

ГУАП

КАФЕДРА № 43

ОТЧЕТ  
ЗАЩИЩЕН С ОЦЕНКОЙ  
ПРЕПОДАВАТЕЛЬ

д-р техн. наук, профессор  
\_\_\_\_\_  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

Скобцов Ю.А.  
\_\_\_\_\_  
инициалы, фамилия

## ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

### **Оптимизация многомерных функций с помощью эволюционной стратегии**

по курсу: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

4132

\_\_\_\_\_  
подпись, дата

Н.И. Карпов  
\_\_\_\_\_  
инициалы, фамилия

Санкт-Петербург 2024

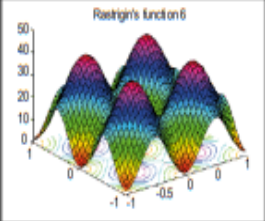
## СОДЕРЖАНИЕ

1	Индивидуальное задание .....	3
2	Краткие теоретические сведения .....	3
3	Результаты выполнения работы.....	5
3.1	Исследование решений при $n=2$ .....	5
3.2	Исследование решений при $n>2$ .....	6
3.3	Исследование решений при разных параметрах.....	7
3.4	Листинг программы .....	8
4	Вывод .....	12

## 1 Индивидуальное задание

Вариант 6:

Необходимо найти минимум многомерной функции с использованием эволюционной стратегии.

6	Rastrigin's function 6	global minimum $f(x)=0$ ; $x(i)=0$ , $i=1:n$ .	$f_6(x) = 10 \cdot n + \sum_{i=1}^{n-1} (x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i))$ $-5.12 \leq x_i \leq 5.12$ $f6(x) = 10 \cdot n + \text{sum}(x(i)^2 - 10 \cdot \cos(2 \cdot \pi \cdot x(i))),$ $i=1:n;$	
---	------------------------	------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------

## 2 Краткие теоретические сведения

Эволюционные стратегии (ЭС), также как и предыдущие парадигмы, основаны на эволюции популяции потенциальных решений, но, в отличие от них, здесь используется генетические операторы на уровне фенотипа, а не генотипа, как это делается в ГА. Разница в том, что ГА работают в пространстве генотипа – кодов решений, в то время как ЭС производят поиск в пространстве фенотипа – векторном пространстве вещественных чисел. В ЭС учитываются свойства хромосомы «в целом», в отличие от ГА, где при поиске решений исследуются отдельные гены. В природе один ген может одновременно влиять на несколько свойств организма. С другой стороны одно свойство особи может определяться несколькими генами. Естественная эволюция основана на исследовании совокупности генов, а не отдельного (изолированного) гена.

В эволюционных стратегиях целью является движение особей популяции по направлению к лучшей области ландшафта фитнес-функции. ЭС изначально разработаны для решения многомерных оптимизационных задач, где пространство поиска – многомерное пространство вещественных чисел. Иногда при решении задачи накладываются некоторые ограничения, например, вида  $g_i(x) > 0$ .

Ранние эволюционные стратегии (ЭС) основывались на популяции, состоящей из одной особи, и в них использовался только один генетический оператор – мутация. Здесь для представления особи (потенциального решения) была использована идея, не представленная в классическом генетическом алгоритме, которая заключается в следующем.

Здесь особь представляется парой действительных векторов

$$v = (\bar{x}, \bar{\sigma}),$$

где  $x$  - точка в пространстве решений и  $\sigma$  - вектор стандартных отклонений (вариабельность) от решения. В общем случае особь популяции определяется вектором потенциального решения и вектором «стратегических параметров» эволюции. Обычно это вектор стандартных отклонений (дисперсия), хотя допускаются (и иногда используются) и другие статистики.

Единственным генетическим оператором в классической ЭС является оператор мутации, который выполняется путем сложения координат вектора-родителя со случайными числами, подчиняющимися закону нормального распределения, следующим образом:

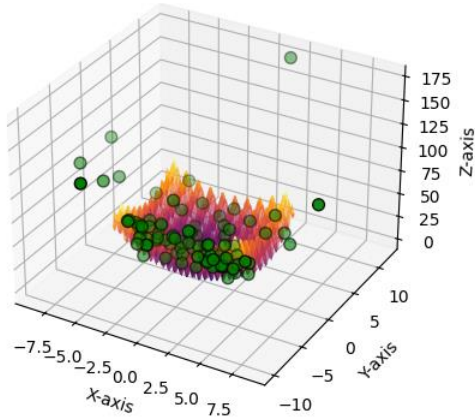
$$\bar{x}^{t+1} = \bar{x}^t + N(0, \bar{\sigma}),$$

где  $N(0, \sigma)$  - вектор независимых случайных чисел, генерируемых согласно распределению Гаусса (например, табличным способом) с нулевым средним значением и стандартным отклонением  $\sigma$ . Как видно из приведенной формулы величина мутации управляется нетрадиционным способом. Иногда эволюционный процесс используется для изменения и самих стратегических параметров  $\sigma$ , в этом случае величина мутации эволюционирует вместе с искомым потенциальным решением. Это соответствует адаптивному ГА с изменяемым шагом мутации

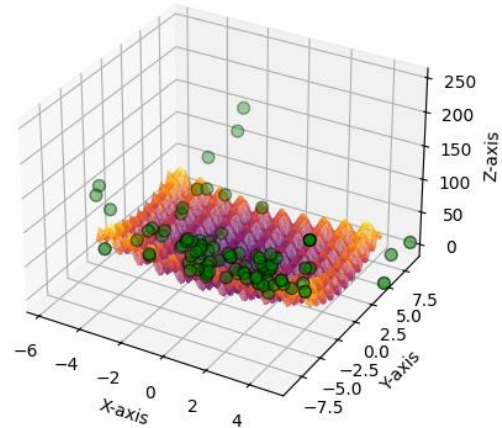
### 3 Результаты выполнения работы

#### 3.1 Исследование решений при $n=2$

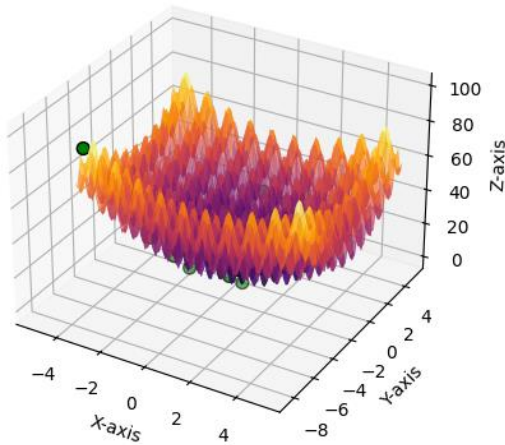
Generation 1. Population size is 100.  
Pc -> 0.9. Pm -> 0.1. Min fitness = 3.5165



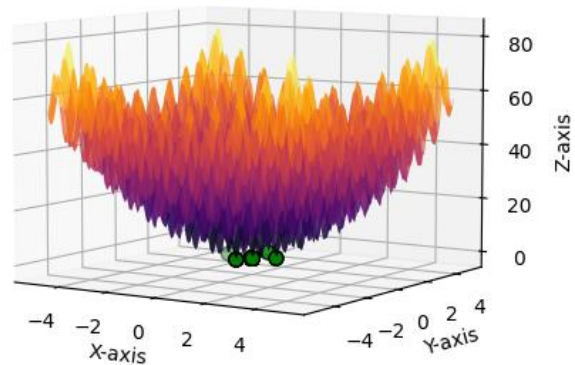
Generation 2. Population size is 100.  
Pc -> 0.9. Pm -> 0.1. Min fitness = 0.1387



Generation 3. Population size is 100.  
Pc -> 0.9. Pm -> 0.1. Min fitness = 0.0023



Generation 4. Population size is 100.  
Pc -> 0.9. Pm -> 0.1. Min fitness = 0.0020



Generation [48/50]      Fitness delta = -0.170      Min fitness = 0.026      Mean: 1.441

Generation [49/50]      Fitness delta = -0.134      Min fitness = 0.026      Mean: 1.307

=====  
Required time: 0.35s. Found answer: 0.026056. Required generations: 50. Fitness mean delta: 0.133800  
n = 2, Pc = 0.9, Pm = 0.1

Т.е. на поиск решения ушло 0,35 секунд и 50 поколений.

### 3.2 Исследование решений при $n > 2$

При  $n = 3$

Generation [48/50]	Fitness delta = -0.122	Min fitness = 1.088	Mean: 7.684
--------------------	------------------------	---------------------	-------------

Generation [49/50]	Fitness delta = -0.277	Min fitness = 1.088	Mean: 7.407
--------------------	------------------------	---------------------	-------------

```
=====
Required time: 0.26s. Found answer: 1.088250. Required generations: 50. Fitness mean delta: 0.277264
n = 3, Pc = 0.9, Pm = 0.1
```

Т.е. на поиск решения ушло 0,26 секунд и 50 поколений.

При  $n = 4$

Generation [48/50]	Fitness delta = -0.211	Min fitness = 2.577	Mean: 16.354
--------------------	------------------------	---------------------	--------------

Generation [49/50]	Fitness delta = -0.137	Min fitness = 2.577	Mean: 16.218
--------------------	------------------------	---------------------	--------------

```
=====
Required time: 0.27s. Found answer: 2.577491. Required generations: 50. Fitness mean delta: 0.136781
n = 4, Pc = 0.9, Pm = 0.1
```

Т.е. на поиск решения ушло 0,27 секунд и 50 поколений.

При  $n = 5$

Generation [48/50]	Fitness delta = -0.255	Min fitness = 5.539	Mean: 22.471
--------------------	------------------------	---------------------	--------------

Generation [49/50]	Fitness delta = -0.314	Min fitness = 5.539	Mean: 22.157
--------------------	------------------------	---------------------	--------------

```
=====
Required time: 0.29s. Found answer: 5.539440. Required generations: 50. Fitness mean delta: 0.314492
n = 5, Pc = 0.9, Pm = 0.1
```

Т.е. на поиск решения ушло 0,29 секунд и 50 поколений.

Точность решения понижается с увеличением количества измерений, скорость значений меняется незначительно. Для компенсации уменьшения точности можно увеличить вероятность мутации, количество поколений или размер популяции.

### 3.3 Исследование решений при разных параметрах

В ходе работы было установлено, что размер популяции положительно влияет на точность, но негативно влияет на скорость вычислений:

Размер популяции 1000:

Generation [48/50]	Fitness delta = -0.237	Min fitness = 2.734	Mean: 32.483
--------------------	------------------------	---------------------	--------------

Generation [49/50]	Fitness delta = -0.287	Min fitness = 2.734	Mean: 32.196
--------------------	------------------------	---------------------	--------------

=====  
Required time: 1.03s. Found answer: 2.734323. Required generations: 50. Fitness mean delta: 0.286779  
n = 3, Pc = 0.9, Pm = 0.1

Размер популяции 10:

Generation [48/50]	Fitness delta = -0.000	Min fitness = 4.985	Mean: 5.008
--------------------	------------------------	---------------------	-------------

Generation [49/50]	Fitness delta = 0.000	Min fitness = 4.985	Mean: 5.008
--------------------	-----------------------	---------------------	-------------

=====  
Required time: 0.19s. Found answer: 4.985424. Required generations: 50. Fitness mean delta: 0.000000  
n = 3, Pc = 0.9, Pm = 0.1

Большая вероятность кроссинговера или мутации уменьшает шанс попасть в локальный минимум, но ухудшает сходимость:

Generation [48/50]	Fitness delta = -0.240	Min fitness = 1.091	Mean: 6.196
--------------------	------------------------	---------------------	-------------

Generation [49/50]	Fitness delta = -0.085	Min fitness = 1.091	Mean: 6.111
--------------------	------------------------	---------------------	-------------

=====  
Required time: 0.26s. Found answer: 1.090679. Required generations: 50. Fitness mean delta: 0.085044  
n = 3, Pc = 0.9, Pm = 0.1

Generation [48/50]	Fitness delta = -0.393	Min fitness = 0.596	Mean: 3.824
--------------------	------------------------	---------------------	-------------

Generation [49/50]	Fitness delta = -0.274	Min fitness = 0.596	Mean: 3.550
--------------------	------------------------	---------------------	-------------

=====  
Required time: 0.27s. Found answer: 0.595723. Required generations: 50. Fitness mean delta: 0.273702  
n = 3, Pc = 0.5, Pm = 0.1

Generation [48/50]	Fitness delta = -0.637	Min fitness = 0.216	Mean: 5.253
Generation [49/50]	Fitness delta = -0.532	Min fitness = 0.216	Mean: 4.721

```
=====
Required time: 0.25s. Found answer: 0.216394. Required generations: 50. Fitness mean delta: 0.531912
n = 3, Pc = 0.5, Pm = 0.01
```

### 3.4 Листинг программы

```
import numpy as np
import genalg

POPULATION_SIZE = 100
OFFSPRING_SIZE = 20
CROSSING_OVER_PROBABILITY = 0.9
MUTATION_PROBABILITY = 0.1
MAX_GENERATIONS = 50
EPSILON = 0.
N = 5

LEFT_EDGE, RIGHT_EDGE = -5.12, 5.12

# Function is Rastrigin's function 6
def fitness_function(chromosome):
    return 10 * len(chromosome) + np.sum(np.power(chromosome, 2) - 10 *
np.cos(2 * np.pi * chromosome))

if __name__ == '__main__':
    genetic_optimizer = genalg.EvolutionaryStrategy(fitness_function, N,
POPULATION_SIZE, MAX_GENERATIONS,
                                                    OFFSPRING_SIZE, CROSS-
ING_OVER_PROBABILITY, MUTATION_PROBABILITY,
                                                    LEFT_EDGE, RIGHT_EDGE)

    genetic_optimizer.start()

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

from time import time
from random import randint, random

def count_mutation_delta(generation, total_generations, limit, b):
    # Random number [0, 1)
    r = random()

    degree = (1 - generation / total_generations) ** b
    return limit ** (1 - r ** degree)

class EvolutionaryStrategy:
    """
```



```

    Solves task of finding maximum of target function (fitness function)
    using genetic algorithm.
    :param fitness_function: target function
    :param n: number of dimensions
    :param population_size: size of the population
    :param max_generations: number of max available generations. For a sit-
    uation when it is impossible to find a satisfying optimum
    :param crossover_probability: probability of a crossover
    :param mutation_probability: probability of a mutation
    :param epsilon: stop algorithm when mean fitness function is less than
    or equal to this value
    """

```

```

    def __init__(self, fitness_function, n: int, population_size: int,
    max_generations: int, offspring_size,
    crossover_probability: float, mutation_probability: float,
    left_edge: float, right_edge: float,
    sigma=0.1, k_success=10):
        self.fitness_function = fitness_function
        self.population_size = population_size
        self.max_generations = max_generations
        self.offspring_size = offspring_size
        self.crossover_probability = crossover_probability
        self.mutation_probability = mutation_probability
        self.left_edge = left_edge
        self.right_edge = right_edge
        self.n = n

```

```

    # Every chromosome is vector with n elements (xi). Initially uni-
    formly randomized.

```

```

    self.population = np.random.uniform(left_edge, right_edge,
    size=(population_size, n))
    self.strategies = np.full((population_size, n), sigma)
    self.fitness_values = np.zeros(population_size, float)

```

```

    # Rule of success 1/5

```

```

    self.k_success = k_success
    self.successful_mutations = 0
    self.total_mutations = 0

```

```

    def start(self):
        generation = 1 # generations counter
        last_fitness_mean = 0. # mean fitness
        mean_fitness_delta = 1. # delta between current and previous mean
        fitness function value

```

```

        start_time = time()

```

```

        while generation < self.max_generations:
            # 1. Evaluate fitness.
            self.evaluate()

            # 2. Generate offspring
            offspring, offspring_strategies, success_flags =
            self.mutate_and_recombine()

            # 3. Evaluate offspring fitness
            offspring_fitness = np.array([self.fitness_function(ind) for
            ind in offspring])

            # 4. Update success statistics

```

```

        self.update_success_statistics(success_flags)

        # 5. Select the next generation ( $\mu + \lambda$  strategy)
        self.select(offspring, offspring_fitness, offspring_strategies)

        # 6. Adjust mutation strength based on the success rule
        if (generation + 1) % self.k_success == 0:
            self.adjust_sigma()

        mean_fitness_delta = self.fitness_values.mean() -
last_fitness_mean
        last_fitness_mean = self.fitness_values.mean()

        print(f'Generation [{generation}/{self.max_generations}]\t\t' +
              f'Fitness delta = {mean_fitness_delta:.3f}\t\tMin fitness
= {min(self.fitness_values):.3f}\t\t' +
              f'Mean: {(sum(self.fitness_values) /
self.population_size):.3f}\n')

        if self.n == 2:
            self.evaluate()
            # self.draw_plot(generation)

        generation += 1

    end_time = time()

    print('\n', '=' * 100)
    print(f'Required time: {end_time - start_time:.2f}s. Found answer:
{min(self.fitness_values):4f}. ',
          f'Required generations: {generation}. Fitness mean delta:
{abs(mean_fitness_delta):3f}\n',
          f'n = {self.n}, Pc = {self.crossover_probability}, Pm =
{self.mutation_probability}')

    def evaluate(self):
        """
        Evaluates the fitness of the population.
        """
        for i in range(self.population_size):
            # Count fitness of every chromosome in population
            self.fitness_values[i] =
self.fitness_function(self.population[i])

    def mutate_and_recombine(self):
        """Generate offspring using mutation and recombination."""
        offspring = []
        offspring_strategies = []
        success_flags = []

        for _ in range(self.offspring_size):
            # Select two parents randomly
            parents_idx = np.random.choice(self.population_size, 2, re-
place=False)
            parent1, parent2 = self.population[parents_idx]
            strategy1, strategy2 = self.strategies[parents_idx]

            # Recombine (arithmetic mean)
            child = 0.5 * (parent1 + parent2)
            child_strategy = 0.5 * (strategy1 + strategy2)

```

```

        # Mutate child
        mutation = np.random.normal(0, child_strategy, size=self.n)
        child += mutation

        # Ensure child stays within bounds
        child = np.clip(child, self.left_edge, self.right_edge)

        # Update mutation strategy
        child_strategy *= np.exp(np.random.normal(0, 0.2, size=self.n))

        # Evaluate success
        parent_fitness = min(self.fitness_function(parent1),
self.fitness_function(parent2))
        child_fitness = self.fitness_function(child)
        success_flags.append(child_fitness < parent_fitness) # Success
if fitness improved

        offspring.append(child)
        offspring_strategies.append(child_strategy)

    return np.array(offspring), np.array(offspring_strategies),
np.array(success_flags)

def update_success_statistics(self, success_flags):
    """Update statistics for success rule."""
    self.successful_mutations += np.sum(success_flags)
    self.total_mutations += len(success_flags)

def select(self, offspring, offspring_fitness, offspring_strategies):
    """Select the top individuals for the next generation."""
    combined_population = np.vstack((self.population, offspring))
    combined_fitness = np.hstack((self.fitness_values, off-
spring_fitness))
    combined_strategies = np.vstack((self.strategies, off-
spring_strategies))

    # Select top  $\mu$  individuals
    best_indices = np.argsort(combined_fitness)[:self.population_size]
    self.population = combined_population[best_indices]
    self.fitness_values = combined_fitness[best_indices]
    self.strategies = combined_strategies[best_indices]

def adjust_sigma(self):
    """Adjust mutation strength based on the success rule."""
    if self.total_mutations == 0:
        return

    success_rate = self.successful_mutations / self.total_mutations
    if success_rate > 0.2:
        self.strategies *= 1.22 # Increase mutation strength
    elif success_rate < 0.2:
        self.strategies *= 0.82 # Decrease mutation strength

    # Reset statistics
    self.successful_mutations = 0
    self.total_mutations = 0

def draw_plot(self, generation):
    dots_n = 100

    x = np.linspace(self.left_edge, self.right_edge, dots_n)

```

```

Z = np.zeros(shape=(dots_n, dots_n))

for i in range(dots_n):
    for j in range(dots_n):
        Z[i][j] = self.fitness_function(np.array([x[i], x[j]]))

# Plot the surface
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y = np.meshgrid(x, x)
ax.plot_surface(X, Y, Z, cmap='inferno', alpha=0.7)

x_dots = self.population[:, 0]
y_dots = self.population[:, 1]
z_dots = self.fitness_values

ax.scatter3D(x_dots, y_dots, z_dots, color='green', marker='o',
s=50, edgecolor='black')

ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_zlabel("Z-axis")
ax.set_title(f"Generation {generation}. Population size is
{self.population_size}.\n" +
            f"Pc -> {self.crossover_probability}. Pm ->
{self.mutation_probability}. " +
            f"Min fitness = {min(self.fitness_values):.4f}")

plt.show()

```

## 4 Вывод

В ходе работы была создана программа для решения оптимизационной задачи с использованием эволюционной стратегии. Полученное решение является весьма эффективным.