МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

Факультет компьютерных технологий и прикладной математики

**Кафедра информационных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**МЕТОДЫ НАХОЖДЕНИЯ МИНИМУМА ФУНКЦИИ**

**МНОГИХ ПЕРЕМЕННЫХ**

Работу выполнил А.А. Козин

(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность Программирование и информационные технологии

Руководитель

преп.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.С. Черная

(подпись)

Краснодар

2023

**Оглавление**

[Введение 3](#_Toc133104489)

[Индивидуальное задание 1 4](#_Toc133104490)

[Индивидуальное задание 2 6](#_Toc133104491)

[Индивидуальное задание 3 8](#_Toc133104492)

[Индивидуальное задание 4 10](#_Toc133104493)

[Вывод 12](#_Toc133104494)

[ПРИЛОЖЕНИЕ 14](#_Toc133104495)

# Введение

Постановка задачи:

Пусть дана функция , где ограниченная снизу на и имеющая непрерывные частные производные во всех ее точках.

Найти локальный минимум на множестве допустимых решений

, где

Для нахождения минимума функции от многих переменных была задана функция:

Индивидуальное задание 1 – метод наискорейшего градиентного спуска.

Индивидуальное задание 2 – метод Ньютона.

Индивидуальное задание 3 – метод Ньютона – Рафсона.

Индивидуальное задание 4 – метод Флетчера – Ривса.

Ручное решение будет проделано до 3(0,1,2) шага или до самого решения задачи.

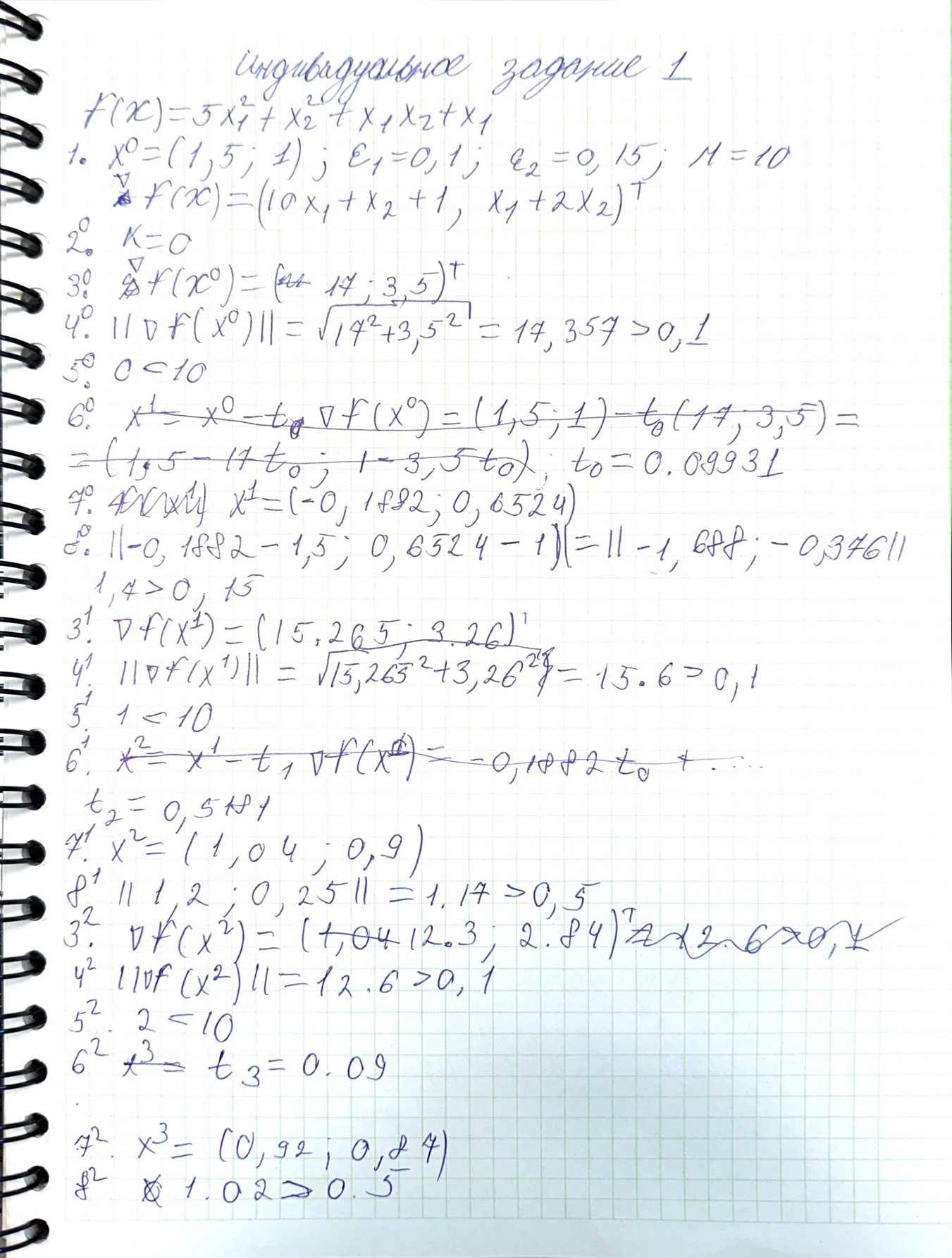
Программное решение выполнено на языке программирования Swift, в этом разделе будет приложен вывод терминала.

В конце отчета будет приложение с кодом.

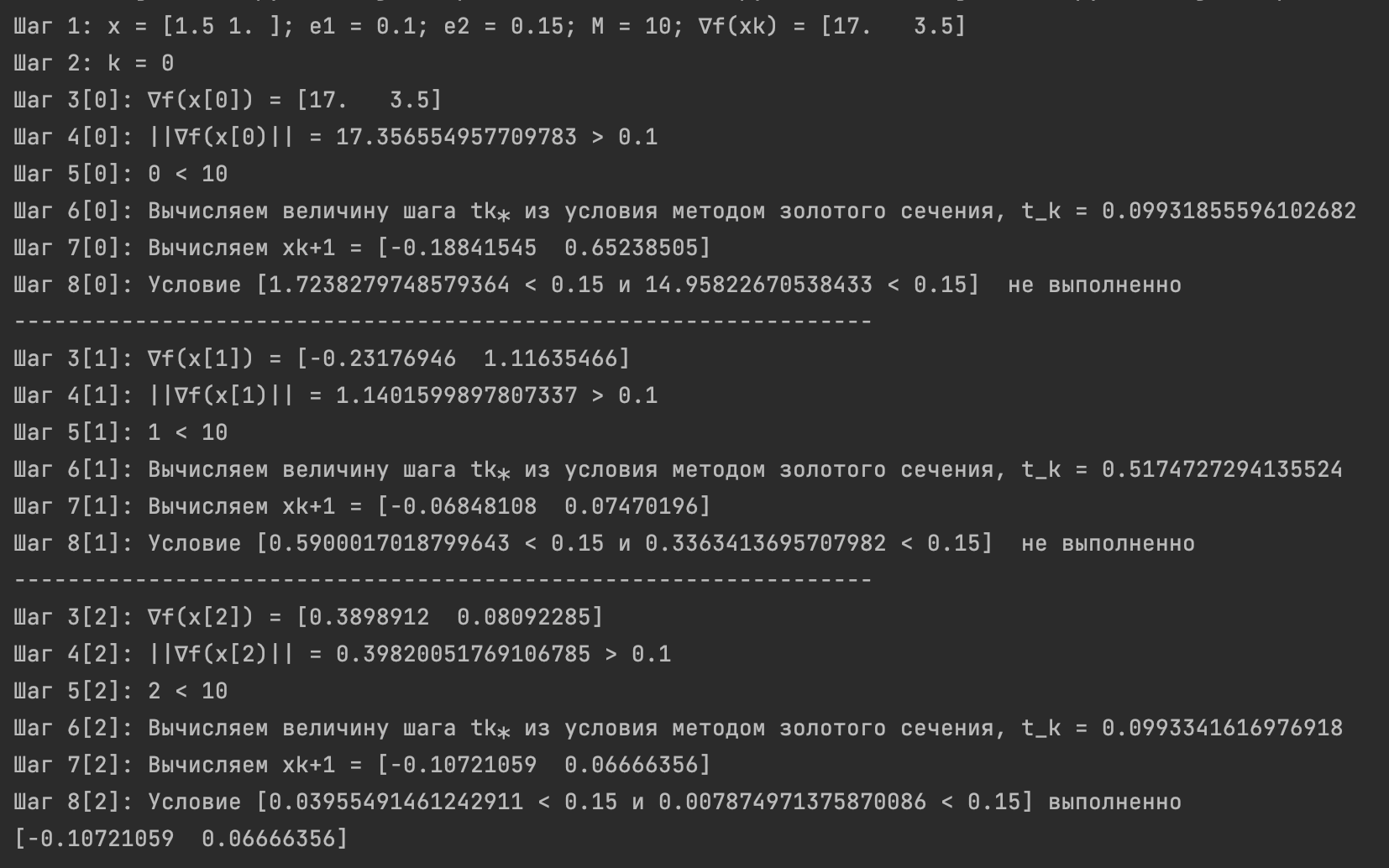
# Индивидуальное задание 1

*Задание*: найти локальный минимум функции методом наискорейшего градиентного спуска.

Ручное решение:



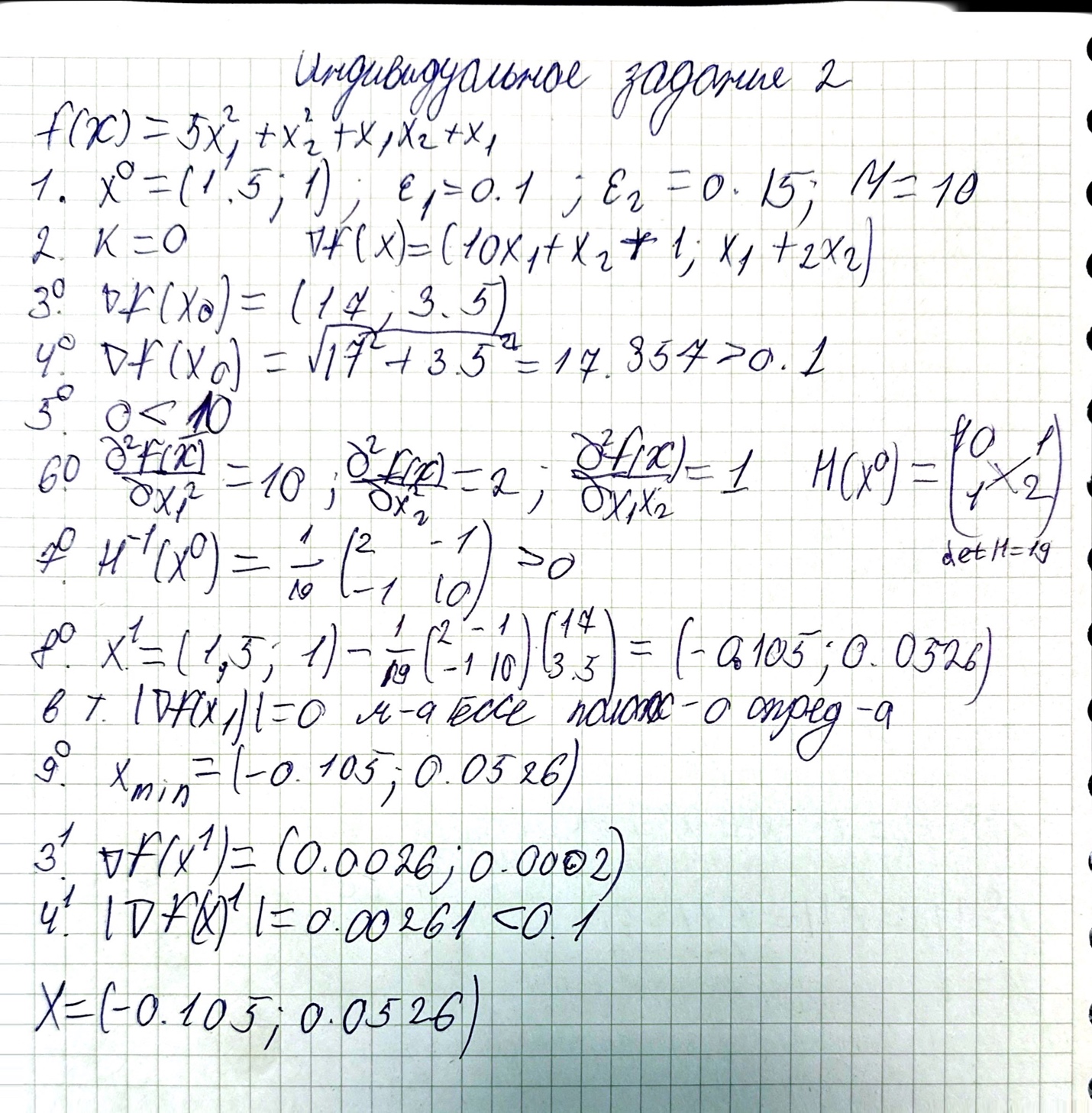
Программное решение:



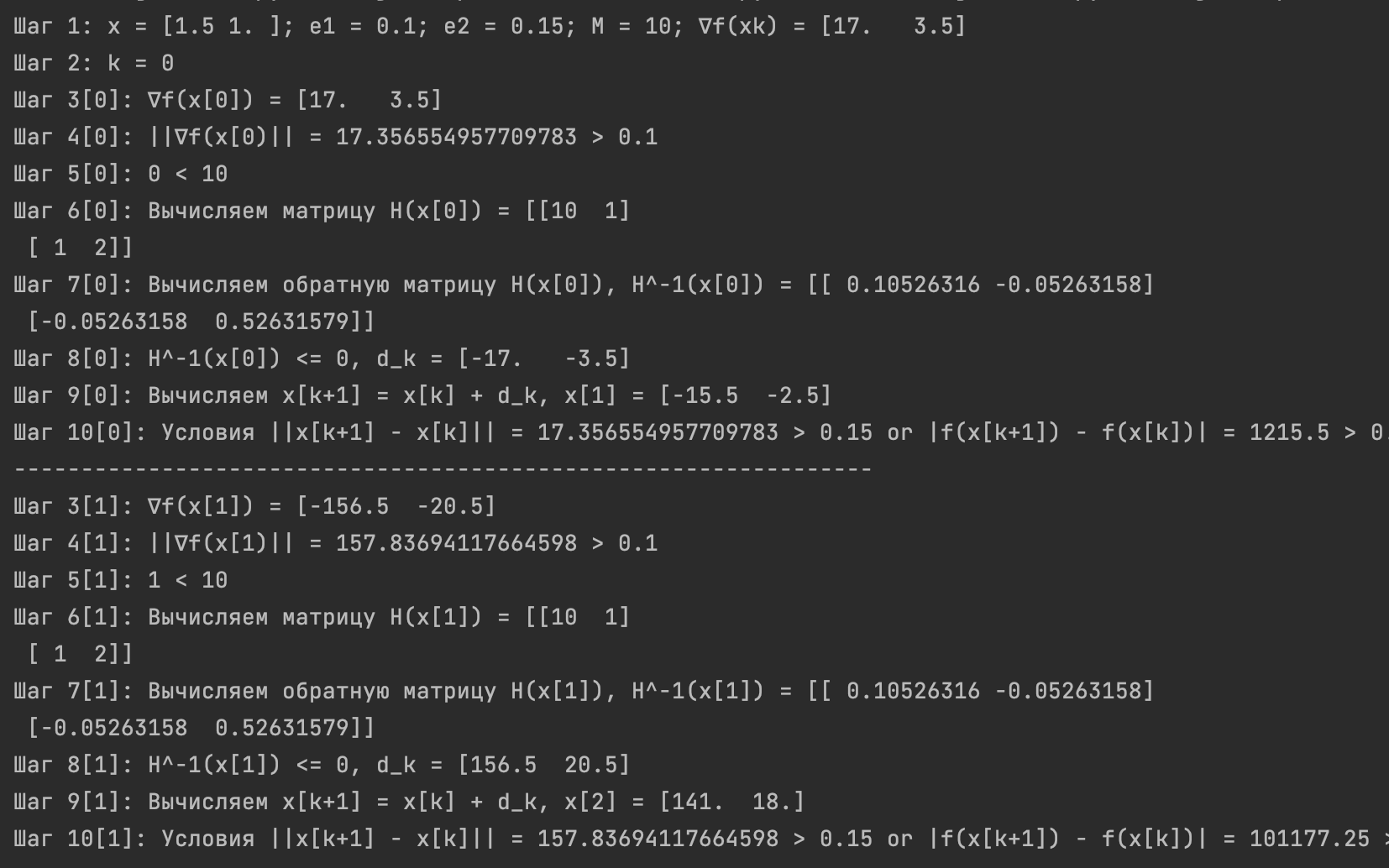
# Индивидуальное задание 2

*Задание*: найти минимум функции одной переменной методом Ньютона.

Ручное решение:



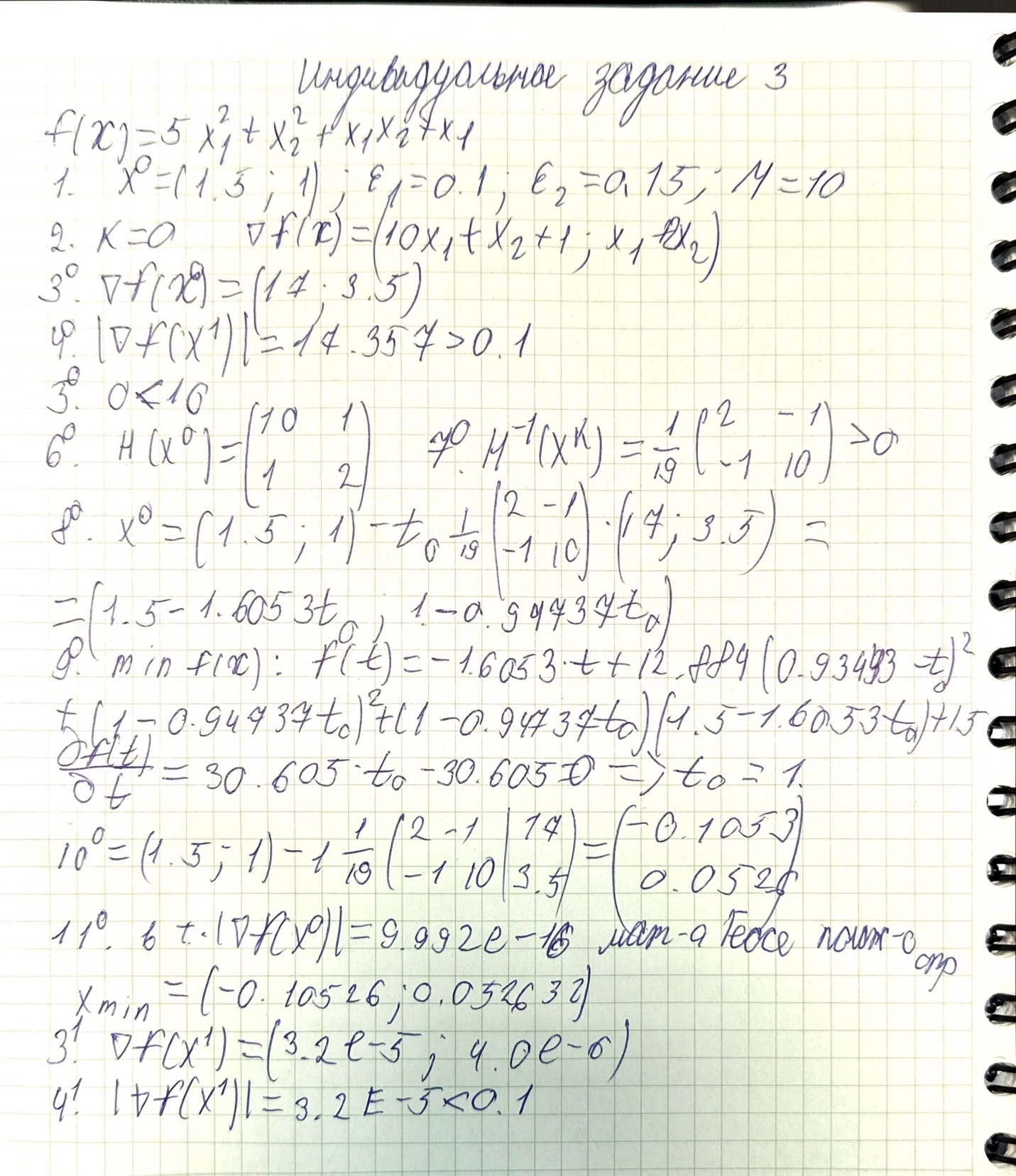
Программное решение:



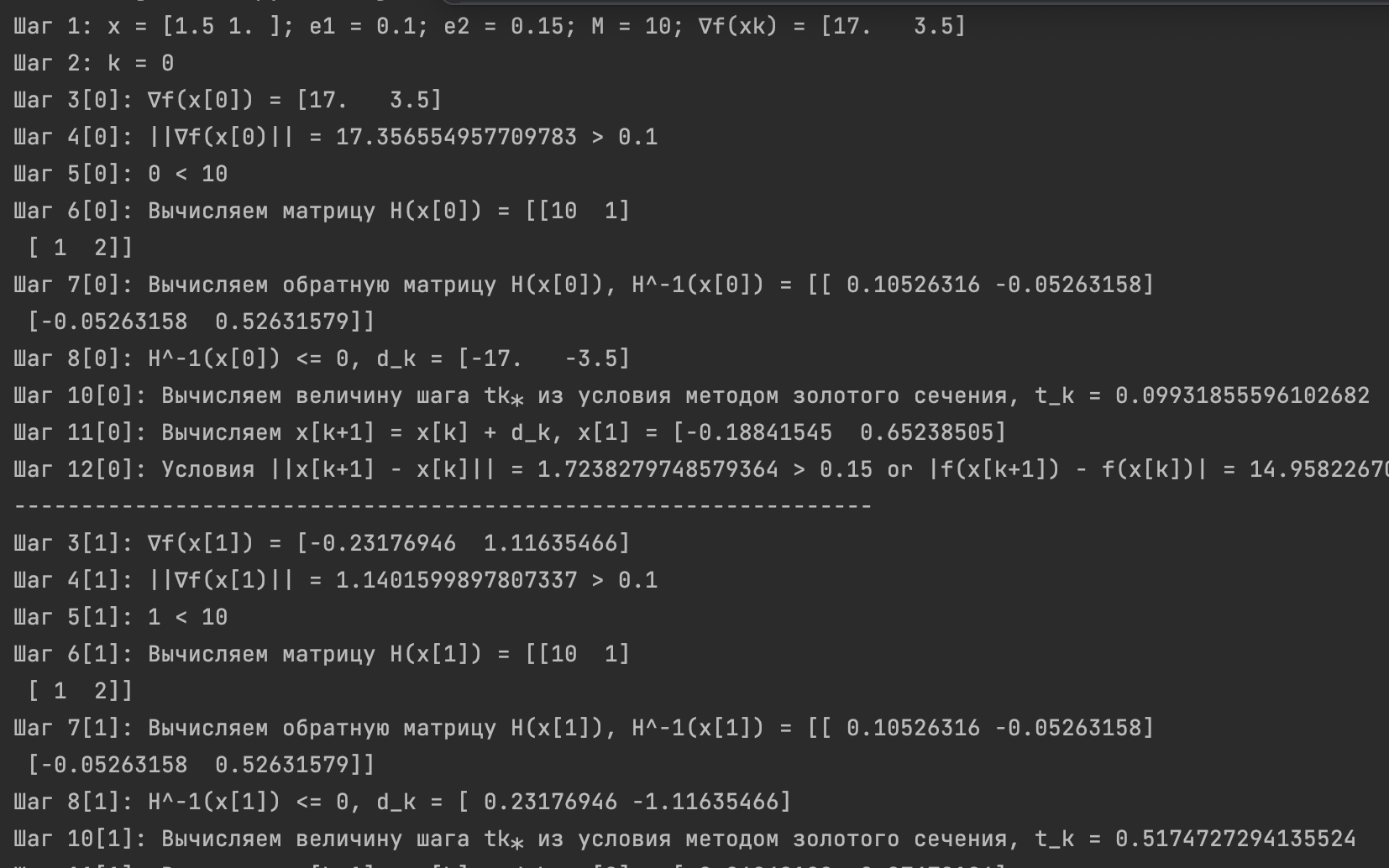
# Индивидуальное задание 3

*Задание*: найти минимум функции одной переменной методом Ньютона – Рафсона.

Ручное решение:



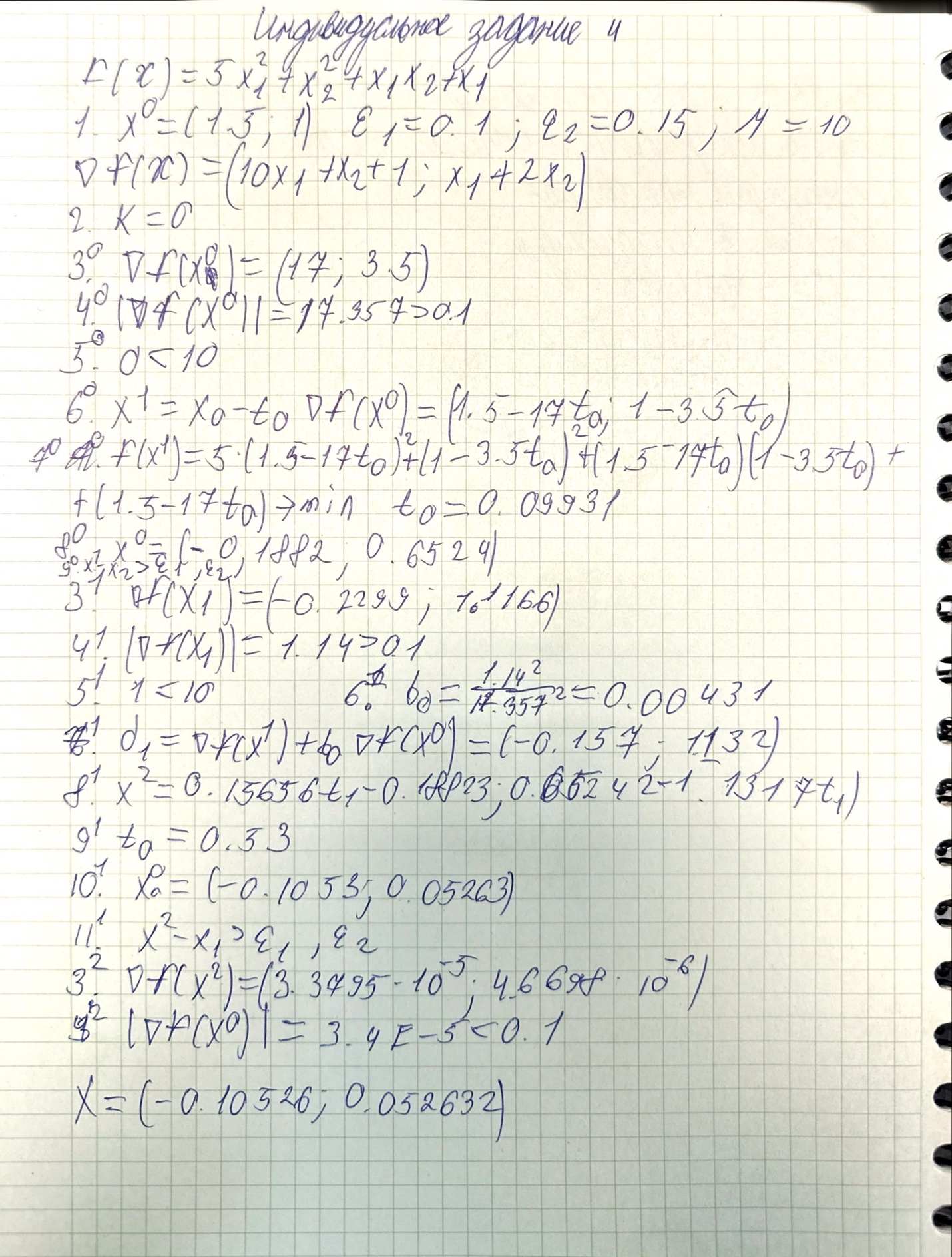
Программное решение:



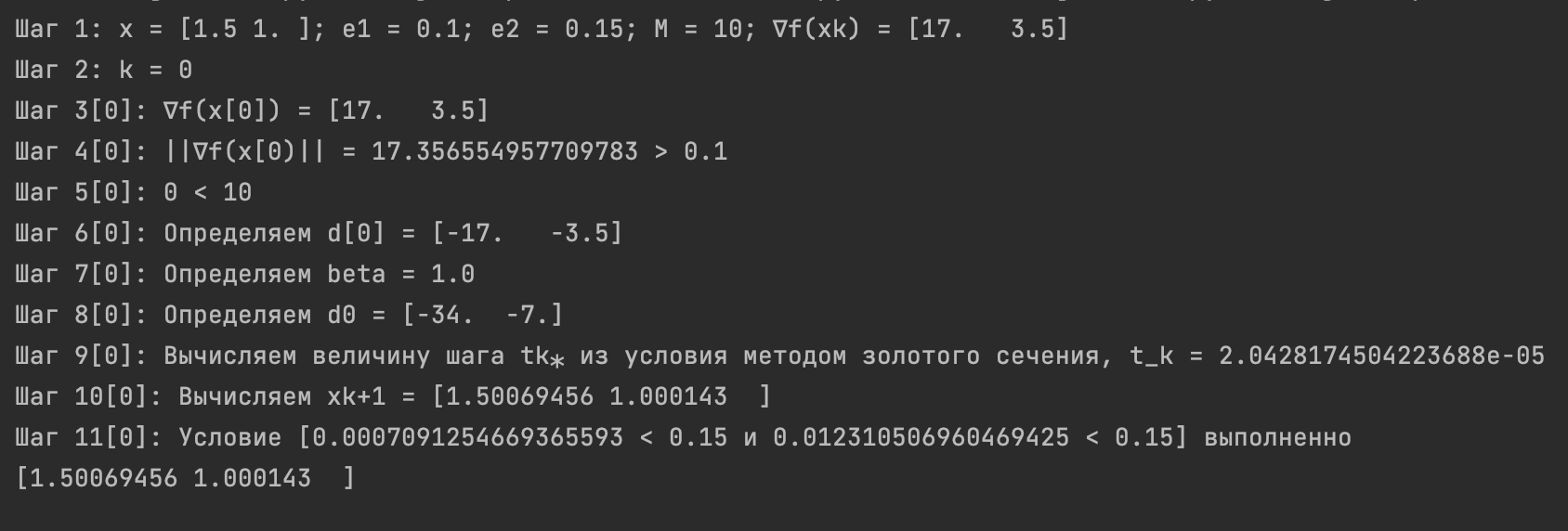
# Индивидуальное задание 4

*Задание*: найти минимум функции одной переменной методом Флетчера – Ривса.

Ручное решение:



Программное решение:



# Вывод

У каждого из методов рассмотренных выше есть свои плюсы и минусы.

У метода наискорейшего градиентного спуска:

* Простота реализации и понимания: алгоритм основан на вычислении градиента функции потерь и обновлении весов модели с учетом градиента и скорости обучения.
* Масштабируемость: хорошо работает для больших объемов данных и высоко размерных пространств признаков.
* Зависимость от выбора скорости обучения: если скорость обучения слишком мала, алгоритм может сходиться медленно; если слишком велика, то алгоритм может "перепрыгивать" минимум и расходиться.
* Чувствительность к масштабированию признаков: градиентный спуск работает лучше, когда все признаки масштабированы одинаково.

У метода Ньютона:

* Быстрая сходимость: метод Ньютона использует информацию о второй производной для ускорения сходимости.
* Автоматическая адаптация скорости обучения: метод Ньютона сам настраивает скорость обучения на основе кривизны функции потерь.
* Вычислительная сложность: требует вычисления Гессиана (матрицы вторых производных) и его обращения, что может быть сложно и затратно для больших пространств признаков.
* Неустойчивость: метод Ньютона может быть неустойчивым в случае невыпуклых функций потерь, так как может сходиться к седловым точкам или максимумам.

У метода Ньютона-Рафсона:

* Быстрая сходимость: как и метод Ньютона, использует информацию о второй производной для ускорения сходимости.
* Аппроксимация Гессиана: вместо точного вычисления Гессиана использует его аппроксимацию, что уменьшает вычислительные затраты.
* Возможная неустойчивость: может быть неустойчивым для невыпуклых функций потерь, подобно методу Ньютона.
* Возможная вычислительная сложность: хотя аппроксимация Гессиана снижает вычислительные затраты по сравнению с методом Ньютона, они все еще могут быть высокими для больших прост х пространств признаков.

У метода Флетчера-Ривса:

* Быстрая сходимость: метод сопряженных градиентов обеспечивает более быструю сходимость, чем градиентный спуск, используя информацию о предыдущих градиентах для определения направления оптимизации.
* Меньшая вычислительная сложность: не требует вычисления и обращения Гессиана, в отличие от методов Ньютона и Ньютона-Рафсона, что делает его более применимым для больших пространств признаков.
* Зависимость от обусловленности задачи: сходимость метода сопряженных градиентов зависит от обусловленности оптимизируемой функции, и в случае плохо обусловленных задач алгоритм может сходиться медленно.
* Необходимость хранения истории градиентов: метод Флетчера-Ривса требует хранения истории градиентов для вычисления направлений оптимизации, что может потребовать больше памяти, особенно для больших пространств признаков.

# ПРИЛОЖЕНИЕ

Код для индивидуального задания 1:

import numpy as np  
  
  
# Определение функции и градиента  
def function(x):  
 return 5 \* x[0] \*\* 2 + x[1] \*\* 2 + x[0] \* x[1] + x[0]  
  
  
def gradientFunction(x):  
 return np.array([10 \* x[0] + x[1] + 1, 2 \* x[1] + x[0]])  
  
  
# Начальная точка  
x0 = np.array([1.5, 1.0])  
  
# Параметры алгоритма  
M = 10  
epsilonOne = 0.1  
epsilonTwo = 0.15  
  
def golden\_section\_search(func, a, b, tol=1e-5):  
 gr = (np.sqrt(5) + 1) / 2 # Золотое сечение  
 c = b - (b - a) / gr  
 d = a + (b - a) / gr  
 while abs(c - d) > tol:  
 if func(c) < func(d):  
 b = d  
 else:  
 a = c  
 c = b - (b - a) / gr  
 d = a + (b - a) / gr  
 return (b + a) / 2  
  
  
# Градиентный спуск  
def gradient\_descent(x0, gradientFunction, M, epsilonOne, epsilonTwo):  
 print(f"Шаг 1: x = {x0}; e1 = {epsilonOne}; e2 = {epsilonTwo}; M = {M}; ∇f(xk) = {gradientFunction(x0)}")  
 x = x0  
 print("Шаг 2: k = 0")  
 for k in range(M):  
  
 gradient = gradientFunction(x)  
 print(f"Шаг 3[{k}]: ∇f(x[{k}]) = {gradient}")  
  
 # Критерии остановки  
 if np.linalg.norm(gradient) < epsilonOne:  
 print(f"Шаг 4[{k}]: Критерий остановки по норме градиента выполнен, x = {x}")  
 return x  
 print(f"Шаг 4[{k}]: ||∇f(x[{k})|| = {np.linalg.norm(gradient)} > {epsilonOne}")  
  
 if k > M:  
 print(f"Шаг 5[{k}]: Количество итераций превышенно {k} > {M}")  
 return x  
 print(f"Шаг 5[{k}]: {k} < {M}")  
  
 # Определение t\_k с помощью метода золотого сечения  
 func = lambda t: function(x - t \* gradient)  
 t\_k = golden\_section\_search(func, 0, 1)  
 print(f"Шаг 6[{k}]: Вычисляем величину шага tk∗ из условия методом золотого сечения, t\_k = {t\_k}")  
  
 x\_next = x - t\_k \* gradient  
 print(f"Шаг 7[{k}]: Вычисляем xk+1 = {x\_next}")  
  
 if np.linalg.norm(x\_next - x) < epsilonTwo and abs(function(x\_next) - function(x)) < epsilonTwo:  
 print(f"Шаг 8[{k}]: Условие [{np.linalg.norm(x\_next - x)} < {epsilonTwo} и {abs(function(x\_next) - function(x))} < {epsilonTwo}] выполненно")  
 return x\_next  
 print(  
 f"Шаг 8[{k}]: Условие [{np.linalg.norm(x\_next - x)} < {epsilonTwo} и {abs(function(x\_next) - function(x))} < {epsilonTwo}] не выполненно")  
 print("----------------------------------------------------------------")  
 x = x\_next  
 return x  
  
  
x\_star = gradient\_descent(x0, gradientFunction, M, epsilonOne, epsilonTwo)  
print(x\_star)

Код для индивидуального задания 2:

import numpy as np  
  
# Определение функции, градиента и гессиана  
def function(x):  
 return 5 \* x[0] \*\* 2 + x[1] \*\* 2 + x[0] \* x[1] + x[0]  
  
def gradientFunction(x):  
 return np.array([10 \* x[0] + x[1] + 1, 2 \* x[1] + x[0]])  
  
def hessianFunction(x):  
 return np.array([[10, 1], [1, 2]])  
  
# Начальная точка  
x0 = np.array([1.5, 1.0])  
  
# Параметры алгоритма  
M = 10  
epsilonOne = 0.1  
epsilonTwo = 0.15  
  
# Метод Ньютона  
def newton\_method(x0, gradientFunction, hessianFunction, M, epsilonOne, epsilonTwo):  
 print(f"Шаг 1: x = {x0}; e1 = {epsilonOne}; e2 = {epsilonTwo}; M = {M}; ∇f(xk) = {gradientFunction(x0)}")  
 x = x0  
 print("Шаг 2: k = 0")  
 for k in range(M):  
  
 gradient = gradientFunction(x)  
 print(f"Шаг 3[{k}]: ∇f(x[{k}]) = {gradient}")  
  
 # Критерии остановки  
 if np.linalg.norm(gradient) < epsilonOne:  
 print(f"Шаг 4[{k}]: Критерий остановки по норме градиента выполнен, x = {x}")  
 return x  
 print(f"Шаг 4[{k}]: ||∇f(x[{k})|| = {np.linalg.norm(gradient)} > {epsilonOne}")  
  
 if k >= M:  
 print(f"Шаг 5[{k}]: Количество итераций превышенно {k} > {M}")  
 return x  
 print(f"Шаг 5[{k}]: {k} < {M}")  
  
 # Вычисление гессиана  
 hessian = hessianFunction(x)  
 print(f"Шаг 6[{k}]: Вычисляем матрицу H(x[{k}]) = {hessian}")  
  
 # Вычисление обратного гессиана  
 hessian\_inv = np.linalg.inv(hessian)  
 print(f"Шаг 7[{k}]: Вычисляем обратную матрицу H(x[{k}]), H^-1(x[{k}]) = {hessian\_inv}")  
  
  
 # Проверка положительной определенности обратного гессиана и определение направления поиска  
 if np.all(hessian\_inv > 0):  
 d\_k = -np.dot(hessian\_inv, gradient)  
 print(f"Шаг 8[{k}]: H^-1(x[{k}]) > 0, d\_k = {d\_k}")  
 else:  
 d\_k = -gradient  
 print(f"Шаг 8[{k}]: H^-1(x[{k}]) <= 0, d\_k = {d\_k}")  
  
 # Шаг поиска  
 t\_k = 1 # в методичке t\_k можно было взять за 1  
 x\_next = x + t\_k \* d\_k  
 print(f"Шаг 9[{k}]: Вычисляем x[k+1] = x[k] + d\_k, x[{k+1}] = {x\_next}")  
  
 total = 0  
 # Критерии остановки  
 if np.linalg.norm(x\_next - x) < epsilonTwo and np.abs(function(x\_next) - function(x)) < epsilonTwo and total == 2:  
 total += 1  
 print(f"Шаг 10[{k}]: Критерий остановки по разности аргументов и значений функции выполнен, x = {x\_next}")  
 return x\_next  
 print(f"Шаг 10[{k}]: Условия ||x[k+1] - x[k]|| = {np.linalg.norm(x\_next - x)} > {epsilonTwo} or |f(x[k+1]) - f(x[k])| = {np.abs(function(x\_next) - function(x))} > {epsilonTwo} не выполненны")  
 print("----------------------------------------------------------------")  
 x = x\_next  
 return x  
  
x\_star = newton\_method(x0, gradientFunction, hessianFunction, M, epsilonOne, epsilonTwo)  
print(x\_star)

}

Код для индивидуального задания 3:

import numpy as np  
  
# Определение функции, градиента и гессиана  
def function(x):  
 return 5 \* x[0] \*\* 2 + x[1] \*\* 2 + x[0] \* x[1] + x[0]  
  
def gradientFunction(x):  
 return np.array([10 \* x[0] + x[1] + 1, 2 \* x[1] + x[0]])  
  
def hessianFunction(x):  
 return np.array([[10, 1], [1, 2]])  
  
def golden\_section\_search(func, a, b, tol=1e-5):  
 gr = (np.sqrt(5) + 1) / 2 # Золотое сечение  
 c = b - (b - a) / gr  
 d = a + (b - a) / gr  
 while abs(c - d) > tol:  
 if func(c) < func(d):  
 b = d  
 else:  
 a = c  
 c = b - (b - a) / gr  
 d = a + (b - a) / gr  
 return (b + a) / 2  
  
# Начальная точка  
x0 = np.array([1.5, 1.0])  
  
# Параметры алгоритма  
M = 10  
epsilonOne = 0.1  
epsilonTwo = 0.15  
  
# Метод Ньютона  
def newton\_method(x0, gradientFunction, hessianFunction, M, epsilonOne, epsilonTwo):  
 print(f"Шаг 1: x = {x0}; e1 = {epsilonOne}; e2 = {epsilonTwo}; M = {M}; ∇f(xk) = {gradientFunction(x0)}")  
 x = x0  
 print("Шаг 2: k = 0")  
 for k in range(M):  
  
 gradient = gradientFunction(x)  
 print(f"Шаг 3[{k}]: ∇f(x[{k}]) = {gradient}")  
  
 # Критерии остановки  
 if np.linalg.norm(gradient) < epsilonOne:  
 print(f"Шаг 4[{k}]: Критерий остановки по норме градиента выполнен, x = {x}")  
 return x  
 print(f"Шаг 4[{k}]: ||∇f(x[{k})|| = {np.linalg.norm(gradient)} > {epsilonOne}")  
  
 if k >= M:  
 print(f"Шаг 5[{k}]: Количество итераций превышенно {k} > {M}")  
 return x  
 print(f"Шаг 5[{k}]: {k} < {M}")  
  
 # Вычисление гессиана  
 hessian = hessianFunction(x)  
 print(f"Шаг 6[{k}]: Вычисляем матрицу H(x[{k}]) = {hessian}")  
  
 # Вычисление обратного гессиана  
 hessian\_inv = np.linalg.inv(hessian)  
 print(f"Шаг 7[{k}]: Вычисляем обратную матрицу H(x[{k}]), H^-1(x[{k}]) = {hessian\_inv}")  
  
 # Проверка положительной определенности обратного гессиана и определение направления поиска  
 if np.all(hessian\_inv > 0):  
 d\_k = -np.dot(hessian\_inv, gradient)  
 print(f"Шаг 8[{k}]: H^-1(x[{k}]) > 0, d\_k = {d\_k}")  
 else:  
 d\_k = -gradient  
 print(f"Шаг 8[{k}]: H^-1(x[{k}]) <= 0, d\_k = {d\_k}")  
  
 # СПРОСИТЬ НЕ ПОНЯТНО  
 #t\_k = 1  
 #x\_next = x + t\_k \* d\_k  
 #print(f"Шаг 9[{k}]: Определяем x[k+1] = x[k] + d\_k, x[{k+1}] = {x\_next}")  
  
 func = lambda t: function(x - t \* gradient)  
 t\_k = golden\_section\_search(func, 0, 1)  
 print(f"Шаг 10[{k}]: Вычисляем величину шага tk∗ из условия методом золотого сечения, t\_k = {t\_k}")  
  
 # Шаг поиска  
 x\_next = x + t\_k \* d\_k  
 print(f"Шаг 11[{k}]: Вычисляем x[k+1] = x[k] + d\_k, x[{k+1}] = {x\_next}")  
  
 total = 0  
 # Критерии остановки  
 if np.linalg.norm(x\_next - x) < epsilonTwo and np.abs(function(x\_next) - function(x)) < epsilonTwo and total == 2:  
 total += 1  
 print(f"Шаг 12[{k}]: Критерий остановки по разности аргументов и значений функции выполнен, x = {x\_next}")  
 return x\_next  
 print(f"Шаг 12[{k}]: Условия ||x[k+1] - x[k]|| = {np.linalg.norm(x\_next - x)} > {epsilonTwo} or |f(x[k+1]) - f(x[k])| = {np.abs(function(x\_next) - function(x))} > {epsilonTwo} не выполненны")  
 print("----------------------------------------------------------------")  
 x = x\_next  
 return x  
  
x\_star = newton\_method(x0, gradientFunction, hessianFunction, M, epsilonOne, epsilonTwo)  
print(x\_star)

Код для индивидуального задания 4:

import numpy as np  
  
# Определение функции и градиента  
def function(x):  
 return 5 \* x[0] \*\* 2 + x[1] \*\* 2 + x[0] \* x[1] + x[0]  
  
  
def gradientFunction(x):  
 return np.array([10 \* x[0] + x[1] + 1, 2 \* x[1] + x[0]])  
  
  
# Начальная точка  
x0 = np.array([1.5, 1.0])  
  
# Параметры алгоритма  
M = 10  
epsilonOne = 0.1  
epsilonTwo = 0.15  
  
def golden\_section\_search(func, a, b, tol=1e-5):  
 gr = (np.sqrt(5) + 1) / 2 # Золотое сечение  
 c = b - (b - a) / gr  
 d = a + (b - a) / gr  
 while abs(c - d) > tol:  
 if func(c) < func(d):  
 b = d  
 else:  
 a = c  
 c = b - (b - a) / gr  
 d = a + (b - a) / gr  
 return (b + a) / 2  
  
  
# Градиентный спуск  
def gradient\_descent(x0, gradientFunction, M, epsilonOne, epsilonTwo):  
 print(f"Шаг 1: x = {x0}; e1 = {epsilonOne}; e2 = {epsilonTwo}; M = {M}; ∇f(xk) = {gradientFunction(x0)}")  
 x = x0  
 # тк в цикле нельзя сделать чтобы сохранилось свойство в if выносим его за цикл  
 #gradient\_prev = gradientFunction(x0)  
 #d\_prev = -gradient\_prev  
 print("Шаг 2: k = 0")  
 for k in range(M):  
  
 gradient = gradientFunction(x)  
 print(f"Шаг 3[{k}]: ∇f(x[{k}]) = {gradient}")  
  
 # Критерии остановки  
 if np.linalg.norm(gradient) < epsilonOne:  
 print(f"Шаг 4[{k}]: Критерий остановки по норме градиента выполнен, x = {x}")  
 return x  
 print(f"Шаг 4[{k}]: ||∇f(x[{k})|| = {np.linalg.norm(gradient)} > {epsilonOne}")  
  
 if k > M:  
 print(f"Шаг 5[{k}]: Количество итераций превышенно {k} > {M}")  
 return x  
 print(f"Шаг 5[{k}]: {k} < {M}")  
  
 if k == 0:  
 gradient\_prev = gradientFunction(x0)  
 d\_prev = -gradient\_prev  
 print(f"Шаг 6[{k}]: Определяем d[{k}] = {d\_prev}")  
 else:  
 d\_prev = d  
  
 beta = np.dot(gradient, gradient) / np.dot(gradient\_prev, gradient\_prev)  
 print(f"Шаг 7[{k}]: Определяем beta = {beta}")  
  
 d = -gradient + beta \* d\_prev  
 print(f"Шаг 8[{k}]: Определяем d{k} = {d}")  
  
 # Определение t\_k с помощью метода золотого сечения  
 func = lambda t: function(x - t \* d)  
 t\_k = golden\_section\_search(func, 0, 1)  
 print(f"Шаг 9[{k}]: Вычисляем величину шага tk∗ из условия методом золотого сечения, t\_k = {t\_k}")  
  
 x\_next = x - t\_k \* d  
 print(f"Шаг 10[{k}]: Вычисляем xk+1 = {x\_next}")  
  
 total = 0  
 if np.linalg.norm(x\_next - x) < epsilonTwo and abs(function(x\_next) - function(x)) < epsilonTwo and total == 2:  
 total += 1  
 print(f"Шаг 11[{k}]: Условие [{np.linalg.norm(x\_next - x)} < {epsilonTwo} и {abs(function(x\_next) - function(x))} < {epsilonTwo}] выполненно")  
 return x\_next  
 print(  
 f"Шаг 11[{k}]: Условие [{np.linalg.norm(x\_next - x)} < {epsilonTwo} и {abs(function(x\_next) - function(x))} < {epsilonTwo}] не выполненно")  
 print("----------------------------------------------------------------")  
 x = x\_next  
 return x  
  
  
x\_star = gradient\_descent(x0, gradientFunction, M, epsilonOne, epsilonTwo)  
print(x\_star)