DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi 110042

Department Of Applied Mathematics

MC-312: ARTIFICIAL INTELLIGENCE
Lab File

Submitted To:
Prof. LN Das
Ms. Anjali
Department of Applied Mathematics

Submitted By:
Ishan Bhateja
2K20/MC/061

MISSION OF THE UNIVERSITY

1. To establish Centres of excellence in emerging areas of science, engineering, technology, management and allied areas.
2. To foster an ecosystem for incubation, product development, transfer of technology and entrepreneurship.
3. To create environment of collaboration, experimentation, imagination and creativity.
4. To develop human potential with analytical abilities, ethics and integrity.
5. To provide environment friendly, reasonable and sustainable solutions for local and global needs.

MISSION OF THE DEPARTMENT

1. To achieve academic excellence through innovative teaching and learning practices.
2. To improve the research competence to address social needs.
3. To inculcate a culture that supports and reinforces ethical, professional behaviors for a harmonious and prosperous society.
4. Strive to make students to understand, appreciate and gain mathematical skills and develop logic, so that they are able to contribute intelligently in decision making which characterizes our scientific and technological age.

CONTENTS

PRACTICAL 1

AIM: Write a program to solve the 8-Puzzle problem using Generate and Test Strategy.
(as discussed in the class, given a current state, generate all the possible next states, also known as neighbouring states, and check whether any one of them is final state or not, if not, repeat the procedure with neighbouring states).

Current State:

8	1	3
4		5
2	7	6

Final State:

1	2	3
8		4
7	6	5

CODE:

```
#include <bits/stdc++.h>
using namespace std;
#define vvi vector<vector<int>>
#define endl "\n"

struct puzzle{
   int loc0x;
   int loc0y;
   vvi currState;
   vector<vvi> movesuntilnow;
   puzzle(){};
   puzzle(vvi &currState){
      this->currState = currState;
      getloc0();
   }
   void getloc0()
   {
      for (int i = 0; i < 3; i++)
      {
         for (int j = 0; j < 3; j++)
         {
            if (!currState[i][j])
            {
               loc0x = i;
```

```
                loc0y = j;
                return;
            }
        }
    }
};

set<vvi> visited;
void inputPuzzle(struct puzzle &p){
    for (int i = 0; i < 3; i++){
        vector<int> temp;
        for (int j = 0; j < 3; j++){
            int x;
            cin >> x;
            temp.push_back(x);
        }
        p.currState.push_back(temp);
    }
    p.getloc0();
    visited.insert(p.currState);
}

void displayPuzzle(struct puzzle &p){
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 3; j++)
            cout << p.currState[i][j] << " ";
        cout << endl;
    }
}

void displayMovesTillNow(struct puzzle &p){
    for (int k = 0; k < p.movesuntilnow.size(); ++k){
        cout << "\nMove " << k + 1 << endl;
        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 3; j++)
                cout << p.movesuntilnow[k][i][j] << " ";
            cout << endl;
        }
    }
}

vector<vvi> generateMoves(struct puzzle &final){
```

```
    vector<vvi> moves;
    vvi temp = final.currState;
    int x = final.loc0x, y = final.loc0y;
    if (x != 0){
        temp[x][y] = temp[x - 1][y];
        temp[x - 1][y] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    if (x != 2){
        temp[x][y] = temp[x + 1][y];
        temp[x + 1][y] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    if (y != 0){
        temp[x][y] = temp[x][y - 1];
        temp[x][y - 1] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    if (y != 2){
        temp[x][y] = temp[x][y + 1];
        temp[x][y + 1] = 0;
        moves.push_back(temp);
        temp = final.currState;
    }
    return moves;
}

void solvePuzzle(struct puzzle &initial, struct puzzle &final){
    queue<puzzle> q;
    q.push(initial);
    while (!q.empty()){
        puzzle curr = q.front();
        q.pop();
        vector<vvi> moves = generateMoves(curr);
        for (int i = 0; i < moves.size(); i++){
            if (moves[i] == final.currState){
                curr.movesuntilnow.push_back(moves[i]);
                displayMovesTillNow(curr);
                cout << "Total number of steps to reach the final state:
                    "<<curr.movesuntilnow.size()<<endl;
```

```
                return;
            }
            if (visited.find(moves[i]) == visited.end()){
                puzzle temp(moves[i]);
                temp.movesuntilnow = curr.movesuntilnow;
                temp.movesuntilnow.push_back(moves[i]);
                visited.insert(moves[i]);
                q.push(temp);
            }
        }
    }
}

int main(){
    cout << endl;
    puzzle initial;
    cout << "Enter the initial puzzle state (Denote blank space using 0):" << endl;
    inputPuzzle(initial);
    cout << endl;
    puzzle final;
    cout << "Enter the final puzzle state (Denote blank space using 0):" << endl;
    inputPuzzle(final);
    cout << endl;
    cout << "The initial puzzle state is: " << endl;
    displayPuzzle(initial);
    cout << endl;
    solvePuzzle(initial, final);
    return 0;
}
```

OUTPUT:

ISHAN BHATEJA 2K20/MC/61

PRACTICAL 2

AIM: Demonstration of Stochastic Process with Continuous Index Set
(a) Discrete State Space (b) Continuous State Space

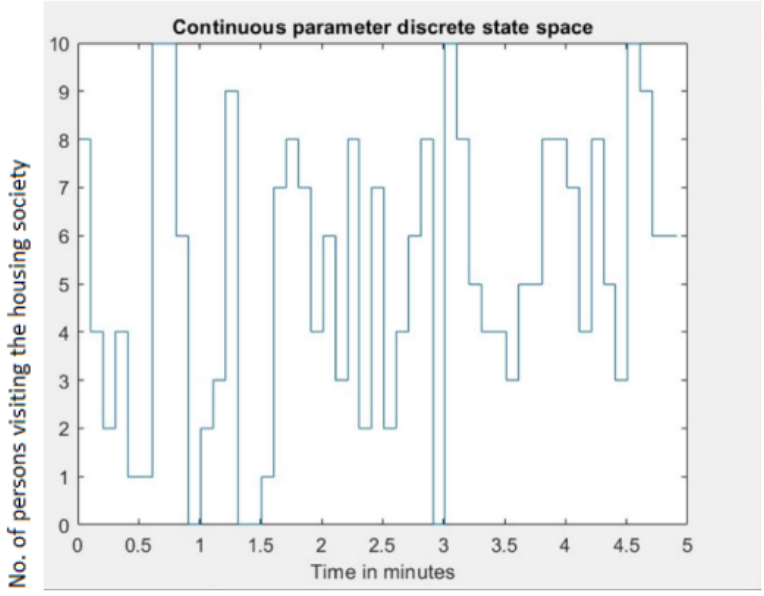
THEORY: A stochastic process is a family of random variables {X(t), t ∈ T} defined on a given probability space, indexed by parameter t, where t varies over an index set T. The values assumed by X(t) are called its states, and the set of all possible values forms the state pace of the process. Stochastic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

(a) Continuous Parameter Stochastic Chain
Eg: Number of persons inside a housing society at any time during the visiting hours.
Considering the time frame to be 5 minutes in the program.

CODE:

```
x = [0.01:0.5:5];
y = randi([0 10],50,1);
p = stairs(x,y);
xlabel('Time in minutes');
ylabel('No. of persons visiting the housing society');
title('Continuous Parameter - Discrete State Space');
```

OUTPUT:

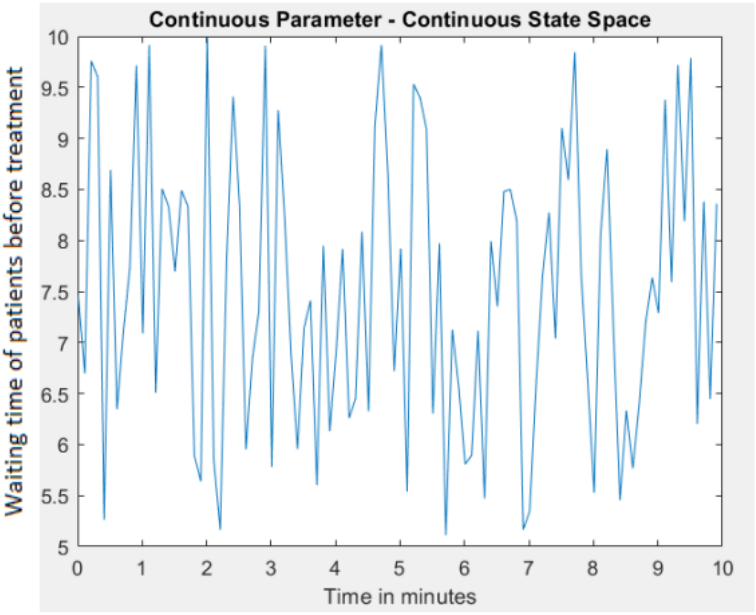


(b) Continuous Parameter Continuous State
Eg: Waiting time of a patient arriving at a doctor's clinic at any time until the treatment begins.
Assuming each patient waits at least 5 minutes before treatment

CODE:

```
x = [0.01:0.1:10];
y = 5 + 5.*rand(100,1);
p = plot(x, y);
xlabel("Time in minutes");
ylabel("Waiting time of patients before treatment");
title("Continuous parameter continuous state space");
```

OUTPUT:



RESULT:
The above graphs clearly show the continuous nature of the index set. The state space used is not countable in the Continuous index set, and the state space is discrete in the Continuous Parameter discrete-state process.

PRACTICAL 3

AIM: Demonstrating Bernoulli process. Specifically, WAP to find the probability that in the case of a Bernoulli process, where
(a) out of n trials k are successes, (b) kth success occurs at the nth trial.
For example, A bulb company produces bulbs, and the effectiveness of a manufactured product is as per Bernoulli distribution with the probability of a specific bulb being defective as 0.10, independent of others.

1. Find the probability of getting 3 working bulbs from a random sample of 6 toys.
2. Find the probability that the fifth working bulb is encountered on the 9th bulb examination.

THEORY:
The Bernoulli Process is an example of a discrete-parameter state process. Suppose Xi, i=1, 2,...,n are independent and identically distributed Bernoulli random variables, each with probability p of success, that is, P{Xi =1} = p, and probability 1-p of failure, that is, P{Xi=0} = 1 – p. Let Sn = X1 + X2 +...+Xn be the number of successes in n Bernoulli trials. Then {Sn, n =1,2,...} is called Bernoulli process with state space {0, 1, 2, ...}, and so is a discrete-parameter discrete-state process. For each n, Sn has a binomial distribution with parameters n and p, and $P\{Sn=k\} = nCk*p^k*(1-p)^{n-k}$, k =0,1,2,...n. Starting at any particular Bernoulli trial, the number of failures F before the next success has a geometric distribution, that is, $P(F=k) = (1-p)^k p$, k=0, 1, 2, ... A negative binomial distribution is when we encounter kth success at nth trial, i.e it implies that previous n-1 trials have seen exactly k-1 successes. Each case of the Bernoulli variate Xi has the distinct parameter pi then the Bernoulli process is called a non-homogenous Bernoulli process. For the purpose of this program, I have used MATLAB and have taken the above-mentioned problem.

CODE:

1. out of n trials k are successes

```
function[] = bernoulli()
        q = 0.10;
        p = 1-q;
        n = 6;
        x = 3;
        prob = nchoosek(n, x)*(p ^ x) * (q ^ (n - x));
        disp(prob)
end
```

2. kth success occurs at the nth trial

```
function[] = bernoulli()
        q = 0.10;
        p = 1 - q;
        n = 8;
        x = 4;
        prob = nchoosek(n, x)*(p ^ x) * p * (q ^ (n - x));
        disp(prob)
end
```

OUTPUT:

1. bernoulli >> 0.01458
2. bernoulli >> 0.00413343

RESULT:

With the help of the above program, we have successfully managed to solve a Bernoulli process using MATLAB, both for simple as well as a negative binomial distribution.

An important part of every Bernoulli trial is that each action must be independent. That means the probabilities must remain the same throughout the trials; each event must be completely separate and have nothing to do with the previous event. The Bernoulli process has immense application in various fields. It is a building block for various discrete distributions like binomial, geometric etc.

PRACTICAL 4

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

THEORY:

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

1. $N(0) = 0$
2. Independent increments (the number of occurrences counted in disjoint intervals are independent)
3. Stationary increments (the probability distribution of the number of occurrences counted in any time interval only depends on the length of the interval.
4. The probability distribution of N(t) is a Poisson distribution.
5. No counted occurrences are simultaneous.

CODE:

1. Homogenous

```
function[p] = poisson_homo(parameter, n)
    p = 0;
    for i = 0:n
        pr = pr + ((exp(-parameter) * (parameter) ^ i) / factorial(i));
    end
end
```

OUTPUT:

```
>>poisson(4, 10)
    ans = 0.9972
```

2. Non Homogenous

```
function[p] = poisson_non_homo(parameter, n)
    p = 0;
    for i = 0:n
        pr = pr + ((exp(-parameter / i) * (parameter / i) ^ i) / factorial(i));
    end
end
```

OUTPUT:

```
>>poisson(1, 10)
    ans = 0.4482
```

RESULT:

With the help of the above program, we have successfully managed to solve a Poisson process using MATLAB, both for homogenous as well as non-homogeneous Poisson process. As observed, the non-homogenous process has uncertainty, as can be seen with the help of the low probability