МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

д-р техн. наук, профессор		Ю.А. Скобцов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Простой генетический алгоритм

по дисциплине: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №	Z1431		М.Д.Быстров
	номер группы	подпись, дата	инициалы, фамилия

Студенческий билет № 2021/3572

Санкт-Петербург 2025

Оглавление

Индивидуальное задание

1. Разработать простой генетический алгоритм для нахождения оптимума заданной по варианту функции одной переменной (таб. 1.1).

Вид экстремума:

Вариант	Вид экстремума
<=15	Максимум
>15	Минимум

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

- число особей в популяции

- вероятность кроссинговера, мутации.

3. Вывести на экран график данной функции с указанием найденного экстремума для каждого поколения.

4. Сравнить найденное решение с действительным.

Вариант 4.

Вариант	Вид функции	Промежуток нахождения решения
1	(1.85-x)*cos(3.5x-0.5)	$x \in [-10,10]$
2	Cos(exp(x))/sin(ln(x))	$x \in [2,4]$
3	$Sin(x)/x^2$	$x \in [3.1,20]$
4	$Sin(2x)/x^2$	$x \in [-20,-3.1]$

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.

ЭВ используют следующие механизмы естественной эволюции:

1) Первый принцип основан на концепции выживания сильнейших и естественного отбора по Дарвину, который был сформулирован им в 1859 году в книге «Происхождение видов путем естественного отбора». Согласно Дарвину особи, которые лучше способны решать задачи в своей среде, выживают и больше размножаются (репродуцируют). В генетических алгоритмах каждая особь представляет собой решение некоторой проблемы. По аналогии с этим принципом особи с лучшими значениями целевой (фитнесс) функции имеют большие шансы выжить и репродуцировать. Формализация этого принципа, как мы увидим далее, дает оператор репродукции.

2) Второй принцип обусловлен тем фактом, что хромосома потомка состоит из частей полученных из хромосом родителей. Этот принцип был открыт в 1865 году Менделем. Его формализация дает основу для оператора скрещивания (кроссинговера).

3) Третий принцип основан на концепции мутации, открытой в 1900 году де Вре. Первоначально этот термин использовался для описания существенных (резких) изменений свойств потомков и приобретение ими свойств, отсутствующих у родителей. По аналогии с этим принципом генетические алгоритмы используют подобный механизм для резкого

изменения свойств потомков и тем самым, повышают разнообразие (изменчивость) особей в популяции (множестве решений). Эти три принципа составляют ядро ЭВ. Используя их, популяция (множество решений данной проблемы) эволюционирует от поколения к поколению.

Эволюцию искусственной популяции – поиска множества решений некоторой проблемы формально можно описать алгоритмом, который представлен на рис.1.1.

ГА берет множество параметров оптимизационной проблемы и кодирует их последовательностями конечной длины в некотором конечном алфавите (в простейшем случае двоичный алфавит «0» и «1») .

Предварительно простой ГА случайным образом генерирует начальную популяцию стрингов (хромосом). Затем алгоритм генерирует следующее поколение (популяцию), с помощью трех основных генетических операторов:

1) Оператор репродукции (ОР);

2) Оператор скрещивания (кроссинговера, ОК);

3) Оператор мутации (ОМ).

Генетические операторы являются математической формализацией приведенных выше трех основополагающих принципов Дарвина, Менделя и де Вре естественной эволюции.

ГА работает до тех пор, пока не будет выполнено заданное количество поколений (итераций) процесса эволюции или на некоторой генерации будет получено заданное качество или вследствие преждевременной сходимости при попадании в некоторый локальный оптимум.

В каждом поколении множество искусственных особей создается с использованием старых и добавлением новых с хорошими свойствами. Генетические алгоритмы - не просто случайный поиск, они эффективно используют информацию накопленную в процессе эволюции. В отличие от других методов оптимизации ГА оптимизируют различные области пространства решений одновременно и более приспособлены к нахождению

новых областей с лучшими значениями целевой функции за счет объединения квазиоптимальных решений из разных популяций.

Результаты выполнения индивидуального задания

Лабораторная работа выполнена на языке программирования Python.

Текст программы:

```
import random
import math
import matplotlib
import matplotlib.axes
import matplotlib.pyplot
import numpy
import scipy
import scipy.optimize

def generate_individ(len):
    """Генерация одной особи
    (из одной хромосомы) длиной len"""
    individ = []

    for i in range(0, len):
        bit=random.choice([0, 1])
        individ += [bit]

    return individ;

def generate_population(num, len):
    """Сгенерировать популяцию размером num и длиной хромосом len"""
    population=[]
    for i in range(0, num): population += [generate_individ(len)]
    return population

def bin2dec(chrom):
    """Из бит в десятичное"""
    dec=int()
    for i in range(0, len(chrom)):
        dec += math.pow(chrom[i] * 2, len(chrom) - i - 1)
    return dec

def phenotype(chrom, start, end, num):
    """Фенотип (значение) из хромосомы (бит)"""
    return start + bin2dec(chrom) * ((end-start) / num)

def x_to_y(f, x) -> list:
    """Массив x в массив y"""
    y = []
    for v in x: y.append(f(v))
    return y

def reproduction(population: list, f, start, end, num):
```

```
    """Оператор репродукции
    """
    #todo
    intermediate_population = [];

    individ_values = []

    # сумма всех значений и значение для каждой особи
    for v in population:
        x = phenotype(v, start, end, num)
        y = f(x)
        individ_values += [y]

    # смещаем, чтобы не было отрицательных значений, считаем сумму
    min_poten = min(individ_values)
    offset = 0 - min_poten

    for i in range(len(individ_values)):
        individ_values[i] += offset

    values_sum = sum(individ_values)

    # все особи одинаковы
    if values_sum == 0:
        return population.copy()

    potentials = []

    # подсчет потенциала для каждой особи
    for i in range(len(individ_values)):
        val = individ_values[i]
        prob = val / values_sum
        potentials += [prob]

    sum_potentials = sum(potentials)

    # выбор такого же кол-ва особей
    for i in range(len(population)):

        # крутите барабан
        shot = random.random() * sum_potentials
        individ_num = 0
        tmp_sum = float()

        # определяем куда попали барабаном
        for j in range(len(population)):
            individ_num = j
            tmp_sum += potentials[individ_num]
            if (tmp_sum >= shot):
```

```
                break

        # добавляем выбранную особь в промежуточную популяцию
        individ = population[individ_num]
        intermediate_population += [individ]

    return intermediate_population

def crossingover(pop : list, p, f, start, end, num):

    """Кроссинговер - скрещивание популяции"""
    pop_copy = pop.copy()
    pop_result = []

    pop_len = len(pop)

    while (len(pop_copy) > 0):
        p1 = pop_copy.pop(random.randrange(len(pop_copy)))
        p2 = pop_copy.pop(random.randrange(len(pop_copy)))

        pop_result += [p1.copy(), p2.copy()]

        if (random.random() < p):
            #выполняем скрещивание
            bit = random.randrange(len(p1))

            part = p1[bit:]
            p1[bit:] = p2[bit:]
            p2[bit:] = part

            pop_result += [p1, p2]

    def cmp(v): return f(phenotype(v, start, end, num))

    pop_result.sort(key = cmp, reverse=True)

    pop_result = pop_result[:pop_len]

    return pop_result

def mutation(pop: list, p):
    """Выполнить мутацию"""
    pop_copy = pop.copy()

    for v in pop_copy:
        if (random.random() < p):
            bit = random.randrange(len(v))
            v[bit] = 1 if v[bit] == 0 else 0

    return pop_copy
```

```
def draw_plot(f, start, end, num = int(math.pow(2, 15))):
    """Нарисовать график функции"""
    x = numpy.linspace(start, end, num)
    y = []
    for i in x: y += [f(i)];
    matplotlib.pyplot.scatter(x, y, s=0.5)

def draw_population(f, pop, start, end, num):
    """Нарисовать точки популяции"""
    for individ in pop:
        x=phenotype(individ, start, end, num)
        matplotlib.pyplot.scatter(x, f(x), s=15, color="red")

def population_phenotypes(phenotype, start, end, num, pop):
    """Получить фенотипы (значения) для популяции"""
    xs = []
    for v in pop:
        xs += [phenotype(v, start, end, num)]
    return xs

def genetic(
        f,
        start,
        end,
        chrom_len,
        mutation_prob,
        crossing_prob,
        population_size,
        localization,
        max_steps,
        is_draw):

    """Генетический алгоритм"""

    combinations_num = 2**chrom_len

    population = generate_population(population_size, chrom_len)

    if (is_draw):
        matplotlib.pyplot.figure(num = "Начальное поколение")
        draw_plot(f, start, end)
        draw_population(f, population, start, end, combinations_num)

    # matplotlib.pyplot.legend()

    step = 1

    while True:
```

```
        if step + 1 > max_steps:
            break

        step += 1

        reproducted_population = reproduction(population, f, start, end,
combinations_num)
        cross_population = crossingover(reproducted_population, crossing_prob, f,
start, end, combinations_num)
        mutated_population = mutation(cross_population, mutation_prob)

        population = mutated_population

        xs = population_phenotypes(phenotype, start, end, combinations_num,
mutated_population)

        ys = x_to_y(f, xs)

        y_max = max(ys)
        x_best = xs[ys.index(y_max)]

        if (is_draw):
            matplotlib.pyplot.figure(num = f"Поколение {step}")
            draw_plot(f, start, end)
            draw_population(f, population, start, end, combinations_num)
            matplotlib.pyplot.scatter([x_best], [y_max], s=25, color="green",
label=f"Максимум = {y_max}")
            matplotlib.pyplot.legend()

        if (abs(min(xs) - max(xs)) < localization):
            break

    x = max(xs)

    print("Работа генетического алгоритма завершена")
    print(f"Размер популяции {population_size}")
    print(f"Вероятность кроссинговера {crossing_prob}")
    print(f"Вероятность мутации {mutation_prob}")
    print(f"Ограничивающая локализация решений: {localization}")
    print(f"Выполнено {step} шагов")

    print(f"Итоговая локализация решений - [{min(xs)}; {max(xs)}] длиной
{abs(min(xs) - max(xs))}")
    print(f"Решение: f({x}) = {f(x)}")

    # проверка решения заведомо надежным решением
    def neg_x(x):
        return - f(x[0])

    real_solve_x, = scipy.optimize.brute(neg_x, [[start, end]])
```

```
    real_solve_y = f(real_solve_x)

    print(f"Реальное решение: f({real_solve_x}) = {real_solve_y}")

    deviation = abs(x - real_solve_x)

    print(f"Отклонение x: {deviation}")
    print(f"Отклонение y: {abs(f(x) - f(real_solve_x))}")
    print()

    if (is_draw):
        matplotlib.pyplot.show(block = True)

    return x, f(x), xs, step


# Вариант 4
# Функция Sin(2x)/x2
def f(x: float): return math.sin(2*x) / math.pow(x, 2)

# Диапазон решений - x[-20,-3.1] - длина 16.9, 16900 значений для точности 0,001
# следовательно, длина хромосомы достаточна - 15 бит
start = -20.
end = -3.1
bits = 15
#combinations_num = int(math.pow(2, bits))

#вероятность мутации
mutation_prob = 0.001

# вероятность потомства
crossing_prob = 0.5

# размер популяции
population_size = 100

# отрезок локализации решения
localization = 0.01

# максимальное кол-во шагов
max_steps = 100

# запуск алгоритма
genetic(
    f,
    start = start,
    end = end,
    chrom_len = bits,
    mutation_prob = mutation_prob,
    crossing_prob = crossing_prob,
```

```
    population_size = 100,
    localization = localization,
    max_steps = max_steps,
    is_draw = True)

exit()

print("популяция 1000 особей")
genetic(
    f,
    start = start,
    end = end,
    chrom_len = bits,
    mutation_prob = mutation_prob,
    crossing_prob = crossing_prob,
    population_size = 1000,
    localization = localization,
    max_steps = max_steps,
    is_draw = False)

print("пониженная вероятность кроссинговера = 0.2")
genetic(
    f,
    start = start,
    end = end,
    chrom_len = bits,
    mutation_prob = mutation_prob,
    crossing_prob = 0.2,
    population_size = population_size,
    localization = localization,
    max_steps = max_steps,
    is_draw = False)

print("повышенная вероятность мутации = 0.1")
genetic(
    f,
    start = start,
    end = end,
    chrom_len = bits,
    mutation_prob = 0.1,
    crossing_prob = crossing_prob,
    population_size = population_size,
    localization = localization,
    max_steps = max_steps,
    is_draw = False)

print("популяция 10 особей")
genetic(
    f,
    start = start,
```

```
    end = end,
    chrom_len = bits,
    mutation_prob = mutation_prob,
    crossing_prob = crossing_prob,
    population_size = 10,
    localization = localization,
    max_steps = max_steps,
    is_draw = False)
```

Условием остановки работы алгоритма является локализация фенотипов всех особей в ограниченной области. По умолчанию в текущей реализации – интервал должен быть менее 0.01 по шкале x.

Зависимость времени поиска, числа поколений (генераций), точности нахождения решения от изменения входных параметров алгоритма

Выполнены запуски программы с изменением параметров алгоритма.

1. Стандартные параметры

Работа генетического алгоритма завершена

Размер популяции 100

Вероятность кроссинговера 0.5

Вероятность мутации 0.001

Ограничивающая локализация решений: 0.01

Выполнено 8 шагов

Итоговая локализация решений - [-5.407968139648439; -5.406936645507814] длиной 0.0010314941406246447

Решение: f(-5.406936645507814) = 0.033642516018743227

Реальное решение: f(-5.406345600328949) = 0.033642540644839236

Отклонение x: 0.0005910451788651017

Отклонение y: 2.4626096009616738e-08

2. популяция 1000 особей

Работа генетического алгоритма завершена

Размер популяции 1000

Вероятность кроссинговера 0.5

Вероятность мутации 0.001

Ограничивающая локализация решений: 0.01

Выполнено 17 шагов

Итоговая локализация решений - [-5.406936645507814; -5.405905151367188] длиной 0.001031494140626421

Решение: f(-5.405905151367188) = 0.03364252790574235

Реальное решение: f(-5.406345600328949) = 0.033642540644839236

Отклонение x: 0.0004404489617613194

Отклонение y: 1.2739096888481072e-08

При увеличении размера популяции кол-во потенциально выполняемых шагов становится больше, но велика вероятность получить более приближенное к реальному решение.

3. пониженная вероятность кроссинговера = 0.2

Работа генетического алгоритма завершена

Размер популяции 100

Вероятность кроссинговера 0.2

Вероятность мутации 0.001

Ограничивающая локализация решений: 0.01

Выполнено 12 шагов

Итоговая локализация решений - [-5.430661010742188; -5.430661010742188] длиной 0.0

Решение: f(-5.430661010742188) = 0.033602300179304316

Реальное решение: f(-5.406345600328949) = 0.033642540644839236

Отклонение x: 0.024315410413239036

Отклонение y: 4.0240465534920056e-05

При уменьшении вероятности кроссинговера алгоритм выполняет больше шагов.

4. повышенная вероятность мутации = 0.1

Работа генетического алгоритма завершена

Размер популяции 100

Вероятность кроссинговера 0.5

Вероятность мутации 0.1

Ограничивающая локализация решений: 0.01

Выполнено 100 шагов

Итоговая локализация решений - [-17.031875610351562; -4.356875610351564] длиной 12.674999999999999

Решение: f(-4.356875610351564) = -0.034379913405785716

Реальное решение: f(-5.406345600328949) = 0.033642540644839236

Отклонение x: 1.0494699899773856

Отклонение y: 0.06802245405062496

Можно заключить, что при повышенной вероятности мутаций условие локализации становится труднодостижимо, и часто выполняется максимальное кол-во шагов.

5. популяция 10 особей

Работа генетического алгоритма завершена

Размер популяции 10

Вероятность кроссинговера 0.5

Вероятность мутации 0.001

Ограничивающая локализация решений: 0.01

Выполнено 4 шагов

Итоговая локализация решений - [-8.100167846679689; -8.100167846679689] длиной 0.0

Решение: f(-8.100167846679689) = 0.007204667583219708

Реальное решение: f(-5.406345600328949) = 0.033642540644839236

Отклонение x: 2.6938222463507397

Отклонение y: 0.02643787306161953

Можно заключить, что при малом количестве особей в популяции с учетом выбранного условия остановки работы алгоритма велика вероятность того, что в области реального решения не сгенерируются особи в начальной популяции. С учетом малой вероятности мутаций особи сгруппируются в имеющихся локальных максимумах, и при достижении необходимой локализации работа алгоритма прервется. Возможно этого избежать с помощью минимального необходимого количества популяций, повышенной вероятности мутаций, другого условия остановки.

Графики одного из решений с указанием максимума

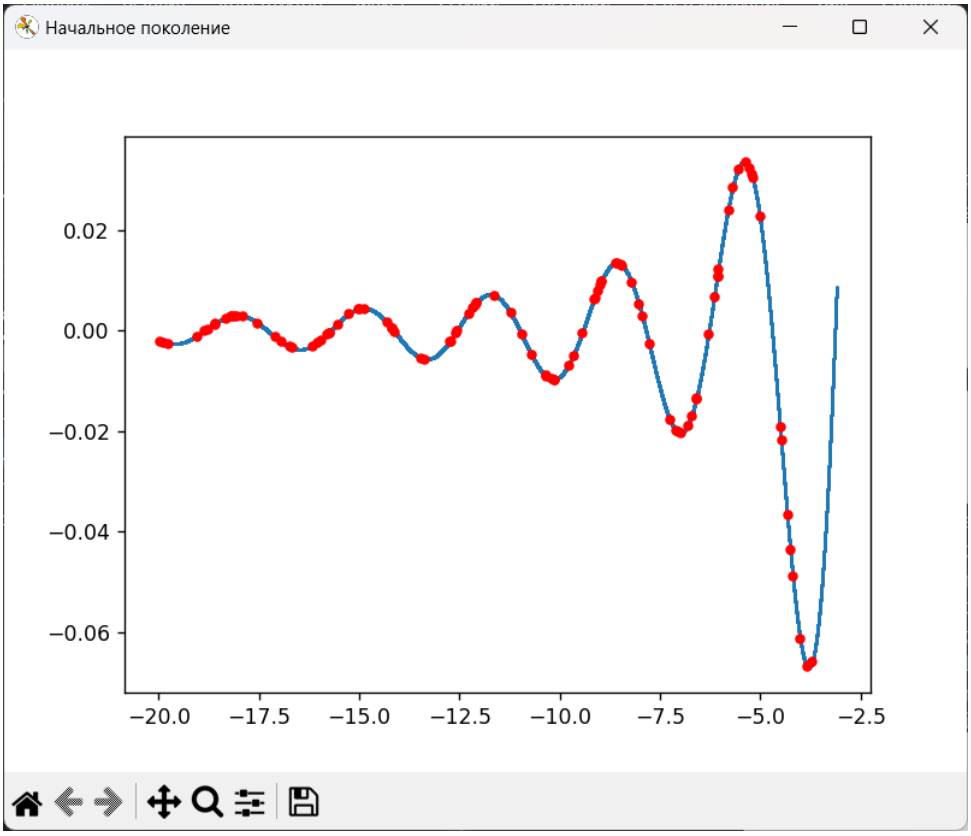

Рисунок 1 Начальное поколение

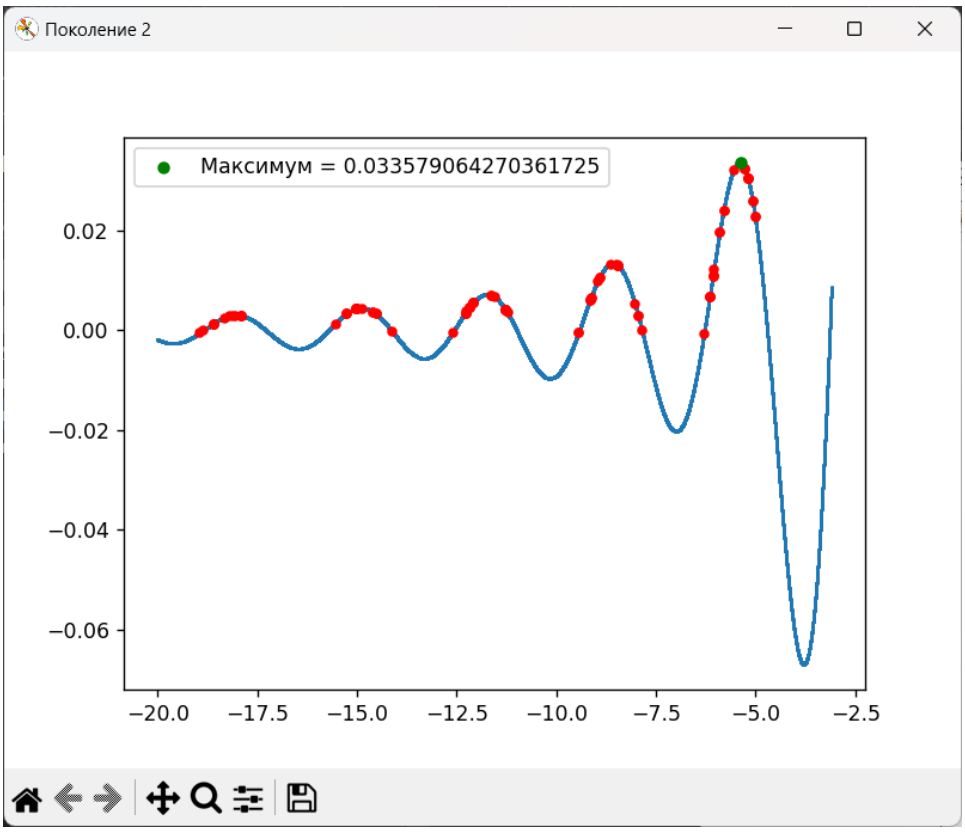

Рисунок 2 Поколение 2

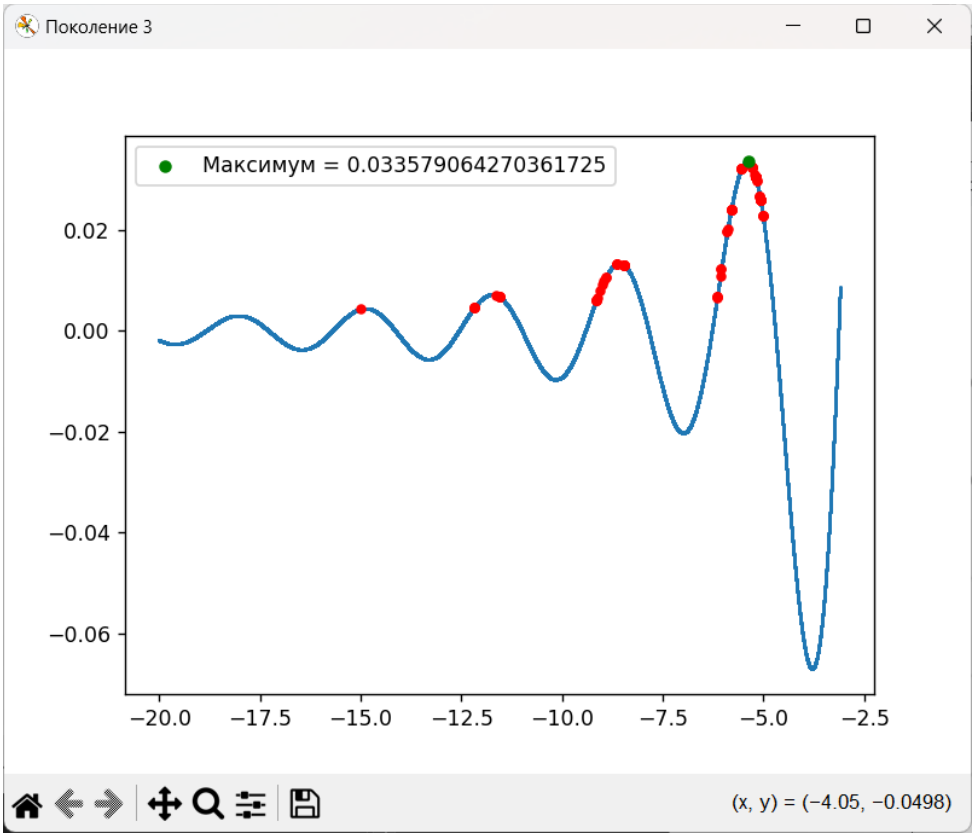

Рисунок 3 Поколение 3

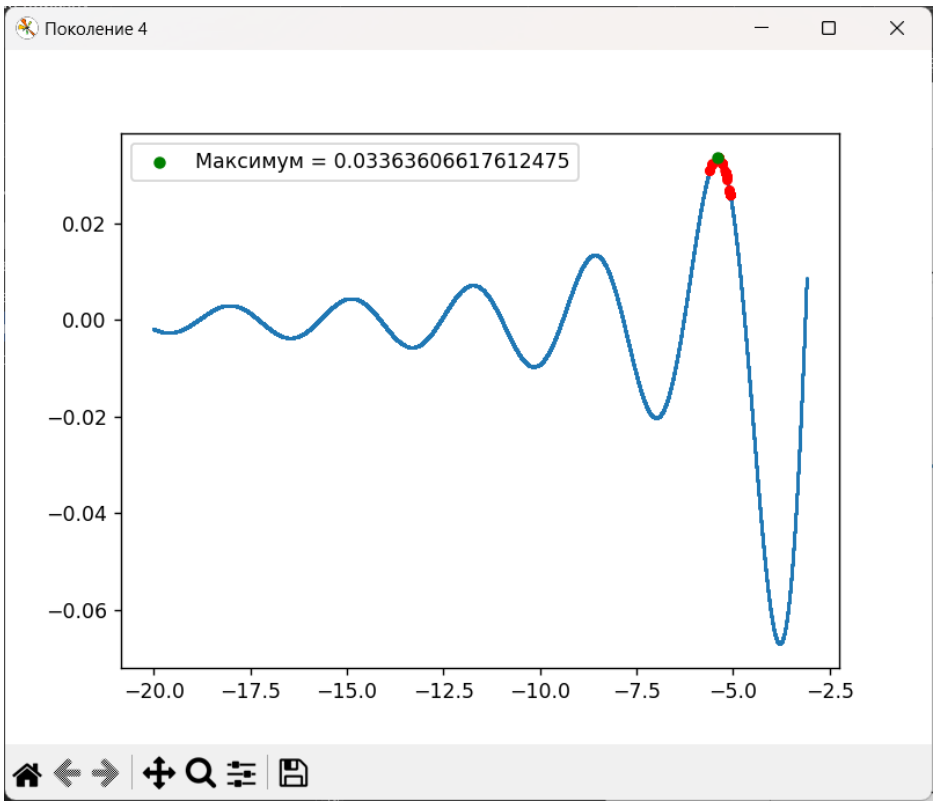

Рисунок 4 Поколение 4

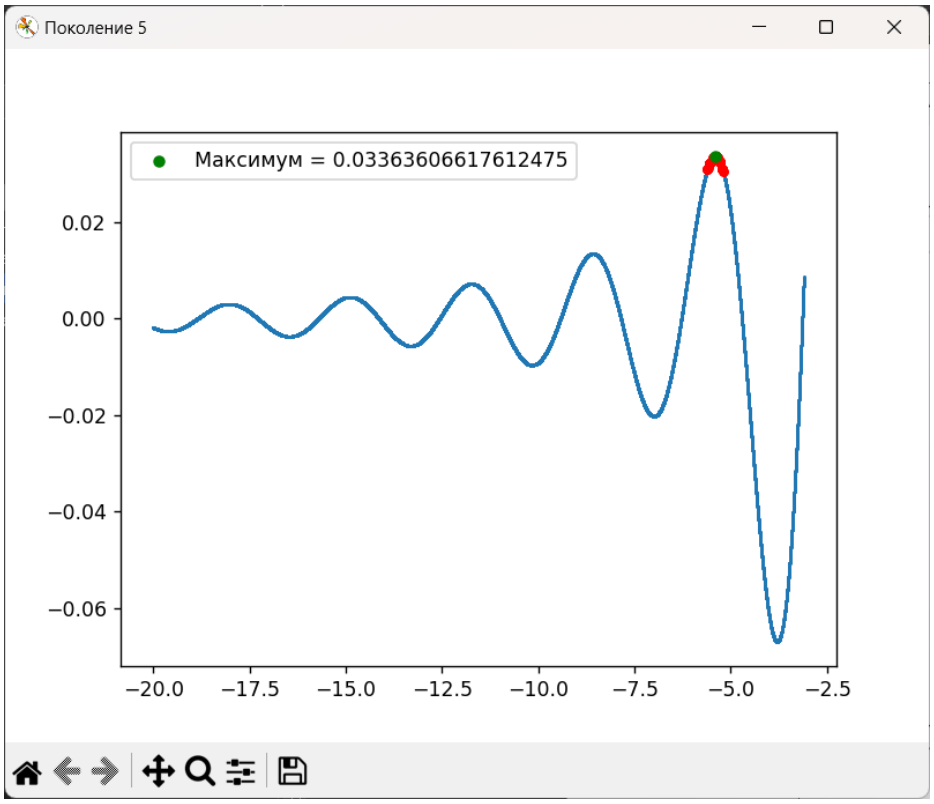

Рисунок 5 Поколение 5

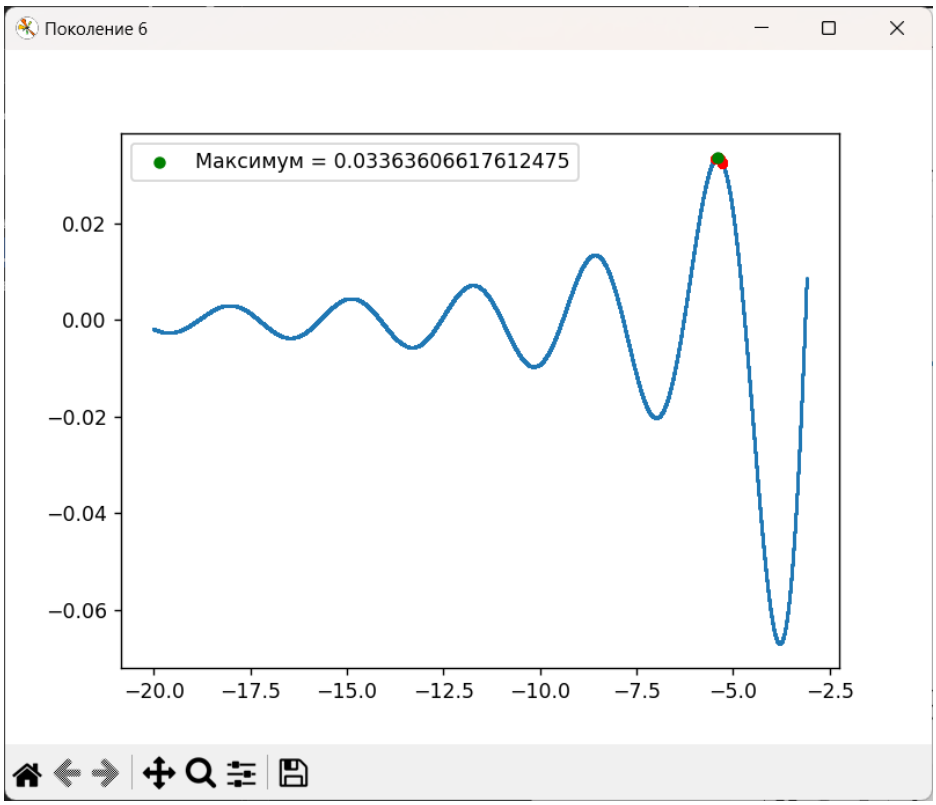

Рисунок 6 Поколение 6

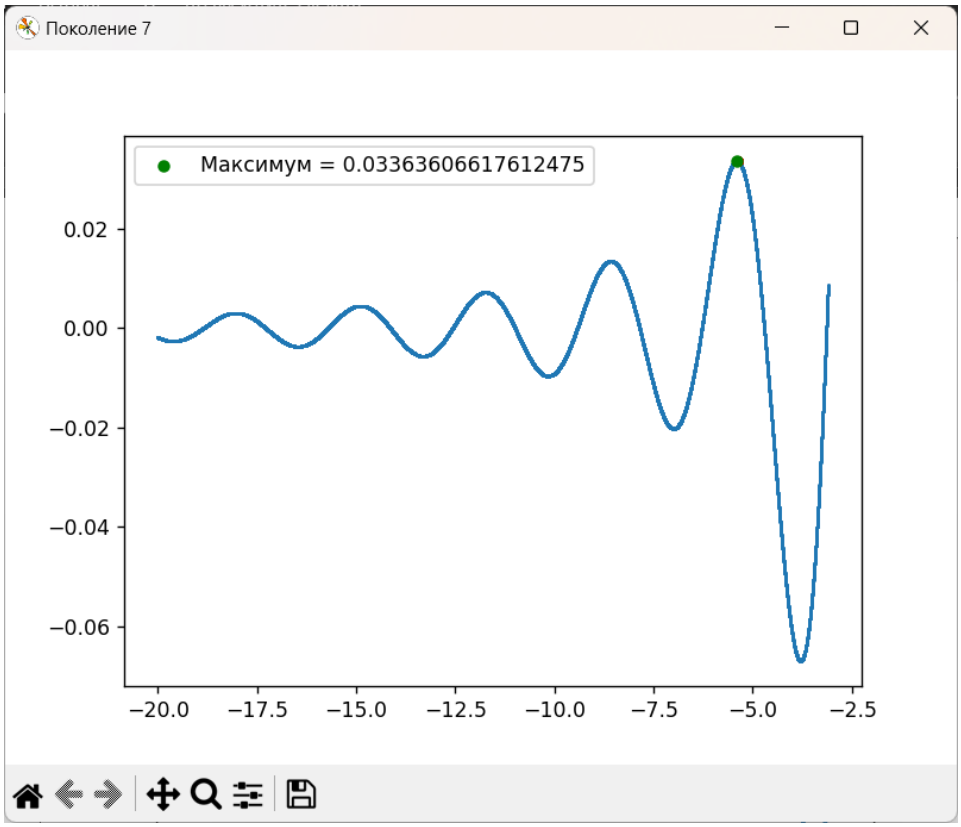

Рисунок 7 Поколение 7

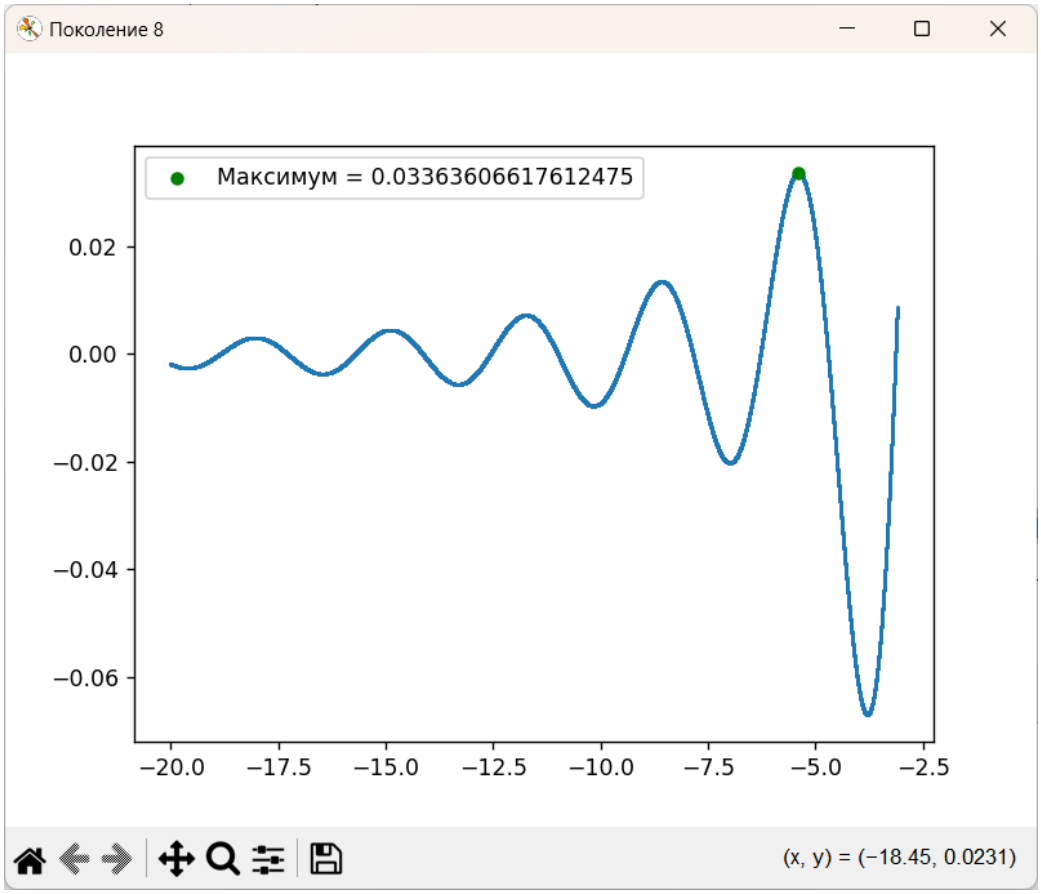

Рисунок 8 Поколение 8

Размер популяции 100

Вероятность кроссинговера 0.5

Вероятность мутации 0.001

Ограничивающая локализация решений: 0.01

Выполнено 8 шагов

Итоговая локализация решений - [-5.396621704101564; -5.396621704101564] длиной 0.0

Решение: f(-5.396621704101564) = 0.03363606617612475

Реальное решение: f(-5.406345600328949) = 0.033642540644839236

Отклонение x: 0.009723896227384898

Отклонение y: 6.474468714483261e-06

Ответ на контрольный вопрос

4. Опишите реализацию ОР (оператора репродукции) в виде колеса рулетки и приведите пример его работы.

```
def reproduction(population: list, f, start, end, num):

    """Оператор репродукции"""
    #todo
    intermediate_population = [];

    individ_values = []

    # сумма всех значений и значение для каждой особи
    for v in population:
        x = phenotype(v, start, end, num)
        y = f(x)
        individ_values += [y]

    # смещаем, чтобы не было отрицательных значений, считаем сумму
    min_poten = min(individ_values)
    offset = 0 - min_poten

    for i in range(len(individ_values)):
        individ_values[i] += offset

    values_sum = sum(individ_values)

    # все особи одинаковы
    if values_sum == 0:
        return population.copy()

    potentials = []

    # подсчет потенциала для каждой особи
    for i in range(len(individ_values)):
        val = individ_values[i]
        prob = val / values_sum
        potentials += [prob]

    sum_potentials = sum(potentials)

    # выбор такого же кол-ва особей
    for i in range(len(population)):

        # крутите барабан
        shot = random.random() * sum_potentials
        individ_num = 0
        tmp_sum = float()

        # определяем куда попали барабаном
```

```
        for j in range(len(population)):
            individ_num = j
            tmp_sum += potentials[individ_num]
            if (tmp_sum >= shot):
                break

        # добавляем выбранную особь в промежуточную популяцию
        individ = population[individ_num]
        intermediate_population += [individ]

    return intermediate_population
```

Каждой особи популяции с помощью фитнесс-функции в соответствие ставится определенное значение, характеризующее уровень его приспособления к условиям, описываемым фитнесс-функцией. В случае реализации ОР с помощью колеса рулетки, каждой особи сопоставляется сектор колеса, по длине доли окружности пропорциональный уровню приспособления особи. Затем колесо крутится, останавливаясь в случайном месте. Для репродукции выбирается особь, в чьем секторе была выполнена остановка.

Реализацию этого алгоритма на языке программирования можно выполнить с помощью генерации случайного числа от 0 до суммы всех показателей приспособлений, а затем итерирования по показателям всех особей с подсчетом промежуточной подсуммы. В момент, когда подсумма становится больше случайного числа, работа алгоритма прерывается, и текущая особь выбирается к репродукции.

Вывод

В ходе выполнения первой лабораторной работы была написана программа для поиска локального максимума функции одной переменной с помощью простого генетического алгоритма.

Выполнено сравнение найденного решения с реальным, изменения времени поиска и качества решения в зависимости от изменения размера популяции, разных вероятностей кроссинговера и мутации.

Показаны графики контрольного запуска программы.

Дан ответ на контрольный вопрос.