**Московский Авиационный Институт**

(Национальный Исследовательский Университет)

Институт №8 «Компьютерные науки и прикладная математика»

**Лабораторная работа №2**

**«Численные методы»**

**Вариант 9**

**Выполнила студентка группы**: М8О-305Б-21

Бондарева Елена Евгеньевна

# Преподаватель: Филиппов Глеб Сергеевич

Оценка: !

Дата: !

Москва, 2024

**Задание 2.1**

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.



**Теоретические сведения**

Численное решение нелинейных (алгебраических или трансцендентных) уравнений вида ***f(x) = 0*** заключается в нахождении значений *x*, удовлетворяющих (с заданной точностью) данному уравнению и состоит из следующих основных этапов:

1. Отделение корней уравнения
2. Уточнение с помощью некоторого вычислительного алгоритма конкретного выделенного корня с заданной точностью

Целью первого этапа является нахождение отрезков из области определения функции *f(x)*, внутри которых содержится только один корень решаемого уравнения. Иногда ограничиваются рассмотрением лишь какой-нибудь части области определения, вызывающей по тем или иным соображениям интерес. Для реализации данного этапа используются графические или аналитические способы.

***Метод Ньютона*** (метод касательных). При нахождении корня уравнения *f(x)=0* методом Ньютона, итерационный процесс определяется формулой

Для начала вычислений требуется задание начального приближения

Условия сходимости метода определяются следующей теоремой:

Теорема. Пусть на отрезке функция имеет первую и вторую производные постоянного знака и пусть .

Тогда если точка выбрана на так, что

То начатая с нее последовательность , определяемая методом Ньютона, монотонно сходится к корню уравнения.

В качестве условия окончания итераций в практических вычислениях часто используется правило

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

Решение ищется путем построения последовательности

начиная с некоторого заданного значения . Если – непрерывная функция, а – сходящаяся последовательность, то значение является решением уравнения.

Условия сходимости метода и оценка его погрешности определяются теоремой:

Теорема. Пусть функция определена и дифференцируема на отрезке . Тогда если выполняются условия:

то уравнение имеет и притом единственный на , к этому корню сходится определяемая методом простой итерации последовательность , начинающаяся с любого

При этом справедливы оценки погрешности

**Код программы:**

import math

import numpy as np

import matplotlib.pyplot as plt

def f(x):

    return x\*\*3 + x\*\*2 - 2\*x - 1

def f\_der(x):

    return 3\*x\*\*2 + 2\*x - 2

def f\_der2(x):

    return 6\*x + 2

def g(x):

    return (-x\*\*2+2\*x+1)\*\*(1/3)

def g\_der(x):

    return -(1/3)\*((2\*x-2)/((-x\*\*2+2\*x+1)\*\*(2/3)))

def simple\_iteration\_method(a,b, tolerance, max\_iterations):

    if ((f(a) > 0 and f(b) > 0) or (f(a) < 0 and f(b) < 0)):

        raise ValueError("Нет решения!")

    iteration = 0

    if (abs(g\_der(a)) >= 1) or (abs(g\_der(b)) >= 1):

        raise ValueError("Нет решения!")

    x=(a+b)/2

    while iteration < max\_iterations:

        x\_new = g(x)

        if abs(x\_new - x) < tolerance:

            return x\_new, iteration + 1

        x = x\_new

        iteration += 1

        if abs(g\_der(x)) >= 1:

            raise ValueError("Нет решения!")

    raise ValueError("Метод не сошелся после максимального числа итераций")

def newton\_method(a, b, tolerance):

    if ((f(a) > 0 and f(b) > 0) or (f(a) < 0 and f(b) < 0)):

        raise ValueError("Нет решения!")

    if (f(a)\*f\_der2(a) <= 0) and (f(b)\*f\_der2(b) <= 0):

        raise ValueError("Нет решения!")

    if (f(a)\*f\_der2(a) <= 0):

        x = b

    else:

        x = a

    iterations = 0

    while True:

        x\_new = x - f(x) / f\_der(x)

        iterations += 1

        if abs(x\_new - x) < tolerance:

            return x\_new, iterations

        x = x\_new

def drawplot():

    x = [0,2]

    y = [0,0]

    x1 = np.arange(0,2,0.1)

    y1 = f(x1)

    plt.plot(x1,y1,x,y)

    plt.show()

a\_value = 4

epsilon\_value = 0.01

max\_iterations\_value = 1000

a = 1

b = 2

tolerance = 0.001

a1 = 1.0

b1 = 3.0

a2 = 0.0

b2 = 4.0

drawplot()

result\_simple\_iteration = simple\_iteration\_method(a, b, 0.0001, 1000)

print("Простой метод итерации:", result\_simple\_iteration)

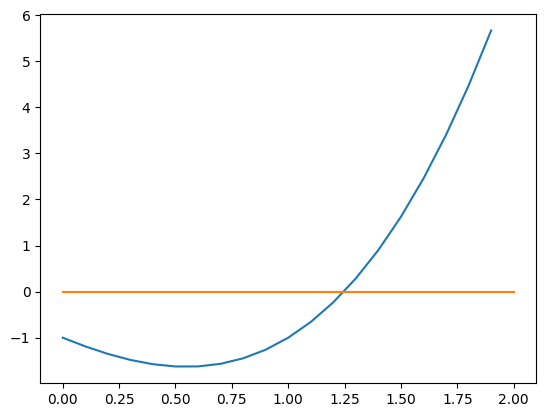
print("Решение:", result\_simple\_iteration[0:2])

result\_newton\_method = newton\_method(a, b, 0.0001)

print("Метод Ньютона:", result\_newton\_method)

print("Решение:",result\_newton\_method[0:2])

**Вывод программы:**

****

Простой метод итерации: (1.2469747604973989, 5)

Решение: (1.2469747604973989, 5)

Метод Ньютона: (1.246979603722876, 5)

Решение: (1.246979603722876, 5

**Вывод:**

В результате выполнения лабораторной работы были изучены численные методы решения нелинейных уравнений методом простой итерации и методом Ньютона. Было установлено, что метод простой итерации является простым и удобным методом, позволяющим найти приближенное решение нелинейного уравнения, а метод Ньютона обеспечивает более быструю сходимость и точность решения. Были разработаны алгоритмы вычисления корней нелинейных уравнений с помощью каждого из методов и проведено сравнение этих методов на примере решения нелинейного уравнения.

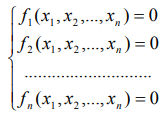
**Задание 2.2**

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.

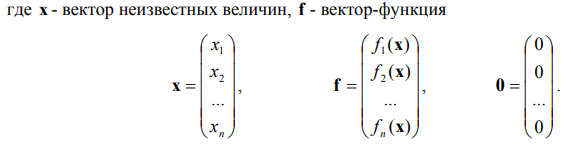
 a = 4

**Теоретические сведения**

Систему нелинейных уравнений с *n* неизвестными можно записать в виде

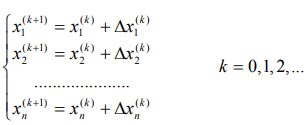


или, более коротко, в векторной форме

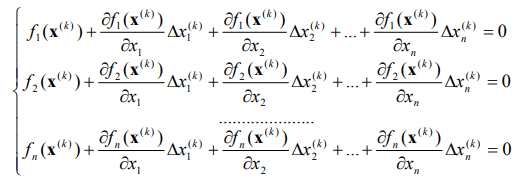


При решении системы двух уравнений, достаточно часто удобным является графический способ, когда месторасположение корней определяется как точки пересечения кривых на плоскости

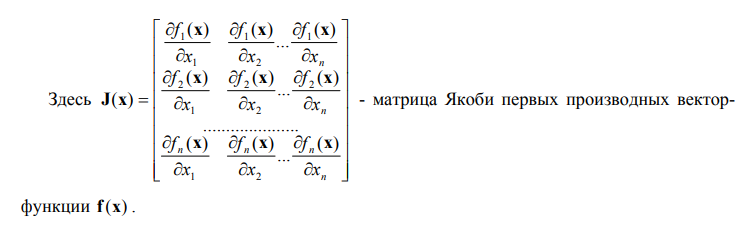
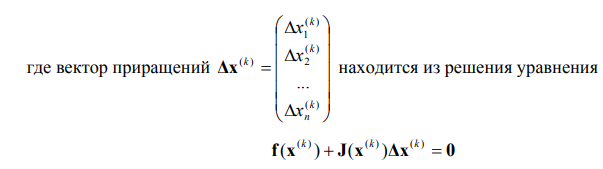
***Метод Ньютона***. Если определено начальное приближение , итерационный процесс нахождения решения системы методом Ньютона можно представить в виде



где значения приращений определяются из решения системы линейных алгебраических уравнений, все коэффициенты которой выражаются через известное предыдущее приближение

**

В векторно-матричной форме расчетные формулы имеют вид

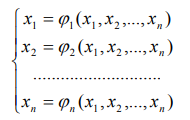


Где  **–** матрица, обратная матрице Якоби.

В практических вычислениях в качестве условия окончания итераций обычно используется критерий

где ε – заданная точность

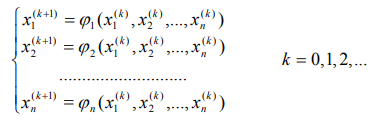
***Метод простой итерации.*** При использовании метода простой итерации система уравнений приводится к эквивалентной системе специального вида



или, в векторной форме:

где функции – определены и непрерывны в некоторой окрестности искомого изолированного решения

Если выбрано некоторое начальное приближение последующие приближения в методе простой итерации находятся по формулам

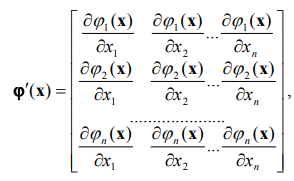


или, в векторной форме

Если последовательность векторов сходится, то она сходится к решению

Достаточное условие сходимости итерационного процесса формулируется следующим образом:

Теорема. Пусть вектор-функция непрерывна, вместе со своей производной



в ограниченной выпуклой замкнутой области *G* и

где q - постоянная. Если и все последовательные приближения

в области *G* и справедливы оценки погрешности :

**Код программы:**

import math

import numpy as np

import matplotlib.pyplot as plt

def f1(x1, x2, a):

    return x1\*\*2 + x2\*\*2 - a\*\*2

def f2(x1, x2, a):

    return x1 - math.exp(x2) + a

def df1\_dx1(x1, x2, a):

    return 2 \* x1

def df1\_dx2(x1, x2, a):

    return 2 \* x2

def df2\_dx1(x1, x2, a):

    return 1

def df2\_dx2(x1, x2, a):

    return -np.exp(x2)

def nextx1(x2, a):

    return (-x2\*\*2 + a\*\*2)\*\*(1/2)

def nextx2(x1,a):

    return np.log(x1+a)

def simple\_iteration\_method\_multiple(a1, b1, a2, b2, a, tolerance, max\_iterations):

    x1 = (a1+b1)/2

    x2 = (a2+b2)/2

    errors = []  # хранение погрешностей на каждой итерации

    for i in range(max\_iterations):

        x1\_new = nextx1(x2,a)

        x2\_new = nextx2(x1,a)

        error = max(abs(x1\_new - x1), abs(x2\_new - x2))

        errors.append(error)

        if error < tolerance:

            return x1\_new, x2\_new, i+1, errors

        x1, x2 = x1\_new, x2\_new

    return None, None, max\_iterations, errors

def newton\_method\_multiple(a1, b1, a2, b2, a, tolerance, max\_iterations):

    x1 = (a1+b1)/2

    x2 = (a2+b2)/2

    errors = []  # для хранения погрешностей на каждой итерации

    for i in range(max\_iterations):

        det = df1\_dx1(x1, x2, a) \* df2\_dx2(x1, x2, a) - df1\_dx2(x1, x2, a) \* df2\_dx1(x1, x2, a)

        if det == 0:

            return None, None, i+1, errors  # если якобиан равен нулю

        x1\_new = x1 - (f1(x1, x2, a) \* df2\_dx2(x1, x2, a) - f2(x1, x2, a) \* df1\_dx2(x1, x2, a)) / det

        x2\_new = x2 - (f2(x1, x2, a) \* df1\_dx1(x1, x2, a) - f1(x1, x2, a) \* df2\_dx1(x1, x2, a)) / det

        error = max(abs(x1\_new - x1), abs(x2\_new - x2))

        errors.append(error)

        if error < tolerance:

            return x1\_new, x2\_new, i+1, errors

        x1, x2 = x1\_new, x2\_new

    return None, None, max\_iterations, errors

def drawplot\_multi(a):

    x1 = np.arange(0,4,0.1)

    f1 = np.sqrt(a\*\*2-x1\*\*2)

    plt.plot(x1,f1,x1,nextx2(x1,a))

    plt.show()

a\_value = 4

epsilon\_value = 0.01

max\_iterations\_value = 1000

a = 1

b = 2

tolerance = 0.001

a1 = 1.0

b1 = 3.0

a2 = 0.0

b2 = 4.0

drawplot\_multi(a\_value)

result\_simple\_iteration = simple\_iteration\_method\_multiple(a1, b1, a2, b2, a\_value, tolerance, max\_iterations\_value)

print("Метод простой итерации:")

print("Решение:", result\_simple\_iteration[0:2])

print("Количество итераций:", result\_simple\_iteration[2])

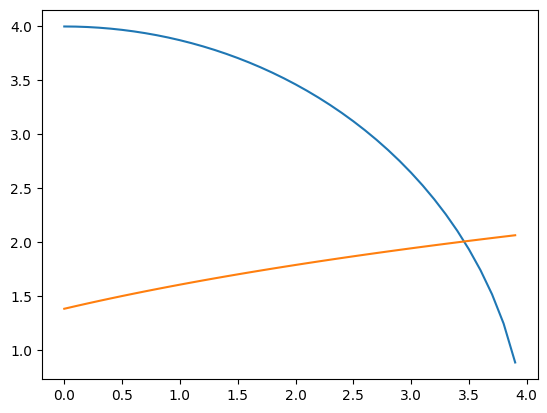
result\_newton\_method = newton\_method\_multiple(a1, b1, a2, b2, a\_value, tolerance, max\_iterations\_value)

print("Метод Ньютона:")

print("Решение:", result\_newton\_method[0:2])

print("Количество итераций:", result\_newton\_method[2])

**Вывод программы:**

****

Метод простой итерации:

Решение: (3.4586681639523387, 2.009472622649738)

Количество итераций: 7

Метод Ньютона:

Решение: (3.458670731045493, 2.009377212458886)

Количество итераций: 4

**Вывод:**

В результате лабораторной работы были изучены численные методы решения системы нелинейных уравнений методом простой итерации и методом Ньютона. Было установлено, что метод простой итерации позволяет решать систему нелинейных уравнений путем последовательного нахождения корней каждого уравнения, а метод Ньютона применяется для решения системы нелинейных уравнений с помощью метода касательных. Также были разработаны алгоритмы вычисления решения системы нелинейных уравнений с использованием каждого из методов и проведено сравнение этих методов на примере решения системы нелинейных уравнений.