UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației
SPECIALIZAREA:Tehnologia informației

Optimizarea funcției Griewank utilizând optimizarea de tip roi de particule

Proiect la disciplina
Inteligență artificială

Student
Ciobanu Denis-Marian

Iași, 2020

Cuprins

Capitolul 1. Introducere. Aspecte teoretice privind algoritmul.

Optmizarea de tip roi de particule (Particle Swarm Optimization) reprezinta o metoda bazata pe indivizi care imita comportamentul stolurilor de pasari sau roiurilor de insecte.

Aceasta metoda consta in cautarea solutiei optime prin intermediul unor agenti, denumiti particule, a caror traiectorie este ajustata in functie de componente stohastice si deterministe.

Fiecare particula este caracterizata de:

- x_i: pozitia curenta
- v_i: viteza curenta
- y_i: cea mai buna pozitie personala
- $\hat{y}_i$: cea mai buna pozitie a vecinatatii

Fiecare particula este influentata, la un anumit moment de timp, de cea mai buna pozitie a ei, dar si de cea mai buna pozitie a grupului, dar, in acelasi timp, tinde sa se miste aleator. La fiecare iteratie fiecare particula isi actualizeaza pozitia in functie de viteza, dupa cum urmeaza in formula:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1)$$

Viteza se actualizeaza conform relatiei:

$$v_{i,j}(t+1) = w \cdot v_{i,j}(t) + c_1 \cdot r_{1,j}(t) \cdot (y_{i,j}(t) - x_{i,j}(t)) + c_2 \cdot r_{2,j}(t) \cdot (\hat{y}_j(t) - x_{i,j}(t))$$

ponderea inerției — componenta cognitivă — componenta socială

In relatia de mai sus r1 si r2 reprezinta numere aleatorii in (0, 1).

Pentru fiecare dintre particule, se evalueaza functia obiectiv a acesteia, f(x_i).

Acest ciclu se repeta pana este satisfacut un criteriu de convergenta, de obicei atingerea unui numar prestabilit de iteratii.

La initializare se alege un numar de particule, pentru fiecare dintre ele initializandu-se aleatoriu pozitiile x_i, precum si vitezele v_i (sau, mai rar, doar acestea din urma, cu 0).

Parametri:

- Ponderea inertiei w: defineste compromisul intre explorare (cautam si alte solutii promitatoare pe langa cea care exista; cu alte cuvinte se executa o cautare globala) si exploatare (se cauta imbunatatirea unei solutii de care algoritmul deja dispune la un moment dat; cu alte cuvinte executam o cautare locala)
- Constantele de acceleratie c1 si c2, au rol in a asigura convergenta algoritmului, in cazul in care se respecta relatia: ((c1 + c2)/ 2) – 1 < w
- Vitezele sunt limitate la o valoare Vmax

Capitolul 2. Descrierea problemei considerate

Problema consta in optimizarea funcției Griewank folosind metoda optimizarii de tip roi de particule (Particle Swarm Optimization).

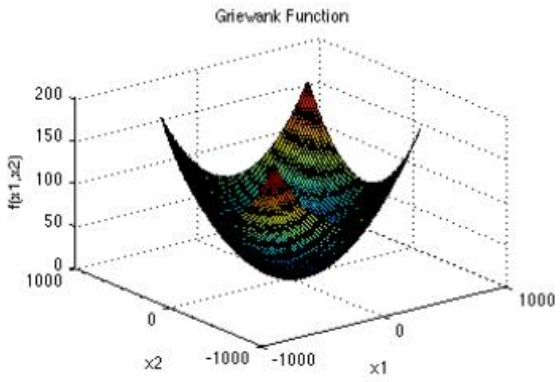
Ecuatia functiei in discutie:

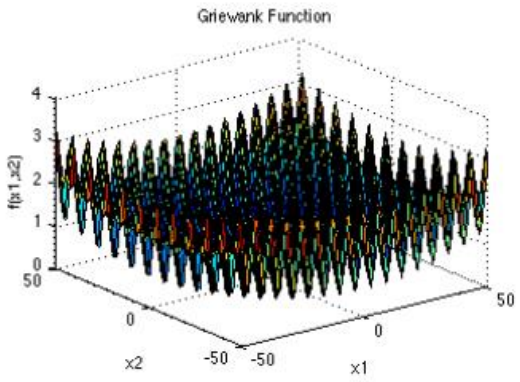
$$f(\mathbf{x}) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

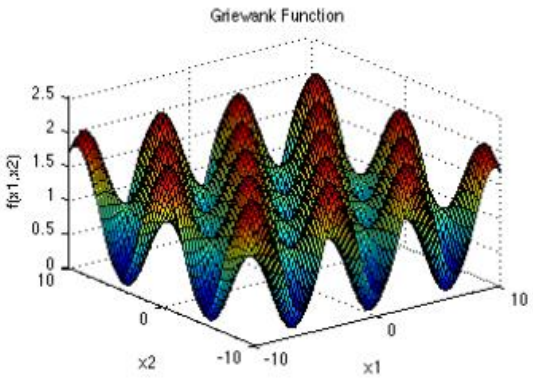
Ea este caracterizata prin doua valori de intrare, si anume:

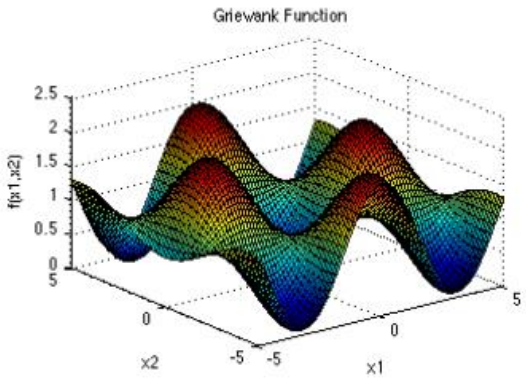
- Dimensiunea problemei: d
- Valorile de intrare: x_i ; acestea pot lua valori din intervalul [-600, 600], pentru i = 1, …, d

Functia Griewank contine multe minime locale larg raspandite, care sunt distribuite regulat. Complexitatea functiei se poate observa din graficele de mai jos:









Minimul global al functiei:

$$f(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (0, \ldots, 0)$$

2.1. Modalitatea de rezolvare

Pentru rezolvarea problemei am folosit limbajul de programare Java si mediul de dezvoltare NetBeans pentru a realiza o aplicatie desktop. Aplicatia va furniza utilizatorului posibilitatea de a introduce parametrii si de a vedea rezultatele.

2.2 Prezentarea coduui

Aplicatia contine 6 module.

Clasa **Parameters** este descrisa de proprietatile ce reprezinta parametrii pe care utilizatorul trebuie sa îi introduca de la tastatura. Aceste proprietati sunt: numarul de particule (nrOfParticles), dimensiunea problemei (d), limita maxima pentru viteza particulelor (maxVelocity), ponderea inertiei (w), constantele de acceleratie (c1, c2) si numarul de iteratii (nrOfIterations).

Clasa **Particle** este caracterizata de urmatoarele proprietati: pozitie (position), functia obectiv caracteristica particulei (cost), viteza particulei (velocity), precum si pozitia personala particulei care este retinuta prin intermediul variabilei personalBest (care reprezinta stadiul particulei cu pozitia cea mai buna; asadar personalBest este o variabila de tip Particle).

Variabilele velocity si position sunt vectori cu un numar de elemente egal cu dimensiunea d (pentru fiecare dintre cele d dimensiuni particular are o anumita pozitie si o anumita viteza).

Clasa **Functii** contine functia pentru limitarea valorilor dintr-un anumit interval.

Clasa **Problem** contine functia griewank() ce descrie in limbaj Java expresia teoretica a functiei Griewank, functia getDomain(), folosita la returnarea domeniului pentru care pozitiile particulei isi pot lua valorile (adica intervalul [-600, 600]), si functia getRandomDoubleBetweenRange() care este folosita pentru a returna valori aleatoare dintr-un anumit interval.

Clasa **PSO** implementeaza algoritmul propriu-zis, prin functia optimize().

Algoritmul incepe cu initializarea roiului (swarm) cu valori aleatoare si cu stabilirea unui "socialBest".

```
Particle[] swarm = this.initSwarm(problem, parameters);
Particle socialBest = getSocialBest(problem, swarm);
```

Pentru prima functie de mai sus se utilizeaza functiile din clasa Problem pentru returnarea valorilor aleatoare. Pentru obtinerea celei mai bune pozitii din vecinatate (socialBest) se itereaza prin particulele roiului si se retine particula care minimizeaza solutia problemei (particula cu functia obiectiv –cost- cea mai mica).

```
for(int i=1; i<swarm.length; ++i){

    if (min > swarm[i].cost){
        min = swarm[i].cost;
        index = i;
    }
}
```

Pentru fiecare dintre particulele roiului se actualizeaza viteza acesteia pentru fiecare dintre dimensiuni (variabila k itereaza prin dimensiunile particulei, in timp ce variabila j itereaza prin particulele roiului).

```
swarm[j].velocity[k] = parameters.w * swarm[j].velocity[k] +
        parameters.c1 * r1 * (swarm[j].personalBest.position[k] - swarm[j].position[k]) +
        parameters.c2 * r2 * (socialBest.position[k] - swarm[j].position[k]);
```

Pentru aceasta se limiteaza potentialele depasiri ale vitezei in afara domeniului impus de utilizator prin intermediul variabilei maxVelocity:

```
swarm[j].velocity[k] = new Functii().limit(swarm[j].velocity[k], -parameters.maxVelocity, parameters.maxVelocity);
```

Apoi se calculeaza pozitia particulei si se limiteaza valorile sale in intervalul [-600,600]:

```
swarm[j].position[k] = swarm[j].position[k] + swarm[j].velocity[k];

swarm[j].position[k] = new Functii().limit(swarm[j].position[k],
            problem.getDomain(k)[0], problem.getDomain(k)[1]);
```

Pentru aceste particule se calculeaza functia obiectiv:

```
swarm[j].cost = problem.griewank(parameters.d ,swarm[j].position);
```

In cele din urma se compara functia obiectiv rezultata cu cea mai buna pozitie personala, iar apoi, daca este cazul, se compara cu cea mai buna pozitia a grupului.

```
if(swarm[j].cost <= swarm[j].personalBest.cost) {

    swarm[j].personalBest = new Particle(swarm[j].position,swarm[j].cost, swarm[j].velocity);
    if(swarm[j].cost < socialBest.cost) {
        socialBest = new Particle(swarm[j].position,swarm[j].cost, swarm[j].velocity);
}
```

Tot acest ciclu se repeta pentru fiecare dintre particule de un numar de ori egal cu valoarea variabilei “nrOfIterations”.

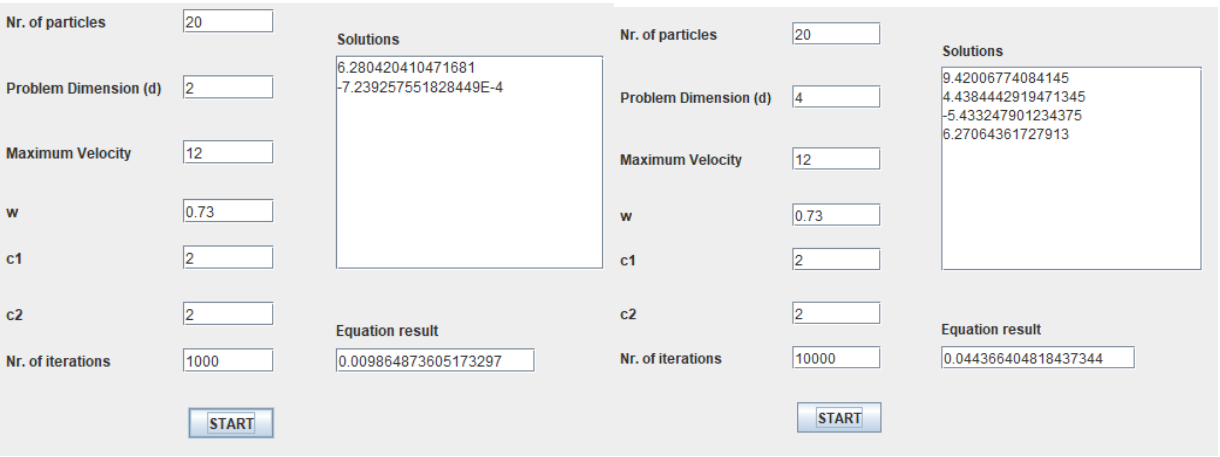
Pentru afisarea interfetei cu utilizatorul s-a folosit JFrame specific limbajului Java:

```
static JFrame frame = new Window();
```

2.3 Prezentarea rezultatelor

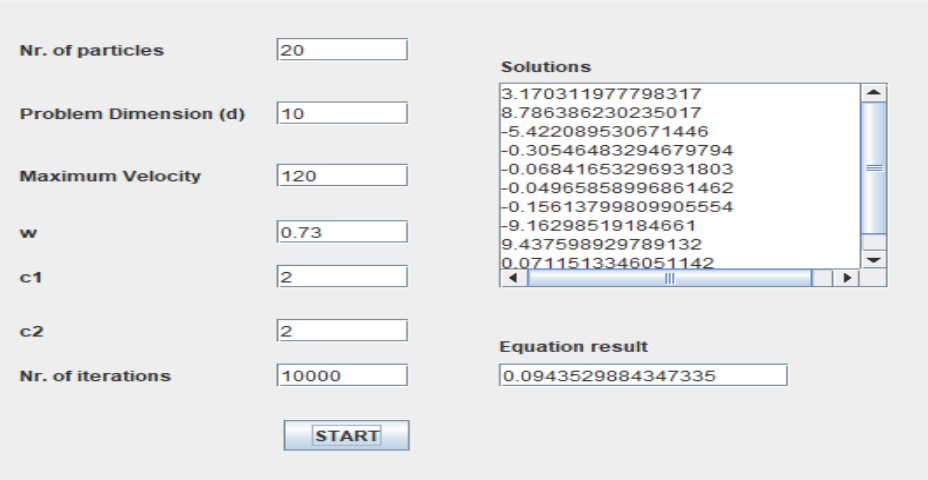
Respectand recomandarile din curs, prima rulare a programului a fost facuta pentru c1 = c2 = 2, ponderea inertiei o valoare constanta din intervalul [0.4, 0.9], si anume 0.73. Dimensiunea populatiei, avand in vedere dificultatea nu tocmai accentuate a problemei, a fost aleasa ca o multime de 20 de indivizi (particule). Pentru viteza maxima a fost aleasa o valoare de 10% din dimensiunea intervalului in care o particular poate lua valori (intervalul este [-600, 600]), in acest caz 12.

Rezultatele primelor rulari se observa in urmatoarea imagine:



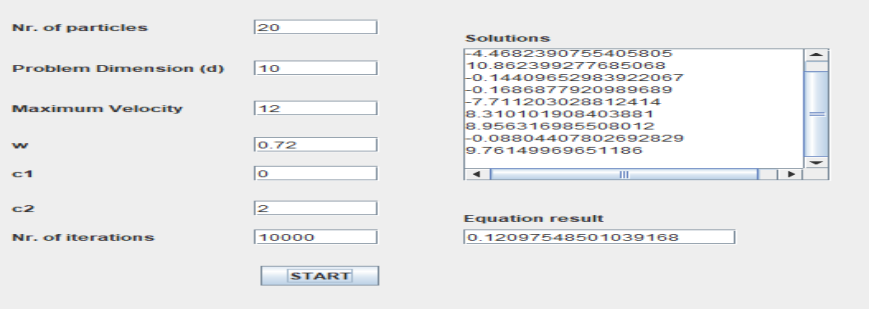
Dupa cum se observa, pentru fiecare dintre dimensiuni, solutiile se regasesc in intervalul [-600, 600], iar rezultatul este apropiat de cel asteptat (de zero).

In exemplul urmator am crescut considerabil dimensiunea problemei la 10:

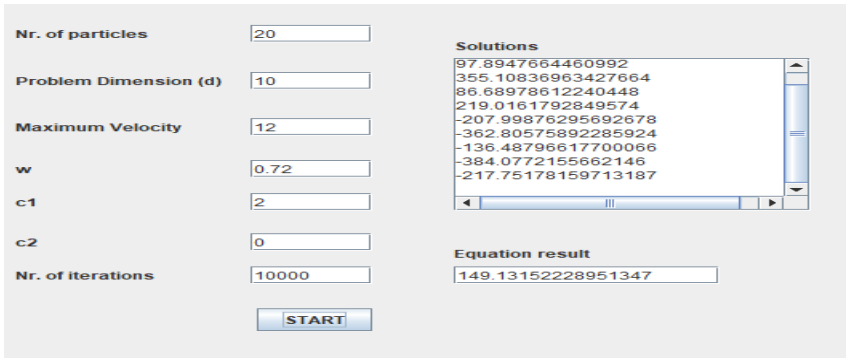


Pentru a verifica impactul constantelor de acceleratie le-am atribuit pe rand valoare nula. Daca in cazul zerorizarii pentru c1 impactul nu a fost major, desi factorul exploatare a avut de suferit, in cazul c2, anulandu-se factorul explorare, implicit cautarea globala si deci axarea exclusiv asupra optimizarii solutiilor locale, performantele au de suferit consistent.

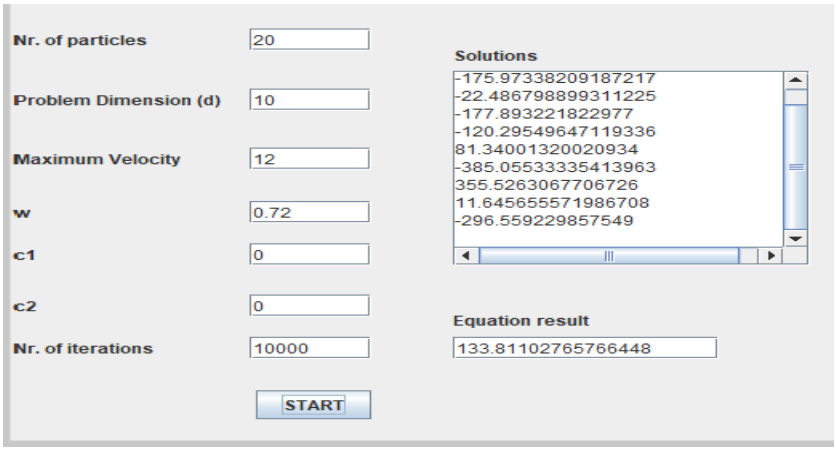
Cazul c1 = 0:



Cazul c2=0:



Cazul c1 = 0 si c2 = 0:



Concluzii

In urma rularilor am observant ca optimizarea de tip roi de particule are pe deoparte avantaje, dar si dezavantaje.

Am observat in primul rand viteza cu care algoritmul gaseste solutiile problemei. PSO nu este influentata in mare masura de numarul de indivizi.

Un dezavantaj il consta faptul ca acest algoritm are nevoie de o selectie prealabila a multiplilor parametri care influenteaza in mare masura performantele executiei. Asadar trebuie realizat un studiu preliminar al problemei cu privire la selectia corecta a parametrilor care sa asigure functionarea optima a algoritmului.

Bibliografie

Functia Griewank:

https://www.sfu.ca/~ssurjano/griewank.html

https://en.wikipedia.org/wiki/Griewank_function

Particle Swarm Optimization:

https://moodle.ac.tuiasi.ro/pluginfile.php/13864/mod_resource/content/4/IA05_Optimizare2.pdf

https://moodle.ac.tuiasi.ro/pluginfile.php/13906/mod_resource/content/3/SintezaIA05C.pdf

https://www.sciencedirect.com/topics/engineering/particle-swarm-optimization

https://en.wikipedia.org/wiki/Particle_swarm_optimization

https://www.youtube.com/watch?v=JhgDMAm-imI