

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

З В І Т

**Лабораторна робота №7
з дисципліни
«Сучасні методи та моделі
інтелектуальних систем керування»**

Виконавець:

аспірант групи АКІТР-23-1а

Косей М.П.

Керівник:

викладач

Тиханський М. П.

Лабораторна робота №7

Стислі теоретичні відомості

Основи теорії генетичних алгоритмів сформульовані Холландом в його основоположній роботі і в подальшому були розвинені рядом дослідників. Найбільш відомою і цитованої в даний час є монографія Д. Голдберга, де систематично викладено основні результати та області практичного застосування генетичних алгоритмів.

Генетичні алгоритми використовують принципи і термінологію, запозичені у біологічної науки – генетики. У генетичних алгоритмах кожна особина представляє потенційне рішення деякої проблеми. У класичному генетичному алгоритмі особина кодується рядком двійкових символів – хромосомою, кожен біт якої називається геном. Множина особин – потенційних рішень становить популяцію. Пошук (суб) оптимального вирішення проблеми виконується в процесі еволюції популяції – послідовного перетворення однієї кінцевої множини рішень у іншу за допомогою генетичних операторів репродукції, кросинговеру та мутації.

Еволюційні обчислення використовують наступні три механізми природної еволюції:

- перший принцип заснований на концепції виживання найсильніших і природного відбору за Дарвіном який був сформульований їм у 1859 році в книзі «Походження видів шляхом природного відбору». Згідно Дарвіну особини, які краще здатні вирішувати завдання в своєму середовищі, виживають і більше розмножуються (репродуцують). У генетичних алгоритмах кожна особина представляє собою рішення деякої проблеми аналогією з цим принципом особини з кращими значеннями цільової (фітнес) функції мають великі шанси вижити і репродукувати. Формалізація цього принципу дає оператор репродукції;
- другий принцип обумовлений тим фактом, що хромосома нащадка складається з частин, отриманих з хромосом батьків. Цей принцип був відкритий в 1865 році Менделем. Його формалізація дає основу для оператора схрещування (кросинговеру);
- третій принцип заснований на концепції мутації, відкритої в 1900 році де Вре. Спочатку цей термін використовувався для опису істотних (різких) змін властивостей нащадків і набуття ними властивостей, відсутніх у батьків. За аналогією з цим принципом генетичні алгоритми використовують подібний механізм для різкої зміни властивостей нащадків і тим самим, підвищують різноманіття особин в популяції (множині рішень).

Ці три принципи складають ядро еволюційних обчислень. Використовуючи їх, популяція (множина рішень даної проблеми) еволюціонує від покоління до покоління.

Генетичний алгоритм бере множину параметрів оптимізаційної проблеми та кодує їх послідовностями кінцевої довжини в деякому кінцевому алфавіті (в найпростішому випадку двійковий алфавіт «0» і «1»).

Простий генетичний алгоритм випадковим чином генерує початкову

популяцію стрінгів (хромосом). Потім алгоритм генерує наступне покоління (популяцію), за допомогою трьох основних генетичних операторів:

- оператор репродукції (OP);
- оператор схрещування (кросинговеру, OK);
- оператор мутації (OM).

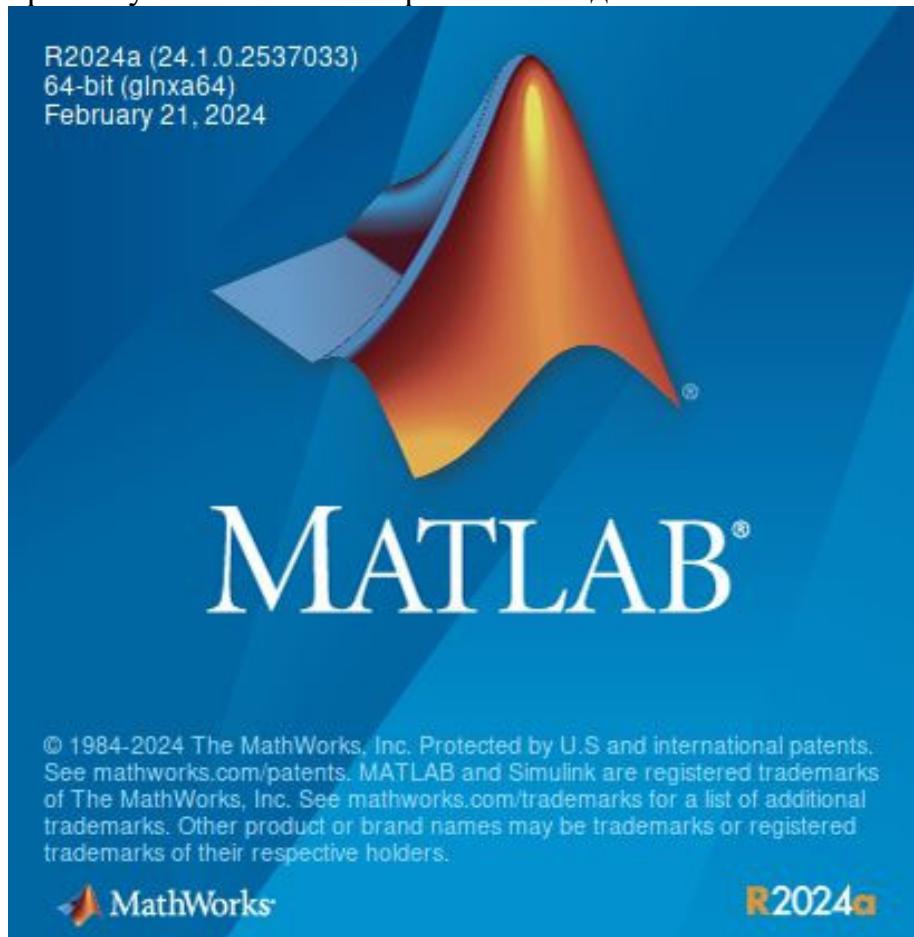
Генетичні оператори є математичної формалізацією наведених вище трьох основоположних принципів Дарвіна, Менделя і де Вре природної еволюції.

Генетичний алгоритм працює до тих пір, поки не буде виконано задану кількість поколінь (ітерацій) процесу еволюції або на деякій генерації буде отримано задану якість або внаслідок передчасної збіжності при попаданні в деякий локальний оптимум.

У кожному поколінні множина штучних особин створюється з використанням старих і додаванням нових з хорошими властивостями. Генетичні алгоритми – не просто випадковий пошук, вони ефективно використовують інформацію, накопичену в процесі еволюції. У процесі пошуку рішення необхідно дотримувати баланс між «експлуатацією» отриманих на поточний момент кращих рішень і розширенням простору пошуку. Різні методи пошуку вирішують цю проблему по-різному.

2) Практична частина

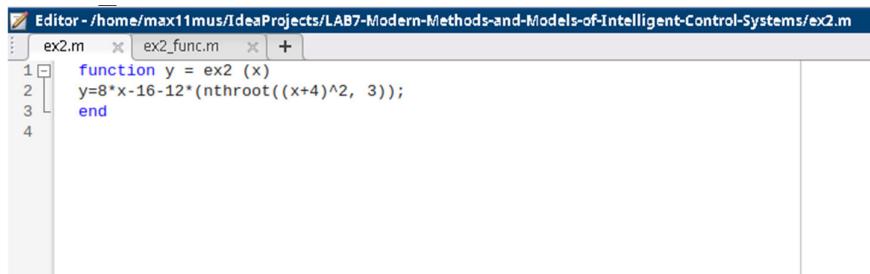
Використовуємо **MATLAB** версія R2024a для Linux.



Необхідно мінімізувати функцію однієї змінної:

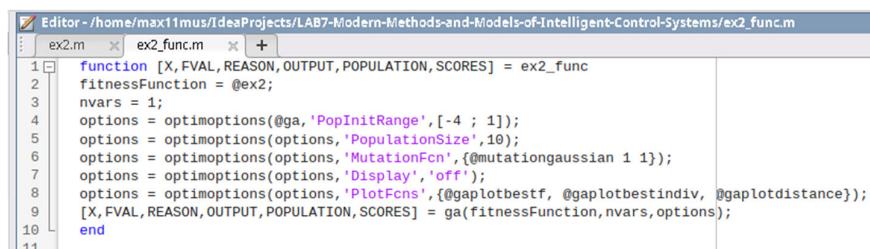
$$f(x) = 8x - 16 - 12\sqrt[3]{(x+4)^2}$$

Для цього, у середовищі *MATLAB*® необхідно створити два скрипти ex2.m та ex2_func.m.



Editor - /home/max11mus/IdeaProjects/LAB7-Modern-Methods-and-Models-of-Intelligent-Control-Systems/ex2.m

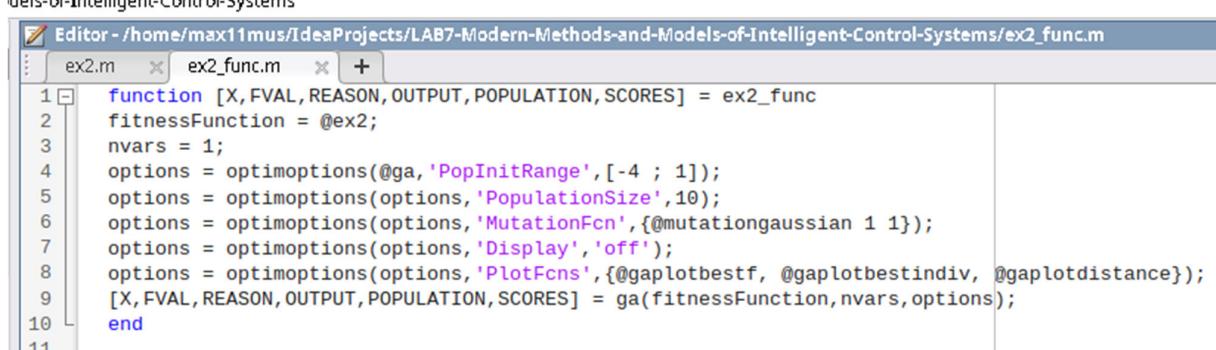
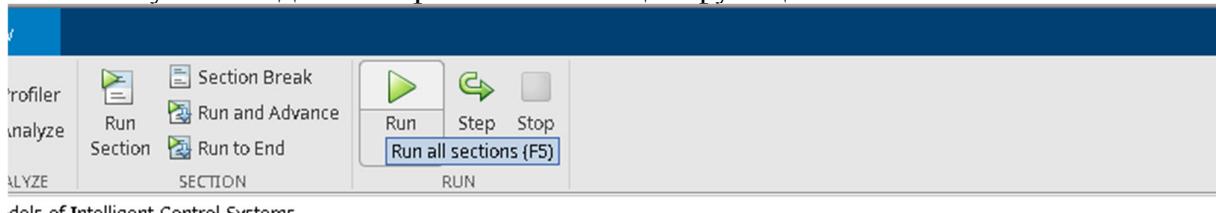
```
1 function y = ex2 (x)
2 y=8*x-16-12*(nthroot((x+4)^2, 3));
3 end
4
```



Editor - /home/max11mus/IdeaProjects/LAB7-Modern-Methods-and-Models-of-Intelligent-Control-Systems/ex2_func.m

```
1 function [X,FVAL,REASON,OUTPUT,POPULATION,SCORES] = ex2_func
2 fitnessFunction = @ex2;
3 nvars = 1;
4 options = optimoptions(@ga, 'PopInitRange', [-4 ; 1]);
5 options = optimoptions(options, 'PopulationSize', 10);
6 options = optimoptions(options, 'MutationFcn', {@mutationgaussian 1 1});
7 options = optimoptions(options, 'Display', 'off');
8 options = optimoptions(options, 'PlotFcns', {@gaplotbestf, @gaplotbestindiv, @gaplotdistance});
9 [X,FVAL,REASON,OUTPUT,POPULATION,SCORES] = ga(fitnessFunction,nvars,options);
10
11
```

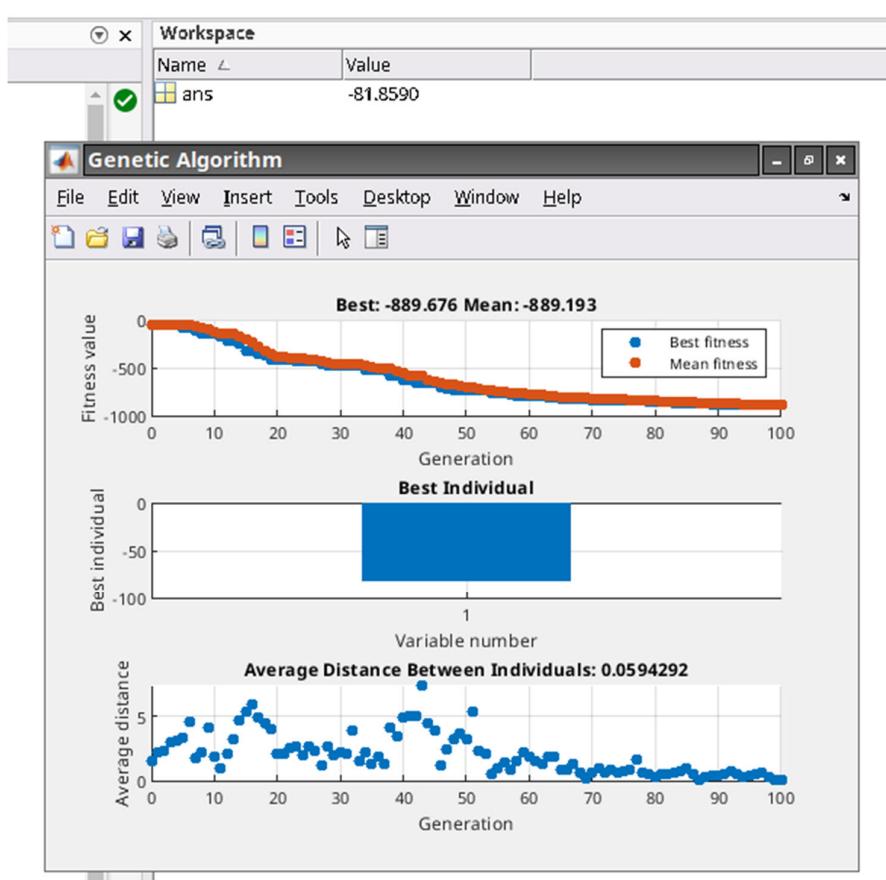
Запускаємо декілька разів оптимізацію функції



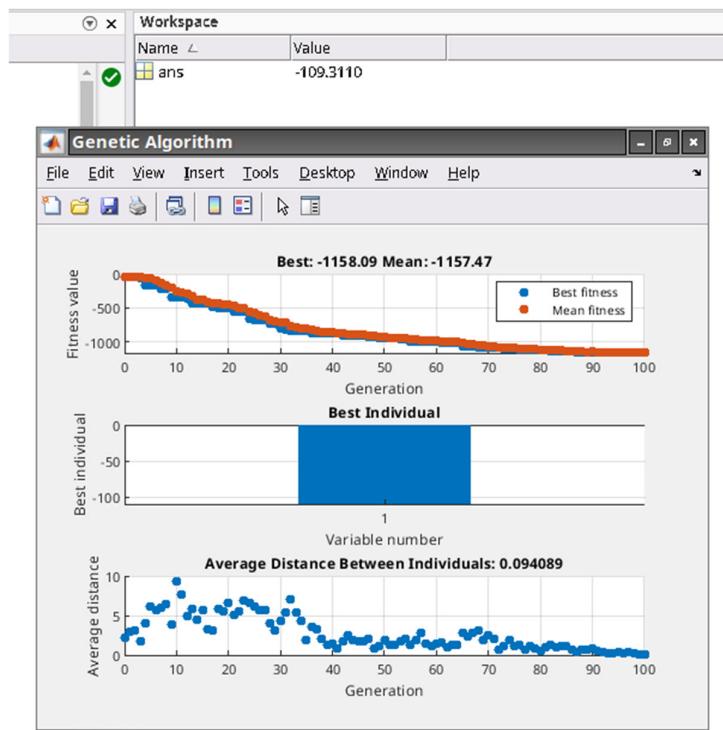
Editor - /home/max11mus/IdeaProjects/LAB7-Modern-Methods-and-Models-of-Intelligent-Control-Systems/ex2_func.m

```
1 function [X,FVAL,REASON,OUTPUT,POPULATION,SCORES] = ex2_func
2 fitnessFunction = @ex2;
3 nvars = 1;
4 options = optimoptions(@ga, 'PopInitRange', [-4 ; 1]);
5 options = optimoptions(options, 'PopulationSize', 10);
6 options = optimoptions(options, 'MutationFcn', {@mutationgaussian 1 1});
7 options = optimoptions(options, 'Display', 'off');
8 options = optimoptions(options, 'PlotFcns', {@gaplotbestf, @gaplotbestindiv, @gaplotdistance});
9 [X,FVAL,REASON,OUTPUT,POPULATION,SCORES] = ga(fitnessFunction,nvars,options);
10
11
```

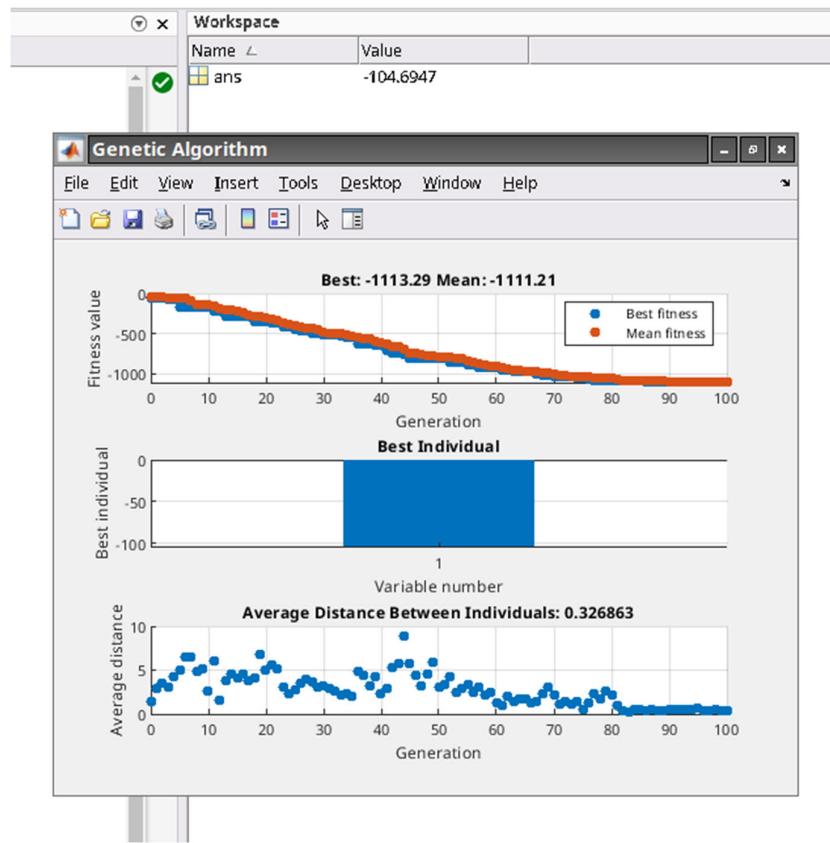
Результати оптимізації 1-ша оптимізація



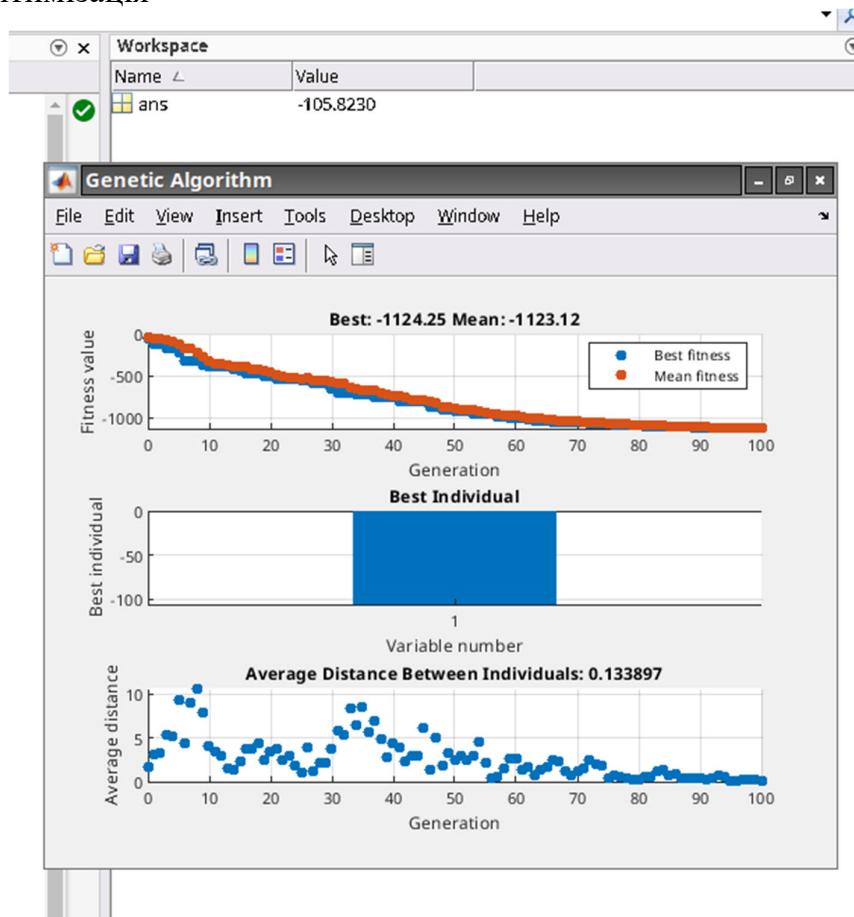
2-а оптимізація



3-я оптимізація



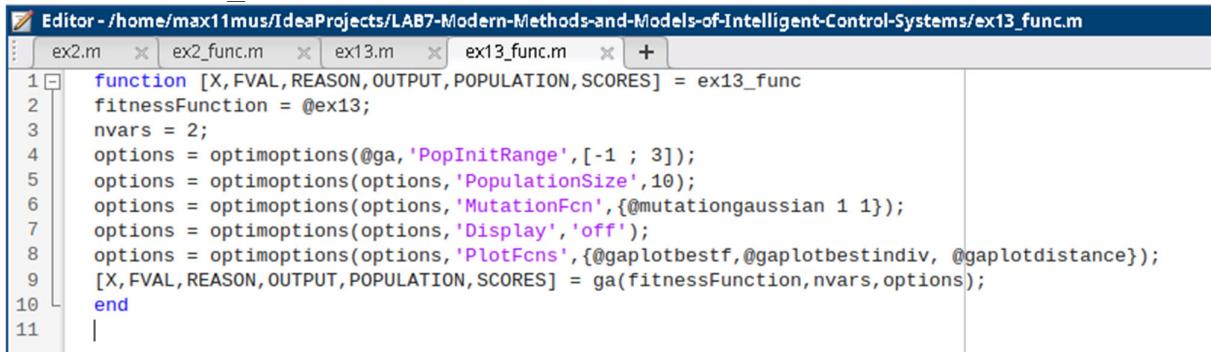
4-а оптимізація



Необхідно мінімізувати функцію двох змінних:

$$z(x, y) = \exp(-x^2 - y^2) + \sin(x + y)$$

Для цього, у середовищі MATLAB® необхідно створити два скрипти ex13.m та ex13_func.m.

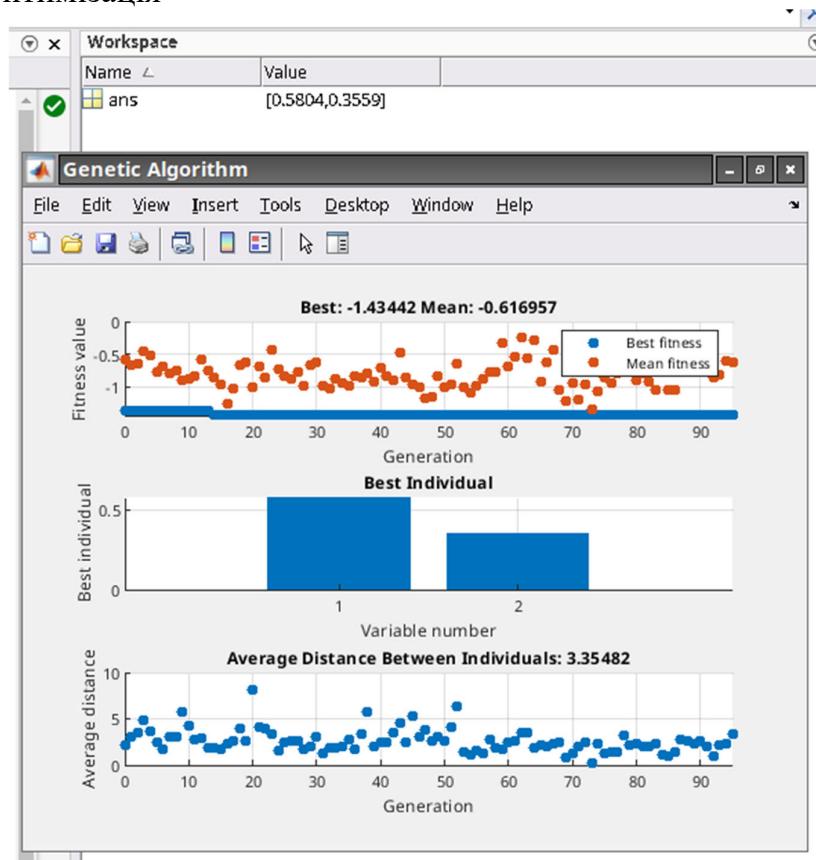


```
Editor - /home/max11mus/IdeaProjects/LAB7-Modern-Methods-and-Models-of-Intelligent-Control-Systems/ex13_func.m
: ex2.m x ex2_func.m x ex13.m x ex13_func.m x +
1 function [X,FVAL,REASON,OUTPUT,POPULATION,SCORES] = ex13_func
2 fitnessFunction = @ex13;
3 nvars = 2;
4 options = optimoptions(@ga, 'PopInitRange', [-1 ; 3]);
5 options = optimoptions(options, 'PopulationSize', 10);
6 options = optimoptions(options, 'MutationFcn', {@mutationgaussian 1 1});
7 options = optimoptions(options, 'Display', 'off');
8 options = optimoptions(options, 'PlotFcns', {@gaplotbestf,@gaplotbestindiv, @gaplotdistance});
9 [X,FVAL,REASON,OUTPUT,POPULATION,SCORES] = ga(fitnessFunction,nvars,options);
10 end
11 |
```

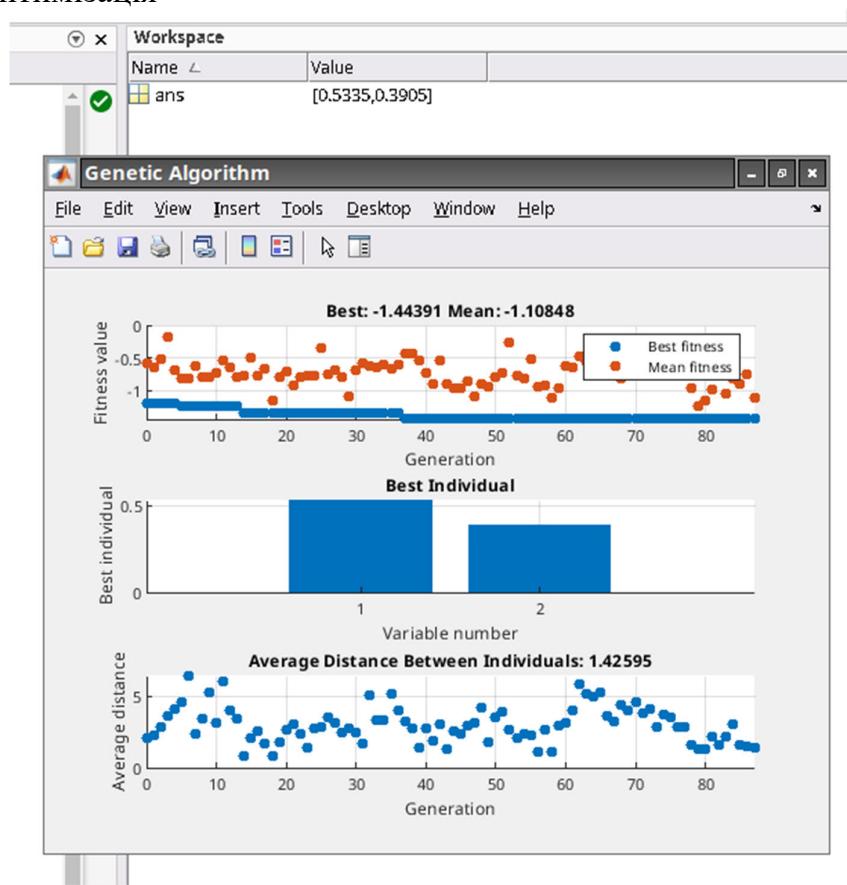
Запускаємо декілька разів оптимізацію функції

Результати оптимізації:

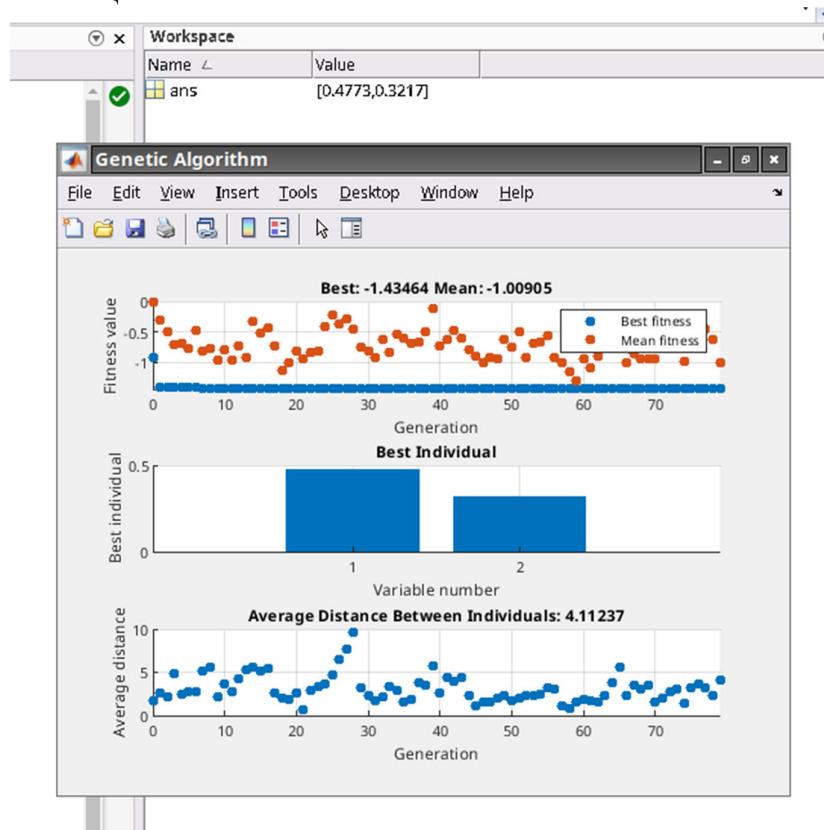
1-ша оптимізація



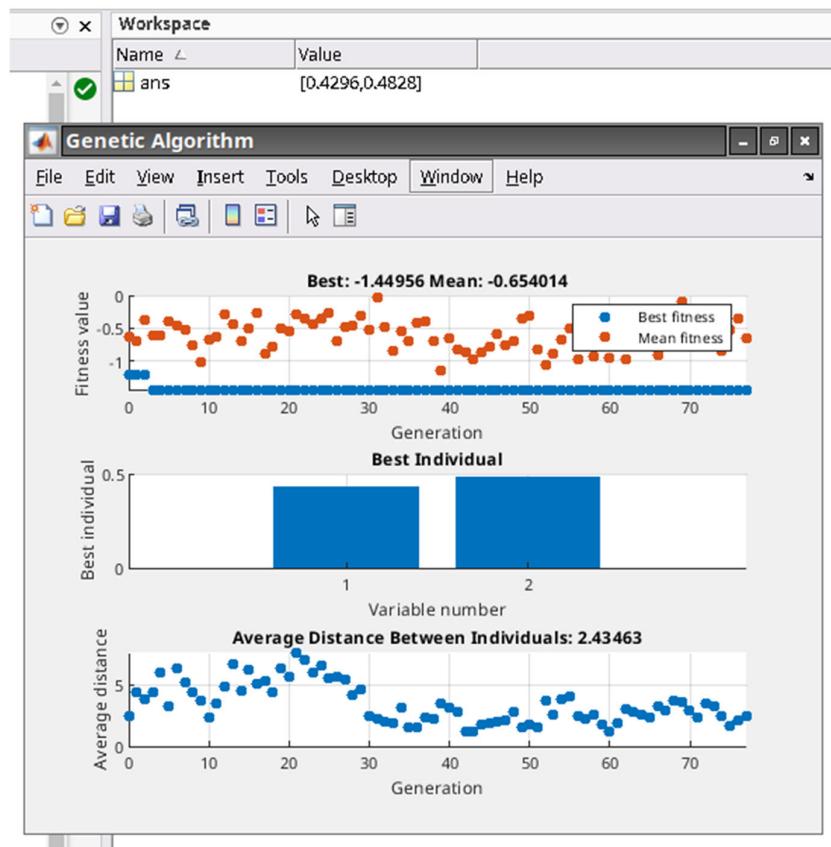
2-а оптимізація



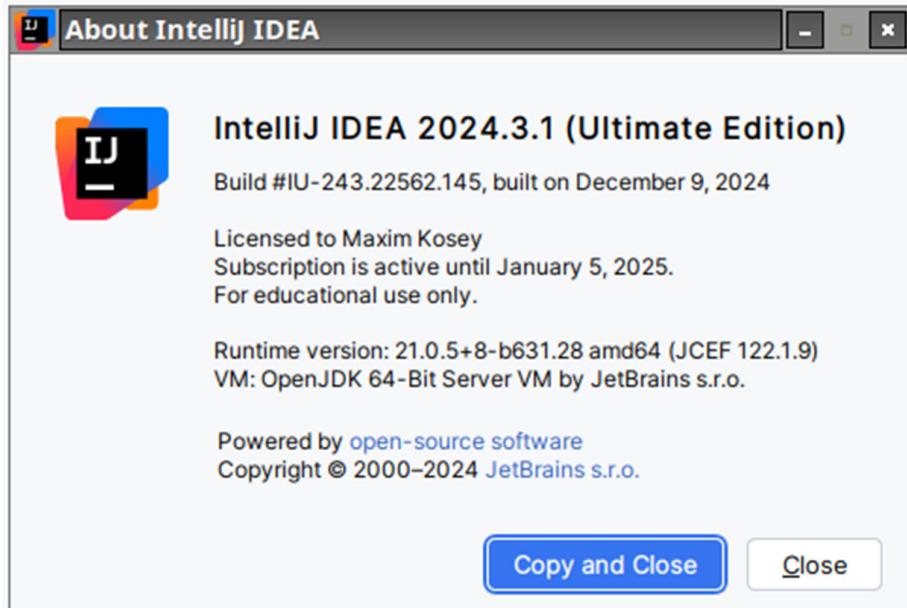
3-я оптимізація



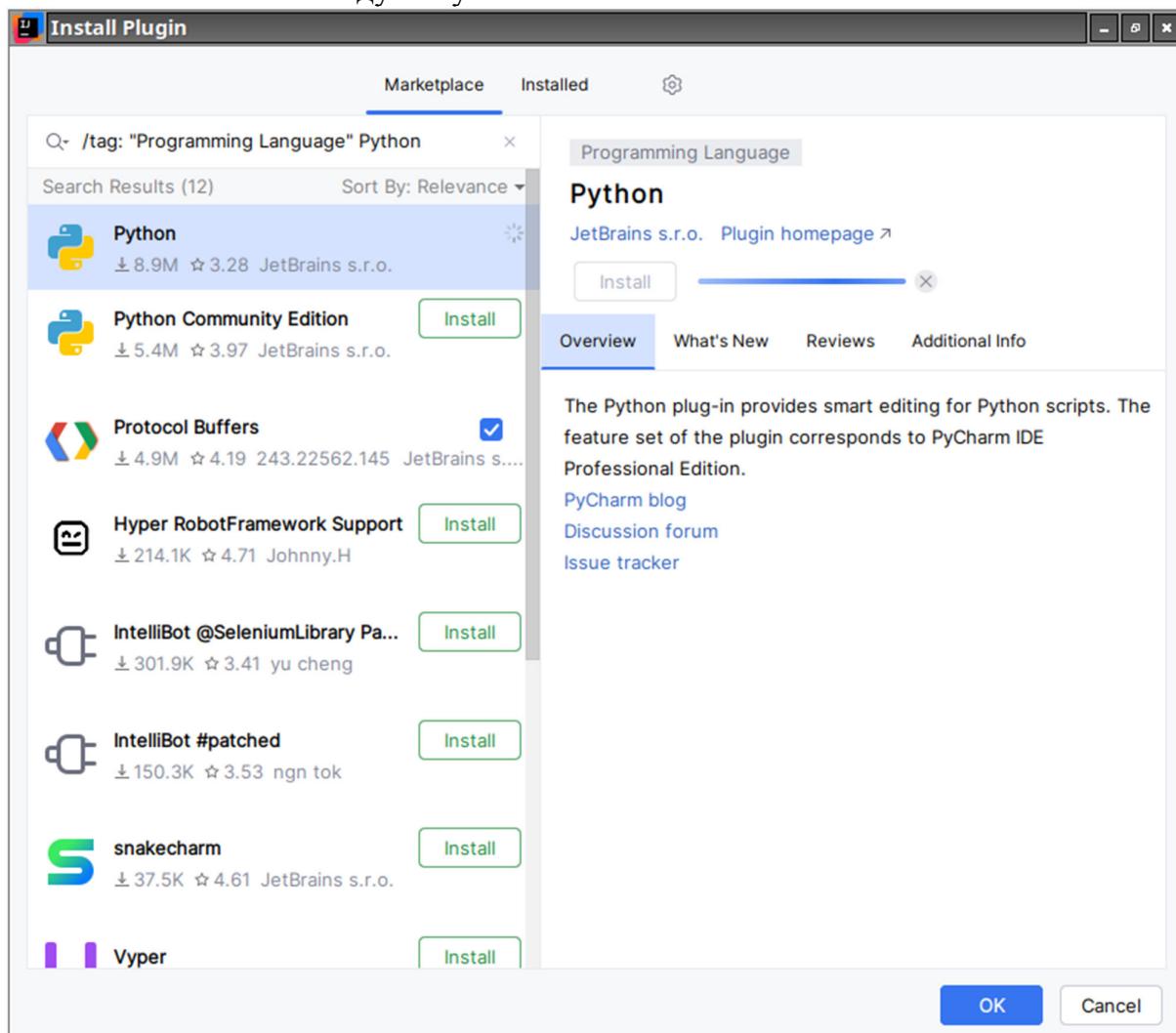
4-а оптимізація



Зробимо оптимізацію інших функцій за допомогою мови програмування Python та IDE IntelliJ IDEA



Встановлюємо модуль Python



Необхідно максимізувати функцію однієї змінної:

$$y(x) = 3\sqrt[3]{(x + 4)^2} - 2x - 8$$

Реалізація генетичного алгоритму на мові Python

```
func_max.py <input>
1 import numpy as np
2
3 # Функція, яку будемо оптимізувати (максимізувати)
4 def objective_function(x): 1 usage
5     return 3 * np.cbrt((x + 4) ** 2) - 2 * 2 - 8
6
7 # Налаштування генетичного алгоритму
8 POPULATION_SIZE = 100 # Розмір популяції
9 NUM_GENERATIONS = 100 # Кількість поколінь
10 MUTATION_RATE = 0.1 # Ймовірність мутації
11 CROSSOVER_RATE = 0.8 # Ймовірність кросинговеру
12 BOUNDS = (-1000, 1000) # Межі для x
13
14 # Ініціалізація популяції
15 def initialize_population(size, bounds): 1 usage
16     return np.random.uniform(bounds[0], bounds[1], size)
17
18 # Оцінка пристосованості (fitness) для кожного індивіда
19 def evaluate_fitness(population): 2 usages
20     return np.array([objective_function(x) for x in population])
21
22 # Турнірний відбір
23 def tournament_selection(population, fitness, k=3): 1 usage
24     selected = []
25     for _ in range(len(population)):
26         indices = np.random.choice(range(len(population)), k, replace=False)
27         best = indices[np.argmax(fitness[indices])]
28         selected.append(population[best])
29     return np.array(selected)
30
31 # Кросинговер (одноточковий)
32 def crossover(parent1, parent2): 1 usage
33     if np.random.rand() < CROSSOVER_RATE:
34         alpha = np.random.rand()
35         child1 = alpha * parent1 + (1 - alpha) * parent2
36         child2 = alpha * parent2 + (1 - alpha) * parent1
37         return child1, child2
38     return parent1, parent2
39
40 # Мутація (гаусівська)
41 def mutate(individual, bounds): 2 usages
42     if np.random.rand() < MUTATION_RATE:
43         mutation = np.random.normal(loc=0, scale=0.1) # Маленький стрибок
44         individual += mutation
45         individual = np.clip(individual, bounds[0], bounds[1]) # Залишаємо в межах
46     return individual
47
```

```
# Основний цикл генетичного алгоритму
48
49 population = initialize_population(POPULATION_SIZE, BOUNDS)
50 for generation in range(NUM_GENERATIONS):
51     fitness = evaluate_fitness(population)
52     selected = tournament_selection(population, fitness)
53
54     # Створення нового покоління
55     next_generation = []
56     for i in range(0, len(selected), 2):
57         parent1, parent2 = selected[i], selected[min(i + 1, len(selected) - 1)]
58         child1, child2 = crossover(parent1, parent2)
59         next_generation.append(mutate(child1, BOUNDS))
60         next_generation.append(mutate(child2, BOUNDS))
61
62     population = np.array(next_generation[:POPULATION_SIZE]) # Оновлення популяції
63
64     # Вивід результатів для кожного покоління
65     best_fitness = np.max(fitness)
66     best_individual = population[np.argmax(fitness)]
67     print(f"Generation {generation + 1}: Best Fitness = {best_fitness:.4f}, Best Individual = {best_individual:.4f}")
68
69 # Результати
70 final_fitness = evaluate_fitness(population)
71 best_fitness = np.max(final_fitness)
72 best_individual = population[np.argmax(final_fitness)]
73 print("\nFinal Solution:")
74 print(f"Best Fitness = {best_fitness:.4f}, Best Individual = {best_individual:.4f})
```

Запускаємо декілька разів оптимізацію функції

Результати оптимізації:

1-ша оптимізація

```
Run func_max x

Generation 91: Best Fitness = 287.7092, Best Individual = 994.4188
Generation 92: Best Fitness = 287.7076, Best Individual = 994.4484
Generation 93: Best Fitness = 287.7259, Best Individual = 994.4850
Generation 94: Best Fitness = 287.7259, Best Individual = 994.5101
Generation 95: Best Fitness = 287.7259, Best Individual = 994.5125
Generation 96: Best Fitness = 287.7333, Best Individual = 994.5730
Generation 97: Best Fitness = 287.7254, Best Individual = 994.5819
Generation 98: Best Fitness = 287.7316, Best Individual = 994.5971
Generation 99: Best Fitness = 287.7578, Best Individual = 994.7237
Generation 100: Best Fitness = 287.7550, Best Individual = 994.6472

Final Solution:
Best Fitness = 287.7549, Best Individual = 994.7748

Process finished with exit code 0
```

2-а оптимізація

```
Run func_max x

Generation 91: Best Fitness = 284.2470, Best Individual = 977.0524
Generation 92: Best Fitness = 284.2429, Best Individual = 977.0632
Generation 93: Best Fitness = 284.2541, Best Individual = 977.3185
Generation 94: Best Fitness = 284.2520, Best Individual = 977.1341
Generation 95: Best Fitness = 284.2698, Best Individual = 977.3193
Generation 96: Best Fitness = 284.3020, Best Individual = 977.3044
Generation 97: Best Fitness = 284.3020, Best Individual = 977.3365
Generation 98: Best Fitness = 284.2991, Best Individual = 977.4049
Generation 99: Best Fitness = 284.2978, Best Individual = 977.5192
Generation 100: Best Fitness = 284.3053, Best Individual = 977.5230

Final Solution:
Best Fitness = 284.3033, Best Individual = 977.5734

Process finished with exit code 0
```

3-я оптимізація

```
Run func_max x

Generation 91: Best Fitness = 276.0558, Best Individual = 936.7152
Generation 92: Best Fitness = 276.0558, Best Individual = 936.7359
Generation 93: Best Fitness = 276.0548, Best Individual = 936.7508
Generation 94: Best Fitness = 276.0611, Best Individual = 936.8281
Generation 95: Best Fitness = 276.0582, Best Individual = 936.8178
Generation 96: Best Fitness = 276.0661, Best Individual = 936.8696
Generation 97: Best Fitness = 276.1037, Best Individual = 936.8686
Generation 98: Best Fitness = 276.1037, Best Individual = 936.9199
Generation 99: Best Fitness = 276.1030, Best Individual = 936.8916
Generation 100: Best Fitness = 276.1445, Best Individual = 937.0299

Final Solution:
Best Fitness = 276.1328, Best Individual = 937.2549

Process finished with exit code 0
```

4-а оптимізація

```
Run func_max :  
Generation 91: Best Fitness = 278.5264, Best Individual = 948.9740  
Generation 92: Best Fitness = 278.5707, Best Individual = 949.0041  
Generation 93: Best Fitness = 278.5707, Best Individual = 948.9945  
Generation 94: Best Fitness = 278.5705, Best Individual = 949.1027  
Generation 95: Best Fitness = 278.5774, Best Individual = 949.1039  
Generation 96: Best Fitness = 278.5859, Best Individual = 949.2076  
Generation 97: Best Fitness = 278.5859, Best Individual = 949.2018  
Generation 98: Best Fitness = 278.6041, Best Individual = 949.2353  
Generation 99: Best Fitness = 278.6041, Best Individual = 949.3230  
Generation 100: Best Fitness = 278.6365, Best Individual = 949.3245  
  
Final Solution:  
Best Fitness = 278.6328, Best Individual = 949.5314  
  
Process finished with exit code 0
```

Мінімізуємо функцію

$$z(x, y) = (y - 3) * \exp(-x^2 - y^2)$$

Реалізація генетичного алгоритму на мові Python

```
func_max.py func_min.py x

1 import numpy as np
2
3 # Функція, яку будемо оптимізувати (мінімізувати)
4 def objective_function(x): 1 usage
5     return (x[1] - 3) * np.exp((-x[0] ** 2 - x[1] ** 2))
6
7 # Налаштування генетичного алгоритму
8 POPULATION_SIZE = 100 # Розмір популяції
9 NUM_GENERATIONS = 100 # Кількість поколінь
10 MUTATION_RATE = 0.1 # Ймовірність мутації
11 CROSSOVER_RATE = 0.8 # Ймовірність кросинговеру
12 BOUNDS = [(-1000, 1000), (-1000, 1000)] # Межі для кожної змінної
13
14 # Ініціалізація популяції
15 def initialize_population(size, bounds): 1 usage
16     return np.array([np.random.uniform(b[0], b[1], size) for b in bounds]).T
17
18 # Оцінка пристосованості (fitness) для кожного індивіда
19 def evaluate_fitness(population): 2 usages
20     return np.array([objective_function(ind) for ind in population])
21
22 # Турнірний відбір
23 def tournament_selection(population, fitness, k=3): 1 usage
24     selected = []
25     for _ in range(len(population)):
26         indices = np.random.choice(range(len(population)), k, replace=False)
27         best = indices[np.argmin(fitness[indices])]
28         selected.append(population[best])
29     return np.array(selected)
30
31 # Кросинговер (одноточковий)
32 def crossover(parent1, parent2): 1 usage
33     if np.random.rand() < CROSSOVER_RATE:
34         alpha = np.random.rand()
35         child1 = alpha * parent1 + (1 - alpha) * parent2
36         child2 = alpha * parent2 + (1 - alpha) * parent1
37         return child1, child2
38     return parent1, parent2
39
40 # Мутація (гаусівська)
41 def mutate(individual, bounds): 2 usages
42     if np.random.rand() < MUTATION_RATE:
43         mutation = np.random.normal(loc=0, scale=0.1, size=len(individual)) # Маленьки стрибки для всіх змінних
44         individual += mutation
45         for i in range(len(individual)):
46             individual[i] = np.clip(individual[i], bounds[i][0], bounds[i][1]) # Залишаємо в межах
47     return individual
48
49 # Основний цикл генетичного алгоритму
50 population = initialize_population(POPULATION_SIZE, BOUNDS)
51 for generation in range(NUM_GENERATIONS):
52     fitness = evaluate_fitness(population)
53     selected = tournament_selection(population, fitness)
54
55     # Створення нового покоління
56     next_generation = []
57     for i in range(0, len(selected), 2):
58         parent1, parent2 = selected[i], selected[min(i + 1, len(selected) - 1)]
59         child1, child2 = crossover(parent1, parent2)
60         next_generation.append(mutate(child1, BOUNDS))
61         next_generation.append(mutate(child2, BOUNDS))
62
63     population = np.array(next_generation[:POPULATION_SIZE]) # Оновлення популяції
64
```

```

65     # Вивід результатів для кожного покоління
66     best_fitness = np.min(fitness)
67     best_individual = population[np.argmin(fitness)]
68     print(f"Generation {generation + 1}: Best Fitness = {best_fitness:.4f}, Best Individual = {best_individual}")
69
70     # Результати
71     final_fitness = evaluate_fitness(population)
72     best_fitness = np.min(final_fitness)
73     best_individual = population[np.argmin(final_fitness)]
74     print("\nFinal Solution:")
75     print(f"Best Fitness = {best_fitness:.4f}, Best Individual = {best_individual}")
76

```

Запускаємо 4-ри рази оптимізацію функції

Результати оптимізації:

```

Generation 91: Best Fitness = -2.6762092166, Best Individual = [-0.02605734 -0.67296166]
Generation 92: Best Fitness = -2.9863936289, Best Individual = [-0.01765927 -0.64289014]
Generation 93: Best Fitness = -2.9863936289, Best Individual = [ 0.03406482 -0.54623113]
Generation 94: Best Fitness = -2.9863936289, Best Individual = [ 0.04499288 -0.55107808]
Generation 95: Best Fitness = -3.0263984968, Best Individual = [-0.04273168 -0.47469136]
Generation 96: Best Fitness = -3.0657788409, Best Individual = [ 0.15199523 -0.31876077]
Generation 97: Best Fitness = -3.0657788409, Best Individual = [-0.01233602 -0.37388505]
Generation 98: Best Fitness = -3.0496427517, Best Individual = [-0.04859707 -0.24905447]
Generation 99: Best Fitness = -3.0633778805, Best Individual = [ 0.07368695 -0.19545123]
Generation 100: Best Fitness = -3.0703608020, Best Individual = [ 0.02560894 -0.25537449]

Final Solution:
Best Fitness = -3.0788418247, Best Individual = [-0.01961725 -0.15240195]

Process finished with exit code 0

Generation 91: Best Fitness = 0.0000, Best Individual = [142.03867713 112.29529686]
Generation 92: Best Fitness = 0.0000, Best Individual = [142.02015135 112.21245813]
Generation 93: Best Fitness = 0.0000, Best Individual = [142.06073701 112.2456771 ]
Generation 94: Best Fitness = 0.0000, Best Individual = [141.99836997 112.3893895 ]
Generation 95: Best Fitness = 0.0000, Best Individual = [142.02013671 112.26456294]
Generation 96: Best Fitness = 0.0000, Best Individual = [141.94434081 112.29066454]
Generation 97: Best Fitness = 0.0000, Best Individual = [142.05033994 112.25318392]
Generation 98: Best Fitness = 0.0000, Best Individual = [142.05874443 112.25146501]
Generation 99: Best Fitness = 0.0000, Best Individual = [142.03463573 112.26187432]
Generation 100: Best Fitness = 0.0000, Best Individual = [142.04969827 112.27364041]

Final Solution:
Best Fitness = 0.0000, Best Individual = [142.04969827 112.27364041]

Process finished with exit code 0

```

Run func_min

```
Generation 91: Best Fitness = -0.0000000000, Best Individual = [-256.7383546 -276.40272625]
Generation 92: Best Fitness = -0.0000000000, Best Individual = [-256.67420291 -276.33956858]
Generation 93: Best Fitness = -0.0000000000, Best Individual = [-256.7158201 -276.48036606]
Generation 94: Best Fitness = -0.0000000000, Best Individual = [-256.69948878 -276.68115406]
Generation 95: Best Fitness = -0.0000000000, Best Individual = [-256.76130594 -276.45406984]
Generation 96: Best Fitness = -0.0000000000, Best Individual = [-256.78829679 -276.47233076]
Generation 97: Best Fitness = -0.0000000000, Best Individual = [-256.73747309 -276.39951312]
Generation 98: Best Fitness = -0.0000000000, Best Individual = [-256.8085565 -276.40535527]
Generation 99: Best Fitness = -0.0000000000, Best Individual = [-256.77220394 -276.48607236]
Generation 100: Best Fitness = -0.0000000000, Best Individual = [-256.77058231 -276.44199633]

Final Solution:
Best Fitness = -0.0000000000, Best Individual = [-256.77058231 -276.44199633]

Process finished with exit code 0
```

```
Generation 91: Best Fitness = -0.0000000000, Best Individual = [-127.80466191 -87.77510081]
Generation 92: Best Fitness = -0.0000000000, Best Individual = [-127.83121439 -87.68964367]
Generation 93: Best Fitness = -0.0000000000, Best Individual = [-127.84600358 -87.75079434]
Generation 94: Best Fitness = -0.0000000000, Best Individual = [-127.83847784 -87.6940746 ]
Generation 95: Best Fitness = -0.0000000000, Best Individual = [-127.80921783 -87.73887308]
Generation 96: Best Fitness = -0.0000000000, Best Individual = [-127.83421405 -87.72124067]
Generation 97: Best Fitness = -0.0000000000, Best Individual = [-127.8323798 -87.7576447]
Generation 98: Best Fitness = -0.0000000000, Best Individual = [-127.86290345 -87.69956369]
Generation 99: Best Fitness = -0.0000000000, Best Individual = [-127.82108299 -87.70242531]
Generation 100: Best Fitness = -0.0000000000, Best Individual = [-127.65241912 -87.76254589]

Final Solution:
Best Fitness = -0.0000000000, Best Individual = [-127.65241912 -87.76254589]

Process finished with exit code 0
```

ВИСНОВКИ

В результаті виконаної лабораторної роботи за допомогою пакета Matlab 2024a та мови програмування Python була досліджена оптимізація функцій за допомогою генетичних алгоритмів.

Усі матеріали викладенні у репозіторії GitHub, за посиланням <https://github.com/Max11mus/LAB7-Modern-Methods-and-Models-of-Intelligent-Control-Systems>.