**Национальный исследовательский технологический университет «МИСиС»**

**Институт Информационных технологий и компьютерных наук (ИТКН)**

**Курс «Методы оптимизации»**

Лабораторная работа № 5

по теме

«Численные методы многомерной оптимизации нулевого порядка»

Вариант №23

Выполнил:

Студент группы БИВТ-20-1

Смирнов А.А.

Проверил:

к.ф.-м.н., доцент, Лычев А.В.

Москва, 2023

Цель: приобретение практических навыков для решения задач многомерной минимизации численными методами нулевого порядка.

# Ход работы:

## Задача

Вариант задания – №23.

Рассматриваемая функция – .

Требуется найти безусловный минимум функции многих переменной y = f(, …, ), то есть найти такую точку .

## Графическое представление функции

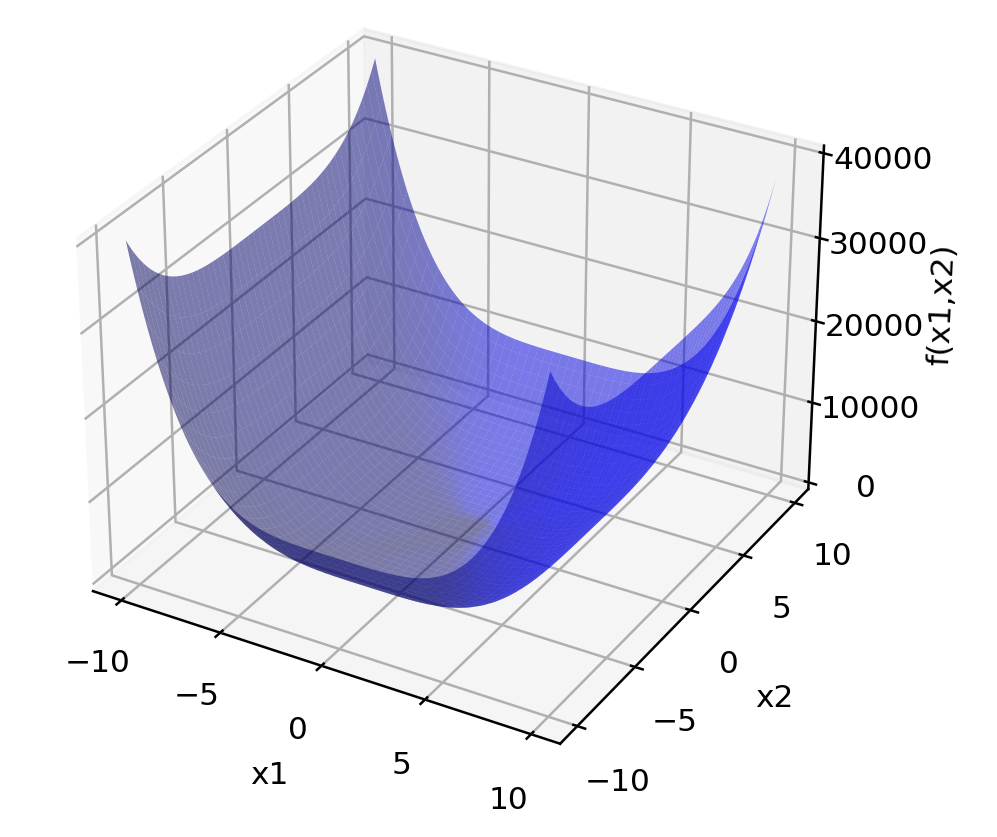


Рисунок 1 – График функции.

# Листинг программ

## 3.1. Метод случайного покоординатного спуска

Метод вычисления функции:

def f(x1, x2):

return 3\*x1\*\*4 - x1\*x2 + x2\*\*4 - 7\*x1 - 8\*x2 + 2

Метод вычисления коэффициента:

def find\_s(x1, x2, a, h):

return (f(x1 + a\*h[0], x2 + a\*h[1]) - f(x1, x2)) / a

Метод вычисления случайного направления:

def rand(n):

return random.randint(0, n)

Реализация метода случайного покоординатного спуска:

def random\_gradient\_decent(x1, x2, e, a):

k = 0

while True:

ej = [0, 0]

j = rand(2)

ej[j-1] = 1

sk = find\_s(x1, x2, a, ej)

x1\_next = x1 - a\*sk\*ej[0]

x2\_next = x2 - a\*sk\*ej[1]

if (sqrt(abs(x1\_next - x1)\*\*2 + abs(x2\_next - x2)\*\*2) <= e):

return [(round(x1\_next, round\_num),

round(x2\_next, round\_num),

round(f(x1\_next, x2\_next), round\_num)),

k]

x1 = x1\_next

x2 = x2\_next

k += 1

## 3.2. Симплексный метод

Метод вычисления функции:

def f(x1, x2):

return 3\*x1\*\*4 - x1\*x2 + x2\*\*4 - 7\*x1 - 8\*x2 + 2

Методы вычисления коэффициентов:

def calc\_r(a, n):

return a \* (sqrt(n + 1) - 1 + n) / (n \* sqrt(2))

def calc\_s(a, n):

return a \* (sqrt(n + 1) - 1) / (n \* sqrt(2))

Методы вычисления расстояния между точками:

def dist(x1, x2):

return sqrt((x1[0] - x2[0])\*\*2 + (x1[1] - x2[1])\*\*2)

Методы поиска индекса максимального значения:

def max\_f(f):

f\_max = max(f)

index = f.index(f\_max)

return index

Метод поиска следующего значения:

def calc\_x\_next(x, index, n):

res = np.array([0,0])

for i in range(len(x)):

if (i == index): continue

res = res + x[i]

res \*= (2 / (n - 1))

res -= x[index]

return res

Метод вычисления центра симплекса:

def calc\_centr(x):

return (x[0][0]+x[1][0]+x[2][0]) / 3, (x[0][1]+x[1][1]+x[2][1]) / 3

Реализация симплексного метода:

def simplexnyi\_method(x0, x1, x2, a, n):

global counter;

k = 0

x = [x0, x1, x2]

counter += 3

while (dist(x[0], x[1]) > e or dist(x[1], x[2]) > e or dist(x[2], x[0]) > e):

f\_list = []

counter += 1

f\_list.append(f(x[0][0], x[0][1]))

f\_list.append(f(x[1][0], x[1][1]))

f\_list.append(f(x[2][0], x[2][1]))

while(True):

f\_values = f\_list

i = max\_f(f\_values)

xn = calc\_x\_next(x, i, n)

fn = f(xn[0], xn[1]); counter += 1

if (f\_values[i] > fn): x[i] = xn ; break

f\_values[i] = min\_val

if (f\_values[0] == min\_val and f\_values[1] == min\_val and f\_values[2] == min\_val):

a /= 2

x[0] = x[0]

x[1] = np.array([x[0][0] + calc\_r(a, n), x[0][1] + calc\_s(a, n)])

x[2] = np.array([x[0][0] + calc\_s(a, n), x[0][1] + calc\_r(a, n)])

break

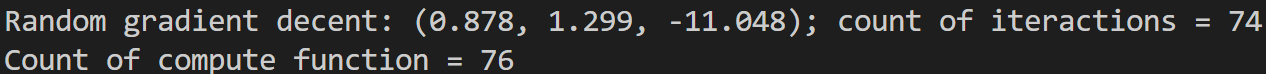
k += 1

point = calc\_centr(x)

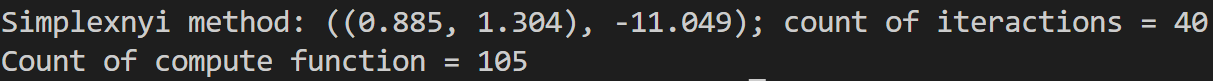
return [(point, f(point[0], point[1])), k]

# Результаты вычислений

## 4.1. Метод случайного покоординатного спуска



## 4.2. Симплексный метод



# Графическое представление траекторий

## 5.1. Метод случайного покоординатного спуска

Начальные условия:

= 0 = 0

e = 0.001 a = 0.01

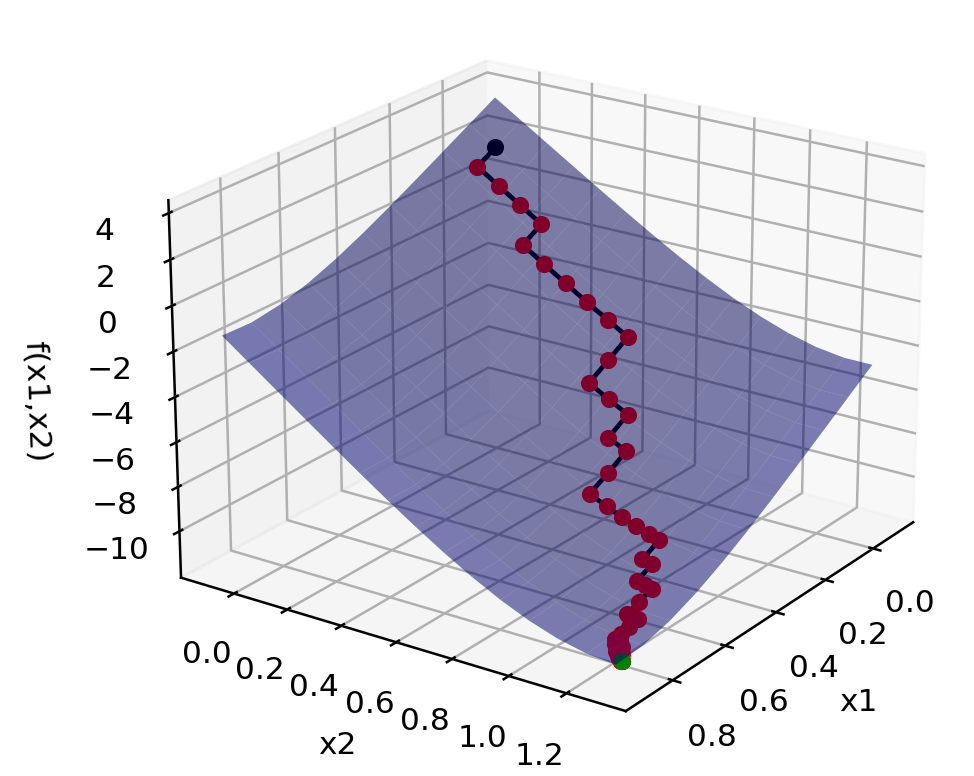


Рисунок 1 – Траектория движения 1.

## 5.2. Симплексный метод

Начальные условия:

= 2 = 2

e = 0.001 a = 2 n = 3

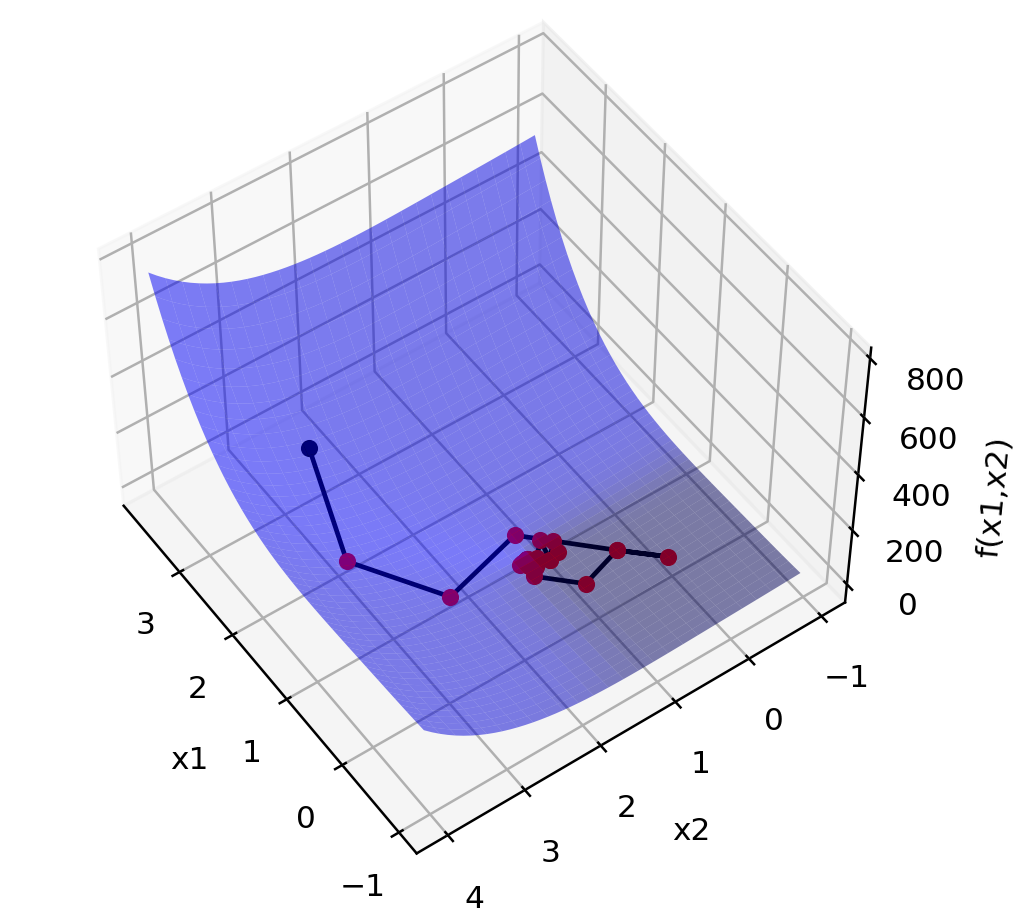


Рисунок 2 – Траектория движения 2.

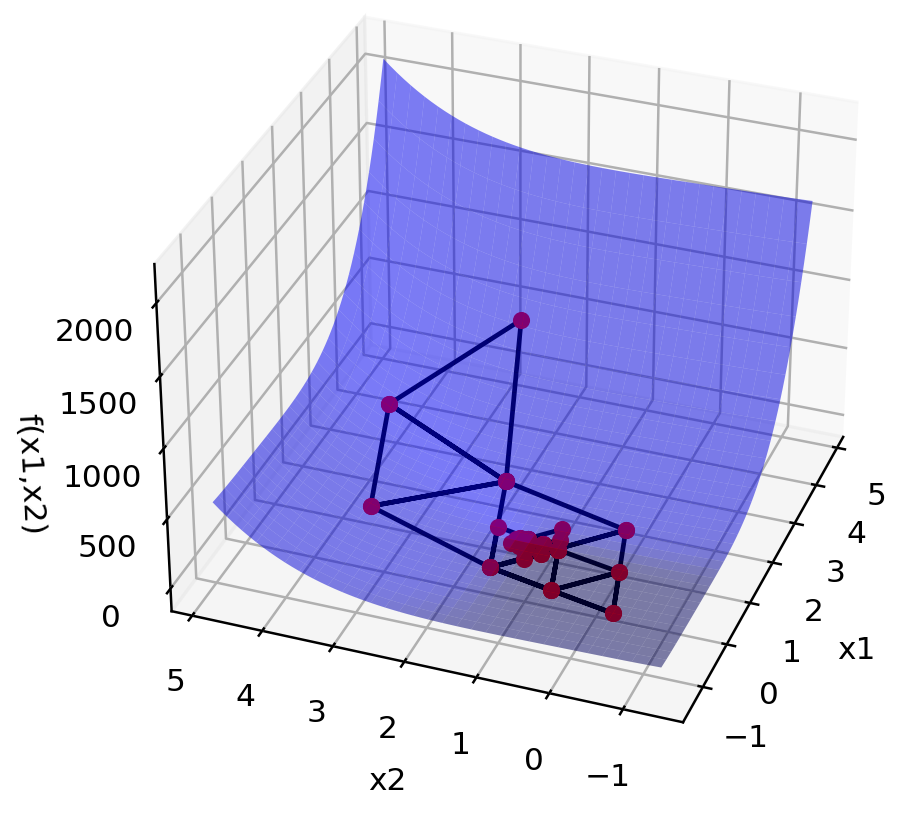


Рисунок 2a – Траектория движения 2.

# Сравнительная характеристика

1. Метод случайного покоординатного спуска
2. Симплексный метод

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | e | a |  |  | k | fk |  |  |  |
| 1 | 0.0001 | 0.01 | 0 | 0 | 74 | 46 | 0.883 | 1.300 | -11.049 |
| 0.0001 | 0.01 | 2 | 2 | 44 | 76 | 0.878 | 1.299 | -11.048 |
| 0.00001 | 0.01 | 0 | 0 | 90 | 92 | 0.879 | 1.299 | -11.049 |
|  | 0.00001 | 0.01 | 2 | 2 | 64 | 66 | 0.880 | 1.300 | -11.049 |
| 2 | 0.001 | 2 | 0 | 0 | 40 | 105 | 0.885 | 1.304 | -11.049 |
| 0.001 | 2 | 2 | 2 | 40 | 105 | 0.886 | 1.305 | -11.049 |
| 0.0001 | 2 | 0 | 0 | 57 | 147 | 0.885 | 1.305 | -11.049 |
|  | 0.0001 | 2 | 2 | 2 | 55 | 143 | 0.884 | 1.305 | -11.049 |

Вывод: в результате выполнение лабораторной работы я приобрел практические навыки для решения задач многомерной минимизации численными методами нулевого порядка. Для заданной функции нашел минимум двумя способами: метод случайного покоординатного спуска и симплексный метод.

Симплексный метод имеет меньше итераций и большую точность, однако больше вычислений функции. Также стоит отметить, что количество итерация для случайного покоординатного спуска непостоянно и различно для каждого отдельно взятого запуска программы.