

ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

профессор

должность, уч. степень, звание

подпись, дата

Ю.А. Скобцов

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7

Оптимизация функций многих переменных с помощью роевых алгоритмов

По дисциплине: Эволюционные методы проектирования программно-
информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.

4134к

подпись, дата

Столяров Н.С.

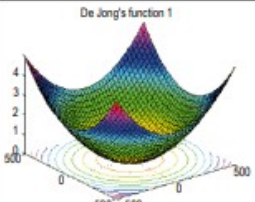

инициалы, фамилия

Санкт-Петербург
2024

Цель работы:

оптимизация функций многих переменных методом роевого интеллекта. Графическое отображение результатов оптимизации.

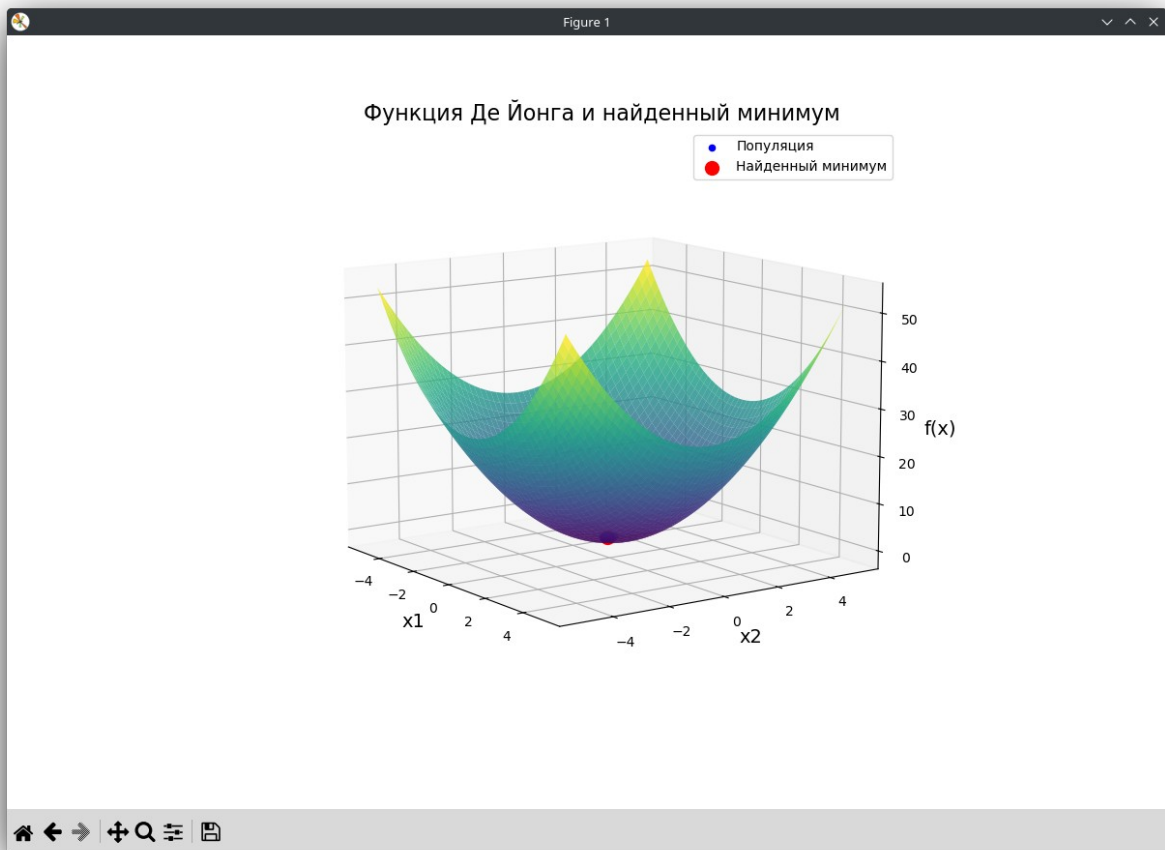
Вариант 1:**Индивидуальные задания на лабораторную работу №2**

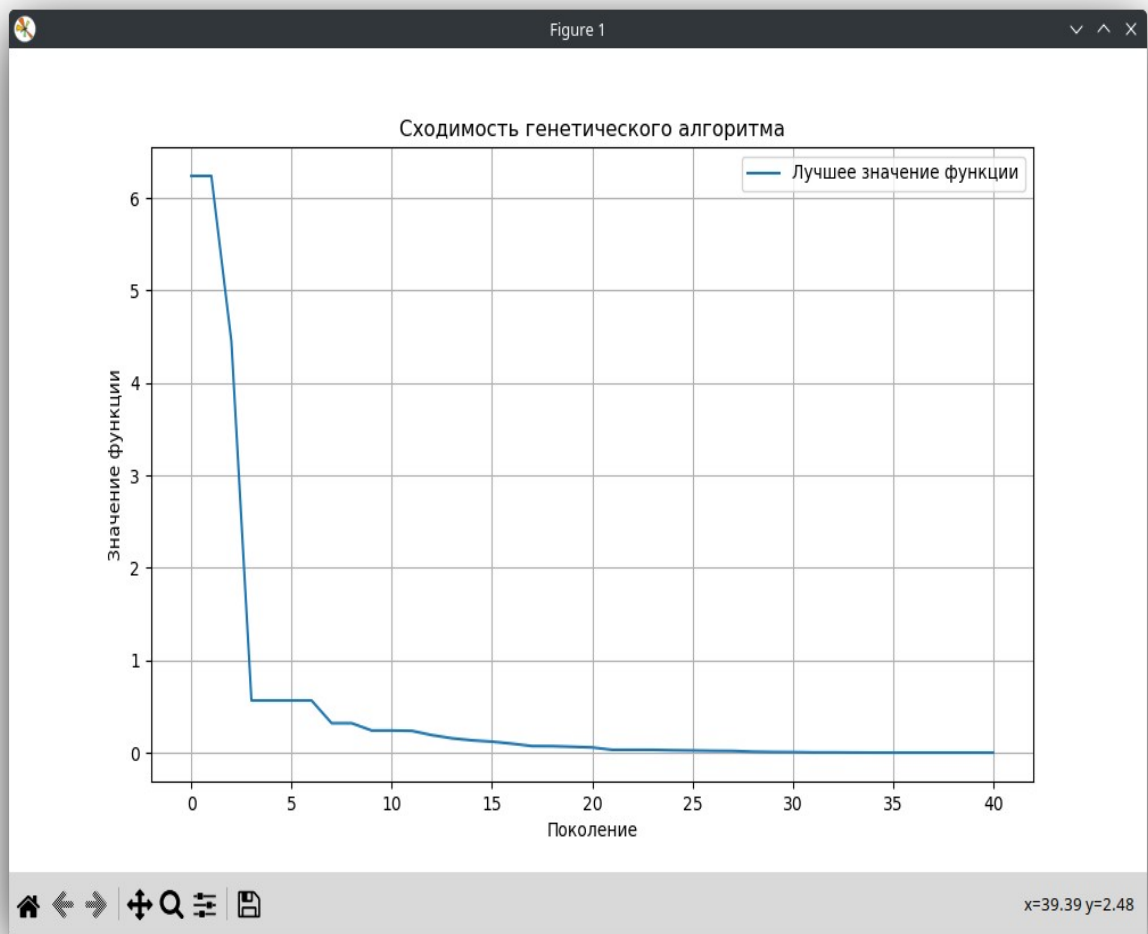
№ вв.	Название	Оптимум	Вид функции	График функции
1	De Jong's function 1	global minimum $f(x)=0; x(i)=0,$ $I=1:n.$	$f_1(x) = \sum_{i=1}^n x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f1(x)=\text{sum}(x(i)^2),$ $i=1:n;$	 <p>De Jong's function 1</p>
2	Axis parallel	global minimum	n	 <p>Axis parallel hyper-ellipsoid</p>

Задание:

1. Разработать программу, использующую РА для нахождения оптимума функции согласно таблице вариантов, приведенной в приложении А. Для всех Benchmark-ов оптимумом является минимум. Программу выполнить на встроенном языке пакета Python.
2. Для $n=2$ вывести на экран график данной функции с указанием найденного экстремума, точек популяции. Для вывода графиков использовать стандартные возможности пакета Python. Предусмотреть возможность пошагового просмотра процесса поиска решения.
3. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:
 - i. число особей в популяции
 - ii. вероятность мутации.
 - iii. Критерий остановки вычислений – повторение лучшего результата заданное количество раз или достижение популяцией определенного возраста (например, 100 эпох).
4. Повторить процесс поиска решения для $n=3$, $n=5$, $n=10$, сравнить результаты, скорость работы программы.

Выполнение:





```
7: bash — Konsole
best_fitness: 0.03151507777556105, best_solution: [ 0.09853967 -0.06736241 0.09734855 -0.06481709 0.059
01004]
best_fitness: 0.02705301531370091, best_solution: [ 0.09189676 -0.08112848 0.02249278 -0.10705325 0.0077
3598]
best_fitness: 0.02443231646771653, best_solution: [ 0.09853967 -0.04011477 0.09734855 -0.00685519 0.0599
1094]
best_fitness: 0.02090284160384256, best_solution: [ 0.09853967 -0.04011477 0.09734855 -0.00685519 0.0077
3598]
best_fitness: 0.01955460846365648, best_solution: [ 0.07715344 -0.02155105 0.08204825 -0.07966019 0.0077
3598]
best_fitness: 0.011384126680661704, best_solution: [ 0.00593888 -0.08112848 0.02249278 -0.06481709 0.007
73598]
best_fitness: 0.007920230725714846, best_solution: [ 0.00593888 -0.05583847 0.02249278 -0.06481709 0.007
73598]
best_fitness: 0.00669273667237158, best_solution: [-0.05448303 0.01687576 0.04466656 -0.03721018 0.0077
3598]
best_fitness: 0.0035447590317115914, best_solution: [-0.01015983 -0.0014116 0.04466656 -0.03721018 0.00
773598]
best_fitness: 0.0035447590317115914, best_solution: [-0.01015983 -0.0014116 0.04466656 -0.03721018 0.00
773598]
best_fitness: 0.0022242087823745684, best_solution: [ 0.00618041 0.01687576 0.02091142 -0.03721018 0.00
890701]
best_fitness: 0.0013100225571827566, best_solution: [-0.01015983 0.01687576 0.02091142 0.02013422 0.00
890701]
best_fitness: 0.0013100225571827566, best_solution: [-0.01015983 0.01687576 0.02091142 0.02013422 0.00
890701]
best_fitness: 0.0013100225571827566, best_solution: [-0.01015983 0.01687576 0.02091142 0.02013422 0.00
890701]
best_fitness: 0.0013100225571827566, best_solution: [-0.01015983 0.01687576 0.02091142 0.02013422 0.00
890701]
best_fitness: 0.0013100225571827566, best_solution: [-0.01015983 0.01687576 0.02091142 0.02013422 0.00
890701]
best_fitness: 0.0013100225571827566, best_solution: [-0.01015983 0.01687576 0.02091142 0.02013422 0.00
890701]
best_fitness: 0.0013100225571827566, best_solution: [-0.01015983 0.01687576 0.02091142 0.02013422 0.00
890701]
best_fitness: 0.001186854641396446, best_solution: [-0.01015983 0.02177632 -0.01131386 -0.02053231 0.007
73598]
Найденное решение: [-0.01015983 0.02177632 -0.01131386 -0.02053231 0.00773598]
Значение функции в найденной точке: 0.001186854641396446
stolar@stolar-NMH-WCX9:~/PROJECTS/Programming-GUAP/ЭМППИС/7$
```

Выводы:

В результате проведенной работы была успешно реализована оптимизация многопараметрической функции методом роевого интеллекта (PSO). Полученные графические результаты наглядно продемонстрировали эффективность данного метода в нахождении глобального минимума, что подтверждает его применимость для решения задач оптимизации в многомерных пространствах.

Листинг

```
import random
import time
import matplotlib.pyplot as plt
from matplotlib import animation
import numpy as np

# Константы генетического алгоритма
POPULATION_SIZE = 50 # количество индивидуумов в популяции
MAX_GENERATIONS = 50 # максимальное количество поколений
P_CROSSOVER = 0.9 # вероятность скрещивания
P_MUTATION = 0.1 # вероятность мутации
N_VECTOR = 3 # размерность задачи (количество генов в хромосоме)
LIMIT_VALUE_TOP = 5.12 # верхняя граница области поиска
LIMIT_VALUE_DOWN = -5.12 # нижняя граница области поиска
RANDOM_SEED = 42 # для воспроизводимости
random.seed(RANDOM_SEED)

class Individual(list):
    """Класс для представления индивидуумов популяции"""
    def __init__(self, *args):
        super().__init__(*args)
        # print(args)
        self.value = 0

# Функция приспособленности (De Jong's function 1)
def fitness_function(f):
    return sum(x**2 for x in f)

# Создание индивидуумов и популяции
def individual_creator():
    return Individual([random.uniform(LIMIT_VALUE_DOWN, LIMIT_VALUE_TOP) for _ in
range(N_VECTOR)])

def population_creator(n=0):
    return [individual_creator() for _ in range(n)]

# Начальная популяция
population = population_creator(POPULATION_SIZE)
fitness_values = list(map(fitness_function, population))
for individual, fitness_value in zip(population, fitness_values):
    individual.value = fitness_value

# Для статистики
min_fitness_values = []
mean_fitness_values = []

# Сортировка начальной популяции по значению приспособленности
population.sort(key=lambda ind: ind.value)

def clone(individual):
    """Клонирование индивидуума"""
```

```

clone_ind = Individual(individual[:])
clone_ind.value = individual.value
return clone_ind

def selection(population, n=POPULATION_SIZE):
    """Турнирная селекция"""
    offspring = []
    for _ in range(n):
        participants = random.sample(population, 4)
        winner = min(participants, key=lambda ind: ind.value)
        offspring.append(winner)
    return offspring

def crossover(parent1, parent2):
    """Одноточечное скрещивание"""
    point = random.randint(1, len(parent1) - 1)
    parent1[point:], parent2[point:] = parent2[point:], parent1[point:]

def mutate(individual, mutation_prob=1.0 / N_VECTOR):
    """Мутация гена"""
    for i in range(len(individual)):
        if random.random() < mutation_prob:
            individual[i] += random.uniform(-0.5, 0.5)
            individual[i] = max(min(individual[i], LIMIT_VALUE_TOP), LIMIT_VALUE_DOWN)

# Анимация процесса оптимизации
generation_counter = 0

def animate(frame):
    global generation_counter, population

    ax.clear()
    ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10, alpha=0.3)
    ax.set_title(f"Поколение: {generation_counter}", fontsize=10)
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Fitness")

    # Отображение текущих индивидов популяции
    for ind in population:
        ax.scatter(ind[0], ind[1], ind.value, color='red', s=20)

    # Вывод значений x и y для каждого индивидуума
    if generation_counter % 5 == 0: # Выводить каждые 5 поколений
        print(f"Поколение {generation_counter}:")
        for ind in population:
            # print(ind)
            # print(f" Индивидуум: x={ind[0]:.6f}, y={ind[1]:.6f}, Fitness={ind.value:.2f}")
            print(f" Индивидуум: x={ind[0]:.6f}, y={ind[1]:.6f}, z={ind.value:.6f}")

    # Селекция, кроссовер и мутация
    offspring = selection(population)

```

```

offspring = list(map(clone, offspring))

for child1, child2 in zip(offspring[::2], offspring[1::2]):
    if random.random() < P_CROSSOVER:
        crossover(child1, child2)

for mutant in offspring:
    if random.random() < P_MUTATION:
        mutate(mutant)

# Вычисление новой функции приспособленности
fresh_fitness_values = list(map(fitness_function, offspring))
for ind, fitness_value in zip(offspring, fresh_fitness_values):
    ind.value = fitness_value

# Обновление популяции
population[:] = offspring

# Сбор статистики
fitness_values = [ind.value for ind in population]
std_fitness = np.std(fitness_values)
min_fitness = min(fitness_values)
mean_fitness = sum(fitness_values) / len(fitness_values)
min_fitness_values.append(min_fitness)
mean_fitness_values.append(mean_fitness)

if generation_counter % 5 == 0:
    print(f"Минимальная приспособленность: {min_fitness:.6f}\nСредняя
    приспособленность: {mean_fitness:.6f}\nСтандартное отклонение: {std_fitness:.6f}")

    generation_counter += 1

# Создание сетки для визуализации функции
X = np.linspace(LIMIT_VALUE_DOWN, LIMIT_VALUE_TOP, 100)
Y = np.linspace(LIMIT_VALUE_DOWN, LIMIT_VALUE_TOP, 100)
X, Y = np.meshgrid(X, Y)
Z = X**2 + Y**2

# Настройка графиков
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 1, 1, projection='3d')

ani = animation.FuncAnimation(fig, animate, frames=MAX_GENERATIONS, interval=50,
repeat=False)
plt.show()

```