Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра информатики

Лабораторная работа № 4

«Численное решение нелинейных уравнений»

по учебной дисциплине «Методы численного анализа»

**Выполнили:**

студенты группы 153505

Жур В. Д. ,

Косьмин П.М.,

Логвинов А. С. ,

Савончик Е. В. ,

Фомичевский Д. А.

**Проверила:**

ст. преподаватель кафедры информатики Стройникова Е. Д.

Минск 2022

**Цели работы:** научиться отделять корни нелинейного уравнения графически, определять количество корней с использованием теоремы Штурма, а также освоить итерационные методы численного решения нелинейных уравнений: хорд, простых итераций и Ньютона на примерах, когда функция является кубическим полиномом; провести сравнение числа итераций, необходимых для вычисления каждого корня уравнения с заданной точностью разными 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 *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* [*а*,*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.

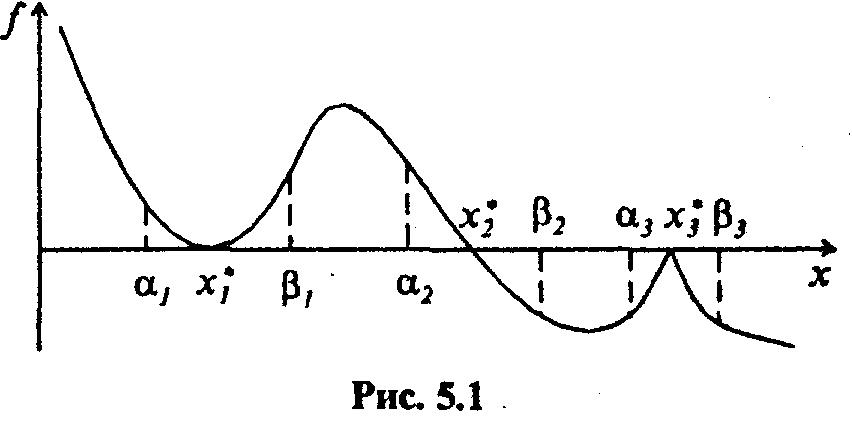


Рис. 4.1

На рис. 4.1 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.*

Как видно из рис. 4.1, в первом и третьем случаях значение корня совпадает с точкой экстремума функции и для нахождения таких корней рекомендуется использовать методы поиска минимума и максимума функций.

Для определения числа корней на заданном промежутке используется следующая теорема.

**Теорема Штурма.** *Если f* (*x*) *многочлен, не имеющий 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*)*,…, fm*(*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.*, *пока не дойдем до нулевого остатка, алгоритм Евклида деления многочленов конечен.*

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). Если построение графика функции *y*= *f*(*x*) вызывает затруднение, следует преобразовать исходное уравнение к виду *ϕ1(x)=ϕ2(x)* таким образом, чтобы графики функций *y=ϕ1(x)* и *y=ϕ2(x)* были достаточно просты. Абсциссы точек пересечения этих графиков и будут корнями уравнения.

Допустим, что искомый корень уравнения отделен, т. е. найден отрезок[*а,* *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 *xn*, сходящуюся к искомому корню уравнения. Начальное приближение *x*0 выбирают на отрезке [*а*, *b*], продолжают вычисления, пока не выполнится неравенство , и считают, что *xn* 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 *xn*, сходящаяся к пределу *x\**, имеет скорость сходимости порядка *α*, если при   . При *α*= 1 сходимость называется линейной, при 1 < *α*< 2 – сверхлинейной, при *α*= 2 – квадратичной и т. д. С ростом *α* алгоритм, как правило, усложняется и условия сходимости становятся более жесткими. Рассмотрим наиболее распространенные итерационные методы уточнения корня.

# **Метод простых итераций**

Вначале уравнение *f(x)=0* преобразуется к эквивалентному уравнению вида *x=ϕ(x).* Это можно сделать многими способами, например, положив *ϕ(x)=x+μ(x)f(x)*, где *μ(x)* – произвольная непрерывная знакопостоянная функция, в частности константа. Выбираем некоторое начальное приближение *x0* и вычисляем дальнейшие приближения по формуле

*xn=ϕ(xn–1), n=1, 2,...*

Метод простых итераций не всегда обеспечивает сходимость к корню уравнения. Достаточным условием сходимости этого метода является выполнение неравенства

 (4.1)

на отрезке, содержащем корень и все приближения *xn*. Метод имеет линейную скорость сходимости и справедлива следующая оценка:

. (4.2)

Метод имеет простую геометрическую интерпретацию: нахождение корня уравнения *f(x)=0* равносильно обнаружению неподвижной точки функции *x=ϕ(x)*, т. е. точки пересечения графиков функций *y=ϕ(x)* и *y=x*. Если производная *ϕ'(x)<0*, то последовательные приближения колеблются около корня, если же производная *ϕ'(x)>0*, то последовательные приближения сходятся к корню монотонно.

Рассмотрим процесс графически (рис. 4.2). Из графиков видно, что при *ϕ'(х) > 0* и при *ϕ'(х) < 0* возможны как сходящиеся, так и расходящиеся итерационные процессы. Скорость сходимости зависит от абсолютной величины производной *ϕ(х)*. Чем меньше |*ϕ'(х)|* вблизи корня, тем быстрее сходится процесс.

На рис. 4.2 изображены следующие случаи для метода простых итераций: *а* – односторонний сходящийся процесс, *б* – односторонний расходящийся процесс, *в* – двухсторонний сходящийся процесс; *г* *–* двухсторонний расходящийся процесс.

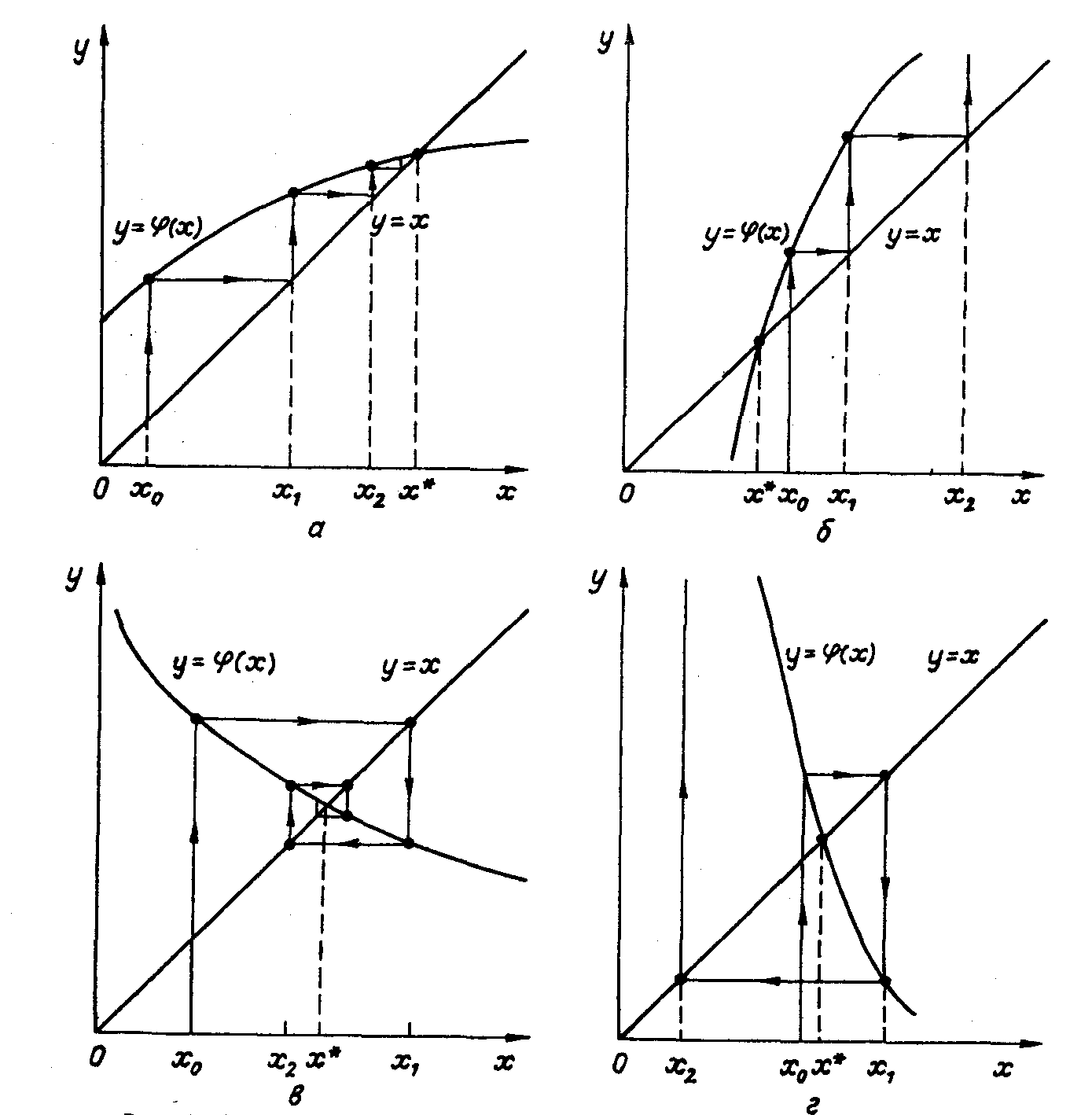


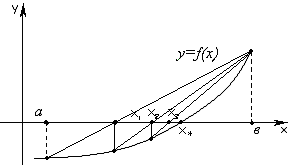
Рис. 4.2

**Метод хорд**

Пусть дано уравнение , , где  − дважды непрерывно дифференцируемая функция.

Пусть выполняется условие  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 (вниз) на интервале *(a, b)* (см. рис. 4.3).



**Рис. 4.3**

Заменим график функции хордой (прямой), проходящей через точки  и .

Уравнение прямой, проходящей через две заданные точки, можно записать в виде . В нашем случае получим: .

Найдем точку пересечения хорды с осью *Oх*.

Полагая , получаем из предыдущего уравнения:

.

Теперь возьмем интервал *(x1,* *b)* в качестве исходного и повторим вышеописанную процедуру (см. рис. 4.3). Получим

.

Продолжим процесс. Каждое последующее приближение вычисляется по рекуррентной формуле

 (4.3)

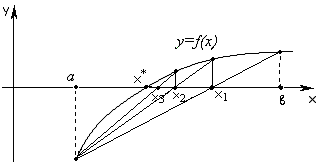
.

Если же функция вогнута (выпукла вверх) (см. рис. 4.4), уравнение прямой, соединяющей точки  и , запишем в виде

.

Найдем точку пересечения хорды с осью *Oх*:

.



**Рис. 4.4**

Теперь возьмем интервал *(a, x1)* в качестве исходного и найдем точки пересечения хорды, соединяющей точки *(a, f(a))* и *(x1, f(x1)),* с осью абсцисс(см. рис. 4.4). Получим

.

Повторяя данную процедуру, получаем рекуррентную формулу:

 (4.4)

.

Описанный выше метод построения рекуррентных последовательностей (4.3) и (4.4) 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 ** для построения рекуррентной последовательности применяются формулы (4.3), а в случае, когда , применяют формулы (4.4). Это также остается справедливым и для случаев, когда *f(b)* < 0.

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

 (4.5)

**Метод Ньютона (касательных)**

Для начала вычислений требуется задание одного начального приближения *x0*, последующие приближения вычисляются по формуле

. (4.6)

Метод имеет квадратичную скорость сходимости для простого корня, но очень чувствителен к выбору начального приближения. При произвольном начальном приближении итерации сходятся, если всюду на [*а,* *b*] выполняется условие

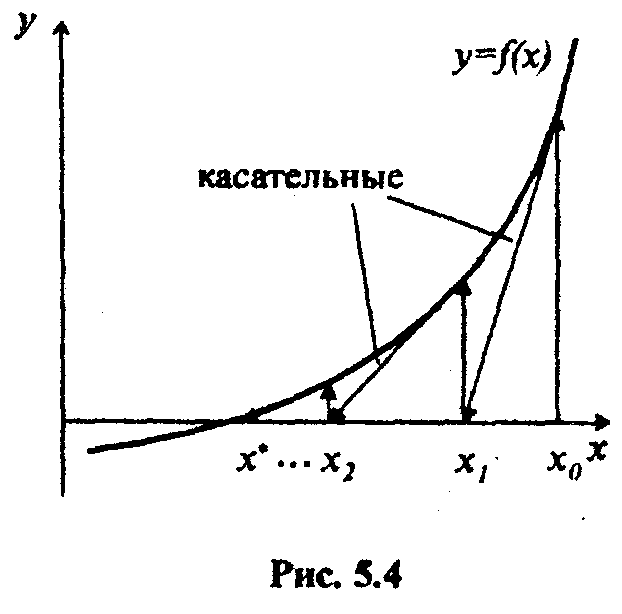
, (4.7)

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 . Если, кроме этого, для отрезка [*а*, *b*], содержащего корень, выполняются условия

  (4.8)

то метод сходится для любых *a* ≤ *x*0 ≤ *b*.

Метод Ньютона получил также второе название – *метод касательных* благодаря геометрической иллюстрации его сходимости, представленной на рис. 4.5.



**Рис. 4.5**

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

**Код программы:**

import numpy as np

from scipy.optimize import fsolve

import matplotlib.pyplot as plt

epsilon = 4

def input\_coefficients():

a = float(input("Enter a: "))

b = float(input("Enter b: "))

c = float(input("Enter c: "))

coefficients = np.array([1.0, a, b, c])

return coefficients

arr = input\_coefficients()

print(arr)

def SturmSeq(f):

arr = []

arr.append(f)

arr.append(np.polyder(f))

while True:

fn = -np.polydiv(arr[-2], arr[-1])[1]

if fn.order > 0 or abs(fn[0]) > 0.0:

arr.append(fn)

else:

break

return arr

def N(stseq, x):

if abs(f(x)) < 10.0 \*\* -epsilon:

raise ValueError("Number in N() is a root")

ans = 0

for i in range(1, len(stseq)):

if stseq[i](x) == 0.0:

raise ValueError("SturmSeq[i] is zero")

if stseq[i - 1](x) \* stseq[i](x) < 0:

ans += 1

return ans

def f(x):

y = x\*\*3 + arr[1] \* x\*\*2 + arr[2] \* x + arr[3]

return y

def build\_plot():

z = np.linspace(-10, 10)

plt.plot(z, f(z))

plt.plot(z, np.zeros(len(z)))

plt.show()

def make\_predictions(solutions\_num):

predictions = []

for i in range(solutions\_num):

prediction = float(input("Enter your prediction of the solution: "))

predictions.append(prediction)

return predictions

def chord\_method():

'''global iters

fder2 = numpy.polyder(f, 2)

if (f(R) \* fder2(R) > 0):

(oldx, x) = (R, L)

elif (f(L) \* fder2(L) > 0):

(oldx, x) = (L, R)

else:

raise ValueError("Bad bounds in first Secant method")

t = oldx

while (abs(x - oldx) > EPS):

# while (abs(f(x)) > EPS):

iters += 1

oldx = x

x = x - f(x) \* (t - x) / (f(t) - f(x))

if (not (numpy.isfinite(x))):

raise RuntimeError("Something went wrong, and x is not a number")

if ((x < L) or (R < x)):

raise RuntimeError("Something went wrong, and x is not in [L, R]")'''

num\_of\_solutions = sturm\_solutions\_num

solutions\_predictions = make\_predictions(num\_of\_solutions)

solutions = fsolve(f, solutions\_predictions)

answer = np.around(solutions, decimals=epsilon)

return answer

def newton\_method():

'''global iters

fder = numpy.polyder(f)

fder2 = numpy.polyder(f, 2)

if (f(L) \* fder2(L) > 0):

(oldx, x) = (R, L)

elif (f(R) \* fder2(R) > 0):

(oldx, x) = (L, R)

else:

raise ValueError("Bad bounds in Newton method")

while (abs(x - oldx) > EPS):

# while (abs(f(x)) > EPS):

iters += 1

oldx = x

x = x - f(x) / fder(x)

if (not (numpy.isfinite(x))):

raise RuntimeError("Something went wrong, and x is not a number")

if ((x < L) or (R < x)):

raise RuntimeError("Something went wrong, and x is not in [L, R]")'''

num\_of\_solutions = sturm\_solutions\_num

solutions\_predictions = make\_predictions(num\_of\_solutions)

solutions = fsolve(f, solutions\_predictions)

answer = np.around(solutions, decimals=epsilon)

return answer

build\_plot()

(func) = np.poly1d(arr)

Sturm = SturmSeq(func)

print("Amount of roots in [-10, 10] by Sturm:")

sturm\_solutions\_num = N(Sturm, -10) - N(Sturm, 10)

print(sturm\_solutions\_num)

chord\_solutions = chord\_method()

print("Chord method solutions:\n")

print(chord\_solutions)

newton\_solutions = newton\_method()

print("Newton method solutions:\n")

print(newton\_solutions)

**Результаты:**

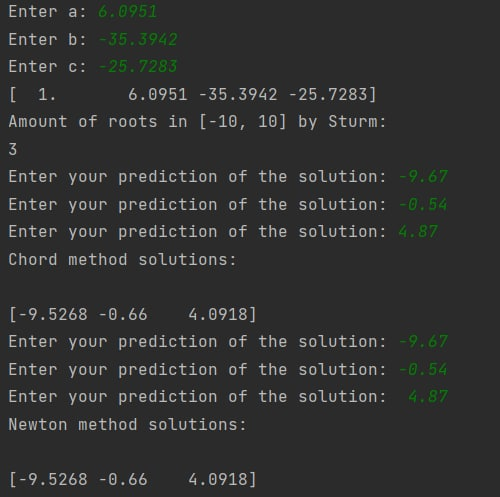


Рис. 1 Ввод данных, получение ответов

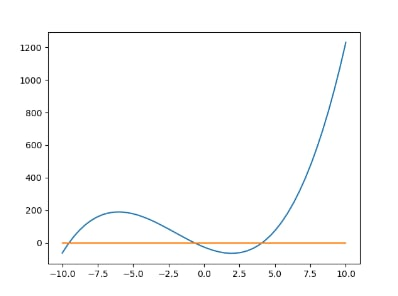


Рис. 2 График функции f(x)

**Вывод:**

В ходе лабораторной работы было изучено численное решение систем нелинейных уравнений методами простых итераций и Ньютона. Было проведено отделение решений, построены и запрограммированы алгоритмы методов, численно решено тестовое задание, было проведено сравнение трудоемкости методов при помощи подсчета количества итераций двух методов.