федеральное государственное автономное образовательное учреждение высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

# ОТЧЕТ

по лабораторной работе №2
«Системы нелинейных уравнений»

Вариант 2ав

по дисциплине «**Вычислительная математика**»

Автор: Кулаков Н.В.

Факультет: ПИиКТ

Группа: P3230

Преподаватель: Перл О.В.


Санкт-Петербург 2022

1. **Описание метода. Расчетные формулы.**

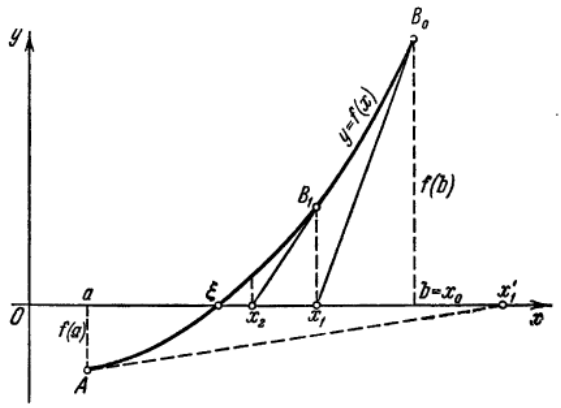
1) Решение нелинейных алгебраических и трансцендентных уравнений методом деления пополам.

Пусть дано уравнение f(x) = 0, заданная на [a, b], где сама функция является непрерывной и на границе принимает разные знаки. Тогда по следствию из теоремы Больцама-Коши найдется хотя бы одна точка, такая что значение функции в ней будет равняться 0. Таким образом для нахождения этой самой точки будем на каждой итерации делить отрезок пополам и проверять значения на его концах: если оно одинакового знака, то возьмем другой отрезок (вторую половину от предыдущей операции), где функция принимает разные знаки. Таким образом, через k итераций, длина отрезка, в котором находится наше значение, будет составлять 2^(-k)*abs(b – a). Если устремиться k к бесконечности, то мы найдем необходимую нам точку.

2) Решение нелинейных алгебраических и трансценденых уравнений методом касетельных (Ньютона).

Пусть у нас имеется начальное приближение x_0. Тогда эта точка будет находиться на расстоянии x – x_0 = h. По формуле Тейлора (берем первый член):

$$0 = f(x_n + h_n) \approx f(x_n) + h_n f'(x_n).$$




Откуда:

$$h_n = -\frac{f(x_n)}{f'(x_n)}.$$

Тогда получим:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Для того, чтобы процесс был сходящимся, достаточно выполнение следующего:

$$f(x_0) f''(x_0) > 0.$$

Если задать в качестве ошибки определенное значение, то на k итерации получим требуемое решение с заданной точностью.

3) Решение системы нелинейных уравнений методом простых итераций.

Рассмотрим систему нелинейных уравнений:

$$\begin{aligned} x_1 &= \varphi_1(x_1, x_2, \ldots, x_n), \\ x_2 &= \varphi_2(x_1, x_2, \ldots, x_n), \\ &\ldots\ldots\ldots\ldots \\ x_n &= \varphi_n(x_1, x_2, \ldots, x_n), \end{aligned}$$

Согласно первому достаточному условию сходимости итерационного процесса, также как и в первой лабораторной работе, в данном методе требуется выполнение условий сходимости матрицы, только уже составленной из производных.

Введем понятия:

$$x = \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}, \quad \varphi(x) = \begin{bmatrix} \varphi_1(x) \\ \cdot \\ \cdot \\ \cdot \\ \varphi_n(x) \end{bmatrix}.$$ - столбец функций n переменных.

$$\varphi'(x) = \left[\frac{\partial \varphi_i}{\partial x_j}\right]$$ - матрица производных

Также введем нормы:

$$\| x \|_l = \sum_{i=1}^{n} |x_i|. \qquad \| \varphi'(x) \|_m = \max_i \sum_{j=1}^{n} \left| \frac{\partial \varphi_i(x)}{\partial x_j} \right| \qquad \| \varphi'(x) \|_I = \max_{x \in G} \| \varphi'(x) \|_m$$

Тогда согласно теореме должно выполняться следующее:

$$\| \varphi'(x) \|_I \leqslant q < 1,$$

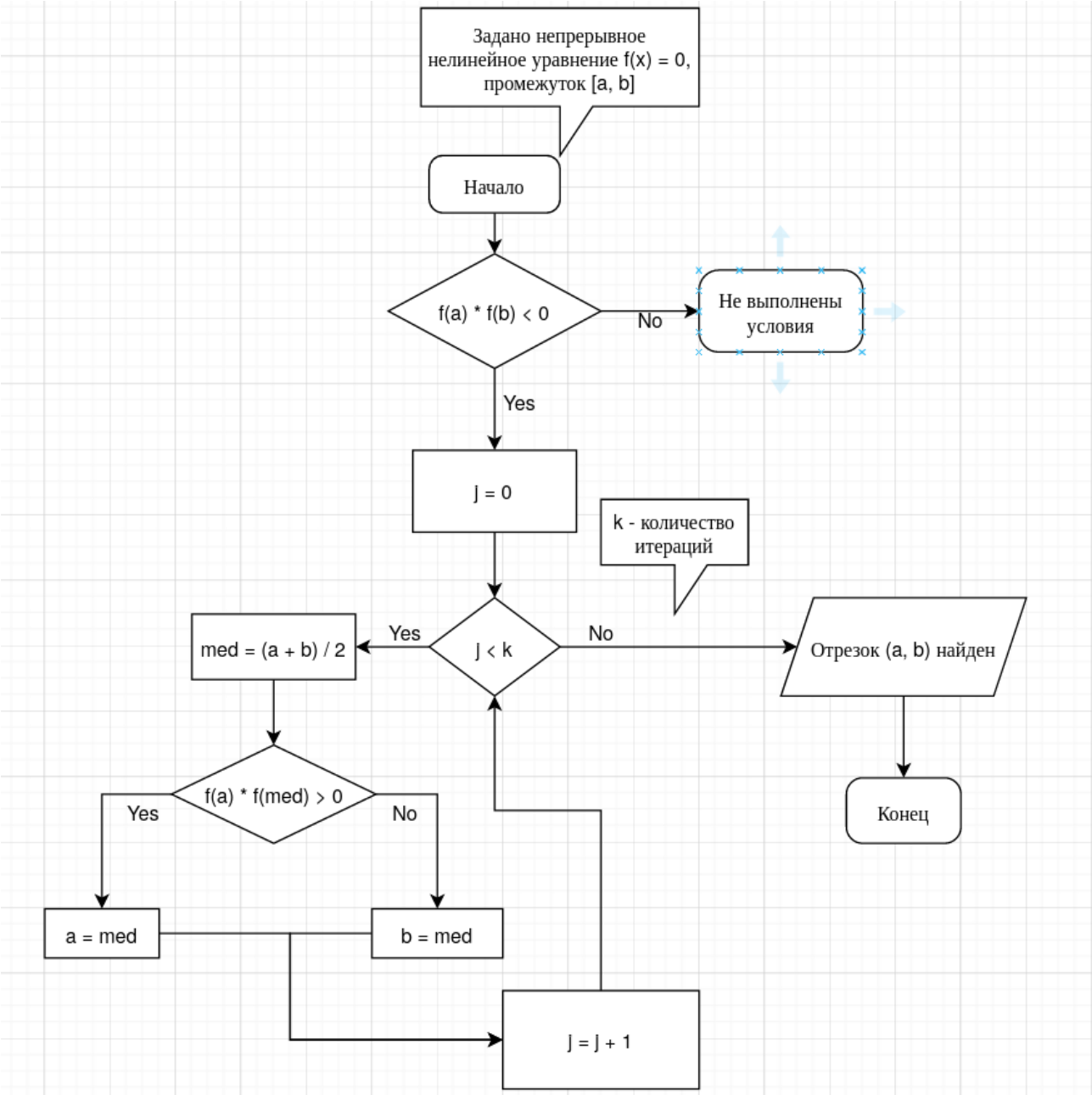
Таким образом, если вышеперечисленное выполняется, то можно переходить к решению самой системы.

Возьмем в качестве начального приближения значения. Если для них выполняется сходимость матрицы, то как в первой лабораторной работе ищем соответствующие решения нелинейного уравнение, указанного в начале пункта.

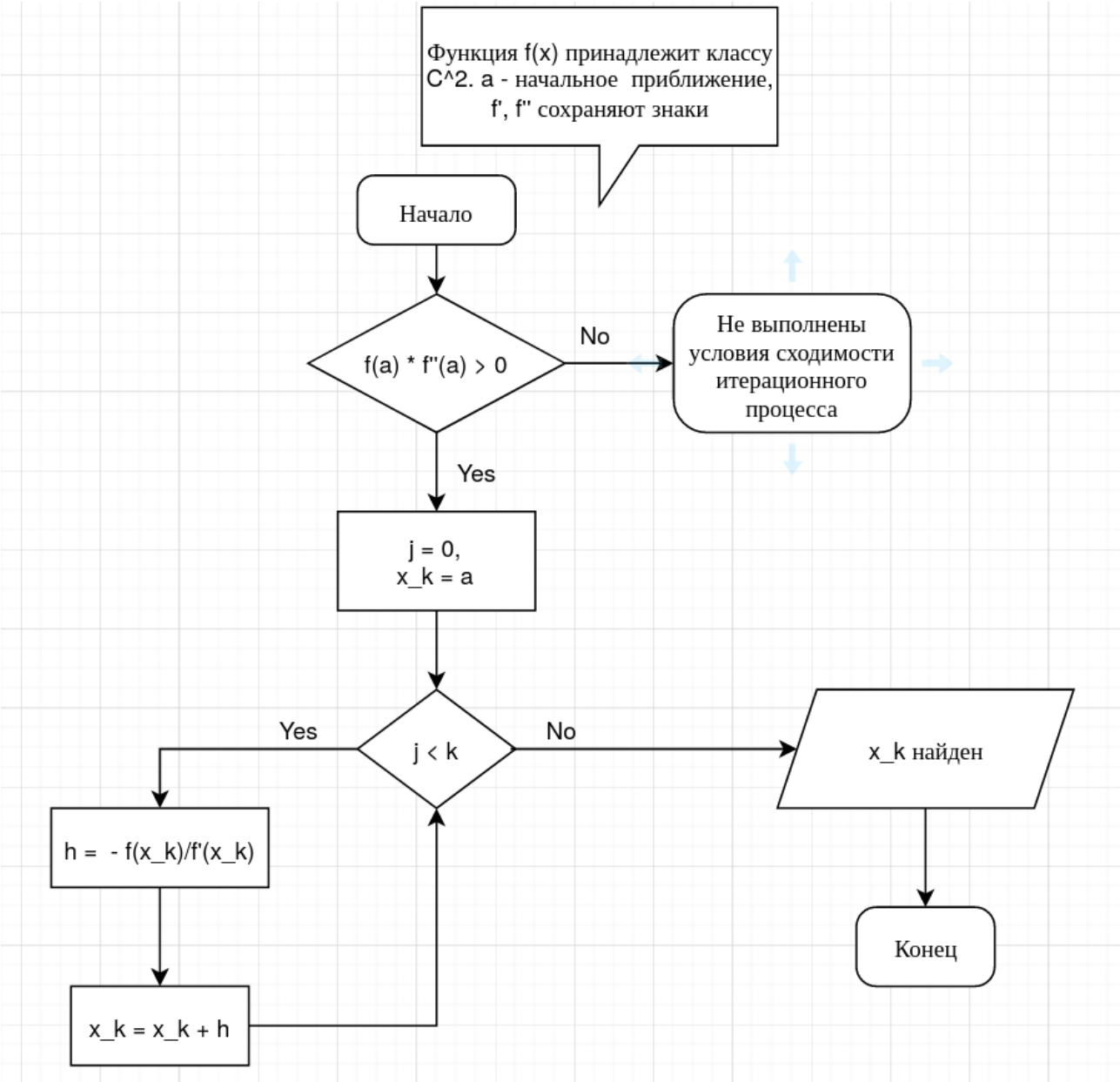
Критерием окончания итерационного процесса является значение разности неизвестных между итерациями, меньшее заданного.

## 2. Блок-схема численного метода.

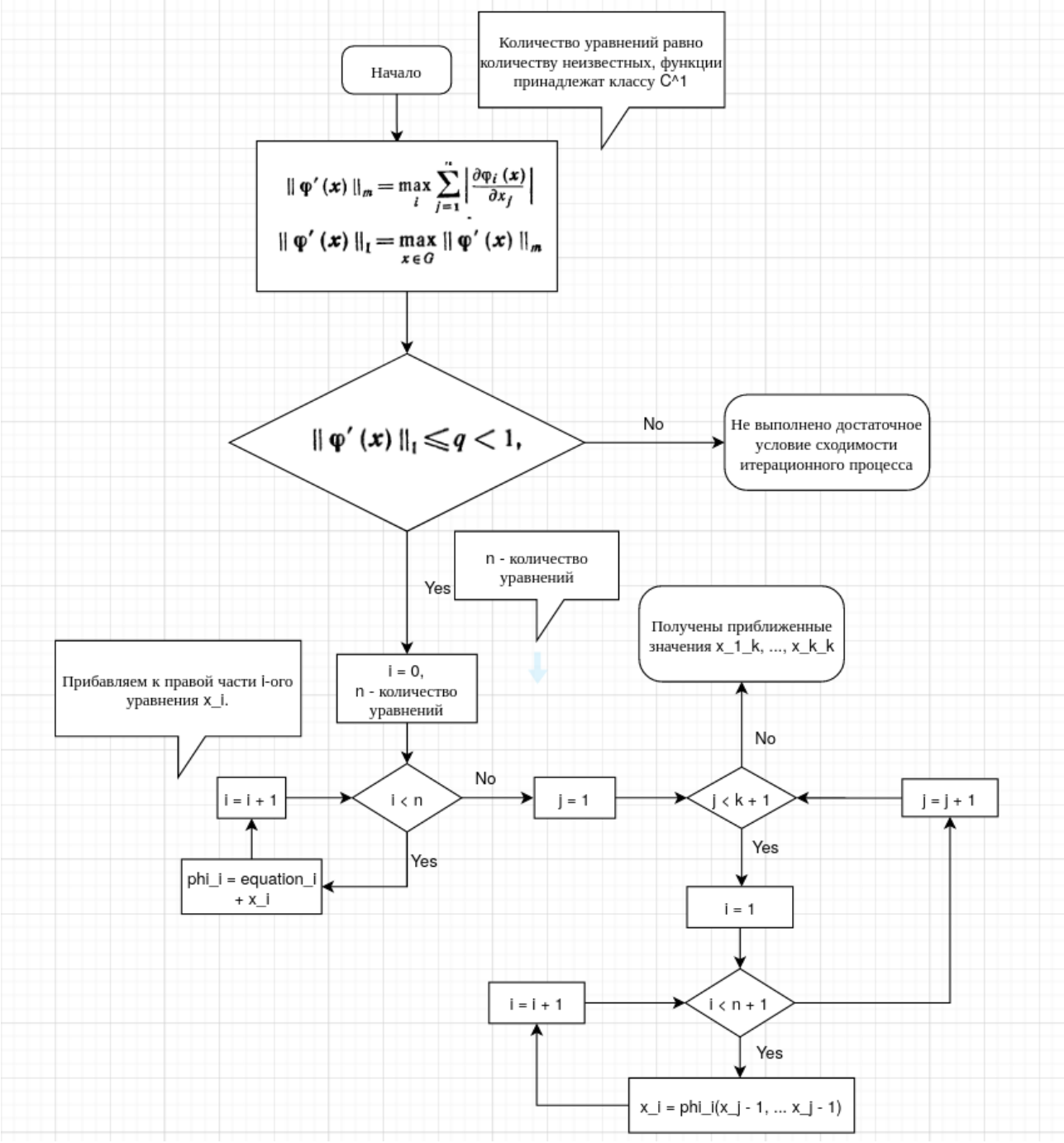
1) Решение нелинейных алгебраических и трансцендентных уравнений методом деления пополам.



2) Решение нелинейных алгебраических и трансценденых уравнений методом касетельных (Ньютона).



3) Решение системы нелинейных уравнений методом простых итераций.



### 3. Листинг реализованного численного метода программы.

```
def split_half(data: dict) -> dict:
    node_root, var_list = parse.parse_expression(data["equation"])
    assert (len(var_list) == 1)
    range_min = data["data"]["range_min"]
    range_max = data["data"]["range_max"]
    fval_range_min = node_root.calculate({var_list[0]: range_min})
    fval_range_max = node_root.calculate({var_list[0]: range_max})
    if (sum([fval_range_min > 0, fval_range_max > 0]) != 1):
        return {"error": "Invalid arguments. Function values of borders are same sigh"}
    for _ in range(data["data"]["iteration"]):
        range_med = (range_max + range_min) / 2
        fval_range_med = node_root.calculate({var_list[0]: range_med})
        if (sum([fval_range_med > 0, fval_range_min > 0]) == 1):
            range_max = range_med
        else:
            range_min = range_med
    return {"result": {"range_min": range_min, "range_max": range_max}}


def tangent(data: dict) -> dict:
    node, var_list = parse.parse_expression(data["equation"])
    assert(len(var_list) == 1)
    x_0 = data["data"]["x_0"]
    f = node.calculate({var_list[0]: x_0})
    f_ll = grad(grad(node_flatten(node, {"x": x_0}, "x")))(x_0)
# проверка достаточного условия
    if (f * f_ll <= 0):
        return {"error": "Invalid arguments. Derivative" * func <= 0. Iteration process doesn't converge"}
    x_prev = x_0
    for _ in range(data["data"]["iterations"]):
        f_x = node.calculate({var_list[0]: x_prev})
        f_x_l = grad(node_flatten(node, {"x": x_prev}, "x"))(x_prev)
        x_prev -= f_x / f_x_l
    return {"result": x_prev}
```

```
def simple_iteration(data: dict) -> dict:
    equation_list = data["equation"]
    node_list = []
    x0_dict = data["data"]["x_0"]
    x0_dict_keys = list(x0_dict.keys())
    assert(len(x0_dict_keys) == len(equation_list))
    iterations = data["data"]["iterations"]
# добавление с правой части числа
    for i in range(len(equation_list)):
        equation_list[i] += "+" + x0_dict_keys[i]
        n, v = parse.parse_expression(equation_list[i])
        node_list.append(n)
    matrix_phi = Matrix(len(equation_list), len(x0_dict_keys))
    # check convergence
    for i in range(matrix_phi.rows):
        for j in range(matrix_phi.columns):
            matrix_phi[i][j] = grad(node_flatten(node_list[i], x0_dict, x0_dict_keys[j]))
(x0_dict[x0_dict_keys[j]])
    norm = max([sum([matrix_phi[i][j] for j in range(matrix_phi.columns)]) for i in range(matrix_phi.rows)])
    if (norm >= 1):
        return {"error" : "This equations for those basic arguments are not convergent"}
# итерация
    x0_dict_prev = copy(x0_dict)
    for _ in range(iterations):
        for i in range(len(node_list)):
            x0_dict[x0_dict_keys[i]] = node_list[i].calculate(x0_dict_prev)
        x0_dict_prev = copy(x0_dict)
    return {"values": x0_dict_prev}
```

## 4. Примеры и результаты работы программы на разных данных.

```
[  # входные данные
  {
    "method" : "split_half",
    "equation" : "x^2 + \\e^4",
    "data" : {
      "range_min": -3,
      "range_max":  2,
      "iterations" : 10
    }
  },
  {
    "method" : "tangent",
    "equation" : "x^4 - 3 * x^2 + 75 * x - 10000",
    "data" : {
      "range_min": -3,
      "range_max":  2,
      "x_0": -11,
      "iterations" : 10
    }
  },
  {
    "method" : "simple_iteration",
    "equation": [
      "x_1^2 + x_2^4 + \\e^4",
      "x_1^4 + x_2^4 + \\e^5"
    ],
    "data": {
      "range_min" : 4,
      "range_max" : 6,
      "iterations": 10,
      "x_0": {
        "x_1" : 4,
        "x_2" : 3
      }
    }
  }
]
```

```
# результат работы
python3 src/main.py
Input file: data/task-2/input.json
Output file:
Lab number: 2
{"error": "Invalid arguments. Function values of borders are same sigh"}
{"result": -10.260964380932977}
{"error": "This equations for those basic arguments are not convergent"}
```

## 5. Вывод

Базовый вывод: в ходе выполнения лабораторной работы я узнал какими способами можно решать системы нелинейных уравнений, а также просто уравнения. Также разобрался с тем, что такое производная, интеграл и как их считать с заданной точностью с помощью компьютера ( для их написания применил способ обертки функций функциями, чтобы работало как в функциональном программировании :)). Кроме того, написал свой личный парсер, который поддерживает основные бинарные операции, скобки, а также константы. Само выражение разбирается в 2 этапа: из выражений составляются токены, парсер с учетом приоритетов операций преобразует токены в дерево, корнями которого являются переменные, костанты и числа. Сложность работы парсера: O(n).
P.S. переписал архитектуру с первой лабораторной работы под модульность, добавил возможность использования потоков ввода-вывода для корректного выводаа выражений (отключил буферизацию).

Вывод-сравнение:

1) Метод деления пополам эффективен в случае, если мы знаем 2 точки, знаки функций от которых различны. Интервал уменьшается за 1 итерацию в 2 раза. Сложность: O(k*d) — d — кол-во слагаемых

2) Метод хорд в сравнении с методом пополам является более эффективным решением, поскольку порядок его сходимости равен золотому сечению (1.618….). Сложность: O(k(d)— d — кол-во слагаемых

3) Метод Ньютона имеет еще больший порядок сходимости (он квадратичен), однако требует того, чтобы функция была дважды дифференциируема, также выполнялись условия сходимости итерационного процесса (касательная должна лежать так, чтобы ее точка пересечения с нулем была ближе к требуемому значению). Сложность: O(k*d) — d — кол-во слагаемых. Кроме того, если не хочется считать производные, то можно воспользоваться упрощенным методом Ньютона.

4) Метод простых итераций — один из простейший методов для решения нелинейных уравнений. Суть заключается в приведении к сжимающемуся отображению. Сходится со скоростью геометрической прогрессии (не уточнял).

5) Метод Ньютона заключается в методе последовательного приближения. Сложность O(k * n^3) — вычисление производной O(n), вычисление значения функции в каждой точке O(n), k — количество итераций. Скорость сходимости очень быстрая (Демидович)

6) Метод последовательных итераций требует выполнения достаточного условия сходимости матрицы производных. Сложность — O(k * n^2), поскольку на каждом этапе нам не нужно вычислять обратную матрицу. Скорость сходимости аналогична методы последовательных итераций для единичных нелинейных уравнений. Расчитывается аналогично.