ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| профессор |  |  |  | Ю.А. Скобцов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| Простой генетический алгоритм |
| по курсу: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134 |  |  |  | Д. В. Самарин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

# Задание

1. Разработать простой генетический алгоритм для нахождения оптимума заданной по варианту функции одной переменной̆.

Вариант 14. Функция cos(x-0.5) , найти максимум.

1. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:

* число особей в популяции,
* вероятность кроссинговера,
* вероятность мутации.

1. Вывести на экран график данной функции с указанием найденного экстремума для каждого поколения.
2. Сравнить найденное решение с действительным.

# Краткие теоретические сведения

ГА используют принципы и терминологию, заимствованные у биологической науки – генетики. В ГА каждая особь представляет потенциальное решение некоторой проблемы. В классическом ГА особь кодируется строкой двоичных символов – хромосомой, каждый бит которой называется геном. Множество особей – потенциальных решений составляет популяцию. Поиск (суб)оптимального решения проблемы выполняется в процессе эволюции популяции - последовательного преобразования одного конечного множества решений в другое с помощью генетических операторов репродукции, кроссинговера и мутации.

ГА позволяют найти оптимальное решение проблемы с помощью операторов репродукции, скрещивания (кроссинговера) и мутации.

# Программа

import numpy as np  
import matplotlib.pyplot as plt  
import random  
  
# Целевая функция  
def target\_function(x):  
 if x == 0:  
 return float('-inf') # Выколотая точка в отрезке, заданном условием  
 return np.cos(x - 0.5) / abs(x)  
  
  
# Генерация начальной популяции  
def generate\_population(pop\_size, x\_min, x\_max):  
 return [random.uniform(x\_min, x\_max) for i in range(pop\_size)]  
  
  
# Оценка приспособленности особей  
def fitness(population):  
 return [target\_function(x) for x in population]  
  
  
# Выбор родителей  
def select\_parents(population, fitness\_values):  
 parents = []  
 for k in range(len(population)):  
 i, j = random.sample(range(len(population)), 2)  
 if fitness\_values[i] > fitness\_values[j]:  
 parents.append(population[i])  
 else:  
 parents.append(population[j])  
 return parents  
  
  
# Кроссинговер  
def crossover(parent1, parent2):  
 return (parent1 + parent2) / 2  
  
  
# Мутация  
def mutate(population, mutation\_rate, x\_min, x\_max):  
 for i in range(len(population)):  
 if random.random() < mutation\_rate:  
 population[i] = random.uniform(x\_min, x\_max)  
 return population  
  
  
# Построение графика  
def plot\_generation(population, generation, x\_min, x\_max, best\_values, count):  
 x = np.linspace(x\_min, x\_max, 1000 \* count)  
 y = [target\_function(val) for val in x]  
  
 plt.figure()  
 plt.plot(x, y, label='f(x) = cos(x-0.5) / |x|')  
 plt.scatter(population, [target\_function(ind) for ind in population], color='red', label='Population')  
  
 plt.title(f'График функции и популяции на поколении {generation}')  
 plt.xlabel('x')  
 plt.ylabel('f(x)')  
 plt.grid(True)  
 plt.legend()  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Инициализация переменных  
 population\_size = 300 # Размер популяции  
 p\_crossover = 0.5 # Вероятность появления потомка  
 p\_mutation = 0.001 # Вероятность мутации у потомка  
 x\_min = -10  
 x\_max = 10  
 population = generate\_population(population\_size, x\_min, x\_max) # Начальная популяция  
 generation = 0 # Номер поколения  
 max\_generations = 50 # Лимит по числу поколений  
 best\_fitness = float('-inf') # Текущая лучшая особь  
 best\_values = [] # Массив лучших особей  
 plot\_steps = [0, 10, 25, 40] # Номера популяций для которых будут строиться графики  
 count = 1 # Вспомогательный счетчик для построения графика  
  
 while generation < max\_generations:  
 fitness\_values = fitness(population)  
 best\_idx = np.argmax(fitness\_values)  
 best\_value = population[best\_idx]  
 best\_values.append(best\_value)  
  
 # Оценка улучшения  
 if fitness\_values[best\_idx] > best\_fitness:  
 best\_fitness = fitness\_values[best\_idx]  
  
 # Построение графика текущего поколения  
 plt.ion()  
 if generation in plot\_steps:  
 plot\_generation(population, generation, x\_min, x\_max, best\_values, count)  
 plt.draw()  
 plt.pause(5)  
 plt.close()  
 count += 100  
 plt.ioff()  
  
 # Селекция родителей  
 parents = select\_parents(population, fitness\_values)  
  
 # Создание новой популяции через кроссинговер  
 new\_population = []  
 for i in range(0, population\_size, 2):  
 parent1, parent2 = random.sample(parents, 2)  
 if random.random() < p\_crossover:  
 offspring1 = crossover(parent1, parent2)  
 offspring2 = crossover(parent2, parent1)  
 else:  
 offspring1, offspring2 = parent1, parent2  
 new\_population.extend([offspring1, offspring2])  
  
 # Мутация  
 population = mutate(new\_population, p\_mutation, x\_min, x\_max)  
 generation += 1  
  
 # Построение графика для последнего поколения  
 plot\_generation(population, generation, x\_min, x\_max, best\_values, count)  
  
 # Вывод  
 print(f"Population size = {population\_size}, P crossover = {p\_crossover}, P mutation = {p\_mutation}")  
 print(f"Maximum found at x = {best\_values[-1]}, at generation = {generation}")

# Результат работы программы

Population size = 300, P crossover = 0.5, P mutation = 0.001

Maximum found at x = -7.740583650858818e-18, at generation = 50

График функции для 0 поколения:

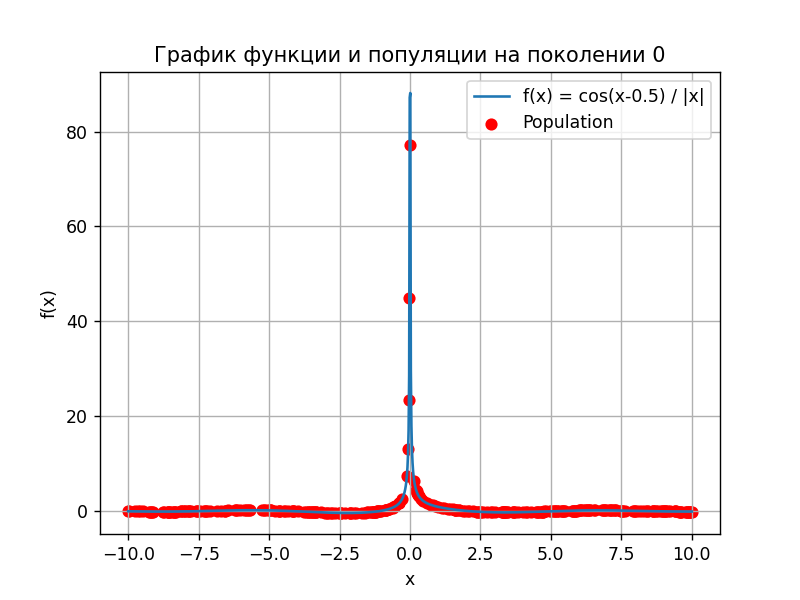


График функции для 25 поколения:

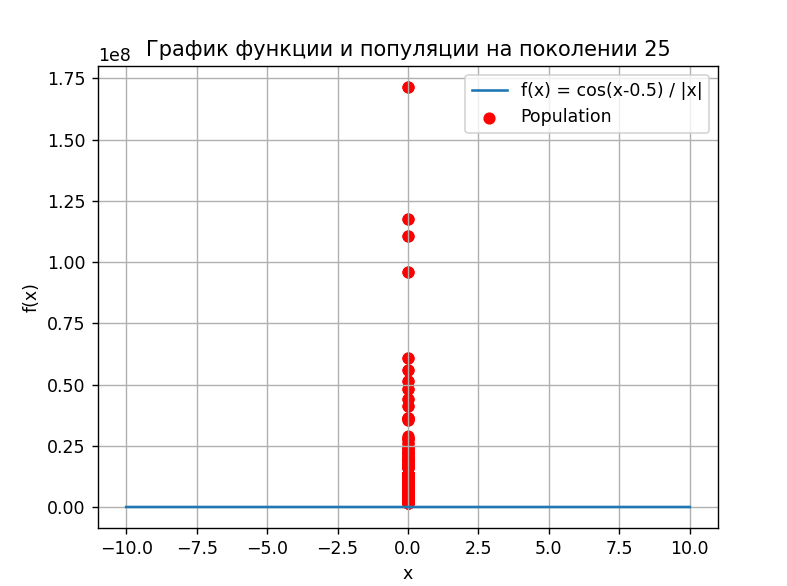
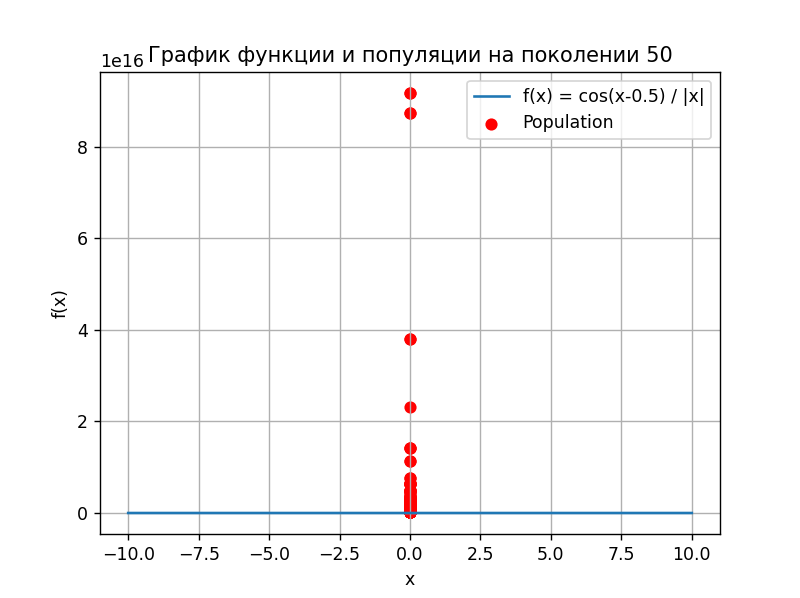


График функции для 50 поколения:



# Контрольный вопрос

14. Исследуйте зависимость работы ПГА от значения вероятности ОК Pc .

При низкой вероятности кроссинговера потомки в основном будут наследовать ген одного родителя, что приведет к слабой вариативности и увеличению количества поколений для достижения оптимального решения, из-за чего произойдет избыточное использование вычислительных мощностей ЭВМ. Но если вероятность кроссинговера будет слишком высокой, то вырастет риск чрезмерного смешения генов и привести к разрушению хороших решений, что опять-таки снизит эффективность алгоритма

# Вывод

В ходе выполнения лабораторной работы были изучены принципы реализации простых генетических алгоритмов.