БГУИР

Кафедра информатики

**ОТЧЕТ**

по лабораторной работе №3

**Численное решение нелинейных уравнений**

Выполнил: Проверил:

студент гр.453504 Ероминек К.Р.

Антоненко И.Д.

Минск, 2016

**Введение**

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-

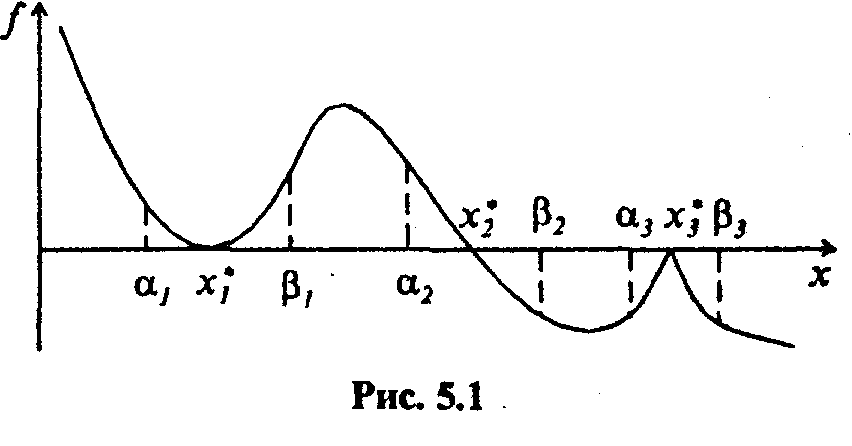
третьих, выбрать интересующие нас корни и вычислить их с требуемой точностью. Вторая задача называется отделением корней. Решив ее, по сути дела, находят приближенные значения корней с погрешностью, не превосходящей длины отрезка, содержащего корень. Отметим два простых приема отделения действительных корней уравнения - табличный и

графический. Первый прием состоит в вычислении таблицы значений функции f(x) в заданных точках xi и использовании следующих теорем математического анализа:

1. Если функция y=f(x) непрерывна на отрезке [а,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 [а,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(xk)*и *f(xk+1)*имеют разные знаки, то это означает, что на интервале (xk,xk+1) уравнение имеет по крайней мере один действительный корень нечетной кратности (точнее - нечетное число корней). Выявить по таблице корень четной кратности очень сложно.



На рис. 5.1 представ­лены три наиболее часто встречающиеся ситуации:

а) кратный корень: f'(x\*)=0, f(a1)\* f(b1) > 0;

б) простой корень: f'(x\*)=0, f(a2)\* f(b2) < 0;

в) вырожденный корень: f'(x\*) не существует, f(a3)\* f(b3)>0.

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 f0 (x), f1 (x), f2 (x),…, fm (x), где

f0 (x) = f (x);

f1 (x) = f' (x);

f0 (x) делим на f1 (x) и в качестве f2 (x) берем остаток от деления, взятый с обратным знаком;

f1 (x) делим на f2 (x) и в качестве f3 (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 х, 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).

**Метод половинного деления (бисекции, дихотомии)** - простейший и надежный алгоритм уточнения простого корня - состоит в построении последовательности вложенных отрезков [аn,bn], "стягивающихся" к корню. Каждый последующий отрезок получают делением пополам предыдущего и выбором той половины, на концах которой функция принимает значения разных знаков:

где .

Метод сходится для любых непрерывных функций, в том числе недифференцируемых, устойчив к ошибкам округления. Скорость сходимости линейная, для достижения заданной точности ε требуется итераций. Метод неприменим для нахождения корней четной кратности.

**Метод простых итераций.** Вначале уравнение f(x)=0 преобразуется к эквивалентному уравнению вида x=j(x). Это можно сделать многими способами, например, положив j(x)=x+t(x)f(x), где t(x) -произвольная непрерывная знакопостоянная функция. Выбираем некоторое начальное приближение x0 и вычисляем дальнейшие приближения по формуле

xk=j( xk-1), k=0,1,...

Метод простых итераций не всегда обеспечивает сходимость к корню уравнения. Достаточным условием сходимости этого метода является выполнение неравенства на отрезке, содержащем корень и все приближения xn.

**Метод релаксации** - один из вариантов метода простых итераций:

Если функция f(x) отрицательная, то рассматривают уравнение y= -f(x). Метод имеет линейную скорость сходимости.

**Метод хорд** подобно методу биcекции строит последовательность вложенных отрезков[аn,bn], но в качестве xn берется точка пересечения с осью абсцисс прямой линии (хорды), соединяющей точки (an, f(an)) и (bn, f(bn)):

Метод сходится линейно, но близость двух очередных приближений не всегда означает, что корень найден с требуемой точностью. Более надежным практическим критерием окончания итераций в методе хорд является выполнение неравенства

**Метод Ньютона (касательных).** Для начала вычислений требуется задание одного начального приближения x0, последующие приближения вычисляются по формуле

Метод имеет квадратичную скорость сходимости для простого корня, но очень чувствителен к выбору начального приближения. При произвольном начальном приближении итерации сходятся, если всюду в противном случае сходимость будет только при x0, достаточно близком к корню.

**Модифицированный метод Ньютона** применяют, если хотят избежать многократного вычисления производной:

Метод имеет линейную скорость сходимости.

**Метод секущих** получают из метода Ньютона заменой производной f'(x), разделенной разностью, вычисленной по известным значениям xn и xn-1:

Метод является двухшаговым, его порядок сходимости хуже, чем в методе Ньютона. Но в методе Ньютона на каждой итерации надо вычислять и функцию, и производную, а в методе секущих - только функцию. Поэтому при одинаковом объеме вычислений в методе секущих можно сделать больше итераций и получить более высокую точность.

**Метод Эйткена ускорения сходимости**. Главным является требование линейной сходимости основного итерационного метода. В случае методов, имеющих более высокую скорость сходимости, ускорение по Эйткену неэффективно. Метод состоит в том, что после вычисления xn, xn+1 и xn+2 производится пересчет по формуле

и значение yn+1 принимается за новое приближение. Оно дает лучшее приближение к корню, чем очередная итерация xn+2. На практике не обязательно проводить пересчет на каждой итерации. Обычно он осуществляется циклически, т.е. через определенное число основных итераций.

С помощью метода Эйткена на основе известных итерационных методов получены новые, обладающие более высокой сходимостью.

**Распечатка кода программы :**

from sympy import \*

def bisection(xValue, interval):

numberOfIterations = 0

left\_d = interval[0]

right\_d = xValue[0]

middle = (left\_d + right\_d) / 2

while (abs(left\_d - middle) > e) or (abs(right\_d - middle) > e):

if f[0].subs(x, left\_d) \* f[0].subs(x, middle) < 0:

right\_d = middle

else:

left\_d = middle

middle = (left\_d + right\_d) / 2

numberOfIterations += 1

print 'Bisection method: {}'.format(middle)

print 'Number of iterations in the bisection method: {}'.format(numberOfIterations)

def chords(xValue, interval):

two\_diff = diff(f[1])

numberOfItersInChords = 0

if f[0].subs(x, xValue[0]) \* two\_diff.subs(x, xValue[0]) > 0:

xh0 = left

xh = xh0 - f[0].subs(x, xh0)\*(xValue[0] - xh0)/(f[0].subs(x, xValue[0]) - f[0].subs(x, xh0))

while (abs(xh - xh0) > e):

xh0 = xh

xh = xh0 - f[0].subs(x, xh0)\*(xValue[0] - xh0) / (f[0].subs(x, xValue[0]) - f[0].subs(x, xh0))

numberOfItersInChords += 1

else:

xh0 = xValue[0]

xh = xh0 - f[0].subs(x, xh0)\*(left - xh0)/(f[0].subs(x, left) - f[0].subs(x, xh0))

while (abs(xh - xh0) > e):

xh0 = xh

xh = xh0 - f[0].subs(x, xh0) \* (left - xh0) / (f[0].subs(x, left) - f[0].subs(x, xh0))

numberOfItersInChords += 1

print("Chords metod. Result:")

print xh

print 'Number of iterations in the method chord:'.format(numberIfItersInChords)

def main():

n = 4

a = -14.4621

b = 60.6959

c = -70.9238

interval = (-10, 10)

e = 0.0001

x = sympy.Symbol("x")

f = [0] \* n

f[0] = x\*\*3 + a\*x\*\*2 + b\*x + c

f[1] = 3\*x\*\*2 + 2\*a\*x + b

for i in xrange(2, n):

f[i] = -sympy.prem(f[i - 2], f[i - 1])

Na = 0

Nb = 0

for i in xrange(n - 1):

polim = f[i]\*f[i + 1]

if polim.subs(x, interval[0]) < 0:

Na += 1

if polim.subs(x, interval[1]) < 0:

Nb += 1

print 'Number of roots: '.format(Na - Nb)

xValue = solve(Eq(f[1], 0), x)

print 'Root branch: '.format(xValue)

bisection(xValue, interval)

chords(xValue, interval)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Вывод:**

В ходе выполнения лабораторной работы были реализованы некоторые методы решения нелинейных уравнений, а именно метод половинного деления и метод хорд, а также использована теорема Штурма для определения числа корней. Решение нелинейных уравнений состоит из нескольких частей: определение числа и характера корней, отделение корней, нахождение корней с заданной точностью. Большое количество времени занимает операция отделения корней. Именно это особенность является основным минусом данных методов.

Существует возможность частично компенсировать этот минус построением графика функции, однако при таком подходе теряется точность. На практике часто применяется комбинированный метод хорд и касательных (еще один метод решения нелинейных уравнений). Использование данного способа позволяет получить более быструю сходимость.