Laborator 4

Functia Schwefel

Algoritm evolutiv pt minimizarea functiei Schwefel

Functia de fitness

```
def fitness(n, x):
    f = 0
    for i in range(n):
        print("val x", x[i])
        c = math.sqrt(abs(x[i]))
        f = (-x[i]) * math.sin(c)
    print("functia f", f)
    return f
```

Functia de selectie flosita si in laboratorul trecut

```
def selectieTurnir(pop, popSize):
    sample = np.random.default_rng().choice(popSize, 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
```

Functia de calcul a mediei

```
def mediePop(pop, n):
    sum = 0.0

   for i in range(0, len(pop)):

        print("normal pop ", pop[i])
         print("fitnss medie1", fitness(n, pop[i]))

        sum = sum + fitness(n, pop[i])

   medie = sum/len(pop)
    print("best medie din fct", medie)
    return medie
```

Functia de incrucisare convexa

```
def incrucisareConvexaSimpla(x, y, n):
    pos = np.random.randint(n)
    alpha = random.random()
    print(pos)
    print(alpha)
    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]

        if x[i] < MIN_INT:
            x[i] = MIN_INT
        if x[i] > MAX_INT:
            x[i] = MAX_INT
        if y[i] < MIN_INT:
            y[i] = MIN_INT
        if y[i] > MAX_INT:
            y[i] = MAX_INT

    return x, y
```

calculeaza valorile dupa formula de mai sus, in functie de alpha luat random

Functia de incrucisare medie

```
def incrucisareMedie(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

    if x[i[0]] < MIN_INT:
        x[i[0]] = MIN_INT
    if x[i[0]] > MAX_INT:
        x[i[0]] = MAX_INT
    if y[i[0]] < MIN_INT:
        y[i[0]] = MIN_INT
    if y[i[0]] > MAX_INT:
        y[i[0]] = MAX_INT

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

    if x[i[1]] < MIN_INT:
        x[i[1]] = MIN_INT
    if x[i[1]] > MAX_INT:
        x[i[1]] = MAX_INT
    if y[i[1]] < MIN_INT:
```

```
            y[i[1]] = MIN_INT
        if y[i[1]] > MAX_INT:
            y[i[1]] = MAX_INT
```

Returneaza 2 copii din cei 2 parinti, verificand daca rezultatele fac parte din domeniu

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 MIN_INT <= x[i] + n <= MAX_INT:
            x[i] += n
```

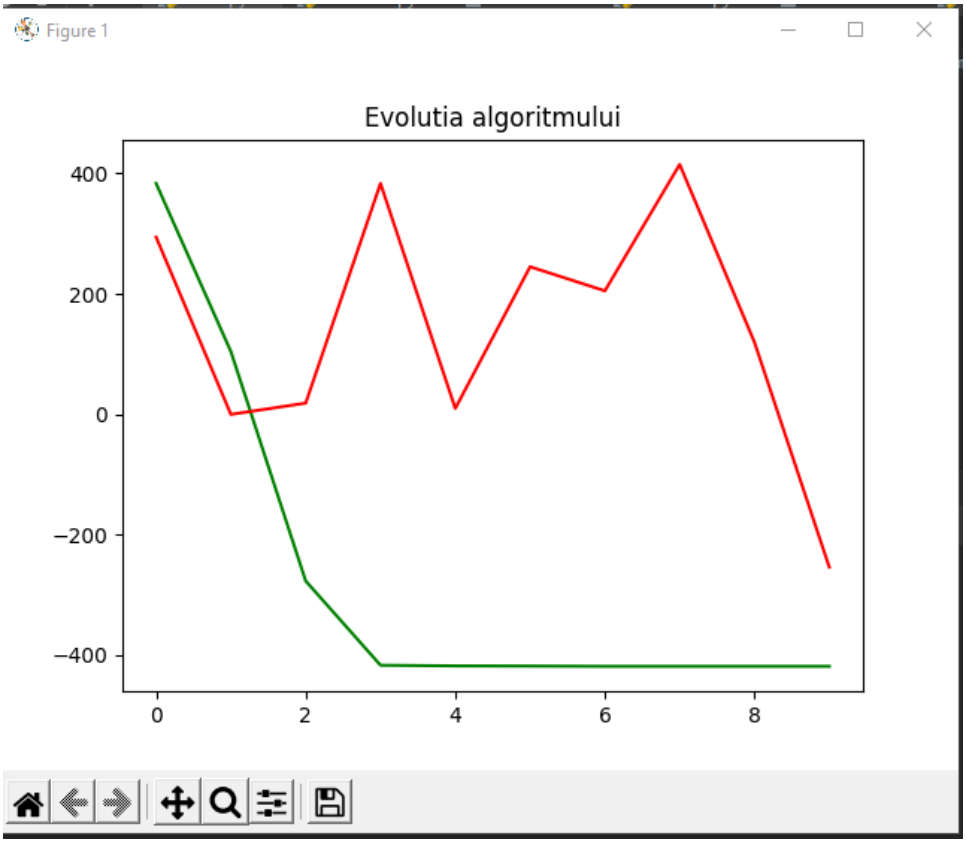
modificam valoara adunand rezultatul functiei gaussiene, daca rezultatul functiei adunat la valloarea individului nu iese din domeniul functiei

Functia principala, este aceeasi functie de agoritm evolutiv folosita si in laboratorul 3, unde initializam populatia, initializam parintii, pe care ii incrucisam pentru a obtine copii, pe care apoi aplicam functiile de mutatie pentru a obtine copii mutanti, din care luam cele mai bune solutii

Executii	Incrucisare	Mutatie	Dimensiune	Nr pop	Nr generatii	average	Timp	Best sol	Best average
10	convexa	gauss	5	10	10				
	convexa	uniforma	5	10	10	-239.472247475799 5	103.96891880 0354	-418.97267414 9225	-408.57786857 97052
	convexa	gauss	5	10	50				
	convexa	uniforma	5	10	50	-272.280222098561	211.90151047 706604	-418.93869364 50739	-312.01957399 78939
	medie	gauss	5	10	10				
	medie	uniforma	5	10	10	-165.802941639749 58		-394.03712401 374986	-361.32220590 34898
	medie	gauss	5	10	50				
	medie	uniforma	5	10	50	-235.507127525081 24	216.10021567 344666	-417.10771415 28115	-329.56383125 07196
	convexa	gauss	10						

	convexa	uniforma	10	10	10	-174.311910994182 03	66.014678239 82239	-417.68373127 05427	-252.13677101 954312
	convexa	gauss	10						
	convexa	uniforma	10	10	50	-272.960117536952 9	436.35254073 143005	-418.89391919 121016	-292.13513090 91334
	medie	uniforma	10	10	10	-283.550663044366	56.526966094 9707	-414.61767578 430903	-359.16691589 635303
	medie	gauss	10	10	10				
	medie	gauss	10	10	50				
	medie	uniforma	10	10	50	-265.638312632807 4	349.34511184 69238	-412.21959894 74075	-343.03884052 30049

Observam ca avem cea mai buna solutie cand aplicam mutatia uniforma si incrucisarea convexa. Cand folosim mutatia uniforma si incrucisarea convexa cu dim 10, pop 10 si gen 50ne apropriem cel mai tare de optimul global, f(x)=- n·418.9829; x(i)=420.968 7



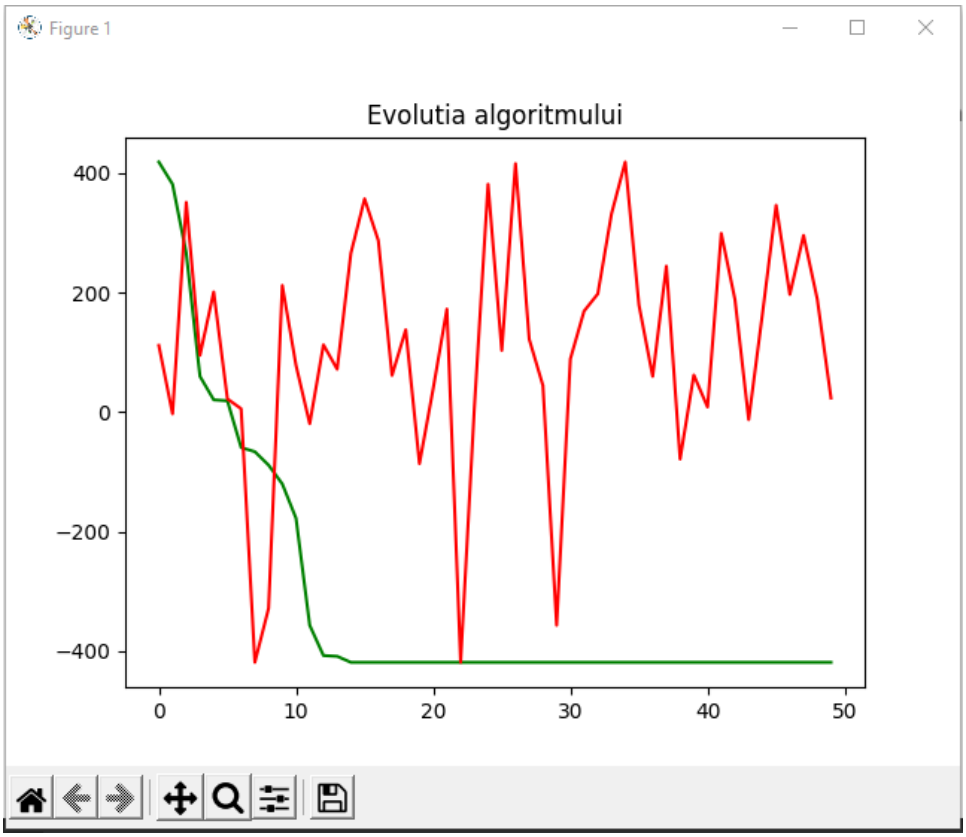


Figure 1
Evolutia algoritmului
400
200
0
−200
−400
0
10
20
30
40
50

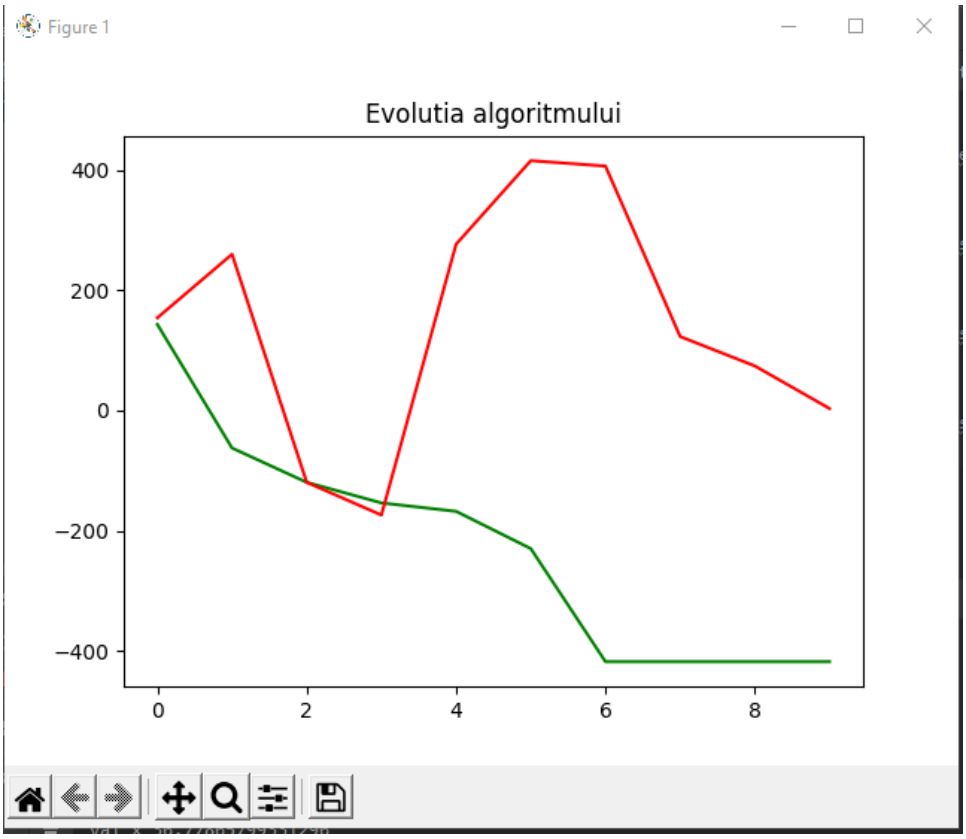


Figure 1
Evolutia algoritmului
400
200
0
−200
−400
0
2
4
6
8

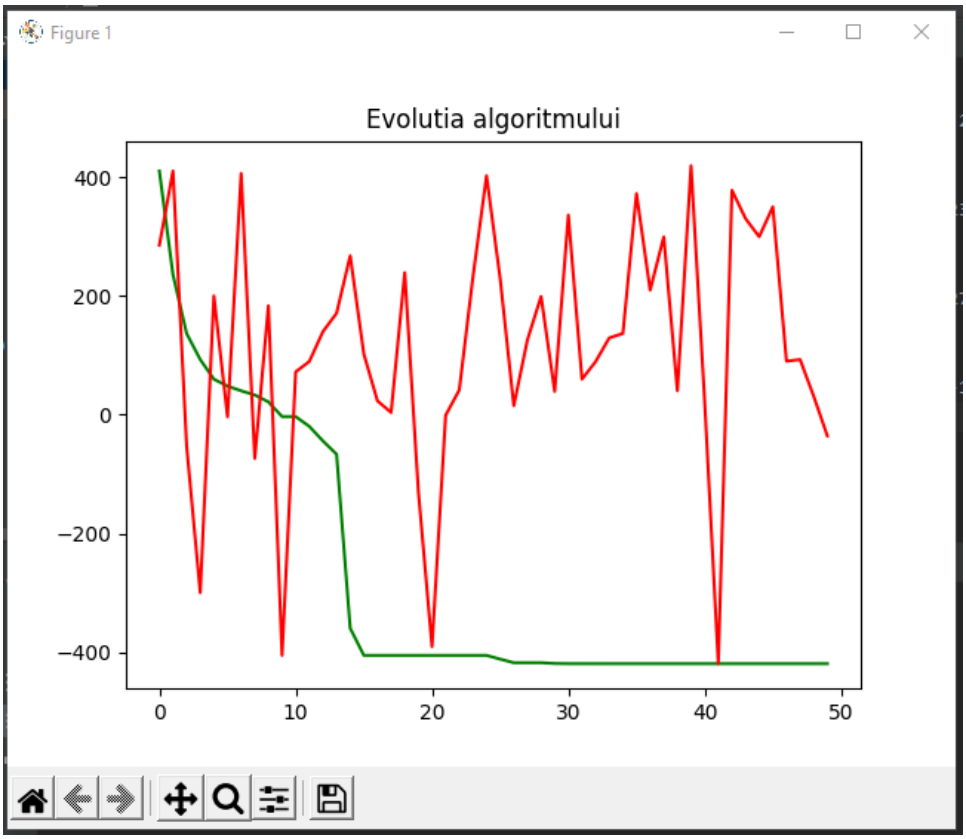


Figure 1
Evolutia algoritmului
400
200
0
−200
−400
0
10
20
30
40
50

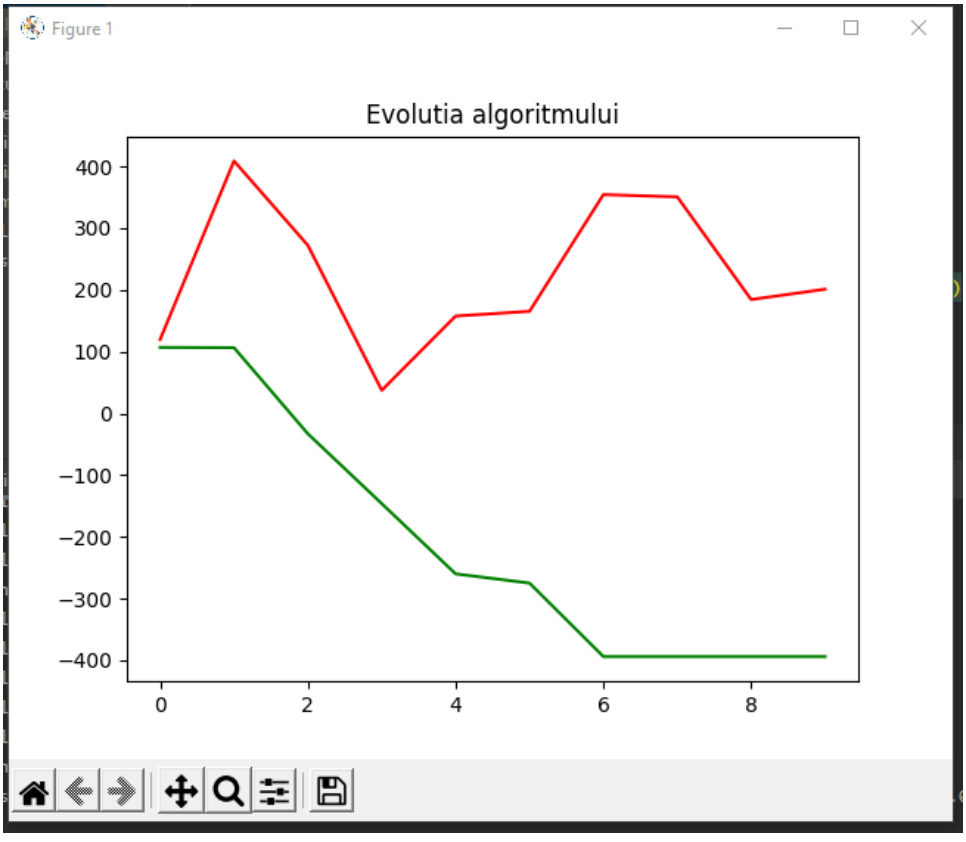


Figure 1
Evolutia algoritmului
400
300
200
100
0
−100
−200
−300
−400
0
2
4
6
8

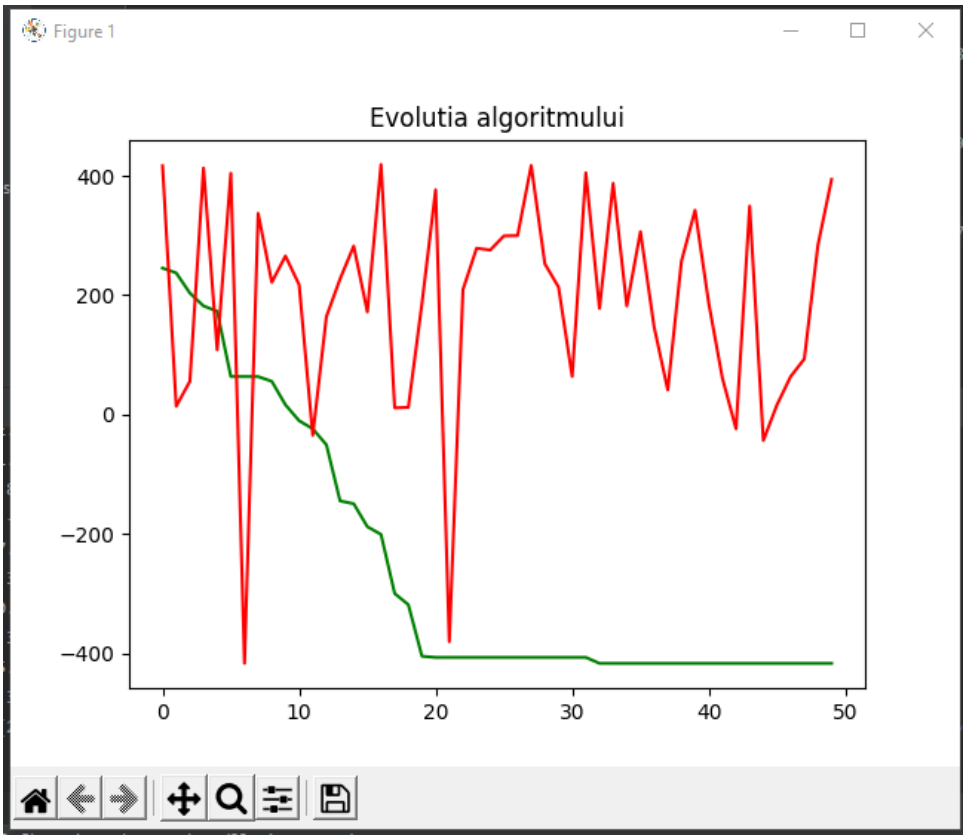


Figure 1
Evolutia algoritmului
400
200
0
−200
−400
0
10
20
30
40
50

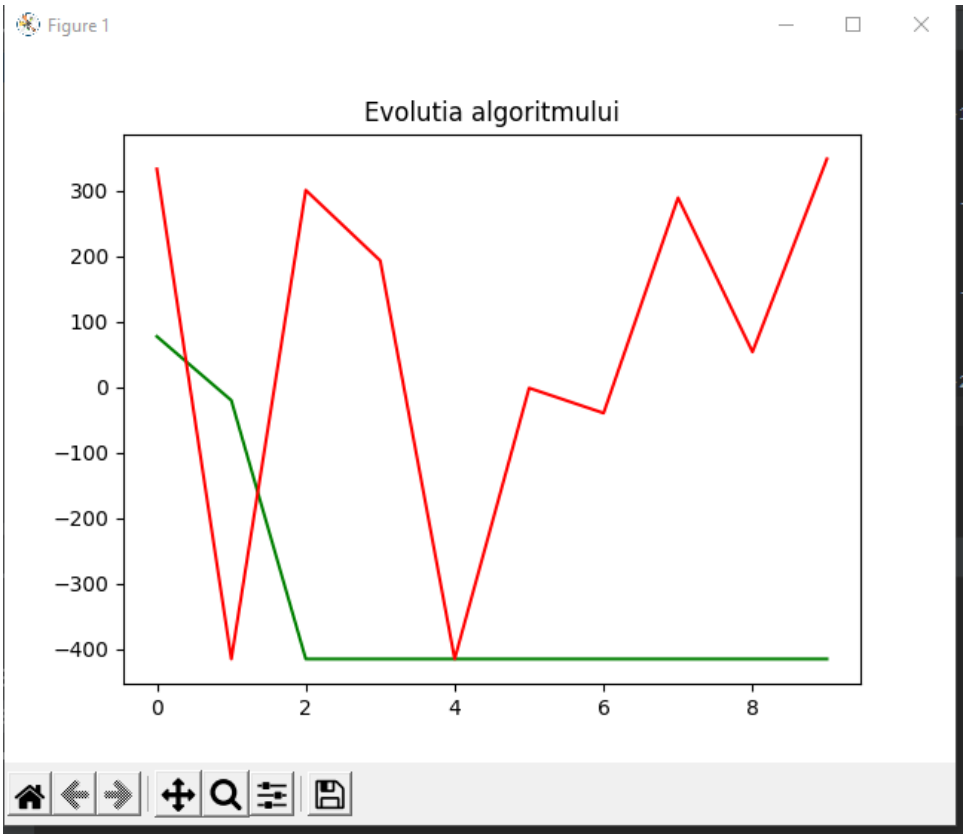
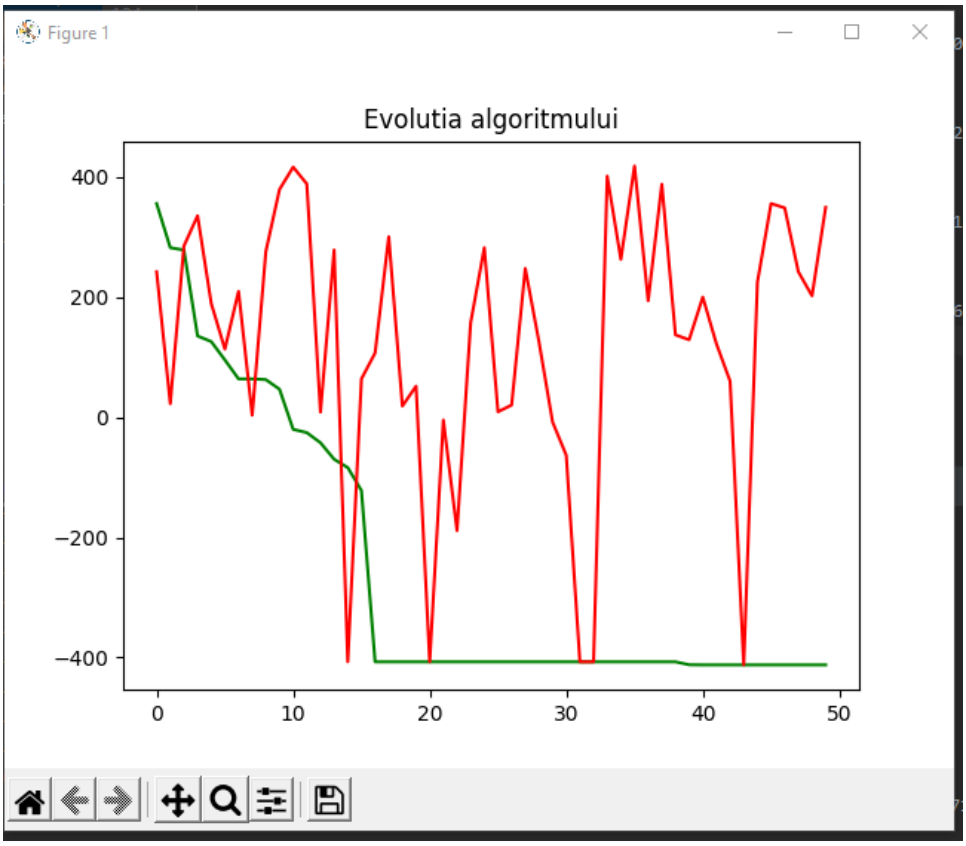


Figure 1
Evolutia algoritmului
300
200
100
0
−100
−200
−300
−400
0
2
4
6
8



Swarm optimisation pso

Avem o clasa Particula, pentru a definii o particula. Avem atributele: pozitia, personal best, viteza fitness

```
class Particula:
    def __init__(self, n):
        self.viteza = [random.uniform(MIN_INT, MAX_INT) for _ in range(n)]
        self.pozitie = [random.uniform(MIN_INT, MAX_INT) for _ in range(n)]
        self.currentFitness = fitness(n, self.pozitie)
        self.pbest = 10000
```

```
Domeniul functiei

MIN_INT: float = -500
MAX_INT: float = 500
```

```
Avem metode de modificare a vitezei dupa formula din curs, de modificare a pozitiei si
metoda de modificare personal best cu fitensul curent daca acesta e mai mic

def modificaViteza(self, gbest, w, c1, c2):
    for i in range(0, len(self.viteza)):
        self.viteza[i] = w * self.viteza[i] + c1 * random.random() * (self.pbest -
self.pozitie[i]) + c2 * random.random() * (gbest - self.pozitie[i])
```

```
def modificaPozitie(self):
    for i in range(0, len(self.pozitie)):
        self.pozitie[i] = self.pozitie[i] + self.viteza[i]

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

execut ii	dim	iterat ii	Nr part	w	c1	c2	best	medie	timp
10	5	10	10	1	2	2	-19493.934692795396	-8338.968762419625	22.642580270767212
	5	10	50				-143970.9412868937	-67797.19647831551	22.393128395080566
	5	50	50				-3690624818716.381	-1241627472140.1501	26.624576091766357
	5	50	100				-29048348380857.816	-13122847028407.295	38.31524038314819
	10	10	10				-11064.093398891122	-7122.421544390143	16.062586545944214
	10	50	50				-2332565076324.5366	-639505955523.675	30.331240177 15454}
	10	50	100				-45331082952522.2	-13867904063769.355	60.88688921928406

Observam ca cu cat crestem dimensiunea, nr de iteratii si nr de particule incepem sa avem rezultate din ce in ce mai bune