
    plt.title("Probability Distribution of Lifetimes")
    plt.show()
```

Task 4

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

```
# Markov Chain Monte Carlo
def MCMC_task4():
    P = np.array([[0.9915, 0.005, 0.0025, 0, 0.001],
                  [0, 0.986, 0.005, 0.004, 0.005],
                  [0, 0, 0.992, 0.003, 0.005],
                  [0, 0, 0, 0.991, 0.009],
                  [0, 0, 0, 0, 1]])

    N = len(P[0])
```

```
    # initial state = 0
    current_state = 0
    lifetime = 0

    cancer_reappearance = False # Boolean recording if
  cancer reappears locally
    survived_12_months = False

    while current_state != N-1: # Death occurs in final
  column
        # performing transition
        transition_probabilities = P[current_state]
        new_state = np.random.choice(list(range(N)), p =
  transition_probabilities) # New column
        lifetime += 1
        # Updating
        if new_state in [1,2,3] and lifetime <= 12:
            cancer_reappearance = True
        if lifetime > 12:
            survived_12_months = True
        current_state = new_state

    return lifetime, cancer_reappearance, survived_12_months
```

```
def get_lifetimes():
    lifetimes = []
    while len(lifetimes) < 1000:
        lifetime, cancer_reappearance, survived_12_months =
  MCMC_task4()
        if cancer_reappearance and survived_12_months:
            lifetimes.append(lifetime)
    return lifetimes
```

```
def examine_dist():
```

```
    lifetimes = get_lifetimes()
    print(np.mean(lifetimes))
```

Task 5

```
import numpy as np
import matplotlib.pyplot as plt
from task1 import MCMC
```

```
def crude_monte_carlo(n=200):
    lifetimes_below_350 = 0
    for _ in range(n):
        lifetime, _ = MCMC()
        if lifetime < 350:
            lifetimes_below_350 += 1
    return lifetimes_below_350/n
```

```
def get_prop(n=100):
    mean_prop = 0
    for _ in range(n):
        mean_prop += crude_monte_carlo()
    return mean_prop/n
```

Part 2: A continuous-time model

Task 7

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy import stats, linalg
import math
```

```
Q = np.matrix('-0.0085 0.005 0.0025 0 0.001; \
               0 -0.014 0.005 0.004 0.005; \
               0 0 -0.008 0.003 0.005; \
```

```
                0 0 0 -0.009 0.009; \
                0 0 0 0 0')
```

```
# calculate confidence interval
def confidence_interval(d):
    confidence = 0.95
    data = 1.0 * np.array(d)
    length = len(data)
    mean = np.mean(data)
    st_error = stats.sem(data)
    h = st_error * stats.t.ppf((1 + confidence) / 2., length
   -1)
    return mean-h, mean+h
```

```
# simulate
n_of_women = 1000
n = Q.shape[0]
liftime_dist = np.zeros((n_of_women, n-1))
reapperead_cancer = np.zeros(n_of_women)
life_time = []
for i in range(n_of_women):
    sojourn_time = np.zeros(n-1)
    state = 0  # state 1
    total_time = 0
    while state < 4:  # state 5
        if ((state > 1) and (sojourn_time[2] == 0 and
   sojourn_time[3] == 0)):
            prop_of_cancer_reapperead = np.sum(sojourn_time
   [:2])
        else:
            prop_of_cancer_reapperead = 0

        # sojourn time - that we will remain in state 1 (
   lifetime probability)
```

```
        sojourn_time[state] = np.random.exponential(-1/Q[
    state, state])
        total_time = total_time + sojourn_time[state]

        # jump to next state
        q = np.array(-Q[state, state+1:]/Q[state, state]).
    flatten()
        new_states = range(state+1, n)

        while True:
            U1 = np.random.uniform(0, 1)
            U2 = np.random.uniform(0, 1)
            I = int(np.floor(len(q)*U1))
            if U2 <= q[I]/np.max(q):
                state = new_states[I]
                break

     life_time.append(total_time)

     liftime_dist[i, :] = sojourn_time
     reapperead_cancer[i] = prop_of_cancer_reapperead
```

```
mean = np.round(np.mean(np.sum(liftime_dist, axis=1)),2)
plt.figure(figsize=(10,7))
plt.hist(np.sum(liftime_dist, axis=1), bins=20, alpha=0.8,
    color='lightsteelblue', edgecolor='grey')
plt.vlines(mean, ymin =0, ymax =200 , color="lightcoral",
    alpha =0.8, label=r"$\bar{x}$"+f"= {mean}")
plt.legend()
plt.xlabel('Months of living')
plt.ylabel('Number of women')
plt.grid()
plt.title('Simulation of lifetime distribution after surgery
```

```
    for the 1000 women');
```

```
print('Lifetime for simulation of 1000 women')
print('Mean', np.round(np.mean(np.sum(liftime_dist, axis=1)
   /12), 3))
print('Confidence interval for mean', np.round(
   confidence_interval(np.sum(liftime_dist, axis=1)/12), 3))
print('The standard deviation',np.round(np.mean(np.std(
   liftime_dist, axis=1)/12), 3))
print('Confidence interval for std', np.round(
   confidence_interval(np.std(liftime_dist, axis=1)/12), 3))
```

```
after_305_months = reapperead_cancer[reapperead_cancer >
   30.5].shape[0]/n_of_women*100
print('Proportion of women has the cancer reappeared
   distantly after 30.5 months', after_305_months, '%')
```

Task 8

```
Qs = np.matrix('-0.0085 0.005 0.0025 0; \
            0 -0.014 0.005 0.004; \
            0 0 -0.008 0.003; \
            0 0 0 -0.009')
```

```
N = int(np.max(np.sum(liftime_dist, axis=1)))
th_lifetime = np.zeros(N)
a = []
for i in range(N):
    n = Q.shape[0]
    init_p = np.array([1,0,0,0])
    exp_Q = linalg.expm(Qs*i)
    F_T = 1 - (np.matmul(np.matmul(init_p, exp_Q), np.ones(n
   -1)))
    th_lifetime[i] = F_T
    life_time.sort()
```

```
    a.append(sum(x > i for x in life_time))
emp = 1 - (np.array(a) / n_of_women)
```

```
# compare theoretical and simulated distributions
plt.figure(figsize=(10,7))
plt.plot(th_lifetime, label='Theoretical', color='orangered'
   )
plt.plot(emp, label='Simulated', color='limegreen')
plt.xlabel('Months of living')
plt.ylabel('Percentage who died')
plt.title('Simulation of 1000 women showing the months of
   living after suergery')
plt.legend();
```

```
# Kolmogorov-Smirnov test
D = max(abs(emp-th_lifetime))
prob = float(len(emp)) * float(len(th_lifetime)) / (float(
   len(emp)) + float(len(th_lifetime)))
p_value = stats.kstwo.sf(D, np.round(prob))
```

```
print('Kolmogorov-Smirnov test')
print('D:', D)
print('p-value:', p_value)
```

Task 9

```
Qt = np.matrix('0 0.0025 0.00125 0 0.001; \
                0 0 0 0.002 0.005; \
                0 0 0 0.003 0.005; \
                0 0 0 0 0.009; \
                0 0 0 0 0')
```

```
for i in range(0,len(Qt)):
    Qt[i,i] = - np.sum(Qt[i,:])
Qt
```

```
# simulate
n_of_women = 1000
n = Qt.shape[0]
liftime_dist_2 = np.zeros((n_of_women, n-1))
reapperead_cancer = np.zeros(n_of_women)
life_time = []
for i in range(n_of_women):
    sojourn_time = np.zeros(n-1)
    state = 0  # state 1
    total_time = 0
    while state < 4:  # state 5
        if ((state > 1) and (sojourn_time[2] == 0 and
   sojourn_time[3] == 0)):
            prop_of_cancer_reapperead = np.sum(sojourn_time
   [:2])
        else:
            prop_of_cancer_reapperead = 0

        # sojourn time - that we will remain in state 1 (
   lifetime probability)
        sojourn_time[state] = np.random.exponential(-1/Qt[
   state, state])
        total_time = total_time + sojourn_time[state]

        # jump to next state
        q = np.array(-Qt[state, state+1:]/Qt[state, state]).
   flatten()
        new_states = range(state+1, n)

        # ath
        while True:
            U1 = np.random.uniform(0, 1)
```

```
            U2 = np.random.uniform(0, 1)
            I = int(np.floor(len(q)*U1))
            if U2 <= q[I]/np.max(q):
                state = new_states[I]
                break

    life_time.append(total_time)

    liftime_dist_2[i, :] = sojourn_time
    reapperead_cancer[i] = prop_of_cancer_reapperead
```

```
# Kaplan Meier estimate for treatment and not treatment
N_treatment = int(np.max(np.sum(liftime_dist_2, axis=1)))
N_not_treatment = int(np.max(np.sum(liftime_dist, axis=1)))
survival_treatment = np.zeros(N_treatment)
survival_not_treatment = np.zeros(N_not_treatment)

for i in range(N_treatment):
    expr = liftime_dist_2[np.sum(liftime_dist_2, axis=1) < i
   ].shape[0]
    St1 = (n_of_women - expr) / n_of_women
    survival_treatment[i] = St1

for i in range(N_not_treatment):
    expr = liftime_dist[np.sum(liftime_dist, axis=1) < i].
   shape[0]
    St2 = (n_of_women - expr) / n_of_women
    survival_not_treatment[i] = St2
```

```
plt.figure(figsize=(10,7))
plt.plot(survival_treatment, label='With treatment', color='
   limegreen')
plt.plot(survival_not_treatment, label='Without treatment',
   color='orangered')
```

```
plt.xlabel('Months')
plt.ylabel('Percent survival')
plt.title('Kaplan-Meier estimator of a survival function')
plt.legend();
```

Part 3: Estimation

Task 12

```
import numpy as np
import matplotlib.pyplot as plt
import math
import random
import scipy
from scipy import stats, linalg
import seaborn as sns
```

```
Q = np.matrix('-0.0085 0.005 0.0025 0 0.001; \
                0 -0.014 0.005 0.004 0.005; \
                0 0 -0.008 0.003 0.005; \
                0 0 0 -0.009 0.009; \
                0 0 0 0 0')
```

```
# simulate
n_of_women = 1000
n = Q.shape[0]
liftime_dist = np.zeros((n_of_women, n-1))
reapperead_cancer = np.zeros(n_of_women)
life_time = []
for i in range(n_of_women):
    sojourn_time = np.zeros(n-1)
    state = 0  # state 1
    total_time = 0
    while state < 4:  # state 5
        if ((state > 1) and (sojourn_time[2] == 0 and
```

```
    sojourn_time[3] == 0)):
            prop_of_cancer_reapperead = np.sum(sojourn_time
    [:2])
        else:
            prop_of_cancer_reapperead = 0

        # sojourn time - that we will remain in state 1 (
    lifetime probability)
        sojourn_time[state] = np.random.exponential(-1/Q[
    state, state])
        total_time = total_time + sojourn_time[state]

        # jump to next state
        q = np.array(-Q[state, state+1:]/Q[state, state]).
    flatten()
        new_states = range(state+1, n)

        # ath
        while True:
            U1 = np.random.uniform(0, 1)
            U2 = np.random.uniform(0, 1)
            I = int(np.floor(len(q)*U1))
            if U2 <= q[I]/np.max(q):
                state = new_states[I]
                break

     life_time.append(total_time)

     liftime_dist[i, :] = sojourn_time
     reapperead_cancer[i] = prop_of_cancer_reapperead
```

```
n_months = 48
max_possible = int(np.ceil(np.max(np.sum(liftime_dist, axis
```

```
   =1))/n_months)*n_months)
months = list(range(0, max_possible+1, n_months))

# calculate Y for all women
Y = np.zeros([n_of_women, len(months)])
for i in range(1, len(months)):
    month = months[i]
    for j in range(n_of_women):
        sojourn_time = liftime_dist[j,:]
        alive = 0
        for state in range(n-1):
            if np.sum(sojourn_time[0:state]) > month:
                Y[j,i] = state
                alive = 1
                break
        if alive == 0:
            Y[j,i] = n-1
```

```
x_labels = [f"{months[i]} months" for i in range(1, len(
   months))]

# Plotting the time series data as a heatmap
fig, ax = plt.subplots(figsize=(10, 7))
sns.heatmap(Y, cmap='viridis', cbar_kws={'label': 'State'},
   ax=ax)

ax.set_xticks(range(len(x_labels)))
ax.set_xticklabels(x_labels, rotation=45)
ax.set_yticks([])
ax.set_xlabel('Observation Points')
ax.set_ylabel('Simulated women')
ax.set_title('Time Series of State Transitions')
plt.tight_layout();
```

5.1 Task 13 - Unfinished

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









def MCMC_task12(Q):

    N = len(Q)

    states = [0]
    prev_time = 0
    intervals = []
    lifetime = 0
    current_state=0

    #record time spent in each state
    times_states = []
    while current_state != N-1: # Death occurs in final
   state



        # performing transition

        time_in_current_state = np.random.exponential(scale
   =-1/Q[current_state][current_state])

        lifetime += round(time_in_current_state, 2)
```

```
        intervals.append((prev_time, lifetime))
        prev_time=lifetime

        transition_probabilities = []

        for i in range(N):

            if i != current_state:
                transition_probabilities.append(-Q[
    current_state][i]/Q[current_state][current_state])
            else:
                transition_probabilities.append(0) # Cannot
    remain in same state any longer


        new_state = np.random.choice(list(range(N)), p =
    transition_probabilities) # New column
        states.append(new_state)

        # Updating
        current_state = new_state


    return states, intervals


def MCMC_13(Q, initial_state, target_state):

    N = len(Q)

    N1 = np.zeros((5, 5))
    S = np.zeros(5)
```

```
 current_state= initial_state
 states = [current_state]
 prev_time = 0
 intervals = []
 lifetime = 0



 while lifetime <= 48 and current_state != N-1: # end of
48 month timeframe

     if initial_state==target_state:
         time_in_current_state = np.random.exponential(
scale=-1/Q[current_state][current_state])

         # Update sojourn time
         S[current_state] += time_in_current_state

         lifetime += round(time_in_current_state, 2)
         states.append(current_state)

     else:

         time_in_current_state = np.random.exponential(
scale=-1/Q[current_state][current_state])

         # Update sojourn time
         S[current_state] += time_in_current_state


         lifetime += round(time_in_current_state, 2)
```

```
            intervals.append((prev_time, lifetime))
            prev_time=lifetime

            transition_probabilities = []

            for i in range(N):
                # Performing transition

                if i != current_state:
                    transition_probabilities.append(-Q[
current_state][i]/Q[current_state][current_state])
                else:
                    transition_probabilities.append(0) #
Cannot remain in same state any longer


            new_state = np.random.choice(list(range(N)), p =
 transition_probabilities) # New column

            # Update sojourn time
            S[current_state] += time_in_current_state
            states.append(new_state)

            # Updating
            current_state = new_state


    # Update jumps
    for i in range (len(states)-1):
        if states[i]!=states[i+1]:
            N1[states[i]] [states[i+1]] += 1
```

```
    return states, N1, S


def get_Y(n, Q):

    Y = []

    for i in range(n):

        states, intervals = MCMC_task12(Q)
        #if i==0:
            #print(f"states, intervals = {states}, {
  intervals}")

        observed_states = []
        time = 0
        lifetime = intervals[-1][1]

        while time <= lifetime:

            for i, state_time in enumerate(intervals): #
  Linear search through intervals

                if time >= state_time[0] and time <
  state_time[1]: # Interval contains current time

                    observed_states.append(states[i])

            time += 48

        observed_states.append(4)
```

```
        Y.append(observed_states)


    return Y


def update_diagonal_elems(Q):
    sum = 0
    for i in range (len(Q)):
        for j in range (len(Q)):

            # set last row all zeros
            if i == len(Q) - 1:
                Q[i][j] = 0

            # set Q as lower triangular matrix
            if i > j:
                Q[i][j] = 0

    for i in range(len(Q)):
        for j in range (len(Q)):
            if i == j:
                sum = 0
                for k in range(i + 1, len(Q)):
                    sum += Q[i][k]
                Q[i][i] = -sum

    return Q
```

```
# start

Q = np.array([[-0.0085, 0.005, 0.0025, 0, 0.001],
                  [0, -0.014, 0.005, 0.004, 0.005],
                  [0, 0, -0.008, 0.003, 0.005],
                  [0, 0, 0, -0.009, 0.009],
                  [0, 0, 0, 0, 0]])

Q0 = np.array([[0, 0.0025, 0.00125, 0, 0.001],
                    [0, 0, 0, 0.002, 0.005],
                    [0, 0, 0, 0.003, 0.005],
                    [0, 0, 0, 0, 0.009],
                    [0, 0, 0, 0, 0]])
Q0 = update_diagonal_elems(Q0)

print(Q0)

# Observations
observations = get_Y(1000, Q)



# Change matrix Q and set it as Q1
Q1 = Q / 10 + Q0*9/10

Q1 = update_diagonal_elems(Q1)


# PROGRAM LOOP
```

```
N = np.zeros((5, 5))
S = np.zeros(5)

errors_collection = []
err = np.inf

k = 0
while err >= 0.0001:
    print(k)
    N = np.zeros((5, 5))
    S = np.zeros(5)
    for obs in observations:

        for i in range (len(obs)-1):
            initial_state = obs[i]
            target_state = obs[i+1]
            accepted = False
            while accepted == False:

                new_sim, jumps, sojourn = MCMC_13(Q1,
  initial_state, target_state)

                # Only accept state if the last generated
  state is the same as the target state
                if new_sim[-1] == target_state:

                    # Update S and N
                    N += jumps
                    S += sojourn
```

```
                    accepted = True

    # Update Q after going through all observations
    Q_before = Q1

    Q1 = np.zeros((5, 5))
    for i in range(len(Q1)):
        for j in range(len(Q1)):
            if i != j:
                Q1[i][j] = (N[i][j] / S[i])

    #set last row to zeros and update diagonal elements
    Q1 = update_diagonal_elems(Q1)

    # Compute error
    mat = (Q_before - Q1)
    err = np.linalg.norm(mat , ord=np.inf)
    errors_collection.append(err)

    # Iterate
    k += 1


print(f"Q = {Q1}")

plt.plot([i for i in range(len(errors_collection))],
   errors_collection, color="lightcoral")
plt.ylabel(r"$||Q_{k+1}-Q_{k}||_{\infty}$")
plt.xlabel(r"$k$")
plt.title("Error value vs Iteration")
plt.show()
```