Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 3

по курсу «Теория искусственных нейронных сетей»

«Методы многомерного поиска»

Студент группы ИУ9-72Б Терентьева А. С.

Преподаватель Каганов Ю. Т.

Москва 2023

1 Цель

1. Изучение алгоритмов многомерного поиска 1-го и 2-го порядка.
2. Разработка программ реализации алгоритмов многомерного поиска 1-го и 2-го порядка.
3. Вычисление экстремумов функции.

2 Задание

Требуется найти минимум тестовой функции Розенброка

1. Методами сопряженных градиентов (методом Флетчера-Ривза и методом Полака-Рибьера).
2. Квазиньютоновским методом (Девидона-Флетчера-Пауэлла).
3. Методом Левенберга-Марквардта.
4. Применяя генетичекий алгоритм.

3 Реализация

Исходный код программы представлен в листингах 1– 6.

Листинг 1: Метод Флетчера-Ривза

```
def fletcher_reeves(func, grad_f, x0):
    alpha = 0.01
    max_iters = 1000
    eps1, eps2 = 1e-6, 1e-16
    prev_x = x0.copy()
    x = x0
    prev_grad = []
    d = -grad_f(x)
    iter = 1
    second_time = False

    for i in range(max_iters):
        grad = grad_f(x)
        alpha = golden_section_search(
```

```
            lambda lr: func(x - lr * grad), 1e-6, 1e-3)
        if len(prev_grad) != 0:
            beta = np.dot(grad, grad) / np.dot(prev_grad, prev_grad)
            d = -grad + beta * d
        prev_x = x.copy()
        prev_grad = grad.copy()
        x += alpha * d

        if np.linalg.norm(x - prev_x) < eps1 and abs(func(x) - func(prev_x)) < eps2:
            #
            if second_time:
                break
            else:
                second_time = True
        else:
            second_time = False
        iter += 1
        xs.append(i)
        ys.append(F(x))

    print("        -                          :", iter)
    return x
```

Листинг 2: Метод Полака-Рибьера

```
def polak_ribiere(func, grad_f, x0):
    alpha = 0.01
    max_iters = 10000
    eps1, eps2, eps3 = 1e-6, 1e-16, 1e-16
    prev_x = x0.copy()
    x = x0
    prev_grad = []
    d = -grad_f(x)
    iter = 1
    n = 5
    second_time = False

    for i in range(max_iters):
        grad = grad_f(x)
        #                                        n-

        if i % n != 0:
            beta = np.dot(grad.T, grad - prev_grad) / np.dot(prev_grad, prev_grad)
            d = -grad + beta * d
        prev_x = x.copy()
```

```
        prev_grad = grad.copy()
        if i % n != 0:
            alpha = golden_section_search(
                lambda lr: func(x + lr * d), 0, 1e-3)
            x += alpha * d
        else:
            alpha = golden_section_search(
                lambda lr: func(x - lr * grad), 0, 1e-3)
            x -= alpha * grad

        iter += 1

        if np.linalg.norm(x - prev_x) < eps1 and abs(func(x) - func(prev_x)) < eps2 or np.linalg.norm(grad) < eps3:
            #
            if second_time:
                break
            else:
                second_time = True
        else:
            second_time = False
        xs.append(i)
        ys.append(F(x))

    print(" - :", iter)
    return x
```

Листинг 3: Метод Девидона-Флетчера-Пауэлла

```
def dfp_method(func, grad_func, x0):
    n = len(x0)
    H = np.eye(n)
    alpha = 0.01
    max_iters = 10000
    eps1, eps2 = 1e-6, 1e-16
    prev_grad = []
    x = x0
    iter = 1

    for i in range(max_iters):
        grad = grad_func(x)
        prev_grad = grad.copy()
        prev_x = x.copy()

        alpha = golden_section_search(
            lambda lr: func(x - lr * grad), 1e-6, 1e-1)

```

```
        p = -np.dot(H, grad)
        s = alpha * p  # dx
        x += s
        grad = grad_func(x)
        y = grad - prev_grad  # dg
        s = s.reshape(-1, 1)
        y = y.reshape(-1, 1)

        A = np.dot(s, s.T) / np.dot(s.T, y)
        B = np.dot(np.dot(np.dot(H, y), y.T), H.T) / np.dot(np.dot(y.T, H), y)
        H += A - B

        if np.linalg.norm(s) < eps1 and abs(func(x) - func(prev_x)) < eps2:
            #
            if second_time:
                break
            else:
                second_time = True
        else:
            second_time = False
        iter += 1
        xs.append(i)
        ys.append(F(x))

    print("     -                    :", iter)
    return x
```

Листинг 4: Метод Левенберга-Марквардта

```
def jacobian(x):
    return np.array([[2*x[0], 0], [0, 2*x[1]]])

def levenberg_marquardt(func, gradient, x0, lamda=1):
    n = len(x0)
    max_iters = 10000
    eps1, eps2 = 1e-6, 1e-16
    alpha = 1
    x = x0
    iter = 1

    for i in range(max_iters):
        grad = gradient(x)
        jac = jacobian(x)
        hessian = np.dot(jac.T, jac) + alpha * np.eye(n)
        step = np.linalg.solve(hessian, -grad)
```

```
        new_x = x + step
        if np.linalg.norm(step) < eps1 and abs(func(x) - func(new_x)) < eps2:
            break

        if func(new_x) < func(x):
            alpha /= 2
            x = new_x
        else:
            alpha *= 2
        iter += 1
        xs.append(i)
        ys.append(F(x))

    print("      -                    :", iter)
    return x
```

Листинг 5: Программа

```
import numpy as np
import time
import matplotlib.pyplot as plt

def F(x):
    a, b, f0 = 180, 2, 15
    return sum(a*(x[i]**2 - x[i+1])**2 + b*(x[i]-1)**2 for i in range(len(x)-1)) + f0

def dF(x):
    a, b = 180, 2
    return np.array([a*2*x[0]*(x[0]**2 - x[1]) + b*2*(x[0]-1), -a*2*(x[0]**2 - x[1])])

#
def golden_section_search(f, a, b, tol=1e-5):
    gr = (5 ** 0.5 - 1) / 2
    x1 = b - (b - a) * gr
    x2 = a + (b - a) * gr
    while abs(x1 - x2) > tol:
        if f(x1) < f(x2):
            b = x2
        else:
            a = x1
        x1 = b - (b - a) * gr
        x2 = a + (b - a) * gr
    return (b + a) / 2

```

```
xs, ys = [], []

methods = [{'name':"                                        ", 'func': gradient_descent, 'x0': np.array([0.0, 0.0])},
           {'name':"                    -           ", 'func': fletcher_reeves, 'x0': np.array([2.0, 0.0])},
           {'name':"                -              ", 'func': polak_ribiere, 'x0': np.array([2.0, 0.0])},
           {'name':"BFGS", 'func': bfgs_method, 'x0': np.array([0.0, 0.0])},
           {'name':"DFP", 'func': dfp_method, 'x0': np.array([0.0, 0.0])},
           {'name':"                              -          ", 'func': levenberg_marquardt, 'x0': np.array([0.0, 0.0])}
           ]

for method in methods:
    xs, ys = [], []
    start_time = time.time()
    print(f"\n{method['name']}:")
    result = method['func'](F, dF, method['x0'])
    print("                              :", time.time() - start_time, "c")
    print("                                          :", result)
    print("                              :", F(result))
    plt.plot(xs[:200], ys[:200], label=method['name'])

plt.legend()
plt.show()
```

Листинг 6: Генетический алгоритм

```
import numpy as np
import time
import random
import matplotlib.pyplot as plt

def F(x):
    a, b, f0 = 180, 2, 15
    return a*(x[0]**2 - x[1])**2 + b*(x[0]-1)**2 + f0

#
def fitness_function(x):
    return 1 / F(x)

#
```

```
def selection(population, fitness_func, retain_ratio=0.5):
    fitness_scores = {tuple(ind): fitness_func(ind) for ind in population}
    sorted_population = [list(ind) for ind in sorted(fitness_scores, key=fitness_scores.get, reverse=True)]
    retain_length = int(len(sorted_population) * retain_ratio)
    retain_length = 2 if retain_length < 2 else retain_length
    parents = sorted_population[:retain_length]
    return parents

#
def crossover(parents):
    children = []
    while len(children) < len(parents):
        father = random.randint(0, len(parents)-1)
        mother = random.randint(0, len(parents)-1)
        if father != mother:
            parent1 = np.array(parents[father])
            parent2 = np.array(parents[mother])
            c = np.random.rand()
            child1 = c * parent1 + (1 - c) * parent2
            child2 = (1 - c) * parent1 + c * parent2
            children.extend([child1, child2])
    return children

#
def mutation(children, mutation_chance=0.2):
    for child in children:
        if random.random() < mutation_chance:
            child[random.randint(0, len(child)-1)] += np.random.uniform(-0.5, 0.5)
    return children

xs = []
ys = []

#
def genetic_algorithm(population_size, dimension, generations, mutation_rate, crossover_rate):
    population = np.random.uniform(low=-5, high=5, size=(population_size, dimension))
    for i in range(generations):
        parents = selection(population, fitness_function, crossover_rate)
        offspring = crossover(parents)
        offspring = mutation(offspring, mutation_rate)
```

```
        population = parents + offspring
        result = population[np.argmax([fitness_function(x) for x in population])]
        xs.append(i)
        ys.append(F(result))

    best_solution = population[np.argmax([fitness_function(x) for x in population])]
    return best_solution

start_time = time.time()
print("                                        :")
result = genetic_algorithm(population_size=60, dimension=2, generations=50, mutation_rate=0.15, crossover_rate=0.5)
plt.plot(xs, ys)
plt.show()
print("                              :", time.time() - start_time, "c")
print("                                         :", result)
print("                            :", F(result))
```

4 Результат работы

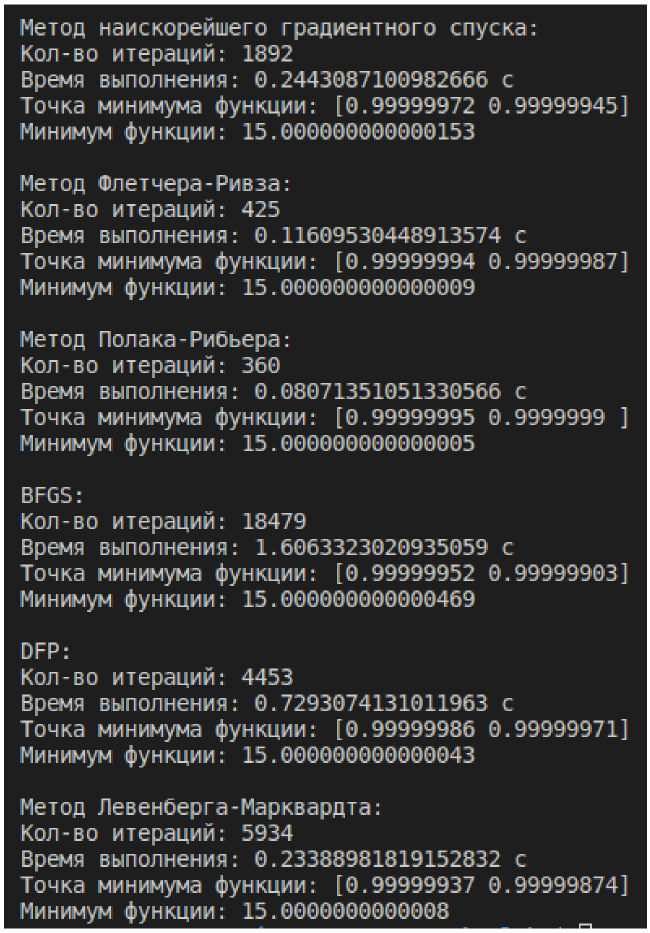
5 Выводы

В ходе выполнения лабораторной работы были изучены алгориты многомерного поиска 1-го и 2-го порядка, была написана их реализация на языке программирования Python.

В ходе эксперимента по исследованию работы программы на основе различных методов поиска экстремума (методы сопряженных градиентов, квазиньютоновский метод, методом Левенберга-Марквардта, генетический алгоритм), были сделаны следующие выводы:

1. Быстрее всего сходились методы сопряженных градиентов, наилучший результат показал метод Полака-Рибьера: и по скорости, и по количеству итераций, и по точности вычислений.

2. Самым медленным методом оказался метод BFGS, после него идет метод Левенберга-Марквардта.



```
Метод наискорейшего градиентного спуска:
Кол-во итераций: 1892
Время выполнения: 0.2443087100982666 с
Точка минимума функции: [0.99999972 0.99999945]
Минимум функции: 15.000000000000153

Метод Флетчера-Ривза:
Кол-во итераций: 425
Время выполнения: 0.11609530448913574 с
Точка минимума функции: [0.99999994 0.99999987]
Минимум функции: 15.000000000000009

Метод Полака-Рибьера:
Кол-во итераций: 360
Время выполнения: 0.08071351051330566 с
Точка минимума функции: [0.99999995 0.9999999 ]
Минимум функции: 15.000000000000005

BFGS:
Кол-во итераций: 18479
Время выполнения: 1.6063323020935059 с
Точка минимума функции: [0.99999952 0.99999903]
Минимум функции: 15.000000000000469

DFP:
Кол-во итераций: 4453
Время выполнения: 0.7293074131011963 с
Точка минимума функции: [0.99999986 0.99999971]
Минимум функции: 15.000000000000043

Метод Левенберга-Марквардта:
Кол-во итераций: 5934
Время выполнения: 0.23388981819152832 с
Точка минимума функции: [0.99999937 0.99999874]
Минимум функции: 15.0000000000008
```

Рис. 1 — Результат вычислений

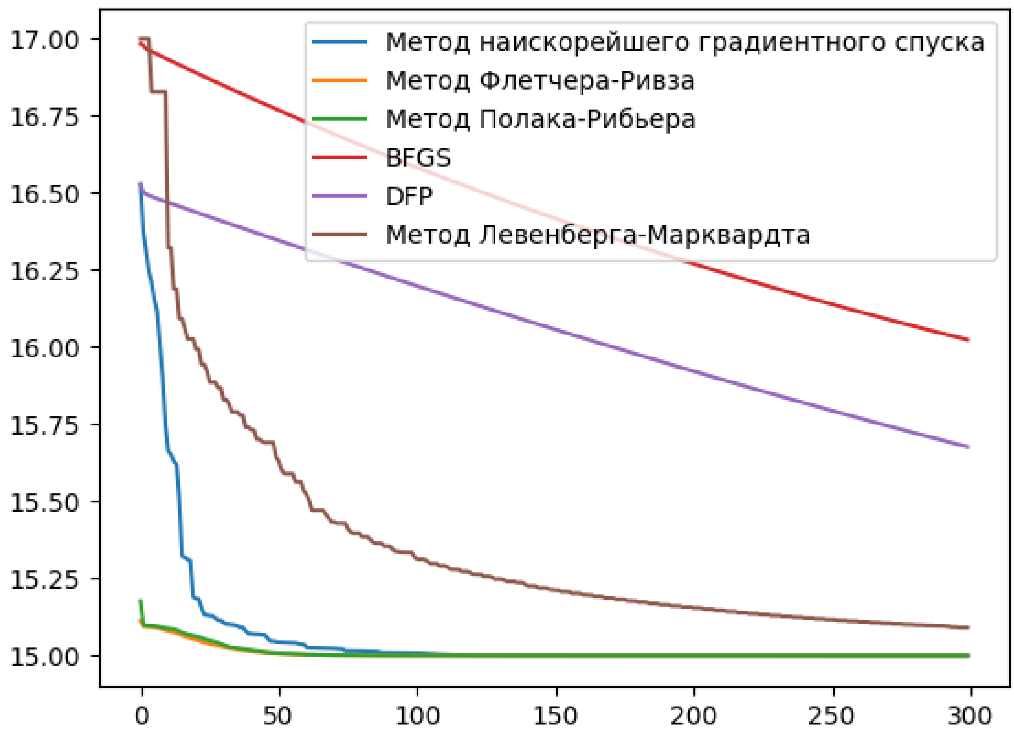


Рис. 2 — Сравнительный график сходимости методов многомерного поиска

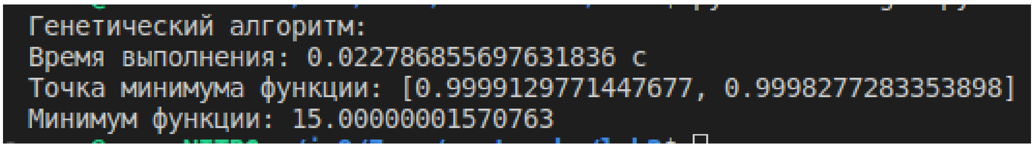


Рис. 3 — Результат работы генетического алгоритма

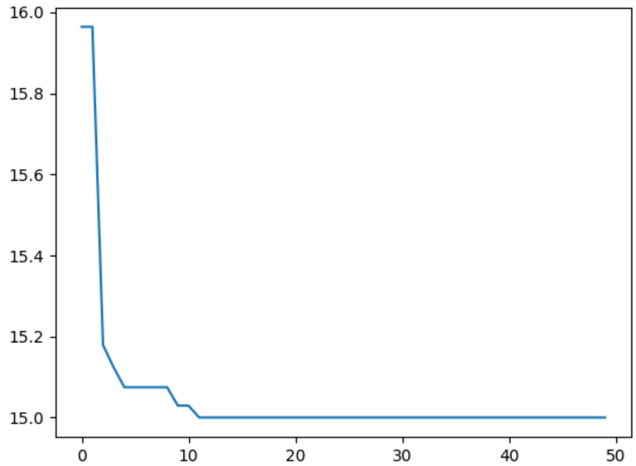


Рис. 4 — График генетического алгоритма

3. Наиболее сложным в реализации оказался генетиский алгоритм, а его результат - непредсказуем, т.к. зависит от случайно сгенерированных параметров.