Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Методы численного анализа»

**ОТЧЕТ**

к лабораторной работе №4

на тему:

**«РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ»**

БГУИР 1-40-04-01

|  |
| --- |
| Выполнил студент группы 253504  Дмитрук Богдан Ярославович |
| 31.10.2023 |
| (дата, подпись студента) |
| Проверил  Анисимов Владимир Яковлевич |
|  |
| (дата, подпись преподавателя) |

Минск 2023

**СОДЕРЖАНИЕ**

[1. Цели выполнения задания 3](#_Toc147528122)

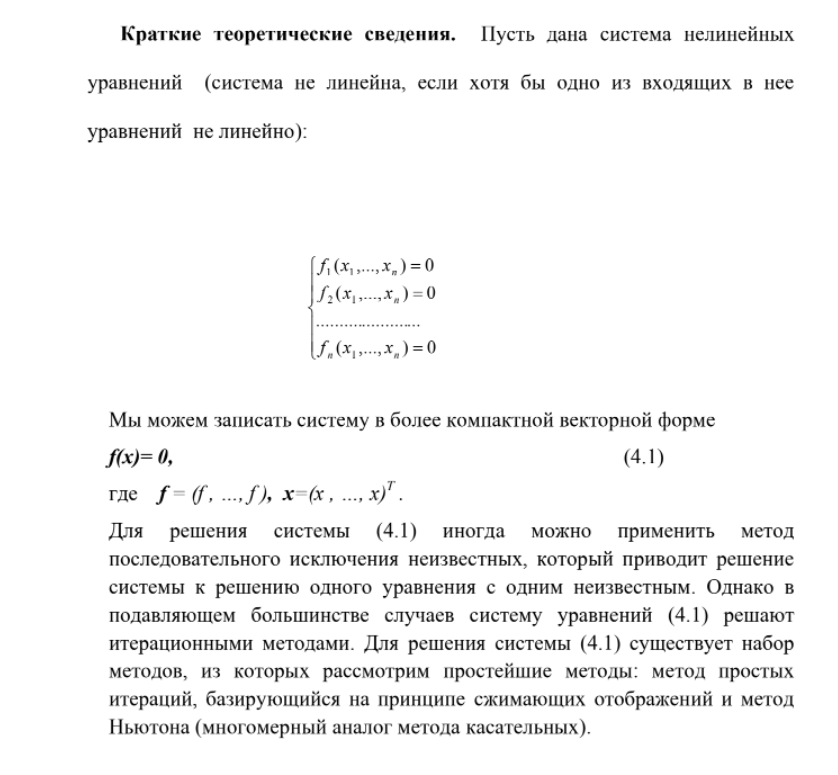
[2. Краткие теоретические сведения 4](#_Toc147528123)

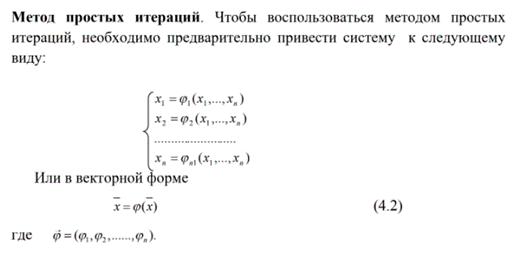
[3. Задание 11](#_Toc147528124)

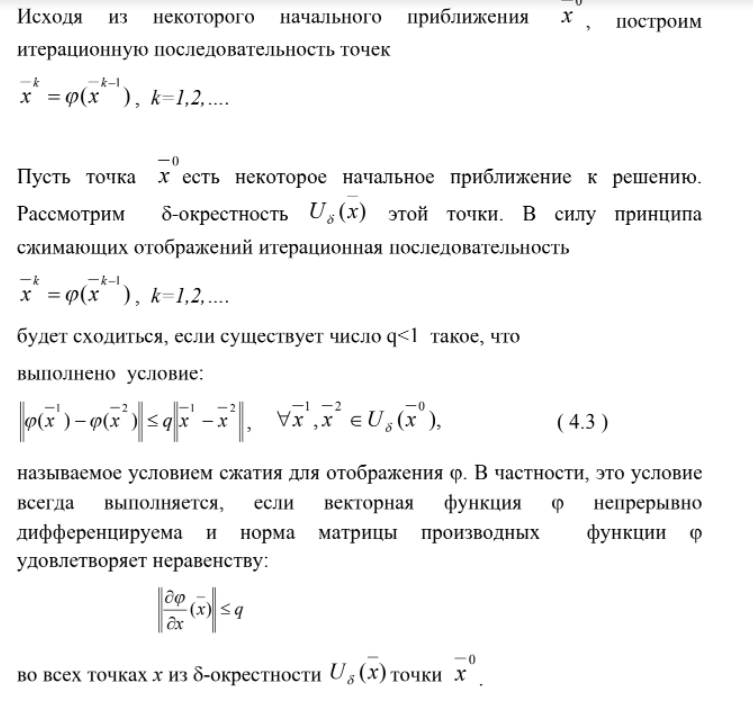
[4. Программная реализация 12](#_Toc147528125)

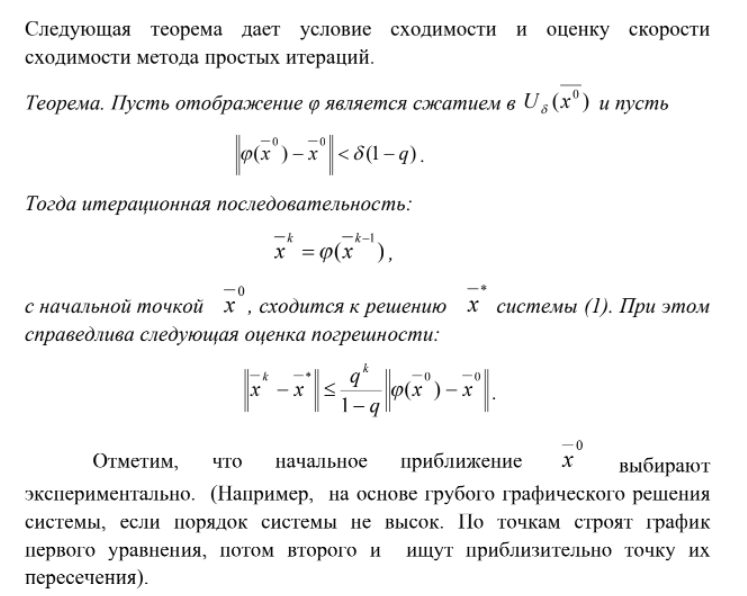
[5. Полученные результаты 16](#_Toc147528126)

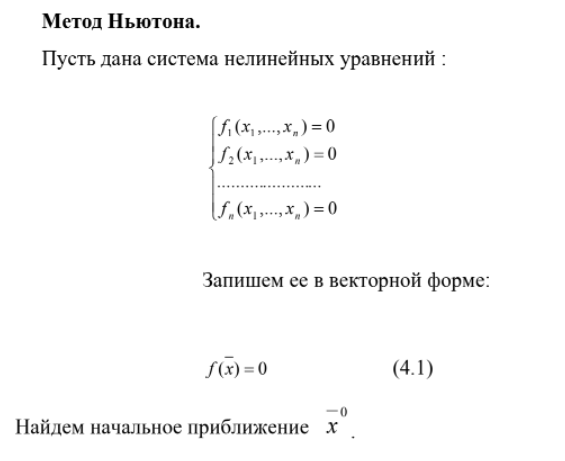
[6. Выводы 18](#_Toc147528127)

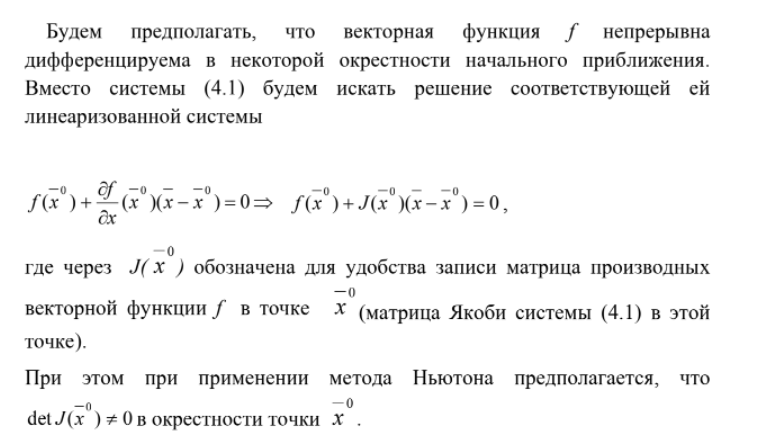
1. ЦЕЛИ ВЫПОЛНЕНИЯ ЗАДАНИЯ
2. Изучить методы численного решения систем нелинейных уравнений (метод простых итераций, метод Ньютона)
3. Составить программу численного решения нелинейных уравнений методами простых итераций и Ньютона
4. Составить алгоритм решения нелинейных уравнений методами простых итераций и Ньютона
5. Проверить правильность работы программы на тестовых примерах
6. Численно решить нелинейное уравнение заданного варианта
7. Сравнить число итераций, необходимого для достижения заданной точности вычисления разными методами
8. **КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**  
     
   

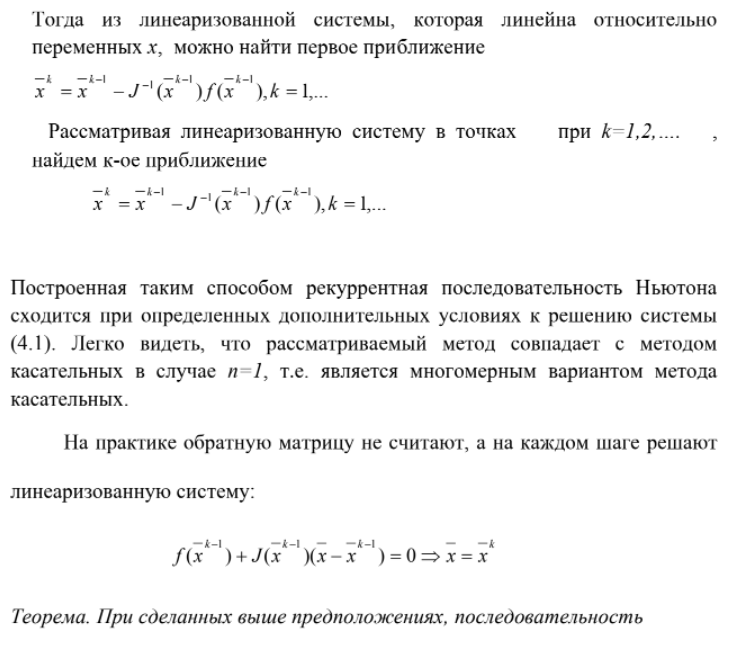


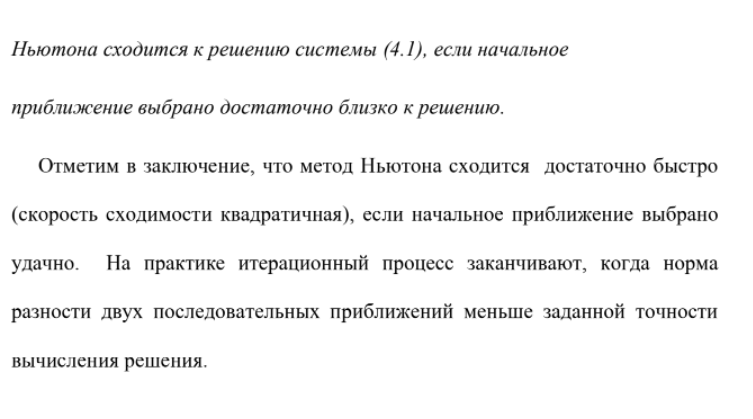


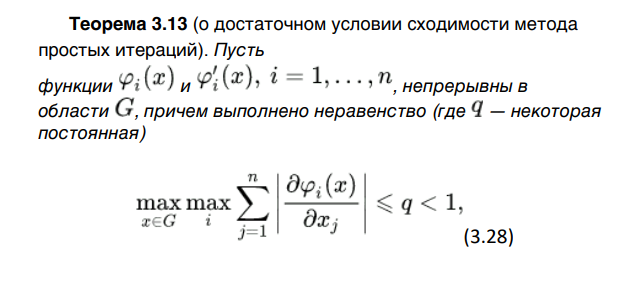


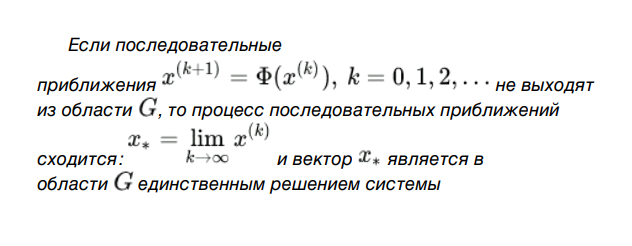


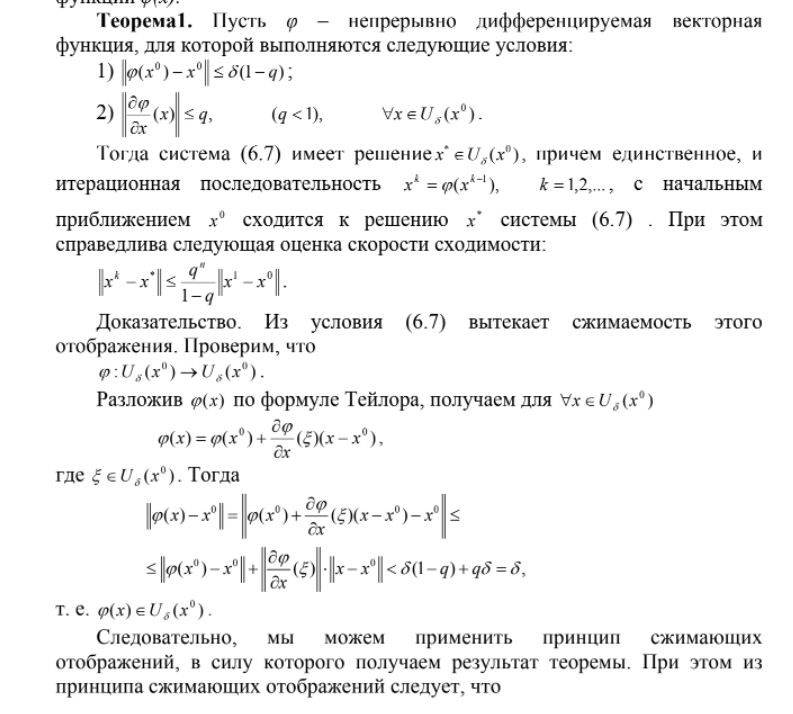


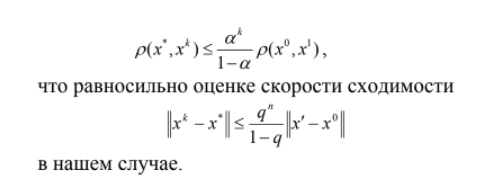


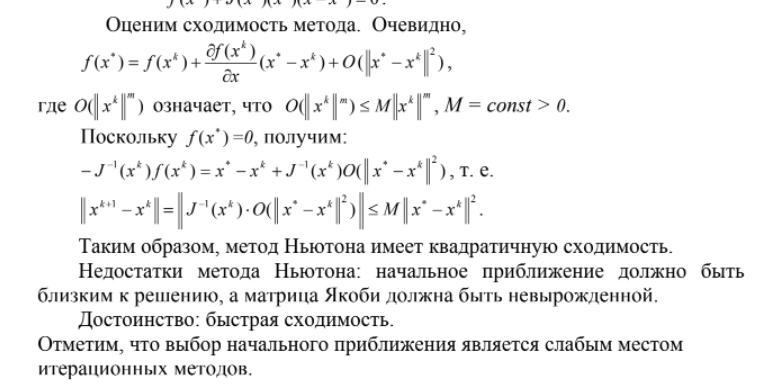




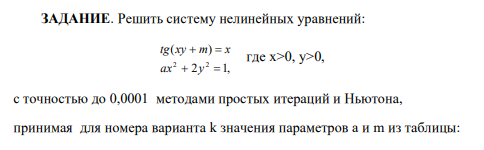








1. **ЗАДАНИЕ**



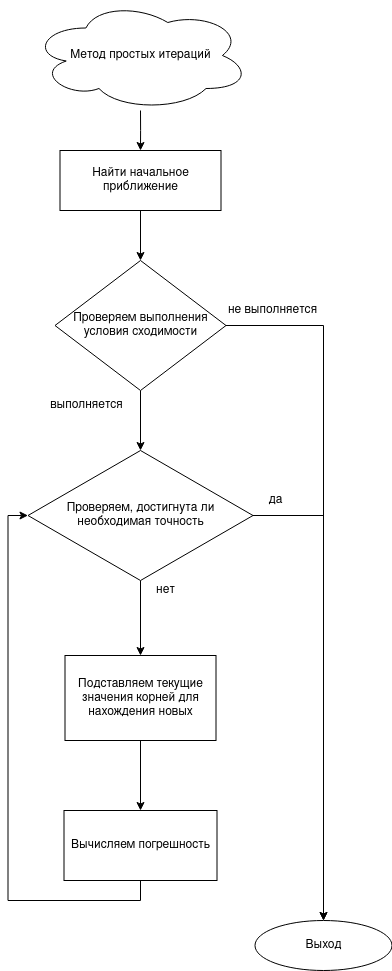
Вариант 5 (k = 5)

m = 0.2

a = 0.9

1. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

**Алгоритм программных методов**



**Метод простых итераций**

def iteration\_solve(system\_equations: np.array, approx, tol=0.00001, verbose=0):

n = system\_equations.shape[0]

x = symbols(f'x:{n}')

fi\_equations = system\_equations[1]

prev\_roots = np.zeros(shape=(n, ))

curr\_roots = list(approx)

inaccuracys = np.zeros(shape=(n, ))

inaccuracy = tol \* 10000

J = get\_jacobi(system\_equations[0])

jacobi\_values = np.zeros(shape=(n, n))

roots\_d = roots\_to\_dict(curr\_roots, x)

for i in range(n):

for j in range(n):

jacobi\_values[i, j] = J[i, j].subs(roots\_d)

# compute q

if verbose == 1:

q\_1 = float(get\_q(fi\_equations[0], (approx[0], approx[1]), 0.1))

q\_2 = float(get\_q(fi\_equations[1], (approx[1], approx[0]), 0.1))

print(f'q\_1 = {q\_1}')

print(f'q\_2 = {q\_2}')

if q\_1>=1 or q\_2>=1:

print("q is greater than 1")

return None, None

iteration = 0

while inaccuracy > tol:

prev\_roots = curr\_roots.copy()

roots\_d = roots\_to\_dict(curr\_roots, x)

for i in range(n):

try:

curr\_roots[i] = float(fi\_equations[i].subs(roots\_d))

except TypeError:

print("Complex numbers!")

inaccuracys[i] = abs(prev\_roots[i] - curr\_roots[i])

roots\_d = roots\_to\_dict(curr\_roots, x)

inaccuracy = np.amax(inaccuracys)

iteration += 1

if verbose == 1:

print('stopped iteration method\n')

return curr\_roots, iteration

**Метод Ньютона**

def newton\_solve(system\_equations: np.array, approx, tol=0.00001):

n = system\_equations.shape[0]

x = symbols(f'x:{n}')

J = get\_jacobi(system\_equations)

error = tol \* 10000

iteration = 0

roots = approx

while error > tol:

iteration += 1

roots\_d = roots\_to\_dict(roots, x)

jacobi\_values = np.zeros(shape=(n, n))

for i in range(n):

for j in range(n):

jacobi\_values[i, j] = J[i, j].subs(roots\_d)

jacobi\_det = np.linalg.det(jacobi\_values)

print(f"Jacobi det = {jacobi\_det}")

if not jacobi\_det:

print("det equal 0. Can't solve system")

exit(0)

F = np.zeros(shape=(n, ))

for i in range(0, n):

F[i] = system\_equations[i].subs(roots\_d)

delta\_x = np.zeros(shape=(n, ), dtype=float)

delta\_x = np.linalg.solve(jacobi\_values, -1 \* F)

roots = delta\_x + roots

error = np.amax(abs(delta\_x))

return roots, iteration

1. ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

Первый тестовый пример:

Вывод:

Initial approximation: (0.6, 0.6)

q\_1 = 1.0220940897677138

q\_2 = 1.199999999999998

ValueError: Compression condition is not satisfied!

q > 1

Второй тестовый пример:

Я пытался сообразить но не получилось такой пример, чтобы критерий сходимости удовлетворялся и при этом были комплексные корни, но проверка на них присутствует в обоих итеративных методах при подстановке и получении новых приближений вот в таком фрагменте кода:

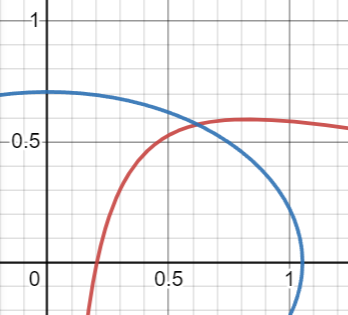
try:

curr\_roots[i] = float(fi\_equations[i].subs(roots\_d))

except TypeError:

print("Complex numbers!")

Исходное условие:



Отсюда за начально приближение берем точку (0,6; 0,6)

Получим:

Initial approximation: (0.6, 0.6)

q\_1 = 0.8765894087167037

q\_2 = 0.504928048014517

stopped iteration method

Simple iteration:

Roots: [0.6190202637188493, 0.57233404659993]

Number of iterations: 9

Initial approximation: (0.6, 0.6)

Jacobi det = -1.2966919501537868

Jacobi det = -1.438284442572687

Jacobi det = -1.4310232132267242

Newton method

Roots: [0.6190172831902083, 0.5723354972447913]

Number of iterations: 3

1. ВЫВОДЫ

Таким образом, в ходе выполнения лабораторной работы были изучены методы численного решения систем нелинейных уравнений (метод простой итерации, метод Ньютона), составлена программа численного решения нелинейных уравнений методами простой итерации и Ньютона, проверена правильность работы программы на тестовых примерах, численно решено нелинейное уравнение заданного варианта, сравнено число итераций, необходимого для достижения заданной точности вычисления разными методами.

Как можно заметить, метод Ньютона более практичен, так как в решенных системах уравнений быстрее сходился к корню с заданной точностью, однако он очень сильно зависим от выбора начального приближения. В общем случае, результат всех итерационных методов зависит от выбора начального приближения.