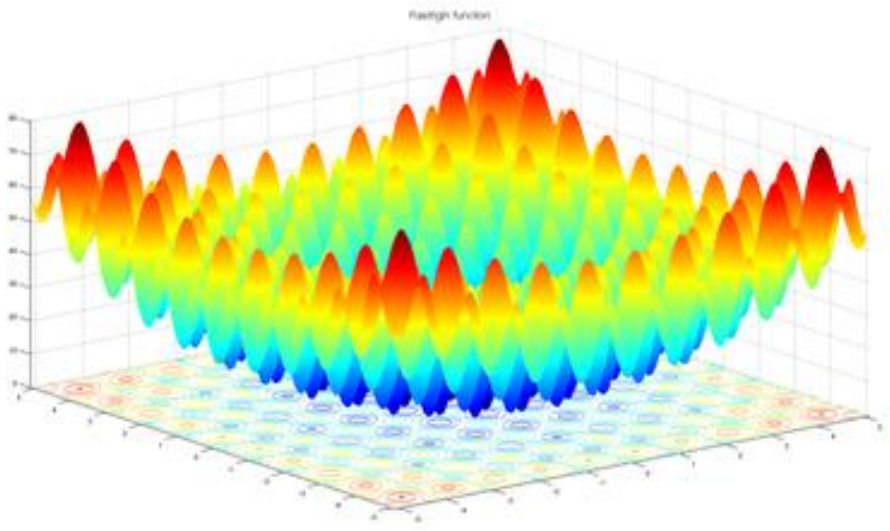
Explicatii algoritmi si pseudo cod

Functia Rastrigin



$$f_6(x) = 10 \cdot n + \sum_{i=1}^{n} \left(x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i) \right)$$

$$-5.12 \leq x_i \leq 5.12$$

Algoritm evolutiv pentru minimizarea functiei Rastrigin

```
def fitness(n, x):
    f = 10 * n
    for i in range(len(x)):
        f = f + x[i] ** 2 - 10 * math.cos(2 * math.pi * x[i])
    return f
```

Functia care calculeaza fitness-ul individului.

```
def selectie_turnir(pop, pop_size):
    sample = np.random.default_rng().choice(pop_size, size=5, replace=False)
    best = pop[sample[0]].copy()
    for i in sample:
        if fitness(len(pop[i]), pop[i]) < fitness(len(best), best):
            best = pop[i].copy()
    return best
```

Am folosit functia pentru selectia turnir a parintilor (functia de la laboratorul trecut).

Selectia parintilor:

```
def sel_turnir(obj, pop, pop_size):
    sample = np.random.default_rng().choice(pop_size, size=5, replace=False)
    best = pop[sample[0]].copy()
    pentru i in sample:
        daca fitness(pop[i], obj[0], obj[1]) > fitness(best, obj[0], obj[1]):
            best = pop[i].copy()
    return best
```

Selectia turnir: se aleg 5 indivizi din populatie, apoi cel mai bun dintre ei este returnat pentru a fi parinte.

Incrucisarea medie

```
def incrucisare_medie(x, y, n):
    i = np.random.randint(n, size=2)
    print(i)
    xcopy = x.copy()
    x[i[0]] = (x[i[0]] + y[i[0]]) / 2
    y[i[0]] = (xcopy[i[0]] + y[i[0]]) / 2

    verificare(x, i[0])
    verificare(y, i[0])

    x[i[1]] = (x[i[1]] + y[i[1]]) / 2
    y[i[1]] = (xcopy[i[1]] + y[i[1]]) / 2

    verificare(x, i[1])
    verificare(y, i[1])
```

Returneaza 2 copii din 2 parinti, verificand valorile pentru a nu avea solutii care nu fac parte din domeniu in generatie.

Incrucisarea convexa simpla

```
def incrucisare_convexa_simpla(x, y, n):
    pos = np.random.randint(n)
    alpha = random.random()
    xcopy = x.copy()
    ycopy = y.copy()
    for i in range(pos, n):
        x[i] = alpha * xcopy[i] + (1 - alpha) * ycopy[i]
        y[i] = alpha * ycopy[i] + (1 - alpha) * xcopy[i]

        verificare(x, i)
        verificare(y, i)
```

Calculeaza valorile indivizilor dupa formula de mai sus, unde alfa este generat aleator.

Mutatia uniforma

```
def mutatie_uniforma(x, t):
    for i in range(len(x)):
        if random.random() < 0.3:
            x[i] = random.uniform(r.MIN_INT, r.MAX_INT)
```

Daca numarul random ales este mai mic decat 0.3, atunci valoarea de pe pozitia i este inlocuita cu alta valoare aleasa aleator din domeniul functiei.

Mutatia gaussiana

```
def mutatie_gaussiana(x, t):
    SIGMA = 1 / (t + 1)
    MU = 0
    for i in range(len(x)):
        n = random.gauss(MU, SIGMA)
        if r.MIN_INT <= x[i] + n <= r.MAX_INT:
            x[i] += n
```

Daca rezultatul functiei gaussiane adunat la valorile individului nu iese din demoniul functiei, atunci modifican valorea adunand rezultatul functiei gaussiene.

Functia principala

```
def rastriginAE(filename, nr_executii, n, nr_gen, pop_size, funcIncrucisare,
funcMutatie):
    exect = 0
    best_of_all_gen = []
    worst_of_all_gen = []
    allavg = []
    timp = []
    bestmean = []
    worstmean = []
    while exect < nr_executii:
        start_time = time.time()
        pop = init_pop(pop_size, n)
        t = 0
        best = [sorted(pop, key=lambda x: fitness(n, x))[0]]
        worst = [sorted(pop, key=lambda x: fitness(n, x))[-1]]
        avg = []
        while t < nr_gen:
            copii = []
            for i in range(pop_size // 2):
                parinte1 = selectie_turnir(pop, pop_size)
                parinte2 = selectie_turnir(pop, pop_size)
                copil1 = parinte1[:]
                copil2 = parinte2[:]
                funcIncrucisare(copil1, copil2, n)

                copii.append(copil1)
                copii.append(copil2)
```

```
            copiimutanti = []
            for i in range(len(copii)):
                funcMutatie(copii[i], t)
            pop = best_of_all(pop, copii,pop_size, n)
            best.append(pop[0])
            worst.append(pop[-1])
            avg.append(meanPop(pop, n))
            t = t + 1
        mean = 0
        for i in range(0, len(avg)):
            mean = mean + avg[i]
        mean = mean / len(avg)
        allavg.append(mean)
        best_of_all_gen.append(sorted(pop, key=lambda x: fitness(n, x))[0])
        worst_of_all_gen.append(sorted(pop, key=lambda x: fitness(n, x))[-1])
        bestmean.append(meanPop(best_of_all_gen, n))
        worstmean.append(meanPop(worst_of_all_gen, n))
        timp.append(start_time)
        exect = exect + 1
    show(best_of_all_gen, worst_of_all_gen, n)
    f.write(filename, bestmean, worstmean, allavg, timp, nr_gen, nr_executii,
pop_size, best_of_all_gen[0])
```

Functia care executa algoritmul evolutiv. Pentru fiecare executie, initializam populatia doar cu solutii valide, apoi salvez cea mai buna solutie si cea mai rea din populatia initiala. Apoi aleg cate 2 parinti, din care vor rezulta 2 copii (prin functia de incrucisare data ca parametru). Dupa ce s-a generat numarul de copii (egal cu cel al parintilor), vor fi aplicate mutatiile pe acestia (functia de mutatie este data ca parametru). Pentru generatia urmatoare se aleg cei mai buni indivizi dintre copii care au suferit mutatii si parinti.

Tabel de valori:

Executii	Incrucisare	Mutatie	Dimensiune	Nr. indivizi	Nr. generatii	Cea mai buna medie a celor mai buni indivizi	Cea mai slaba medie a celor mai slabi indivizi	Average	Timpul mediu de executie
10	medie	gauss	5	200	100	2.58	25.09	21.27	1619470572.35
	medie	uniforma	5	200	100	61.09	122.69	123.90	1619470970.08
	medie	gauss	5	200	500	4.97	3.21	9.76	1619471448.90
	medie	uniforma	5	200	500	0.12	9.10	8.38	1619472569.88
	medie	uniforma	5	200	1000	0.0045	4.79	4.29	1619503636.75
	medie	uniforma	10	200	100	5.62	36.30	34.73	1619504903.24

	medie	gauss	10	200	100	1.51	18.97	15.75	1619505292.24
	medie	uniforma	10	200	500	0.10	0.21	0.10	1619506255.70
	medie	gauss	10	200	500	3.73	8.42	6.39	1619508000.86
10	convexa	uniforma	10	200	100	8.11	43.20	40.0	1619509244.10
	convexa	gauss	10	200	100	2.01	20.66	16.53	1619510492.81
	convexa	uniforma	10	200	500	0.166	14.33	11.91	1619513042.11
	convexa	gauss	10	200	500	3.48	6.39	5.74	1619515045.08
	convexa	uniforma	5	200	100	0.0050	6.28	5.38	1619516215.82
	convexa	gauss	5	200	100	0.0029	4.85	4.44	1619516493.14
	convexa	uniforma	5	200	500	1.0149673208559307e-06	1.62	1.20	1619517128.79
	convexa	gauss	5	200	500	5.339870796 38365e-05	2.79	1.43	1619518428.93
	convexa	uniforma	5	200	1000	3.880555140 227671e-05	0.53	0.58	1619519960.90
	convexa	gauss	5	200	1000	7.533112607 127635e-06	0.45	0.42	1619522337.51

Observatii:

Incrucisarea medie: cea mai buna medie (0.10) obtinuta a fost pentru 500 de generatii si 200 de indivizi si mutatia uniforma. Media solutiilor bune si a celor slabe este tot de 0.10, asta insemnand ca valorile slabe sunt foarte apropiate de cele bune. Cea mai slaba solutie a fost obtinuta tot cu incrucisarea uniforma, pentru 200 de indivizi si 100 de generatii. Dintre cele doua, mutatia uniforma a dat cele mai bune rezultate: 0.10 (mentionat mai sus) si 0.12 (pentru 200 si 500 de generatii).

Incrucisarea convexa simpla: Cea mai buna valoare a fost obtinuta pentru 200 de indivizi si 500 de generatii: 1.0149673208559307e-06. Aceasta incrucisare a dat rezultate mult mai bune decat incrucisarea medie. Mediile celor mai slabi indivizi sunt in general mai mici decat pentru incrucisarea medie, cea mai mica fiind 0.45. Cu ambele mutatii s-au obtinut rezultate bune, dar mutatia uniforma a dat rezultate putin mai bune (nu semnificativ). De exemplu, pentru mutatia

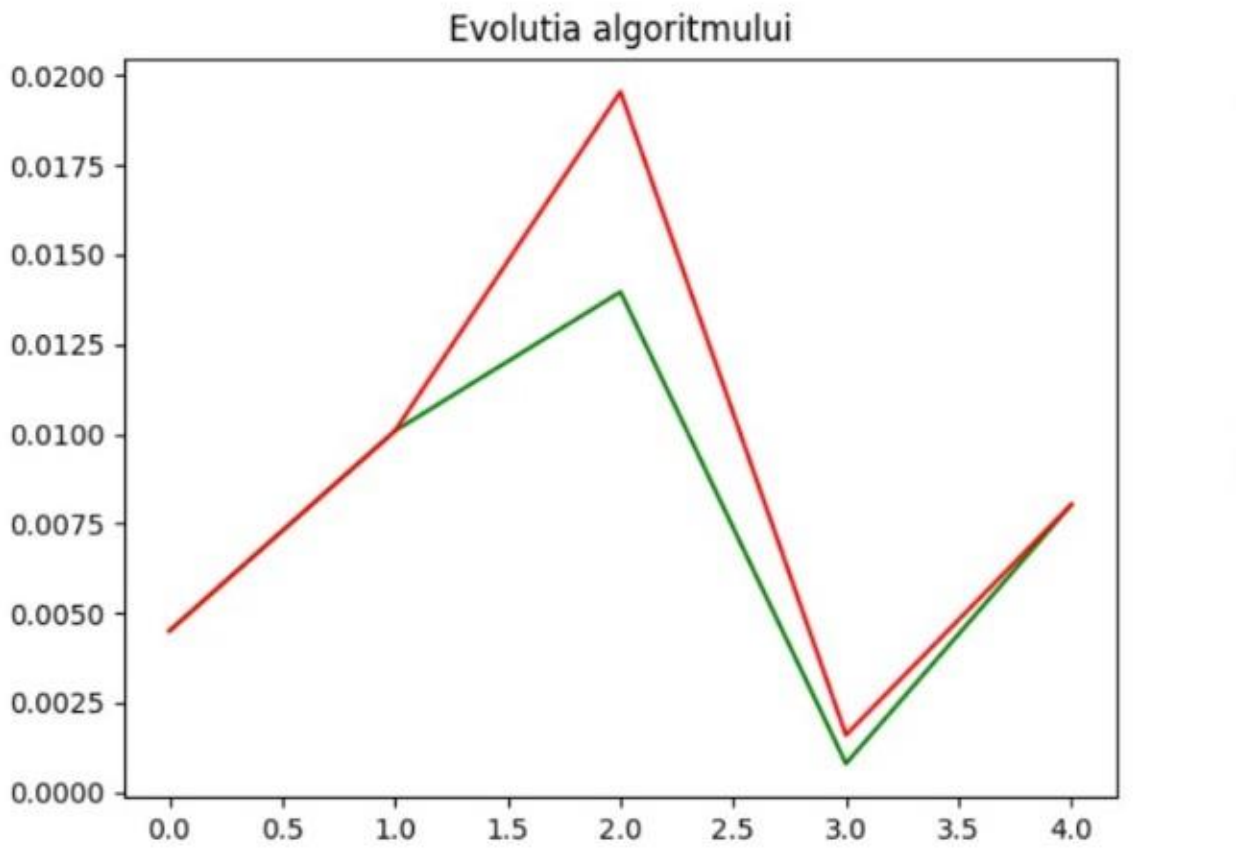
uniforma avem rezultatul 1.0149673208559307e-06, iar pentru gauss 1.0149673208559307e-06, care este mai mare, dar a ajuns tot la acelasi numar de zecimale.

Dintre cele 2 incrucisari, cea convexa simpla a dat cele mai bune rezultate, iar dintre cele 2 mutatii, cea uniforma.

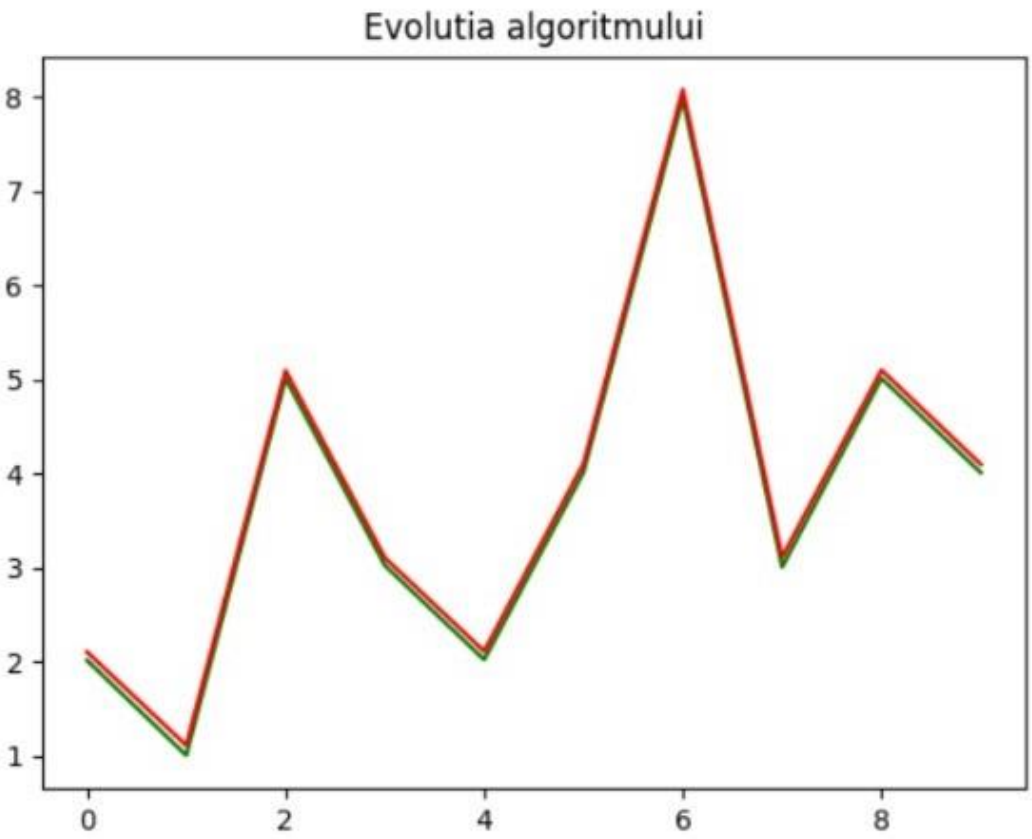
Grafice:

Rosu – cei mai slabi indivizi

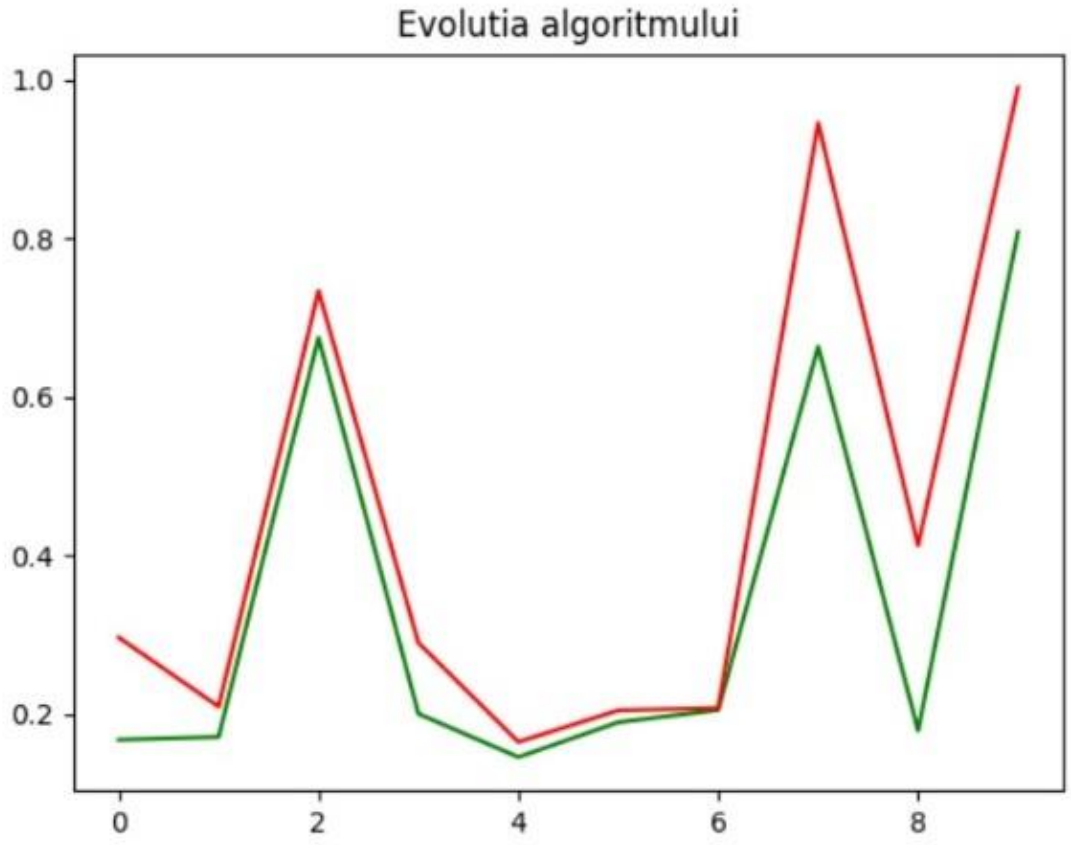
Verde – cei mai buni indivizi



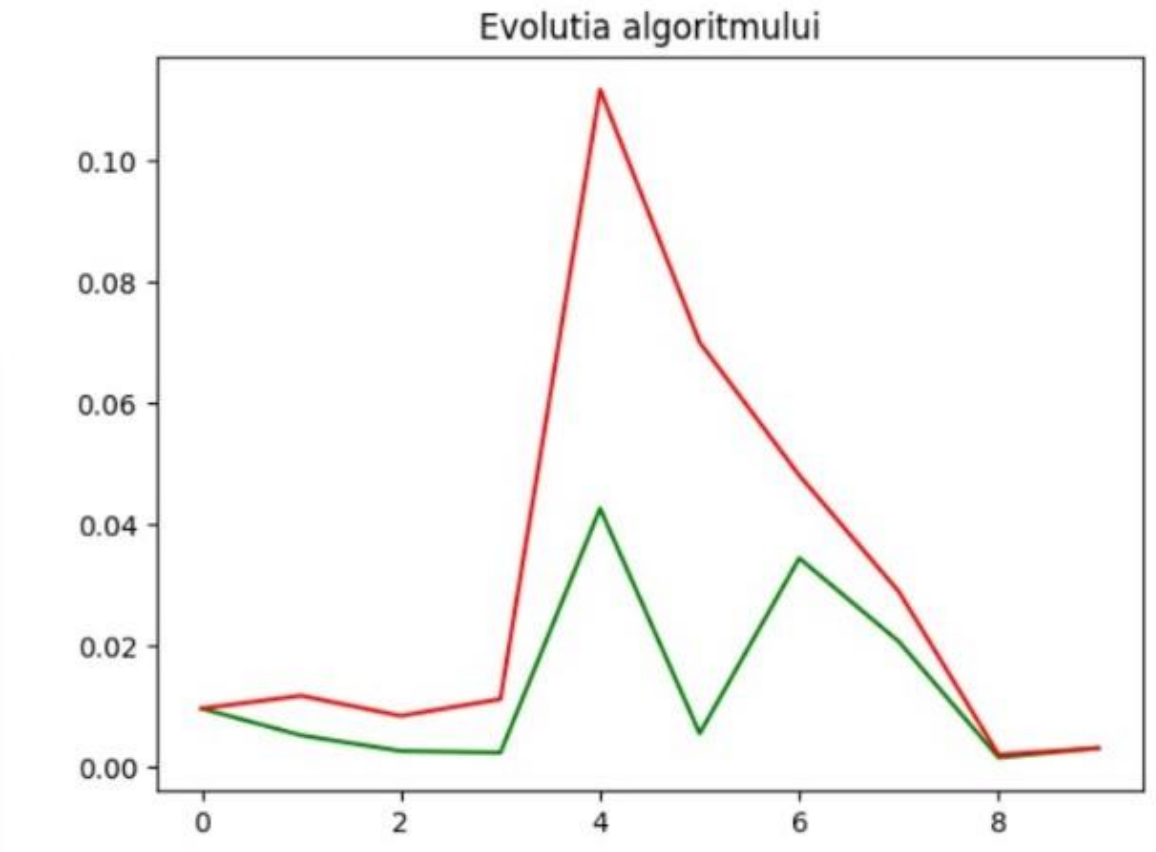
Incrucisare medie, mutatie gauss, dimensiune 5, 200 indivizi, 500 generatii.



Incrucisare convexa simpla, mutatie uniforma, dimensiune 10, 200 indivizi, 100 generatii



Incrucisare convexa simpla, mutatie uniforma, dimensiune 10, 200 indivizi, 100 generatii



Incrucisare convexa simpla, mutatie uniforma, dimensiune 10, 200 indivizi, 500 generatii

PSO

```
class Particle:
    def __init__(self, n):
        self.positions = [random.uniform(MIN_F, MAX_F) for _ in range(n)]
        self.pbest = 10000
        self.velocity = [random.uniform(MIN_V, MAX_V) for _ in range(n)]
        self.current_fitness = fitness(n, self.positions)
```

Pentru a defini o particula, am creat clasa Particle, care are ca atribute pozitia, personal best, viteza si fitness-ul. Pozitia si viteza sunt initializate la crearea obiectului. Pozitia ia valori intre

-5.12 si 5.12, iar viteza ia valori cuprinse intre -10% si +10% din domeniu.

Clasa Particle are metodele:

```
def modify_velocity(self, gbest, w, c1, c2):
    v = self.velocity.copy()
    for i in range(0, len(v)):
```

```
        v[i] = w * self.velocity[i] + c1 * random.random() * (self.pbest[i] -
self.positions[i]) + c2 * random.random() * (gbest[i] - self.positions[i])
        if MIN_V <= v[i] <= MAX_V:
            self.velocity[i] = v[i]
```

Metoda aceasta modifica viteza prin formula data in curs.

```
def modify_position(self):
    for i in range(0, len(self.positions)):
        if MIN_F <= self.positions[i] + self.velocity[i] <= MAX_F:
            self.positions[i] = self.positions[i] + self.velocity[i]
    self.current_fitness = fitness(len(self.positions), self.positions)
```

Metoda aceasta modifica pozitia particulei, verificand ca valorile nu ies din domeniu, iar apoi actualizeaza fitness-ul.

```
def update_pbest(self):
    if self.current_fitness < self.pbest:
        self.pbest = self.current_fitness
```

Actualizeaza personal best cu fitness-ul curent, daca fitness-ul curent este mai mic decat personal best.

Tabel de valori pentru Swarm Intelligence:

Executii	Iteratii	Dimensiune	Nr particule	w	C1	C2	General best	Medie general best	Timpul mediu de executie
10	50	5	10	1	2	2	26.78	45.22	1619471173.74
	50	5	50	1	2	2	16.59	27.15	1619471199.99
	350	5	50	1	2	2	23.18	27.83	1619471241.64
	350	5	70	1	2	2	19.04	28.03	1619471488.54
	50	10	10	1	2	2	78.34	111.66	1619471637.21
	350	10	10	1	2	2	85.49	110.06	1619472396.28
	350	10	50	1	2	2	82.06	92.24	1619472695.35
	350	10	70	1	2	2	78.68	92.98	1619472758.55
	350	10	70	0.75	2	2	77.06	87.35	1619503686.36
	350	10	70	0.75	2	2	75.49	86.33	1619504108.60

	350	10	70	0.75	1	3	83.55	94.52	1619504301.28
	350	10	70	0.75	1	3	72.50	87.84	1619505072.57
	700	10	70	0.75	1	3	92.58	92.58	1619505470.88
	700	10	70	0.75	3	1	82.99	94.01	1619505821.34
	700	10	70	0.75	3	1	75.32	88.91	1619506187.34
	350	5	50	0.75	3	1	17.21	24.81	1619506324.82
	350	5	70	0.75	3	1	20.80	27.28	1619506487.60
	350	5	70	0.75	1	3	13.61	25.42	1619506650.25
	350	5	70	0.75	1	3	18.07	28.56	1619506725.72
	350	5	70	0.75	2	2	14.14	25.92	1619506841.64
	350	5	70	0.75	2	2	16.52	25.70	1619506926.81
	700	5	70	0.75	2	2	22.71	28.18	1619507061.47
	700	5	70	0.5	2	2	8.76	27.66	1619507429.57
	700	5	70	0.5	2	1	18.84	28.36	1619507678.95

Observatii:

Pentru Swarm Intelligence nu am obtinut rezultate bune ca in algoritmul evolutiv. Cea mai buna valoare generala este 8.76, obtinuta pentru w = 0.5, c1 = 2 si c2 = 2. Cea mai buna medie a valorilor general best a fost obtinuta pentru w = 0.75, c1 = 3 , c2 = 1. Pentru c1 = 2 si c2 = 2 rezultatele au fost mai bune decat pentru alte valori ale acestor parametrii.