Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирования»

Лабораторная работа №2 по курсу «Численные методы»

Студент: Т. Д. Голубев
Преподаватель: И. Э. Иванов
Группа: М8О-306Б-22
Дата:
Оценка:
Подпись:

Москва, 2025

Лабораторная работа №2.1

Задача: Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

$$x^4 - 2x - 1 = 0$$

Описание

Для решения нелинейных уравнений вида $f(x) = 0$ рассмотрим два основных итерационных метода.

Метод простой итерации требует преобразования исходного уравнения к виду:

$$x = \varphi(x) \tag{1}$$

с последующим итерационным процессом:

$$x_{k+1} = \varphi(x_k), \quad k = 0, 1, 2, \ldots \tag{2}$$

Условия сходимости метода:

- Функция $\varphi(x)$ должна отображать отрезок $[a, b]$ в себя
- Существует $q < 1$ такое, что $|\varphi'(x)| \leq q$ для всех $x \in [a, b]$

Оценка погрешности на k-й итерации:

$$|x^* - x_k| \leq \frac{q^k}{1 - q}|x_1 - x_0| \tag{3}$$

Метод Ньютона реализуется по формуле:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \ldots \tag{4}$$

Достаточные условия сходимости:

- $f(x) \in C^2[a, b]$

- $f(a)f(b) < 0$
- $f'(x)$ и $f''(x)$ сохраняют знак на $[a, b]$
- Начальное приближение x_0 удовлетворяет $f(x_0)f''(x_0) > 0$

Метод Ньютона обладает квадратичной скоростью сходимости при выполнении условий.
Для практической реализации важны критерии остановки:

$$|x_{k+1} - x_k| < \varepsilon \quad \text{или} \quad |f(x_k)| < \varepsilon \tag{5}$$

Начальное приближение x_0 определяется графически из анализа поведения функции $f(x)$ на интересующем интервале.
Анализ зависимости погрешности от числа итераций показывает:

- Для метода простой итерации: $\ln \varepsilon \sim k$
- Для метода Ньютона: $\ln \ln \varepsilon \sim k$

Выбор между методами зависит от конкретной задачи и требований к точности вычислений.

Исходный код

```
package cat.mood;

import java.util.function.Function;

public class A {
    public static double derivative(Function<Double, Double> f, double x, double eps)
        double dy = f.apply(x + eps) - f.apply(x);
        return dy / eps;
    }

    public static double secondDerivative(Function<Double, Double> f, double x, doubl
        double fPlus = f.apply(x + eps);
        double fMinus = f.apply(x - eps);
        double fCenter = f.apply(x);

        return (fPlus - 2 * fCenter + fMinus) / (eps * eps);
```

```
}

public static boolean checkFunction(Function<Double, Double> phi, double eps, doub
    double x = a;
    while (x < b) {
        double y = phi.apply(x);
        if (y < a || y > b) {
            return false;
        }
        if (Math.abs(derivative(phi, x, eps)) >= 1) {
            return false;
        }
        x += eps;
    }

    return true;
}

public static double iteration(Function<Double, Double> phi, double eps, double a
    boolean check = checkFunction(phi, eps, a, b);
    if (!check) {
        throw new RuntimeException("Не выполнено условие сходимости");
    }

    double prev = a;
    double cur = phi.apply(prev);
    int iters = 1;
    while (Math.abs(cur - prev) > eps) {
        prev = cur;
        cur = phi.apply(prev);
        ++iters;
    }

    System.out.println("Количество итераций: " + iters);

    return cur;
}

public static double newton(Function<Double, Double> f, double eps, double a, doub
    if (f.apply(a) * f.apply(b) >= 0) {
        throw new RuntimeException("Не выполнено условие сходимости");
```

```
        }

        double prev = b;
        while (prev > a) {
            if (f.apply(prev) * secondDerivative(f, prev, eps) > 0) {
                break;
            }
            prev -= eps;
        }
        if (f.apply(prev) * secondDerivative(f, prev, eps) <= 0) {
            throw new RuntimeException("Не выполнено условие сходимости");
        }
        int iters = 1;
        double cur = prev - f.apply(prev) / derivative(f, prev, eps);
        while (Math.abs(cur - prev) > eps) {
            prev = cur;
            cur = prev - f.apply(prev) / derivative(f, prev, eps);
            ++iters;
        }

        System.out.println("Количество итераций: " + iters);

        return cur;
    }

    public static void main(String[] args) {
        System.out.println("Метод простой итерации:");
        System.out.println(iteration(x -> (Math.pow(2 * x + 1, 0.25)), 0.000001, 0, 2
        System.out.println("Метод Ньютона:");
        System.out.println(newton(x -> (Math.pow(x, 4) - 2 * x - 1), 0.000001, 0, 2))
    }
}
```

Результат

```
Метод простой итерации:
Количество итераций: 10
1.3953368880468564
Метод Ньютона:
Количество итераций: 6
```

```
1.3953369944670735
```

Вывод

В ходе выполнения работы были успешно реализованы и протестированы два численных метода решения нелинейных уравнений: метод простой итерации и метод Ньютона. Проведенные вычисления позволили сделать следующие выводы:

Сходимость методов:

- Метод Ньютона продемонстрировал более быструю сходимость (6 итераций) по сравнению с методом простой итерации (10 итераций)
- Оба метода сошлись к близким значениям корня: 1.395336888 (простая итерация) и 1.395336994 (метод Ньютона)

Точность результатов: Различие между полученными значениями составляет около 1.06×10^{-7}, что свидетельствует о хорошей точности обоих методов

Эффективность методов:

- Метод Ньютона оказался более эффективным по количеству требуемых итераций
- Метод простой итерации, хотя и потребовал больше вычислений, проще в реализации и не требует вычисления производной

Практические рекомендации:

- Для задач с вычислительно сложными производными целесообразно использовать метод простой итерации
- Когда доступно аналитическое выражение производной и важна скорость сходимости, предпочтительнее метод Ньютона

Результаты работы подтвердили теоретические положения о скорости сходимости рассматриваемых методов и продемонстрировали их практическую применимость для решения нелинейных уравнений.

Лабораторная работа №2.2

Задача: Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

$$\begin{cases} x_1^2 - 2\lg x_2 - 1 = 0 \\ x_1^2 - x_1 x_2 + 1 = 0 \end{cases}$$

Описание

1 Постановка задачи

Рассмотрим систему n нелинейных уравнений с n неизвестными:

$$\Big\{ f_1(x_1, x_2, ..., x_n) = 0 \; f_2(x_1, x_2, ..., x_n) = 0 \vdots \; f_n(x_1, x_2, ..., x_n) = 0 \tag{6}$$

Требуется найти решение $\mathbf{x}^* = (x_1^*, x_2^*, ..., x_n^*)^T$ с положительными компонентами.

Метод простой итерации

Преобразуем систему к виду:

$$\mathbf{x} = \Phi(\mathbf{x}) \tag{7}$$

где $\Phi = (\varphi_1, \varphi_2, ..., \varphi_n)^T$ - вектор-функция итерационного преобразования.
Итерационный процесс:

$$\mathbf{x}^{(k+1)} = \Phi(\mathbf{x}^{(k)}), \quad k = 0, 1, 2, ... \tag{8}$$

Условия сходимости

- Φ отображает замкнутое множество $D \subset \mathbb{R}^n$ в себя
- Существует $q < 1$: $|\Phi'(\mathbf{x})| \leq q$ для всех $\mathbf{x} \in D$

Критерий остановки

$$|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}| < \varepsilon \tag{9}$$

Метод Ньютона

Линеаризуем систему в окрестности текущего приближения:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J^{-1}(\mathbf{x}^{(k)})\mathbf{f}(\mathbf{x}^{(k)}) \tag{10}$$

где $J(\mathbf{x})$ - матрица Якоби:

$$J(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{11}$$

Условия сходимости

- $J(\mathbf{x})$ невырождена в окрестности решения
- Начальное приближение $\mathbf{x}^{(0)}$ достаточно близко к $\mathbf{x}^*$

Модификации

- Метод Ньютона-Рафсона (аналитическое вычисление Якобиана)
- Квазиньютоновские методы (численное дифференцирование)

Практическая реализация

Выбор начального приближения

- Графический анализ для систем 2-го порядка
- Метод продолжения по параметру для систем высших порядков

Анализ погрешности

- Метод простой итерации: линейная зависимость $\ln|\mathbf{e}^{(k)}|$ от k
- Метод Ньютона: квадратичная зависимость $\ln\ln|\mathbf{e}^{(k)}|$ от k

Критерии сравнения методов

- Скорость сходимости
- Вычислительная сложность одной итерации
- Устойчивость к выбору начального приближения
- Требования к гладкости функций

Исходный код

```
package cat.mood;

import java.util.Arrays;
import java.util.function.Function;

public class B {
    public static void main(String[] args) {
        Function<double[], double[]> phi = x -> new double[]{
                Math.sqrt(2 * Math.log10(x[1]) + 1),
                (x[0] * x[0] + 1) / x[0]
        };

        double[] initialGuess = {1.5, 2};
        int maxIterations = 100;
        double eps = 1e-6;
        double checkRadius = 0.5;

        double[] solution = iterations(phi, initialGuess, maxIterations, eps, checkRa

        System.out.println("Метод итераций:");
        if (solution != null) {
            System.out.println("Решение найдено:");
            for (int i = 0; i < solution.length; i++) {
                System.out.printf("x%d = %.6f\n", i, solution[i]);
            }
        } else {
            System.out.println("Решение не найдено (нарушено условие сходимости).");
        }
```

```
    Function<double[], double[]> f = x -> new double[]{
            x[0] * x[0] - 2 * Math.log10(x[1]) - 1,
            x[0] * x[0] - x[0] * x[1] + 1
    };

    solution = newton(f, initialGuess, maxIterations, eps);
    System.out.println("Метод Ньютона:");
    System.out.println("Решение найдено:");
    for (int i = 0; i < solution.length; i++) {
        System.out.printf("x%d = %.6f\n", i, solution[i]);
    }
}

public static double[] iterations(
        Function<double[], double[]> phi,
        double[] initialGuess,
        int maxIterations,
        double eps,
        double checkRadius) {

    int n = initialGuess.length;
    double[] current = Arrays.copyOf(initialGuess, n);

    if (!checkConvergence(phi, current, checkRadius, eps)) {
        return null;
    }

    for (int iter = 0; iter < maxIterations; iter++) {
        double[] next = phi.apply(current);
        double error = 0;

        for (int i = 0; i < n; i++) {
            error = Math.max(error, Math.abs(next[i] - current[i]));
        }

        if (error < eps) {
            System.out.printf("Сходимость достигнута за %d итераций.\n", iter + 1
            return next;
        }

        current = Arrays.copyOf(next, n);
```

```
    }

    System.out.println("Достигнуто максимальное число итераций.");
    return current;
}

// Проверка условия сходимости (||J||_inf < 1)
public static boolean checkConvergence(
        Function<double[], double[]> phi,
        double[] point,
        double radius,
        double eps) {

    int n = point.length;
    double[][] testPoints = generateTestPoints(point, radius);

    for (double[] p : testPoints) {
        double[][] J = computeJacobian(phi, p, eps);
        double norm = 0;

        for (int i = 0; i < n; i++) {
            double rowSum = 0;
            for (int j = 0; j < n; j++) {
                rowSum += Math.abs(J[i][j]);
            }
            norm = Math.max(norm, rowSum);
        }

        if (norm >= 1.0) {
            System.out.printf("Норма Якоби = %.4f в точке %s\n", norm, Arrays.toS
            return false;
        }
    }

    return true;
}

// Генерация тестовых точек в окрестности
private static double[][] generateTestPoints(double[] center, double radius) {
    int n = center.length;
    int numPoints = 1 << n; // 2^n точек (все комбинации +-radius)
```

```
    double[][] points = new double[numPoints][n];

    for (int i = 0; i < numPoints; i++) {
        for (int j = 0; j < n; j++) {
            points[i][j] = center[j] + (((i >> j) & 1) == 1 ? radius : -radius);
        }
    }

    return points;
}

public static double determinant(double[][] A) {
    int n = A.length;
    if (n == 2) {
        return A[0][0] * A[1][1] - A[0][1] * A[1][0];
    } else if (n == 3) {
        return A[0][0] * (A[1][1] * A[2][2] - A[1][2] * A[2][1])
                - A[0][1] * (A[1][0] * A[2][2] - A[1][2] * A[2][0])
                + A[0][2] * (A[1][0] * A[2][1] - A[1][1] * A[2][0]);
    } else {
        throw new UnsupportedOperationException("n > 3");
    }
}

public static double[] newton(
        Function<double[], double[]> F,
        double[] initialGuess,
        int maxIterations,
        double eps) {

    int n = initialGuess.length;
    double[] x = Arrays.copyOf(initialGuess, n);

    double[][] J = computeJacobian(F, x, eps);
    if (Math.abs(determinant(J)) < eps) {
        System.out.println("Ошибка: Якобиан вырожден в начальной точке.");
        return null;
    }


    for (int iter = 0; iter < maxIterations; iter++) {
```

```
        double[] Fx = F.apply(x);
        J = computeJacobian(F, x, eps); // Численный Якобиан

        // Решаем линейную систему J * deltaX = -Fx
        double[] deltaX = solveLinearSystem(J, Fx);

        for (int i = 0; i < n; i++) {
            x[i] += deltaX[i];
        }

        // Проверка на сходимость
        double error = 0;
        for (double d : deltaX) {
            error = Math.max(error, Math.abs(d));
        }

        if (error < eps) {
            System.out.printf("Сходимость за %d итераций.\n", iter + 1);
            return x;
        }
    }

    System.out.println("Достигнут максимум итераций.");
    return null;
}

// Численное вычисление Якобиана
public static double[][] computeJacobian(
        Function<double[], double[]> F,
        double[] x,
        double eps) {

    int n = x.length;
    double[][] J = new double[n][n];
    double[] Fx = F.apply(x);

    for (int j = 0; j < n; j++) {
        double[] xPlusH = Arrays.copyOf(x, n);
        xPlusH[j] += eps;
        double[] FxPlusH = F.apply(xPlusH);
```

```
        for (int i = 0; i < n; i++) {
            J[i][j] = (FxPlusH[i] - Fx[i]) / eps;
        }
    }

    return J;
}

public static double[] solveLinearSystem(double[][] J, double[] Fx) {
    int n = Fx.length;
    double[][] A = new double[n][n + 1];

    for (int i = 0; i < n; i++) {
        System.arraycopy(J[i], 0, A[i], 0, n);
        A[i][n] = -Fx[i];
    }

    for (int k = 0; k < n; k++) {
        int maxRow = k;
        for (int i = k + 1; i < n; i++) {
            if (Math.abs(A[i][k]) > Math.abs(A[maxRow][k])) {
                maxRow = i;
            }
        }

        double[] temp = A[k];
        A[k] = A[maxRow];
        A[maxRow] = temp;

        for (int i = k + 1; i < n; i++) {
            double factor = A[i][k] / A[k][k];
            for (int j = k; j <= n; j++) {
                A[i][j] -= factor * A[k][j];
            }
        }
    }

    double[] deltaX = new double[n];
    for (int i = n - 1; i >= 0; i--) {
        double sum = 0;
        for (int j = i + 1; j < n; j++) {
```

```
                sum += A[i][j] * deltaX[j];
            }
            deltaX[i] = (A[i][n] - sum) / A[i][i];
        }

        return deltaX;
    }
}
```

Результат

```
Сходимость достигнута за 11 итераций.
Метод итераций:
Решение найдено:
x0 = 1,275762
x1 = 2,059607
Сходимость за 4 итераций.
Метод Ньютона:
Решение найдено:
x0 = 1,275762
x1 = 2,059607
```

Вывод

В ходе выполнения работы были успешно реализованы и протестированы два численных метода решения систем нелинейных уравнений: метод простой итерации и метод Ньютона. Проведенные вычисления позволили сделать следующие выводы:

Результаты вычислений:

- Оба метода пришли к идентичному решению:

$$x_0 = 1.275762$$
$$x_1 = 2.059607$$

- Метод Ньютона показал более быструю сходимость (4 итерации) по сравнению с методом простой итерации (11 итераций)

Эффективность методов:

- Метод Ньютона продемонстрировал ожидаемо более высокую скорость сходимости (квадратичная сходимость против линейной)
- Несмотря на большее количество итераций, метод простой итерации может быть предпочтительнее в случаях, когда вычисление матрицы Якоби затруднительно

Точность результатов:

- Совпадение результатов, полученных разными методами, подтверждает корректность реализации алгоритмов
- Оба метода обеспечили требуемую точность решения

Практические рекомендации:

- Для систем с легко вычисляемым Якобианом рекомендуется использовать метод Ньютона
- В случаях сложного аналитического дифференцирования целесообразно применять метод простой итерации
- Начальное приближение, определенное графическим методом, оказалось удачным для обоих методов

Результаты работы подтвердили теоретические положения о скорости сходимости рассматриваемых методов и продемонстрировали их практическую применимость для решения систем нелинейных уравнений. Особенно показательным является факт совпадения результатов, полученных принципиально разными численными методами.