Stochastic Process

MC 303

Practical File

Submitted By

Sidharth

2K18/MC/114

Author:

- **Sidharth**
- **2K18/ MC/ 114**

Objective:

Verification of mean and variance of random variable X Probability of which is represented with Binomial Distibution Function.

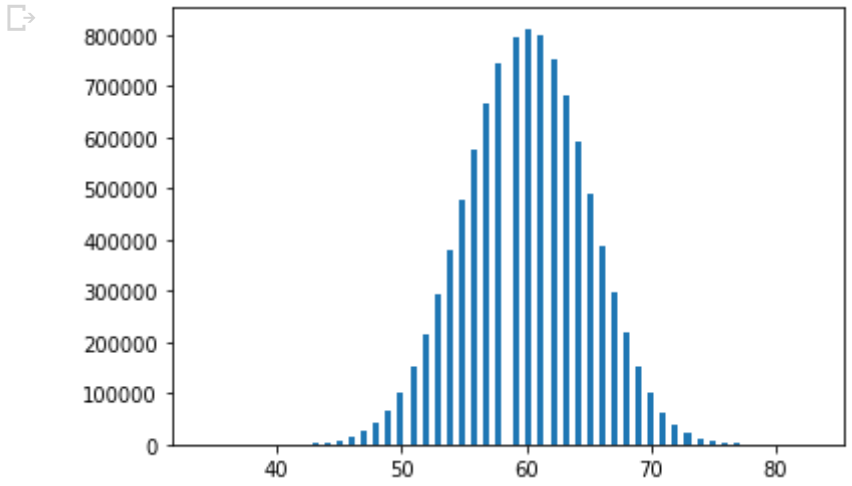
Theory:

The binomial distribution function specifies the number of times (x) that an event occurs in n independent trials where p is the probability of the event occurring in a single trial. It is an exact probability distribution for any number of discrete trials. If n is very large, it may be treated as a continuous function.

Distribution	Functional Form	Mean	Standard Deviation
Binomial	$f_b(x) = \frac{n!p^x(1-p)^{n-x}}{x!(n-x)!}$	np	$\sqrt{np(1-p)}$

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

```
# picks out 10^7 samples from a binomial distribution with n=100 & p=0.6 & q=0.4
binomial_dist= np.random.binomial(100,.6,10**7)
plt.hist(binomial_dist, bins = 100)
plt.savefig('Sample size {}'.format(10**7))
plt.show()
binomial_dist = np.random.binomial(100,.6,10**7)
```



Results:

```
print('The theoretical average is: {},\nThe experimentally calculated average is: {}'
.format(60,np.average(binomial_dist)))

print('The theoretical deviation is: {},\nThe experimentally calculated deviation is: {}'
.format((100*.4*.6)**.5,np.std(binomial_dist)))
```

```
The theoretical average is: 60,
The experimentally calculated average is: 59.9995078
The theoretical deviation is: 4.898979485566356,
The experimentally calculated deviation is: 4.8989495565620125
```

Discussion:

The values calculated theoretically & experimentally for Average & Deviation are equivalent (very close). If large sample size is taken there is no observable difference among them.

Sidharth

2K18/MC/114

Experiment 2

Objective:

Demonstrating a Stochastic Process with Discrete Index Set

(a) with discrete state space, (b) with continuous state space.

Theory:

A stochastic process is a family of random variables {X(t), t ∈ T} defined on a given probability space, indexed by parameter t, where t varies over an index set T. The values assumed byX(t) are called its states, and the set of all possible values forms the state pace of the process.

Stochastic processes are classified on the basis of the underlying index set T and state space S. If T = {0,1,2,..}, or T = {0, ±1, ±2,..}, the stochastic process is said to be discrete parameter process and is usually indicated by {Xn} .The state space is classified as discrete if it is finite or countable and process is classified as continuous if it consists of an interval, finite or infinite of the real line.

Code:

A: Discrete state space

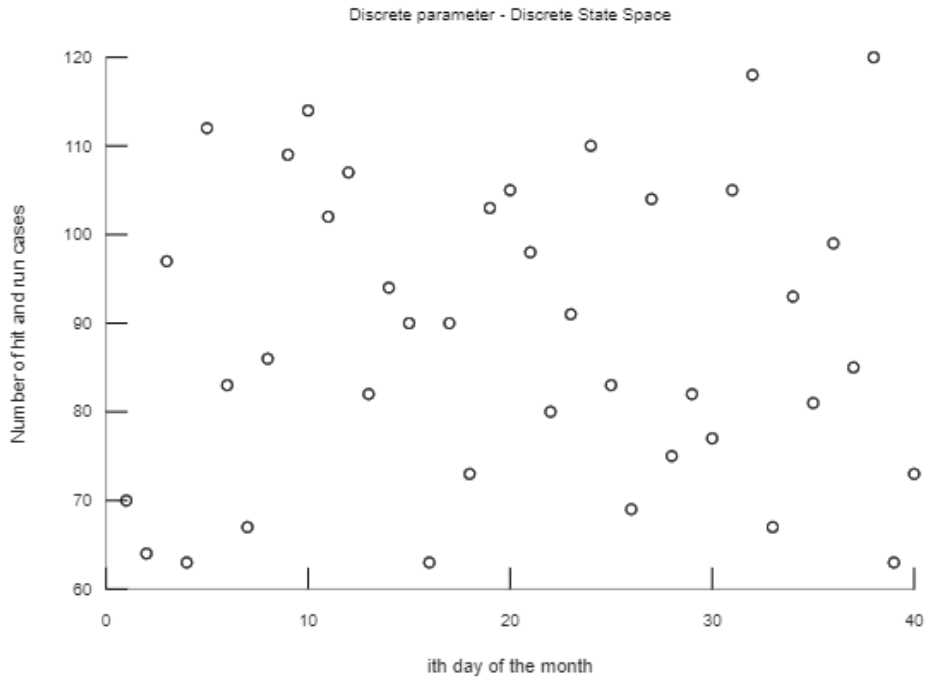
```
x = [1:1:40];

y = 60 +randi ([0 60], 40, 1);

p = scatter (x,y);

xlabel ("ith day of the month");
```

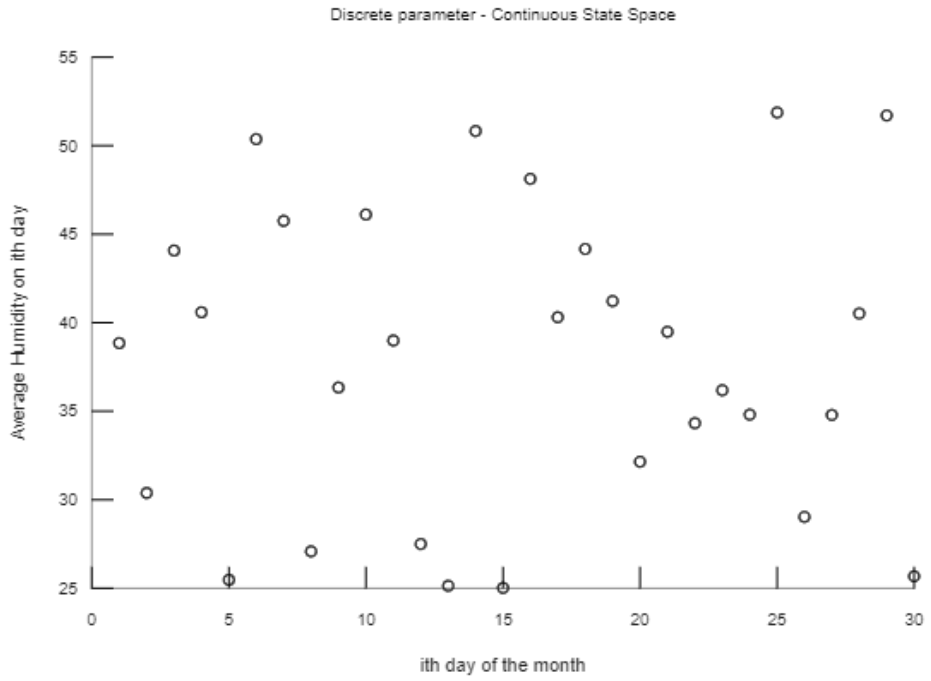
```
ylabel ("Number of hit and run cases");
title ("Discrete parameter - Discrete State Space");
```

B: Continuous state space

```
x = [1:1:30];
y = 25 + 27 * rand(30, 1);
p = scatter (x,y);
xlabel ("ith day of the month");
ylabel ("Average Humidity on ith day");
title ("Discrete parameter - Continuous State Space");
```

Result:





Discussion:

The graphs for demonstrating a stochastic process with discrete index set

(A) with discrete state space, (ith day vs No. of hit and run cases)

(B) with continuous state space (ith day vs Average Humidity on ith day) are obtained.

Sidharth

2K18/MC/114

Experiment 3

Objective:

Demonstrating a Stochastic Process with Continuous Index Set (a) with discrete state space, (b) with continuous state space.

Theory:

A stochastic process is a family of random variables {X(t), t ∈ T} defined on a given probability space, indexed by parameter t, where t varies over an index set T. The values assumed byX(t) are called its states, and the set of all possible values forms the state pace of the process. Stochastic processes are classified on the basis of the underlying index set T and state space S. If T = {0,1,2,..}, or T = {0, ±1, ±2,..}, the stochastic process is said to be discrete parameter process and is usually indicated by {Xn} .The state space is classified as discrete if it is finite or countable and process is classified as continuous if it consists of an interval, finite or infinite of the real line.

Code:

A. Discrete state space

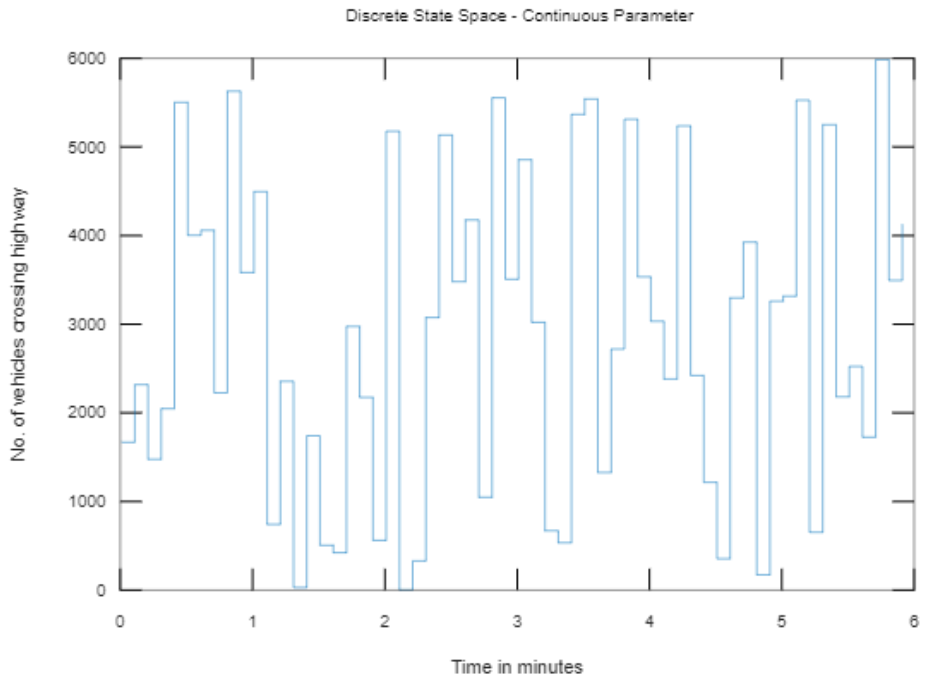
```
a= [0.01:0.1:6];

b= randi([0 6000],60,1);

pl = stairs(a,b);

xlabel('Time in minutes');

ylabel('No. of vehicles crossing highway');
```

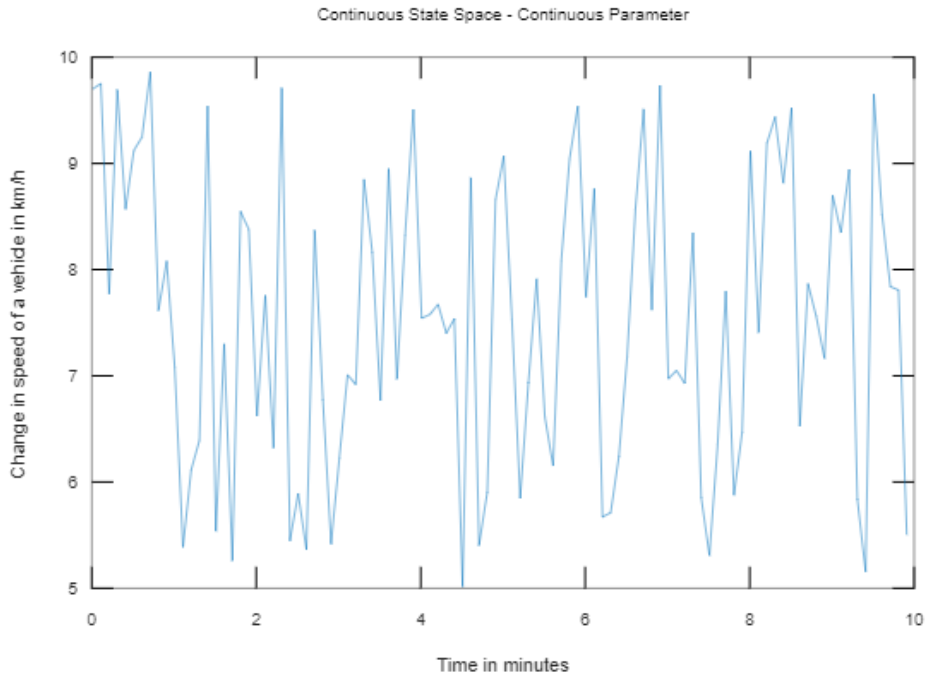
```
title(' Discrete State Space - Continuous Parameter');
```

B. Continuous state space.

```
a = [0.01:0.1:10];
b = 5 + 5.*rand(100,1);
pl = plot(a,b);
xlabel('Time in minutes');
ylabel('Change in speed of a vehicle in km/h');
title('Continuous State Space - Continuous Parameter');
```

Result:





Discussion:

The graphs for demonstrating a stochastic process with discrete index set

(A) with discrete state space, (Time vs No. of vehicles crossing highway)

(B) with continuous state space (Time vs Change in speed of a vehicle in km/h)

are obtained.

Author:

- **Sidharth**
- *2K18/ MC/ 114*

Objective:

Demonstrate the unrestricted random walk and analyse the probability of being at a given state after n steps.

Theory:

A random walk is a mathematical object, known as a stochastic or random process, that describes a path that consists of a succession of random steps on some mathematical space such as the integers. An elementary example of a random walk is the random walk on the integer number line, Z , which starts at 0 and at each step moves +1 or −1 with equal probability.

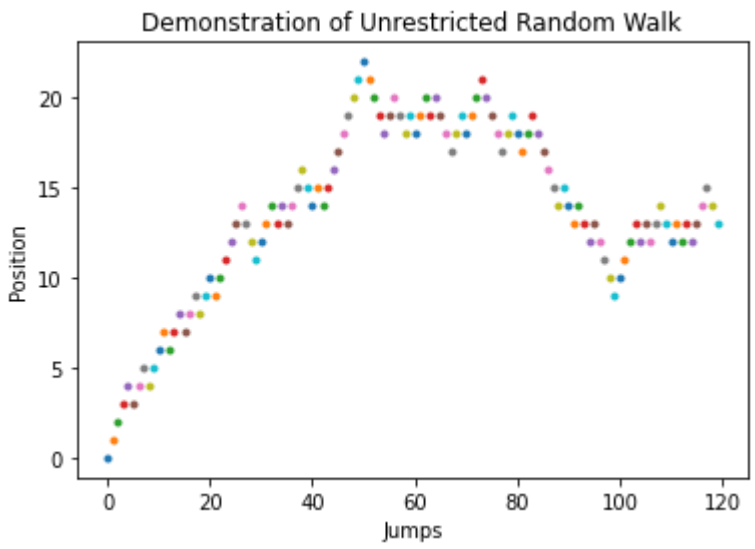
A simple random (or unrestricted random walk) walk on a line or in one dimension occurs with probability p when walker step forward (+1) and/or has probability q=1−p if walker steps back (−1). For ith step, the modified Bernoulli random variable Wi (takes the value +1 or −1 instead of {0,1}) is observed and the position of the walk at the nth step can be found by,

$$\begin{aligned} X_n &= X_0 + W_1 + W_2 + \cdots + W_n \\ &= X_0 + \sum_{i=1}^{n} W_i \\ &= X_{n-1} + W_n \end{aligned}$$

Code and Output:

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

```
p = 0.55
q = 0.45
tsteps = 120
initial = 0
step = 0


while(step<tsteps):
    plt.plot(step,initial,'.')
    if(np.random.random() < p):
        initial += 1
    else:
        initial -= 1
    step += 1

plt.ylabel('Position')
plt.xlabel('Jumps')
plt.title('Demonstration of Unrestricted Random Walk')
plt.show()
```




```
from scipy.stats import norm

p = 0.55
q = 0.45
state = 20
steps = 120
if(p+q<1):
    r = 1-p-q
    c = 0.5
else:
    c = 1
```

```
mean = p-q
sd = np.sqrt(p+q - (p-q)**2)
norm1 = norm.cdf((state+c- steps*mean)/(sd*np.sqrt(steps)))
norm2 = norm.cdf((state-c-steps*mean)/(sd*np.sqrt(steps)))
prob = norm1 - norm2

print('Probabily of being at state ',state,' after ',steps,' steps is: ',prob)
```

```
Probabily of being at state  20  after  120  steps is:  0.05588162653138029
```

Results:

We Demostrated 1D random walk assuming p = 0.55 & q = 0.45 We also calculated probabilty of being at a state after n steps using normal cdf.

Discussion:

Random Walk can give us a good understanding of the statistical processes involved in genetic drift, and they describe an ideal chain in polymer physics. They are also important in finance.

Author:

- **Sidharth**
- *2K18/ MC/ 114*

Objective:

Demonstrating a Bernoulli process. Specifically, WAP to find the probability that in case of a Bernoulli process,

1. Out of n trials k are successes
2. kth success occurs at the nth trial

Theory:

If X is a Binomial random variable, we denote this X ~ Bin(n, p), where p is the probability of success in a given trial. A binomial random variable has the following properties:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Suppose the probability of hitting a target with a rock is 0.4 and missing a target is 0.6 and assume that each throw is independent. A person throws the rock 20 times we have to find probability -

1. He hits the target exactly 13 times
2. He hits the target 7th time at the 14th throw

Code:

```
import math

n = 20
k = 13
p = 0.4

ans = math.factorial(n)
```

```
ans = ans/math.factorial(n-k)
ans = ans/math.factorial(k)
ans = ans*pow(p,k)*pow(1-p,n-k)
print("Probability of hitting the target exactly",k,"times is:",ans)

n = 14
k = 7

temp = math.factorial(n-1)/(math.factorial(n-k) * math.factorial(k-1))
temp = temp * pow(p,k-1) * pow(1-p,n-k) * p
print("Probability of hitting target 7th time at 14th throw is:",temp)
```

```
Probability of hitting the target exactly 13 times is: 0.014563052125736147
Probability of hitting target 7th time at 14th throw is: 0.07870384963584003
```

Result:

The probability calculated are as follows:

```
print("Probability of hitting the target exactly",k,"times is:",ans)
print("Probability of hitting target 7th time at 14th throw is:",temp)
```

```
Probability of hitting the target exactly 7 times is: 0.014563052125736147
Probability of hitting target 7th time at 14th throw is: 0.07870384963584003
```

Discussion:

Computed the probability in case a Bernoulli process, (a) out of n trials k are successes, (b) kth success occurs at the nth trial.

Author:

- **Sidharth**
- *2K18/ MC/ 114*

Objective:

Demonstrating Poisson Process. WAP to find the probability that in case of Poisson process with rate λ, in a length of time t there are exactly k arrivals.

For example: Consider a transistor battery having exponential lifetime with mean as 2 months. In case six such spares batteries are available and if the time to replace a battery is negligible, find the probability that transistor will work for at least one year. In case mean failure time of the successive spare batteries are given by 2/n, then find this probability.

Theory:

The basic form of Poisson process, often referred to simply as "the Poisson process", is a continuous time counting process {N(t), t ≥ 0} that possesses the following properties:

- Independent increments (the numbers of occurrences counted in disjoint intervals are independent of each other).
- Stationary increments (the probability distribution of the number of occurrences counted in any time interval only depends on the length of the interval).
- The probability distribution of N(t) is a Poisson distribution.
- No counted occurrences are simultaneous.

Consequences of this definition include:

- The probability distribution of the waiting time until the next occurrence is an exponential distribution.
- The occurrences are distributed uniformly on any interval of time. (Note that N(t), the total number of occurrences, has a Poisson distribution over (0, t], whereas the location of an individual occurrence on t ∈ (a, b] is uniform).

Code and Output:

```
import math
def poisson_homo(parameter, n):
    p=0;
```

```
    for i in range(n+1):
        temp = math.exp(-parameter) * math.pow(parameter, i)
        temp /= math.factorial(i)
        p += temp
    return p

ans = poisson_homo(12, 12)
print(ans)
```

```
0.5759652485730646
```

```
import math
def poisson_non_homo(parameter, n):
    p=0;
    for i in range(1, n+1):
        temp = math.exp(-parameter/i) * math.pow(parameter/i, i)
        temp /= math.factorial(i)
        p += temp
    return p

ans = poisson_non_homo(12, 12)
print(ans)
```

```
0.48202206526406666
```

Result:

With the help of the above program, we have successfully managed to solve a Poisson process, both for homogenous as well as non homogeneous poisson process.

Discussion:

As observed, non homogenous process has uncertainty as can be seen with the help of the low probability.

Sidharth

2K18/MC/114

Experiment 7

Objective:

Demonstrate Simple Random Walk. WAP to find the probability that in case of an unrestricted simple random walk the particle is at the kth position at time n using

(a) Central Limit Theorem (b) Generating Function

Theory:

Suppose that the random walks start at origin and the particle is free to move indefinitely in either direction.

- By central limit theorem, we can say that χ n will be normally distributed. Therefore,
  prob(χ_n = k) = $\Phi((k + 1/2 + c - n\mu)/(\sigma\sqrt{n}))$ - $\Phi((k - 1/2 - c - n\mu)/(\sigma\sqrt{n}))$
  where ϕ(y) is the standard normal distribution function.

- The probability generation function of χ n is given by,
  G(z, s) = $(z/(-spz^2 + z\{1 - s(1 - p - q)\} - sp))$
  where prob(χ n = k) = coeff. of z k s n in G(z, s)

For this experiment consider:
A random walk in which the particle starts at origin. Finding the probability that after 10 steps, the particle will be at position 4. Given that probability of positive jump is 0.42 and that of negative jump is 0.40.

So,

p = 0.42, q = 0.40
j = 3.5, k = 4.5

Code:

Central Limit Theorem:

```
import math from scipy.integrate
import quad as integrate
import sympy as sp
x = sp.Symbol('x')
def expression(x):
    return(math.exp(-((x-(n*mu))**2)/(2*(var)*n)))
def findNorm(j, k):
    i = integrate(expression, j, k)
    norm = (1/math.sqrt(2*math.pi*n*(var)))*i[0]
    print("The probability of finding x at k: "+str(norm))
    mu = 0.02
    var = 0.8196
    n = 10
    findNorm(3.5, 4.5)
```

Output: The probability of finding x at k: 0.1285

Generating Function:

```
from sympy import symbols, diff, N
import math
z, s, p, q = symbols('z s p q', real=True)
f = (z) / ((-s * (z ** 2) * p) + (1 - (s * (1 - p - q))) * z - q * s)
n = 10
k = 4
```

```
f = f.subs(p, 0.49)
f = f.subs(q, 0.44)
for _ in range(n):
        f = diff(f, s)
for _ in range(k):
        f = diff(f, z)

f = f.coeff(s * z)
f = f / (math.factorial(10) * math.factorial(4))
f = f.subs(z, 1)
f = f.subs(s, 1)
print("The Probability of Particle being at K=4 after n steps is: ",
float(f))
```

Output: The Probability of Particle being at K=4 after n steps is: 0.1285

Result: used central limit theorem and generating function to calculate the probabilities.

Discussion: we use continuity correction c for better approximation. The value of c = 1 if p + q = 1, and the value of c = ½ if p + q < 1.

Author:

- **Sidharth**
- *2K18/ MC/ 114*

Objective:

Demonstrate the Renewal Process. WAP to find the expected waiting time until the nth renewal in case of a renewal process with renewal cycle length distributed

1. normally with mean μ and standard deviation σ, (μ > 3σ)
2. exponentially with parameter λ.

Demonstrate it by taking suitable values for (a) and (b) both.

Theory:

A Renewal Process is a general case of Poisson Process in which the inter arrival time of the process or the time between failures does not necessarily follow the exponential distribution. A counting process N(t) that represents the total number of occurrences of an event in the time interval (0,t) is called a renewal process. If the time between failures are independent and identically distributed random variables.

The probability that there are exactly n failures occurring by time t can be written as,

> P{N(t) = n} = P{N(t) ≥ n} – P{N(t) >n}

And,

> Tk = Wk + Wk-1

Code & Output:

1. Normally with mean μ and standard deviation σ, (μ > 3σ)

```
n = int(input('Enter value of n: '))
mu = int(input('Enter value of μ: '))
sigma = input('Enter value of σ: ')
ans = n*mu
print('Expected waiting time until the', n, 'th renewal = ', ans)
```

```
Enter value of n: 87
Enter value of μ: 3
Enter value of σ: 0.6
Expected waiting time until the 87 th renewal =  261
```

2. Exponentially with parameter λ

```
n = int(input('Enter value of n: '))
l = float(input('Enter value of λ: '))
ans = n/l
print('Expected waiting time until the', n, 'th renewal = ', ans)
```

```
Enter value of n: 122
Enter value of λ: 0.245
Expected waiting time until the 122 th renewal =  497.9591836734694
```

Result:

Demonstrated the Renewal Process and calculated the expected waiting time until the nth renewal.

Discussion:

Calculation of the expected waiting time until the nth renewal in case of a renewal process with renewal cycle length distributed normally with mean μ and standard deviation σ and exponentially with parameter λ are being calculated through above formula.

Author:

- **Sidharth**
- *2K18/ MC/ 114*

Objective:

Demonstrate Markov Chain. WAP to implement Markov Chain

1. Gambler Ruin Problem

2. Weather Forecasting Problem

Theory:

Modern probability theory studies chance processes for which the knowledge of previous outcomes influences predictions for future experiments. In principle, when we observe a sequence of chance experiments, all of the past outcomes could influence our predictions for the next experiment. For example, this should be the case in predicting a student’s grades on a sequence of exams in a course. But to allow this much generality would make it very difficult to prove general results.

A Markov chain is a stochastic process, but it differs from a general stochastic process in that a Markov chain must be "memory-less." That is, (the probability of) future actions are not dependent upon the steps that led up to the present state. This is called the Markov property. While the theory of Markov chains is important precisely because so many "everyday" processes satisfy the Markov property, there are many common examples of stochastic properties that do not satisfy the Markov property.

We describe a Markov chain as follows: We have a set of states, S = {s1, s2,...,sr}. The process starts in one of these states and moves successively from one state to another. Each move is called a step. If the chain is currently in state si, then it moves to state sj at the next step with a probability denoted by pij , and this probability does not depend upon which states the chain was in before the current state.

Code and Output:

1. Gambler Ruin Problem

> A gambler has a fortune of ₹20 and he bets ₹1 at a time and wins ₹1 with probability 3/10. He stops playing if he loses all his fortune or doubles it. Write the transition probability matrix that he loses his fortune at the end of 30 plays.

```
p = 0.3
n = 30
fortune = 20

import numpy as np
tm=np.zeros(shape=(int(2*fortune+1), int(2*fortune+1)))
print("Absorbing barriers are at", 0, "and", 2*fortune)
for i in range(0, int(2*fortune+1)):
    for j in range(0 ,int(2*fortune+1)):
        if i==0 or i==int(2*fortune):
            tm[i][i]=1
        else:
            if i+1==j:
                tm[i][j]=p
            elif i-1==j:
                tm[i][j]=1-p

b=np.linalg.matrix_power(tm, n)
print("Probability of gambler losing his fortune at the end of", n, '
print(b[int(fortune)][0])
```

```
Absorbing barriers are at 0 and 40
Probability of gambler losing his fortune at the end of 30 plays:
0.08067671520774745
```

2. Weather Forecasting Problem

> Provided with every states probability that rain occurs yesterday and today. Let it rained on both Saturday and Sunday. What is the probability that it will rain on Monday?

> We have Probabilities:

- State 0: Rained today and yesterday: 0.4
- State 1: Rained today but not yesterday: 0.2
- State 2: Rained yesterday but not today: 0.3
- State 3: Didn't rain today or yesterday: 0.5

```
import numpy as np
import math

state0 = 0.4
state1 = 0.2
state2 = 0.3
state3 = 0.5
wid = 4
hig = 4

tm = [[0 for x in range(wid)] for y in range(hig)]

tm[0][0] = state0
tm[0][2] = 1-state0
tm[1][0] = state1
tm[1][2] = 1-state1
tm[2][1] = state2
tm[2][3] = 1-state2
tm[3][1] = state3
tm[3][3] = 1-state3

print("The transition matrix:\n")
for i in a:
    print(i)

a=np.linalg.matrix_power(a, 2)
print("\nProbability of raining tommorow with the past two days havir
print(a[0][0]+a[0][1])
```

```
The transition matrix:

[0.08933236 0.26764818 0.26783304 0.37518642]
[0.08927768 0.26764124 0.26749552 0.37558556]
[0.08921606 0.26813975 0.26764124 0.37500295]
[0.0893301  0.26785925 0.2682754  0.37453525]

Probability of raining tommorow with the past two days having rained:
0.35714289990042936
```

Result:

Implemented Markov Chain for Gambler Ruin Problem and Weather Forecasting Problem.

Discussion:

Markov chains are an important concept in stochastic processes. They can be used to greatly simplify processes that satisfy the Markov property, namely that the future state of a stochastic variable is only dependent on its present state.

Author:

- **Sidharth**
- *2K18/ MC/ 114*

Objective:

Demonstrate Markov Chain Process. WAP to implement Markov Chain special cases

1. To find steady state probabilities in case of ergodic Markov Chain.
2. To find that the specific state in a Markov chain is a recurrent or transient.

Theory:

A Markov chain is a mathematical system that experiences transitions from one state to another according to certain probabilistic rules. The defining characteristic of a Markov chain is that no matter how the process arrived at its present state, the possible future states are fixed. In other words, the probability of transitioning to any particular state is dependent solely on the current state and time elapsed. The state space, or set of all possible states, can be anything: letters, numbers, weather conditions, baseball scores, or stock performances.

Code & Output:

1. To find steady state probabilities in case of ergodic Markov Chain.

```
import numpy as np
Arr = np.array([[0,0.5,0.5,0],[0.25,0,0.5,0.25],
```

```
                [0.6,0.4,0,0],[0.3,0.4,0.3,0]])
n = 4
print("Transition Probability Matrix: ")
print(Arr)
Arr = Arr.T
for i in range(0,n):
    Arr[i][i] -= 1
    Arr[0][i] = 1
B = [0]*n
B[0] = 1
z = np.linalg.solve(Arr,B)
print("\nSteady State Probability Distribution given:")
print(z)
```

```
Transition Probability Matrix:
[[0.   0.5  0.5  0.  ]
 [0.25 0.   0.5  0.25]
 [0.6  0.4  0.   0.  ]
 [0.3  0.4  0.3  0.  ]]

Steady State Probability Distribution given:
[0.29353779 0.30668127 0.32311062 0.07667032]
```

▾ 2. To find that the specific state in a Markov chain is a recurrent or transient.

```
import numpy as np
m = np.array([[1, 0 , 0 , 0, 0], [0 , 1, 0 ,0, 0] , [0 ,0 ,0.5, 0.5
              [0 , 0.5, 0, 0.5, 0] , [0.25, 0.25, 0 , 0, 0.5]])
print("Transition Probability Matrix: ")
print(m)
transient=[]
recurrent=[]
row, col = m.shape
for i in range(row):
    flag=True
    for j in range(col):
        if m[i][j]>0:
            if m[j][i]==0:
                flag=False
                transient.append(i)
                break
    if flag :
        recurrent.append(i)
print("\nTransient State:", transient)
print("Recurrent State:", recurrent)
```

```
Transition Probability Matrix:
[[1.   0.   0.   0.   0.  ]
 [0.   1.   0.   0.   0.  ]
 [0.   0.   0.5  0.5  0.  ]
 [0.   0.5  0.   0.5  0.  ]
 [0.25 0.25 0.   0.   0.5 ]]

Transient State: [2, 3, 4]
Recurrent State: [0, 1]
```

Result:

Implemented Markov Chain special cases to find steady state probabilities in case of ergodic Markov Chain and to find that the specific state in a Markov chain is a recurrent or transient.

Discussion:

The probability of transitioning to any particular state is dependent solely on the current state and time elapsed. The state space, or set of all possible states, can be anything: letters, numbers, weather conditions, baseball scores, or stock performances.

Sidharth

2K18/MC/114

Experiment 11

Objective:

Demonstrate Markov Chain Process. WAP to find Fundamental Matrix from Absorbing Markov chain. Demonstrate application of Fundamental Matrix by considering a suitable example.

Theory:

An absorbing Markov chain is a Markov chain in which every state can reach an absorbing state. An absorbing state is a state that, once entered, cannot be left.

Like general Markov chains, there can be continuous-time absorbing Markov chains with an infinite state space. However, this article concentrates on the discrete-time discrete-state-space case.

A basic property about an absorbing Markov chain is the expected number of visits to a transient state *j* starting from a transient state *i* (before being absorbed). The probability of transitioning from *i* to *j* in exactly *k* steps is the (*i*,*j*)-entry of Q^k. Summing this for all *k* (from 0 to ∞) yields the fundamental matrix, denoted by *N*. It can be proven that

$$N = \sum_{k=0}^{\infty} Q^k = (I_t - Q)^{-1},$$

where I_t is the *t*-by-*t* identity matrix. The (*i*, *j*) entry of matrix *N* is the expected number of times the chain is in state *j*, given that the chain started in state *i*. With the matrix *N* in hand, other properties of the Markov chain are easy to obtain.

Code:

```
P = [0.2 0.0 0.0 0.8 0.0 ;
     0.1 0.1 0.6 0.2 0.0 ;
     0.0 0.7 0.0 0.3 0.0 ;
     0.2 0.0 0.2 0.1 0.5 ;
     0.0 1.0 0.0 0.0 0.0];
Q = [0.1 0.6 0.0; 0.4 0.0 0.5; 0.0 0.2 0.0];
R = [0.1 0.0; 0.0 0.0; 0.2 0.6];
I = [1 0 0; 0 1 0; 0 0 1];
fprintf('Fundamental Matrix: \n\n');
N = (I - Q)^-1;
disp(N);
U=N*R;
fprintf('\n\nAbsorbed Probabilities: \n\n');
disp(U);
```

Output:

```
Fundamental Matrix:

   1.5789   1.0526   0.5263
   0.7018   1.5789   0.7895
   0.1404   0.3158   1.1579


Absorbed Probabilities:

   0.2632   0.3158
   0.2281   0.4737
   0.2456   0.6947
```

Result: fundamental matrix is obtained for absorbing Markov chain.

Discussion: A state i is an absorbing state if once the system reaches state i, it stays in that state, that is $P_{ii} = 1$. The sum of the entries of a row of the fundamental matrix gives us the expected number of steps before absorption for the non-absorbing state associated with that row.

Author:

- **Sidharth**
- *2K18/ MC/ 114*

Objective:

Demonstrate M/M/1 Queuing Model using a suitable example.

Theory:

In queueing theory, a discipline within the mathematical theory of probability, an M/M/1 queue represents the queue length in a system having a single server, where arrivals are determined by a Poisson process and job service times have an exponential distribution. An M/M/1 queue is a stochastic process whose state space is the set {0,1,2,3,...} where the value corresponds to the number of customers in the system, including any currently in service.

- Arrivals occur at rate λ according to a Poisson process and move the process from state i to i + 1.
- Service times have an exponential distribution with rate parameter μ in the M/M/1 queue, where 1/μ is the mean service time.
- A single server serves customers one at a time from the front of the queue, according to a first-come, first-served discipline.
- The buffer is of infinite size, so there is no limit on the number of customers it can contain.

> Consider an Airport runway for arrivals only, arriving aircraft join a single queue for the runway,
>
> Exponentially distributed service time with a rate, μ = 27 arrivals/hour
>
> Poisson arrivals with a rate, λ = 20 arrivals/hour

Code & Output:

```
lamb = 20 # Poisson arrival rate
mu = 27 # Service rate

ls = lamb/(mu - lamb)
lq = lamb**2/(mu*(mu-lamb))
ws = ls/lamb
wq = lq/lamb

print('The Expected number of aircraft on the system:', ls)
print('The Expected number of aircraft on the runway:', lq)
print('The Average time spent on the system:', ws*60, 'minutes')
print('The Average time spent on the runway:', wq*60, 'minutes')
```

```
The Expected number of aircraft on the system: 2.857142857142857
The Expected number of aircraft on the runway: 2.1164021164021163
The Average time spent on the system: 8.571428571428571 minutes
The Average time spent on the runway: 6.349206349206349 minutes
```

Result:

Demonstrated MM1 Queuing Model using a airport runway for arrivals only.

Discussion:

The M/M/1 system is made of a Poisson arrival, one exponential (Poisson) server, queue of unlimited capacity and unlimited customer population. These assumptions are very strong, not satisfied for practical systems. Nevertheless the M/M/1 model shows clearly the basic ideas and methods of Queuing Theory.