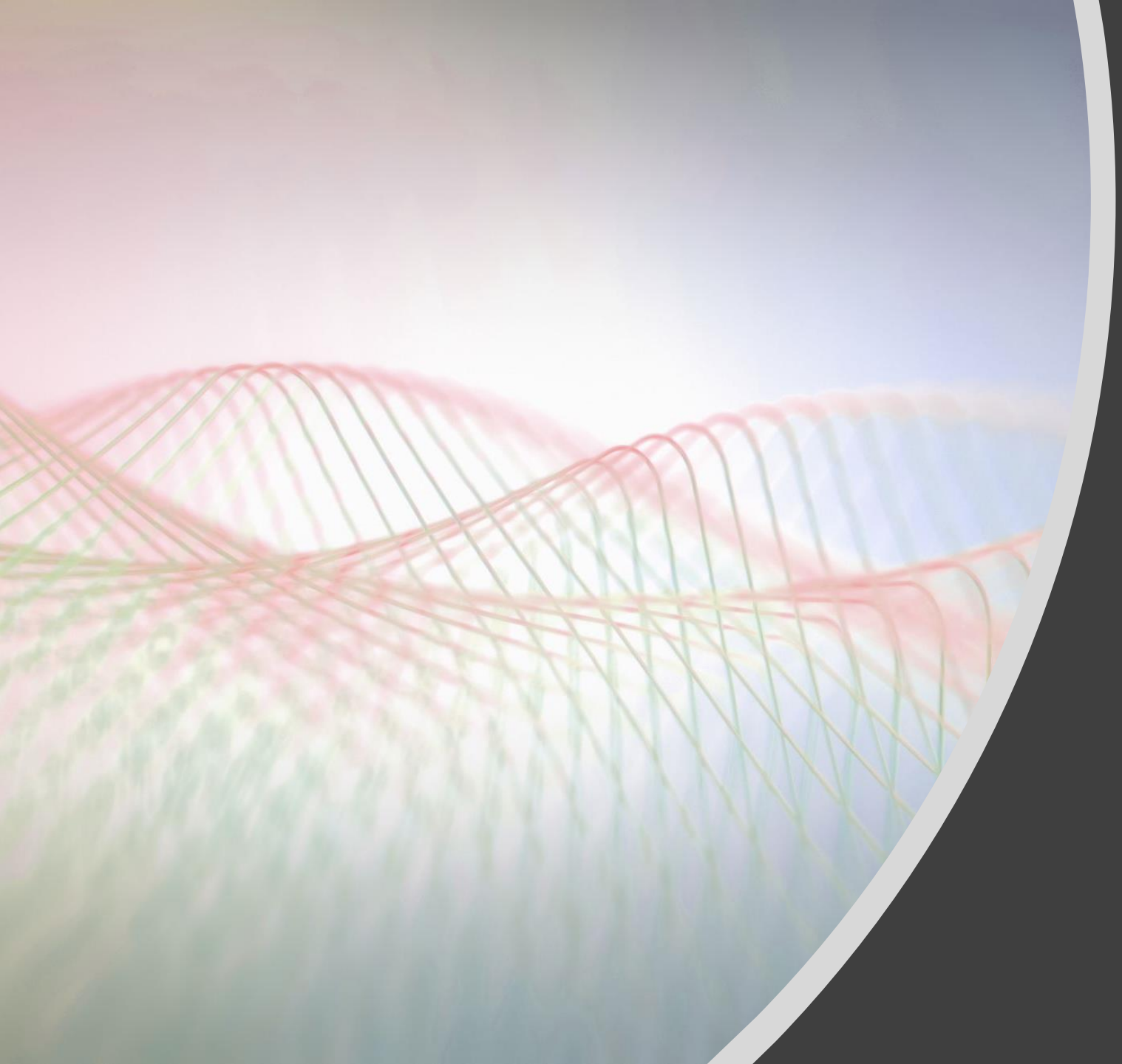


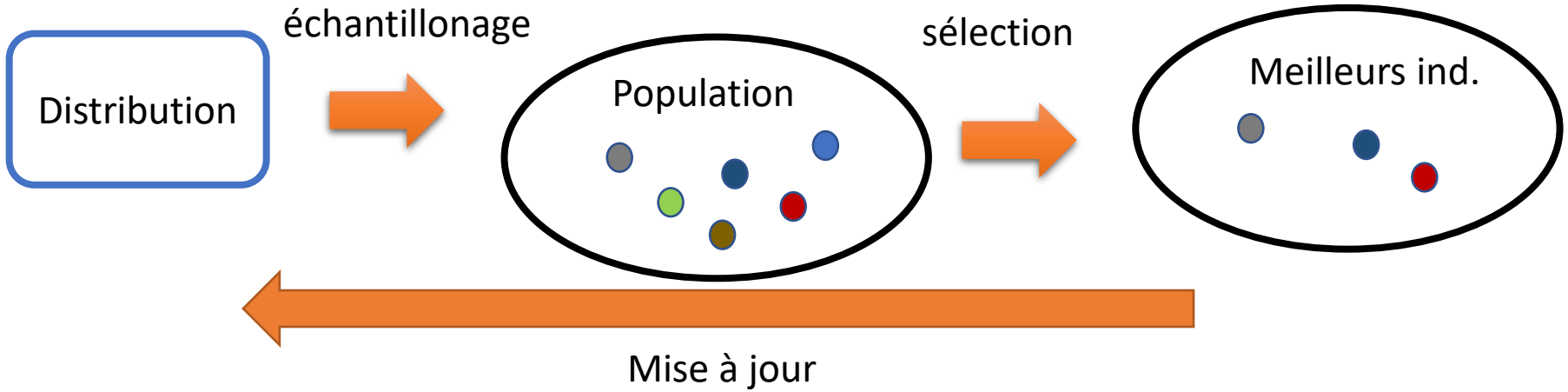
Implémentation d’un algorithme génétique par estimation de distribution avec DEAP

Application sur le problème du OneMax

Schéma général de l'algorithme proposé

Pseudo Code

```
Initialization of distribution model (vector) to represent uniform
distribution over solutions
while (termination criteria not met) do
    population := generate N>0 candidate solutions by sampling model
    evaluation of population
    update of distribution model by selecting best individuals
```



Déclaration des paramètres

```
# taille du problème
ONE_MAX_LENGTH = 15
# Paramètres AG
POPULATION_SIZE = 20
MAX_GENERATIONS = 500
NB_SEL = 10
NB_RUNS = 20
```

Intialisation de la distribution sous la forme d'un simple vecteur

```
def genere_distib_initiale():
    distrib=[]
    for position in range(ONE_MAX_LENGTH):
        distrib.append(random.random())
    return distrib
```

distrib	0,5	0,2	0,8	0,1	0,5	0,3

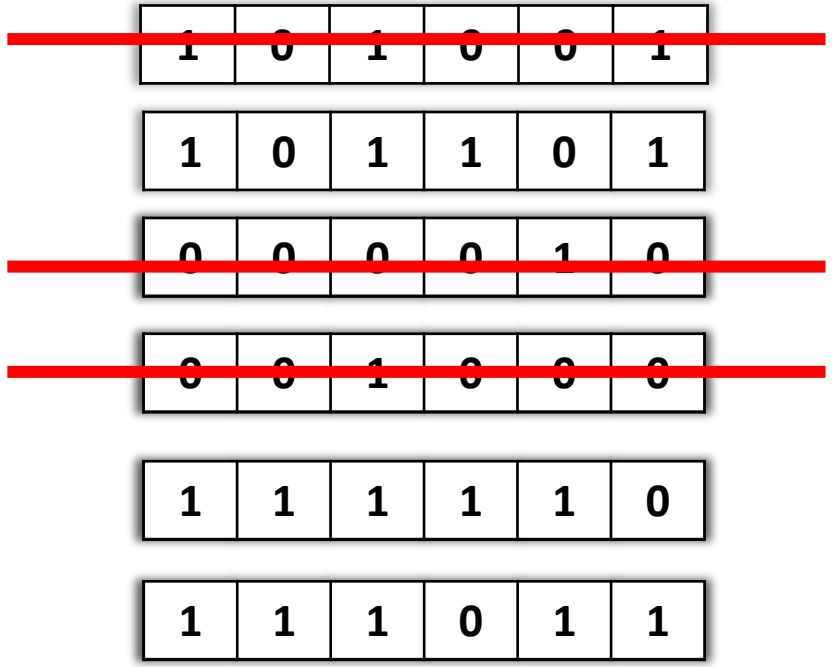
Génération d'une population à partir de la distribution

```
def genere_population_distribution(population,distrib):
    for individu in population:
        for position in range(ONE_MAX_LENGTH):
            if random.random() < distrib[position]:
                individu[position] = 1
            else:
                individu[position] = 0
```

distrib	0,5	0,2	0,8	0,1	0,5	0,3
	1	0	1	0	0	1
	0	0	0	0	1	0
	0	0	1	0	0	0
	1	1	1	1	1	0
	1	1	1	0	1	1
	0	0	0	0	1	0

Mise à jour de la distribution

```
def maj_estimation_distribution(population,distrib,k):
    k_best = tools.selBest(population,k)
    for position in range(ONE_MAX_LENGTH):
        somme = 0
        for ind in k_best:
            somme=somme+ind[position]
        distrib[position]=somme/k
```



Sélection

Mise à jour de la distribution

```
def maj_estimation_distribution(population,distrib,k):
    k_best = tools.selBest(population,k)
    for position in range(ONE_MAX_LENGTH):
        somme = 0
        for ind in k_best:
            somme=somme+ind[position]
        distrib[position]=somme/k
```

	1	0	1	1	0	1
	1	1	1	1	1	0
	1	1	1	0	1	1
distrib	1	0,66	1	0,66	0,66	0,66

MAJ

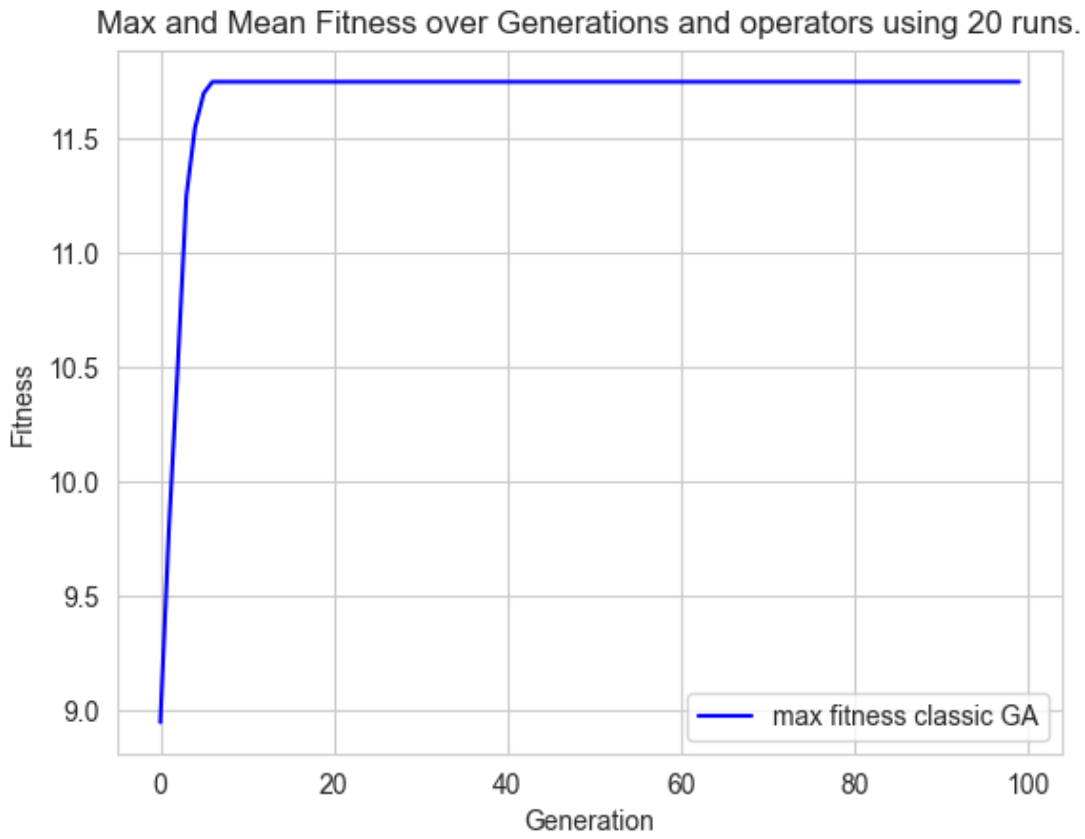
Corps de la boucle principale

```
genere_population_distribution(population,distrib)
            fitnessValues = list(map(toolbox.evaluate, population))
            for individual, fitnessValue in zip(population, fitnessValues):
                individual.fitness.values = fitnessValue
            fitnessValues = [individual.fitness.values[0] for individual in population]

maj_estimation_distribution(population,distrib,NB_SEL)
```

Ajout d’une boucle pour effecteur plusieurs runs puis faire une moyenne par génération

```
Mean_maxFitnessValues = []
    for g in range(MAX_GENERATIONS):
        somme_max = 0
        for r in range(NB_RUNS):
            somme_max = somme_max +
maxFitness_history[r][g]

Mean_maxFitnessValues.append(somme_max/NB_RUNS)
```

Exécution


Max and Mean Fitness over Generations and operators using 20 runs.
Fitness
11.5
11.0
10.5
10.0
9.5
9.0
0
20
40
60
80
100
Generation
max fitness classic GA