МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА № 3

по дисциплине

“Вычислительная математика”

Вариант “Метод простых итераций”

Выполнила:

Студентка группы P3232

Копалина Майя
Алексеевна

Преподаватель:

Перл Ольга

Вячеславовна

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург, 2024

Задание:

Придумать алгоритм и написать код, решающий системы нелинейных уравнений с метода простых итераций.

Описание метода:

Метод простых итераций - это численный метод решения системы нелинейных уравнений (СНУ), основанный на последовательном обновлении приближенных значений корней системы. Сначала система уравнений переписывается в эквивалентную форму с использованием функции преобразования. Затем начальное приближение корней задается, и их значения обновляются итерационно согласно определенному правилу. Процесс продолжается до тех пор, пока изменения приближенных значений станут достаточно малы или до достижения предельного числа итераций.

Есть система нелинейных уравнений:

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \dots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases}$$

Мы можем переписать её в виде:

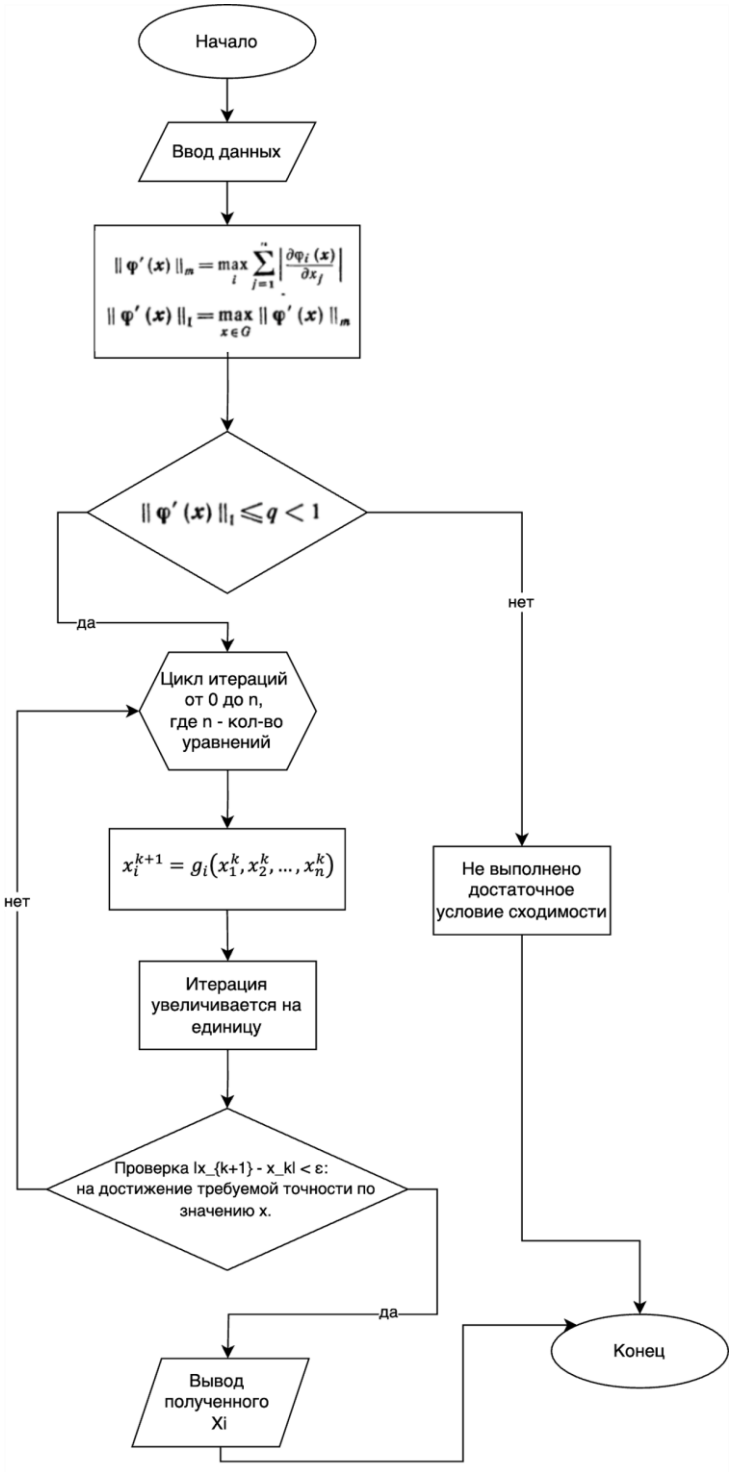
$$\begin{cases} x_1 = g_1(x_1, x_2, \dots, x_n) \\ x_2 = g_2(x_1, x_2, \dots, x_n) \\ \dots \\ x_n = g_n(x_1, x_2, \dots, x_n) \end{cases}$$

Значения обновляем итерационно, по следующему правилу: $x_i^{k+1} = g_i\left(x_1^k, x_2^k, \dots, x_n^k\right)$.

Словесное описание алгоритма:

1. Записать систему уравнений в виде F(x) = 0, где F(x) - функция преобразования системы уравнений.
2. Выбрать начальное приближение для корней системы уравнений.
3. Определить правило итерационного обновления приближенных значений корней. Обычно это делается путем преобразования системы уравнений к виду $x_n = g_n(x_1, x_2, \dots, x_n)$, где g(x) - функция итерационного обновления.
4. Начать итерационный процесс: вычислить новые значения корней по формуле $x_{n+1} = g_т(x_1, x_2, \dots x_n)$
5. Повторять шаг 4 до тех пор, пока изменения приближенных значений корней не станут достаточно малы (используется критерий остановки) или до достижения предельного числа итераций.
6. Проверить полученные значения корней, чтобы убедиться в их точности и сходимости метода.

Блок-схема:


Начало
Ввод данных
$\| \varphi'(x) \|_m = \max_i \sum_{j=1}^{n} \left| \frac{\partial \varphi_i(x)}{\partial x_j} \right|$
$\| \varphi'(x) \|_1 = \max_{x \in G} \| \varphi'(x) \|_m$
$\| \varphi'(x) \|_1 \leqslant q < 1$
да
нет
Цикл итераций от 0 до n, где n - кол-во уравнений
$x_i^{k+1} = g_i(x_1^k, x_2^k, \dots, x_n^k)$
Итерация увеличивается на единицу
Проверка |x_{k+1} - x_k| < ε: на достижение требуемой точности по значению x.
нет
да
Не выполнено достаточное условие сходимости
Вывод полученного Xi
Конец

Программа, написанная на python:

```
def first_transform(args: []) -> float:
    return 0.0

def second_transform(args: []) -> float:
    return 0.0

def third_transform(args: []) -> float:
    sub_expr = pow(args[0], 2) * pow(args[1], 2) - 6 * pow(args[1], 3) + 8
    if sub_expr < 0:
        return float('nan')
    return pow((pow(args[0], 2) * pow(args[1],2) - 6 * pow(args[1], 3) + 8) / 3, 1/3)

def fourth_transform(args: []) -> float:
    return (pow(args[0], 4) + 2) / 9

def fifth_transform(args: []) -> float:
    return -pow(args[0], 2) + 2 * args[1] * args[2] + 0.1

def sixth_transform(args: []) -> float:
    return -pow(args[1], 2) - 3 * args[0] * args[2] - 0.2

def seventh_transform(args: []) -> float:
    return -pow(args[2], 2) - 2 * args[0] * args[1] + 0.3

def get_transform_functions(n: int):
    if n == 1:
        return [first_transform, second_transform]
    elif n == 2:
        return [third_transform, fourth_transform]
    elif n == 3:
        return [fifth_transform, sixth_transform, seventh_transform]
    else:
        return [default_function]

def solve_by_fixed_point_iterations(system_id, number_of_unknowns,
initial_approximations):
    functions = get_functions(system_id)
    transforms = get_transform_functions(system_id)
    current_approximations = initial_approximations[:]
    next_approximations = [0.0] * number_of_unknowns
    max_iterations = 100
    tolerance = 1e-5
    step_length = 0.00001
```

```
    if any(abs(approx) > 1e20 for approx in initial_approximations):
        return [float('nan')] * number_of_unknowns

    for iter in range(max_iterations):
        for i in range(number_of_unknowns):
            next_approximations[i] = transforms[i](current_approximations)

            if math.isnan(next_approximations[i].real) or
math.isinf(next_approximations[i].real):
                return [float('nan')] * number_of_unknowns

        converged = all(abs(next_approximations[i] - current_approximations[i]) <=
tolerance for i in range(number_of_unknowns))
        if converged:
            break

        current_approximations = next_approximations[:]

    return next_approximations
```

Примеры работы программы:

Метод простых итераций успешно запускается и сходится к решению на различных данных.
При правильном выборе начальных приближений и условиях сходимости, метод позволяет находить корни систем нелинейных уравнений.
Однако, при неправильном выборе начальных приближений или отсутствии условий сходимости метод может расходиться или сходиться к неверному решению.

1) Пример 1
Вход:
1
2
1.0
2.0
Выход:
0.9916078527646449
1.9900660838909958

2) Пример 2
Вход:
3
3
0.5
-1.0
2.0
Выход:
0.4538072181466193
-1.0304576235085257
1.9538281206568267

3) Пример 3
Вход:
3
3
0.0
0.0
0.0
Выход:
0.0009950238612479096
-0.001990174530636173
0.0029851821351471173

4) Пример 4

Вход:
1
1
0.0
Выход:
0.0

5) Пример 5
Вход:
2
2
1.0
2.0
Выход:
1.445721981124815
2.1435695839180386

Вывод:

Метод простых итераций для решения систем нелинейных уравнений представляет собой простой и понятный подход, основанный на последовательном уточнении начального приближения до достижения требуемой точности. Метод простых итераций является удобным инструментом для численного решения СНУ, но требует внимательного подхода и анализа для достижения стабильных результатов.

Преимущества:

- Простая реализация и понятная геометрическая интерпретация.
- Не требует вычисления производных функций.
- Применим к широкому классу нелинейных систем уравнений.

Недостатки:

- Может иметь медленную сходимость.
- Чувствителен к выбору начальных приближений.
- Может не сойтись к решению при неподходящих условиях.
- Алгоритмическая сложность может быть высокой.

Метод простых итераций подходит для решения простых систем нелинейных уравнений с хорошим начальным приближением. В более сложных случаях могут потребоваться другие методы, например, метод Ньютона.

Алгоритмическая сложность:
O(N*M*I), где: N – кол-во неизвестных переменных, M – кол-во функций преобразования в системе, I – кол-во количество итераций.

Дополнительные проверки:

- Проверка на сходимость.
- Проверка на переполнение/деление на ноль.
- Проверка начальных приближений.
- Выбор шага итераций.

Ограничения:

- Может не сойтись к решению.
- Может найти только локальное решение.
- Может иметь ограниченную точность.

Сравнение с другими методами:

- Метод Ньютона: более быстрая сходимость, но требует вычисления производных.
- Метод бисекции: прост, но применим только к уравнениям с одной переменной.

При сравнении с методом Ньютона и методом бисекции, метод простых итераций отличается своей простотой в реализации и отсутствием необходимости вычисления производных функций. Однако его сходимость

может быть медленной, что требует большего числа итераций для достижения точности. Метод Ньютона обеспечивает быструю сходимость, но требует вычисления производных, что может быть сложным. Метод бисекции прост в использовании и гарантирует сходимость к решению на отрезке с изменением знака функции, но медленно сходится и применим только к уравнениям с одной переменной.

Анализ численной ошибки метода:
Численная ошибка метода может возникать из-за неточности начальных приближений, ошибок округления при выполнении арифметических операций и условий сходимости. Контроль численной ошибки важен для обеспечения правильной сходимости метода и получения точных результатов.

Результат выполнения программы на платформе.
Тесты пройдены (кроме одного...).

Assignment results

Score	Plagiarism
73.875 / 100	**35.353535%**

Number	Status	Score
1	Success	12 / 12
2	Success	5.7749996 / 11
3	Success	5.7749996 / 11
4	Success	5.7749996 / 11
5	Success	11 / 11
6	Success	11 / 11
7	Success	5.7749996 / 11
8	Success	5.7749996 / 11
9	Success	11 / 11