Министерство науки и высшего образования РФ ФГАОУ ВПО

Национальный исследовательский технологический университет «МИСИС»

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра автоматизированных систем управления (АСУ)

Отчет по лабораторной работе №4

по дисциплине «Методы оптимизаций»

Вариант 5

Выполнил:

Студент: Безыкорнов Н.Б.

         Группа: БИВТ-20-4

Проверил:

Лычев А.В.

Москва, 2023

**Цель работы:**

Приобретение практических навыков для решения задач многомерной минимизации различными численными методами нулевого порядка.

**Постановка задачи:**

Требуется найти безусловный минимум функции многих переменных y = f(x1, . . . , xn), то есть такую точку x ∗ ∈ R n , что f(x ∗ ) = min x∈Rn f(x)

Вариант 5.

В лабораторной работе использовались методы №№ 5 и 7. Метод случайного поиска с парными пробами и метод минимизации по правильному симплексу.

Функция:

Минимум функции находится в точке [5, 4] и равен 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название метода | Число итераций | Количество вычислений функций | Найденное решение | Значение функции |
| Случайного поиска с парными пробами | 100 | 2001 | [5.08, 4.10] | 1.01 |
| Минимизации по правильному симплексу | 34 | 167 | [5.00, 4.00] |  |

Изображение выглядит как диаграмма

Автоматически созданное описание

Рис 1 – график функции

Ход работы:

1. Метод случайного поиска с парными пробами

import random  
  
random.seed(13)  
  
# определяем функцию, для которой ищем минимум  
def f(x, y):  
 global calcs  
 calcs += 1  
 return ((x - 5) \*\* 2) \* ((y - 4) \*\* 2) + (x - 5) \*\* 2 + (y - 4) \*\* 2 + 1  
  
  
# задаем интервалы, в которых будем искать минимум  
x\_range = [-10, 10]  
y\_range = [-10, 10]  
calcs = 0  
  
# задаем количество итераций и количество парных проб  
num\_iterations = 100  
num\_pairs = 10  
  
# задаем начальную точку  
best\_x = random.uniform(\*x\_range)  
best\_y = random.uniform(\*y\_range)  
best\_result = f(best\_x, best\_y)  
  
# выполняем цикл поиска  
for i in range(num\_iterations):  
 # генерируем пары точек и сравниваем результаты  
 for j in range(num\_pairs):  
 x1 = random.uniform(\*x\_range)  
 y1 = random.uniform(\*y\_range)  
 result1 = f(x1, y1)  
 x2 = random.uniform(\*x\_range)  
 y2 = random.uniform(\*y\_range)  
 result2 = f(x2, y2)  
 if result1 < result2:  
 x = x1  
 y = y1  
 result = result1  
 else:  
 x = x2  
 y = y2  
 result = result2  
 # обновляем лучший результат, если он улучшился  
 if result < best\_result:  
 best\_x = x  
 best\_y = y  
 best\_result = result  
  
  
print("Значение x:", best\_x)  
print("Значение y:", best\_y)  
print("Значение функции:", best\_result)  
print(f"Кол-во итераций {num\_iterations}, кол-во вычислений функции {calcs}")

листинг 1 – реализация алгоритма

Изображение выглядит как текст

Автоматически созданное описание

Рис 2 – результат работы программы

1. Метод минимизации по правильному симплексу

import numpy as np  
  
  
def F(x, y):  
 global func\_count  
 func\_count += 1  
 return ((x - 5) \*\* 2) \* ((y - 4) \*\* 2) + (x - 5) \*\* 2 + (y - 4) \*\* 2 + 1  
  
  
def nelder\_mead\_method():  
 alpha = 1.0  
 beta = 0.5  
 gamma = 2.0  
 eps = 1e-6  
 maxIterations = 10000  
  
 simplex = np.array([[1.0, 0.0, 0.0],  
 [0.0, 1.0, 0.0],  
 [0.0, 0.0, 1.0]])   
 iter\_count = 0  
  
 while iter\_count < maxIterations:  
 iter\_count += 1  
  
 fx = np.array([F(simplex[j, 0], simplex[j, 1]) for j in range(3)])  
  
 sortedIndices = np.argsort(fx)  
  
 xc = np.array([(simplex[sortedIndices[0], j] + simplex[sortedIndices[1], j]) / 2.0 for j in range(2)])  
  
 xr = xc + alpha \* (xc - simplex[sortedIndices[2], :2])  
 fr = F(xr[0], xr[1])  
  
 if fr < fx[0]:  
  
 xe = xc + gamma \* (xr - xc)  
 fe = F(xe[0], xe[1])  
  
 if fe < fx[0]:  
  
 simplex[sortedIndices[2], :2] = xe  
 else:  
 simplex[sortedIndices[2], :2] = xr  
 elif fx[0] <= fr and fr < fx[1]:  
  
 simplex[sortedIndices[2], :2] = xr  
 else:  
  
 xs = xc + beta \* (simplex[sortedIndices[2], :2] - xc)  
 fs = F(xs[0], xs[1])  
  
 if fs < fx[2]:  
  
 simplex[sortedIndices[2], :2] = xs  
 else:  
  
 for j in range(1, 3):  
 simplex[sortedIndices[j], :2] = simplex[sortedIndices[0], :2] + \  
 beta \* (simplex[sortedIndices[j], :2] - simplex[sortedIndices[0],  
 :2])  
  
 maxDiff = np.max(np.abs(fx - fx[0]))  
 if maxDiff < eps:  
 break  
  
 return simplex[sortedIndices[0], :2], fx[0], iter\_count, func\_count  
  
  
func\_count = 0  
x, fval, num\_iter, num\_func\_calls = nelder\_mead\_method()  
  
print(f"Минимум в точке: {x}")  
print(f"Значение в минимуме: {fval}")  
print(f"Вычисления функции: {num\_func\_calls}")  
print(f"Итерации: {num\_iter}")

Листинг 2 – реализация метода

Изображение выглядит как текст

Автоматически созданное описание

Рис 3 – результат работы программы при начальном симлексе как указано в программе

Изменим симплекс на

simplex = np.array([[1.0, 0.0, 0.0],  
 [0.0, 2.0, 0.0],  
 [0.0, 0.0, 3.0]])

листинг 3 – измененим симплекс

Изображение выглядит как текст

Автоматически созданное описание

Рис 4 – результат работы с измененными начальными данными

Вывод:

Метод случайного поиска не подходит для эффективного нахождения точек минимумов функций так как сильно зависит от случайных переменных, более того функция вычисляется слишком много раз. Метод минимизации по правильному симплексу более эффективен и может использоваться более эффективно.