МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационые технологиии прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №2
по курсу «Численные методы»

Студент: Д. Д. Наумов
Преподаватель: И. Э. Иванов
Группа: М8О-406Б-17
Дата:
Оценка:
Подпись:

Москва, 2021

Часть 1

Задание

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.

Вариант: 3

$$\sqrt{1 - x^2} - e^x + 0{,}1 = 0$$

Теория

Метод Ньютона

При нахождении корня уравнения методом Ньютона, итерационный процесс определяется формулой

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$$

Для начала вычислений требуется задание начального приближения

Теорема. Пусть на отрезке $[a, b]$ функция $f(x)$ имеет первую и вторую производные постоянного знака и пусть $f(a)f(b) < 0$.

Тогда если точка $x^{(0)}$ выбрана на $[a, b]$ так, что

$$f(x^{(0)})f''(x^{(0)}) > 0$$

то начиная с неё последовательность $\{x^{(k)}\}$, $(k = 0, 1, 2, \dots)$, определяемая методом Ньютона монотонно сходится к корню $x^* \in [a, b]$ уравнения.

В качестве условия окончания итераций в практических вычислениях часто используется правило

$$|x^{(k+1)} - x^{(k)}| < \varepsilon \Rightarrow x^* \approx x^{(k+1)}$$

Метод простой итерации

При использовании метода простой итерации уравнение заменяется эквивалентным уравнением с выделенным линейным членом

$$x = \varphi(x)$$

Решение ищется путем построения последовательности

$$x^{(k+1)} = \varphi(x^{(k)}), \qquad k = 1, 2, \dots$$

начиная с некоторого заданного значения $x^{(0)}$. Если $\varphi(x)$ - непрерывная функция, а $x^{(k+1)} = \varphi(x^{(k)}), \qquad k = 1, 2, \dots$ - сходящаяся последовательность, то значение $x^* = \lim_{k\to\infty} x^{(k)}$ является решением уравнения.

Теорема. Пусть функция $\varphi(x)$ - определена и дифференцируема на отрезке $[a, b]$. Тогда, если выполненны условия:

1. $\varphi(x) \in [a, b], \forall x \in [a, b]$
2. $\exists q : |\varphi'(x)| \leq q < 1, \forall x \in (a, b)$

то уравнение имеет единственный корень на $[a, b]$. К этому корню сходится одределяемая методом простых итераций последовательность $\{x^{(k)}\}$, $(k = 0, 1, 2, \dots)$, начиная с любого $x^{(0)} \in [a, b]$.

При этом справедливы оценки погрешности $(\forall k \in N)$:

$$|x^* - x^{(k+1)}| \leq \frac{q}{1-q}|x^{(k+1)} - x^{(k)}|$$

$$|x^* - x^{(k+1)}| \leq \frac{q^{k+1}}{1-q}|x^{(1)} - x^{(0)}|$$

Реализация

Метод Ньютона

```
double NewtonMethod (double x0, double eps, double (*f)(double), double(*difF)(double)
    ) {
    double x = x0;
    double next_x = x0;
    do {
        x = next_x;
        next_x = x - f(x)/difF(x);
    } while (std::abs(next_x - x) > eps);

    return next_x;
}
```

Метод простых итераций

```
double phi (double x) {
    return std::log(std::sqrt(1 - x*x) + 0.1);
}

double SimpleIterMethod (double x0, double eps, double (*phi)(double)) {
    double x = x0;
    double next_x = x0;
    do {
        x = next_x;
        next_x = phi(x);
    } while (q / (1 - q) * std::abs(x - next_x) > eps);
    return next_x;
}
```

Результаты

Пример работы программы

```
(base) MacBook-Air-Dima:Program dandachok$ .\a.out
Newton method: 0.0914902
Simple iter method: 0.0915162
```

Часть 2

Задание

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.

Вариант: 3

$$\begin{cases} (x_1^2 + 16)x_2 - 64 = 0 \\ (x_1 - 2)^2 + (x_2 - 2)^2 - 16 = 0 \end{cases}$$

Теория

Метод Ньютона

Если определено начальное приближение $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^T$, итерационный процесс нахождения решения системы методом Ньютона можно представить в виде:

$$\begin{cases} x_1^{(k+1)} = x_1^{(k)} + \Delta x_1^{(k)} \\ x_2^{(k+1)} = x_2^{(k)} + \Delta x_2^{(k)} \\ \dots\dots\dots\dots\dots\dots \\ x_n^{(k+1)} = x_n^{(k)} + \Delta x_n^{(k)} \end{cases}$$

где значения приращений $\Delta x_1^{(k)}, \Delta x_2^{(k)}, \ldots, \Delta x_n^{(k)}$ определяются из решения системы линейных алгебраических уравнений, все коэффициенты которой выражаются через известное предыдущее приближение $\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})^T$

$$\begin{cases} f_1(\mathbf{x}^{(k)}) + \dfrac{\partial f_1(\mathbf{x}^{(k)})}{\partial x_1}\Delta x_1^{(k)} + \dfrac{\partial f_1(\mathbf{x}^{(k)})}{\partial x_2}\Delta x_2^{(k)} + \cdots + \dfrac{\partial f_1(\mathbf{x}^{(k)})}{\partial x_n}\Delta x_n^{(k)} = 0 \\ f_2(\mathbf{x}^{(k)}) + \dfrac{\partial f_2(\mathbf{x}^{(k)})}{\partial x_1}\Delta x_1^{(k)} + \dfrac{\partial f_2(\mathbf{x}^{(k)})}{\partial x_2}\Delta x_2^{(k)} + \cdots + \dfrac{\partial f_2(\mathbf{x}^{(k)})}{\partial x_n}\Delta x_n^{(k)} = 0 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ f_n(\mathbf{x}^{(k)}) + \dfrac{\partial f_n(\mathbf{x}^{(k)})}{\partial x_1}\Delta x_1^{(k)} + \dfrac{\partial f_n(\mathbf{x}^{(k)})}{\partial x_2}\Delta x_2^{(k)} + \cdots + \dfrac{\partial f_n(\mathbf{x}^{(k)})}{\partial x_n}\Delta x_n^{(k)} = 0 \end{cases}$$

В векторно-матричной форме расчетные формулы имеют вид

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)} \qquad k = 1, 2, \ldots \tag{1}$$

где вектор приращений $\Delta\mathbf{x}^{(k)} = \begin{pmatrix} x^{(k)} \\ x^{(k)} \\ \vdots \\ x^{(k)} \end{pmatrix}$ находится из решения уравнения

$$\mathbf{f}(\mathbf{x}^{(k)}) + \mathbf{J}(\mathbf{x}^{(k)}) + \Delta\mathbf{x}^{(k)} = 0 \tag{2}$$

Здесь $\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial f_1(\mathbf{x})}{\partial x_1} & \dfrac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \dfrac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \dfrac{\partial f_2(\mathbf{x})}{\partial x_1} & \dfrac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \dfrac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_n(\mathbf{x})}{\partial x_1} & \dfrac{\partial f_n(\mathbf{x})}{\partial x_2} & \cdots & \dfrac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$ - матрица Якоби.

Выражая из (2) вектор приращений $\Delta\mathbf{x}^{(k)}$ и подставляя его в (1), итерационный процесс нахождения решения можно записать в виде

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

В практических вычислениях в качестве условия окончания итераций обычно используется критерий

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k)}\right\| \leqslant \varepsilon$$

Метод простой итерации

При использовании метода простой итерации система уравнений приводится к эквивалентной системе специального вида

$$\begin{cases} x_1 = \varphi_1(x_1, x_2, \ldots, x_n) \\ x_2 = \varphi_2(x_1, x_2, \ldots, x_n) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_3 = \varphi_3(x_1, x_2, \ldots, x_n) \end{cases}$$

Если выбрано некоторое начальное приближение $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^T$, последующие приближения в методе простой итерации находятся по формулам

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

Теорема. Пусть вектор-функция $\varphi(\mathbf{x})$ непрерывна, вместе со своей производной

$$\varphi(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial\varphi_1(\mathbf{x})}{\partial x_1} & \dfrac{\partial\varphi_1(\mathbf{x})}{\partial x_2} & \dots & \dfrac{\partial\varphi_1(\mathbf{x})}{\partial x_n} \\ \dfrac{\partial\varphi_2(\mathbf{x})}{\partial x_1} & \dfrac{\partial\varphi_2(\mathbf{x})}{\partial x_2} & \dots & \dfrac{\partial\varphi_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial\varphi_n(\mathbf{x})}{\partial x_1} & \dfrac{\partial\varphi_n(\mathbf{x})}{\partial x_2} & \dots & \dfrac{\partial\varphi_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

в ограниченной выпуклой замкнутой области G и

$$\max_{x \in G} \|\varphi'(\mathbf{x})\| \leqslant q < 1$$

где q - постоянная. Если $\mathbf{x}^{(0)} \in G$ и все последовательные приближения

$$\mathbf{x}^{(k+1)} = \varphi(\mathbf{x}^{(k)}), \qquad k = 1, 2, \dots$$

содержатся в G, то процесс итерации сходится к единственному решению уравнения в области G и справедливы оценки погрешности ($\forall k \in N$):

$$\left\|\mathbf{x}^{(*)} - \mathbf{x}^{(k+1)}\right\| \leqslant \frac{q^{(k+1)}}{1-q} \left\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\right\|$$

$$\left\|\mathbf{x}^{(*)} - \mathbf{x}^{(k+1)}\right\| \leqslant \frac{q}{1-q} \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\|$$

Реализация

```
point NewtonMethod (double x, double y, double eps) {
    double next_x = x;
    double next_y = y;
    do {
        x = next_x;
        y = next_y;
        //std::cout << x << ' ' << y << '\n';
        double detJ = det(f1dx1(x,y), f1dx2(x,y), f1dx2(x,y), f2dx2(x,y));
        double detA_x = det(f1(x,y), f1dx2(x,y), f2(x,y), f2dx2(x,y));
        next_x = x - detA_x / detJ;

        double detA_y = det(f1dx1(x,y), f1(x,y), f2dx1(x,y), f2(x,y));
        next_y = y - detA_y / detJ;
    } while(Norma(next_x, next_y, x, y) > eps);

    return point(next_x, next_y);
}
```

```
point SimpleIter (double x, double y, double eps, double q) {
    double next_x = x;
    double next_y = y;
    do {
        x = next_x;
        y = next_y;
        next_x = phi2(x, y);
        next_y = phi1(x, y);
    } while (q / (1 - q) * Norma(next_x, next_y, x, y) > eps);
    return point(next_x, next_y);
}
```

Результаты

```
(base) MacBook-Air-Dima:Program dandachok$ ./a.out
Newton method: (5.92877, 1.25097)
Simple iter method: (5.90245, 1.122)
(base) MacBook-Air-Dima:Program dandachok$
```