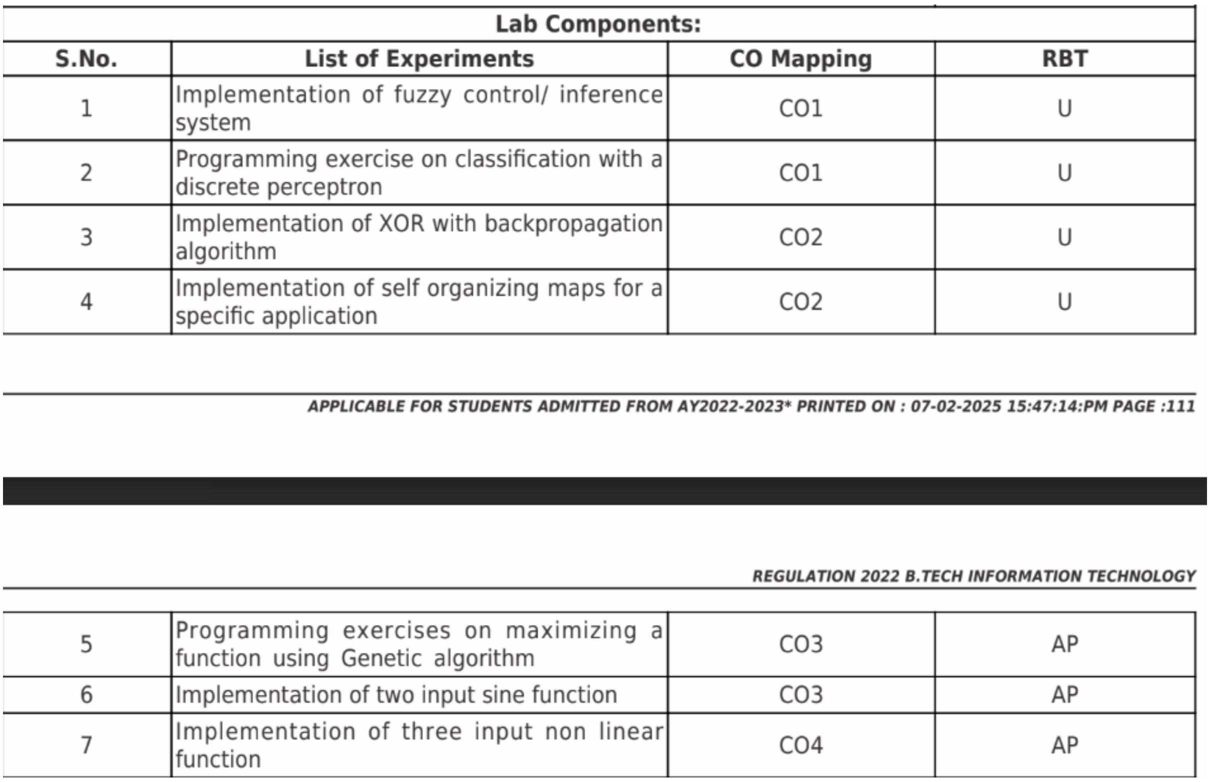

Lab Components:			
S.No.	**List of Experiments**	**CO Mapping**	**RBT**
1	Implementation of fuzzy control/ inference system	CO1	U
2	Programming exercise on classification with a discrete perceptron	CO1	U
3	Implementation of XOR with backpropagation algorithm	CO2	U
4	Implementation of self organizing maps for a specific application	CO2	U
5	Programming exercises on maximizing a function using Genetic algorithm	CO3	AP
6	Implementation of two input sine function	CO3	AP
7	Implementation of three input non linear function	CO4	AP

Exp1 - Implementation of fuzzy control/ inference system

CODE:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

temperature = ctrl.Antecedent(np.arange(0, 101, 1), 'temperature')
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')

temperature['low'] = fuzz.trimf(temperature.universe, [0, 0, 50])
temperature['medium'] = fuzz.trimf(temperature.universe, [0, 50, 100])
temperature['high'] = fuzz.trimf(temperature.universe, [50, 100, 100])

fan_speed['low'] = fuzz.trimf(fan_speed.universe, [0, 0, 50])
fan_speed['medium'] = fuzz.trimf(fan_speed.universe, [0, 50, 100])
fan_speed['high'] = fuzz.trimf(fan_speed.universe, [50, 100, 100])

rule1 = ctrl.Rule(temperature['low'], fan_speed['low'])
rule2 = ctrl.Rule(temperature['medium'], fan_speed['medium'])
rule3 = ctrl.Rule(temperature['high'], fan_speed['high'])

fan_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
fan_speed_ctrl = ctrl.ControlSystemSimulation(fan_ctrl)

temperature_value = 75
fan_speed_ctrl.input['temperature'] = temperature_value
fan_speed_ctrl.compute()

print("Fan Speed:", fan_speed_ctrl.output['fan_speed'])

temperature.view()
fan_speed.view()
fan_speed.view(sim=fan_speed_ctrl)
```

Exp2 – perceptron

CODE:

```
import numpy as np

w=np.zeros(2)
b=0
lr=0.1

def predict(x):
  return int(np.dot(w,x)+b>0)
def test(X,y,epo=100):
  global w,b
  for i in range(epo):
    for xi,target in zip(X,y):
      error=target-predict(xi)
      w+=lr*error*xi
      b+=lr*error

class0=np.array([[2,3],[3,2],[1,1]])
class1=np.array([[5,7],[6,8],[7,6]])

X=np.vstack([class0,class1])
y=np.array([0,0,0,1,1,1])

test(X,y)

test_data=np.array([[4,5],[2,2]])
for i in test_data:
  print(i,"belongs to ",predict(i))
```

Exp3 - XOR with backpropagation algorithm

```
import numpy as np

sig = lambda x: 1 / (1 + np.exp(-x))
dsig = lambda x: x * (1 - x)

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

w1 = np.random.randn(2,2)
w2 = np.random.randn(2,1)
lr = 0.1

for _ in range(10000):
   h = sig(X @ w1)
   o = sig(h @ w2)
   w2 += h.T @ ((y - o) * dsig(o)) * lr
   w1 += X.T @ (((y - o) * dsig(o)) @ w2.T * dsig(h)) * lr

for data in X:
   out = sig(sig(data @ w1) @ w2)
   print(f"Input: {data} → Output: {out[0]:.4f}")
```

5) Maximizing a function using Genetic algorithm
6) Implementation of two input sine function CO3 AP
7) Implementation of three input non linear

CODE:

```
import random
import math

# --- CHANGE ONLY THIS LINE ---
fitness = lambda x: -x[0]**2 + 6*x[0] + 9
# fitness = lambda x: math.sin(x[0]) + math.sin(x[1])
# fitness = lambda x: -(x[0]**2 + x[1]**2 + x[2]**2) + 10 * (
#    math.cos(2 * math.pi * x[0]) +
#    math.cos(2 * math.pi * x[1]) +
#    math.cos(2 * math.pi * x[2])
# )

# --- GA core ---
def ga(fit, dim, gen=50, pop_size=50, lb=-5, ub=5):
    pop = [[random.uniform(lb, ub) for _ in range(dim)] for _ in range(pop_size)]

    for g in range(gen):
        new = []
        for _ in range(pop_size // 2):
            p1, p2 = random.sample(pop, 2)
            c1 = [(a + b) / 2 for a, b in zip(p1, p2)]
            c2 = [(a - b) / 2 for a, b in zip(p1, p2)]

            for c in (c1, c2):
                if random.random() < 0.1:
                    i = random.randrange(dim)
                    c[i] += random.uniform(-0.1, 0.1)
                new.append(c)

        pop = new
        best = max(pop, key=fit)
        print(f"Gen {g + 1}: Best - {tuple(round(v, 3) for v in best)}, Fit = {round(fit(best), 3)}")

    return best
```

```
# --- Run ---
best = ga(fitness, dim=2, gen=30, pop_size=60, lb=-2 * math.pi, ub=2 * math.pi)
print(f"\nBest individual - {tuple(round(v, 3) for v in best)}")
```