|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| СПКм-12 | 4 | **Генетичні алгоритми пошуку пошуку** |  |  |
| Баландюх О.А. | |
| № залікової: | |
| 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N варіанту | a | b | c | d |
| 2 | 30 | -50 | -55 | 3 |

**Максимальне і мінімальне значення цільової функції**

Графік функції згідно варіанту показаний на рис.1.

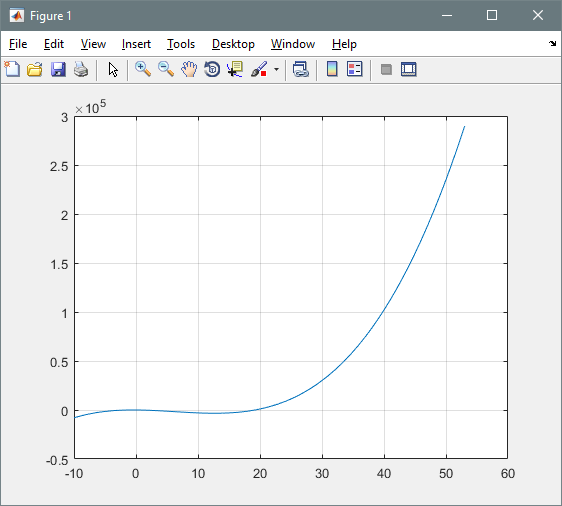


Рис.1. Графік функції на інтервалі [-10;53]

Для знаходження мінімуму функції було вибрано інтервал [-10;53]. Пошук мінімуму цільової функції для перевірки реалізовано у Matlab за допомогою функції fminbnd() і рівний: -3.3309e+03 (-3330.9).

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

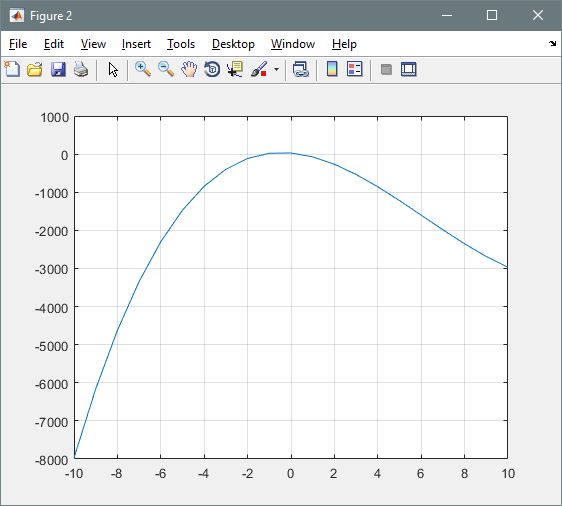


Рис.2. Графік функції на інтервалі[-10;10]

**Результати виконання програми**

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

Результати:

|  |  |  |  |
| --- | --- | --- | --- |
| Кількість поколінь, розмір популяції | 10 | 100 | 300 |
| fmin | -252881.14 | -276738.25 | -288612.16 |
| fmax | 40.226086 | 40.99248 | 41.09547 |

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

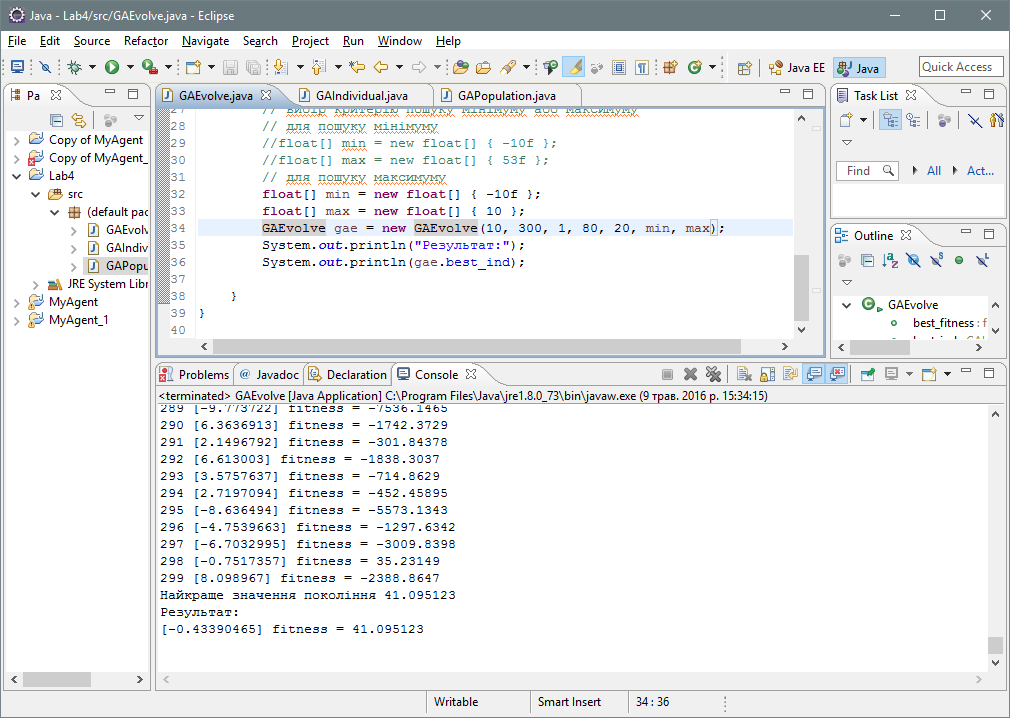


Рис.3. Результати виконання програми

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 за допомогою програмної мови Java програмне забезпечення 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, але при збільшенні цих параметрів похибка прямує до 0.