GeneticAlgo_MONKOUN

April 27, 2022

1 Définition des fonctions

[ ]:
```
import sympy
import math
import numpy
from copy import copy
```

1.1 Création de la fonction

[ ]:
```
fonction=lambda x: numpy.sin((x*numpy.pi)/(255))
```

[ ]:
```
#binary = bin(16)
#binary
X=[1,1,1,1,1,1,1,1]
Y=[1,0,0,0,1,0,0,0]
```

[ ]:
```
def liste_en_decimale(X):
    binaire=0
    taille=len(X)-1

    for i in X:
        binaire+=(2**taille)*i
        taille=taille-1

    return binaire
```

[ ]:
```
#Test:
a=liste_en_decimale(X)
a
```

[ ]: 255

1.2 Génération de la population

[ ]:
```
def randpop(pop_size=20):
    population=[]
    for i in range(pop_size):
        nouveau=numpy.random.choice((0,1),8)
        population.append(nouveau)
    return population
```

[ ]:
```
#Application de création de la population
population=randpop(10)
population
```

[ ]:
```
[array([1, 0, 0, 0, 0, 0, 0, 0]),
 array([0, 0, 1, 1, 0, 0, 0, 1]),
 array([1, 1, 1, 0, 1, 0, 0, 1]),
 array([0, 1, 1, 1, 0, 0, 0, 1]),
 array([0, 0, 1, 0, 1, 1, 1, 1]),
 array([1, 0, 1, 0, 1, 0, 1, 1]),
 array([0, 0, 1, 1, 0, 0, 0, 0]),
 array([1, 1, 1, 0, 1, 0, 0, 1]),
 array([1, 0, 0, 1, 1, 1, 1, 1]),
 array([1, 1, 0, 1, 1, 0, 0, 1])]
```

1.3 Selection deux à deux des parents

[ ]:
```
def select(pop_size=10,n_parents=2):
    liste_selection=[]
    for i in range(pop_size):
        sous_liste=[]
        for i in range(n_parents):
            a=numpy.random.randint(pop_size)
            sous_liste.append(a)
        liste_selection.append(sous_liste)

    return liste_selection
```

[ ]:
```
#Application selection de la population
list_sel=select()
list_sel
```

[ ]:
```
[[9, 8],
 [2, 3],
 [8, 2],
 [1, 7],
```

```
 [1, 8],
 [7, 4],
 [2, 7],
 [1, 3],
 [1, 9],
 [1, 6]]
```

1.4 Croisements Mi

[ ]:
```python
def elements_croisement_mi(n_parents):

    #n_parents doit etre une liste de parents
    #Pour chaque parent on va créer deux sous chaines moitié moitié
    #Puis insérer toutes ces moitiés dans une liste unique

    liste=[]
    for parent_i in n_parents:
        longueur=len(parent_i)

        mi=math.floor(longueur/2)

        parent_i1=parent_i[0:mi]
        liste.append(parent_i1)
        parent_i2=parent_i[mi:longueur]
        liste.append(parent_i2)

    #Combinaison

    return liste
```

[ ]:
```python
#Test et combinaison
X1=[1,1,1,1,1,1,1,1]
X2=[1,0,0,0,1,0,0,0]

l1=elements_croisement_mi([X1,X2])
l2=[]
from itertools import combinations
for i,j in combinations(l1,2):
    l2.append(i+j)
l2
```

[ ]:
```
[[1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 1, 1, 0, 0, 0],
 [1, 1, 1, 1, 1, 0, 0, 0],
 [1, 1, 1, 1, 1, 0, 0, 0],
 [1, 1, 1, 1, 1, 0, 0, 0],
```

```
 [1, 0, 0, 0, 1, 0, 0, 0]]
```

1.5 Croisements Uniforme

```
[ ]: def crossover(parent_a, parent_b):
         fils=[];
         for i in range(len(parent_a)):
             choix=numpy.random.randint(2) #Choisir entre deux nombres 0 et 1

             if (choix==0):
                 fils.append(parent_a[i])
             if (choix==1):
                 fils.append(parent_b[i])
         return fils
```

```
[ ]: #Test sur deux parents pris au hasard
     f=crossover(population[1],population[3])
     f
```

```
[ ]: [0, 0, 1, 1, 0, 0, 0, 1]
```

1.6 Croisements des parents de chaque ligne du select pour avoir la pop. fille

```
[ ]: def offspring(list_sel=list_sel, population_=population):
         pop_fille=[]
         for i in list_sel:
             fille=crossover(population_[i[0]],population_[i[1]])
             pop_fille.append(fille)
         return pop_fille
```

```
[ ]: #Application du croisement par génération d'une population fille
     population_fille=offspring()
     population_fille
```

```
[ ]: [[1, 0, 0, 1, 1, 1, 1, 1],
      [0, 1, 1, 0, 0, 0, 0, 1],
      [1, 1, 0, 0, 1, 1, 0, 1],
      [0, 1, 1, 1, 0, 0, 0, 1],
      [0, 0, 0, 1, 1, 0, 1, 1],
      [0, 0, 1, 0, 1, 0, 1, 1],
      [1, 1, 1, 0, 1, 0, 0, 1],
      [0, 0, 1, 1, 0, 0, 0, 1],
      [1, 0, 1, 1, 1, 0, 0, 1],
      [0, 0, 1, 1, 0, 0, 0, 0]]
```

1.7 Mise en commun des deux populations

[ ]:
```
array_pop_fille=numpy.array(population_fille)
array_pop_fille
```

[ ]:
```
array([[1, 0, 0, 1, 1, 1, 1, 1],
       [0, 1, 1, 0, 0, 0, 0, 1],
       [1, 1, 0, 0, 1, 1, 0, 1],
       [0, 1, 1, 1, 0, 0, 0, 1],
       [0, 0, 0, 1, 1, 0, 1, 1],
       [0, 0, 1, 0, 1, 0, 1, 1],
       [1, 1, 1, 0, 1, 0, 0, 1],
       [0, 0, 1, 1, 0, 0, 0, 1],
       [1, 0, 1, 1, 1, 0, 0, 1],
       [0, 0, 1, 1, 0, 0, 0, 0]])
```

[ ]:
```
new_population=numpy.vstack((population, array_pop_fille))
new_population
```

[ ]:
```
array([[1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 0, 0, 0, 1],
       [1, 1, 1, 0, 1, 0, 0, 1],
       [0, 1, 1, 1, 0, 0, 0, 1],
       [0, 0, 1, 0, 1, 1, 1, 1],
       [1, 0, 1, 0, 1, 0, 1, 1],
       [0, 0, 1, 1, 0, 0, 0, 0],
       [1, 1, 1, 0, 1, 0, 0, 1],
       [1, 0, 0, 1, 1, 1, 1, 1],
       [1, 1, 0, 1, 1, 0, 0, 1],
       [1, 0, 0, 1, 1, 1, 1, 1],
       [0, 1, 1, 0, 0, 0, 0, 1],
       [1, 1, 0, 0, 1, 1, 0, 1],
       [0, 1, 1, 1, 0, 0, 0, 1],
       [0, 0, 0, 1, 1, 0, 1, 1],
       [0, 0, 1, 0, 1, 0, 1, 1],
       [1, 1, 1, 0, 1, 0, 0, 1],
       [0, 0, 1, 1, 0, 0, 0, 1],
       [1, 0, 1, 1, 1, 0, 0, 1],
       [0, 0, 1, 1, 0, 0, 0, 0]])
```

1.8 Mutation

[ ]:
```
def mutation(individu):

    individu2=copy(individu)
    #Associer une probabilité à chaque case
```

```
    #Une probabilité est un nombre entre 0 et 1 donc 1/p avec p non nul
    #Si p <= seuil on mute la case sinon on le garde

    seuil=1/8
    p=100

    for i in range(0,len(individu2)):
        test="Refaire"
        while(test=="Refaire"):
            deno=numpy.random.randint(p)
            if (deno!=0):
                prob=1/deno
                test="Arret"
                if prob<seuil:
                    individu2[i]=numpy.abs(1-individu2[i])

    return individu2
```

[ ]:
```
#Test mutation sur un individu
individu_q=population[5]
print(individu_q)
individu_=mutation(individu_q)
print(individu_)
```

```
[1 0 1 0 1 0 1 1]
[0 1 0 1 0 0 0 0]
```

[ ]:
```
#Mutations sur toute la population:
population_muted=mutation(new_population)
population_muted
```

[ ]:
```
array([[0, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 0, 0, 1, 1, 1, 0],
       [0, 0, 0, 1, 0, 1, 1, 0],
       [0, 1, 1, 1, 0, 0, 0, 1],
       [1, 1, 0, 1, 0, 0, 0, 0],
       [0, 1, 0, 1, 0, 1, 0, 0],
       [1, 1, 0, 0, 1, 1, 1, 1],
       [0, 0, 0, 1, 0, 1, 1, 0],
       [0, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 1, 1, 0],
       [0, 1, 1, 0, 0, 0, 0, 0],
       [0, 1, 1, 0, 0, 0, 0, 1],
       [1, 1, 0, 0, 1, 1, 0, 1],
       [1, 0, 0, 0, 1, 1, 1, 0],
       [1, 1, 1, 0, 0, 1, 0, 0],
       [1, 1, 0, 1, 0, 1, 0, 0],
```

```
[0, 0, 0, 1, 0, 1, 1, 0],
[1, 1, 0, 0, 1, 1, 1, 0],
[0, 1, 0, 0, 0, 1, 1, 0],
[1, 1, 0, 0, 1, 1, 1, 1]])
```

1.9 Survival - Plus hautes fitness

[ ]:
```
#def survival(f=fonction, pop, n_survivors=10):
def survival(f, pop, n_survivors):
    #Conversion de la population en liste:
    pop_list=pop.tolist()

    #Forme decimale des individus
    pop_decimale=[]
    for i in pop_list:
        i=liste_en_decimale(i)
        pop_decimale.append(i)

    #Calcul de la fitness
    fitness=[]
    for i in pop_decimale:
        fitness.append(f(i))

    #Ordonner les fitness
    fitness1=copy(fitness)
    fitness1.sort(reverse=True) #Procéder comme ça trie par ordre décroissant
    final_survivors_fitness=copy(fitness1[:n_survivors])

    #Ici on veut avoir les indices des éléments triés avec argsort.
    fitness2=copy(fitness)
    sort_croissant_index = numpy.argsort(fitness2).tolist()

    #Mais on a trie par ordre croissant donc on va renverser cette liste
    #Pour les avoir en decroissant
    taille = len(sort_croissant_index) - 1
    sort_decroissant_index = []
    while (taille >= 0):
        sort_decroissant_index.append(sort_croissant_index[taille])
        taille = taille - 1

    #Choix des n_survivors premiers avec meilleure fitness
    #Ayant leurs indices dans sort_decroissant_index
    premiers_index=copy(sort_decroissant_index[:n_survivors])

    #Les individus ayant premiers_index
    final_survivors=[]
```

```
    for i in premiers_index:
        final_survivors.append(pop_list[i])


    return (final_survivors,final_survivors_fitness)
```

[ ]:
```
#Application sur la population mutée
survivants=survival(fonction,population_muted,10)
survivants
# Il s'agira d'un tuple avec à l'index 0 les survivants et à  1 leurs fitness
```

[ ]:
```
([[0, 1, 1, 1, 1, 1, 1, 1],
  [0, 1, 1, 1, 0, 0, 0, 1],
  [1, 0, 0, 0, 1, 1, 1, 0],
  [0, 1, 1, 0, 0, 0, 0, 1],
  [0, 1, 1, 0, 0, 0, 0, 0],
  [0, 1, 1, 0, 0, 0, 0, 0],
  [0, 1, 0, 1, 0, 1, 0, 0],
  [0, 1, 0, 0, 0, 1, 1, 0],
  [1, 1, 0, 0, 1, 1, 0, 1],
  [1, 1, 0, 0, 1, 1, 1, 0]],
 [0.9999810273487268,
  0.9840863373026044,
  0.9840863373026044,
  0.9302293085467402,
  0.9256376597815562,
  0.9256376597815562,
  0.8597998514483723,
  0.7594049166547071,
  0.5777738314082512,
  0.5676747161445903])
```

1.10 Survival - Plus basses fitness

[ ]:
```
def survival_(f, pop, n_survivors):
    #Conversion de la population en liste:
    pop_list=pop.tolist()

    #Forme decimale des individus
    pop_decimale=[]
    for i in pop_list:
        i=liste_en_decimale(i)
        pop_decimale.append(i)

    #Calcul de la fitness
    fitness=[]
```

```
    for i in pop_decimale:
        fitness.append(f(i))

    #Ordonner les fitness
    fitness2=copy(fitness)
    fitness2.sort() #Procéder comme ça trie par ordre décroissant
    final2_survivors_fitness=copy(fitness2[:n_survivors])

    #Ici on veut avoir les indices des éléments triés avec argsort.
    fitness2=copy(fitness)
    sort_croissant_index = numpy.argsort(fitness2).tolist()


    #Choix des n_survivors premiers avec meilleure fitness
    #Ayant leurs indices dans sort_decroissant_index
    derniers_index=copy(sort_croissant_index[:n_survivors])

    #Les individus ayant premiers_index
    final2_survivors=[]
    for i in derniers_index:
        final2_survivors.append(pop_list[i])


    return (final2_survivors,final2_survivors_fitness)
```

[ ]:
```
#Application sur la population mutée
survivants2=survival_(fonction,population_muted,10)
survivants2
```

[ ]:
```
([[0, 0, 0, 1, 0, 1, 1, 0],
  [0, 0, 0, 1, 0, 1, 1, 0],
  [0, 0, 0, 1, 0, 1, 1, 0],
  [1, 1, 1, 0, 0, 1, 0, 0],
  [0, 0, 1, 0, 0, 1, 1, 0],
  [1, 1, 0, 1, 0, 1, 0, 0],
  [1, 1, 0, 1, 0, 0, 0, 0],
  [1, 1, 0, 0, 1, 1, 1, 1],
  [1, 1, 0, 0, 1, 1, 1, 1],
  [1, 1, 0, 0, 1, 1, 1, 0]],
 [0.26773300332246786,
  0.26773300332246786,
  0.26773300332246786,
  0.32653871284008334,
  0.4512440570453228,
  0.505325183948948,
  0.5472195469221109,
  0.5574894393428858,
```

```
  0.5574894393428858,
  0.5676747161445903])
```

1.11 Elimination des duplicata

```
[ ]: from scipy.spatial.distance import cdist

     def eliminate_duplicates(X):
         D = cdist(X, X)
         D[numpy.triu_indices(len(X))] = numpy.inf
         return numpy.all(D > 1e-32, axis=1)
```

```
[ ]: #Recupération des survivants
     survivants_=survivants[0]
     survivants_fitn=survivants[1]
     survivants_=numpy.array(survivants_)
     survivants_fitn=numpy.array(survivants_fitn)
     survivants_
```

```
[ ]: array([[0, 1, 1, 1, 1, 1, 1, 1],
            [0, 1, 1, 1, 0, 0, 0, 1],
            [1, 0, 0, 0, 1, 1, 1, 0],
            [0, 1, 1, 0, 0, 0, 0, 1],
            [0, 1, 1, 0, 0, 0, 0, 0],
            [0, 1, 1, 0, 0, 0, 0, 0],
            [0, 1, 0, 1, 0, 1, 0, 0],
            [0, 1, 0, 0, 0, 1, 1, 0],
            [1, 1, 0, 0, 1, 1, 0, 1],
            [1, 1, 0, 0, 1, 1, 1, 0]])
```

```
[ ]: #Recupération du masque bool
     #Le masque bool est retour de la fonction eliminate_duplicates sur survivants_
     pop_sansduplicata_maskbool=eliminate_duplicates(survivants_)
     pop_sansduplicata_maskbool
```

```
[ ]: array([ True,  True,  True,  True,  True, False,  True,  True,  True,
             True])
```

```
[ ]: #Application sur les survivants via le masque booléen
     pop_sansduplicata=survivants_[pop_sansduplicata_maskbool]
     pop_sansduplicata
```

```
[ ]: array([[0, 1, 1, 1, 1, 1, 1, 1],
            [0, 1, 1, 1, 0, 0, 0, 1],
            [1, 0, 0, 0, 1, 1, 1, 0],
            [0, 1, 1, 0, 0, 0, 0, 1],
```

```
       [0, 1, 1, 0, 0, 0, 0, 0],
       [0, 1, 0, 1, 0, 1, 0, 0],
       [0, 1, 0, 0, 0, 1, 1, 0],
       [1, 1, 0, 0, 1, 1, 0, 1],
       [1, 1, 0, 0, 1, 1, 1, 0]])
```

[ ]:
```
#Leur fitness
fitness_pop_sansduplicata=survivants_fitn[pop_sansduplicata_maskbool]
fitness_pop_sansduplicata
```

[ ]:
```
array([0.99998103, 0.98408634, 0.98408634, 0.93022931, 0.92563766,
       0.85979985, 0.75940492, 0.57777383, 0.56767472])
```

2 Creation du main

[55]:
```
def main(pop_size=10, nb_parents=2, nb_generations=2, nb_survivants=20,␣
 ↪fonction_=fonction):
    # Creation de la population
    population_=randpop(pop_size)

    #Multiplication de la population
    population_generee=copy(population_)
    for i in range(nb_generations):
        #Selection
        list_select=select(pop_size,nb_parents)
        #Croisements
        population_fille_=offspring(list_select, population_generee)
        array_pop_fille_=numpy.array(population_fille_)
        new_population_=numpy.vstack((population_generee, array_pop_fille))
        #Mutations
        population_muted_=mutation(new_population_)
        #Mise à jour variables
        population_generee=population_muted_
        pop_size=len(population_generee)

    #Les plus forts
    les_survivants=survival(fonction_,population_generee,nb_survivants)

    les_survivants_=les_survivants[0]
    les_survivants_fitn=les_survivants[1]

    les_survivants_=numpy.array(les_survivants_)
    les_survivants_fitn=numpy.array(les_survivants_fitn)

    pop_sansduplicata_maskbool_=eliminate_duplicates(les_survivants_)
```

```
    pop_sansduplicata_=les_survivants_[pop_sansduplicata_maskbool_]

    fitness_pop_sansduplicata_=les_survivants_fitn[pop_sansduplicata_maskbool_]

    #Les plus faibles
    les_survivants2=survival_(fonction_,population_generee,nb_survivants)

    les_survivants_2=les_survivants2[0]
    les_survivants_fitn2=les_survivants2[1]

    les_survivants_2=numpy.array(les_survivants_2)
    les_survivants_fitn2=numpy.array(les_survivants_fitn2)

    pop_sansduplicata_maskbool_2=eliminate_duplicates(les_survivants_2)

    pop_sansduplicata_2=les_survivants_2[pop_sansduplicata_maskbool_2]

    ␣
 ↪fitness_pop_sansduplicata_2=les_survivants_fitn2[pop_sansduplicata_maskbool_2]

    a=pop_sansduplicata_
    b=fitness_pop_sansduplicata_
    c=pop_sansduplicata_2
    d=fitness_pop_sansduplicata_2
    e=population_generee

    return (a,b,c,d,e)
```

[ ]:
```
def transformations_supp(pop_forte,fitness_forte,pop_faible,fitness_faible,␣
 ↪la_population): #Le retourn de main
    #Conversion de la population en liste:
    popu_list=la_population.tolist()

    #Forme decimale des individus
    popu_decimale=[]
    for i in popu_list:
        i=liste_en_decimale(i)
        popu_decimale.append(i)

    #Calcul de la fitness
    fitnesses=[]
    for i in popu_decimale:
        fitnesses.append(fonction(i))

    fitness_array=numpy.array(fitnesses)
    moy=fitness_array.mean()
```

```
    min_x=liste_en_decimale(pop_faible[0])
    min_y=fitness_faible[0]
    min=[min_x,min_y]

    max_x=liste_en_decimale(pop_forte[0])
    max_y=fitness_forte[0]
    max=[max_x,max_y]

    return (popu_decimale,fitnesses,moy,min,max)
```

2.0.1 Premier test du main

[ ]:
```
(a,b,c,d,e)=main(pop_size=10, nb_parents=2,nb_generations=1, nb_survivants=20,␣
↪fonction_=fonction)
```

[ ]:
```
a #Les survivants les plus forts
```

[ ]:
```
array([[1, 0, 0, 0, 1, 1, 1, 0],
       [1, 0, 0, 1, 0, 1, 0, 1],
       [1, 0, 0, 1, 1, 1, 0, 1],
       [1, 0, 0, 1, 1, 1, 1, 0],
       [0, 1, 1, 0, 0, 0, 0, 0],
       [0, 1, 0, 1, 1, 1, 1, 1],
       [1, 0, 1, 0, 1, 1, 1, 0],
       [0, 1, 0, 0, 0, 1, 1, 1],
       [0, 1, 0, 0, 0, 1, 1, 0],
       [0, 0, 1, 1, 1, 1, 1, 0],
       [0, 0, 1, 1, 0, 0, 1, 0],
       [1, 1, 0, 0, 1, 1, 1, 0],
       [1, 1, 0, 0, 1, 1, 1, 1],
       [0, 0, 1, 0, 1, 0, 1, 1],
       [1, 1, 0, 1, 0, 1, 0, 0],
       [0, 0, 1, 0, 0, 1, 1, 0],
       [1, 1, 1, 0, 0, 1, 0, 0],
       [0, 0, 0, 1, 1, 0, 1, 1],
       [0, 0, 0, 1, 0, 1, 1, 0]])
```

[ ]:
```
b #Les fitness des plus forts
```

[ ]:
```
array([0.98408634, 0.96512409, 0.93467977, 0.93022931, 0.92563766,
       0.92090552, 0.84034407, 0.76736268, 0.75940492, 0.69169844,
       0.57777383, 0.56767472, 0.55748944, 0.50532518, 0.50532518,
       0.45124406, 0.32653871, 0.32653871, 0.267733  ])
```

[ ]:
```
c #Les survivants les plus faibles
```

```
[ ]: array([[0, 0, 0, 1, 0, 1, 1, 0],
           [0, 0, 0, 1, 1, 0, 1, 1],
           [1, 1, 1, 0, 0, 1, 0, 0],
           [0, 0, 1, 0, 0, 1, 1, 0],
           [1, 1, 0, 1, 0, 1, 0, 0],
           [0, 0, 1, 0, 1, 0, 1, 1],
           [1, 1, 0, 0, 1, 1, 1, 1],
           [1, 1, 0, 0, 1, 1, 1, 0],
           [0, 0, 1, 1, 0, 0, 1, 0],
           [0, 0, 1, 1, 1, 1, 1, 0],
           [0, 1, 0, 0, 0, 1, 1, 0],
           [0, 1, 0, 0, 0, 1, 1, 1],
           [1, 0, 1, 0, 1, 1, 1, 0],
           [0, 1, 0, 1, 1, 1, 1, 1],
           [0, 1, 1, 0, 0, 0, 0, 0],
           [1, 0, 0, 1, 1, 1, 1, 0],
           [1, 0, 0, 1, 1, 1, 0, 1],
           [1, 0, 0, 1, 0, 1, 0, 1],
           [1, 0, 0, 0, 1, 1, 1, 0]])
```

```
[ ]: d #Les fitness des plus faibles
```

```
[ ]: array([0.267733  , 0.32653871, 0.32653871, 0.45124406, 0.50532518,
           0.50532518, 0.55748944, 0.56767472, 0.57777383, 0.69169844,
           0.75940492, 0.76736268, 0.84034407, 0.92090552, 0.92563766,
           0.93022931, 0.93467977, 0.96512409, 0.98408634])
```

```
[ ]: e #Toute la dernière population
```

```
[ ]: array([[0, 0, 0, 1, 1, 0, 1, 1],
           [1, 0, 0, 1, 0, 1, 0, 1],
           [0, 1, 0, 0, 0, 1, 1, 1],
           [1, 0, 1, 0, 1, 1, 1, 0],
           [0, 0, 1, 0, 1, 0, 1, 1],
           [0, 0, 1, 1, 1, 1, 1, 0],
           [0, 1, 0, 1, 1, 1, 1, 1],
           [0, 0, 1, 0, 0, 1, 1, 0],
           [1, 0, 0, 1, 1, 1, 0, 1],
           [0, 0, 1, 0, 0, 1, 1, 0],
           [0, 1, 1, 0, 0, 0, 0, 0],
           [1, 0, 0, 1, 1, 1, 1, 0],
           [0, 0, 1, 1, 0, 0, 1, 0],
           [1, 0, 0, 0, 1, 1, 1, 0],
           [1, 1, 1, 0, 0, 1, 0, 0],
           [1, 1, 0, 1, 0, 1, 0, 0],
           [0, 0, 0, 1, 0, 1, 1, 0],
           [1, 1, 0, 0, 1, 1, 1, 0],
```

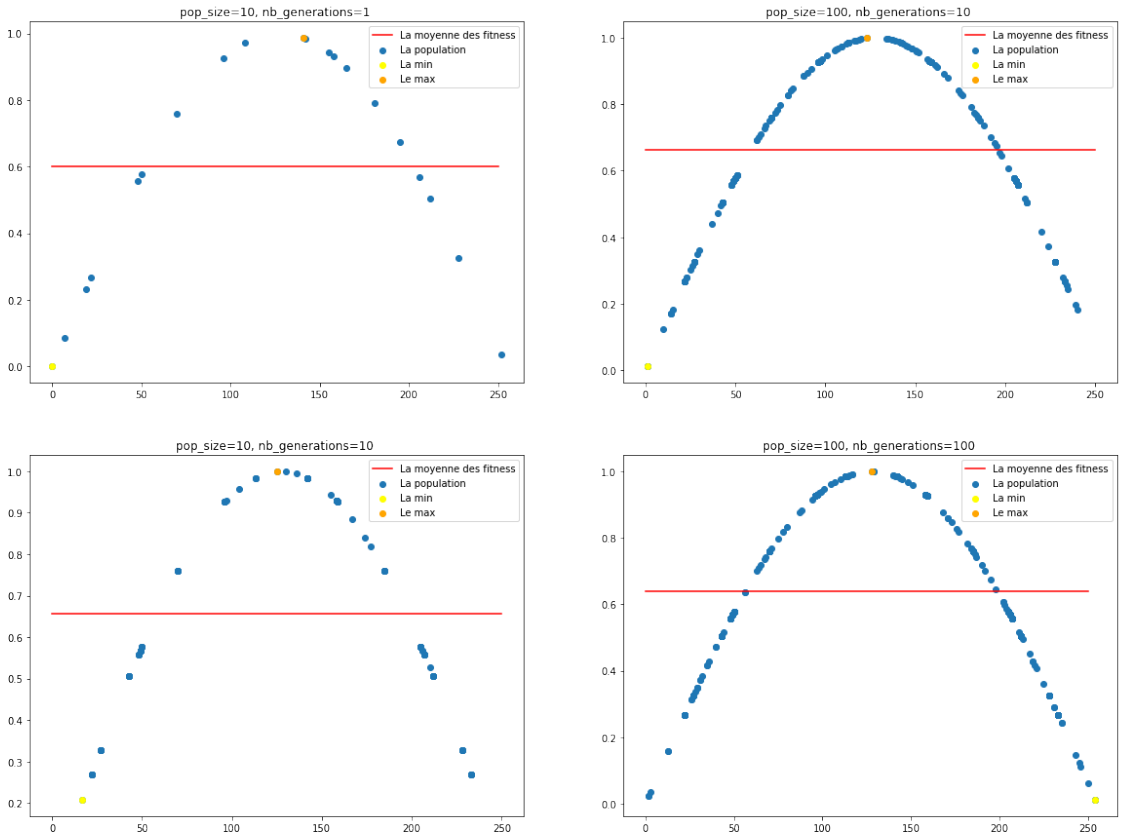
```
[0, 1, 0, 0, 0, 1, 1, 0],
[1, 1, 0, 0, 1, 1, 1, 1]])
```

3 Graphiques

[ ]:
```python
import matplotlib.pyplot as plt
```

[ ]:
```python
plt.figure(figsize=(20,15))

plt.subplot(2,2,1)
(a,b,c,d,e)=main(pop_size=10, nb_parents=2,nb_generations=1, nb_survivants=20,␣
 ↪fonction_=fonction)
(popu_decimale,fitnesses,moy,min,max)=transformations_supp(a,b,c,d,e)
plt.scatter(popu_decimale , fitnesses, label="La population")
plt.plot([0,250],[moy,moy],c="red", label="La moyenne des fitness")
plt.scatter(min[0],min[1],c="yellow", label="La min")
plt.scatter(max[0],max[1],c="orange", label="Le max")
plt.title("pop_size=10, nb_generations=1 ")
plt.legend()

plt.subplot(2,2,2)
(a,b,c,d,e)=main(pop_size=100, nb_parents=2,nb_generations=10,␣
 ↪nb_survivants=20, fonction_=fonction)
(popu_decimale,fitnesses,moy,min,max)=transformations_supp(a,b,c,d,e)
plt.scatter(popu_decimale , fitnesses, label="La population")
plt.plot([0,250],[moy,moy],c="red", label="La moyenne des fitness")
plt.scatter(min[0],min[1],c="yellow", label="La min")
plt.scatter(max[0],max[1],c="orange", label="Le max")
plt.title("pop_size=100, nb_generations=10 ")
plt.legend()

plt.subplot(2,2,3)
(a,b,c,d,e)=main(pop_size=10, nb_parents=2,nb_generations=10, nb_survivants=20,␣
 ↪fonction_=fonction)
(popu_decimale,fitnesses,moy,min,max)=transformations_supp(a,b,c,d,e)
plt.scatter(popu_decimale , fitnesses, label="La population")
plt.plot([0,250],[moy,moy],c="red", label="La moyenne des fitness")
plt.scatter(min[0],min[1],c="yellow", label="La min")
plt.scatter(max[0],max[1],c="orange", label="Le max")
plt.title("pop_size=10, nb_generations=10 ")
plt.legend()

plt.subplot(2,2,4)
(a,b,c,d,e)=main(pop_size=100, nb_parents=2,nb_generations=100,␣
 ↪nb_survivants=20, fonction_=fonction)
```

```
(popu_decimale,fitnesses,moy,min,max)=transformations_supp(a,b,c,d,e)
plt.scatter(popu_decimale , fitnesses, label="La population")
plt.plot([0,250],[moy,moy],c="red", label="La moyenne des fitness")
plt.scatter(min[0],min[1],c="yellow", label="La min")
plt.scatter(max[0],max[1],c="orange", label="Le max")
plt.title("pop_size=100, nb_generations=100 ")
plt.legend()

plt.show()
```



[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```