

**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

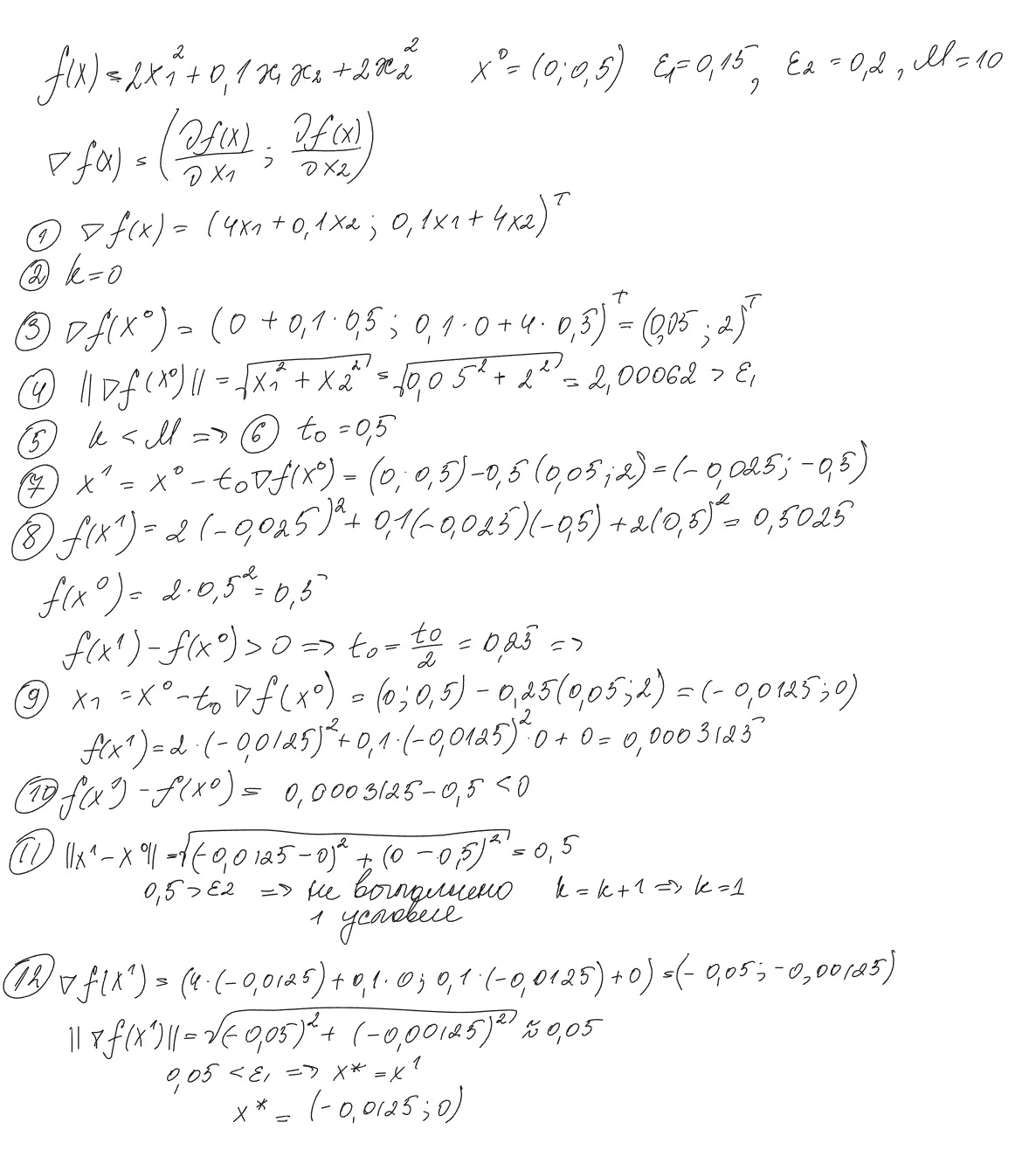
|  |
| --- |
| Лабораторная работа №2 «Численные методы многомерной оптимизации (методы первого порядка)» Вариант № 8 |

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

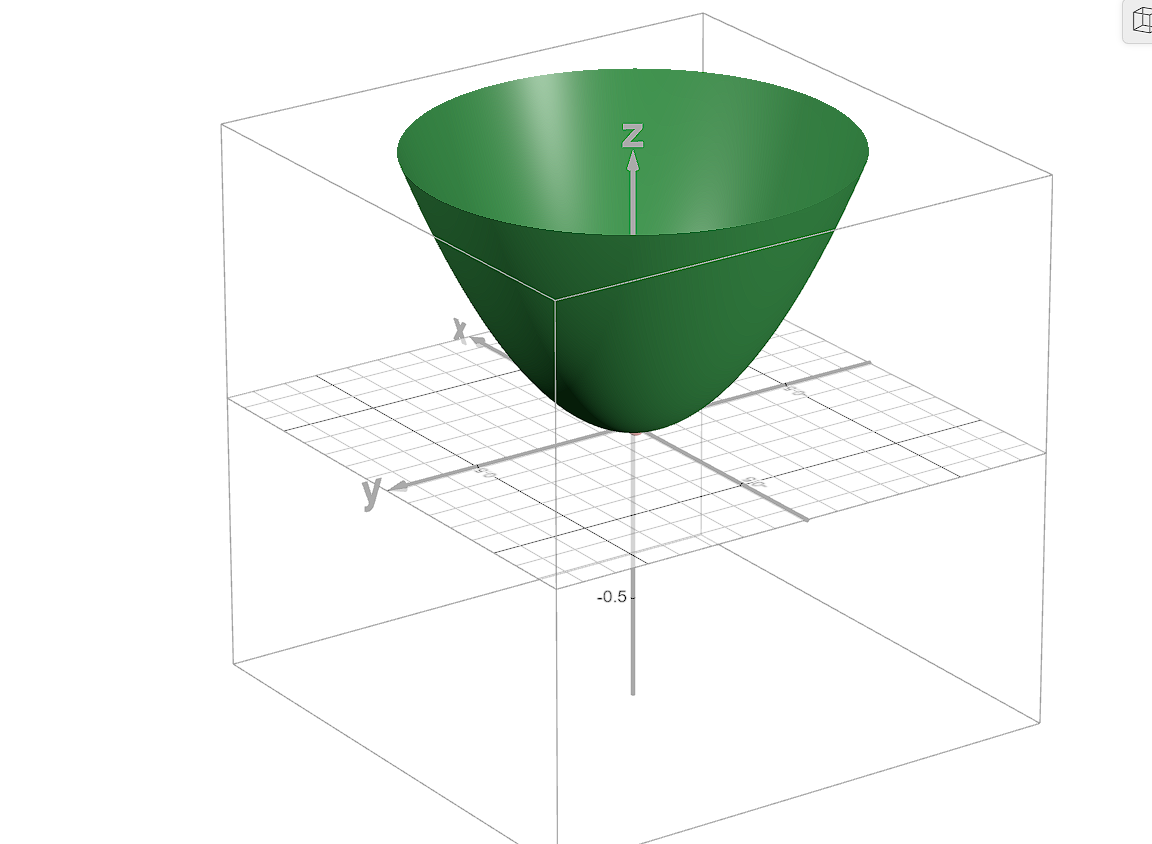
|  |  |  |  |
| --- | --- | --- | --- |
| Отчет сдан «20» мая 2025 г. | | | |
|  |  |  |  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
|  | | | |
| Преподаватель | Волкова О. Р., к.т.н. |  |  |
|  | *(Ф.И.О., должность, степень, звание.)* |  | *(подпись)* |

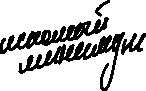
МОСКВА 2025

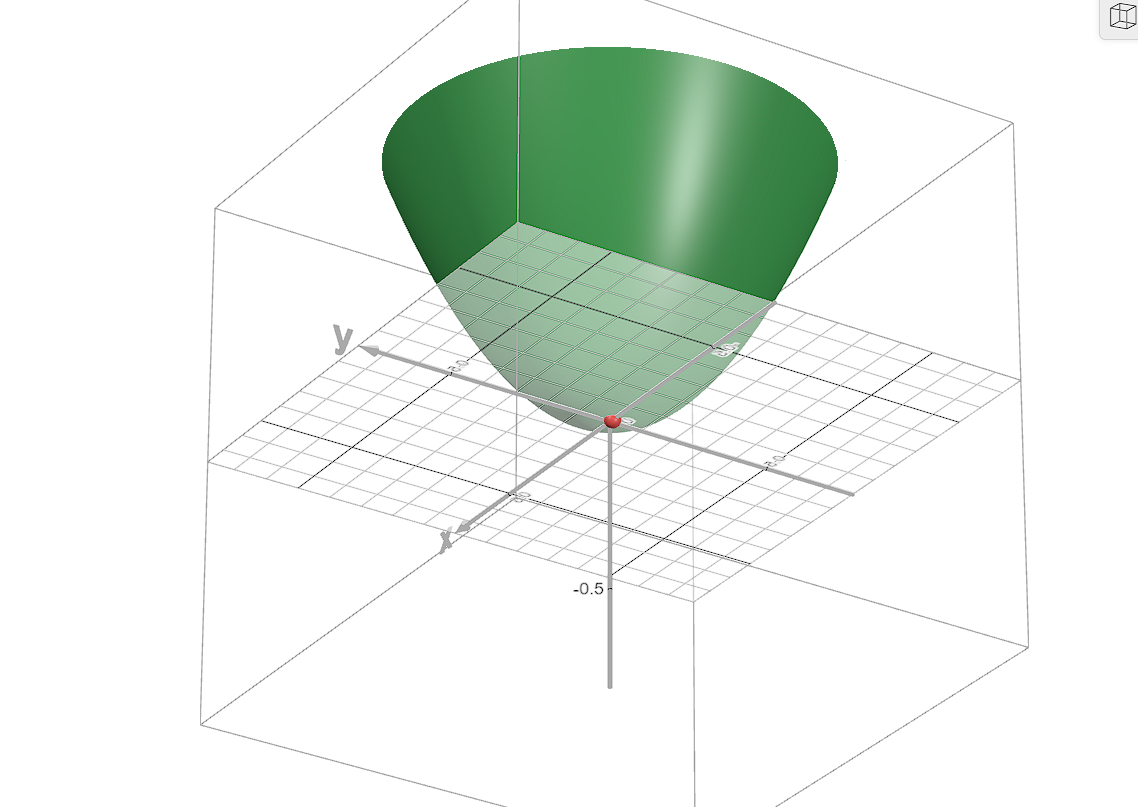
1. Найти минимум функции из индивидуального задания методом градиентного спуска с постоянным шагом.

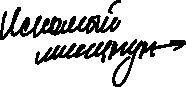


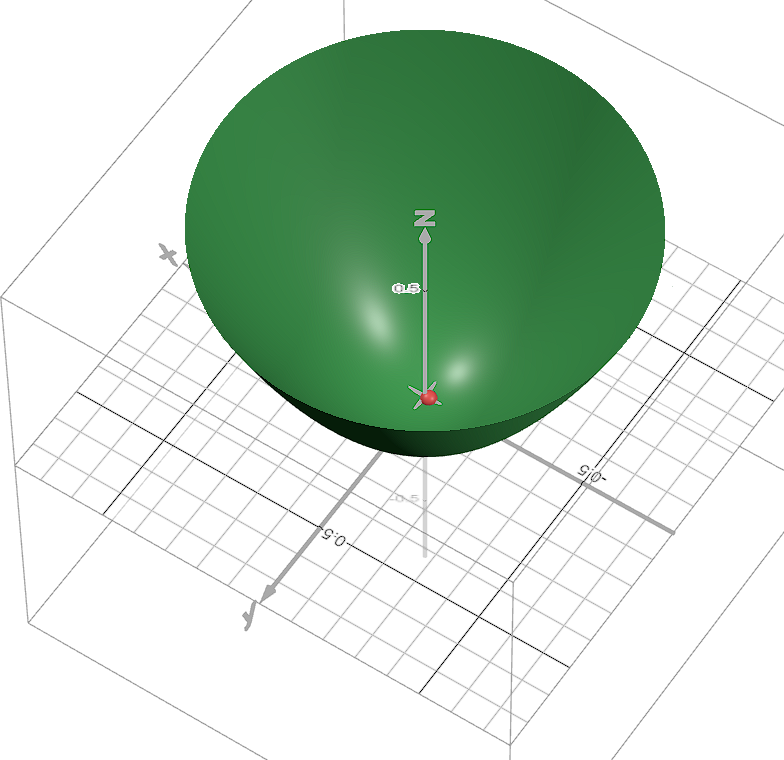
1. Построить график функции из индивидуального задания, отметить на графике найденный минимум.

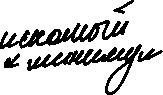




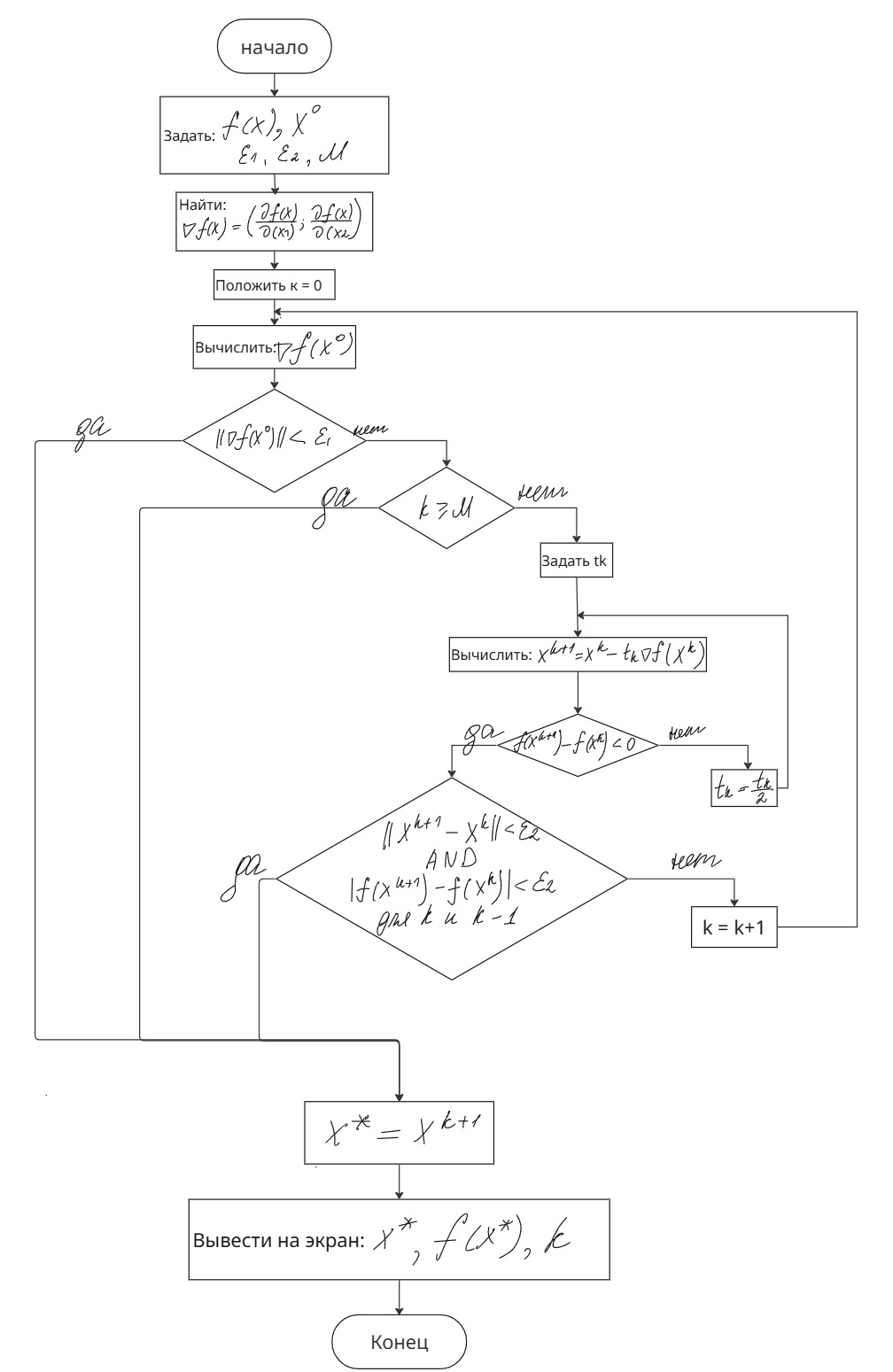








1. Построить блок-схему алгоритма поиска экстремума по методу градиентного спуска с постоянным шагом.



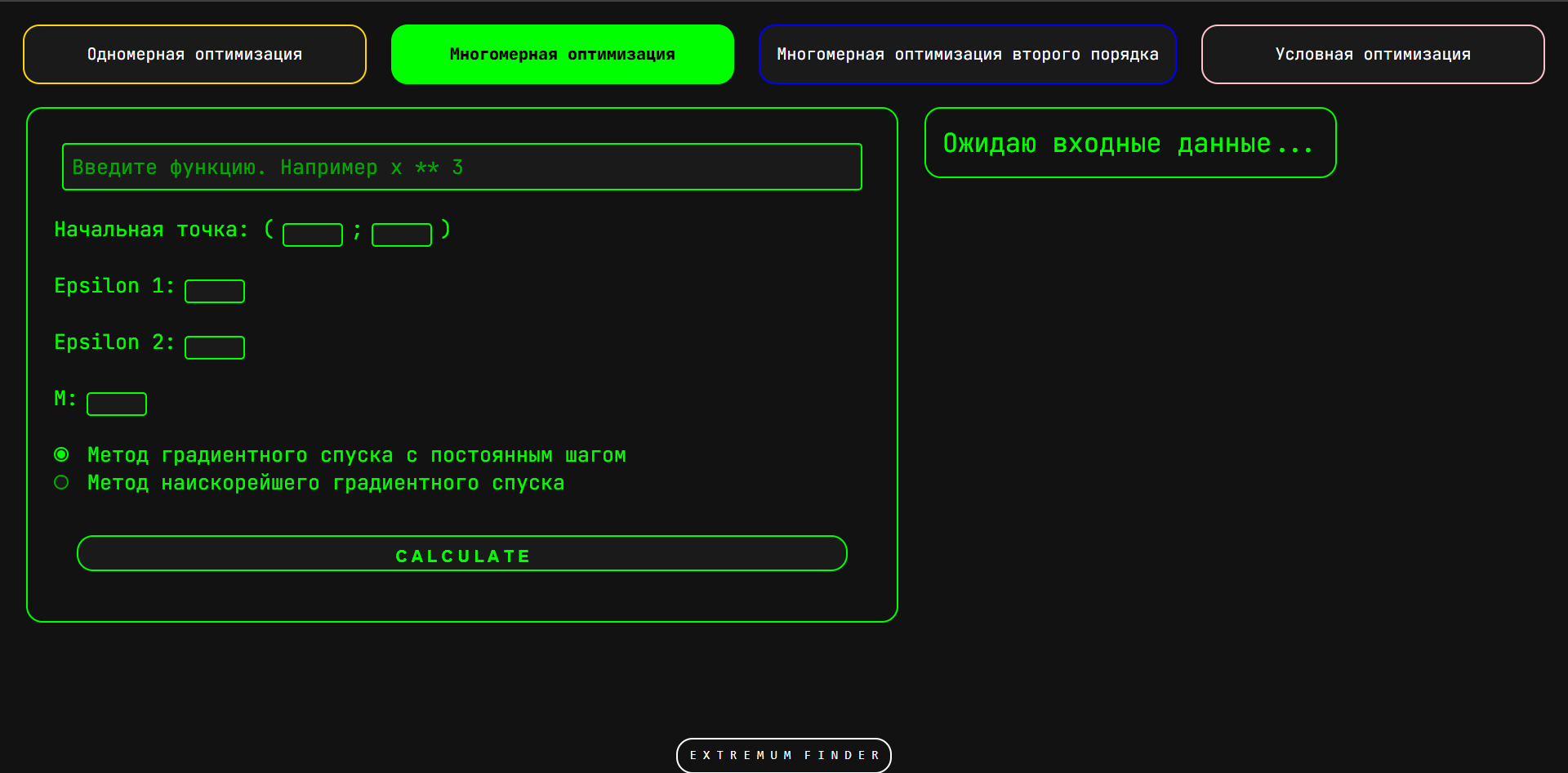
1. Разработать программный модуль решения задачи индивидуального задания методом градиентного спуска с постоянным шагом.
2. Разработать программный модуль решения задачи индивидуального задания методом наискорейшего градиентного спуска.

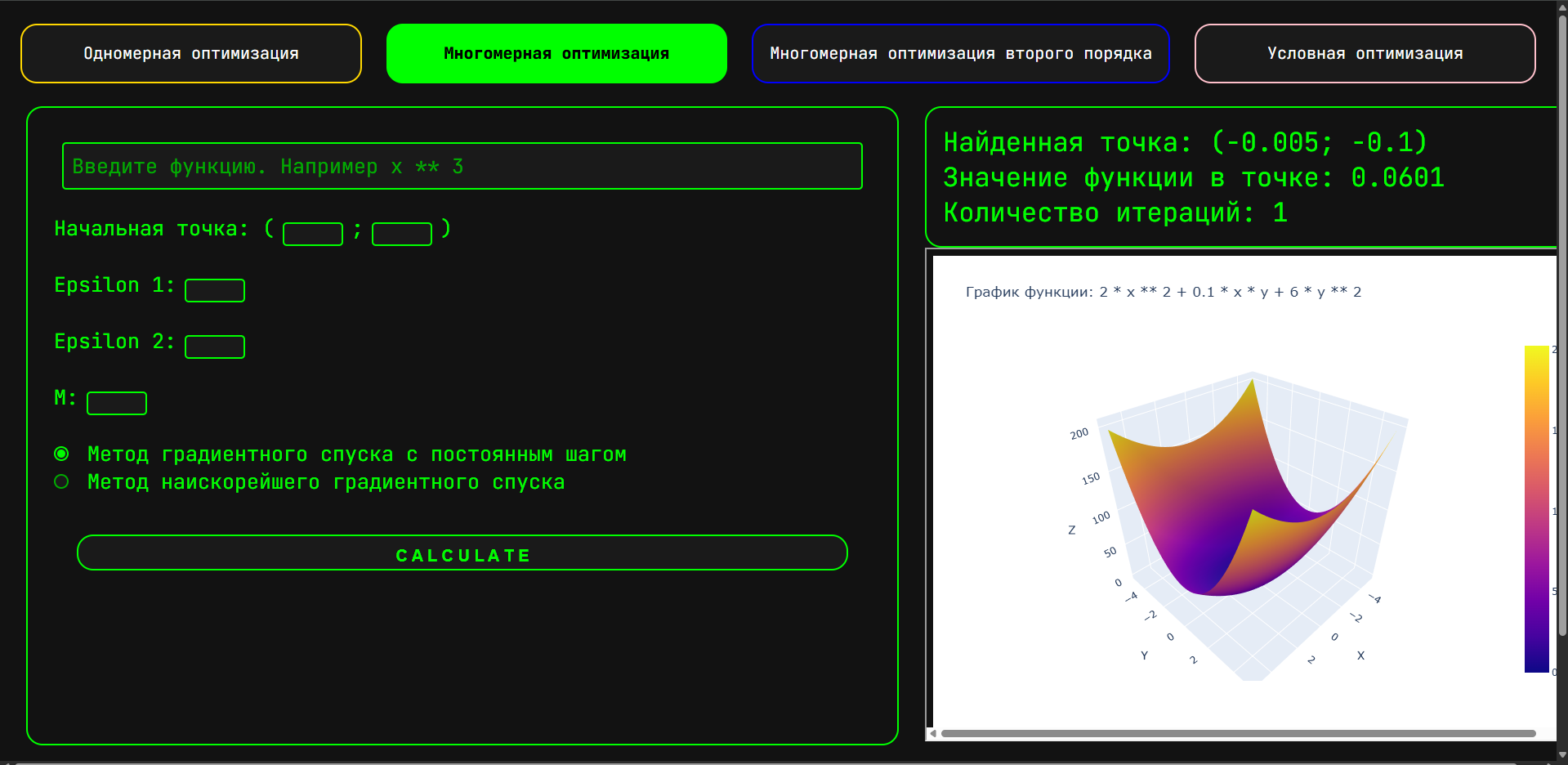
Задания 4 и 5 представлены в качестве 1 веб-приложения с меню выбора метода нахождения минимума.

Серверная часть веб приложения:

|  |
| --- |
| **from** string **import** ascii\_letters  **import** numpy **as** np  **import** sympy **as** sp  **from** sympy**.**parsing**.**sympy\_parser **import** parse\_expr  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** gradient\_descent\_constant\_step\_str(*self***,** **x0,** **epsilon1,** **epsilon2,** **M,** **t=**0.1)**:**          symbols **=** [sp**.**Symbol("x")**,** sp**.**Symbol("y")]          expr **=** parse\_expr(*self***.**func)          gradient\_expr **=** [sp**.**diff(expr**,** var) **for** var **in** symbols]          f **=** sp**.**lambdify(symbols**,** expr**,** 'numpy')          grad\_f **=** sp**.**lambdify(symbols**,** gradient\_expr**,** 'numpy')          x **=** np**.**array(x0**,** **dtype=**float)          k **=** 0  **while** k **<** M**:**              gradient **=** np**.**array(grad\_f(**\***x))    **if** np**.**linalg**.**norm(gradient) **<** 1e-10**:**  **break**                new\_x **=** x **-** t **\*** gradient              new\_fx **=** f(**\***new\_x)              fx **=** f(**\***x)    **if** np**.**isnan(new\_fx) **or** np**.**isinf(new\_fx)**:**                  t **=** t **/** 2  **continue**    **if** new\_fx **<** fx**:**                  delta\_x **=** np**.**linalg**.**norm(new\_x **-** x)                  delta\_f **=** abs(new\_fx **-** fx)    **if** delta\_x **<** epsilon1 **and** delta\_f **<** epsilon2**:**  **break**                    x **=** new\_x                  k **+=** 1  **else:**                  t **=** t **/** 2  **if** t **<** 1e-10**:**  **break**  **return** round(x[0]**,** 5)**,** round(x[1]**,** 5)**,** round(f(**\***x)**,** 5)**,** k    **def** steepest\_gradient\_descent(*self***,** **x0,** **epsilon1,** **epsilon2,** **M**)**:**          symbols **=** [sp**.**Symbol("x")**,** sp**.**Symbol("y")]          expr **=** parse\_expr(*self***.**func)          gradient\_expr **=** [sp**.**diff(expr**,** var) **for** var **in** symbols]          f **=** sp**.**lambdify(symbols**,** expr**,** 'numpy')          grad\_f **=** sp**.**lambdify(symbols**,** gradient\_expr**,** 'numpy')          x **=** np**.**array(x0**,** **dtype=**float)          k **=** 0          prev\_x **=** None          prev\_fx **=** None  **while** True**:**              gradient **=** np**.**array(grad\_f(**\***x))                gradient\_norm **=** np**.**linalg**.**norm(gradient)    **if** gradient\_norm **<** epsilon1**:**  **break**    **if** k **>=** M**:**  **break**    **def** phi(**t**)**:**                  new\_point **=** x **-** t **\*** gradient  **return** f(**\***new\_point)                a**,** b **=** 0**,** 1              golden\_ratio **=** (1 **+** np**.**sqrt(5)) **/** 2              tolerance **=** 1e-5                c **=** b **-** (b **-** a) **/** golden\_ratio              d **=** a **+** (b **-** a) **/** golden\_ratio    **while** abs(c **-** d) **>** tolerance**:**  **if** phi(c) **<** phi(d)**:**                      b **=** d  **else:**                      a **=** c                  c **=** b **-** (b **-** a) **/** golden\_ratio                  d **=** a **+** (b **-** a) **/** golden\_ratio                t\_k **=** (a **+** b) **/** 2                new\_x **=** x **-** t\_k **\*** gradient              new\_fx **=** f(**\***new\_x)              fx **=** f(**\***x)    **if** prev\_x **is** **not** None**:**                  delta\_x **=** np**.**linalg**.**norm(new\_x **-** x)                  delta\_f **=** abs(new\_fx **-** fx)    **if** delta\_x **<** epsilon2 **and** delta\_f **<** epsilon2**:**                      delta\_x\_prev **=** np**.**linalg**.**norm(x **-** prev\_x)                      delta\_f\_prev **=** abs(fx **-** prev\_fx)    **if** delta\_x\_prev **<** epsilon2 **and** delta\_f\_prev **<** epsilon2**:**                          x **=** new\_x                          k **+=** 1  **break**                prev\_x **=** x**.**copy()              prev\_fx **=** fx              x **=** new\_x              k **+=** 1    **return** round(x[0]**,** 5)**,**  round(x[1]**,** 5)**,** round(f(**\***x)**,** 5)**,** k **+** 1      **def** test()**:**      obj **=** ExtremumFinder()      obj**.**SetFunc("2 \* x \*\* 2 + 0.1 \* x \* y + 6 \* y \*\* 2")      x **=** [0**,** 0.5]      e1 **=** 0.15      e2 **=** 0.2      M **=** 10      x1**,** x2**,** y**,** k **=** obj**.**gradient\_descent\_constant\_step\_str(x**,** e1**,** e2**,** M)      print(x1**,** x2)      print(y)      print(k)  *# test()* |

Скриншоты:





6.Построить блок-схему алгоритма поиска экстремума по методу наискорейшего градиентного спуска.

