ОТЧЕТ

о выполнении долгосрочного домашнего задания по

дисциплине «основы теории оптимизации»

Вариант №14

Выполнил: Гаврилин М.И.

Проверил: Тумачек А.С.

МОСКВА 2024

**Задание**:

Используя метод Флетчера–Ривса, минимизировать квадратичную функцию Химмельблау:

*f(x) = (x12+x2-11)2+(x1+x22-7)2*

Параметры алгоритма: *ITER\_MAX\_CG* – максимальное количество итераций метода Флетчера–Ривса, *ITER\_MAX\_GR* – максимальное количество итераций вспомогательного одномерного метода (метод золотого сечения) , *TOLERANCE\_CG* – параметр сходимости метода Флетчера–Ривса, *TOLERANCE\_GR*– параметр сходимости метода золотого сечения (меньше или равен *TOLERANCE\_CG*).

Значения параметров по умолчанию:

*ITER\_MAX\_CG* = 10000

*ITER\_MAX\_GR* = 1000

*TOLERANCE\_CG* = 1e-10

*TOLERANCE\_GR =* 1e-10

Алгоритм реализован на языке программирования Python.

**Описание метода:**

Метод сопряженных направлений (метод Флетчера–Ривса) – это метод оптимизации, который использует градиент функции для поиска минимума, начиная с заданной точки. В данной работе реализован метод сопряженных направлений, в котором каждое новое сопряженное направление рассчитывается по формуле Флетчера–Ривса.

Формула Флетчера–Ривса имеет следующий вид:

где:

Обозначения:

– вектор сопряженного направления на k–ом шаге

– вектор антиградиента на k-ом шаге

Идея метода состоит в уменьшении функции при последовательном движении по сопряженным направлениям. Для нахождения минимума при движении по сопряженному направлению используется метод золотого сечения.

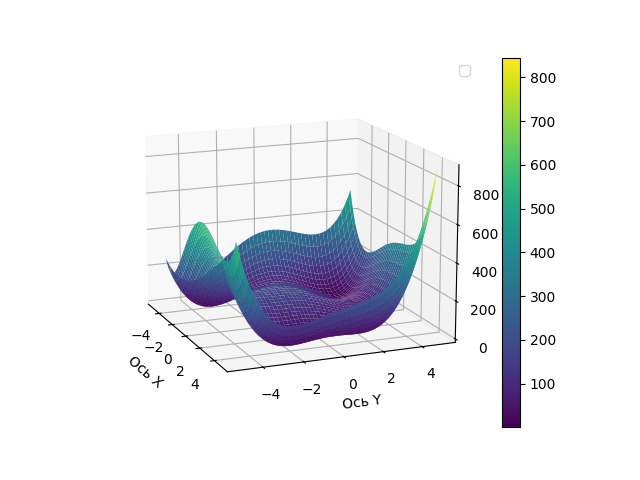
**Реализация алгоритма:**

В процессе реализации и тестирования алгоритма получилось достичь всех 4 –х точек минимума функции Химмельблау.

**Исходный код:**

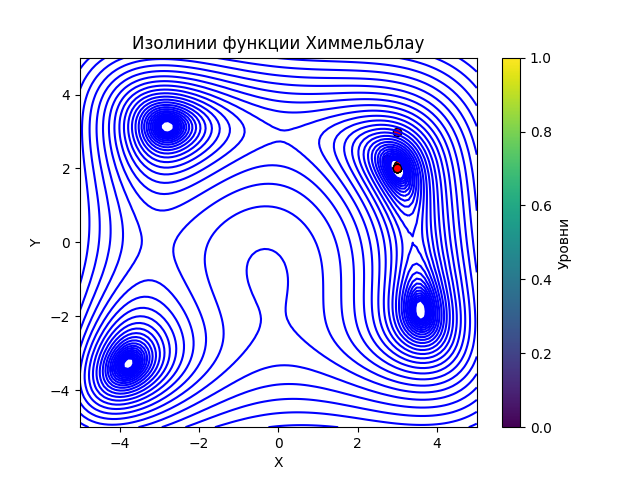
import numpy as np  
import numpy.typing as npt  
import matplotlib.pyplot as plt  
from typing import Callable, Tuple  
  
ITER\_MAX\_CG = 10000  
ITER\_MAX\_GR = 1000  
TOLERANCE\_CG = np.float64(1e-10)  
TOLERANCE\_GR = np.float64(1e-10)  
  
  
# Определяем квадратичную функцию Химмельблау для построения графиков  
def him\_pl(x: float, y: float) -> float:  
 return (x\*\*2+y-11)\*\*2+(x+y\*\*2-7)\*\*2  
  
  
# Определяем квадратичную функцию Химмельблау для оптимизации  
def himmelblau(x: npt.NDArray[np.float64]) -> np.float64:  
 if len(x) != 2:  
 raise ValueError("Массив должен состоять из 2-х элементов")  
 return np.float64((x[0]\*\*2+x[1]-11)\*\*2+(x[0]+x[1]\*\*2-7)\*\*2)  
  
  
# Вручную задаем градиент целевой функции  
def gradient(x: npt.NDArray[np.float64]) -> npt.NDArray[np.float64]:  
 return np.array([4\*x[0]\*(x[0]\*\*2+x[1]-11)+2\*x[0]+2\*x[1]\*\*2-14,  
 2\*x[0]\*\*2+4\*x[1]\*(x[0]+x[1]\*\*2-7)+2\*x[1]-22], dtype=np.float64)  
  
  
# Функция поиска минимума с использованием метода золотого сечения  
def golden\_ratio\_search(f: Callable[[np.float64], np.float64], # Функция, для которой ищем минимум  
 a: np.float64, b: np.float64, # Левый и правый концы интервала  
 itr=ITER\_MAX\_GR, tol=TOLERANCE\_GR # Максимальное число итераций и точность  
 ) -> np.float64:  
 if tol > TOLERANCE\_CG:  
 tol = TOLERANCE\_CG  
 iterations = 0  
 while iterations < itr:  
 iterations += 1  
 phi = np.float64((1+np.sqrt(5))/2)  
 x1 = b - (b - a) / phi  
 x2 = a + (b - a) / phi  
 y1 = f(x1)  
 y2 = f(x2)  
 if y1 >= y2:  
 a = x1  
 else:  
 b = x2  
 if np.abs(b - a) < tol:  
 break  
 return (a + b) / 2  
  
  
def cg(fun: Callable[[npt.NDArray[np.float64]], np.float64], # Целевая функция для оптимизации  
 grad: Callable[[npt.NDArray[np.float64]], npt.NDArray[np.float64]], # Ее градиент  
 x0: npt.NDArray[np.float64], itr: int = ITER\_MAX\_GR, # Нач точка и число итераций  
 tol: np.float64 = TOLERANCE\_CG # Погрешность метода  
 ) -> Tuple[npt.NDArray[np.float64], list]:  
 # Одномерный алгоритм оптимизации использующий метод золотого сечения  
 def goldenratio\_fcg(x\_f, d\_f) -> np.float64:  
 def func\_x(x: np.float64):  
 return fun(x\_f + x \* d\_f)  
 # 1 шаг - устанавливаем интервал  
 delt = np.float64(0.01)  
 a0 = np.float64(0)  
 a\_it = a0 + delt  
 b\_it = a0  
 k = 1  
 while True:  
 # Спускаемся в выбранном направлении  
 f\_old, f\_new = func\_x(b\_it), func\_x(a\_it)  
 if f\_old >= f\_new:  
 b\_it = a\_it  
 a\_it += 2 \*\* k \* delt  
 k += 1  
 continue  
 if f\_old <= f\_new:  
 break  
 # 2 шаг - уменьшаем интервал, используя метод золотого сечения  
 return golden\_ratio\_search(func\_x, b\_it-delt, a\_it+delt)  
  
 # Находим размерность алгоритма  
 dim = x0.size  
 x\_it = x0  
 convergence = [x0]  
 # 1 шаг - вычисляем анти градиент функции в начальной точке  
 r = -1 \* grad(x\_it)  
 d = r  
 iterations = 0  
 while iterations < itr and np.sqrt(np.dot(r.T, r)) > tol:  
 # 2 шаг - используя одномерную минимизацию, находим коэффициент a  
 a = goldenratio\_fcg(x\_it, d)  
 # 3 шаг - переход в точку, в которой функция принимает минимальное значение  
 x\_it = x\_it + a \* d  
 # 4 шаг - вычисляем анти градиент в этой точке  
 r\_old = r  
 r = -1 \* grad(x\_it)  
 # 5 шаг - вычисляем коэффициенты алгоритма  
 if iterations % (dim + 1) != 0 or iterations == 0:  
 b = np.dot(r.T, r) / np.dot(r\_old.T, r\_old)  
 else:  
 b = 0  
 # 6 шаг - вычисляем новое сопряженное направления  
 d = r + b \* d  
 # Добавляем точку в массив, увеличиваем число итераций  
 convergence.append(x\_it)  
 iterations += 1  
 if iterations == ITER\_MAX\_CG:  
 print(f'Алгоритм завершил работу, поскольку число итераций превысило {ITER\_MAX\_CG}')  
 if np.sqrt(np.dot(r.T, r)) < tol:  
 print(f'Алгоритм завершил работу, поскольку значение градиента '  
 f'функции в найденной точке меньше чем {tol}')  
 print(f'Число итераций - {iterations}')  
 print(f'Найденное решение - {x\_it}')  
 print(f'Значение функции в точке минимума - {fun(x\_it)}')  
 return x\_it, convergence  
  
  
# Решаем задачу оптимизации  
(sol, conv) = cg(himmelblau, gradient, np.array([2, -2]))  
fval = himmelblau(sol)  
  
# Построение графиков.  
# Создаем сетку значений для x и y  
x\_lin = np.linspace(-5, 5, 100)  
y\_lin = np.linspace(-5, 5, 100)  
x\_lin, y\_lin = np.meshgrid(x\_lin, y\_lin)  
  
# Вычисляем z на основе функции f  
z = him\_pl(x\_lin, y\_lin)  
  
# Создаем фигуру и ось 3D  
fig1 = plt.figure(1)  
ax = fig1.add\_subplot(111, projection='3d')  
  
# Рисуем поверхность  
surface = ax.plot\_surface(x\_lin, y\_lin, z, cmap='viridis', alpha=0.4)  
  
# Добавляем на поверхность точки  
conv = np.array(conv)  
ax.scatter(conv[:, 0], conv[:, 1], him\_pl(conv[:, 0], conv[:, 1]),  
 color='red', edgecolor='black', label='Точки сходимости')  
ax.plot(conv[:, 0], conv[:, 1], him\_pl(conv[:, 0], conv[:, 1]),  
 linestyle='--', color='red')  
for i, point in enumerate(conv):  
 ax.text(point[0], point[1], him\_pl(point[0], point[1]),  
 f'{i}', color='black', fontsize=10)  
# Добавляем цветовую шкалу  
fig1.colorbar(surface)  
  
# Добавляем метки осей  
ax.set\_xlabel('Ось X')  
ax.set\_ylabel('Ось Y')  
ax.set\_zlabel('Ось Z')  
  
# Добавляем легенду графика  
ax.legend()  
# Показываем график  
plt.show()  
# Строим график с изолиниями  
fig2 = plt.figure(2)  
plt.contour(x\_lin, y\_lin, z, levels=np.logspace(0, 5, 50), colors='blue')  
plt.scatter(conv[:, 0], conv[:, 1], color='red', edgecolor='black', label='Точки сходимости')  
plt.plot(conv[:, 0], conv[:, 1], linestyle='--', color='red')  
plt.colorbar(label='Уровни')  
# Добавляем метки осей и заголовок  
plt.xlabel('X')  
plt.ylabel('Y')  
plt.title('Изолинии функции Химмельблау')  
  
# Показываем график  
plt.show()  
  
# Строим график сходимости  
fig3 = plt.figure(3)  
eps\_arr = [np.sqrt(np.dot(gradient(p).T, gradient(p))) for p in conv]  
plt.plot(eps\_arr)  
plt.title('График сходимости алгоритма')  
plt.xlabel('Итерация')  
plt.ylabel('|∇f(x)| (логарифмический масштаб)')  
plt.yscale('log')  
plt.xticks(range(len(eps\_arr)))  
plt.show()

**График функции f(x)=(x12+x2-11)2+(x1+x22-7)2:**



**Иллюстрация движения алгоритма из разных точек:**

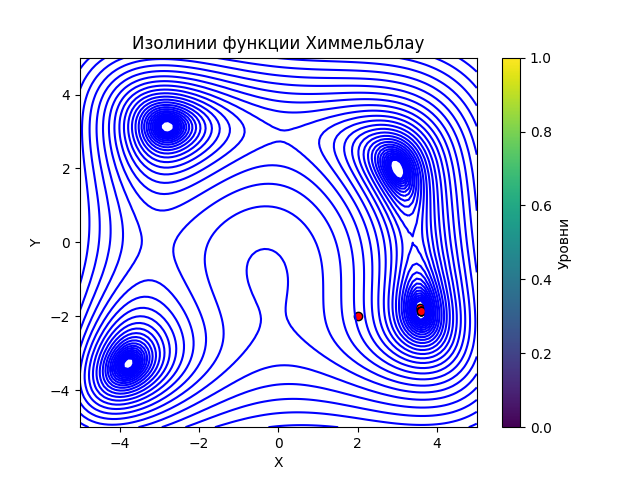
1. *Исходная точка [3, 3]*

**

*Минимум достигнут в точке: (3.0000000000030127, 1.9999999999959355)*

*Значение функции в минимуме: 3.717741662530276e-22*

1. *Исходная точка [2, -2]*

**

*Минимум достигнут в точке: (-2.80511808695264, 3.1313125182505708)*

*Значение функции в минимуме: 3.5617464301181294e-25*

1. *Исходная точка [-1, 3]*

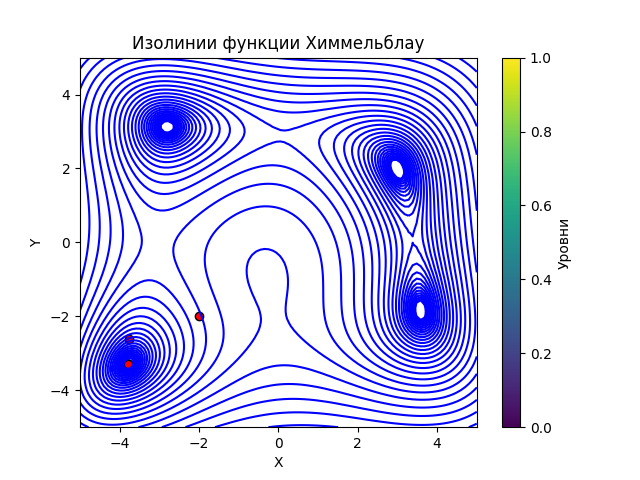
*A diagram of a graph

Description automatically generated with medium confidence*

*Минимум достигнут в точке: (3.584428340386681, -1.84812652694275)*

*Значение функции в минимуме: 1.8074142742704456e-19*

1. *Исходная точка [-2, -2]*

**

*Минимум достигнут в точке: (-3.7793102533872682, -3.283185991293991)*

*Значение функции в минимуме: 5.865328505868541e-21*

* На графиках видно, что алгоритм позволяет быстро прийти к точке минимума.

**Графики сходимости ошибки метода сопряженных направлений для каждой из начальных точек:**

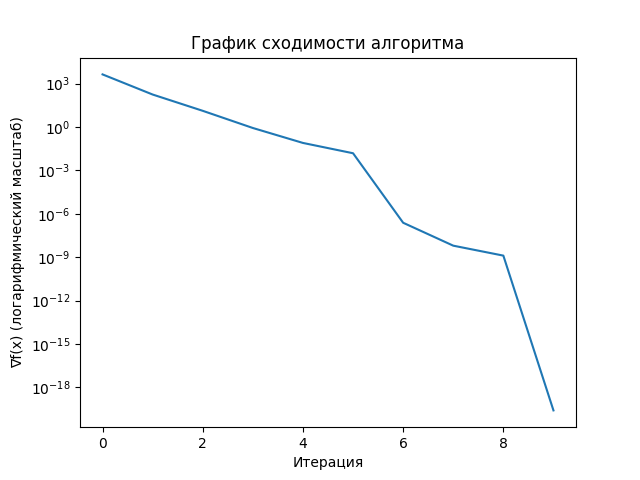


Рис. 1 Исходная точка (3, 3)

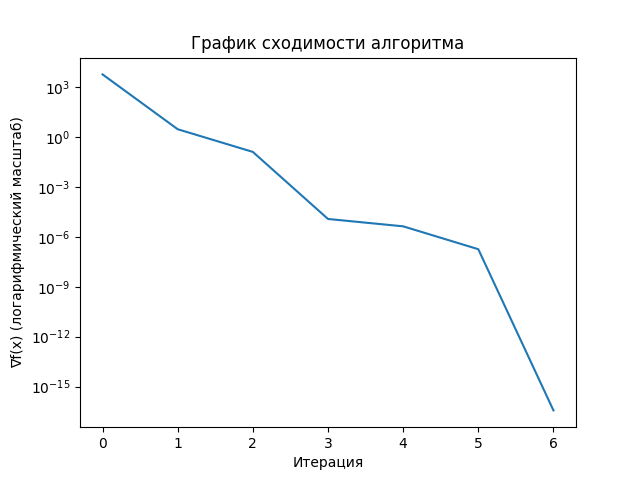


Рис. 2 Исходная точка (2, -2)

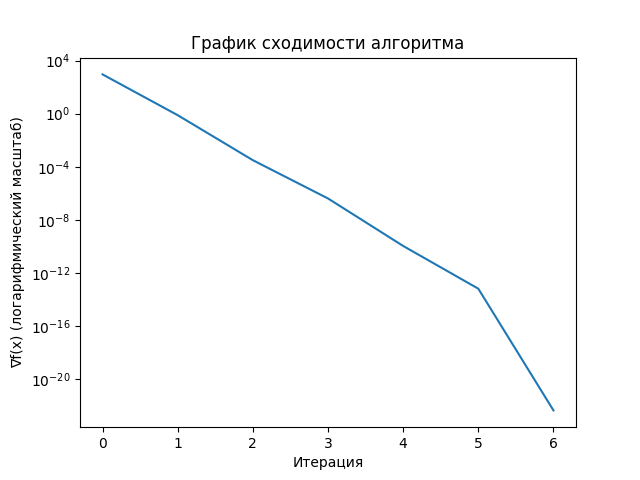


Рис. 3 Исходная точка (-1, 3)

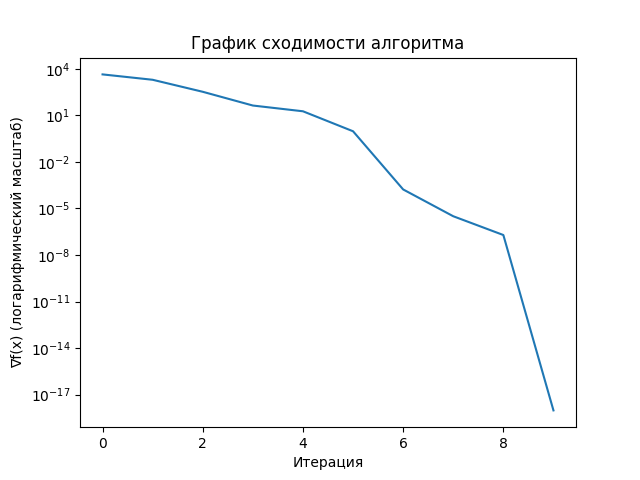


Рис. 4 Исходная точка (-2, -2)

* Графики показывают, насколько быстро уменьшается норма градиента в зависимости от числа итераций

**Вывод:**

В ходе выполнения задания был изучен алгоритм минимизации функции Химмельблау методом сопряженных направлений.

Были построены графики движения алгоритма для четырех различных начальных точек.

Были построены графики сходимости ошибки метода для каждой из начальных точек.