**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра автоматизированных систем управления (АСУ)

**Оптимизация функции** **нескольких переменных**

Отчет по лабораторной работе №3

По дисциплине

«Методы оптимизации»

|  |
| --- |
| Выполнили: |
| Студент гр. 439-2 |
| Е.В. Раззоренов  Студент гр. 439-1  И.Е. Кривоносов |
| 3.10.2021 г. |
|  |

|  |
| --- |
| Руководитель: |
| А.А. Шелестов |
| «\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г. |

2021

**Задание**

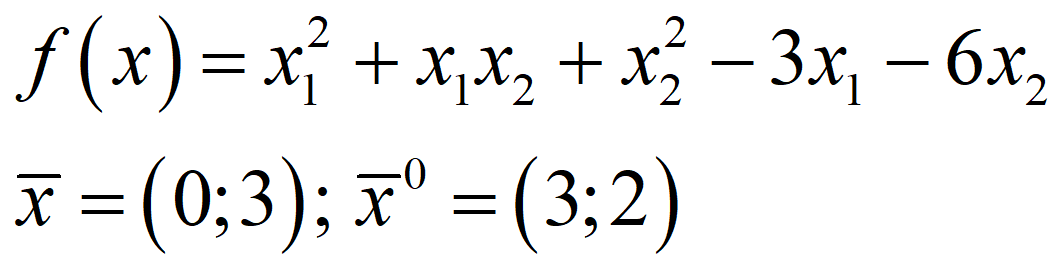
Найти минимум функции двух переменных. Использовать следующие методы:

Два прямых метода (симплексный метод, метод Хука-Дживса);

Точность .

**Вариант задания**

Вариант №1:



**График функции**

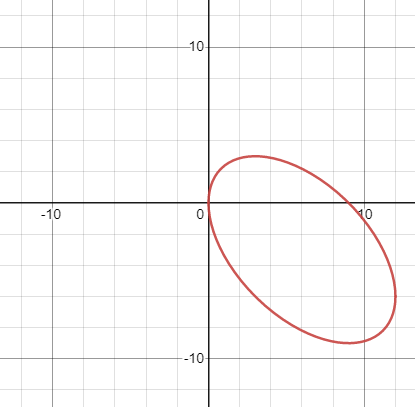


Рис. 1 - График исходной функции

**Метод Хука-Дживса**

Нахождение в окрестности текущей точки наилучшей и движение в этом направлении. Если значение в окрестных точках больше, чем в текущей, то происходит уменьшение шага.

**Алгоритм Хука-Дживса**

**Шаг 1.** Определить начальную точку ; приращения (шаги) коэффициент уменьшения шага α > 1; параметр окончания поиска ε .

**Шаг 2.** Провести исследующий поиск.

**Шаг 3.** Был ли исследующий поиск удачным (найдена ли точка с меньшим значением ЦФ)?

Да: переход на Шаг 5. Нет: продолжить, т.е. переход на Шаг 4.

**Шаг 4.** Проверка на окончание поиска. Выполняется ли неравенство ? Да: окончание поиска, т.е. текущая точка аппроксимирует точку экстремума . Нет: уменьшить приращение . Переход на Шаг 2.

**Шаг 5.** Провести поиск по образцу: .

**Шаг 6.** Провести исследующий поиск, используя точку в качестве временной базовой точки. Пусть в результате получена точка .

**Шаг 7.** Выполняется ли неравенство: ?

Да: положить . Переход на Шаг 5.

Нет: переход на Шаг 4.

**Симплексный метод**

Приближение к минимальной точке с помощью изменения координат вершин симплекса

**Алгоритм Симплексного метода**

**Шаг 1.** Задается исходная вершина симплекса.

Задается коэффициент сжатия γ∈[0,1] и размер симплекса L. Строится симплекс

Здесь j-я строка – это координаты j-ой вершины . , где n - размерность пространства (размерность вектора x), i – номер координаты . Определение координат , начиная со второй, производится по формуле

где - матрица размерности

Где

Векторы, соответствующие вершинам , составят одинаковые углы с координатными осями .

**Шаг 2.** В вершинах симплекса вычисляется ЦФ .

**Шаг 3.** Проверяем условия:

Если «да», то конец; если «нет», то переходим на Шаг 4.

**Шаг 4.** Находится «наихудшая» вершина симплекса (при поиске минимума «наихудшая» вершина – та, в которой значение функции максимально).

**Шаг 5.** Осуществляется расчет координат новой вершины (вершина отражения ):

**Шаг 6.** Если точка оказывается «хуже» всех остальных точек симплекса, то осуществляется возврат к исходному симплексу с последующим его сжатием относительно «лучшей» из вершин

Переход на Шаг 2.

Если не является «худшей» в новом симплексе, то перейти на шаг 3.

**Листинг**

import math  
import numpy as np  
  
  
class Polynomial:  
 def \_\_init\_\_(self):  
 data\_list = [line.strip() for line in open(f'Input.txt', 'r')]  
 self.pol\_str = data\_list[0]  
 self.points = data\_list[1].split()  
 self.F = lambda x: x[0] \*\* 2 + x[0] \* x[1] + x[1] \*\* 2 - 3 \* x[0] - 6 \* x[1]  
 self.f = lambda x,y: x \*\* 2 + x \* y + y \*\* 2 - 3 \* x - 6 \* y  
  
 def simplex\_method(self):  
 x1 = float(self.points[0])  
 x2 = float(self.points[1])  
 gamma = 0.5  
 n = len(self.points)  
 delta = 1  
 p = delta \* ((n + 1) \*\* 0.5 + n - 1) / (n \* 2 \*\* 0.5)  
 g = p - delta \* (2 \*\* 0.5 / 2)  
  
 simplex = np.zeros((n + 1, n))  
 simplex[0] = x1, x2  
  
 for i in range(1, n + 1):  
 for j in range(n):  
 if i - 1 == j:  
 simplex[i][j] = simplex[0][j] + p  
 else:  
 simplex[i][j] = simplex[0][j] + g  
  
 while True:  
 sum\_col1, sum\_col2 = self.sum\_simplex(simplex)  
 f\_list = [self.F(row) for row in simplex]  
  
 middle\_x = [sum\_col1 / (n + 1), sum\_col2 / (n + 1)]  
 max\_index = f\_list.index(max(f\_list))  
  
 r\_x = [(sum\_col1 - simplex[max\_index][0]) \* 2 / n - simplex[max\_index][0],  
 (sum\_col2 - simplex[max\_index][1]) \* 2 / n - simplex[max\_index][1]]  
  
 if self.F(r\_x) <= self.F(simplex[max\_index]):  
 simplex[max\_index] = r\_x  
 else:  
 delta \*= gamma  
 min\_index = f\_list.index(min(f\_list))  
  
 a = [simplex[min\_index][0] \* (1 - gamma), simplex[min\_index][1] \* (1 - gamma)]  
  
 for i in range(n + 1):  
 simplex[i] = summ(gamma \* simplex[i], a)  
  
 # for i in range(n + 1):  
 # for j in range(n + 1):  
 # if i != j:  
 # print("check = ", np.linalg.norm(sub(simplex[i], simplex[j])), delta)  
  
 sum\_col1, sum\_col2 = self.sum\_simplex(simplex)  
 new\_x = [sum\_col1 / (n + 1), sum\_col2 / (n + 1)]  
  
 if np.linalg.norm(sub(new\_x, middle\_x)) <= 0.00001 and abs(self.F(new\_x) - self.F(middle\_x)) <= 0.00001:  
 return new\_x  
  
 def sum\_simplex(self, simplex):  
 temp = 0  
 temp1 = 0  
 for i in range(len(self.points) + 1):  
 temp += simplex[i][0]  
 temp1 += simplex[i][1]  
 return temp, temp1  
  
 def jivs\_method(self):  
 res = [float(self.points[0]), float(self.points[1])]  
 x = res  
 gamma = 2  
 delta = [0.3, 0.3]  
  
 i = 0  
 while True:  
 e = [0, 1] if i % 2 == 0 else [1, 0]  
  
 new\_x = [x[0] + delta[0] \* e[0], x[1] + delta[1] \* e[1]]  
 if self.F(new\_x) < self.F(x):  
 x = new\_x  
 new\_res = self.sample\_search(x, res)  
 else:  
 new\_x = [x[0] - delta[0] \* e[0], x[1] - delta[1] \* e[1]]  
 if self.F(new\_x) < self.F(x):  
 x = new\_x  
 new\_res = self.sample\_search(x, res)  
 else:  
 new\_res = res  
 delta = [delta[0] / gamma, delta[1] / gamma]  
  
 if np.linalg.norm(delta) <= 0.00001 and abs(self.F(new\_res) - self.F(res)) <= 0.00001:  
 return new\_res  
 x = res = new\_res  
 i += 1  
  
 def sample\_search(self, base\_x, x):  
 sample\_x = summ(base\_x, sub(base\_x, x))  
 return sample\_x if self.F(sample\_x) < self.F(base\_x) else base\_x  
  
  
def summ(a, b):  
 return [x + y for x, y in zip(a, b)]  
  
  
def sub(a, b):  
 return [x - y for x, y in zip(a, b)]  
  
  
p = Polynomial()  
s\_res = p.simplex\_method()  
print(f'x = {s\_res}')  
j\_res = p.jivs\_method()  
print(f'x = {j\_res}')

**Результат работы программы**

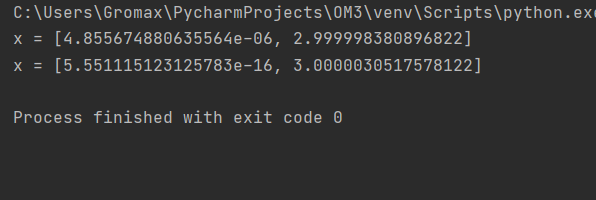


Рис. 2 – Результат работы программы