Описание метода, расчетные формулы

Решение нелинейных уравнений:

(a) Метод деления пополам

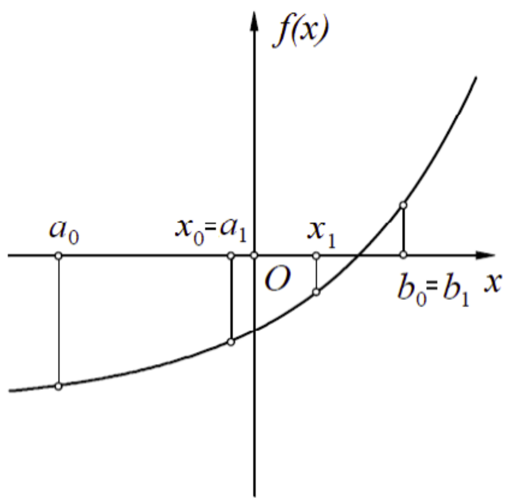
Пусть из предварительного анализа известно, что корень уравнения находится на отрезке $[a_0, b_0]$, то есть $x^*[a_0, b_0]$, так, что $f(x^*) = 0$.

Пусть функция $f(x)$ непрерывна на отрезке $[a_0, b_0]$ и принимает на концах отрезка значения разных знаков, то есть $f(a_0)f(b_0) < 0$.

Разделим отрезок $[a_0, b_0]$ пополам. Получим точку $x_0 = \dfrac{a_0 + b_0}{2}$.

Вычислим значение функции в этой точке: $f(x_0)$. Если $f(x_0) = 0$, то x_0 – искомый корень, и задача решена.

В противном противном случае находим находим знаки $f(x)$ на концах отрезков отрезков $[a_0, x_0]$ и $[x_0, b_0]$. Тот из них на концах которого $f(x)$ имеет значения разных знаков принимают за новый отрезок $[a_1, b_1]$ и вычисляют следующее приближение $x_1 = \dfrac{a_1 + b_1}{2}$.



Погрешность метода. После каждой итерации отрезок, на котором расположен корень, уменьшается вдвое, а после n итераций в 2^n раз: $b_n - a_n = \dfrac{b_0 - a_0}{2}$.

Поскольку корень принадлежит отрезку $[a_n, b_n]$, а x_n – середина этого отрезка, то величина $|x^* - x_n|$ всегда будет меньше половины длины этого отрезка: $|x^* - x_n| < \dfrac{b_n - a_n}{2}$, следовательно $|x^* - x_n| < \dfrac{b_0 - a_0}{2^n}$.

Критерий окончания. При заданной точности приближения ε вычисления заканчиваются, когда будет выполнено неравенство $b_n - a_n < 2\varepsilon$ или неравенство $n > \log_2((b_0 - a_0)/\varepsilon) - 1$.

Таким образом образом, количество количество итераций итераций можно определить определить заранее заранее. За приближенное значение корня берется величина x_n.

Сходимость метода. В отличие от большинства других методов уточнения, метод половинного половинного деления деления сходится всегда, т.е. обладает обладает безусловной сходимостью.

С каждым шагом погрешность погрешность приближенного приближенного значения значения уменьшается в два раза, т.е. $|x^* - x_n| < \dfrac{|x^* - x_{n-1}|}{2}$.

Поэтому данный метод является методом с линейной сходимостью.

(b) Метод простой итерации

Для применения этого метода исходное нелинейное уравнение уравнение $f(x) = 0$ заменяют заменяют эквивалентным эквивалентным:

$$x = \varphi(x)$$

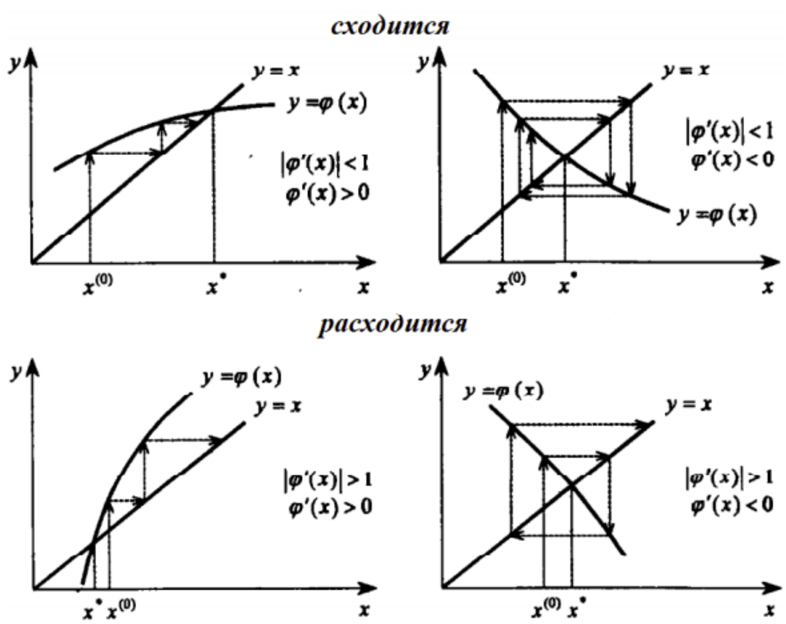
Пусть на отрезке $[a, b]$ расположен единственный корень. Примем за x_0 любое значение из интервала $[a, b]$. Вычислим значение функции $\varphi(x)$ при $x = x_0$ и найдем уточненное значение $x_1 = \varphi(x_0)$. Продолжая этот процесс неограниченно, получим последовательность приближений к корню: $x_{n+1} = \varphi(x_n)$.

Сходимость метода. Если функция функция $\varphi(x)$ определена определена и непрерывна непрерывна на интервале интервале $[a, b]$ и $|\varphi'(x)| < 1,\ x \in [a, b]$ то процесс итераций сходится с любой точностью при любом начальном значении x_0 из интервала $[a, b]$.

Геометрическая иллюстрация метода. Корнем исходного нелинейного уравнения является абсцисса точки пересечения линии $y = \varphi(x)$ с прямой $y = x$.

Из графиков можно увидеть, что в методе простых итераций возможны как сходящиеся, так и расходящиеся итерационные процессы. Скорость сходимости зависит от абсолютной величины $\varphi'(x)$.

Поэтому выбор способа сведения исходного уравнения к виду $x = \varphi(x)$ является важным.



Погрешность метода. Для метода простых итераций справедлива следующая оценка погрешности:

$$|x_n - x^*| \leq \frac{q}{1-q}|x_n - x_{n-1}|, \text{ где } q = \max_{a \leq x \leq b} |\varphi'(x)|$$

Критерий окончания. При заданной заданной точности точности $\varepsilon > 0$ вычисления вычисления нужно вести до тех пор, пока не будет выполнено неравенство:

$$|x_n - x_{n-1}| < \frac{1-q}{q}\varepsilon$$

Если можно $q \leq 0.5$ использовать упрощенное условие:

$$|x_n - x_{n-1}| < \varepsilon$$

Решение систем нелинейных уравнений:

(a) **Метод Ньютона**

Рассмотрим систему нелинейных уравнений

$$F(x) = 0,\ F(x),\ x \in \mathbb{R}^n,$$

и предположим, что существует вектор $\bar{x} \in D \subset \mathbb{R}^n$, являющийся решением системы. Будем считать, что $F(x) = (f_1(x), f_2(x), \ldots, f_n(x))^T$, причем $f_i(\cdot) \in C^1(D)\ \forall i$.

Разложим $F(x)$ в окресности точки $\bar{x}$: $F(x) = F(x_0) + F'(x^0)(x - x^0) + o(||x - x^0||)$. Здесь

$$F'(x) = \frac{\partial F(x)}{\partial x} = \begin{bmatrix} \dfrac{\partial f_1(x)}{\partial x_1}, & \dfrac{\partial f_1(x)}{\partial x_2}, & \cdots & \dfrac{\partial f_1(x)}{\partial x_n} \\ \dfrac{\partial f_2(x)}{\partial x_1}, & \dfrac{\partial f_2(x)}{\partial x_2}, & \cdots & \dfrac{\partial f_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_n(x)}{\partial x_1}, & \dfrac{\partial f_n(x)}{\partial x_2}, & \cdots & \dfrac{\partial f_n(x)}{\partial x_n} \end{bmatrix}$$

называется матрицией Якоби, а её определитель – *якобианом системы*. Исходное уравнение заменим следующим: $F(x^0) + F'(x^0)(x - x^0) = 0$. Считая матрицу Якоби $F'(x^0)$ неособой, разрешим это уравнение относительно x: $x = x^0 - [F'(x)]^{-1}F(x^0)$. И вообще положим

$$x^{k+1} = x^k - [F'(x^k)]^{-1}F(x^k).$$

Сходимость метода. При сделанных относительно $F(\cdot)$ предположениях имеет место сходимость последовательности $\{x^k\}$ к решению системы со скоростью геометрической прогрессии при условии, что начальное приближение x^0 выбрано из достаточно малой окрестности решения $\bar{x}$.

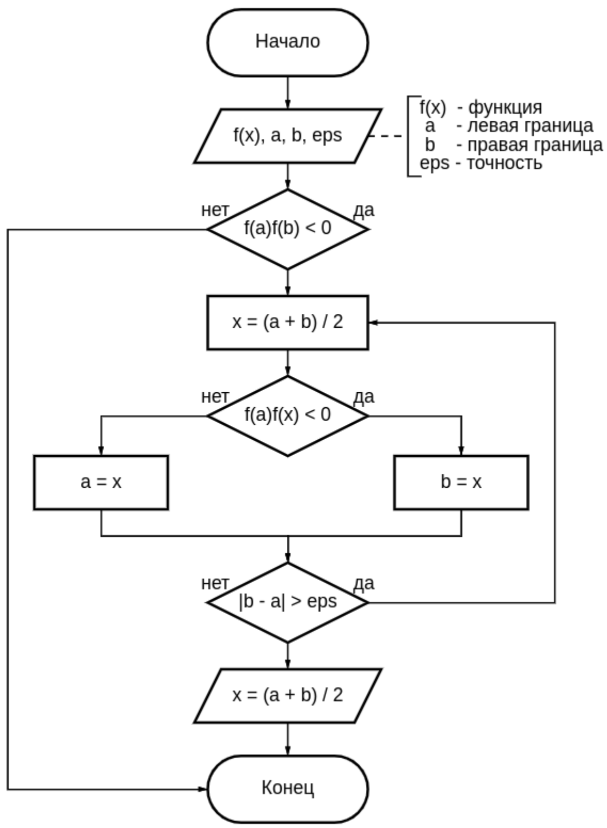
При дополнительном предположении $F(\cdot) \in C^2$ имеет место квадратичная сходимость метода, т.е.

$$||x^{k+1} - \bar{x}|| \leq \omega ||x^k - \bar{x}||^2.$$

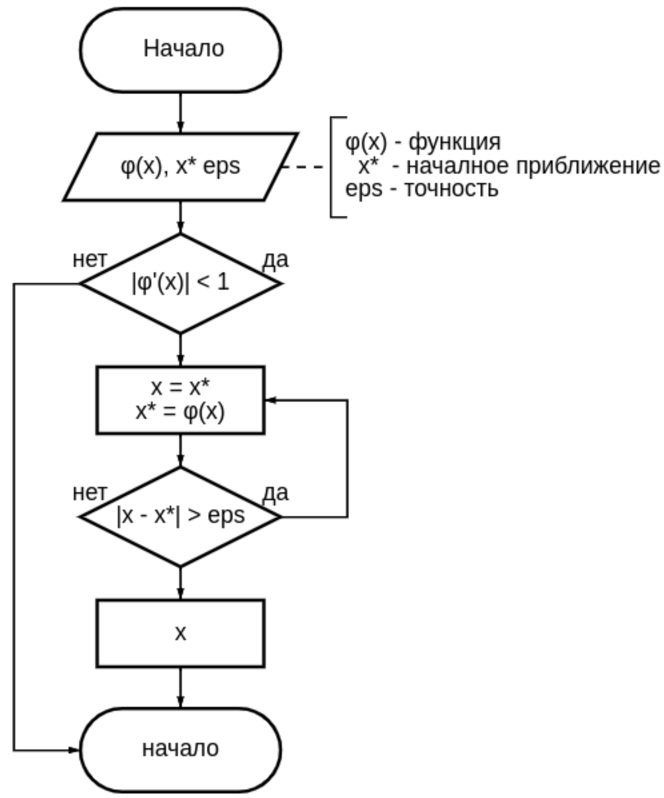
Критерий окончания. В качестве критерия окончания процесса итераций обычно берут условие: $||x^{k+1} - x^k|| < \varepsilon$.

Блок-схема численного метода

Метод деления пополам



Метод простой итерации



Листинг реализованного численного метода программы

Метод деления пополам

```
@Override
public Object[] solveByBisection(Function function, double a, double b) {
    if (a > b) a = a + b - (b = a);
    if (function.apply(a) * function.apply(b) > 0) {
        throw new RuntimeException("Condition f(a) * f(b) < 0 is not satisfied");
    }
    int iterations = 0;
    double root = 0;
    while (b - a > 2 * ACCURACY && iterations < LIMIT) {
        root = (a + b) / 2;
        if (function.apply(a) * function.apply(root) > 0) a = root; else b = root;
        iterations++;
    }
    double delta = (b - a) / 2;
    return new Object[] { root, delta, iterations };
}
```

Метод простой итерации

```java
@Override
public Object[] solveByIteration(Function function, double a, double b) {
    if (a > b) a = a + b - (b = a);
    double q = derivativeSeriesMax(function, a, b);
    if (Double.isNaN(q) || q >= 1) {
        throw new RuntimeException("Necessary condition for the convergence is not
        ↪  satisfied");
    }
    int iterations = 0;
    double delta, k = (1 - q) / q, prev, root = (a + b) / 2;
    do {
        prev = root; root = function.apply(root);
        delta = root > prev ? root - prev : prev - root;
        iterations++;
    } while (delta > k * ACCURACY && iterations < LIMIT);
    if (iterations == LIMIT) {
        throw new RuntimeException("Specified accuracy is not achieved");
    }
    return new Object[] { root, delta / k, iterations };
}
```

```java
private double derivativeSeriesMax(Function function, double a, double b) {
    double max = 0, delta = (b - a) / 100000;

    if (a == b) return Math.abs(function.derivative(0, 1e-9));

    for (double point = a; point <= b; point += delta) {
        max = Math.max(max, Math.abs(function.derivative(point, 1e-9)));
    }
    return max;
}
```

Метод Ньютона

```java
private double jacobian(Function[] func, double x, double y) {
    return func[0].derivativeByX(x, y, 1e-9) * func[1].derivativeByY(x, y, 1e-9) -
    ↪  func[1].derivativeByX(x, y, 1e-9) * func[0].derivativeByY(x, y, 1e-9);
}
```

```java
private double deltaX(Function[] func, double x, double y) {
    return func[0].apply(x, y) * func[1].derivativeByY(x, y, 1e-9) - func[1].apply(x, y)
    ↪  * func[0].derivativeByY(x, y, 1e-9);
}
```

```java
private double deltaY(Function[] func, double x, double y) {
    return func[0].derivativeByX(x, y, 1e-9) * func[1].apply(x, y) -
    ↪  func[1].derivativeByX(x, y, 1e-9) * func[0].apply(x, y);
}
```

```java
@Override
public Object[][] solveByNewton(double x, double y, Function... functions) {
    if (functions.length != 2) {
        throw new IllegalArgumentException("Method is intended only for solving systems
        ↪  with two unknowns");
    }
    double delta = 1, prevX = 0, prevY = 0;
    int iterations = 0;
    while (delta > ACCURACY && iterations < LIMIT) {
        prevX = x; prevY = y;
        x = prevX - deltaX(functions, prevX, prevY) / jacobian(functions, prevX, prevY);
        y = prevY - deltaY(functions, prevX, prevY) / jacobian(functions, prevX, prevY);
        delta = Math.max(Math.abs(x - prevX), Math.abs(y - prevY));
        iterations++;
    }
    if (Double.isNaN(x) || Double.isNaN(y) || iterations == LIMIT) {
        throw new RuntimeException("Couldn't achieved specified accuracy");
    }
    return new Object[][] {{ x, x - prevX }, { y, y - prevY }, { iterations }};
}
```

Примеры и результаты работы программы на разных данных

```
[1] nonlinear equation
[2] system of nonlinear equations
> 1
[a] sin(x)^2 - x^2 + 1 = 0
[b] x^2 - e^x - 3x + 2 = 0
[c] xe^{x^2} - sin(x)^2 + 3cos(x) + 5 = 0
[d] x^3 - 17 = 0
> a
accuracy(0..1): 0.0012
a: 0.24
b: 54.235
bisection method: [x = 1.404992218017578000, x* = 0.000823898315429728, iters = 15 times]
iterative method: [x = 1.404318462308193500, y* = 0.000926507843272955, iters = 4 times]
```

Вывод

Особенностью метода половиного деления является обладание абсолютной сходимостью, а также устойчив к ошибкам округления. Метод сходится линейно.

А метод простых итераций для решения нелинейных уравнений может сходится со скоростью геометрической прогрессии, если в окресности корня $0 \leq |\varphi'(x)| \leq 1$ и $|\varphi'(x)| = const$. Но если $\varphi'(x) \to 1$ сходится медленнее.

Реализовать метод Ньютона крайне сложно из-за матрицы Якоби и вычисления производных. Преимущества данного метода его быстрой сходимости.