|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САПР** | | Тема | оцінка | підпис |
| СПКм-12 | 4 | ПОШУК МІНІМАЛЬНОГО І МАКСИМАЛЬНОГО ЗНАЧЕНЬ ЦІЛЬОВОЇ ФУНЦІЇ |  |  |
| Кикуш О.М. | |
| № залікової: 1408502 | |
| 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.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

**Варіант -3**

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

x = [-10, 53].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варіанту | a | b | c | d |
| 3 | 10 | -20 | -40 | 1 |

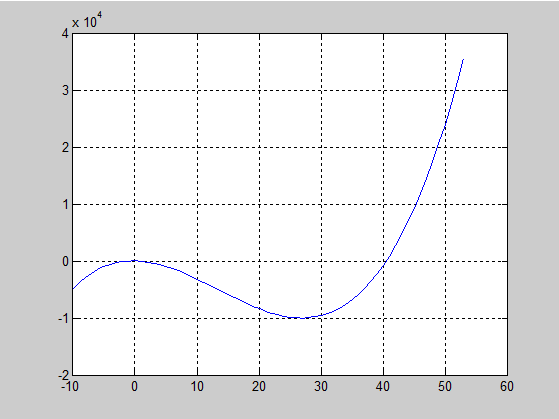


Рис.1. Графік функції на інтервалі [-10;53]

Для знаходження мінімуму функції було вибрано інтервал [20;30]. Пошук мінімуму цільової функції для перевірки реалізовано у Matlab за допомогою функції fminbnd() і рівний: -1.0007e+04 при x = 26.9144.

Для знаходження максимуму функції було вибрано інтервал [0;20]. Пошук мінімуму цільової функції для перевірки реалізовано у Matlab за допомогою функції fminbnd() змінивши функцію на протилежну і рівний: 20.0563 при x = 0.0376. Графік функції на інтервалі [0;20] показаний на рис.2.

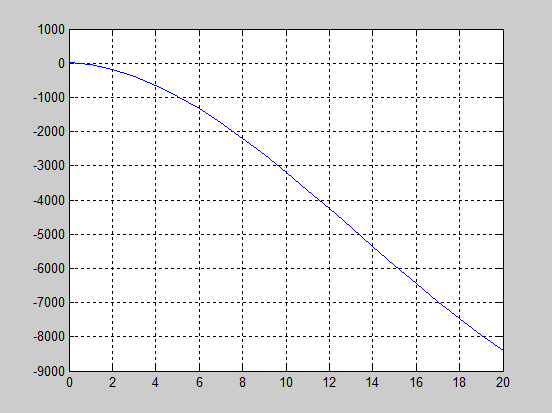


Рис.2. Графік функції на інтервалі[0;20]

**виконання роботи**

**Результати виконання**

Для виконання індивідуального завдання була вибрано використовувати турнірний відбір, рівномірне схрещування і класична мутація обміну.

|  |  |  |  |
| --- | --- | --- | --- |
| Кількість поколінь, розмір популяції | 10 | 100 | 300 |
| fmin | -10007.0 | -10007.236 | -10007.303 |
| xmin | 27.0 | 26.874432 | 26.915789 |
| fmax | 9.533673 | 9.9 | 9.9 |
| xmax | 0.022320509 | 2.3841858E-7 | 4.7683716E-7 |

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

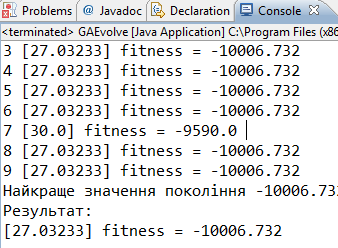


Рис.3. Результати виконання програми

Результат виконання програми для десяти поколінь з розміром популяції 30 для знаходження максимуму функції:

Початкова популяція

0 [9.48126] fitness = -2923.086

1 [15.320711] fitness = -6089.24

2 [0.614084] fitness = -17.134075

3 [4.7555056] fitness = -882.1584

4 [3.6559463] fitness = -548.89154

5 [15.447675] fitness = -6157.8906

6 [16.596087] fitness = -6768.0635

7 [6.0929084] fitness = -1370.6091

8 [2.030133] fitness = -187.0932

9 [11.966555] fitness = -4243.6763

Нова популяція

0 [0.6248262] fitness = -17.868898

1 [0.022320509] fitness = 9.533673

2 [0.614084] fitness = -17.134075

3 [2.0535667] fitness = -191.09662

4 [1.5298476] fitness = -110.6338

5 [0.614084] fitness = -17.134075

6 [2.0222871] fitness = -185.76112

7 [0.614084] fitness = -17.134075

8 [15.447675] fitness = -6157.8906

9 [0.614084] fitness = -17.134075

Найкраще значення покоління 9.533673

…

…

…

Нова популяція

0 [0.022320509] fitness = 9.533673

1 [0.022320509] fitness = 9.533673

2 [0.022320509] fitness = 9.533673

3 [0.022320509] fitness = 9.533673

4 [0.022320509] fitness = 9.533673

5 [0.022320509] fitness = 9.533673

6 [0.022320509] fitness = 9.533673

7 [0.022320509] fitness = 9.533673

8 [0.022320509] fitness = 9.533673

9 [0.022320509] fitness = 9.533673

Найкраще значення покоління 9.533673

Результат:

[0.022320509] fitness = 9.533673

Результат виконання програми для десяти поколінь з розміром популяції 30 для знаходження мінімуму функції:

Початкова популяція

0 [22.225594] fitness = -9214.659

1 [23.100014] fitness = -9470.013

2 [26.319641] fitness = -9993.098

3 [23.819696] fitness = -9646.741

4 [26.161995] fitness = -9984.662

5 [23.052086] fitness = -9457.141

6 [27.770597] fitness = -9976.801

7 [21.111515] fitness = -8830.753

8 [27.42998] fitness = -9996.332

9 [23.327986] fitness = -9529.386

Нова популяція

0 [27.285448] fitness = -10001.639

1 [26.406527] fitness = -9996.926

2 [27.455355] fitness = -9995.219

3 [27.42998] fitness = -9996.332

4 [27.407028] fitness = -9997.291

5 [26.267931] fitness = -9990.547

6 [27.15712] fitness = -10004.883

7 [30.0] fitness = -9590.0

8 [26.161995] fitness = -9984.662

9 [27.42998] fitness = -9996.332

Найкраще значення покоління -10004.883

…

…

…

Нова популяція

0 [27.0] fitness = -10007.0

1 [27.0] fitness = -10007.0

2 [27.0] fitness = -10007.0

3 [27.03186] fitness = -10006.734

4 [27.0] fitness = -10007.0

5 [27.0] fitness = -10007.0

6 [27.0] fitness = -10007.0

7 [30.0] fitness = -9590.0

8 [15.15712] fitness = -6000.503

9 [27.0] fitness = -10007.0

Найкраще значення покоління -10007.0

Нова популяція

0 [27.0] fitness = -10007.0

1 [27.0] fitness = -10007.0

2 [27.0] fitness = -10007.0

3 [27.0] fitness = -10007.0

4 [27.0] fitness = -10007.0

5 [27.0] fitness = -10007.0

6 [27.0] fitness = -10007.0

7 [30.0] fitness = -9590.0

8 [30.0] fitness = -9590.0

9 [27.0] fitness = -10007.0

Найкраще значення покоління -10007.0

Результат:

[27.0] fitness = -10007.0

**Код програми**

**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++) {

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

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

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

}

// мутація:

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

**int** n = (**int**)(Math.*random*() \* p.pop\_size);

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

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

s\_index = tmp;

s\_fitness = ind[tmp].fitness;

}

}

**return** s\_index;

}

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

// рівномірне схрещення

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

Random random = **new** Random();

**for** (**int** k = 0; k < f.genome\_size; k++) {

String fs = *floatToBinary*(f.genome[k]);

String ms = *floatToBinary*(m.genome[k]);

**int**[] maska = **new** **int**[fs.length()];

**for**(**int** i = 0; i < fs.length(); i++){

maska[i] = random.nextInt(2);

}

String childs = "";

**for** (**int** i = 0; i < fs.length(); i++) {

**if**(maska[i] == 0) childs += fs.charAt(i);

**else** childs += ms.charAt(i);

}

child[0] = *binaryToFloat*(childs);

}

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

}

**public** GAIndividual mutate(GAIndividual gaIndividual, **float**[] max\_reg) {

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

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

result[i] = genome[i];

// класична мутація обміну

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

String string = *floatToBinary*( gaIndividual.genome[i]);

**int** n = string.indexOf(".");

**int** pp1 = (**int**)(Math.*random*()\*n);

**int** pp2 = (**int**)(Math.*random*()\*n);

**if**(pp1 > pp2) { **int** q = pp1; pp1 = pp2; pp2 = q;}

**char** [] charmas = string.toCharArray();

**char** c = charmas[pp1];

charmas[pp1] = charmas[pp2];

charmas[pp2] = c;

String end = "";

**for**(**int** j = 0; j < charmas.length; j++) {

end += charmas[j];}

result[i] = *binaryToFloat*(end);

**if**(result[i] > max\_reg[i]) result[i] = max\_reg[i];

}

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

}

**Висновки:** виконавши лабораторну роботу я реалізував в мові Java програму для пошуку оптимумів функції в якому використав турнірний відбір, рівномірне схрещування і класичну мутацію обміну. Програма показує результати за короткий період часу з невеликою похибкою навіть при невеликій кількості поколінь і розміру популяції, і при збільшенні цих параметрів похибка прямує до 0.