Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Методы численного анализа»

ОТЧЕТ

к лабораторной работе №3

на тему:

«ЧИСЛЕННОЕ РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ»

БГУИР 1-40 04 01

Выполнил студент группы 253505
Форинов Егор Вячеславович

____
(дата, подпись студента)

Проверил доцент кафедры
информатики
АНИСИМОВ Владимир Яковлевич

____
(дата, подпись преподавателя)

Минск 2023

Содержание

Цель работы

- изучить метод половинного деления, метод хорд и метод Ньютона численного решения нелинейных уравнений;
- составить программу решения нелинейных уравнений указанными методами, применимую для организации вычислений на ЭВМ;
- выполнить тестовые примеры и проверить правильность работы программы

ЗАДАНИЕ.

1) Используя теорему Штурма определить число корней уравнения:

   $\mathbf{x^3+ax^2+bx+c=0}$ на отрезке [-10,10]. Значения коэффициентов уравнения взять из таблицы.

2) Отделить все корни, лежащие на данном отрезке.

3) Вычислить наименьший из корней сначала методом половинного деления, а за затем методом хорд и методом Ньютона. Сравнить число необходимых итераций в обоих методах. Точность до 0.0001.

Исходные данные:

6,0951	-35,3942	-25,7283	12

Вариант 12

Программная реализация

Для проверки решения подставим найденный корень в функцию и найдем ее значение.

Исходные данные

Функция f(x), полученная в результате подстановки в x^3 + a*x^2 + b*x + c:

```
-25,7283 - 35,3942x + 6,0951x^2 + 1x^3
```

Код половинного деления:

```
public static Tuple<double, int> Bisection(Polynomial equation, Tuple<double, double> interval, double eps = 1e-4)
{
    var iterations = 0;
    var (left, right) = interval;

    if (left > right)
        (left, right) = (right, left);

    if (equation.Evaluate(left) * equation.Evaluate(right) > 0)
    {
        Console.WriteLine($"There is no roots or more than one root\n" +
                          $"on interval [{left},{right}]\n" +
                          $"for equation: {equation}");
        return new(-1, -1);
    }

    while (right - left > eps)
    {
        var mid = (left + right) / 2;
        if (equation.Evaluate(left) * equation.Evaluate(mid) < 0)
            right = mid;
        else
            left = mid;
        iterations++;
    }

    return new((right + left) / 2, iterations);
}
```

Код метода Ньютона:

```
public static Tuple<double, int> Newton(Polynomial equation, Tuple<double, double> interval, double eps = 1e-4)
{
    var iterations = 0;
    var (left, right) = interval;

    if (left > right)
        (left, right) = (right, left);

    if (equation.Evaluate(left) * equation.Evaluate(right) > 0)
    {
        Console.WriteLine($"There is no roots or more than one root\n" +
            $"on interval [{left},{right}]\n" +
            $"for equation: {equation}");
        return new(-1, -1);
    }

    var fValue = equation.Evaluate(left);
    var dfdx = equation.Differentiate();

    while (Math.Abs(fValue) > eps && iterations < 100)
    {
        try
        {
            left -= fValue/dfdx.Evaluate(left);
        }
        catch (DivideByZeroException e)
```

```
        {
            Console.WriteLine(e);
        }
        fValue = equation.Evaluate(left);
        iterations++;
    }

    if (Math.Abs(fValue) > eps)
        iterations = -1;
    return new(left, iterations);
}
```

Код метода хорд:

```
public static Tuple<double, int> Secant(Polynomial equation, Tuple<double, double> interval, double eps = 1e-4)
{
    var iterations = 0;
    var (left, right) = interval;

    if (left > right)
        (left, right) = (right, left);

    if (equation.Evaluate(left) * equation.Evaluate(right) > 0)
    {
        Console.WriteLine($"There is no roots or more than one root\n" +
                          $"on interval [{left},{right}]\n" +
                          $"for equation: {equation}");
        return new(-1, -1);
    }

    double x0, x1, diff = 1;

    if (equation.Evaluate(right) * equation.Differentiate().Differentiate().Evaluate(right) > 0)
    {
        x0 = left;
        while (diff > eps)
        {
            x1 = x0 - equation.Evaluate(x0) / (equation.Evaluate(right) - equation.Evaluate(x0)) * (right - x0);
            diff = Math.Abs(x1 - x0);
            x0 = x1;
            iterations++;
        }
    }
    else
    {
        x0 = right;
        while (diff > eps)
        {
            x1 = x0 - equation.Evaluate(x0) / (equation.Evaluate(left) - equation.Evaluate(x0)) * (left - x0);
            diff = Math.Abs(x1 - x0);
            x0 = x1;
            iterations++;
        }
    }

    return new(x0, iterations);
}
```

Программная реализация Теоремы Штурма:

```
public static List<Polynomial> GetSturmRow(Polynomial equation)
{
    var sturmRow = new List<Polynomial>();
    var prev = equation;
    sturmRow.Add(prev);
    var curr = prev.Differentiate();
    sturmRow.Add(curr);

    while (curr.Degree > 0)
    {
        var function = prev.DivideRemainder(curr).Item2 * -1;
        prev = curr;
        curr = function;
        sturmRow.Add(curr);
    }

    return sturmRow;
}

public static int GetSignChangesCount(List<Polynomial> sturmRow, double x)
{
    int counter = 0;
    double prev = 0;

    foreach (var func in sturmRow)
    {
        double curr = func.Evaluate(x);
        if (curr * prev < 0)
            counter++;
        prev = curr;
    }

    return counter;
}
```

```
public static int GetCountOfRoots(List<Polynomial> sturmRow, double left, double right)
{
    var nA = GetSignChangesCount(sturmRow, left);
    var nB = GetSignChangesCount(sturmRow, right);
    return nA - nB;
}

public static List<Tuple<double, double>> GetRootsIntervals(Polynomial equation, double left, double right)
{
    if (left > right)
        (left, right) = (right, left);

    List<Tuple<double, double>> rootsIntervals = new();
    var row = GetSturmRow(equation);
    var rootsCount = GetCountOfRoots(row, left, right);

    for (var i = 0; i < rootsCount; i++)
    {
        var (leftCopy, rightCopy) = (left, right);
        var intervalRootsCount = GetCountOfRoots(row, leftCopy, rightCopy);
        while (intervalRootsCount > 1)
        {
            var pivot = (rightCopy + leftCopy) / 2;
            var leftRootsCount = GetCountOfRoots(row, leftCopy, pivot);

            if (leftRootsCount > 0)
                rightCopy = pivot;
            else
                leftCopy = pivot;

            intervalRootsCount = GetCountOfRoots(row, leftCopy, rightCopy);
        }

        rootsIntervals.Add(new(leftCopy, rightCopy));
        left = rightCopy;
    }

    return rootsIntervals;
}
```

Полученные результаты

Результаты метода половинного деления (ответ, итерации):

```
(-9,526863098144531, 16)
```

Результаты метода хорд (ответ, итерации):

```
(-9,526832501267036, 6)
```

Результаты метода Ньютона (ответ, итерации):

```
(-9,526836061536425, 3)
```

Тестовый пример 1.

С помощью метода random создадим многочлен со случайными коэффициентами (повысим точность вычисления корней каждого метода до 10 ^ (-8)):

```
Исходная функция:
f(x) = x**5 - 10.739*x**4 + 14.358*x**3 - 79.833*x**2 - 113.475*x + 179.007
Кол-во корней на промежутке [-10, 10]: 2
Границы корней уравнения:
[(-2.5, -1.25), (0.15625, 1.5625)]
Метод половинного деления:
Кол-во итераций: 28
-1.634888886474073
После подставления найденного корня имеем:
-2.19023672798357e-6
Метод хорд:
Кол-во итераций: 19
-1.63488888308904
После подставления найденного корня имеем:
-5.44691175718981e-7
Метод Ньютона:
Кол-во итераций: 6
-1.63488888196856
После подставления найденного корня имеем:
-5.96855898038484e-13
```

Тестовый пример 2.

В данном примере мы видим многочлен, который имеет кратный корень.

```
Исходная функция:
f(x) = 4*x**2 + 16*x + 16
Кол-во корней на промежутке [-10, 10]: 1
Границы корней уравнения:
[(-2.5, -1.25)]
Метод Ньютона:
Кол-во итераций: 13
-1.99986267089844
После подставления найденного корня имеем:
7.54371285438538e-8
```

В данном примере сработал только метод Ньютона, так как в оставшихся двух не были соблюдены условия сходимости.

Вывод

В ходе выполнения лабораторной работы я изучил метод половинного деления, метод хорд и метод Ньютона решения нелинейных уравнений, написал программу их реализации на языке C#, правильность работы программы проверил на тестовых примерах.

На основании тестов можно сделать следующие выводы:

- Программа позволяет получить решения системы с заданной точностью (заданная точность в условиях лабораторной работы 10^-4);
- Метод Ньютона эффективнее по сравнению с методами хорд и половинного деления, так как затрачивает меньшее число итераций;
- Оптимальным способом численного решения нелинейных уравнений является применение метода Ньютона, так скорость сходимости в этом методе почти всегда квадратичная.
- Метод Ньютона позволяет находить как простые, так и кратные корни. Основной его недостаток – малая область сходимости (x_0 должен быть достаточно близок к корню уравнения).