**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 2**

по курсу «Численные методы»

Тема: Решение нелинейных уравнений и систем нелинейных уравнений.

Студент: Сорокин Н.Э.

Группа: М80-303Б-20

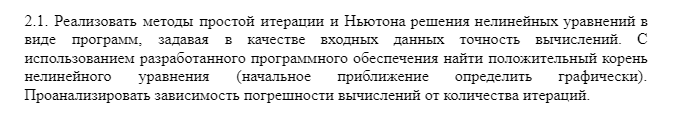
Преподаватель: Иванов И.Э.

Оценка:

Москва, 2023

**№1**

**Постановка задачи:**

****

**Вариант 22:**

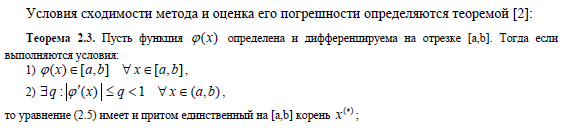
****

**Теория:**

**Метод простой итерации:**

1) Исходное уравнение заменяется эквивалентным уравнением с выделенным линейным членом

2) Выбирается начальное приближение и начиная с него строится последовательность . Если – непрерывная функция, а – сходится, то решением уравнения будет



3) Условие окончания итерационного процесса можно записать так:

**Метод Ньютона:**

1) Выбирается отрезок [a,b] такой, что f(a)f(b) < 0

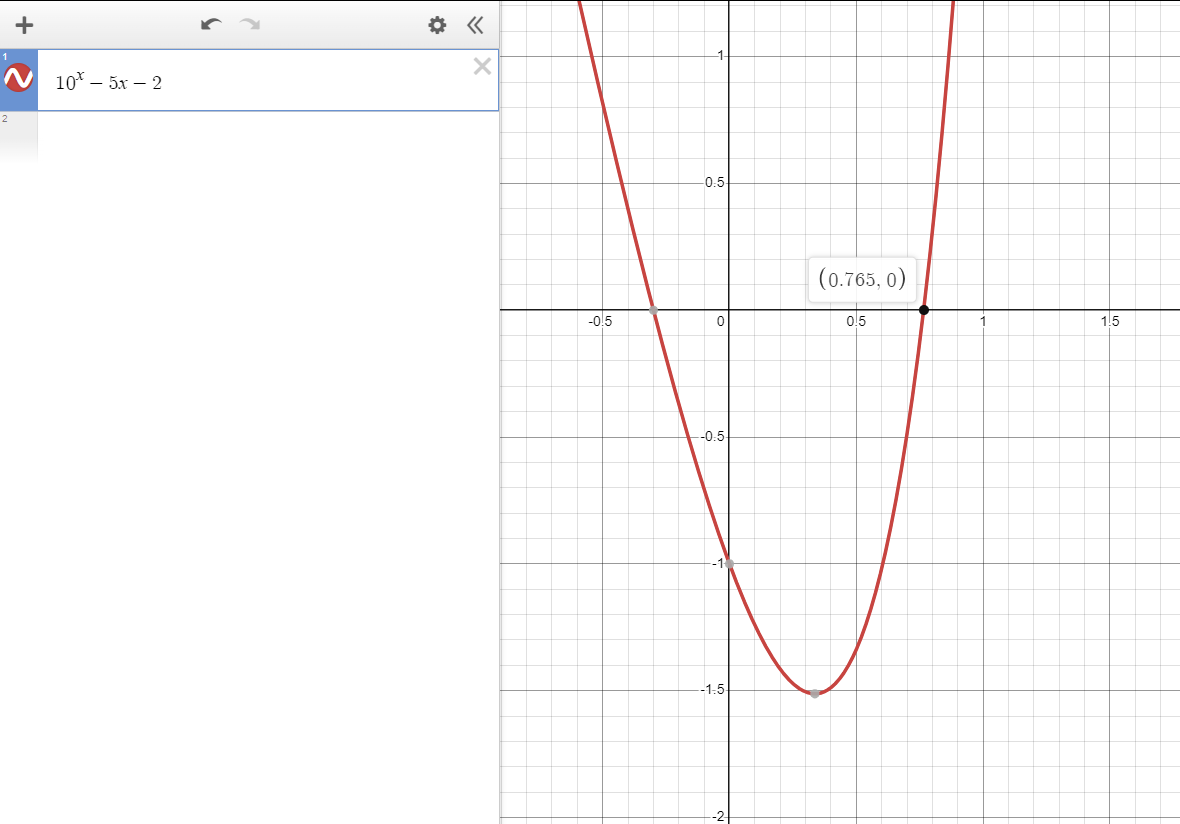
2) Выбирается начальное приближение на [a,b] так, чтобы ) > 0

3) Строится последовательность , которая, согласно теореме, будет сходится корню.

4) Условие окончания итерационного процесса можно записать так:

**Приближённое нахождение корней:**

Для локализации корней я использовал графический метод. Построим график:



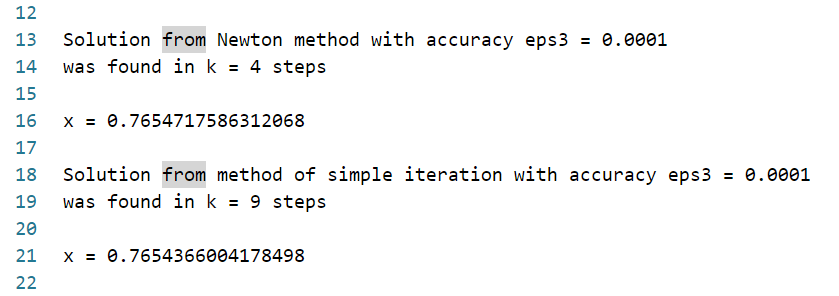
Из графика видно, что искомый корень лежит на отрезке [0, 1]. Возьмем начальное приближение точку x\_0 = 0.9

**Получение функции phi(x) для метода простой итерации:**

Для того чтобы получить получить функции phi(x) с нормой меньшей единицы “вытаскиваем” х из показательной степени. Получаем:



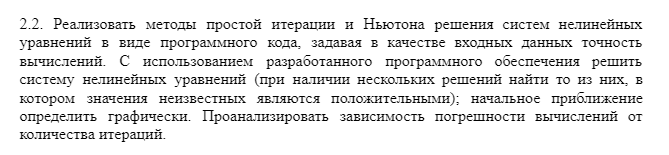
**Полученный ответ:**

****

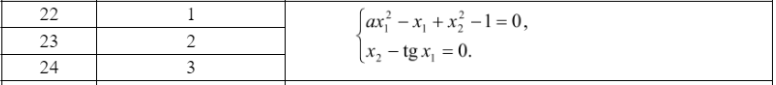
**Вывод:** Для решения нелинейных уравнений методов линейной алгебры уже не хватает. Поэтому используется следующая процедура: во-первых графически ищется приближенное решение уравнения, во-вторых это приближенное решение уточняется методом простой итерации (основывающийся на принципе сжимающих отображений) и методом Ньютона. Важно иметь в виду, что гарантировать сходимость таких методов можно проверяя достаточные условия сходимости.

**№2**

**Постановка задачи:**



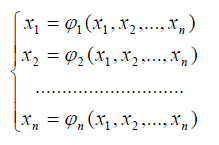
**Вариант 22:**

****

**Теория:**

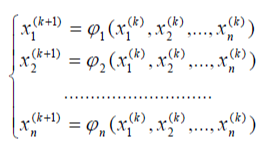
**Метод простой итерации:**

1) Исходная система заменяется на эквивалентную систему вида



2) Выбираем начальное приближение

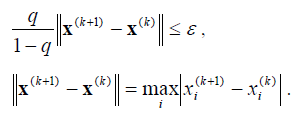
и строим последующие приближения по формулам:



3) Условие сходимости итерационного процесса:



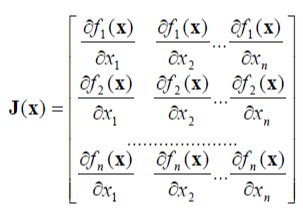
4) Условие окончания итерационного процесса:



**Метод Ньютона:**

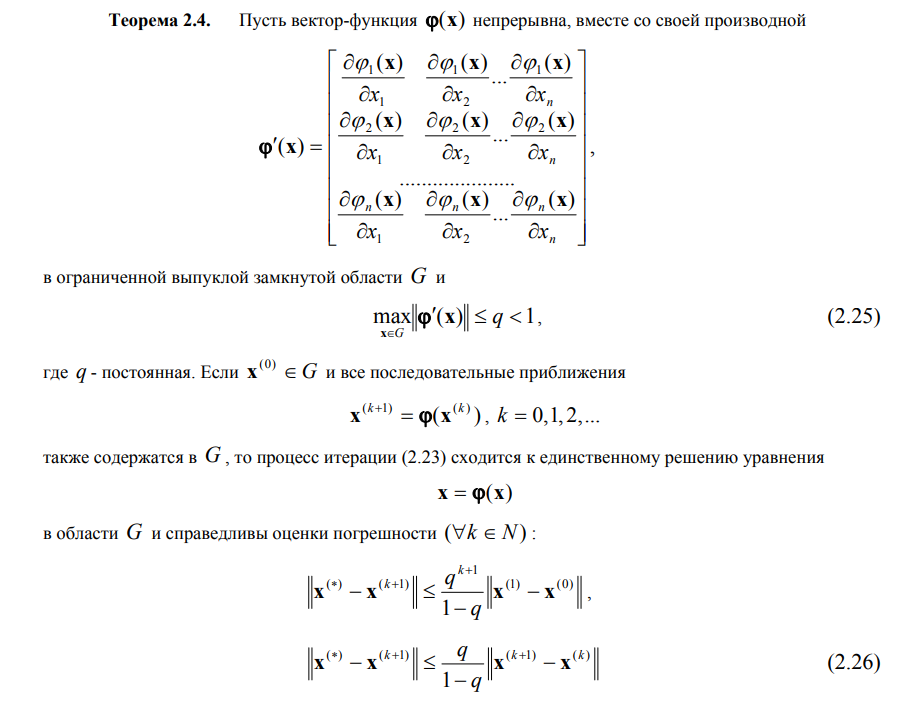
Итерационная формула метода:





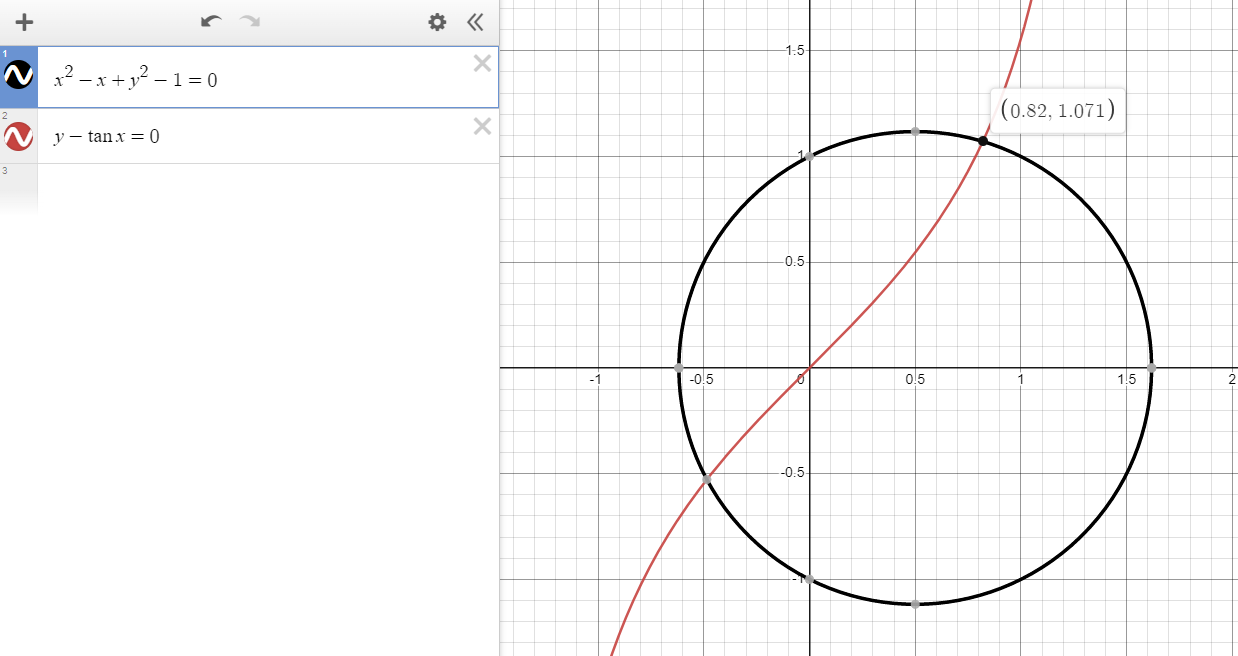
Условие окончания итерационного процесса можно записать так:

Для многомерного метода простой итерации справедливо достаточное условие сходимости:



**Приближённое нахождение корней:**

Для локализации корней я использовал графический метод. Построим графики:



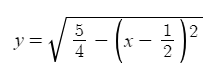
Из графика видно, что искомый корень лежит на отрезке [0.8, 0.9]x[1, 1.1]. Возьмем начальное приближение точку x =(0.5, 1.2)

**Получение функций phi(x) для метода простой итерации:**

Для выполнения достаточного условия сходимости метода простой итерации выражаем функции phi следующим образом:



Из первого уравнения системы выражаем y и берем верхнюю арку окружности:



Из второго уравнения выражаем х:



Реализованная функция метода простой итерации проверяет достаточное условие сходимости. Вывод программы:

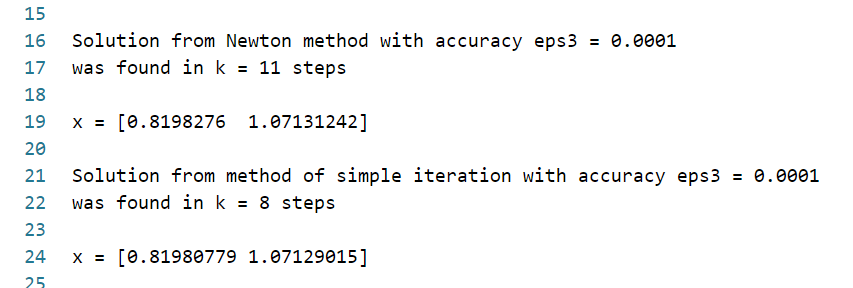
max||phi'(x)|| = 0.9284766908852593

Полученная величина меньше 1, следовательно существует константа q, такая что:



Достаточное условие выполняется.

**Полученный ответ:**



**Вывод:** Для решения систем нелинейных уравнений естественным соображением является обобщить известные методы Ньютона и простой итерации на случай нескольких переменных.

В случае метода простой итерации это реализуется выражением каждой переменной входящей в уравнение в виде: .

В случае метода Ньютона это реализуется заменой производной функции f(x) на матрицу Якоби **J**(**x**).

**Программный код:**

import numpy as np

from numpy import linalg as LA

from LinearAlgebra import \*

#module21

def Newton\_method(f, der\_f, x\_0, eps):

if(f(x\_0) \* der\_f(x\_0) <= 0):

return

k = 0

x\_prev = x\_0

norm = eps + 1

while(norm > eps and k < 100):

k += 1

x\_curr = x\_prev - f(x\_prev) / der\_f(x\_prev)

norm = abs(x\_curr - x\_prev)

x\_prev = x\_curr

return x\_curr, k

def simple\_iteration\_method(f, phi, der\_phi, a, b, eps):

x = np.linspace(a, b, 100)

q = max(der\_phi(x))

if (q >= 1):

return

k = 0

x\_prev = (a + b) / 2

norm = eps + 1

while(norm > eps and k < 100):

k += 1

x\_curr = phi(x\_prev)

norm = abs(x\_curr - x\_prev)

x\_prev = x\_curr

return x\_curr, k

#module22

def Newton\_method(f, der\_f, x\_0, eps):

k = 0

x\_prev = x\_0

norm = eps + 1

while(norm > eps and k < 100):

k += 1

L, U, b = get\_LU(der\_f(x\_prev), -f(x\_prev))

x\_del = solve\_LU(L, U, b)

x\_curr = x\_prev + x\_del

norm = LA.norm(x\_curr - x\_prev)

x\_prev = x\_curr

return x\_curr, k

def simple\_iteration\_method(f, phi, der\_phi, a, b, eps):

x = ((a[0] + b[0]) / 2, (a[1] + b[1]) / 2)

q = LA.norm(der\_phi(x), np.inf)

if (q >= 1):

return

k = 0

x\_prev = x

norm = eps + 1

while(norm > eps and k < 100):

k += 1

x\_curr = phi(x\_prev)

norm = LA.norm(x\_curr - x\_prev)

x\_prev = x\_curr

return x\_curr, k