|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, САПР, СПК** | | **Тема** | **Оцінка:** | **Підпис:** |
| КНСП-11 | 4 | Використання генетичних алгоритмів з бітовим представленням хромосом |  |  |
| Янчук Н. Ю. | |
| Варіант 10 | |
| Методи нечіткої логіки та  еволюційні алгоритми | | **Викладач:**  Кривий Р. З. | |

**Мата:** навчитися застосовувати генетичні алгоритми з побітовим представленням хромосом.

**Хід роботи**

Для виконання завдання була використана функція ga пакету MatLab, і окремо реалізовані функції для побітової мутації і побітового схрещування.

|  |  |
| --- | --- |
| Цільові функції для пошуку мінімуму та максимуму: | |
| function [output\_args] = FitnessFcn( input\_args )  % input\_args = [x1]  % варіант 10  a = 26; b = -86; c = -59; d = 3;  x = input\_args(1);  f = a + b\*x + c\*(x^2) + d\*(x^3);  output\_args = f;  end | function [output\_args] = MaxFitnessFcn( input\_args )  output\_args =  (-1)\*FitnessFcn( input\_args );  end |

|  |
| --- |
| Побітова мутація |
| function [ mutationChildren ] = MutationFcn( parents, options, nvars, ...  FitnessFcn, state, thisScore, thisPopulation )  % parents - номер особини в популяції, що мутує  % nvars - кількість змінних  % state - інформація про поточну популяцію  % thisScore - оцінки поточної популяції  % thisPopulation - поточна популяція  % маска мутації. змінює випадковий біт на протилежний  mask = zeros(1, 6);  mask(randi(6)) = 1;    mutant = thisPopulation(parents, :)+10;  for i=1:1:nvars  dm = mutant(i);  if dm > 63  dm = de2bi(dm);  dm = dm(1:6); %відтинаємо лишні біти  else  dm = de2bi(dm, 6);  end  dm = bitxor(dm, mask);  mutant(i) = bi2de(dm)-10;  end    mutationChildren = mutant;  end |

|  |
| --- |
| Побітове схрещування |
| function [ xoverKids ] = CrossoverFcn( parents, options, nvars, FitnessFcn, ...  unused,thisPopulation )  % parents - індекси батьків в поточній популяції, що беруть участь у  % схрещуванні. вектор з парною кількістю елементів  % nvars - кількість змінних (генів)  % unused - вектор-стовбець із оцінкою кожної особини  % thisPopulation - поточна популяція (матриця)  ret = zeros(length(parents)/2, nvars);  for i = 1:2:length(parents)  p1 = thisPopulation(i, :);  p2 = thisPopulation(i+1, :);    c = thisPopulation(i, :);  for j = 1:1:nvars  p1\_bit = toBitArr(p1(j)+10);  p2\_bit = toBitArr(p2(j)+10);    c\_bit = [p1\_bit(1:3), p2\_bit(4:6)];  c(j) = bi2de(c\_bit)-10;  end  ret((i+1)/2,:) = c;  end;  xoverKids = ret;  end  function [bitVal] = toBitArr(decVal)  if decVal > 63  dm = de2bi(decVal);  dm = dm(1:6); %відтинаємо лишні біти  else  dm = de2bi(decVal, 6);  end  bitVal = dm;  end |

Результати кожної ітерації зберігаються в глобальну змінну, після чого виводяться на екран.

|  |
| --- |
| Функція для збереженя везультатів кожної ітерації |
| function [ state,options,optchanged ] = OutputFcn( options,state,flag )  global RET;  ci = state.Generation;  RET.generation = ci;  key = strcat('s',num2str(ci));  RET.population(:).(key) = state.Population;  RET.fvals(:).(key) = state.Score;  optchanged = false;  end |

Результати виконання:

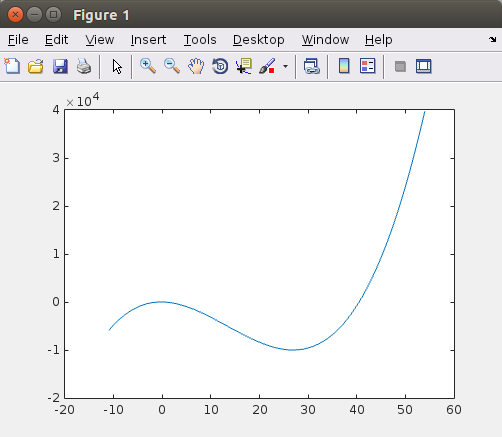


Рис. 1. Графік функції

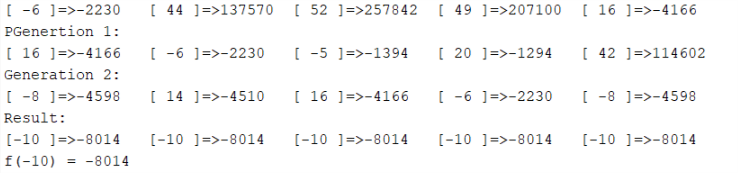


Рис. 2. Результат пошуку мінімуму

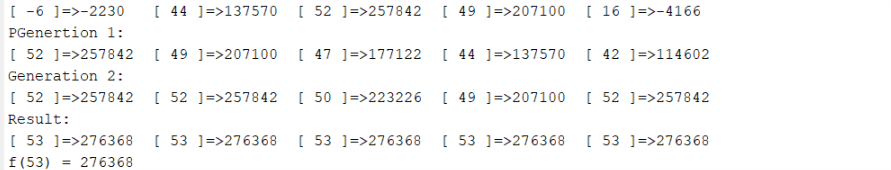


Рис. 3. Результат пошуку максимуму

**Висновок:** якщо вхідні дані цілі числа, то побітове представлення хромосоми є хорошим варіантом для зберігання цієї умови під час виконання генетичного алгоритму.