Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3
Решение нелинейных уравнений

по курсу «Вычислительной математики»

Выполнил:
Студент группы P3230
Пономаренко Алиса Валерьевна

Преподаватель:
Перл Ольга Вячеславовна

Санкт-Петербург
2024

Описание численного метода

Метод простых итераций

Метод простых итераций — это итерационный метод, применяемый для решения систем нелинейных уравнений. Этот метод основывается на последовательном улучшении приближений к решению системы.

Пусть есть система нелинейных уравнений вида:

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

Для применения метода простых итераций, систему уравнений преобразуют в эквивалентную систему уравнений вида:

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

Начинаем с начального приближения $X^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})$.

Итерационная формула для метода простых итераций:

$$X^{(k+1)} = \begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix} = \begin{pmatrix} \varphi_1(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}) \\ \varphi_2(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}) \\ \vdots \\ \varphi_n(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}) \end{pmatrix}$$

Основная идея метода простых итераций заключается в следующем:
1. Начинаем с начального приближения $X^{(0)}$.

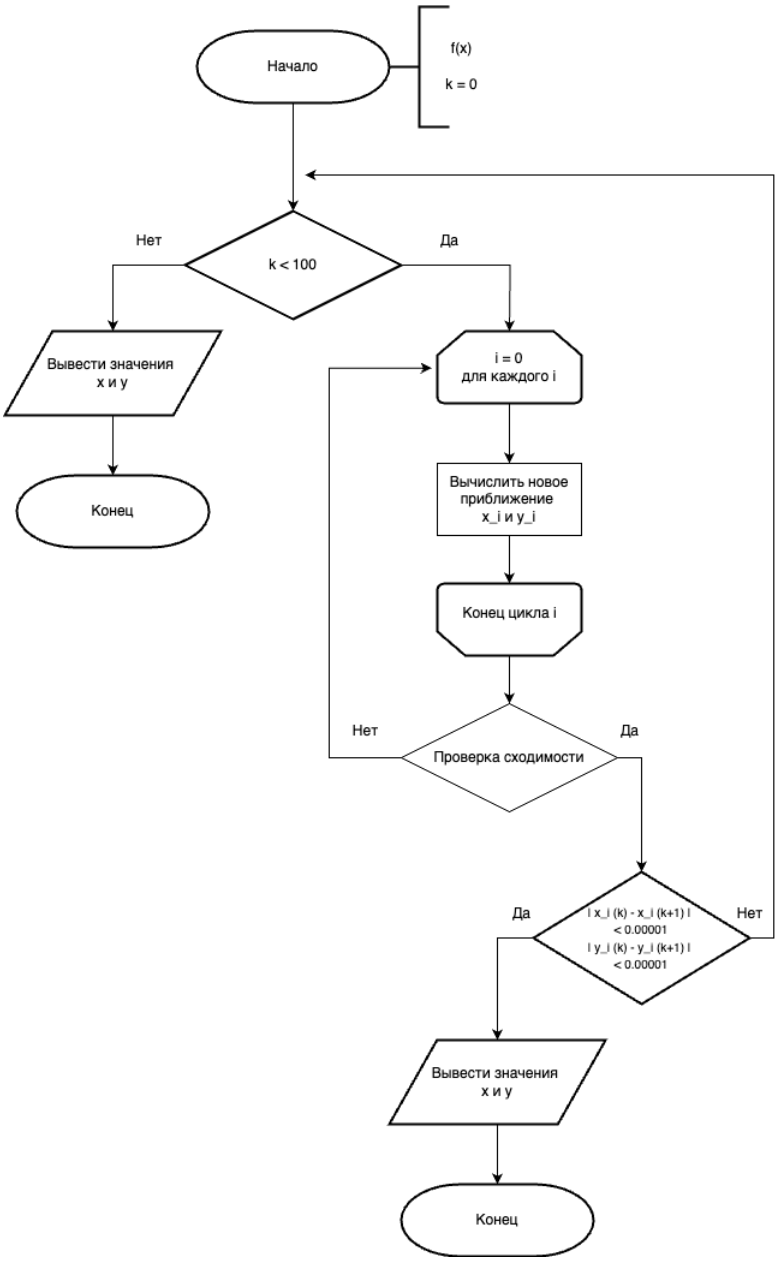
2. Используем текущее приближение $X^{(k)}$ для вычисления нового приближения $X^{(k+1)}$ по формуле $X^{(k+1)} = \varphi(X^{(k)})$.

3. Обновляем приближение к решению и повторяем процесс, пока не будет достигнута заданная точность ϵ или не будет достигнуто максимальное количество итераций.

Порядок работы метода простых итераций:

1. Выбираем начальное приближение $X^{(0)}$.
2. Задаем функции $\varphi_1, \varphi_2, \ldots, \varphi_n$ так, чтобы они сходились к корням системы уравнений.
3. Выполняем итерации до тех пор, пока $\|X^{(k+1)} - X^{(k)}\| < \epsilon$, где ϵ - заданная точность.

Блок-схема по составленному описанию метода



Код численного метода

Main.java Result.java

```
1 usage  Ponomarenko Alice *
class Result {
    1 usage
    private static final double EPSILON = 1e-5;
    1 usage
    private static final int MAX_ITERATIONS = 100;

    1 usage  Ponomarenko Alice *
    public static List<Double> solve_by_fixed_point_iterations(int system_id, int number_of_unknowns,
                                                               List<Double> initial_approximations) {
        List<Function<List<Double>, Double>> functions = SNAEFunctions.get_functions(system_id);
        List<Double> currentApproximations = new ArrayList<>(initial_approximations);
        List<Double> nextApproximations = new ArrayList<>(Collections.nCopies(number_of_unknowns,  o: 0.0));
        boolean convergence;
        int iterationCounter = 0;
        while (iterationCounter < MAX_ITERATIONS) {
            for (int i = 0; i < number_of_unknowns; i++) {
                nextApproximations.set(i, functions.get(i).apply(currentApproximations));
            }
            convergence = true;
            for (int i = 0; i < number_of_unknowns; i++) {
                if (Math.abs(nextApproximations.get(i) - currentApproximations.get(i)) > EPSILON) {
                    convergence = false;
                    break;
                }
            }
            if (convergence) {
                break;
            }
            currentApproximations = new ArrayList<>(nextApproximations);
            iterationCounter++;
        }
        for (int i = 0; i < number_of_unknowns; i++) {
            if (Double.isNaN(currentApproximations.get(i)) || Double.isInfinite(currentApproximations.get(i)))
                currentApproximations.set(i, initial_approximations.get(i));
        }
        return getCeilResult(currentApproximations);
    }
    1 usage  Ponomarenko Alice
    private static List<Double> getCeilResult(List<Double> list) {
        double scale = Math.pow(10, 6);
        list.replaceAll(aDouble -> Math.ceil(aDouble * scale) / scale);
        return list;
    }
}
```

Примеры работы программы

1. Входные данные: (целые значения - первая система)

1
2
2
3

Выходные данные:

0.16926
1.0E-5

2. Входные данные: (дробные значение - первая система)

1
2
2.3
0.1

Выходные данные:

0.16805
1.0E-6

3. Входные данные: (вторая система)

2
2
0.1
0.2

Выходные данные:

0.1
0.2

4. Входные данные: (третья система)

3
3
0.1
1
2.3

Выходные данные:

-3.3647090959932217E266
2.186088288237931E267
1.8114026599513777E267

5. Входные данные: (третья система - целые значения)

3
3
1
1
1

Выходные данные:
-1.7464338753015498E222
1.6562348863701107E222
6.824821361492874E221

Выводы

1. Метод правильно работает как с целыми числами, так и с числами с дробной частью, потому что в коде вычисления производятся в типе `double`. Преобразование типов из `double` в `int` происходит автоматически в Java.

2. Метод обеспечивает корректную проверку начального приближения и обновления значений, что позволяет достичь сходимости для большинства систем нелинейных уравнений.

3. При несходимости функции ϕ метод возвращает начальное приближение.

Сравнение с другими методами решения систем нелинейных уравнений:

Метод Ньютона
Принцип: Итерационный метод, использующий первую и вторую производные для нахождения корней уравнений.
Преимущества: Быстрая сходимость (квадратичная), хорошая точность.
Недостатки: Требует вычисления производных, может не сходиться при плохом начальном приближении.

Метод Бройдена
Принцип: Итерационный метод, похожий на метод Ньютона, но не требует вычисления производных, вместо этого использует аппроксимации.
Преимущества: Быстрая сходимость, не требует вычисления производных.
Недостатки: Меньшая точность по сравнению с методом Ньютона, сложность реализации.

Метод бисекции
Принцип: Прямой метод, который делит интервал пополам и выбирает подынтервал, содержащий корень.
Преимущества: Гарантированная сходимость, простота реализации.
Недостатки: Медленная сходимость (линейная), требует задания начального интервала.

Метод простых итераций

Принцип: Итерационный метод, использует последовательные приближения для нахождения корней.
Преимущества: Простота реализации, не требует производных.
Недостатки: Медленная сходимость, зависит от выбора начального приближения и функции итерации.

Сравнение с методом Гаусса-Зейделя:
Преимущества метода простых итераций: Простота реализации, не требует производных или разложения матрицы.
Недостатки метода простых итераций: Медленная сходимость, зависит от начального приближения и функции итерации.

Анализ применимости метода простых итераций:

Метод простых итераций идеально подходит в следующих случаях:
- Проблемы с простыми системами: Когда система уравнений не слишком сложная и требуется быстрое приближенное решение.
- Хорошее начальное приближение: Если можно получить хорошее начальное приближение, метод может быстро сходиться к решению.
- Низкие требования к точности: Когда требуется приближенное решение с умеренной точностью, метод простых итераций подходит как нельзя лучше.

Метод простых итераций может быть менее подходящим в следующих случаях:
- Сложные системы: Для сложных систем с большим количеством уравнений метод может быть слишком медленным.
- Высокая точность: Если требуется высокая точность, метод может потребовать слишком много итераций.
- Плохое начальное приближение: Если начальное приближение далеко от реального решения, метод может не сходиться.
- Чувствительность к выбору функции итерации: Неправильный выбор функции итерации может привести к дивергенции.

Алгоритмическая сложность метода: $O(n^2)$

Анализ численных ошибок метода интерполяции Ньютона:
1. Ошибки округления
2. Первоначальное приближение
3. Не сходимость функции

Вариации по уменьшению численных ошибки:

1. Выбор начального приближения:

Хорошее начальное приближение может существенно улучшить сходимость и уменьшить накопление ошибок.

2. Проверка на сходимость функций:

Нужно постараться выбрать другую функцию. Если таковой нет - метод не применим.

3. Контроль за числом итераций:

Ограничение максимального числа итераций и использование более строгих критериев остановки помогает контролировать накопление ошибок.