МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ | Факультет прикладной
математики и информатики

Кафедра прикладной математики

Практическое задание № 3

по дисциплине «Цифровые модели и оценивание параметров»

ПРОСТОЙ ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

Группа ПМ-05

Вариант 4

БОЛДЫРЕВ СЕРГЕЙ

ГРУШЕВ АНДРЕЙ

ПУЧКОВ ДМИТРИЙ

Преподаватели ВАГИН ДЕНИС ВЛАДИМИРОВИЧ

Новосибирск,2023

1 Задание

Задача Реализовать ПГА. Генотип состоит из N чисел. Фенотип состоит из M чисел. Числа фенотипа вычисляются регулярным образом из чисел генотипа. Числа генотипа и фенотипа являются вещественными значениями. Значение функционала – корень из суммы квадратов чисел фенотипа. Мутация меняет число не более чем на 100 % в плюс или минус. Кроссинговер выполнять только среди лучших особей.

2 Описание генетического алгоритма

Задача подразумевает использование простого генетического алгоритма с элитарной стратегией (только лучшие особи дают потомство).

В нашей реализации фенотипом является последовательность, получаемая из генотипа путем суммирования пар чисел из генотипа:

$$fen_i = gen_j + gen_{j+1}$$
$$i = \overline{1, M},\ j = \{1, 3, 5 \ldots N-1\}$$

где fen_i – компонента фенотипа, gen_j – компонента генотипа, N – размерность генотипа, M – размерность фенотипа.

Функционал, который необходимо минимизировать:

$$F = \sqrt{\sum_{i=1}^{M} fen_i^2}$$

В реализованном ПГА используется кроссинговер с одной точкой разрыва, в мутации случайным образом выбирается число из фенотипа и увеличивается или уменьшается на случайную величину не, по модулю не большую выбранного числа. Вероятность мутации была большая, так как в элитарных стратегиях мутации необходимы на случай, если эволюция пойдет не по нужному пути.

В ходе работы, опытным путем были выбраны следующие макропараметры:

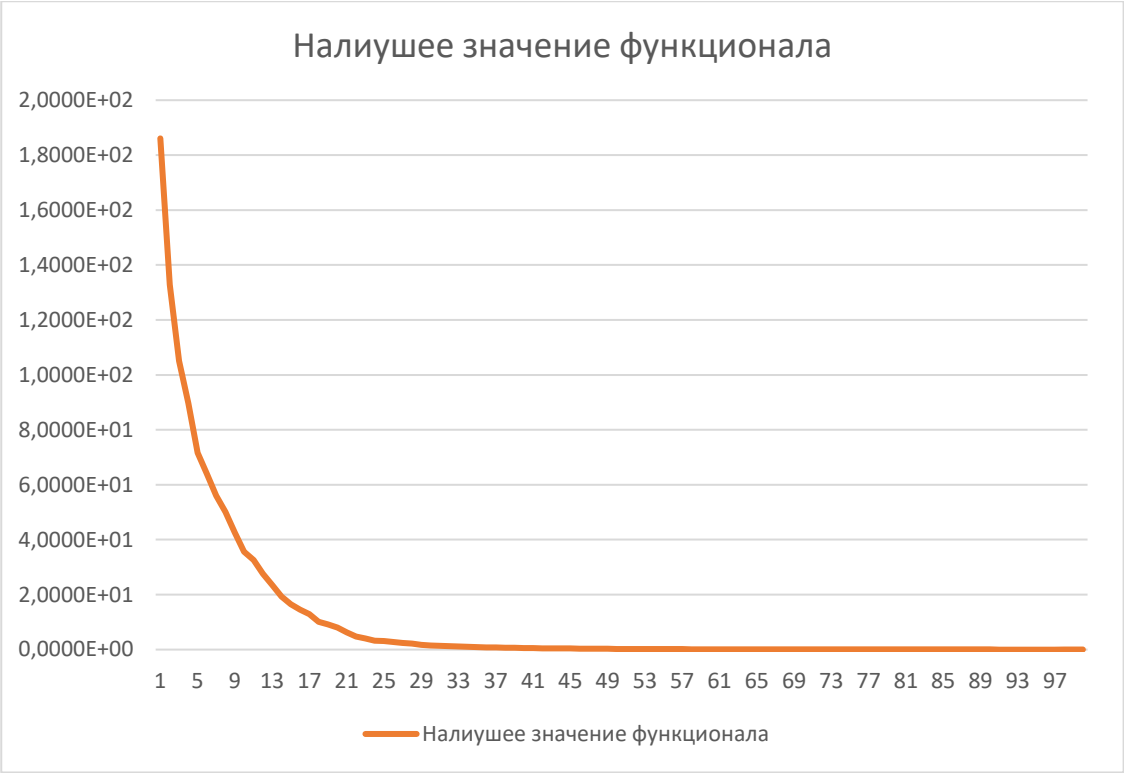
Размер популяции	18 000
Элита (особи, которые будут давать потомство)	50
Вероятность мутации	0.9

3 Результаты работы программы

iter	generation_best	Налиушее значение функционала
1	1,8610E+02	1,8610E+02
2	1,3284E+02	1,3284E+02
3	1,0509E+02	1,0509E+02
4	8,9682E+01	8,9682E+01
5	7,1679E+01	7,1679E+01
6	6,3941E+01	6,3941E+01
7	5,6009E+01	5,6009E+01
8	5,0051E+01	5,0051E+01
9	4,2661E+01	4,2661E+01
10	3,5662E+01	3,5662E+01
11	3,2555E+01	3,2555E+01
12	2,7636E+01	2,7636E+01
13	2,3458E+01	2,3458E+01
14	1,9354E+01	1,9354E+01
15	1,6505E+01	1,6505E+01
16	1,4509E+01	1,4509E+01
17	1,2912E+01	1,2912E+01
18	1,0111E+01	1,0111E+01
19	9,2297E+00	9,2297E+00
20	7,9800E+00	7,9800E+00
21	6,2255E+00	6,2255E+00
22	4,7399E+00	4,7399E+00
23	4,0813E+00	4,0813E+00
24	3,1879E+00	3,1879E+00
25	3,0587E+00	3,0587E+00
26	2,7163E+00	2,7163E+00
27	2,4098E+00	2,4098E+00
28	2,1467E+00	2,1467E+00
29	1,7513E+00	1,7513E+00
30	1,4410E+00	1,4410E+00
31	1,3400E+00	1,3400E+00

32	1,2091E+00	1,2091E+00
33	1,0967E+00	1,0967E+00
34	1,0152E+00	1,0152E+00
35	8,7251E-01	8,7251E-01
36	7,4574E-01	7,4574E-01
37	7,3592E-01	7,3592E-01
38	7,0737E-01	7,0737E-01
39	6,2431E-01	6,2431E-01
40	6,0533E-01	6,0533E-01
41	5,1882E-01	5,1882E-01
42	4,8043E-01	4,8043E-01
43	4,4384E-01	4,4384E-01
44	4,0851E-01	4,0851E-01
45	3,7934E-01	3,7934E-01
46	3,3700E-01	3,3700E-01
47	2,9028E-01	2,9028E-01
48	2,8163E-01	2,8163E-01
49	2,6272E-01	2,6272E-01
50	2,4383E-01	2,4383E-01
51	2,3939E-01	2,3939E-01
52	2,2238E-01	2,2238E-01
53	2,0604E-01	2,0604E-01
54	1,9783E-01	1,9783E-01
55	1,9128E-01	1,9128E-01
56	1,7523E-01	1,7523E-01
57	1,6119E-01	1,6119E-01
58	1,3986E-01	1,3986E-01
59	1,3193E-01	1,3193E-01
60	1,2849E-01	1,2849E-01
61	1,2377E-01	1,2377E-01
62	1,2061E-01	1,2061E-01
63	1,1106E-01	1,1106E-01
64	1,0337E-01	1,0337E-01
65	9,9385E-02	9,9385E-02
66	9,5400E-02	9,5400E-02

67	9,2446E-02	9,2446E-02
68	8,7539E-02	8,7539E-02
69	8,1113E-02	8,1113E-02
70	7,4425E-02	7,4425E-02
71	7,0284E-02	7,0284E-02
72	5,9862E-02	5,9862E-02
73	5,8295E-02	5,8295E-02
74	5,8098E-02	5,8098E-02
75	5,5610E-02	5,5610E-02
76	5,5351E-02	5,5351E-02
77	5,2308E-02	5,2308E-02
78	5,0548E-02	5,0548E-02
79	4,8172E-02	4,8172E-02
80	4,5733E-02	4,5733E-02
81	4,3329E-02	4,3329E-02
82	4,3325E-02	4,3325E-02
83	3,9145E-02	3,9145E-02
84	3,9132E-02	3,9132E-02
85	3,8074E-02	3,8074E-02
86	3,0541E-02	3,0541E-02
87	3,0541E-02	3,0541E-02
88	2,8819E-02	2,8819E-02
89	2,8595E-02	2,8595E-02
90	2,7895E-02	2,7895E-02
91	2,5323E-02	2,5323E-02
92	2,4530E-02	2,4530E-02
93	2,4370E-02	2,4370E-02
94	2,3349E-02	2,3349E-02
95	2,1200E-02	2,1200E-02
96	2,0015E-02	2,0015E-02
97	2,0015E-02	2,0015E-02
98	1,9330E-02	1,9330E-02
99	1,9329E-02	1,9329E-02
100	1,9329E-02	1,9329E-02



4 Вывод

Реализованный алгоритм сократил значение функционала на ~4 порядка. Таким образом была подтверждена применимость генетических алгоритмов для поиска минимума функционала. Однако генетические алгоритмы будут эффективны, когда нет возможности использовать классические методы оптимизации.

5 Код программы

```
import random
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from dataclasses import dataclass
import itertools
import pandas as pd
from tqdm import tqdm


N = 50
M = 25
ELITE = 50
POPULATION = 18000
MAXITER = 100
MUTATION_PROB = 0.9
FUNCTIONAL_MIN = 1e-5


def loss(fen):
    return np.sqrt(np.sum(np.power(np.array(fen), 2)))


class Individual:

    def __init__(self, gen):
        self.gen = list(gen)
        self.fen = self.calc_fen()
        self.fitness = loss(self.fen)

    def calc_fen(self):
        i = 0
```

```
        fen = []
        while i < len(self.gen):
            fen.append(self.gen[i]+self.gen[i+1])
            i += 2
        return fen


def create_child(p1, p2, mutation_prob):
    gen = krossingover(p1, p2)
    if np.random.random() < mutation_prob:
        gen = mutation(gen)
    return Individual(gen)


def krossingover(p1, p2):
    kross_idx = random.randint(1, N) # чтобы хотя бы 1 ген менялся
    return p1.gen[:kross_idx] + p2.gen[kross_idx:]


def mutation(gen):
    idx = np.random.randint(0, N-1)
    max_delta = abs(gen[idx])
    # [-max_delta; max_delta]
    mutation_value = 2 * max_delta * np.random.random_sample() - max_delta
    gen[idx] += mutation_value
    return gen


def create_population(parents, population_size, elite_size, mutation_prob):
    children = []
    for p1 in parents[:elite_size]:
        p2 = parents[np.random.randint(0, elite_size)]
        for _ in range(population_size//elite_size):
            children.append(create_child(p1, p2, mutation_prob))
    return children


def create_random_population(population_size):
    return [Individual(gen) for gen in 200 * np.random.random_sample((1000, N)) - 100]


def sort_population(population):
    return sorted(population, key=lambda x: x.fitness)


def get_best_individuals(population, elite_size):
    return sort_population(population)[:elite_size]


def sga(population_size, elite_size, mutation_prob):
    np.random.seed(12)
    population = create_random_population(population_size)
    best_individ = sort_population(population)[0]

    train_data = []
    for it in tqdm(range(MAXITER)):
        parents = get_best_individuals(population, elite_size)
        population = create_population(parents, population_size, elite_size, mutation_prob)

        generation_best = sort_population(population)[0]
        if best_individ.fitness > generation_best.fitness:
            best_individ = generation_best

        train_data.append((it+1, generation_best.fitness, best_individ.fitness))

        if best_individ.fitness < FUNCTIONAL_MIN:
            break
    return pd.DataFrame(train_data, columns=['iter', 'generation_best', 'best'])


if __name__ == '__main__':
    df = sga(POPULATION, ELITE, MUTATION_PROB)
    df.to_excel('./sga.xlsx', index=False)
```