

ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

профессор

должность, уч. степень, звание

подпись, дата

Ю.А. Скобцов

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2

Оптимизация многомерных функций с помощью ГА

по дисциплине: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.

4134к

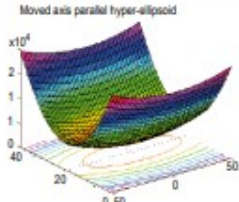
подпись, дата

Костяков Н.А.

инициалы, фамилия

Санкт-Петербург
2024

Цель работы: модификация представления хромосомы и операторов рекомбинации ГА для оптимизации многомерных функций. Графическое отображение результатов оптимизации

№ вв.	Название	Оптимум	Вид функции	График функции
4	Moved axis parallel hyper- ellipsoid function	global minimum $f(x)=0$; $x(i)=5 \cdot i$, $i=1:n$	$f_{1c}(x) = \sum_{i=1}^n 5i \cdot x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f1c(x)=\text{sum}(5*i \cdot x(i)^2),$ $i=1:n;$	

Задание:

1. Создать программу, использующую ГА для нахождения оптимума функции согласно таблице вариантов, приведенной в приложении А. Для всех Benchmark-ов оптимумом является минимум. Программу выполнить на встроенном языке пакета Matlab.
 2. Для $n=2$ вывести на экран график данной функции с указанием найденного экстремума, точек популяции. Для вывода графиков использовать стандартные возможности пакета Matlab. Предусмотреть возможность пошагового просмотра процесса поиска решения.

3. Повторить нахождение решения с использованием стандартного Genetic Algorithm toolbox. Сравнить полученные результаты.

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

- число особей в популяции
- вероятность кроссинговера, мутации.

Критерий остановки вычислений – повторение лучшего результата заданное количество раз или достижение популяцией определенного возраста (например, 100 эпох).

5. Повторить процесс поиска решения для $n=3$, сравнить результаты, скорость работы программы.

Выполнение:

1: Реализация генетического алгоритма для оптимизации функции

1. Представление хромосомы: для начала рассматривается одномерная функция, где каждая хромосома будет представлять значение переменной x из диапазона $[-10, 10]$.

2. Генерация начальной популяции: инициализируется популяция случайных значений x из указанного диапазона. Количество особей в популяции задается параметром N .

3. Фитнес-функция: для каждой хромосомы рассчитывается значение целевой функции $f(x)$, где функция оценивает минимизацию. Чем меньше значение функции, тем выше присваивается “фитнес”.

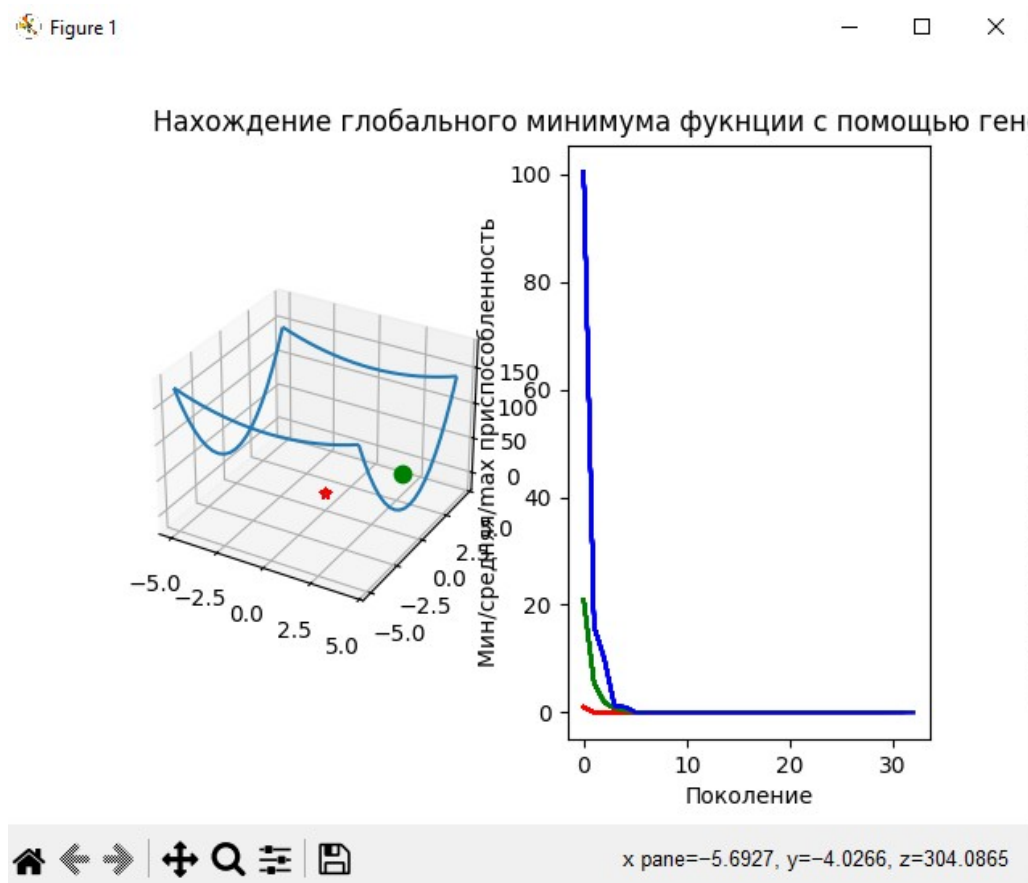
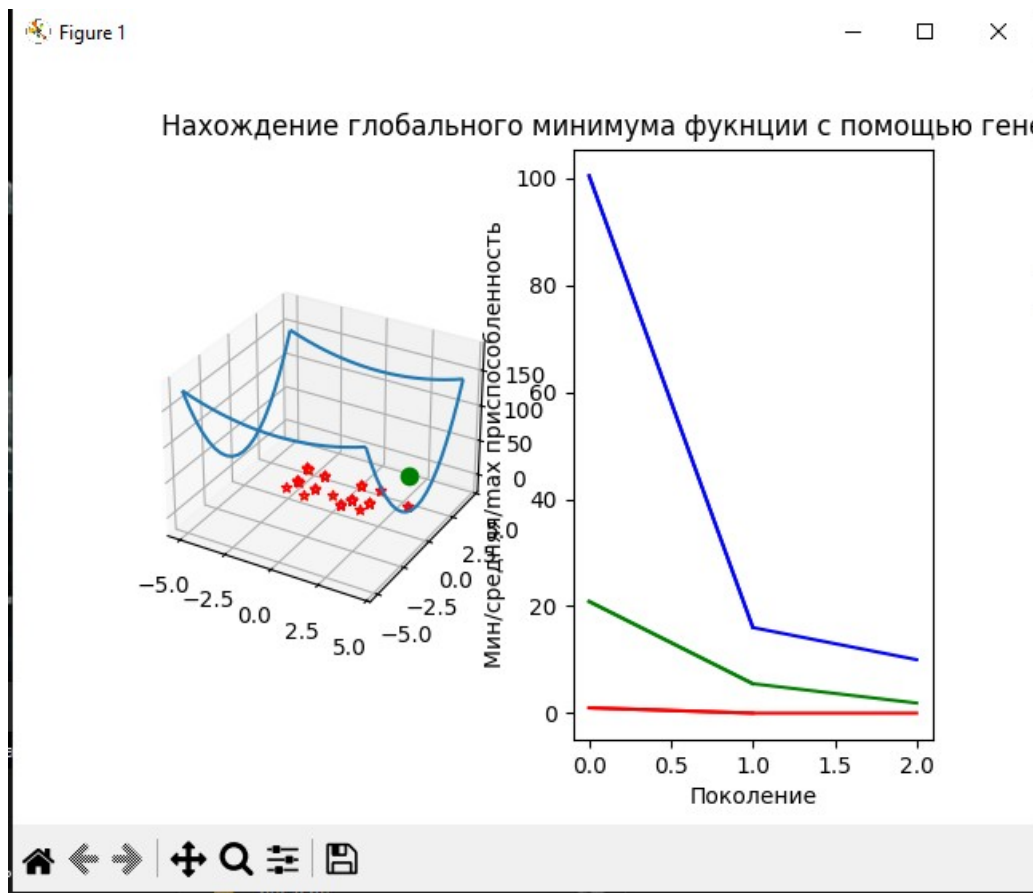
4. Операторы рекомбинации и мутации: используются классические операторы кроссинговера (двойной точечный кроссинговер) и мутации (случайное изменение значений хромосом).

5. Оператор отбора: применяется метод отбора по турниру, чтобы выбрать лучшие особи для создания следующего поколения.

3

6. Критерий остановки: Алгоритм прекращает выполнение, если на протяжении

заданного количества поколений не наблюдается улучшения результата.



```

Поколение 1: Функция приспособленности. = 1, Средняя приспособ.= 20.8693
Лучший индивидум = 1 0

Поколение 2: Функция приспособленности. = 0, Средняя приспособ.= 5.501
Лучший индивидум = 0 0

Поколение 3: Функция приспособленности. = 0, Средняя приспособ.= 1.88645000
Лучший индивидум = 0 0

Поколение 4: Функция приспособленности. = 0, Средняя приспособ.= 0.7402
Лучший индивидум = 0 0

Поколение 5: Функция приспособленности. = 0.0, Средняя приспособ.= 0.22
Лучший индивидум = 0 0.0

Поколение 6: Функция приспособленности. = 0.0, Средняя приспособ.= 0.0
Лучший индивидум = 0 0.0

```

1. Влияние числа особей в популяции

Увеличение размера популяции (числа особей) влечет за собой следующие изменения:

- **Время поиска:** Время выполнения алгоритма увеличивается с ростом числа особей в популяции, так как возрастает количество особей, которые необходимо оценивать и обрабатывать в каждой генерации.
- **Количество поколений:** Чаще всего, с увеличением числа особей, количество поколений, необходимое для достижения оптимального решения, также увеличивается. Это связано с большим числом возможных решений, что может потребовать больше времени для сходимости.
- **Точность нахождения решения:** Более крупные популяции могут обеспечить более разнообразные генетические материалы, что потенциально повышает вероятность нахождения более точного решения.

2. Влияние вероятности кроссинговера

Вероятность кроссинговера влияет на:

- **Время поиска:** С увеличением вероятности кроссинговера, время выполнения алгоритма может увеличиваться из-за большего числа операций кроссинговера, однако это может быть компенсировано более быстрым нахождением качественных решений.
- **Количество поколений:** Оптимальные значения вероятности кроссинговера могут способствовать сокращению числа поколений, необходимых для достижения хорошего решения, поскольку эффективно комбинируются сильные особи.
- **Точность нахождения решения:** Высокая вероятность кроссинговера обычно улучшает точность, так как усиливает обмен генетической информацией между особями.

3. Влияние вероятности мутации

Вероятность мутации оказывает влияние следующим образом:

- **Время поиска:** Более высокая вероятность мутации может увеличить время поиска, так как в каждом поколении будут происходить изменения в большем количестве особей.
- **Количество поколений:** Умеренная вероятность мутации может способствовать снижению числа поколений, так как она помогает избежать преждевременной сходимости к локальным минимумам, позволяя находить более качественные решения.
- **Точность нахождения решения:** Оптимальные значения вероятности мутации обеспечивают разнообразие популяции и помогают избежать застоя в поиске, что в свою очередь увеличивает шансы нахождения глобального минимума.

Критерий остановки

Критерий остановки вычислений в данном алгоритме реализован через два параметра:

1. **Максимальное количество поколений** (например, 100 эпох) — это основной параметр, который ограничивает время выполнения алгоритма и предотвращает бесконечный цикл.

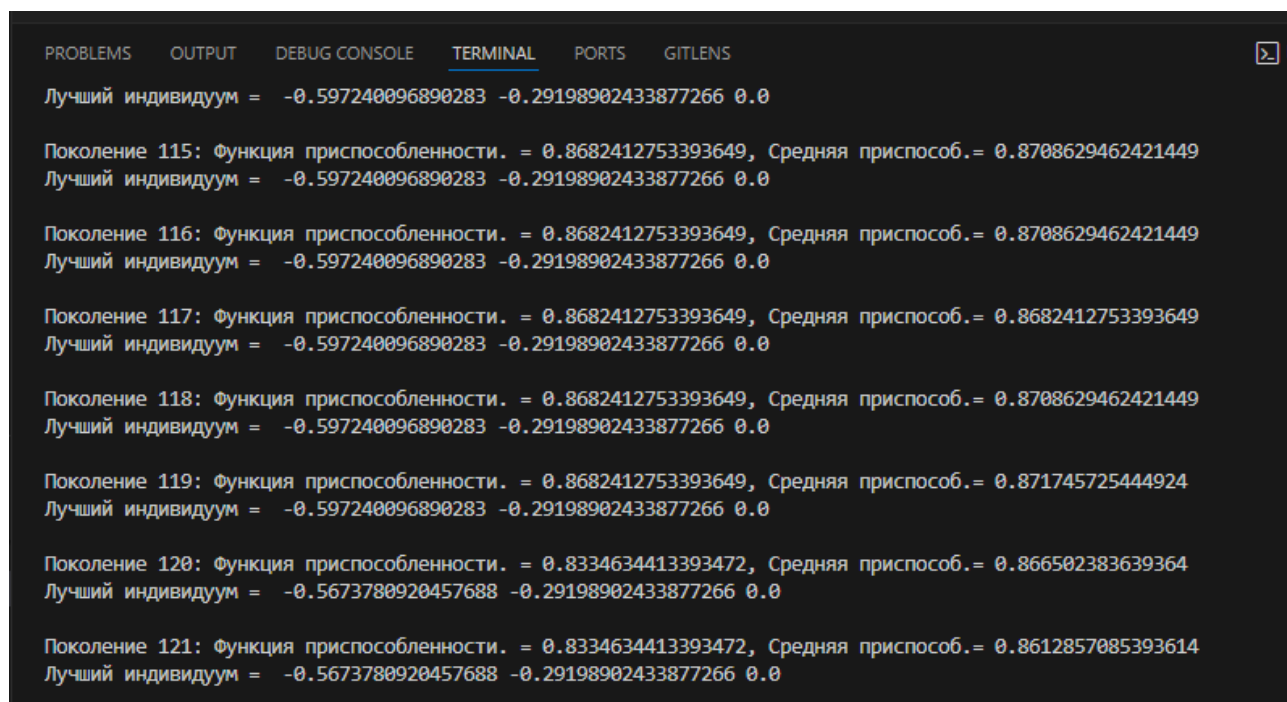
2. Количество поколений без улучшения (стагнация) — алгоритм останавливается, если не происходит улучшения решения за заданное число поколений. Это позволяет избежать ненужных вычислений, если алгоритм не показывает прогресса.

- Количество поколений: Умеренная вероятность мутации может способствовать снижению числа поколений, так как она помогает избежать преждевременной сходимости к локальным минимумам, позволяя находить более качественные решения.
- Точность нахождения решения: Оптимальные значения вероятности мутации обеспечивают разнообразие популяции и помогают избежать застоя в поиске, что в свою очередь увеличивает шансы нахождения глобального минимума.

Критерий остановки

Критерий остановки вычислений в данном алгоритме реализован через два параметра:

1. Максимальное количество поколений (например, 100 эпох) — это основной параметр, который ограничивает время выполнения алгоритма и предотвращает бесконечный цикл.
 2. Количество поколений без улучшения (стагнация) — алгоритм останавливается, если не происходит улучшения решения за заданное число поколений. Это позволяет избежать ненужных вычислений, если алгоритм не показывает прогресса.
 5. Повторить процесс поиска решения для $n=3$, сравнить результаты, скорость работы программы.
- Значения при $n=2$:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  [x] P

Лучший индивидум = -0.597240096890283 -0.29198902433877266 0.0

Поколение 115: Функция приспособленности. = 0.8682412753393649, Средняя приспособ.= 0.8708629462421449
Лучший индивидум = -0.597240096890283 -0.29198902433877266 0.0

Поколение 116: Функция приспособленности. = 0.8682412753393649, Средняя приспособ.= 0.8708629462421449
Лучший индивидум = -0.597240096890283 -0.29198902433877266 0.0

Поколение 117: Функция приспособленности. = 0.8682412753393649, Средняя приспособ.= 0.8682412753393649
Лучший индивидум = -0.597240096890283 -0.29198902433877266 0.0

Поколение 118: Функция приспособленности. = 0.8682412753393649, Средняя приспособ.= 0.8708629462421449
Лучший индивидум = -0.597240096890283 -0.29198902433877266 0.0

Поколение 119: Функция приспособленности. = 0.8682412753393649, Средняя приспособ.= 0.871745725444924
Лучший индивидум = -0.597240096890283 -0.29198902433877266 0.0

Поколение 120: Функция приспособленности. = 0.8334634413393472, Средняя приспособ.= 0.866502383639364
Лучший индивидум = -0.5673780920457688 -0.29198902433877266 0.0

Поколение 121: Функция приспособленности. = 0.8334634413393472, Средняя приспособ.= 0.8612857085393614
Лучший индивидум = -0.5673780920457688 -0.29198902433877266 0.0
```

Результаты схождения для трех генов

Сходимость по сравнению с 2 генами сильно дольше

Причины различий:

1. Размерность пространства:

- При увеличении размерности (с $n = 2$ до $n = 3$) сложность оптимизации возрастает, поскольку пространство решений становится больше.

2. Условия остановки:

- Остановка встроенного алгоритма может быть связана с его настройками (например, FunctionTolerance). Если изменение значений функции становится меньше заданного

порога, это может привести к преждевременной остановке.

3. Генерация решения:

- Различные методы отбора, кроссинговера и мутации могут влиять на скорость сходимости алгоритма.

4. Свойства функции:

- Если функция более “плоская” в одной размерности, это может сделать поиск более трудным. В этом случае более низкие значения могут потребовать больше итераций, что влияет на скорость и точность нахождения оптимума. Причины различий:

1. Размерность пространства:

- При увеличении размерности (с $n = 2$ до $n = 3$) сложность оптимизации возрастает, поскольку пространство решений становится больше.

2. Условия остановки:

- Остановка встроенного алгоритма может быть связана с его настройками (например, `FunctionTolerance`). Если изменение значений функции становится меньше заданного порога, это может привести к преждевременной остановке.

3. Генерация решения:

- Различные методы отбора, кроссинговера и мутации могут влиять на скорость сходимости алгоритма.

4. Свойства функции:

- Если функция более “плоская” в одной размерности, это может сделать поиск более трудным. В этом случае более низкие значения могут потребовать больше итераций, что влияет на скорость и точность нахождения оптимума.

Выводы:

В ходе выполнения задания была успешно разработана программа на языке python, использующая генетический алгоритм для нахождения оптимума заданной функции.

Листинг

```
import random
import time

import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.animation as animation

# константы генетического алгоритма
POPULATION_SIZE = 20 # количество индивидуумов в популяции
MAX_GENERATIONS = 5 # максимальное количество поколений

P_CROSSOVER = 0.9 # вероятность скрещивания
P_MUTATION = 0.1 # вероятность мутации индивидуума

N_VECTOR = 3 # количество генов в хромосоме

LIMIT_VALUE_TOP = 5
LIMIT_VALUE_DOWN = -5

RANDOM_SEED = 1
random.seed(RANDOM_SEED)
```

```
class Individual(list):
    def __init__(self, *args):
        super().__init__(*args)
        self.value = 0
```

```
def fitness_function(f):
    return sum((5*i + 1) * (f[i] ** 2) for i in range(len(f)))
```

```
def individualCreator():
    return Individual([random.randint(LIMIT_VALUE_DOWN, LIMIT_VALUE_TOP) for i in
range(N_VECTOR)])
```

```
def populationCreator(n=0):
    return list([individualCreator() for i in range(n)])
```

```
population = populationCreator(n=POPULATION_SIZE)
```

```
fitnessValues = list(map(fitness_function, population))
```

```
for individual, fitnessValue in zip(population, fitnessValues):
    individual.value = fitnessValue
```

```
MinFitnessValues = []
meanFitnessValues = []
BadFitnessValues = []
```

```
population.sort(key=lambda ind: ind.value)
print(str(ind) + ", " + str(ind.value) for ind in population)
```

```
def clone(value):
    ind = Individual(value[:])
    ind.value = value.value
    return ind
```

```
def selection(popula, n=POPULATION_SIZE):
    offspring = []
    for i in range(n):
        i1 = i2 = i3 = i4 = 0
        while i1 in [i2, i3, i4] or i2 in [i1, i3, i4] or i3 in [i1, i2, i4] or i4 in
[i1, i2, i3]:
            i1, i2, i3, i4 = random.randint(0, n - 1), random.randint(0, n - 1),
random.randint(0,
n - 1), random.randint(
```



```

    0, n - 1)

    offspring.append(
        min([popula[i1], popula[i2], popula[i3], popula[i4]], key=lambda ind:
ind.value))

    return offspring

def crossbreeding(object_1, object_2):
    s = random.randint(1, len(object_1) - 1)
    object_1[s:], object_2[s:] = object_2[s:], object_1[s:]

def mutation(mutant, indpb=0.04, percent=0.05):
    for index in range(len(mutant)):
        if random.random() < indpb:
            mutant[index] += random.randint(-1, 1) * percent * mutant[index]

generationCounter = 0

def animate(i):
    global generationCounter

    ax.clear()
    ax.plot_wireframe(X, Y, Z, rstride=30, cstride=30)
    ax.text(LIMIT_VALUE_DOWN, LIMIT_VALUE_DOWN, 40000, "Поколение:" +
str(generationCounter))

    ax.scatter(2, 4, -12, marker='o', edgecolors='green', linewidths=4)

    for individ in population:
        ax.scatter(individ[0], individ[1], individ.value, marker='*', edgecolors='red')

    if generationCounter == 1:
        time.sleep(10)

    generationCounter += 1
    offspring = selection(population)
    offspring = list(map(clone, offspring))

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

    for mutant in offspring:
        if random.random() < P_MUTATION:
            mutation(mutant, indpb=1.0 / N_VECTOR)

```



```

freshFitnessValues = list(map(fitness_function, offspring))

for individual, fitnessValue in zip(offspring, freshFitnessValues):
    individual.value = fitnessValue

population[:] = offspring

fitnessValues = [ind.value for ind in population]

minFitness = min(fitnessValues)
meanFitness = sum(fitnessValues) / len(population)
maxFitness = max(fitnessValues)
MinFitnessValues.append(minFitness)
meanFitnessValues.append(meanFitness)
BadFitnessValues.append(maxFitness)

plt.plot(MinFitnessValues[int(MAX_GENERATIONS * 0.04):], color='red')
plt.plot(meanFitnessValues[int(MAX_GENERATIONS * 0.04):], color='green')
plt.plot(BadFitnessValues[int(MAX_GENERATIONS * 0.04):], color='blue')
print(
    f"Поколение {generationCounter}: Функция приспособленности. = {minFitness},
    Средняя приспособ.= {meanFitness}")

best_index = fitnessValues.index(min(fitnessValues))
print("Лучший индивидум = ", *population[best_index], "\n")

fig = plt.figure()
ax = fig.add_subplot(1, 2, 1, projection='3d')
ax2 = fig.add_subplot(1, 2, 2)

X = np.arange(LIMIT_VALUE_DOWN, LIMIT_VALUE_TOP, 0.5)
Y = np.arange(LIMIT_VALUE_DOWN, LIMIT_VALUE_TOP, 0.5)
X, Y = np.meshgrid(X, Y)
Z = np.array([[fitness_function([x, y]) for x in X[0]] for y in Y[:, 0]])
ani = animation.FuncAnimation(fig, animate, interval=900)

plt.xlabel('Поколение')
plt.ylabel('Мин/средняя/маx приспособленность')

plt.title("Нахождение глобального минимума функции с помощью генетического
алгоритма")

mng = plt.get_current_fig_manager()

plt.show()

```