KADI SARVA VISHWAVIDYALAYA
LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH
GANDHINAGAR

Department of
Information Technology

Subject: Soft Computing (CT605A-N)

Laboratory Manual

Prepared By

Patel Joy J.

19BEIT30033

6IT

LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH

GANDHINAGAR

DEPARTMENT OF COMPUTER ENGINEERING
&
INFORMATION TECHNOLOGY

CERTIFICATE

Mr.______________________________of 6th IT Enrollment No.______________ Exam No, _____________________has satisfactorily completed his term work in ***Soft Computing (CT605A-N)*** for the term ending in **Dec-May-2022**.

Date: _______________

_________________ Dr. Mehul Barot

Subject Faculty **HOD-IT**

INDEX

Practical No	Name of Experiment	Page	Date of Submission	Sign
1	Create a perceptron with appropriate no. of inputs and outputs. Train it using fixed increment learning algorithm until no change in weights is required. Output the final weights.		21-01-21	
2	Create a simple ADALINE network with appropriate no. of input and output nodes. Train it using delta learning rule until no change in weights is required. Output the final weights.			
3	Train the autocorrelator by given patterns: A1=(-1,1,-1,1), A2=(1,1,1,-1), A3=(-1, -1, - 1, 1). Test it using patterns: Ax=(-1,1,-1,1), Ay=(1,1,1,1), Az=(-1,-1,-1,-1).			
4	Train the hetrocorrelator using multiple training encoding strategy for given patterns: A1=(000111001) B1=(010000111), A2=(111001110) B2=(100000001), A3=(110110101) B3(101001010). Test it using pattern A2.			
5	Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relation by Cartesian product of any two fuzzy sets and perform maxmin composition on any two fuzzy relations.			
6	Solve Greg Viot's fuzzy cruise controller using MATLAB Fuzzy logic toolbox.			
7	Solve Air Conditioner Controller using MATLAB Fuzzy logic toolbox			
8	Implement TSP using GA.			

PRACTICAL-1

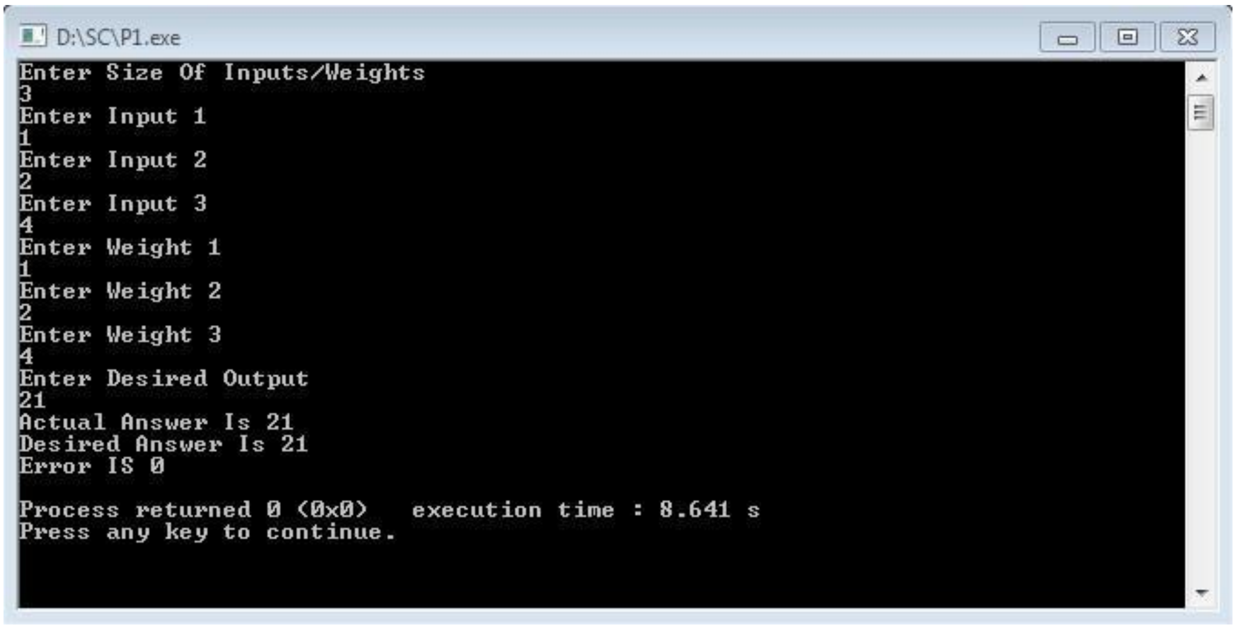
AIM: Create a perceptron with appropriate no. of inputs and outputs. Train it using fixed increment learning algorithm until no change in weights is required. Output the final weights.

CODE:

1. C++

```
#include<iostream>
using namespace std;
int main()
{
   int n,i,j;
   cout<<"Enter Size Of Inputs/Weights "<<endl;
   cin>>n;
   int a[n],w[n],ans=0,desired,t,e,token=2;
   for (i=0;i<n;i++)
   {
      cout<<"Enter Input "<<i+1<<" "<<endl;
      cin>>a[i];
   }
   for (i=0;i<n;i++)
   {
      cout<<"Enter Weight "<<i+1<<" "<<endl;
      cin>>w[i];
   }
   cout<<"Enter Desired Output "<<endl;;
   cin>>desired;
   while(token!=1 && e!=0)
   {
      ans=0;
      for (i=0;i<n;i++)
      {
         ans=ans+a[i]*w[i];
      }
      cout<<"Actual Answer Is "<<ans<<endl;
      cout<<"Desired Answer Is "<<desired<<endl;
      e=desired-ans;
      cout<<"Error IS "<<e<<endl;
      if(e==0)
      {
         break;
      }
      cout<<"Enter 1 To Terminate"<<endl;;
      cin>>token;
      if(e<0)
      {
        for (i=0;i<n;i++)
         {
            w[i]=w[i]-1;
         }
      }
      else
      {
            for (i=0;i<n;i++)
  {
                w[i]=w[i]+1;
            }
        }
    }
    return 0;
}
```

OUTPUT:



```
Enter Size Of Inputs/Weights
3
Enter Input 1
1
Enter Input 2
2
Enter Input 3
4
Enter Weight 1
1
Enter Weight 2
2
Enter Weight 3
4
Enter Desired Output
21
Actual Answer Is 21
Desired Answer Is 21
Error IS 0

Process returned 0 (0x0)   execution time : 8.641 s
Press any key to continue.
```

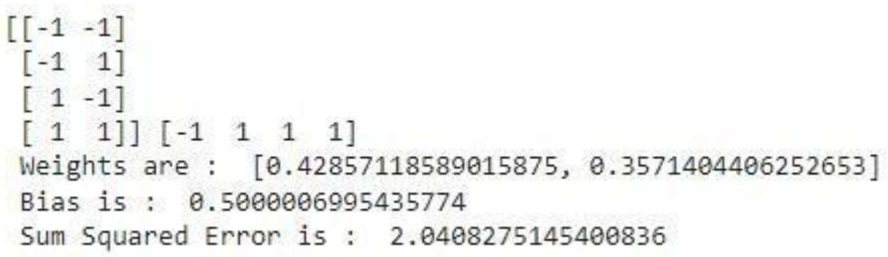
PRACTICAL-2

AIM: Create a simple ADALINE network with appropriate no. of input and output nodes. Train it using delta learning rule until no change in weights is required. Output the final weights

CODE:

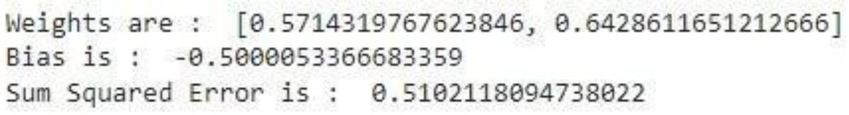
```
import numpy as np
inputs = np.array([ [-1, -1], [-1, 1],[1, -1],[1, 1]])
outputs = np.array([-1, 1, 1, 1])

print(inputs, outputs)
weight=[1,1]bias=0 errors=[]
learning_rate=0.2epochs=10

for i in range(epochs): sum_squared_error = 0
for j in range(inputs.shape[0]):
actual = outputs[j]x1 = inputs[j][0] x2 = inputs[j][1]
unit = (x1 * weight[0]) + (x2 * weight[1]) + biaserror = actual - unit
#print("error =", error) sum_squared_error += error * error weight[0] += learning_rate * error * x1weight[1]
+= learning_rate * error * x2bias += learning_rate * error

#print("sum of squared error = ", sum_squared_error/4, "\n\n")print(" Weights are : ",weight)
print(" Bias is : ",bias)

print(" Sum Squared Error is : ",sum_squared_error/4)

#AND Gate

outputs = np.array([-1, -1, -1, 1])


weight=[1,1]bias=0 errors=[]
learning_rate=0.2epochs=10

for i in range(epochs): sum_squared_error = 0
for j in range(inputs.shape[0]):actual = outputs[j]
x1 = inputs[j][0]x2 = inputs[j][1]
unit = (x1 * weight[0]) + (x2 * weight[1]) + biaserror = actual - unit
#print("error =", error) sum_squared_error += error * error weight[0] += learning_rate * error * x1weight[1]
+= learning_rate * error * x2bias += learning_rate * error
#print("sum of squared error = ", sum_squared_error/4, "\n\n")print(" Weights are : ",weight)
print(" Bias is : ",bias)

print(" Sum Squared Error is : ",sum_squared_error/4)
```

OUTPUT:



```
[[-1 -1]
 [-1  1]
 [ 1 -1]
 [ 1  1]] [-1  1  1  1]
 Weights are :  [0.42857118589015875, 0.3571404406252653]
 Bias is :  0.5000006995435774
 Sum Squared Error is :  2.0408275145400836
```

AND GATE



```
Weights are :  [0.5714319767623846, 0.6428611651212666]
Bias is :  -0.5000053366683359
Sum Squared Error is :  0.5102118094738022
```

PRACTICAL-3

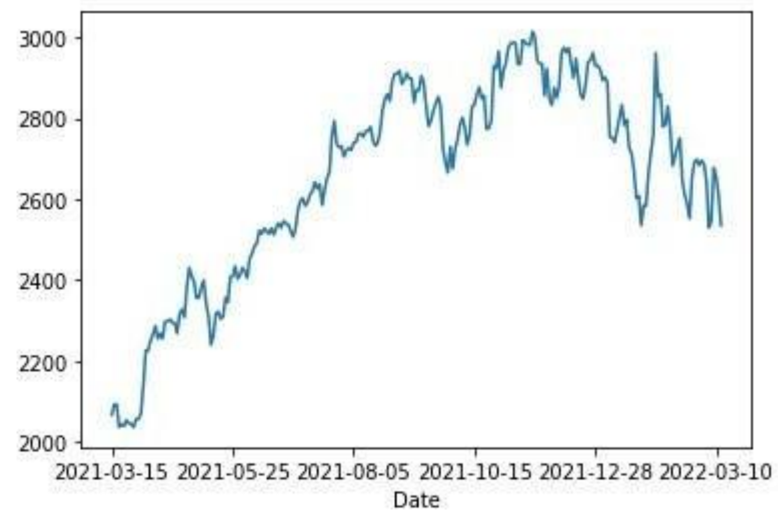
AIM: Train the autocorrelator by given patterns: A1=(-1,1,-1,1), A2=(1,1,1,-1), A3=(-1, -1, -1, 1). Test it using patterns: Ax=(-1,1,-1,1), Ay=(1,1,1,1), Az=(-1,-1,-1,-1).

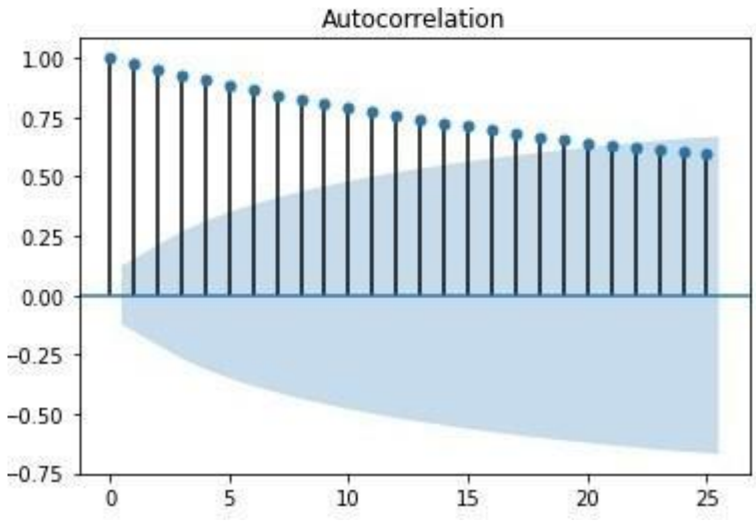
CODE:

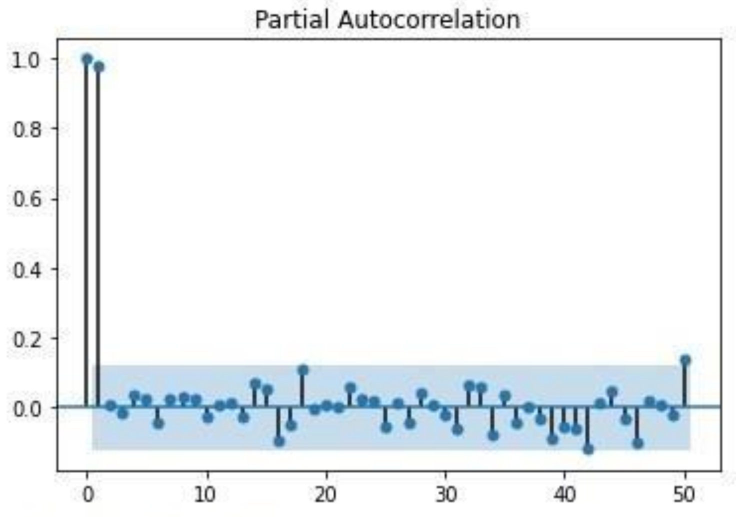
1.Python

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.stats.stattools import durbin_watson
from statsmodels.regression.linear_model import OLS
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf goog_stock_Data =
pd.read_csv('GOOG.csv', header=0, index_col=0) goog_stock_Data['Adj Close'].plot()
plt.show()
plot_acf(goog_stock_Data['Adj Close'], alpha =0.05)
plt.show()
plot_pacf(goog_stock_Data['Adj Close'], alpha =0.05, lags=50) plt.show()
df = pd.DataFrame(goog_stock_Data,columns=['Date','Adj Close']) X
=np.arange(len(df[['Adj Close']]))
Y = np.asarray(df[['Adj Close']]) X
= sm.add_constant(X)
ols_res = OLS(Y,X).fit()
durbin_watson(ols_res.resid)
```

OUTPUT:







```
0.057744198145846186
```

PRACTICAL-4

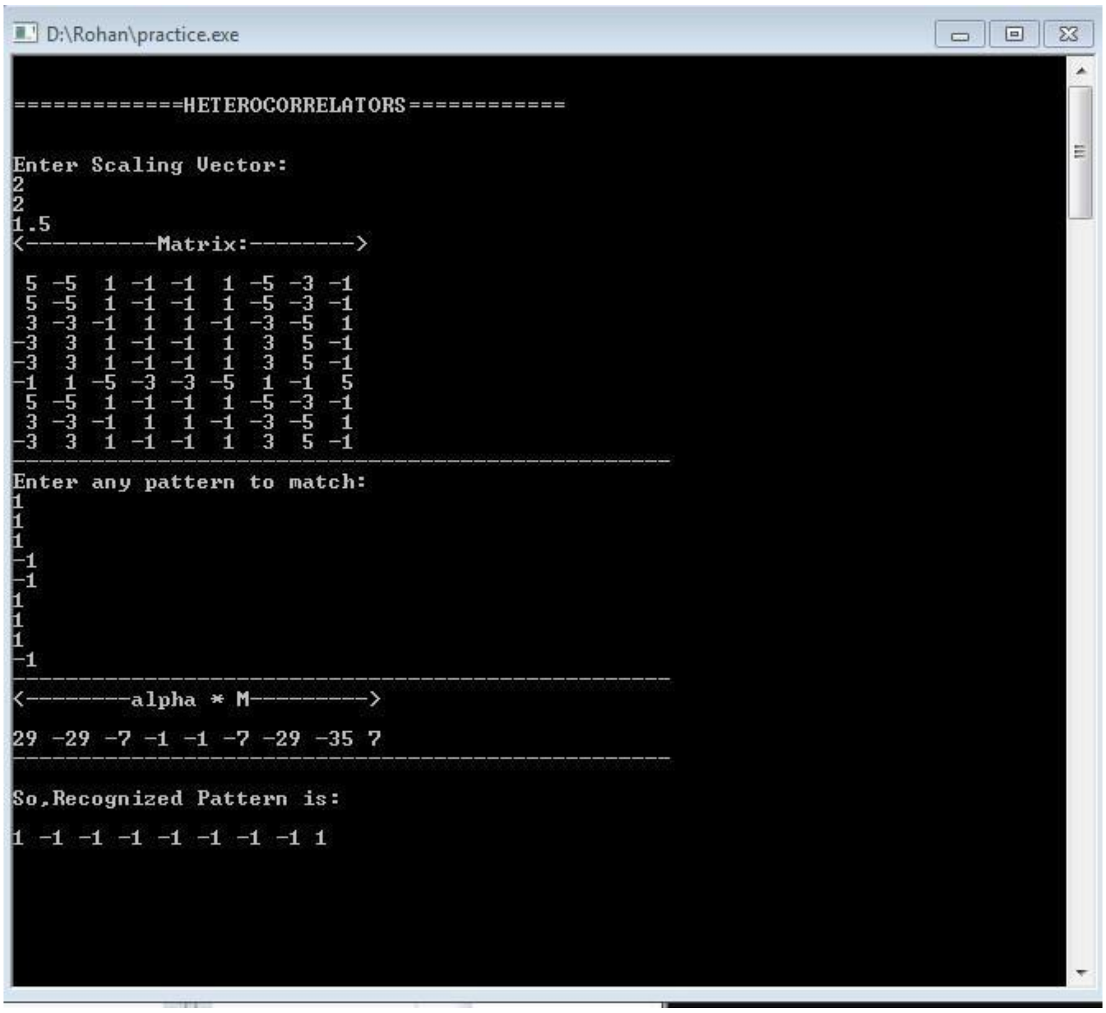
AIM : Train the hetrocorrelator using multiple training encoding strategy for given patterns: A1=(000111001) B1=(010000111), A2=(111001110) B2=(100000001), A3=(110110101) B3(101001010). Test it using pattern A2.

CODE:

1.C

```
#include<stdio.h>
#include<conio.h>
int x1[1][9],x2[1][9],x3[1][9],p[1][9],pp[1][9],T[9][9]={0},s=0;
int y1[1][9],y2[1][9],y3[1][9],Hp[1][9],temp;
float q[3];
int i,j;
void printline();
x1[1][9]={-1,-1,-1,1,1,1,-1,-1,1};
x2[1][9]={1,1,1,-1,-1,1,1,1,-1};
x3[1][9]={1,1,-1,1,1,-1,1,-1,1};
y1[1][9]={-1,1,-1,-1,-1,-1,1,1,1};
y2[1][9]={1,-1,-1,-1,-1,-1,-1,-1,1};
y3[1][9]={1,-1,1,-1,-1,1,-1,1,-1};
void main()
{
printf("\n\n=============HETEROCORRELATORS============\n\n\n");
printf("Enter Scaling Vector:\n");
for(i=0;i<3;i++)
scanf("%f",&q[i]);
for(i=0;i<=8;i++)
{
 for(j=0;j<=8;j++)
 {
 temp=q[0]*x1[0][i];
 T[i][j]=T[i][j]+temp*y1[0][j];
 }}
for(i=0;i<=8;i++)
{
 for(j=0;j<=8;j++)
 {
 temp=q[1]*x2[0][i];
 T[i][j]=T[i][j]+temp*y2[0][j];
 }}
for(i=0;i<=8;i++)
{
 for(j=0;j<=8;j++)
 {
      temp=q[2]*x3[0][i];
 T[i][j]=T[i][j]+temp*y3[0][j];
 }}
printf("<----------Matrix-------- >\n\n");
for(i=0;i<=8;i++)
 {
 for(j=0;j<=8;j++)
 {
 if(T[i][j]>=0)
 printf(" %d ",T[i][j]);
 else
 printf("%d ",T[i][j]);
 }
 printf("\n");
}
printline();
printf("Enter any pattern to match:\n");
for(j=0;j<9;j++)
{
 scanf("%d",&p[0][j]);
}
for(i=0;i<=8;i++)
{
 for(j=0;j<=8;j++)
 {
// printf("%d\n",T[0][j]);
 s=s+T[j][i]*p[0][j];
 }
 pp[0][i]=s;
 if(s>0)
 Hp[0][i]=1;
 if(s==0)
 Hp[0][i]=p[0][i];
 if(s<0)
 Hp[0][i]=-1;
 s=0;
}
printline();
printf("<--------alpha * M-------- >\n\n");
for(j=0;j<9;j++)
printf("%d ",pp[0][j]);
printf("\n");
printline();
printf("\nSo,Recognized Pattern is:\n\n");
for(j=0;j<9;j++)
    printf("%d ",Hp[0][j]);
printf("\n");
getch();
}
void printline()
{
int p;
for(p=0;p<25;p++)
printf("--");
printf("\n");
}
```

OUTPUT:



```
=============HETEROCORRELATORS============

Enter Scaling Vector:
2
2
1.5
<----------Matrix:-------->

 5 -5  1 -1 -1  1 -5 -3 -1
 5 -5  1 -1 -1  1 -5 -3 -1
 3 -3 -1  1  1 -1 -3 -5  1
-3  3  1 -1 -1  1  3  5 -1
-3  3  1 -1 -1  1  3  5 -1
-1  1 -5 -3 -3 -5  1 -1  5
 5 -5  1 -1 -1  1 -5 -3 -1
 3 -3 -1  1  1 -1 -3 -5  1
-3  3  1 -1 -1  1  3  5 -1
--------------------------------------------------
Enter any pattern to match:
1
1
1
-1
-1
1
1
1
-1
--------------------------------------------------
<--------alpha * M--------->

29 -29 -7 -1 -1 -7 -29 -35 7
--------------------------------------------------

So,Recognized Pattern is:

1 -1 -1 -1 -1 -1 -1 -1 1
```

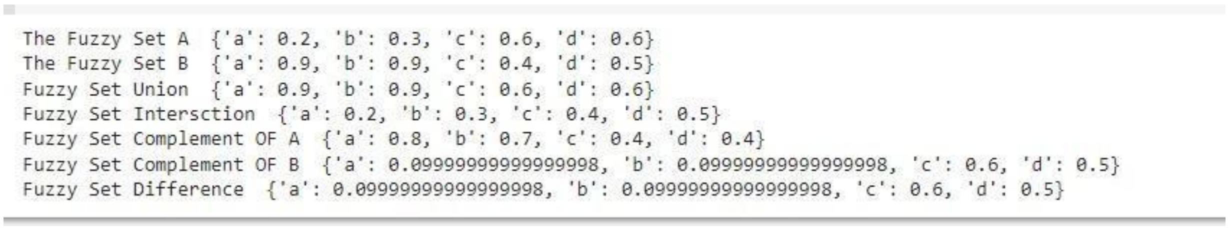
PRACTICAL-5

AIM: Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relation by Cartesian product of any two fuzzy sets and perform maxmin composition on any two fuzzy relations

CODE:

```
a=dict() b=dict() c=dict()
a={'a':0.2,'b':0.3,'c':0.6,'d':0.6,}
b={'a':0.9,'b':0.9,'c':0.4,'d':0.5,}
print('The Fuzzy Set A ',a)
print('The Fuzzy Set B ',b)
for a_key,b_key in zip(a,b):
  a_value=a[a_key]
  b_value=b[b_key]
  if a_value>b_value:
    c[a_key]=a_value
  else:
    c[b_key]=b_value
print('Fuzzy Set Union ',c)

for a_key,b_key in zip(a,b):
  a_value=a[a_key]
  b_value=b[b_key]
  if a_value<b_value:
    c[a_key]=a_value
  else:
    c[b_key]=b_value
print('Fuzzy Set Intersction ',c)

for a_key in a:
  c[a_key]=1-a[a_key]
print('Fuzzy Set Complement OF A ',c)

for b_key in b:
  c[b_key]=1-b[b_key]
print('Fuzzy Set Complement OF B ',c)

for a_key,b_key in zip(a,b):
  a_value=a[a_key]
  b_value=b[b_key]
  b_value=1-b_value if
  a_value<b_value:
    c[a_key]=a_value
  else:
    c[b_key]=b_value print('Fuzzy
Set Difference ',c)
```

OUTPUT:



```
The Fuzzy Set A  {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Fuzzy Set B  {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
Fuzzy Set Union  {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}
Fuzzy Set Intersction  {'a': 0.2, 'b': 0.3, 'c': 0.4, 'd': 0.5}
Fuzzy Set Complement OF A  {'a': 0.8, 'b': 0.7, 'c': 0.4, 'd': 0.4}
Fuzzy Set Complement OF B  {'a': 0.09999999999999998, 'b': 0.09999999999999998, 'c': 0.6, 'd': 0.5}
Fuzzy Set Difference  {'a': 0.09999999999999998, 'b': 0.09999999999999998, 'c': 0.6, 'd': 0.5}
```

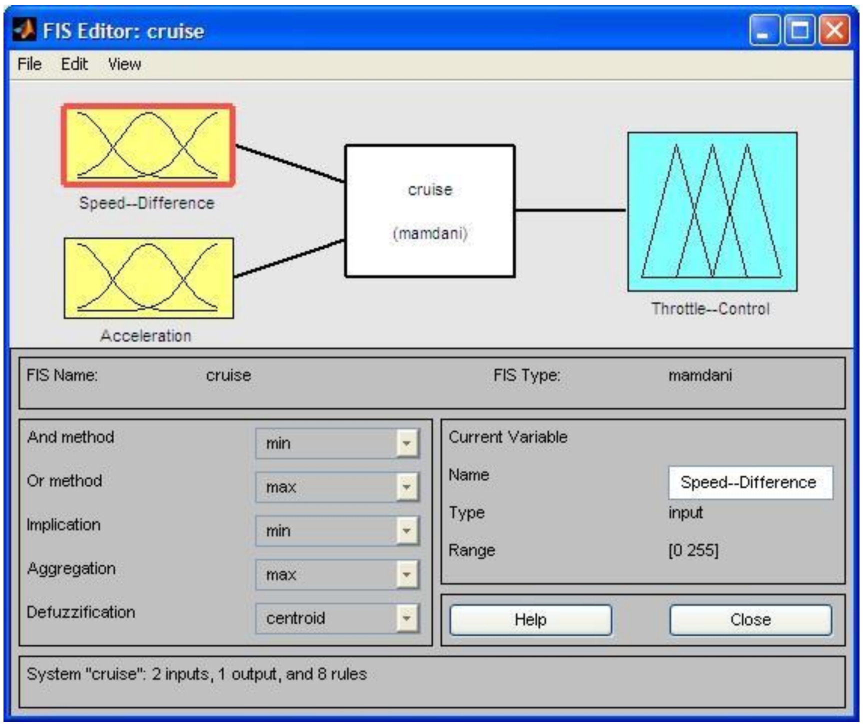
PRACTICAL-6

AIM:Solve Greg Viot's fuzzy cruise controller using MATLAB Fuzzy logic toolbox

CODE:

```
[System]
Name='Controller'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=3
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'
[Input1]
Name='DT'
Range=[-5 5]
NumMFs=3
MF1='PS':'trapmf',[-4 1.5 3.5 5]
MF2='PL':'trapmf',[0 0.8 5 5]
MF3='ZE':'trimf',[-5 0 5]
[Input2]
Name='dDT/dt'
Range=[-3 3]
NumMFs=3
MF1='PL':'trapmf',[0.6 1.5 3 3]
MF2='ZE':'trimf',[-3 0 3]
MF3='NS':'trimf',[-2.7 -0.5 1.9]
[Output1]
Name='Dial--Turn'
Range=[-1 1]
NumMFs=3
MF1='NL':'trapmf',[-1 -1 -0.5 -0.2]
MF2='NM':'trimf',[-1 -0.5 0]
MF3='ZE':'trimf',[-1 0 1]
[Rules]
3 1, 1 (1) : 1
2 2, 2 (1) : 1
1 3, 3 (1) : 1
```

OUTPUT:

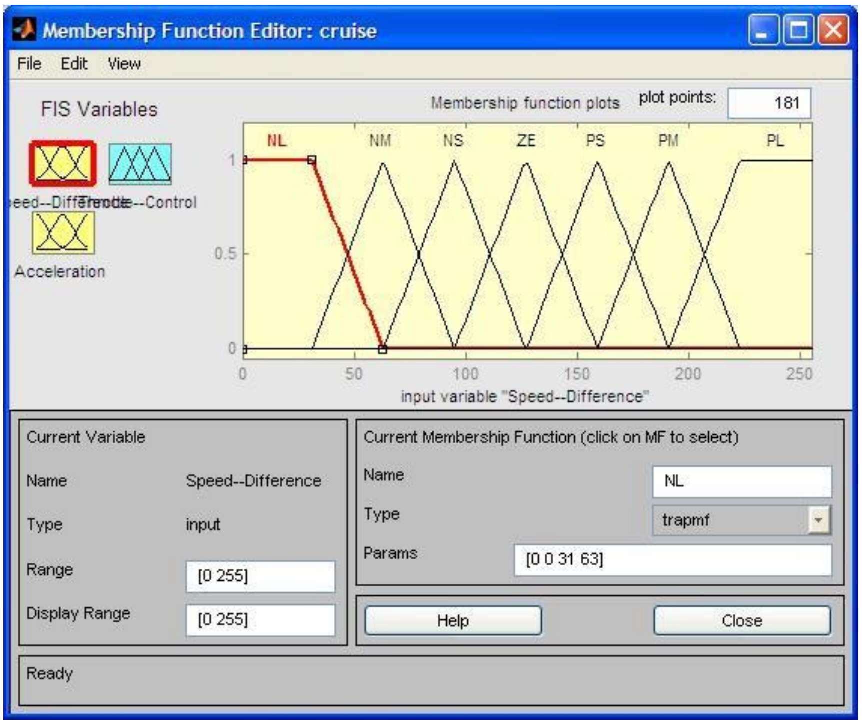


FIS Name: cruise FIS Type: mamdani

And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid

Current Variable
Name: Speed--Difference
Type: input
Range: [0 255]

System "cruise": 2 inputs, 1 output, and 8 rules



plot points: 181

Current Variable

Name Speed--Difference

Type input

Range [0 255]

Display Range [0 255]

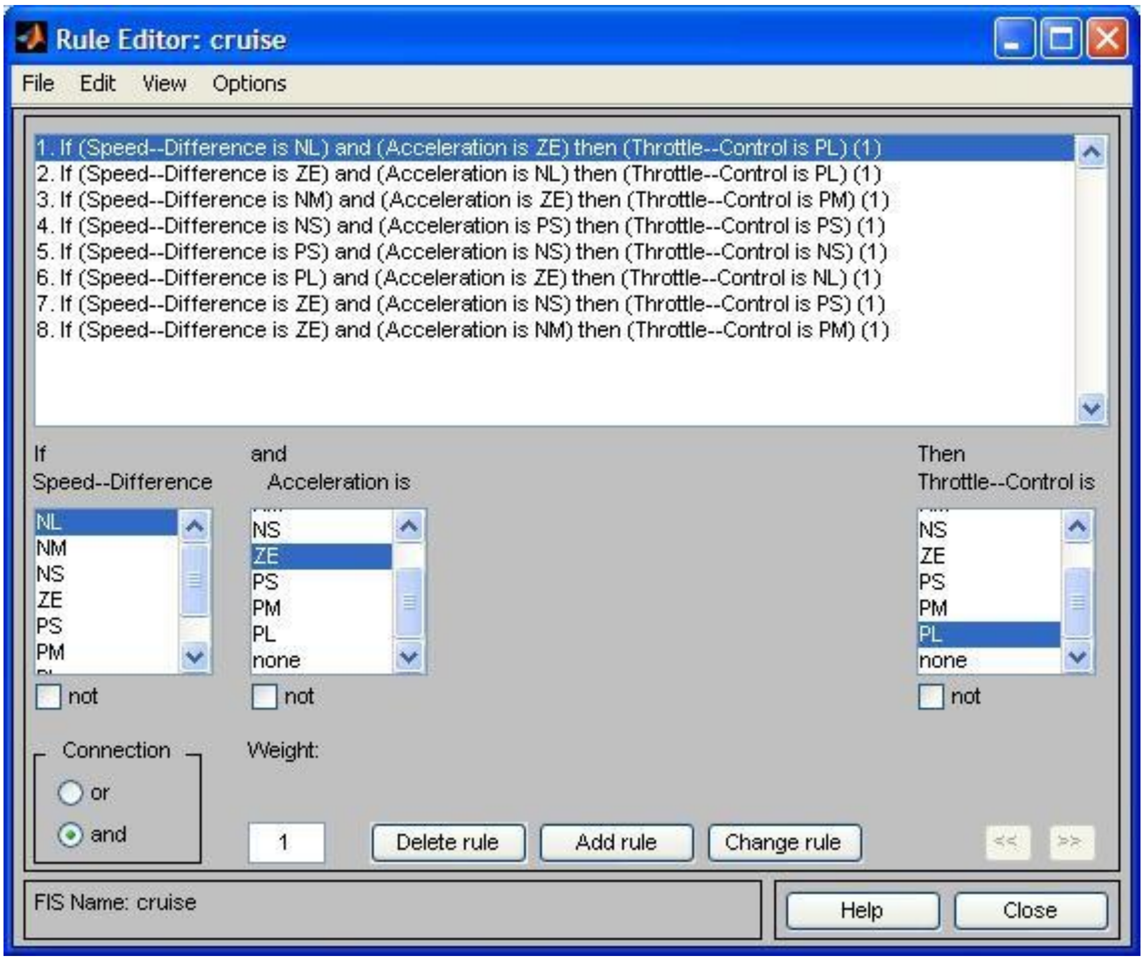
Current Membership Function (click on MF to select)

Name NL

Type trapmf

Params [0 0 31 63]

Ready



1. If (Speed--Difference is NL) and (Acceleration is ZE) then (Throttle--Control is PL) (1)
2. If (Speed--Difference is ZE) and (Acceleration is NL) then (Throttle--Control is PL) (1)
3. If (Speed--Difference is NM) and (Acceleration is ZE) then (Throttle--Control is PM) (1)
4. If (Speed--Difference is NS) and (Acceleration is PS) then (Throttle--Control is PS) (1)
5. If (Speed--Difference is PS) and (Acceleration is NS) then (Throttle--Control is NS) (1)
6. If (Speed--Difference is PL) and (Acceleration is ZE) then (Throttle--Control is NL) (1)
7. If (Speed--Difference is ZE) and (Acceleration is NS) then (Throttle--Control is PS) (1)
8. If (Speed--Difference is ZE) and (Acceleration is NM) then (Throttle--Control is PM) (1)

If Speed--Difference: [✓ NL] NM NS ZE PS PM — not

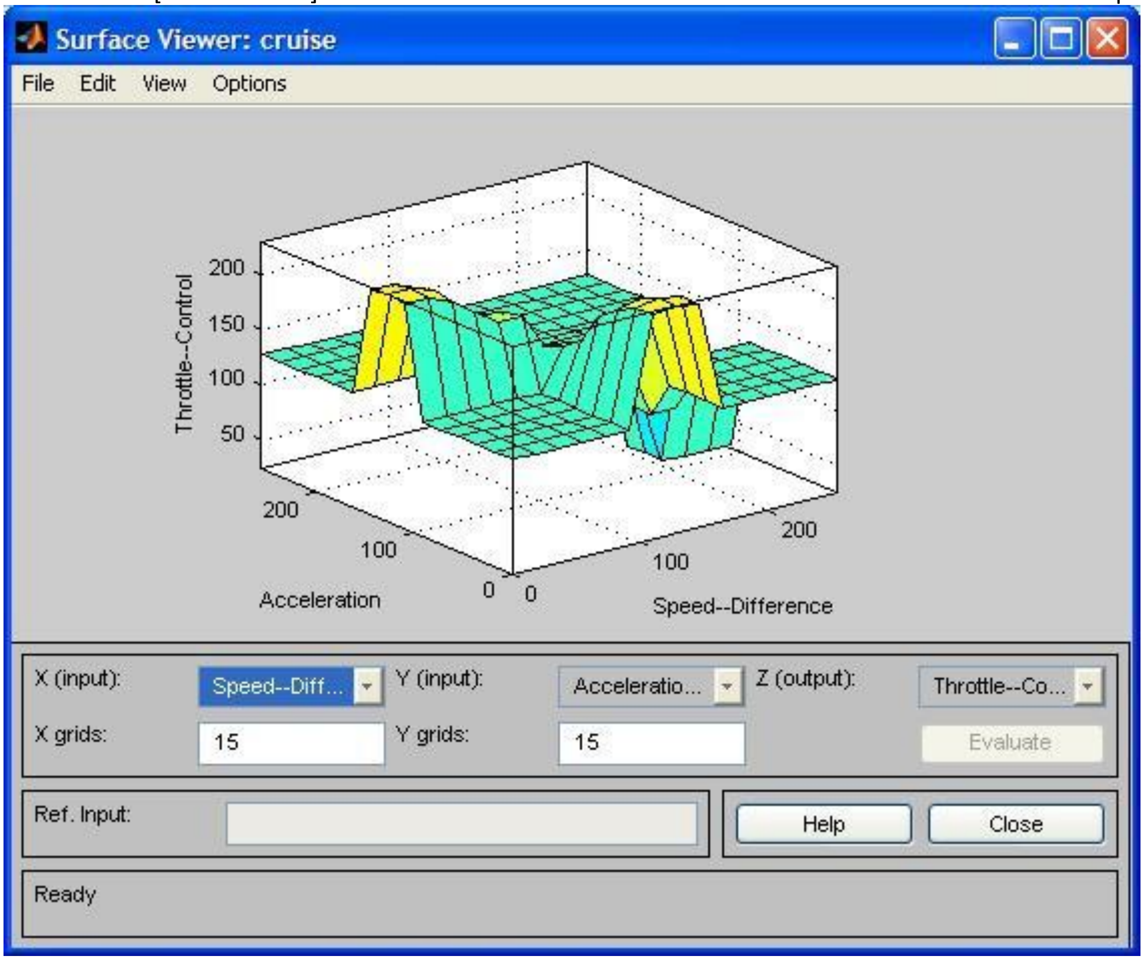
and Acceleration is: NS [✓ ZE] PS PM PL none — not

Then Throttle--Control is: NS ZE PS PM [✓ PL] none — not

Connection: or [✓ and]

Weight: 1

FIS Name: cruise

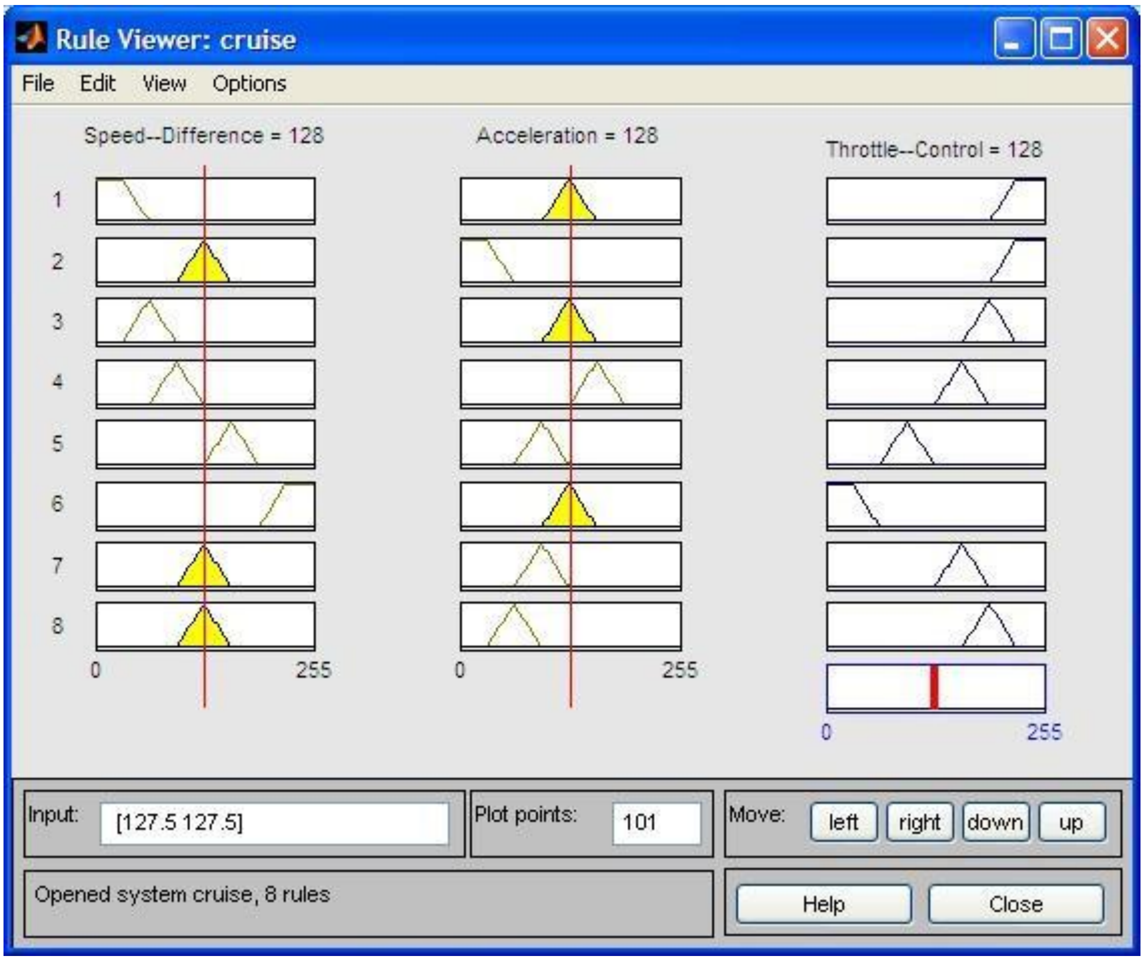


X (input): Speed--Diff... Y (input): Acceleratio... Z (output): Throttle--Co...

X grids: 15 Y grids: 15

Ref. Input:

Ready



Speed--Difference = 128 Acceleration = 128 Throttle--Control = 128

Input: [127.5 127.5] Plot points: 101 Move:

Opened system cruise, 8 rules

PRACTICAL-7

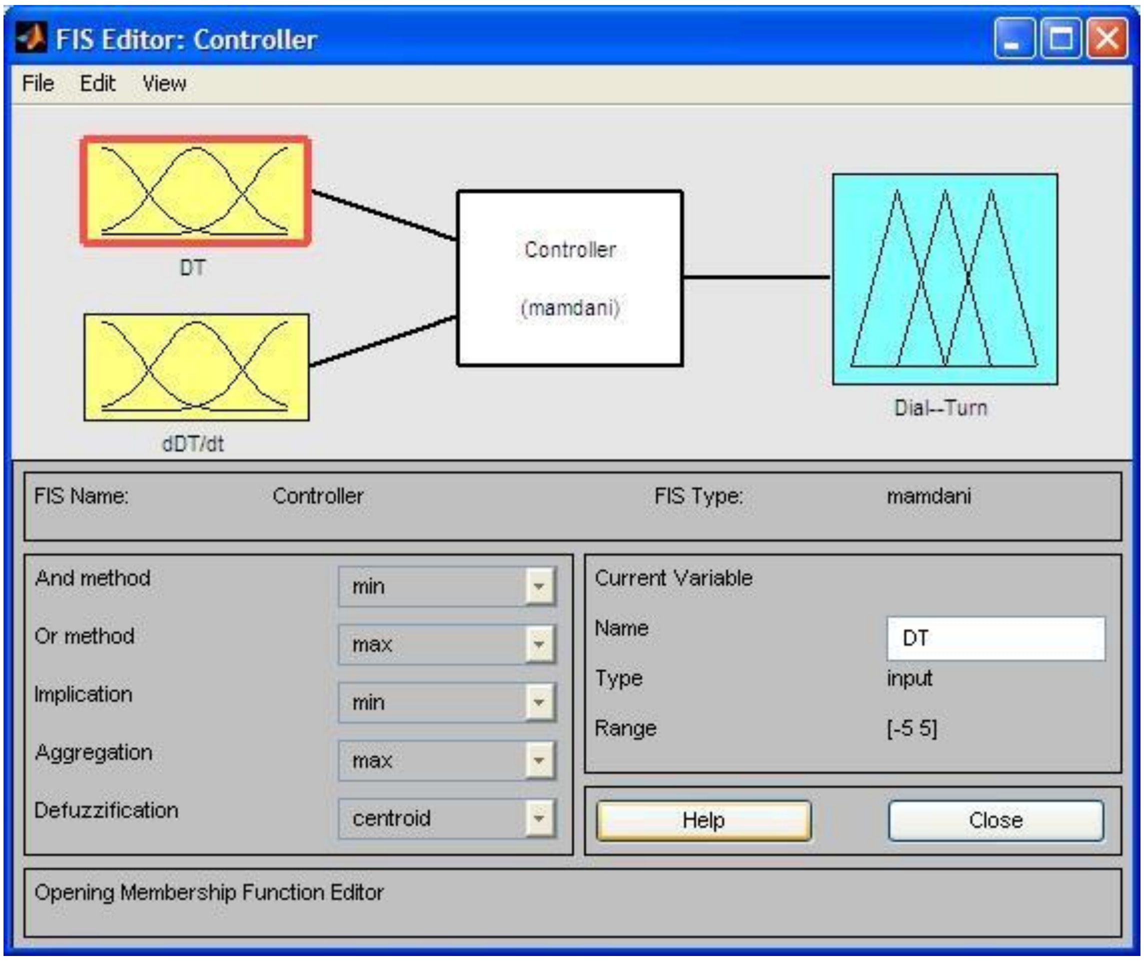
AIM:Solve Air Conditioner Controller using MATLAB Fuzzy logic toolbox

CODE:

```
[System]
Name='cruise'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=8
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'
[Input1]
Name='Speed--Difference'
Range=[0 255]
NumMFs=7
MF1='NL':'trapmf',[0 0 31 63]
MF2='NM':'trimf',[31 63 95]
MF3='NS':'trimf',[63 95 127]
MF4='ZE':'trimf',[95 127 159]
MF5='PS':'trimf',[127 159 191]
MF6='PM':'trimf',[159 191 223]
MF7='PL':'trapmf',[191 223 255 255]
[Input2]
Name='Acceleration'
Range=[0 255]
NumMFs=7
MF1='NL':'trapmf',[0 0 31 63]
MF2='NM':'trimf',[31 63 95]
MF3='NS':'trimf',[63 95 127]
MF4='ZE':'trimf',[95 127 159]
MF5='PS':'trimf',[127 159 191]
MF6='PM':'trimf',[159 191 223]
MF7='PL':'trapmf',[191 223 255 255]
[Output1]
Name='Throttle--Control'
Range=[0 255]
NumMFs=7
MF1='NL':'trapmf',[0 0 31 63]
MF2='NM':'trimf',[31 63 95]
MF3='NS':'trimf',[63 95 127]
MF4='ZE':'trimf',[95 127 159]
MF5='PS':'trimf',[127 159 191]
MF6='PM':'trimf',[159 191 223]
MF7='PL':'trapmf',[191 223 255 255]
```

```
[Rules]
1 4, 7 (1) : 1
4 1, 7 (1) : 1
2 4, 6 (1) : 1
3 5, 5 (1) : 1
5 3, 3 (1) : 1
7 4, 1 (1) : 1
4 3, 5 (1) : 1
4 2, 6 (1) : 1
```

OUTPUT:

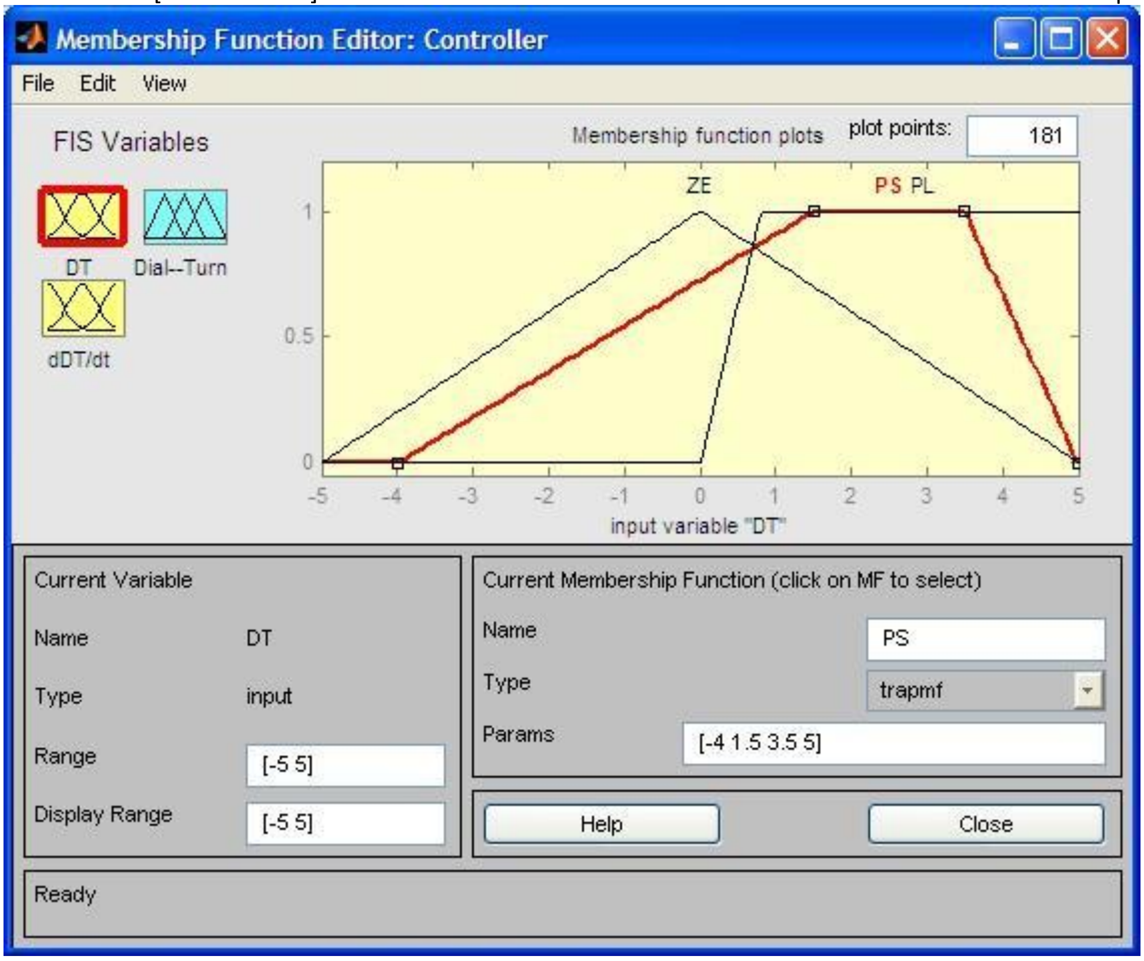


FIS Name: Controller FIS Type: mamdani

And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid

Current Variable
Name: DT
Type: input
Range: [-5 5]

Opening Membership Function Editor



plot points: 181

Current Variable

Name DT

Type input

Range [-5 5]

Display Range [-5 5]

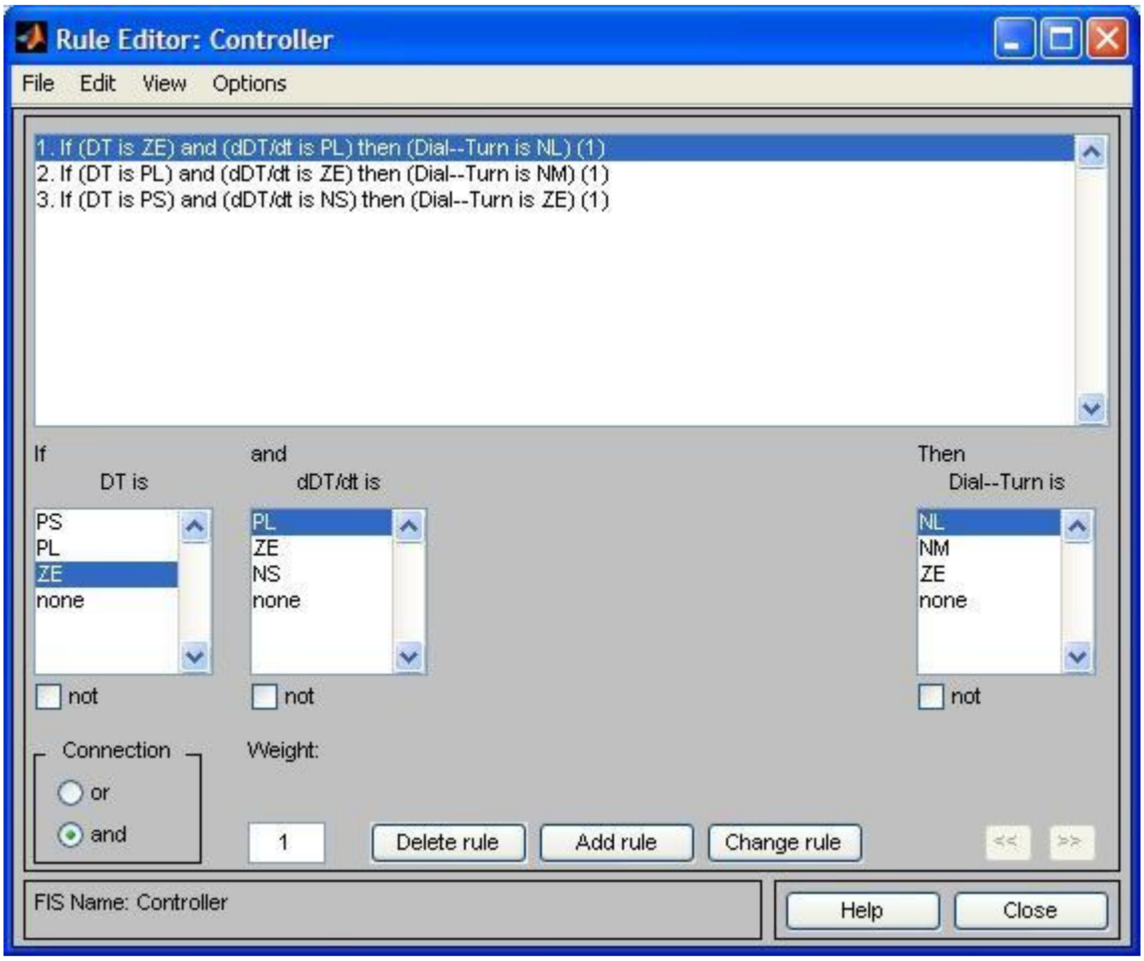
Current Membership Function (click on MF to select)

Name PS

Type trapmf

Params [-4 1.5 3.5 5]

Ready



1. If (DT is ZE) and (dDT/dt is PL) then (Dial--Turn is NL) (1)
2. If (DT is PL) and (dDT/dt is ZE) then (Dial--Turn is NM) (1)
3. If (DT is PS) and (dDT/dt is NS) then (Dial--Turn is ZE) (1)

If DT is: PS, PL, [✓ ZE], none; not

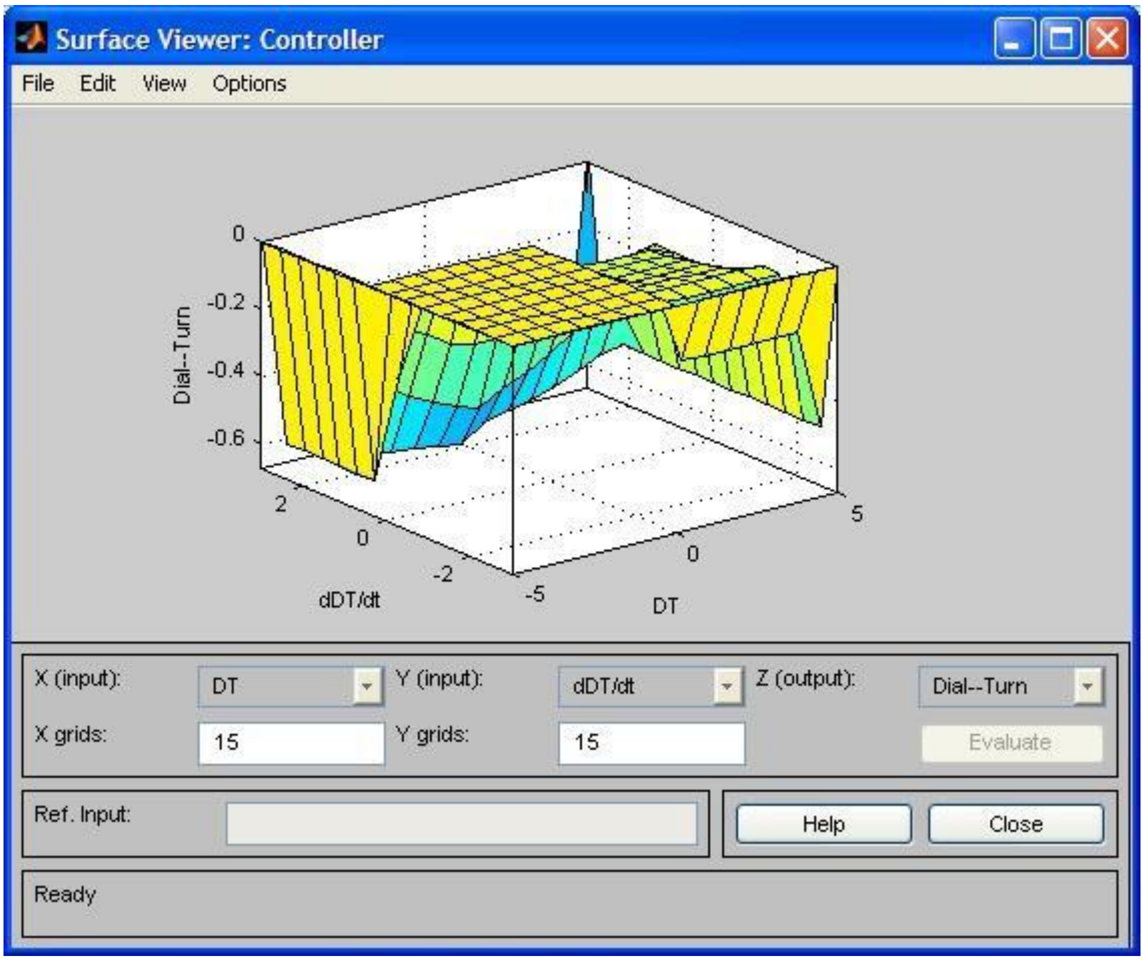
and dDT/dt is: [✓ PL], ZE, NS, none; not

Then Dial--Turn is: [✓ NL], NM, ZE, none; not

Connection: or [✓ and]

Weight: 1

FIS Name: Controller

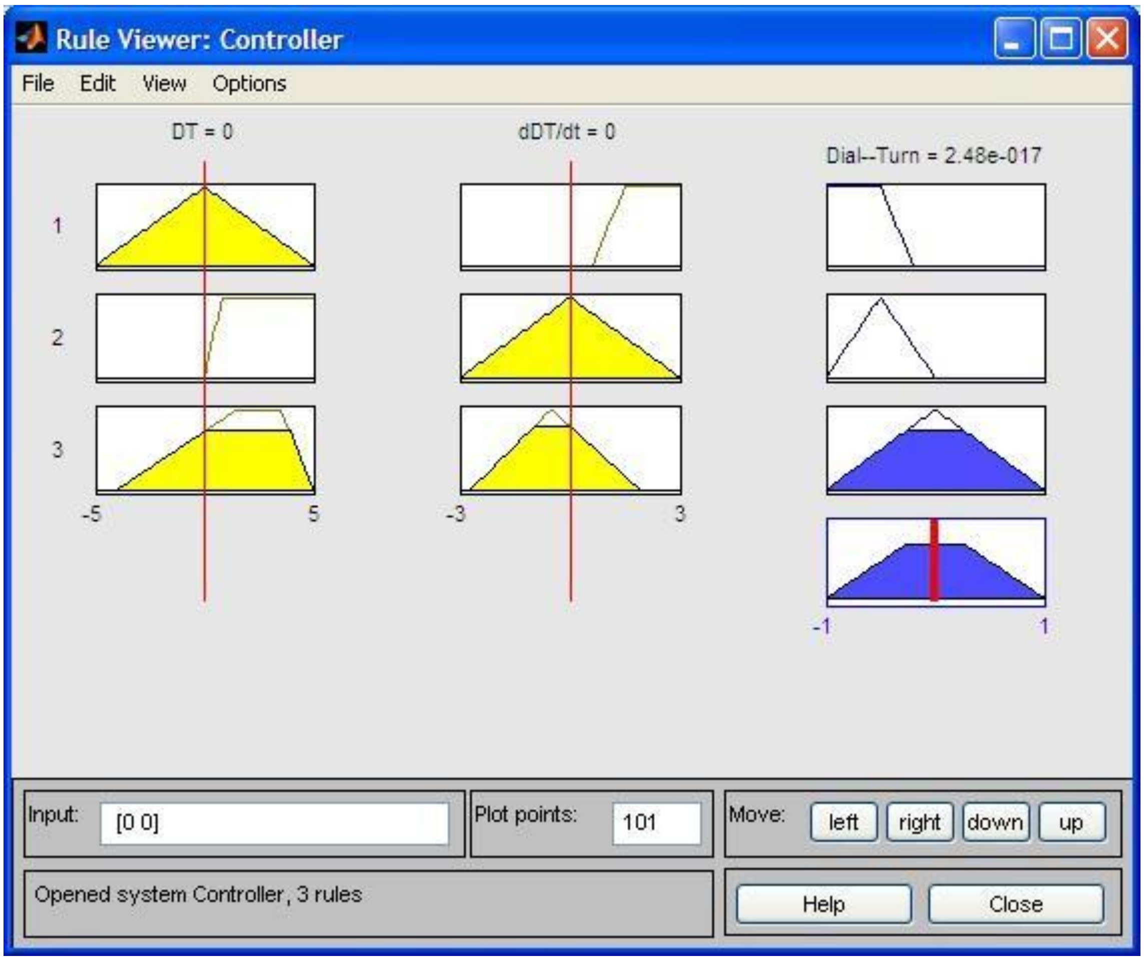


X (input): DT Y (input): dDT/dt Z (output): Dial--Turn

X grids: 15 Y grids: 15

Ref. Input:

Ready



DT = 0 dDT/dt = 0 Dial--Turn = 2.48e-017

Input: [0 0] Plot points: 101

Opened system Controller, 3 rules

PRACTICAL-8

AIM: Implement TSP using GA.

CODE:

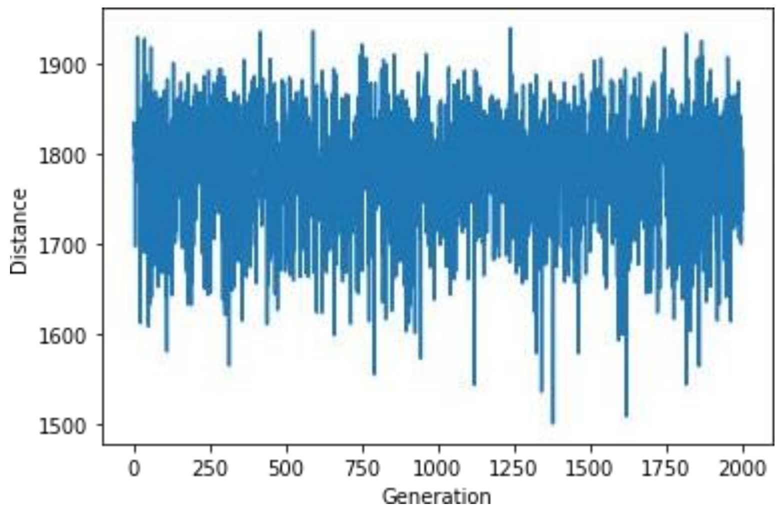
1.Python

```
import numpy as np, random, operator, pandas as pd
import matplotlib.pyplot as plt
def create_starting_population(size,Number_of_city):
    population = []
    for i in range(0,size):
        population.append(create_new_member(Number_of_city))
    return population
def pick_mate(N):
    i=random.randint(0,N)
    return i
def distance(i,j):
    return np.sqrt((i[0]-j[0])**2 + (i[1]-j[1])**2)
def score_population(population, CityList):
    scores = []
    for i in population:
        scores.append(fitness(i, CityList))
    return scores
def fitness(route,CityList):
    score=0
    for i in range(1,len(route)):
        k=int(route[i-1])
        l=int(route[i])
        score = score + distance(CityList[k],CityList[l])
    return score
def create_new_member(Number_of_city):
    pop=set(np.arange(Number_of_city,dtype=int))
    route=list(random.sample(pop,Number_of_city))
    return route
def crossover(a,b):
    child=[]
    childA=[]
    childB=[]
    geneA=int(random.random()* len(a))
    geneB=int(random.random()* len(a))
    start_gene=min(geneA,geneB)
    end_gene=max(geneA,geneB)
    for i in range(start_gene,end_gene):
        childA.append(a[i])
    childB=[item for item in a if item not in childA]
    child=childA+childB
    return child
    def mutate(route,probablity): route=np.array(route)
    for swaping_p in range(len(route)):
        if(random.random() < probablity):
            swapedWith = np.random.randint(0,len(route))
            temp1=route[swaping_p]
            temp2=route[swapedWith]
            route[swapedWith]=temp1 route[swaping_p]=temp2
    return route
def selection(popRanked, eliteSize):
    selectionResults=[]
    result=[]
    for i in popRanked:
        result.append(i[0])
    for i in range(0,eliteSize):
        selectionResults.append(result[i])
    return selectionResults
def rankRoutes(population,City_List):
    fitnessResults = {}
    for i in range(0,len(population)):
        fitnessResults[i] = fitness(population[i],City_List)
    return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse
= False)
def breedPopulation(mating_pool):
    children=[]
    for i in range(len(mating_pool)-1):
            children.append(crossover(mating_pool[i],mating_pool[i+1]))
    return children
def mutatePopulation(children,mutation_rate):
    new_generation=[]
    for i in children:
        muated_child=mutate(i,mutation_rate)
        new_generation.append(muated_child)
    return new_generation
def matingPool(population, selectionResults):
    matingpool = []
    for i in range(0, len(selectionResults)):
        index = selectionResults[i]
        matingpool.append(population[index])
    return matingpool
def next_generation(City_List,current_population,mutation_rate,elite_size):
    population_rank=rankRoutes(current_population,City_List)
    selection_result=selection(population_rank,elite_size)
    mating_pool=matingPool(current_population,selection_result)
    children=breedPopulation(mating_pool)
    next_generation=mutatePopulation(children,mutation_rate)
    return next_generation
```

```
def genetic_algorithm(City_List,size_population=1000,elite_size=75,mutation_Rate
=0.05,generation=2000):



pop=[]
    progress = []
    Number_of_cities=len(City_List)
    population=create_starting_population(size_population,Number_of_cities)
    progress.append(rankRoutes(population,City_List)[0][1])
    print(f"initial route distance {progress[0]}")
    print(f"initial route {population[0]}")
    for i in range(0,generation):
        pop = next_generation(City_List,population,mutation_Rate,elite_size)
        progress.append(rankRoutes(pop,City_List)[0][1])
    rank_=rankRoutes(pop,City_List)[0]
    print(f"Best Route :{pop[rank_[0]]} ")
    print(f"best route distance {rank_[1]}")
    plt.plot(progress) plt.ylabel('Distance')
    plt.xlabel('Generation')
    plt.show()
    return rank_, pop
cityList = []
for i in range(0,25):
    x=int(random.random() * 200)
    y=int(random.random() * 200)
    cityList.append((x,y))
```

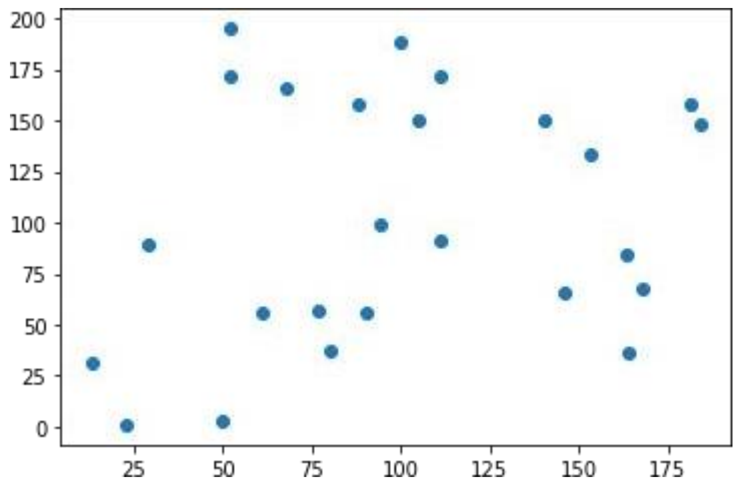
OUTPUT:

```
initial route distance 1808.5488313262974
initial route [14, 9, 20, 18, 5, 19, 10, 0, 7, 23, 12, 4, 3, 15, 8, 22, 13, 24, 16, 11, 1, 2, 21, 6, 17]
Best Route :[12 0 1 23 22 19 11 18 14 6 3 5 10 2 7 9 20 24 17 8 21 13 16  4
 15]
best route distance 1738.0991346786866
```



```
x_axis=[]
y_axis=[]
for i in cityList:
    x_axis.append(i[0])
    y_axis.append(i[1])
plt.scatter(x_axis,y_axis)
plt.show()
```

OUTPUT:



200
175
150
125
100
75
50
25
0
25
50
75
100
125
150
175