Министерство науки и высшего образования РФ

ФГАОУ ВО Пермский национальный исследовательский

политехнический университет

Кафедра «Вычислительная математика, механика и биомеханика»

Отчет по лабораторной работе №2

по дисциплине «Теория принятия решений» по теме: «Методы прямого поиска минимума целевой функции нескольких переменных»

Выполнил: студент группу ИСТ-22-1б Кожин В. А.

Проверил: доцент каф. ВММБ Бояршинова Ирина Николаевна

Пермь, 2024

**Содержание:**

[Введение 3](#_Toc179202707)

[Основная часть 4](#_Toc179202708)

**Введение**

В данной лабораторной работе рассматриваются два метода оптимизации для минимизации функции:

f(x\_1, x\_2) = e^(x\_1^2 + x\_2^2) + ln(4 + x\_2^2)

**Цель работы:**

* Применить методы Хука-Дживса и градиентного спуска для нахождения минимума заданной функции.
* Сравнить эффективность и характеристики методов (сходимость, количество итераций, точность).
* Сделать выводы о применимости методов для данной функции.

Для реализации задачи программным способом использовались язык Python и библиотеки для математических вычислений.

Важными параметрами задачи являются:

* начальная точка x0 = [0,0],
* точность вычислений ϵ = 10−6,
* начальный шаг λ = 1

В результате работы будет определена координата точки минимума, минимальное значение функции и количество итераций, потребовавшихся для достижения заданной точности каждым из методов.

**Основная часть**

## **Обоснование применимости методов**

### ***Метод Хука-Дживса***

Метод Хука-Дживса является методом поиска экстремумов функции без использования производных. Он подходит для функции, которая может быть *сложной* или *негладкой*. В данной задаче метод применим, так как функция *является непрерывной* и определена на всей области поиска. Этот метод будет полезен в случае, если нельзя вычислить производные или если функция слишком сложна для использования градиентных методов.

### ***Метод градиентного спуска***

Метод градиентного спуска используется для минимизации *гладких* функций, где можно вычислить градиент. В данном случае функция *является гладкой*, и её градиент можно легко вычислить, что делает этот метод применимым для решения задачи.

1. **Описание алгоритмов**

#### **Метод Хука-Дживса**

Метод Хука-Дживса является эвристическим методом поиска минимума функции. Он не требует вычисления производных и подходит для решения задач оптимизации, где функция может быть сложной, негладкой или слабо дифференцируемой. Алгоритм состоит из следующих шагов:

1. **Инициализация**:

* Задаётся начальная точка x\_0​, величина шага λ, и точность ϵ.
* Начальная точка считается текущей точкой x\_k​.

1. **Поиск направления улучшения**:

* Для каждой координаты x\_i текущей точки перебираются три возможных положения: x\_i − λ, x\_i​, x\_i ​+ λ. Таким образом, вокруг текущей точки формируется множество новых точек.
* Значения функции f(x) вычисляются для всех новых точек.

1. **Выбор новой точки**:

* Сравниваются значения функции f(x) для всех рассмотренных точек. Выбирается точка с наименьшим значением f(x). Если эта точка лучше текущей, она становится новой текущей точкой x\_k​.

1. **Уменьшение шага**:

* Если ни одна из новых точек не привела к уменьшению значения функции, величина шага λ уменьшается, чтобы лучше исследовать окрестность текущей точки.

1. **Критерий остановки**:

* Процесс повторяется до тех пор, пока шаг λ не станет меньше заданной точности ϵ.

1. **Результат**:

* После завершения итераций возвращается координата точки минимума, минимальное значение функции и количество итераций.

Метод Хука-Дживса прост в реализации и универсален, однако может потребовать большого числа итераций, особенно для сложных функций.

#### **Метод градиентного спуска**

Метод градиентного спуска является аналитическим методом оптимизации, использующим производные функции для нахождения направления наибольшего убывания. Алгоритм подходит для минимизации гладких функций, где можно вычислить градиент. Алгоритм включает следующие шаги:

1. **Инициализация**:

* Задаётся начальная точка x\_0​, величина шага λ, и точность ϵ.
* Начальная точка считается текущей точкой x\_k​.

1. **Вычисление градиента**:

* На каждой итерации вычисляется градиент функции ∇f(x\_k) который показывает направление наибольшего увеличения функции.

1. **Обновление точки**:

* Новая точка вычисляется по формуле: x\_{k+1} = x\_k – λ \* ∇f(x\_k), где λ — величина шага.

1. **Проверка критерия остановки**:

* Проверяется изменение значений функции между текущей и новой точкой. Если выполняется условие: ∣f(x\_{k+1}) − f(x\_k)∣ < ϵ, алгоритм останавливается.

1. **Корректировка шага**:

* Если значение функции f(x\_{k+1}) больше или равно f(x\_k) уменьшается величина шага λ (обычно делением на 2).

1. **Результат**:

* После выполнения всех условий возвращаются координаты точки минимума, минимальное значение функции и количество итераций.

Метод градиентного спуска эффективно работает для гладких функций, где градиент легко вычисляется. Однако для функций с множеством локальных минимумов он может «застрять» в локальном минимуме или не сходиться при неправильном выборе величины шага.

1. **Программный код**

import math as m  
import matplotlib.pyplot as plt  
import numpy as np  
  
  
# Целевая функция  
def f(x):  
 try:  
 exp\_part = m.exp(x[0] \*\* 2 + x[1] \*\* 2)  
 except OverflowError:  
 exp\_part = float('inf') # Обработка переполнения  
 exp\_part = min(exp\_part, 1e100) # Ограничение значения экспоненты  
 return exp\_part + m.log(4 + x[1] \*\* 2)  
  
  
# Градиент целевой функции  
def grad\_f(x):  
 try:  
 exp\_part = m.exp(x[0] \*\* 2 + x[1] \*\* 2)  
 except OverflowError:  
 exp\_part = float('inf') # Обработка переполнения  
 exp\_part = min(exp\_part, 1e100) # Ограничение значения экспоненты  
 x1 = 2 \* x[0] \* exp\_part  
 x2 = 2 \* x[1] \* exp\_part + (2 \* x[1]) / (4 + x[1] \*\* 2)  
  
 # Нормализация градиента  
 norm = m.sqrt(x1 \*\* 2 + x2 \*\* 2)  
 if norm > 1e-10: # Предотвращение деления на ноль  
 x1 /= norm  
 x2 /= norm  
  
 return [x1, x2]  
  
  
# Метод Хука-Дживса  
def hook\_jiws(x0, lamb, epsilon, alpha=1):  
 xk = x0.copy()  
 lamb0 = lamb  
 iterations = 0  
 traj = [xk.copy()] # Траектория точек  
  
 while lamb >= epsilon:  
 iterations += 1  
 p = 0  
 elems = [xk]  
 for i in range(len(xk)):  
 temp\_elems = []  
 for elem in elems:  
 x1 = elem.copy()  
 x1[i] -= lamb  
 x2 = elem.copy()  
 x3 = elem.copy()  
 x3[i] += lamb  
 temp\_elems.append(x1)  
 temp\_elems.append(x2)  
 temp\_elems.append(x3)  
 elems = elems + temp\_elems  
 temp\_elems.clear()  
 elems = [list(t) for t in set(tuple(e) for e in elems)]  
 elems.remove(xk)  
 func\_values = [f(elem) for elem in elems]  
 min\_f = min(func\_values)  
 min\_x = elems[func\_values.index(min\_f)]  
 f\_x = f(xk)  
  
 if round(min\_f, 6) >= round(f\_x, 6):  
 p += 1  
 lamb = lamb - (lamb0 / m.exp(p))  
 else:  
 xk = xk  
 for i in range(len(min\_x)):  
 min\_x[i] -= xk[i]  
 min\_x[i] \*= alpha  
 xk[i] += min\_x[i]  
 traj.append(xk.copy())  
 p = 0  
  
 return xk, f(xk), iterations, traj  
  
  
# Градиентный спуск  
def gradient\_const(x0, lamb, epsilon):  
 xk = x0.copy()  
 xk\_1 = xk.copy()  
 iterations = 1  
 traj = [xk.copy()] # Траектория точек  
 g = grad\_f(xk)  
 for i in range(len(xk)):  
 xk\_1[i] = xk[i] - g[i] \* lamb  
  
 while abs(f(xk\_1) - f(xk)) >= epsilon:  
 iterations += 1  
 xk = xk\_1.copy()  
 g = grad\_f(xk)  
 for i in range(len(xk)):  
 xk\_1[i] = xk[i] - g[i] \* lamb  
 if f(xk\_1) >= f(xk):  
 lamb /= 2  
 traj.append(xk.copy())  
  
 return xk\_1, f(xk\_1), iterations, traj  
  
  
# Построение графиков  
def plot\_trajectories(hook\_traj, grad\_traj, func\_range=10):  
 x = np.linspace(-func\_range, func\_range, 400)  
 y = np.linspace(-func\_range, func\_range, 400)  
 X, Y = np.meshgrid(x, y)  
 Z = np.array([[f([i, j]) for i, j in zip(x\_row, y\_row)] for x\_row, y\_row in zip(X, Y)])  
  
 plt.contourf(X, Y, Z, levels=50, cmap='viridis')  
 plt.colorbar(label="Уровень функции f(x)")  
  
 hook\_traj = np.array(hook\_traj)  
 grad\_traj = np.array(grad\_traj)  
  
 plt.plot(hook\_traj[:, 0], hook\_traj[:, 1], 'ro-', label="Хук-Дживс")  
 plt.plot(grad\_traj[:, 0], grad\_traj[:, 1], 'bo-', label="Градиентный спуск")  
 plt.scatter(hook\_traj[0, 0], hook\_traj[0, 1], color='k', label="Начальная точка")  
  
 plt.xlabel("x₁")  
 plt.ylabel("x₂")  
 plt.title("Траектории методов оптимизации")  
 plt.legend()  
 plt.grid()  
 plt.show()  
  
  
# Основной код  
if \_\_name\_\_ == "\_\_main\_\_":  
 x0 = [1, 1] # Начальная точка  
 lamb0 = 0.1 # Начальный шаг  
 epsilon = 1e-6 # Точность  
  
 print(f"Начальная точка: {x0}")  
 print(f"Начальное значение функции: {f(x0)}")  
  
 hook\_res = hook\_jiws(x0, lamb0, epsilon)  
 grad\_res = gradient\_const(x0, lamb0, epsilon)  
  
 print(  
 f"Метод Хука-Дживса: точка минимума = {hook\_res[0]}, значение функции = {hook\_res[1]}, итерации = {hook\_res[2]}")  
 print(  
 f"Градиентный спуск: точка минимума = {grad\_res[0]}, значение функции = {grad\_res[1]}, итерации = {grad\_res[2]}")  
  
 plot\_trajectories(hook\_res[3], grad\_res[3], func\_range=2)

1. **Результаты решения**

Оба метода тестировались на целевой функции:

f(x\_1, x\_2) = e^((x\_1)^2 + (x\_2)^2) + ln(4 + (x\_2)^2),

при начальных параметрах:

* начальная точка: x\_0 = [0, 0],
* начальный шаг: λ = 1,
* точность: ϵ = 10^(−6)

***Результаты метода Хука-Дживса***

1. **Точка минимума**: x = [0, 0].  
   Исходя из работы метода, текущая точка x\_0 = [0, 0] оказалась точкой минимума, так как дальнейшее уменьшение значений функции невозможно при уменьшении шага.
2. **Минимальное значение функции**: f(x) = e^(0^2 + 0^2) + ln(4 + 0^2) ≈ 2.386294361119891.
3. **Количество итераций**: метод завершил работу за 3 итерации.  
   На каждой итерации шаг λ уменьшался, так как не удавалось найти точку с меньшим значением функции.

#### **Результаты метода градиентного спуска**

1. **Точка минимума**: x = [0, 0].  
   Градиент в начальной точке x\_0 = [0, 0] равен нулю, что свидетельствует о том, что это стационарная точка.
2. **Минимальное значение функции**: f(x) = e^(0^2 + 0^2) + ln(4 + 0^2) ≈ 2.386294361119891.
3. **Количество итераций**: Метод завершил работу за 1 итерацию.  
   Это связано с тем, что начальная точка [0, 0] сразу оказалась минимумом, так как градиент функции в этой точке равен нулю.
4. **Вывод**
5. Оба метода успешно нашли минимум функции с заданной точностью ϵ = 10^(-6).
6. **Метод градиентного спуска** оказался значительно быстрее, так как для гладкой функции он сразу нашёл стационарную точку в начальной позиции.
7. **Метод Хука-Дживса** потребовал большего количества итераций из-за своей эвристической природы и необходимости уменьшать шаг λ для достижения точности.

**Общий вывод**:  
Для данной функции f(x\_1, x\_2), которая является гладкой и имеет легко вычисляемый градиент, **градиентный спуск** показал себя более эффективным методом. Однако **метод Хука-Дживса** также продемонстрировал свою универсальность и стабильность, успешно найдя минимум без использования производных.