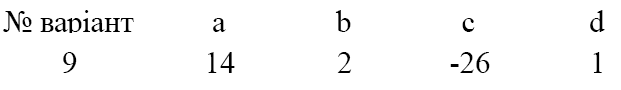
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| СПКм-12 | 4 | КОМБІНАТОРНА  ОПТИМІЗАЦІЯ ЗА ДОПОМОГОЮ ЕВОЛЮЦІЙНИХ  МЕТОДІВ |  |  |
| Дикий Б. Я. | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р.З. | |

**Мета роботи**: реалізувати генетичний алгоритм пошуку максимального і мінімального значення цільової функції згідно варіанту.

**Завдання:** реалізувати генетичний алгоритм пошуку максимального і мінімального значення цільової функції f(x) = a + bx + cx2 + dx3 на інтервалі

x = [-10, 53].

В - 2





**Максимальне і мінімальне значення цільової функції**

Графік функції згідно варіанту показаний на рис.1.

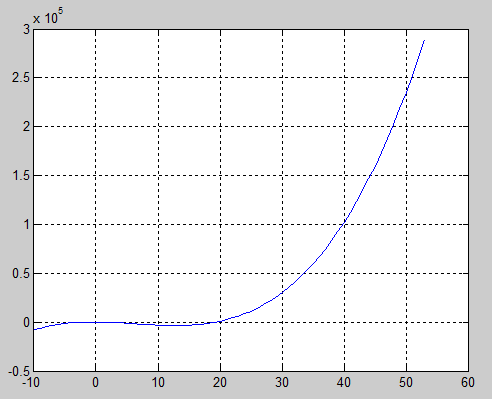


Рис.1. Графік функції на інтервалі [-10;53]

Для знаходження мінімуму функції було вибрано інтервал [0;53]. Пошук мінімуму цільової функції для перевірки реалізовано у Matlab за допомогою функції fminbnd() і рівний: -3.3309e+03 при x = 12.6610.

Для знаходження максимуму функції було вибрано інтервал [-10;10]. Пошук мінімуму цільової функції для перевірки реалізовано у Matlab за допомогою функції fminbnd() змінивши функцію на протилежну і рівний: 41.0965 при x = -0.4388. Графік функції на інтервалі [-10;10] показаний на рис.2.

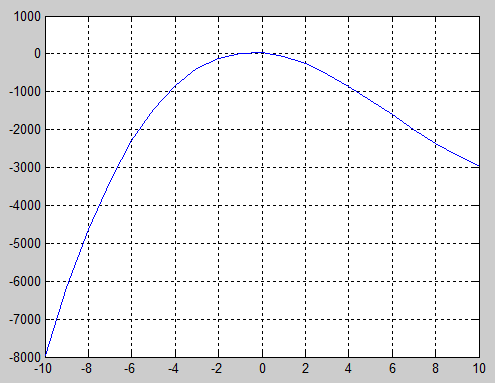


Рис.2. Графік функції на інтервалі[-10;10]

**Результати виконання програми**

Для виконання індивідуального завдання була вибрано використовувати турнірний відбір, одно точковий кросинговер і точкова мутація.

Результати:

|  |  |  |  |
| --- | --- | --- | --- |
| Кількість поколінь, розмір популяції | 10 | 100 | 300 |
| fmin | -2512.1626 | -2555.169 | -2555.2246 |
| xmin | 15.97067 | 17.248497 | 17.294273 |
| fmax | -12.889909 | 14.03714 | 14.038519 |
| xmax | 1.0796688 | 0.031248808 | 0.03855741 |

Отже похибка при знаходженні мінімуму і максимуму залежать від кількості поколінь і розміру популяції, при їхньому збільшенні похибки прямують до 0.

Результат виконання програми для десяти поколінь з розміром популяції 20 для знаходження мінімуму функції:

Початкова популяція

0 [5.9824066] fitness = -1595.2092

1 [13.825651] fitness = -3246.1948

2 [8.544825] fitness = -2541.3364

3 [9.140724] fitness = -2731.2422

4 [14.969405] fitness = -2979.8682

5 [9.198299] fitness = -2748.6252

6 [10.46744] fitness = -3078.9062

7 [7.037057] fitness = -2000.0333

8 [9.124159] fitness = -2726.207

9 [13.553963] fitness = -3281.752

10 [6.050377] fitness = -1621.4478

11 [10.415984] fitness = -3067.7239

12 [8.804463] fitness = -2626.2163

13 [12.057975] fitness = -3310.1113

14 [11.835693] fitness = -3292.4238

15 [10.269802] fitness = -3034.8438

16 [5.5944424] fitness = -1445.8193

17 [14.897933] fitness = -3002.3408

18 [12.765596] fitness = -3330.2422

19 [11.527979] fitness = -3259.5767

Нова популяція

0 [12.057975] fitness = -3310.1113

1 [12.057975] fitness = -3310.1113

2 [12.765596] fitness = -3330.2422

3 [12.765596] fitness = -3330.2422

4 [12.765596] fitness = -3330.2422

5 [12.765596] fitness = -3330.2422

6 [12.765596] fitness = -3330.2422

7 [12.765596] fitness = -3330.2422

8 [12.765596] fitness = -3330.2422

9 [13.553963] fitness = -3281.752

10 [11.835693] fitness = -3292.4238

11 [12.765596] fitness = -3330.2422

12 [12.765596] fitness = -3330.2422

13 [12.057975] fitness = -3310.1113

14 [11.527979] fitness = -3259.5767

15 [12.765596] fitness = -3330.2422

16 [14.863309] fitness = -3012.9375

17 [7.546364] fitness = -2190.1943

18 [12.989822] fitness = -3324.4116

19 [8.272869] fitness = -2449.2688

Найкраще значення покоління -3330.2422

...

Нова популяція

0 [12.649832] fitness = -3330.8828

1 [12.649832] fitness = -3330.8828

2 [12.649832] fitness = -3330.8828

3 [12.649832] fitness = -3330.8828

4 [12.649832] fitness = -3330.8828

5 [12.649832] fitness = -3330.8828

6 [12.649832] fitness = -3330.8828

7 [12.649832] fitness = -3330.8828

8 [12.649832] fitness = -3330.8828

9 [12.649832] fitness = -3330.8828

10 [12.649832] fitness = -3330.8828

11 [12.649832] fitness = -3330.8828

12 [12.649832] fitness = -3330.8828

13 [12.649832] fitness = -3330.8828

14 [12.649832] fitness = -3330.8828

15 [12.649832] fitness = -3330.8828

16 [14.653169] fitness = -3073.2207

17 [9.036072] fitness = -2699.1846

18 [6.731345] fitness = -1883.6609

19 [5.425605] fitness = -1381.1819

Найкраще значення покоління -3330.8828

Результат:

[12.649832] fitness = -3330.8828

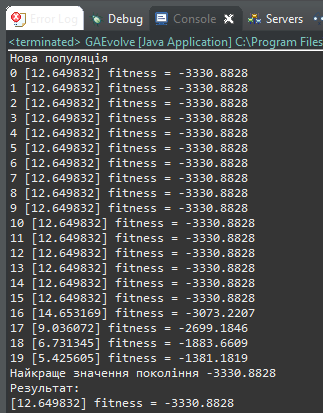


Рис.3. Результати виконання програми для знаходження мінімуму

Результат виконання програми для десяти поколінь з розміром популяції 20 для знаходження максимуму функції:

Початкова популяція

0 [-9.558561] fitness = -7137.1924

1 [-3.1828022] fitness = -464.75012

2 [-3.6743555] fitness = -677.65234

3 [-0.48911285] fitness = 40.946884

4 [-8.680109] fitness = -5641.921

5 [-1.808897] fitness = -77.277824

6 [8.684746] fitness = -2587.4658

7 [-7.809347] fitness = -4362.5376

8 [-2.2758985] fitness = -176.45483

9 [-7.8454733] fitness = -4411.757

10 [-5.3193054] fitness = -1711.7897

11 [-1.6849747] fitness = -56.25558

12 [8.184767] fitness = -2418.808

13 [2.6741972] fitness = -439.66083

14 [-3.0572128] fitness = -416.92276

15 [5.153261] fitness = -1277.697

16 [6.0146217] fitness = -1607.644

17 [-3.4926486] fitness = -594.10645

18 [0.47938251] fitness = -6.278047

19 [1.0735054] fitness = -83.346664

Нова популяція

0 [-1.6849747] fitness = -56.25558

1 [1.0735054] fitness = -83.346664

2 [-1.6849747] fitness = -56.25558

3 [-1.808897] fitness = -77.277824

4 [-0.48911285] fitness = 40.946884

5 [-1.6849747] fitness = -56.25558

6 [0.47938251] fitness = -6.278047

7 [0.47938251] fitness = -6.278047

8 [-0.48911285] fitness = 40.946884

9 [0.47938251] fitness = -6.278047

10 [0.47938251] fitness = -6.278047

11 [-0.48911285] fitness = 40.946884

12 [-2.2758985] fitness = -176.45483

13 [-0.48911285] fitness = 40.946884

14 [-0.48911285] fitness = 40.946884

15 [-0.48911285] fitness = 40.946884

16 [3.6496906] fitness = -739.2535

17 [7.821354] fitness = -2290.234

18 [4.5522346] fitness = -1054.3622

19 [-4.946363] fitness = -1431.4004

Найкраще значення покоління 40.946884

...

Нова популяція

0 [-0.47621536] fitness = 41.01382

1 [-0.47621536] fitness = 41.01382

2 [-0.47621536] fitness = 41.01382

3 [-0.47621536] fitness = 41.01382

4 [-0.47621536] fitness = 41.01382

5 [-0.47621536] fitness = 41.01382

6 [-0.47621536] fitness = 41.01382

7 [-0.47621536] fitness = 41.01382

8 [-0.47621536] fitness = 41.01382

9 [-0.47621536] fitness = 41.01382

10 [-0.47621536] fitness = 41.01382

11 [-0.47621536] fitness = 41.01382

12 [-0.47621536] fitness = 41.01382

13 [-0.47621536] fitness = 41.01382

14 [-0.47621536] fitness = 41.01382

15 [-0.47621536] fitness = 41.01382

16 [-2.5003076] fitness = -235.71152

17 [-3.3935118] fitness = -550.9384

18 [-5.374733] fitness = -1755.8817

19 [7.958414] fitness = -2339.2495

Найкраще значення покоління 41.01382

Результат:

[-0.47621536] fitness = 41.01382

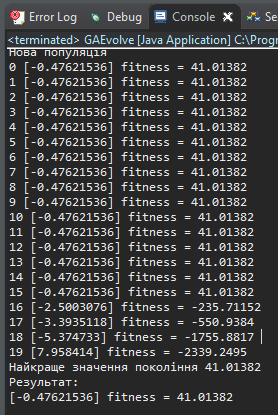


Рис.4. Результати виконання програми для знаходження максимуму

Код програми на мові програмування Java:

**class GAEvolve**

**public** **class** GAEvolve {

**public** **float**[] best\_fitness;// best\_fitness[i] є найкращим значенням для

// і-го покоління даної еволюції

**public** GAIndividual best\_ind; // найкращий індивідум(х) даної еволюції

**public** GAEvolve(**int** generations, **int** pop\_size, **int** genome\_size, **int** xrate,

**int** mrate, **float**[] min\_range, **float**[] max\_range) {

// xrate: частота схрещування

// mrate: частота мутації

// xrate і mrate можуть змінюватися під час покоління генерації

best\_fitness = **new** **float**[generations];

GAPopulation gap = **new** GAPopulation(pop\_size, genome\_size, min\_range,

max\_range);

best\_fitness[0] = gap.ind[gap.best\_index].fitness;

**for** (**int** i = 1; i < generations; i++) {

gap = gap.generate(gap, xrate, mrate, min\_range, max\_range);

best\_fitness[i] = gap.ind[gap.best\_index].fitness;

// if(i<11)

System.***out***

.println("Найкраще значення покоління " + best\_fitness[i]);

}

best\_ind = gap.ind[gap.best\_index];

}

**public** **static** **void** main(String[] args) {

// вибір критерію пошуку мінімуму або максимуму

// для пошуку мінімуму

// float[] min = new float[] { 5f };

// float[] max = new float[] { 15f };

// для пошуку максимуму

**float**[] min = **new** **float**[] { -10f };

**float**[] max = **new** **float**[] { 10 };

GAEvolve gae = **new** GAEvolve(10, 20, 1, 80, 20, min, max);

System.***out***.println("Результат:");

System.***out***.println(gae.best\_ind);

}

}

**class GAIndividual**

**public** **class** GAIndividual {

**private** **static** Random *randg* = **new** Random(); // Рандомний генератор

**public** **int** genome\_size;

**public** **float**[] genome;

**public** **float** fitness;

**public** GAIndividual(**int** gsize, **float**[] min\_range, **float**[] max\_range) {

// створити випадкового індивідума довжиною gsize

// і-й ген повине бути в діапазоні між min\_range[i] і max\_range[i]

genome\_size = gsize;

genome = **new** **float**[genome\_size];

**for** (**int** i = 0; i < genome\_size; i++) {

genome[i] = *randg*.nextFloat() \* (max\_range[i] - min\_range[i])

+ min\_range[i];

}

evalFitness();// оцінити придатність цього нового індивідума

}

**public** GAIndividual(**float** d[]) {

// Створити індивідума, що його ген є таки же як d []

genome\_size = d.length;

genome = **new** **float**[genome\_size];

**for** (**int** i = 0; i < genome\_size; i++) {

genome[i] = d[i];

}

evalFitness();// оцінити придатність цієї нового індивідума

}

**public** GAIndividual mutate(**float**[] min\_range, **float**[] max\_range) {

// rate це шанс кожного гена мутувати

**float** rate = 1.0f / (**float**) genome\_size;

**float**[] result = **new** **float**[genome\_size];

**for** (**int** i = 0; i < genome\_size; i++)

result[i] = genome[i];

// застосування точкової мутації

**for** (**int** i = 0; i < genome\_size; i++)

**if** ((**float**) Math.*random*() < rate)

result[i] = *randg*.nextFloat() \* (max\_range[i] - min\_range[i])

+ min\_range[i];

**return** **new** GAIndividual(result);

}

**public** **static** GAIndividual xover1p(GAIndividual f, GAIndividual m) {

// одноточковий кросинговер

Random rng = **new** Random();

**int** xpoint = 1 + rng.nextInt(1);

**float**[] child = **new** **float**[f.genome\_size];

**for** (**int** i = 0; i < xpoint; i++) {

child[i] = f.genome[i];

}

**for** (**int** i = xpoint; i < f.genome\_size; i++) {

child[i] = m.genome[i];

}

**return** **new** GAIndividual(child);

}

**public** String toString() {

String s = "[";

s += genome[0] + "]";

s += " fitness = " + fitness;

**return** s;

}

**private** **void** evalFitness() {

**int** a = 30;

**int** b = -50;

**int** c = -55;

**int** d = 3;

// цільова функція

fitness = (a + b \* genome[0] + c \* genome[0] \* genome[0] + d

\* genome[0] \* genome[0] \* genome[0]);

}

}

**class GAPopulation**

**public** **class** GAPopulation {

**private** **static** Random *randg* = **new** Random();// Рандомний генератор

**public** **int** pop\_size;

**public** GAIndividual[] ind;

**public** **int** best\_index; // індекс найкращого індивідума в масиві ind

// best\_fitness = ind[best\_index].fitness

**public** GAPopulation(**int** psize, **int** gsize, **float**[] min\_range,

**float**[] max\_range) {

// створення рандомної популяції кількісью pop\_size

// psize: довжина популяції

// gsize: довжина геному

pop\_size = psize;

ind = **new** GAIndividual[pop\_size];

System.***out***.println("Початкова популяція");

**for** (**int** i = 0; i < pop\_size; i++)

{ind[i] = **new** GAIndividual(gsize, min\_range, max\_range);

System.***out***.println(i+ " "+ ind[i]+" ");}

//System.out.println();

evaluate();

}

**public** GAPopulation(GAIndividual[] p) {

// Створення популяції з такимиж індивідумами як в p

pop\_size = p.length;

ind = **new** GAIndividual[pop\_size];

System.***out***.println("Нова популяція");

**for** (**int** i = 0; i < pop\_size; i++){

ind[i] = p[i];

System.***out***.println(i+ " "+ ind[i]+" ");}

evaluate();

}

**public** GAPopulation generate(GAPopulation p, **int** xrate, **int** mrate,

**float**[] min\_range, **float**[] max\_range) {

//Створення нової популяції з р, xrate відсотків індивідумів нового населення є

//схрещування, mrate відсотків з них створюються в результаті мутації, а інші по відтворення.

**if** (xrate < 0 || xrate > 100 || mrate < 0 || mrate > 100

|| xrate + mrate > 100)

System.***err***.println("error: xrate і/чи mrate неправилно встановлені");

GAIndividual[] newg = **new** GAIndividual[p.pop\_size];

**int** newg\_index = 0;

**int** xn = xrate \* p.pop\_size / 100;

//xn: Кількість нащадків, які будуть схрешення

**int** mn = mrate \* p.pop\_size / 100;

// mn: кількість нащадків які будуть створенні мутацією

// схрещування:

**for** (**int** i = 0; i < xn; i++) {

// select to parents for cross-over:

**int** p1 = p.tr\_select();

**int** p2 = p.tr\_select();

newg[newg\_index++] = GAIndividual.*xover1p*(p.ind[p1], p.ind[p2]);

}

// мутація:

**for** (**int** i = 0; i < mn; i++)

newg[newg\_index++] = p.ind[p.tr\_select()].mutate(min\_range,

max\_range);

// відтворення:

**for** (**int** i = newg\_index; i < p.pop\_size; i++)

newg[i] = p.ind[p.tr\_select()];

**return** **new** GAPopulation(newg);

}

**public** **int** tr\_select() {

//турнірна вибірка розміром pop\_size/10

//вона повертає індекс вибраного особи в ind []

**int** s\_index = *randg*.nextInt(pop\_size);

// індекс вибраного індивідума

**float** s\_fitness = ind[s\_index].fitness;

**int** tr\_size = Math.*min*(10, pop\_size);

**for** (**int** i = 1; i < tr\_size; i++) {

**int** tmp = *randg*.nextInt(pop\_size);

// вибір критерію пошуку мінімуму або максимуму

**if** (ind[tmp].fitness > s\_fitness) {//< для min//>для max

s\_index = tmp;

s\_fitness = ind[tmp].fitness;

}

}

**return** s\_index;

}

**private** **void** evaluate() {

//оцінювання

**int** best = 0;

// індекс найкрощого індивідума

**float** best\_fitness = ind[0].fitness;

// сума придатності особин даної популяції

**for** (**int** i = 1; i < pop\_size; i++) {

// вибір критерію пошуку мінімуму або максимуму

**if** (ind[i].fitness > best\_fitness) {//< для min//>для max

best = i;

best\_fitness = ind[i].fitness;

}

}

best\_index = best;

}

**public** String toString() {

String s = "best individual = " + ind[best\_index];

**return** s;

}

}

Код програми на мові програмування Matlab для знаходження екстремумів функції:

clc

a=30;

b=-50;

c=-55;

d=3;

x=-10:1:53

y=a+b\*x+c\*x.^2+d\*x.^3;

plot(x,y)

grid on

minX=fminbnd(@(x) a+b\*x+c\*x^2+d\*x^3,0,53)

minY=a+b\*minX+c\*minX^2+d\*minX^3

x=-10:1:10

y=a+b\*x+c\*x.^2+d\*x.^3;

figure

plot(x,y)

grid on

maxX=fminbnd(@(x) -(a+b\*x+c\*x^2+d\*x^3),-10,10)

maxY=a+b\*maxX+c\*maxX^2+d\*maxX^3

**Висновки:** виконавши лабораторну роботу я реалізував за допомогою програмної мови Java програмне забезпечення для пошуку оптимумів функції в якому використав турнірний відбір, одноточковий ію. Програма показує результати за короткий період часу з невеликою похибкою навіть при невеликій кількості поколінь і розміру популяції, але при збільшенні цих параметрів похибка прямує до 0.