Lab: 07

Task 01

```
from numpy.random import randint
from numpy.random import rand
import sympy as sym
import math
n_bits=20
n_pop=100
pop=[]
scores=[]
parent=int()
crossover=int()
muta=int()
set1=int(0)
parsel = {1:'rank',2:'fitness',3:'tournament'}
cross={1:'onepoint',2:'twopoint'}
mutation={1:'insert',2:'flip',3:'swap'}
r_cross = 0.9
r_mut = 1.0 / float(n_bits)
def generate_parent():
    for _ in range(n_pop):
        pop.append(randint(0,2,n_bits).tolist())

def onemax(x):
    scores.append(-sum(x))

def Tournament_selection(pop, scores, k=3):
    # first random selection
    selection_ix = randint(len(pop))
    for ix in randint(0, len(pop), k-1):
        # check if better (e.g. perform a tournament)
        if scores[ix] < scores[selection_ix]:
            selection_ix = ix
    return pop[selection_ix]

def Rank_based_selection(pop,scores):
    scores.sort()
    rank={}
    prob=[]
    rankli=[]
    for i in range(len(scores)):
        rank[i]=scores[i]
```

```
    form=int(((len(scores)+1)*len(scores))/2)

    for i in rank.keys():
        prob.append(int(i/form))
    while(len(prob)!=0):
        rankli.append(pop[rank[prob.index(max(prob))]])
        prob.remove(max(prob))
    return rankli

def fitness_scale(pop,scores):
    low= min(scores)
    high=max(scores)
    suplow=int()
    suphigh=int()
    a,b=sym.symbols('a,b')
    c=int(0)
    c1=int(0)
    fit=[]
    rankli=[]
    for i in scores:
        if i < high and i > low and c!=1:
            suplow=i
            c+=1

        if i > high and i < low and c!=1:
            suphigh=i
            c1+=1

        if c ==1 and c1==1:
            break
    eq1=sym.Eq(a*low+b,suplow)
    eq2=sym.Eq(a*high+b,suphigh)
    result = sym.solve([eq1,eq2],(a,b))

    for i in scores:
      fit.append((result[a]*i)+result[b])

    while(len(fit)!=0):
        rankli.append(pop[scores[fit.index(max(fit))]])
        fit.remove(max(fit))
    return rankli

def order1crossover(p1,p2,r_cross):
    c1,c2 = p1.copy(),p2.copy()
```

```
    if rand() < r_cross:
        pt = randint(1, len(p1)-2)
        # perform crossover
        c1 = p1[:pt] + p2[pt:]
        c2 = p2[:pt] + p1[pt:]
    return [c1, c2]

def order2crossover(p1,p2,r_cross):
    c1,c2 = p1.copy(),p2.copy()

    if rand() < r_cross:
        pt1 = randint(0,9)
        pt2 = randint(10,19)

        c1 = p1[:pt1] + p2[pt1:pt2+1] + p1[:pt2]
        c2 = p2[:pt1] + p1[pt1:pt2+1] + p2[:pt2]
    return [c1,c2]

def insertmutation(bitstring,r_mut):
    b1=[]
    p1= randint(0,9)
    p2 = randint(10,19)
    if rand() < r_mut:
        for i in range(len(bitstring)):
            if i == p1:
                b1.append(bitstring[p1])
                b1.append(bitstring[p2])
            else:
                b1.append(i)
    return b1

def flipmutation(bitstring,r_mut):
     for i in range(len(bitstring)):
        # check for a mutation
        if rand() < r_mut:
            # flip the bit
            bitstring[i] = 1 - bitstring[i]

def swapmutation(bitstring,r_mut):
    p1= randint(0,9)
    p2 = randint(10,19)

    if rand() < r_mut:
        bitstring[p1] = bitstring[p2]
        bitstring[p2]=bitstring[p1]
```

```
generate_parent()
#print(pop)

while(set1!=1):
    pa = int(input("Select the method for parent selection (1:'rank',2:'fitness',
3:'tournament'): "))
    co = int(input("Select the method for crossover(1:'onepoint',2:'twopoint'): "
))
    mu = int(input("Select the method for mutation(mut={1:'insert',2:'flip',3:'sw
ap'}): "))

    if pa in parsel.keys() and co in cross.keys() and mu in mutation.keys():
        parent=pa
        crossover=co
        muta=mu
        set1+=1
    else:
        print("Error! Please enter correct number")

best, best_eval = 0,onemax(pop[0])

def generate_scores():
    for c in pop:
        onemax(c)
best, best_eval = 0,scores[0]
for i in range(n_pop):
    selected=[]
    children=list()
    generate_scores()
    for j in range(n_pop):
        if scores[j] < best_eval:
            best, best_eval = pop[i], scores[i]
            print(">%d, new best f(%s) = %.3f" % (i,pop[j],scores[j]))
    if parent ==1 :
        a= Rank_based_selection(pop,scores)
        for k in a:
            selected.append(k)
    elif parent ==2:
        a= fitness_scale(pop,scores)
        for k in a:
            selected.append(k)
    elif parent ==3 :
        for _ in range(n_pop):
```

```
            selected.append(Tournament_selection(pop,scores))

    if crossover ==1 and muta==1:
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i+1]
            # crossover and mutation
            for c in order1crossover(p1, p2, r_cross):
                # mutation
                insertmutation(c, r_mut)
                # store for next generation
                children.append(c)
        # replace population
        pop = children

    elif crossover ==1 and muta==2:
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i+1]
            # crossover and mutation
            for c in order1crossover(p1, p2, r_cross):
                # mutation
                flipmutation(c, r_mut)
                # store for next generation
                children.append(c)
        # replace population
        pop = children

    elif crossover ==1 and muta==3:
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i+1]
            # crossover and mutation
            for c in order1crossover(p1, p2, r_cross):
                # mutation
                swapmutation(c, r_mut)
                # store for next generation
                children.append(c)
        # replace population
        pop = children
    elif crossover ==2 and muta==1:
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i+1]
            # crossover and mutation
```

```
            for c in order2crossover(p1, p2, r_cross):
                # mutation
                insertmutation(c, r_mut)
                # store for next generation
                children.append(c)
        # replace population
        pop = children
    elif crossover ==2 and muta==2:
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i+1]
            # crossover and mutation
            for c in order2crossover(p1, p2, r_cross):
                # mutation
                flipmutation(c, r_mut)
                # store for next generation
                children.append(c)
        # replace population
        pop = children
    elif crossover ==2 and muta==3:
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i+1]
            # crossover and mutation
            for c in order2crossover(p1, p2, r_cross):
                # mutation
                swapmutation(c, r_mut)
                # store for next generation
                children.append(c)
        # replace population
        pop = children
```

Task:02

```python
import math
def EQ1function():
    x=[2,1,0,-1,-2]
    y=[2,1,0,-1,-2]
    queue={}

    for i in x:
        for j in y:
            a=-(pow(i,2)+pow(j,2))
            ans= math.exp(a)
            queue[(i,j)]=ans
    value = []
    key = []
    for i,j in queue.items():
        value.append(j)
        key.append(i)
    maximum = max(value)

    print("The maximum value of this function is",str(maximum),"w

def EQ2function():
    x=[3.0,2.5,2.0,1.5,1.0,0.5,0.0,-0.5,-1.0,-1.5,-2.0]
    y=[3.0,2.5,2.0,1.5,1.0,0.5,0.0,-0.5,-1.0,-1.5,-2.0]
    queue={}
    for i in x:
        for j in y:
            a = pow(i-1,2)
            b=100*pow(j-pow(i,2),2)
            ans = a+b
            queue[(i,j)]=ans
    value = []
    key = []
    for i,j in queue.items():
        value.append(j)
        key.append(i)
    maximum = max(value)

    print("The maximum value of this function is",str(maximum),"w

print("For Equation1")
EQ1function()
print("For Equation2")
EQ2function()
```

Output

```
For Equation1
For Equation2
The maximum value of this function is 12104.0 which is occur at (3.0, -2.0)
```