Universitatea Tehnică ”Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare
Domeniul Calculatoare și Tehnologia Informației
Specializarea Tehnologia Informației

INTELIGENȚĂ ARTIFICIALĂ

Rezolvarea Ecuațiilor de Gradul X folosind algoritmul de Evoluție Diferențială

Studenți,
Păuleț Faustina-Cristina, Grupa 1411A
Roman Sorin-Valentin, Grupa 1411A

An universitar 2022-2023

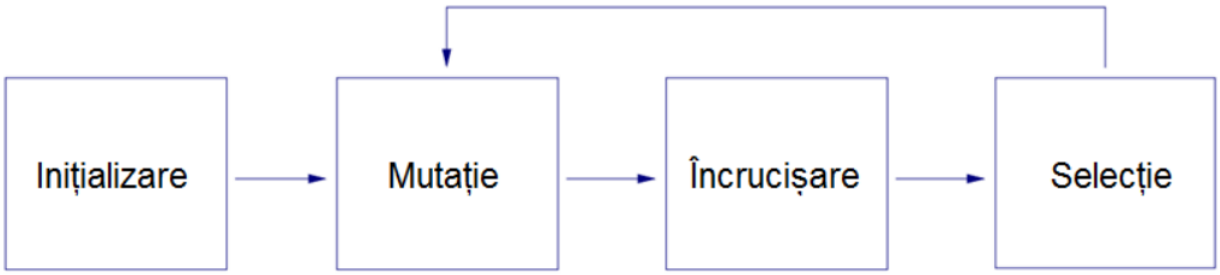
Cuprins

1.Introducere

Evoluția diferențială este o euristică de optimizare asemănătoare cu un algoritm evolutiv clasic.Are operatori de selecție, încrucișare, mutație. Caracteristică este operația de generare a noilor cromozomi, care implică adăugarea diferenței dintre doi cromozomi la al treilea și compararea cu al patrulea.

Algoritmii evolutivi sunt aplicați cu succes în proiectarea circuitelor digitale, a filtrelor dar și a unor structuri de calcul precum rețelele neuronale. Ca metode de estimare a parametrilor unor sisteme care optimizează anumite criterii, se aplică în diverse domenii din inginerie cum ar fi: proiectarea avioanelor, proiectarea reactoarelor chimice, proiectarea structurilor în construcții etc. Uneori nici nu se cunoaște o expresie analitică a funcțiilor care trebuie optimizate, ci acestea se estimează din simulări . În aceste cazuri, nu pot fi aplicate ușor metodele de optimizare diferențială.

Pentru a ghida căutarea către soluția problemei, asupra populației se aplică transformări specifice evoluției naturale:



- ***Selecția.*** Determină părinții care se vor reproduce pentru a forma următoarea generație. Indivizii mai adaptați din populație (care se apropie mai mult de soluția problemei) sunt favorizați, în sensul ca au mai multe șanse de reproducere;
- ***Încrucișarea.*** Pornind de la doi părinți, se generează copii;

- ***Mutația.*** Pentru a asigura diversitatea populației, se aplică transformări cu caracter aleatoriu asupra copiilor nou generați, permițând apariția unor trăsături care nu ar fi apărut în cadrul populației doar prin selecție și încrucișare.

2. Descrierea problemei considerate

Algoritmul de evoluție diferențială a fost folosit în rezolvarea ecuațiilor de gradul X. Programul care este scris în Visual Studio, folosind limbajul C#, dispune de o interfață care îl ajută pe utilizator să rezolve ecuații matematice de un anumit grad. Acest poate introduce dimensiunea populației, numărul generațiilor și numărul de gene.

Plecând de la numărul de indivizi introduși de utilizator se generează în mod aleatoriu gene, cu scopul de a acoperi un spațiu de căutare cât mai mare pentru soluția cea mai optimă. Folosind operația de încrucișare, spațiul de căutare se lărgește și în etapa următoare trec cele mai bune soluții.
Pentru a avea un rezultat cât mai bun, vor trece în etapa următoare doar soluțiile cele mai bune rezultate în urma Selecției. Indivizii noi care nu s-ar putea crea prin încrucișare, se obțin prin Mutație.

3. Structura aplicației

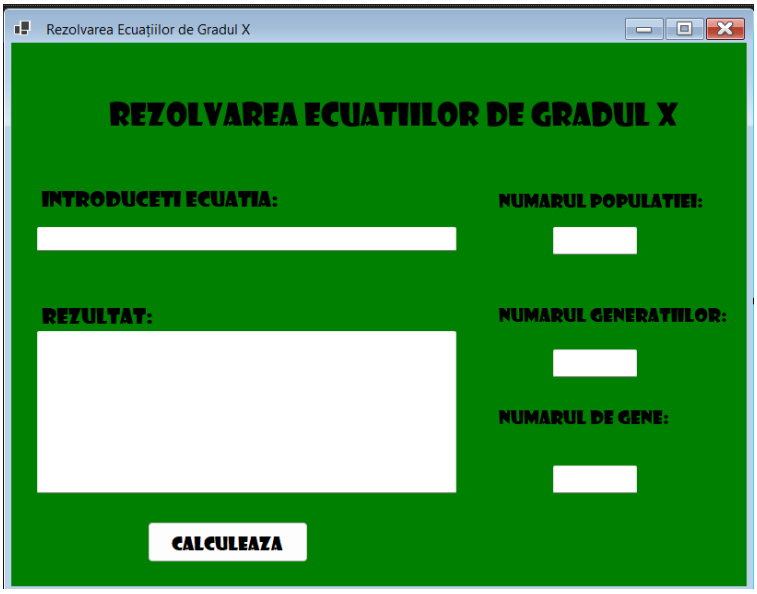
Proiectul conține următoarele clase:

- ***Chromosome.cs***
- ***EquationSolver.cs***
- ***Exceptii.cs***
- ***Rezolvare.cs***
- ***SolutieInvalida.cs***
- ***IProblem.cs***

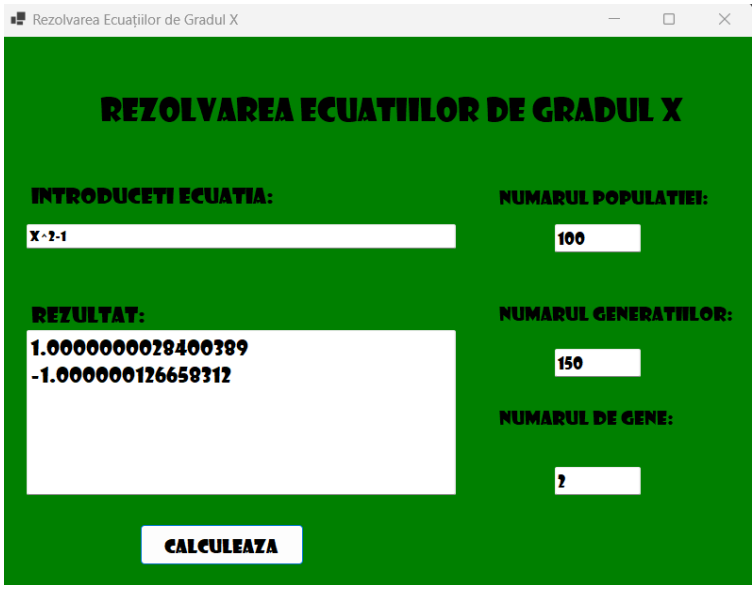
Modulul de Testare:

- ***TestProject***

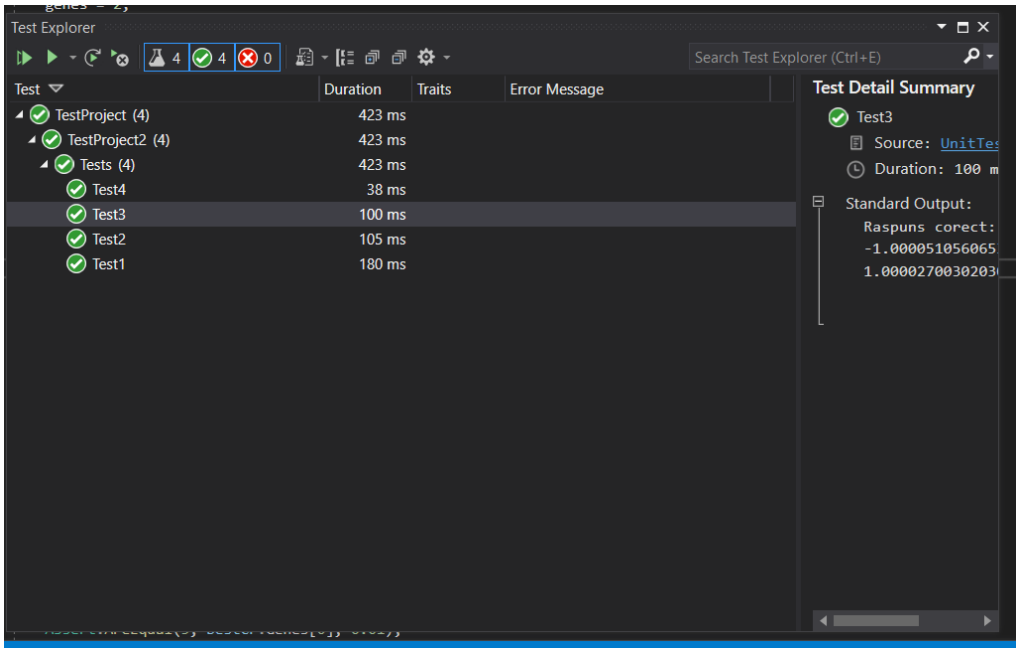
Interfața grafică:



Utilizatorul poate introduce ecuația care are gradul x,numărul de gene, poate alege numărul populației care trebuie să fie mai mare ca 2. Pentru a funcționa corect programul este necesar ca numărul de generații să fie mai mare ca 0.



Pentru a testa corectitudinea programului s-au realizat 4 teste.



Elementul cheie al algoritmul de evoluției diferențiale este funcția de Fitness. Datorită acestei funcții, după utilizarea operatorilor de mutație și încrucișare, vom putea determina calitatea unui nou individ în relație cu un individ dintr-o generație anterioară. Numai cei mai bine adaptați indivizi vor ajunge la următoarea generație.

```
3 references
public void ComputeFitness(Chromosome c)
{
    double sum = 0;

    if (c.Genes.Length > 0)
    {
        foreach (var gene in c.Genes)
        {
            sum += -Math.Abs(mathEquation.calculate(gene));
        }
        c.Fitness = sum;
    }
    else
    {
        c.Fitness = -Math.Abs(mathEquation.calculate(c.Genes[0]));
    }

}
```

În primă fază se inițializează populația care contine cromozomi

```
List<Chromosome> population = new List<Chromosome>();

// initializare
for (int i = 0; i < populationSize; i++)
{
    population.Add(p.MakeChromosome());
    p.ComputeFitness(population[i]);
}
```

În fiecare generație se adaugă diferența dintre doi cromozomi la al treilea și se compară cu al patrulea. O sa alegem cel mai bun cromozom și îl trecem în următoarea generație fără să sufere modificari. Repetăm acest proces până ajungem la soluția căutată.

```
for (int gen = 0; gen < maxGenerations; gen++)
{
    List<Chromosome> newPopulation = new List<Chromosome>();

    // mutatie
    foreach (Chromosome cr in population)
    {
        int ind1, ind2, ind3;

            ind1 = _rand.Next(0, population.Count);
            ind2 = _rand.Next(0, population.Count);
            ind3 = _rand.Next(0, population.Count);

        Chromosome cr1 = population[ind1];
        Chromosome cr2 = population[ind2];
        Chromosome cr3 = population[ind3];

        Chromosome individPotential = new Chromosome(cr1);
        int pctDivizare = _rand.Next(0, population.Count);
        for (int i = 0; i < individPotential.Genes.Length; i++)
        {
            // incrucisare
            if (_rand.NextDouble() < pc  || i == pctDivizare)
            {
                individPotential.Genes[i] = cr1.Genes[i] + pm * (cr2.Genes[i] - cr3.Genes[i]);
            }
            else
            {
                individPotential.Genes[i] = cr.Genes[i];
            }
```

```
            }

            // selectie
            p.ComputeFitness(individPotential);
            if (individPotential.Fitness >= cr.Fitness)
            {
                newPopulation.Add(individPotential);
            }
            else
            {
                newPopulation.Add(cr);
            }
```

Explicarea Testelor:

Ca exemplu am testat algoritmul atunci când numărul populației este egal cu 100, numărul de generații este 100, avem o singură geană iar ecuația este x+2.
După cum se poate deduce, soluția va fi -2.

```
[Test]
0 references
public void Test1()
{
    var function = new org.mariuszgromada.math.mxparser.Function("equation", "x+2", "x");
    populationSize = 100;
    generations = 100;
    genes = 1;
    var equation = new EquationSolver(function, genes);
    Chromosome bestCr = algorithm.Solve(equation, populationSize, generations);
    string solutions = "";
    foreach (var gene in bestCr.Genes)
    {
        solutions = solutions + gene.ToString() + System.Environment.NewLine;
    }
    System.Console.WriteLine("Raspuns corect: -2");
    System.Console.WriteLine(solutions);
    Assert.AreEqual(-2, bestCr.Genes[0], 0.01);
}
```

4. Rolul membrilor echipei

Păuleț Faustina-Cristina

-documentație

-2 teste unitare

-interfață

-algoritmul de evoluție diferențială

Roman Sorin-Valentin

-documentație

-2 teste unitare

-algoritmul de evoluție diferențială

5. Concluzii

În concluzie, evoluția diferențială funcționează bine pentru foarte multe probleme clasice de optimizare . Unele probleme în care este utilizat, oferă reguli în alegerea parametrilor evoluției diferențiale care permite controlul comportamentului algoritmului cu a fost și în cazul rezolvării ecuațiilor cu un anumit grad.

6. Bibliografie

- Curs 5 Inteligență Artificială - Florin Leon
- Laborator 8 Inteligență Artificială - Florin Leon
- https://mathparser.org/mxparser-tutorial/user-defined-functions/
- https://en.wikipedia.org/wiki/Differential_evolution
- https://arxiv.org/ftp/arxiv/papers/1308/1308.4675.pdf