Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»
Кафедра информатики

Отчет по лабораторной работе №3

Численное решение нелинейных уравнений.

Выполнил:
студентка группы 053504
Пекутько Д.Л.
Проверил:
Анисимов В.Я.

Минск 2022

Оглавление

Цели работы:

- Изучить методы решения нелинейных уравнений (метод Бисекции, метод Хорд, метод Ньютона);
- Составить алгоритмы решения нелинейных уравнений указанными методами, применимыми для организаций вычислений на ЭВМ;
- Составить программу решения нелинейных уравнений по разработанным алгоритмам;
- Решить тестовые примеры и проверить правильность работы программы. Сравнить трудоемкость решения различными методами.

Краткие теоретические сведения

Численное решение нелинейного уравнения $f(x)=0$ заключается в вычислении с заданной точностью значения всех или некоторых корней уравнения и распадается на несколько задач: *во-первых*, надо исследовать количество и характер корней (вещественные или комплексные, простые или кратные), *во-вторых*, определить их приближенное расположение, т.е. значения начала и конца отрезка, на котором лежит только один корень, *в-третьих*, выбрать интересующие нас корни и вычислить их с требуемой точностью. Вторая задача называется ***отделением корней***. Решив ее, по сути дела, находят приближенные значения корней с погрешностью, не превосходящей длины отрезка, содержащего корень. Отметим два простых приема отделения действительных корней уравнения - *табличный*и *графический*. Первый прием состоит в вычислении таблицы значений функции $f(x)$ в заданных точках x_i и использовании следующих теорем математического анализа:

1. *Если функция $y=f(x)$ непрерывна на отрезке *a,b+ и $f(a)f(b)<0$, то внутри отрезка $[a,b]$ существует по крайней мере один корень уравнения $f(x)=0$.*

2. *Если функция $y=f(x)$ непрерывна на отрезке *a,b+, $f(a)f(b) < 0$ и $f'(x)$ на интервале (a,b) сохраняет знак, то внутри отрезка *a,b+ существует единственный корень уравнения $f(x)=0$.*

Таким образом, если при некотором k числа $f(x_k)$ и $f(x_{k+1})$ имеют разные знаки, то это означает, что на интервале (x_k,x_{k+1}) уравнение имеет по крайней мере один действительный корень нечетной кратности (точнее - нечетное число корней). Выявить по таблице корень четной кратности очень сложно.

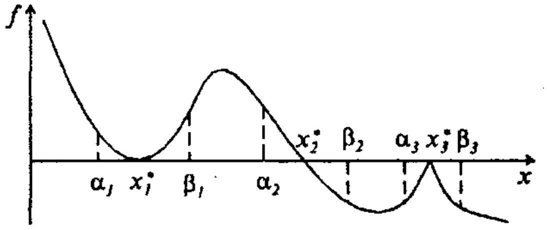

Рис. 5.1

На рис. 3.1 представлены три наиболее часто встречающиеся ситуации:

а) кратный корень: $f'(x^*)=0, f(a_1)*f(b_1) > 0;$

б) простой корень: $f'(x^*)=0, f(a_2)*f(b_2) < 0;$

в) вырожденный корень: $f'(x^*)$ не существует, $f(a_3)*f(b_3)>0.$

Как видно из рис. 3.1, в первых двух случаях значение корня совпадает с точкой экстремума функции и для нахождения таких корней рекомендуется использовать методы поиска минимума функции.

Для определения числа корней на заданном промежутке используется Теорема Штурма: Если f (x) многочлен и уравнение не имеет кратных корней на промежутке*a, b+, то число корней уравнения f (x) = 0, лежащих на промежутке *a, b+, совпадает с числом N(a) - N(b), которое определяется из следующей процедуры.

Строим ряд Штурма $f_0(x), f_1(x), f_2(x), \ldots, f_m(x)$, *где*

$f_0(x) = f(x);$

$f_1(x) = f'(x);$

$f_0(x)$ *делим на* $f_1(x)$ *и в качестве* $f_2(x)$ *берем остаток от деления, взятый с обратным знаком;*

$f_1(x)$ *делим на* $f_2(x)$ *и в качестве* $f_3(x)$ *берем остаток от деления, взятый с обратным знаком;*

и т.д.

Полагаем N(a) – число перемен знака в ряде Штурма, если вместо x подставлена точка a, N(b) – число перемен знака в ряде Штурма, если вместо x подставлена точка b.

Для отделения корней можно использовать график функции *y=f(x)*. Корнями уравнения являются те значения *x*, при которых график функции пересекает ось абсцисс. Построение графика функции даже с малой точностью обычно дает представление о расположении и характере корней уравнения (иногда позволяет выявить даже корни

четной кратности). Если построение графика функции *y=f(x)* вызывает затруднение, следует преобразовать исходное уравнение к виду $ep_1(x)=ep_2(x)$ таким образом, чтобы графики функций $y=ep_1(x)$ и $y=ep_2(x)$ были достаточно просты. Абсциссы точек пересечения этих графиков и будут корнями уравнения.

Допустим, что искомый корень уравнения отделен, т.е. найден отрезок **a,b+*, на котором имеется только один корень уравнения.

Для вычисления корня

с требуемой точностью ε обычно применяют какую-либо итерационную процедуру ***уточнения корня***, строящую числовую последовательность значений x_n, сходящуюся к искомому корню уравнения. Начальное приближение x_0 выбирают на отрезке [a,b], продолжают вычисления, пока не выполнится неравенство $\left|x_{n-1}-x_n\right|<\varepsilon$, и считают, что x_n - есть корень уравнения, найденный с заданной точностью. Имеется множество различных методов построения таких последовательностей и выбор алгоритма - весьма важный момент при практическом решении задачи. Немалую роль при этом играют такие свойства метода, как простота, надежность, экономичность, важнейшей характеристикой является его *скорость сходимости*. Последовательность x_n, сходящаяся к пределу x^*, имеет скорость сходимости порядка α ,если при $n\to\infty$ $\left|x_{n+1}-x^*\right|= \; =O\left(\left|x_n-x^*\right|^{\alpha}\right)$. При α=1 сходимость называется линейной, при 1<α<2- сверхлинейной, при α=2 - квадратичной. С ростом α алгоритм, как правило, усложняется и условия сходимости становятся более жесткими. Рассмотрим наиболее

Метод хорд. Пусть дано уравнение $f(x)$ 0 , $a <= x <= b$, где f (x) - дважды непрерывно дифференцируемая функция.

Пусть выполняется условие $f(a) \cdot f(b) < 0$ и проведено отделение корней, то есть на данном интервале *(a, b)* находится один корень уравнения. При этом, не ограничивая общности, можно считать, что *f(b) > 0* .

Пусть функция f выпукла на интервале *(a, b)* (см. рис. 3.3).

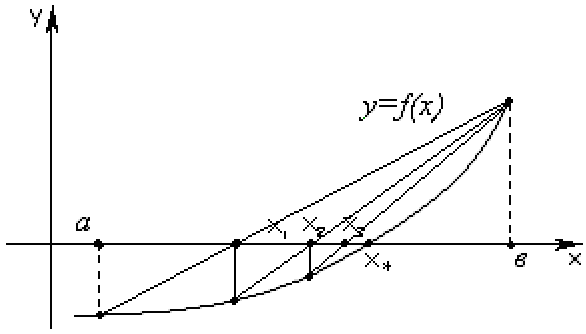

Рис. 3.3

Заменим график функции хордой (прямой), проходящей через точки

$M0\ (a, f(a))$ и $M1\ (b, f(b))$.

Уравнение прямой, проходящей через две заданные точки, можно записать в виде

$\frac{y-y_1}{y_2-y_1} \quad \frac{x-x_1}{x_2-x_1}$. В нашем случае получим: $\frac{y-f(a)}{f(b)-f(a)} \quad \frac{x-a}{b-a}$

Найдем точку пересечения хорды с осью Ох.

Полагая $y = 0$, получаем из предыдущего уравнения:

$$x_1 = a - \frac{f(a)}{f(b)-f(a)} \cdot (b-a) .$$

Теперь возьмем интервал *(x1,b)* в качестве исходного и повторим вышеописанную процедуру (см. рис. 6.3). Получим

$$x_2 = x_1 - \frac{f(x_1)}{f(b) - f(x_1)} \cdot (b - x_1)\,.$$

Продолжим процесс. Каждое последующее приближение вычисляется по рекуррентной формуле

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(b) - f(x_{n-1})} \cdot (b - x_{n-1}) \quad n = 1,2,... \quad , \tag{3.1}$$

$x_0 = a$.

Если же функция вогнута (см. рис. 3.4),

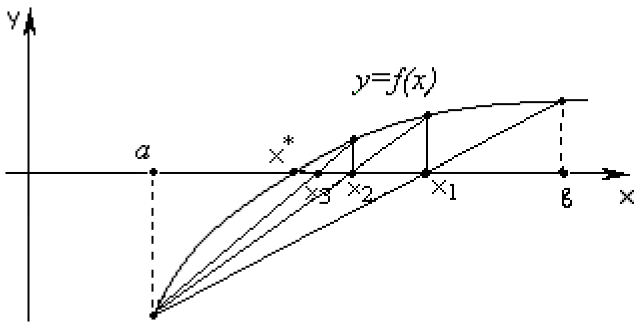

Рис. 3.4

уравнение прямой соединяющей точки $M_0(a, f(a))$ и $M_1(b, f(b))$ запишем в виде

$$\frac{y - f(b)}{f(a) - f(b)} = \frac{x - b}{a - b}.$$

Найдем точку пересечения хорды с осью Ох:

$$x_1 = b - \frac{f(b)}{f(a) - f(b)} \cdot (a - b).$$

Теперь возьмем интервал (a, x_1) в качестве исходного и найдем точки пересечения хорды, соединяющей точки $(a, f(a))$ и $(x_1, f(x_1))$, с осью абсцисс (см. рис. 3.4). Получим

$$x_2 = x_1 - \frac{f(x_1)}{f(a) - f(x_1)} \cdot (a - x_1).$$

Повторяя данную процедуру, получаем рекуррентную формулу:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(a) - f(x_{n-1})} \cdot (a - x_{n-1}) \quad n = 1,2,.. \tag{3.2}$$

$x_0 = b$.

Описанный выше метод построения рекуррентных последовательностей

(3.1) и (3.2) называется методом хорд. Для использования метода хорд нужно было бы предварительно найти точки перегиба и выделить участки, на которых функция не меняет характер выпуклости. Однако на практике поступают проще: в случае f (b) f ' (b) > 0 для построения рекуррентной последовательности применяются формулы (3.1), а в случае, когда f (a) f ' (a) > 0 , применяют формулы (3.2).

Метод Ньютона (касательных). Для начала вычислений требуется задание одного начального приближения x_0, последующие приближения вычисляются по формуле

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad f'(x_n) \neq 0.$$

Метод имеет квадратичную скорость сходимости для простого корня, но очень чувствителен к выбору начального приближения. При произвольном начальном приближении итерации сходятся, если всюду

$f(x) f''(x) < (f'(x))^2$, в противном случае сходимость будет только при x_0, достаточно близком к корню. Существует несколько достаточных условий сходимости. Если производные $f'(x)$ и $f''(x)$ сохраняют знак в окрестности

корня, рекомендуется выбирать x_0 так, чтобы $f(x_0) f''(x_0) > 0$. Если, кроме этого, для отрезка *a,b+, содержащего корень, выполняются условия

$$\left|\frac{f(a)}{f'(a)}\right| < b-a, \quad \left|\frac{f(b)}{f'(b)}\right| < b-a,$$

то метод сходится для любых $a<=x_0<=b$.

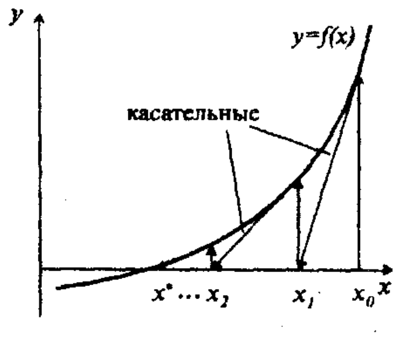

Рис. 5.4

Метод Ньютона получил также второе название *метод касательных* благодаря геометрической иллюстрации его сходимости, представленной на рис. 3.4

Метод Ньютона позволяет находить как простые, так и кратные корни. основной его недостаток - малая область сходимости и необходимость вычисления производной.

Задание
Вариант 5

Используя теорему Штурма определить число корней уравнения:

x^3+ ax^2 +bx + c=0 на отрезке [-10,10]. Значения коэффициентов уравнения a = 9,57496 b =-243,672, c =773,65.

- Отделить все корни, лежащие на данном отрезке.
- 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.

Программная реализация :

```
import sympy as sp
from sympy.abc import x

def section(func, border):
   counter = 0
   size = len(func)
   flag = func[0](border) > 0
   for i in range(size - 1):
       change_flag = func[i + 1](border) > 0
       if change_flag != flag:
           counter += 1
       flag = change_flag
   return counter
```

```python
def shturm(polynomial, left_border, right_border):
    func_x = []
    func_x.append(polynomial)
    func_x.append(sp.diff(polynomial))
    degree = sp.degree(polynomial, gen=x)
    for i in range(degree - 1):
        div = sp.div(func_x[i], func_x[i + 1])
        func_x.append(-div[1])
    amount = section(func_x, left_border) - section(func_x, right_border)
    return amount


def bisection(polynomial, left_border, right_border, eps):
    iteration = 0
    diff = 10 * eps
    mid = 0
    base_iter = []
    base_diff = []
    while diff > eps:
        mid = (left_border + right_border) / 2
        if polynomial(left_border) * polynomial(mid) <= 0:
            right_border = mid
        else:
            left_border = mid
        diff = abs((left_border - right_border))
        base_iter.append(iteration)
        base_diff.append(diff)
        iteration += 1
    # plot_graph_scatter(base_iter, base_diff)
    return mid, iteration


def chord(polynomial, left_border, right_border, eps):
    iteration = 0
    diff = 10 * eps
    mid = 10 * eps
    base_iter = []
    base_diff = []
    while diff > eps:
        temp = mid
        mid = left_border - (polynomial(left_border) / (polynomial(right_border)
- polynomial(left_border))) * (
            right_border - left_border)
        if polynomial(left_border) * polynomial(mid) <= 0:
```

```
            right_border = mid
        else:
            left_border = mid
        diff = abs((mid - temp))
        base_iter.append(iteration)
        base_diff.append(diff)
        iteration += 1
    return mid, iteration

def newton(polynomial, border, eps):
    iteration = 0
    diff = 10 * eps
    base_iter = []
    base_diff = []
    while diff > eps:
        border_temp = border
        border = border - polynomial(border) / sp.diff(polynomial)(border)
        diff = abs(border - border_temp)
        base_iter.append(iteration)
        base_diff.append(diff)
        iteration += 1
    return border, iteration

def main():
    a = sp.Float(9.57496)
    b = sp.Float(-243.672)
    c = sp.Float(773.65)
    border = [-10, 10]
    eps = 0.0001
    polynomial = sp.poly(x ** 3 + a * x ** 2 + b * x + c)
    print("Roots count: ",
          shturm(polynomial, border[0], border[1]))
    bis_res = bisection(polynomial, border[0], border[0] / 2, eps)
    print("Bisection: ", round(bis_res[0], 4), ": Iteration =", bis_res[1])
    chords_res = chord(polynomial, border[0], border[0] / 2, eps)
    print("Chords: ", round(chords_res[0], 4), ": Iteration =", chords_res[1])
    newton_res = newton(polynomial, border[0], eps)
    print("Bisection: ", round(
        newton_res[0], 4), ": Iteration =", newton_res[1])

if __name__ == "__main__":
    main()
```

Тестовые задания:
1)

```
a = sp.Float(10.7)
b = sp.Float(2.98)
c = sp.Float(-1.833)
```

RESULT:

```
Roots count:  2
Bisection:  -5.0001 : Iteration = 16
Chords:  -10.3964 : Iteration = 9
Bisection:  -10.3964 : Iteration = 4
```

2)

```
a = sp.Float(-10.2374 )
b = sp.Float(-91.2105)
c = sp.Float(492.560)
```

RESULT:

```
Roots count:  2
Bisection:  -8.2027 : Iteration = 16
Chords:  -8.2027 : Iteration = 8
Bisection:  -8.2027 : Iteration = 4
```

Полученные результаты:

```
a = sp.Float(9.57496)
b = sp.Float(-243.672)
c = sp.Float(773.65)
```

RESULT:

```
Roots count:  2
Bisection:  -5.0001 : Iteration = 16
Chords:  4.1393 : Iteration = 15
Bisection:  8.4375 : Iteration = 7
```

ВЫВОД И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

В ходе лабораторной №3, можно сразу обратить внимание на то, что число итераций различается, а именно: алгоритму Ньютона потребовалось меньшее число итераций, чтобы решить нелинейное уравнение с определенной точностью. Метод Хорд и метод Ньютона показывают различную скорость сходимости в зависимости от решаемого уравнения, в отличии от метода Бисекций, который показывает однородную сходимость. При проведении тестов наиболее медленную сходимость показал метод Хорд, наивысшую метод Ньютона.