

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

|  |  |
| --- | --- |
| **Институт**  **информационных**  **технологий** | **Кафедра**  **информационных технологий и вычислительных систем** |

ОТЧЕТ О ВЫПОЛНЕНИИ   
ИНДИВИДУАЛЬНОГО ЗАДАНИЯ ПО ДИСЦИПЛИНЕ   
«Методы оптимизации»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| СТУДЕНТА | *2* | КУРСА | | *бакалавриата* | ГРУППЫ | *ИДБ-23-16* |
|  | | | *(уровень профессионального образования)* | |  | |

|  |
| --- |
| **Грачева Николай Романовича** |
| *(ФИО)* |

НА ТЕМУ

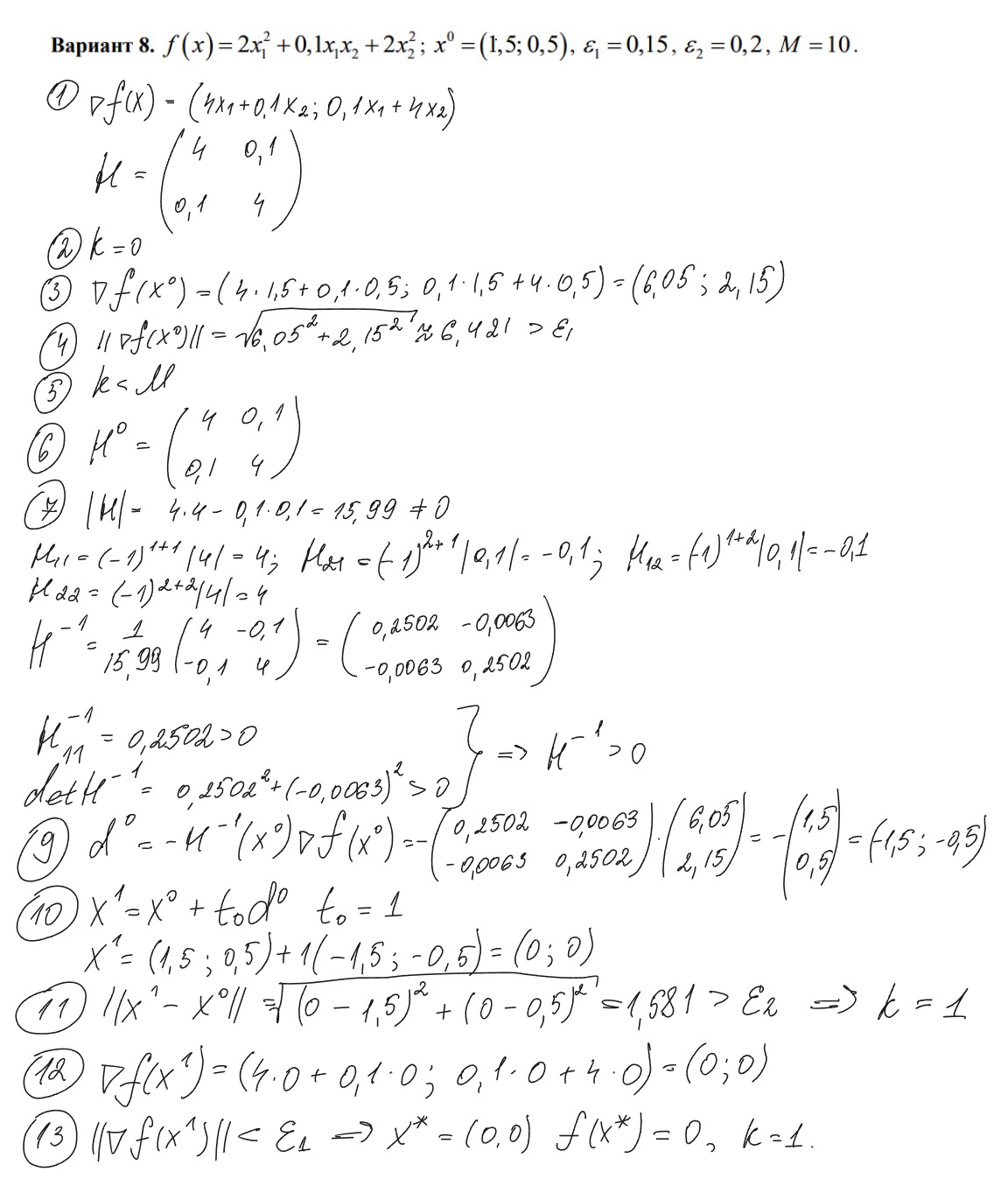
|  |
| --- |
| Лабораторная работа №3 «Методы второго порядка. Метод Ньютона.» Вариант № 8 |

|  |  |
| --- | --- |
| Направление: | 09.03.04 Программная инженерия |
| Профиль подготовки: | Системный анализ и проектирование программных комплексов |

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет сдан «20» мая 2025 г. | | | |
|  |  |  |  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
|  | | | |
| Преподаватель | Волкова О. Р., к.т.н. |  |  |
|  | *(Ф.И.О., должность, степень, звание.)* |  | *(подпись)* |

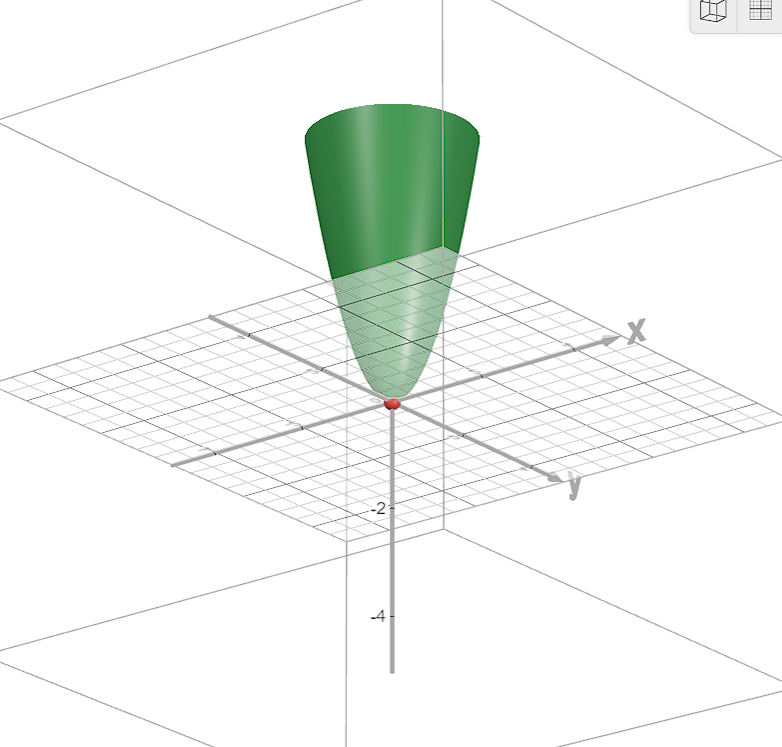
МОСКВА 2025

1. Найти локальный минимум функции из индивидуального задания методом Ньютона.



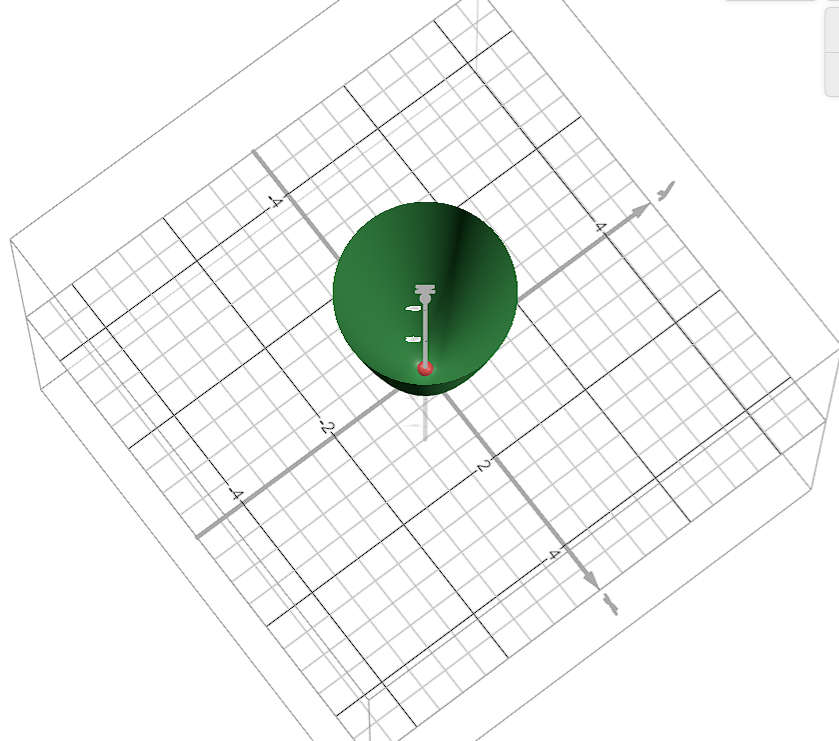
1. Построить график функции из индивидуального задания, отметить на графике найденный минимум.

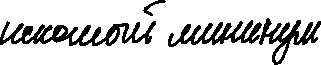
Вид снизу:

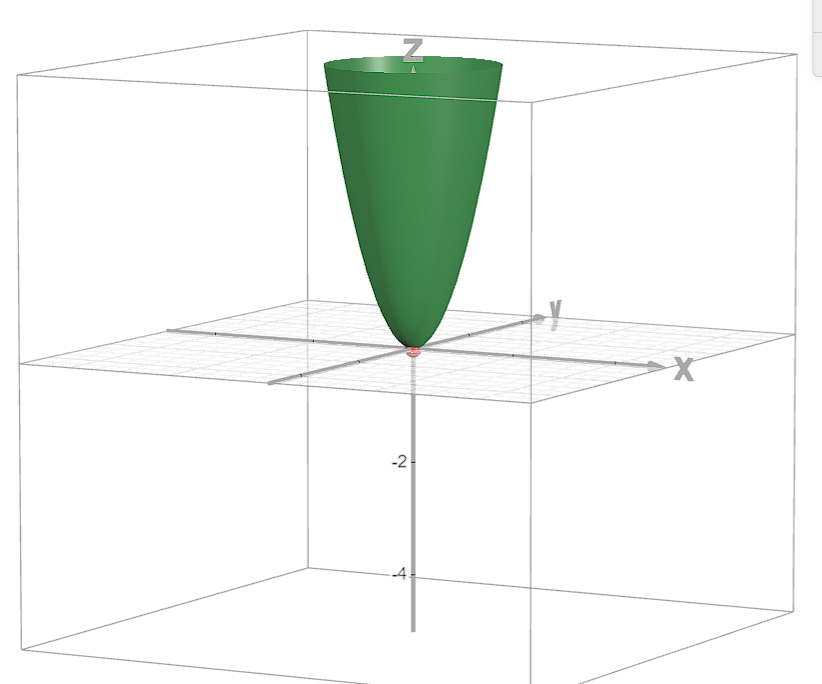




Вид сверху:

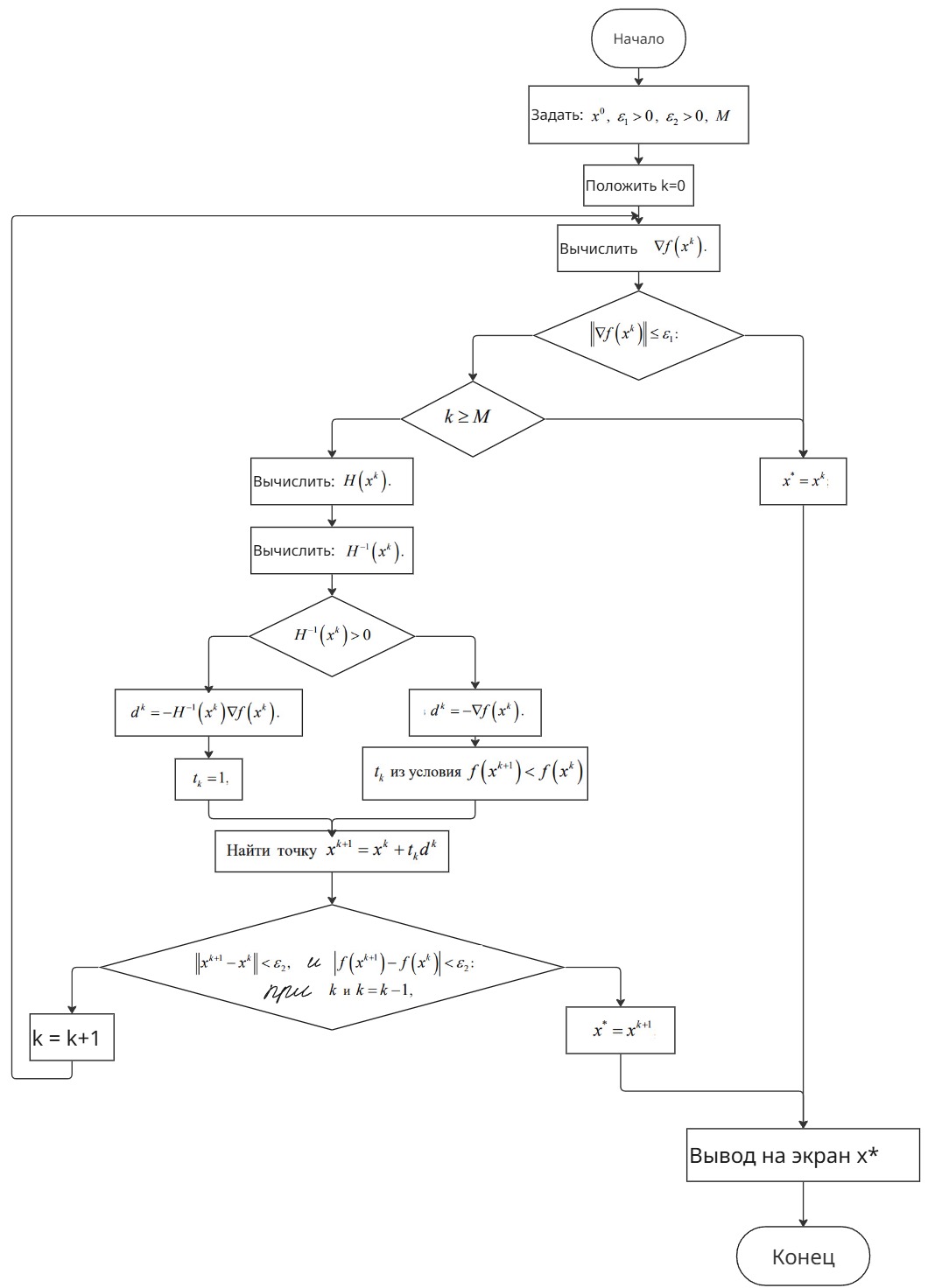




Вид сбоку:  




1. Построить блок-схему алгоритма поиска локального минимума функции для метода Ньютона.

  
4\*. Разработать веб приложение для решения задачи индивидуального задания методом Ньютона.

**import** numpy **as** np

**from** sympy **import** symbols**,** diff**,** hessian**,** lambdify

**from** string **import** ascii\_letters

class ExtremumFinder**:**

**def** \_\_init\_\_(*self*)**:**

*self***.**func **=** None

**def** SetFunc(*self***,** **func**)**:**

        variable **=** list()

**if** '=' **in** func**:**

            func **=** func[func**.**index('=') **+** 1**:**]

        func **=** func**.**replace("^"**,** "\*\*")

        func **=** func**.**replace("e"**,** "2.71828182845904")

*self***.**func **=** func

**for** letter **in** ascii\_letters**:**

**if** letter **in** func**:**

                variable**.**append(letter)

**if** len(variable) **==** 2**:**

**break**

*self***.**func **=** *self***.**func**.**replace(variable[0]**,** "(x)")

*self***.**func **=** *self***.**func**.**replace(variable[1]**,** "(y)")

**def** nyuton(*self***,** **x,** **e1,** **e2,** **M**)**:**

        x\_sym**,** y\_sym **=** symbols('x y')

        f\_expr **=** eval(*self***.**func**,** {'x'**:** x\_sym**,** 'y'**:** y\_sym})

        gradient **=** [diff(f\_expr**,** var) **for** var **in** (x\_sym**,** y\_sym)]

        grad\_func **=** lambdify((x\_sym**,** y\_sym)**,** gradient**,** 'numpy')

        hessian\_matrix **=** hessian(f\_expr**,** (x\_sym**,** y\_sym))

        hessian\_func **=** lambdify((x\_sym**,** y\_sym)**,** hessian\_matrix**,** 'numpy')

        k **=** 0

        x\_k **=** np**.**array(x**,** **dtype=**float)

**while** True**:**

            grad **=** np**.**array(grad\_func(x\_k[0]**,** x\_k[1])**,** **dtype=**float)

**if** np**.**linalg**.**norm(grad) **<=** e1**:**

                x **=** x\_k1**.**tolist()

**return** round(x[0]**,** 5)**,** round(x[1]**,** 5)**,** round(float(eval(*self***.**func**,** {'x'**:** x[0]**,** 'y'**:** x[1]}))**,** 5)**,** k

**if** k **>=** M**:**

                x **=** x\_k1**.**tolist()

**return** round(x[0]**,** 5)**,** round(x[1]**,** 5)**,** round(float(eval(*self***.**func**,** {'x'**:** x[0]**,** 'y'**:** x[1]}))**,** 5)**,** k

            H **=** np**.**array(hessian\_func(x\_k[0]**,** x\_k[1])**,** **dtype=**float)

            H\_inv **=** np**.**linalg**.**inv(H)

**if** np**.**all(np**.**linalg**.**eigvals(H\_inv) **>** 0)**:**

                d\_k **=** **-**H\_inv **@** grad

**else:**

                d\_k **=** **-**grad

**if** np**.**array\_equal(d\_k**,** **-**H\_inv **@** grad)**:**

                x\_k1 **=** x\_k **+** d\_k

**else:**

                t **=** 1.0

**while** True**:**

                    x\_k1 **=** x\_k **+** t **\*** d\_k

                    f\_k **=** float(eval(*self***.**func**,** {'x'**:** x\_k[0]**,** 'y'**:** x\_k[1]}))

                    f\_k1 **=** float(eval(*self***.**func**,** {'x'**:** x\_k1[0]**,** 'y'**:** x\_k1[1]}))

**if** f\_k1 **<** f\_k**:**

**break**

                    t **\*=** 0.5

**if** np**.**linalg**.**norm(x\_k1 **-** x\_k) **<** e2 **and** abs(float(eval(*self***.**func**,** {'x'**:** x\_k1[0]**,** 'y'**:** x\_k1[1]})) **-** abs(float(eval(*self***.**func**,** {'x'**:** x\_k[0]**,** 'y'**:** x\_k[1]}))) **<** e2)**:**

                x **=** x\_k1**.**tolist()

**return** round(x[0]**,** 5)**,** round(x[1]**,** 5)**,** round(float(eval(*self***.**func**,** {'x'**:** x[0]**,** 'y'**:** x[1]}))**,** 5)**,** k

            x\_k **=** x\_k1

            k **+=** 1

**def** test()**:**

    finder **=** ExtremumFinder()

    finder**.**SetFunc("2 \* x \*\* 2 + 0.1 \* x \* y + 2 \* y \*\* 2")

    x **=** [1.5**,** 0.5]

    e1 **=** 0.0001

    e2 **=** 0.0001

    M **=** 1000

    result **=** finder**.**nyuton(x**,** e1**,** e2**,** M)

    print(result)

*# test()*

Скриншоты

