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 Нелинейных уравнений»**

БГУИР 1-40 04 01

|  |
| --- |
| Выполнил студент группы 253505  БЕКАРЕВ Станислав Сергеевич |
|  |
| (дата, подпись студента) |
| Проверил доцент кафедры информатики  АНИСИМОВ Владимир Яковлевич |
|  |
| (дата, подпись преподавателя) |

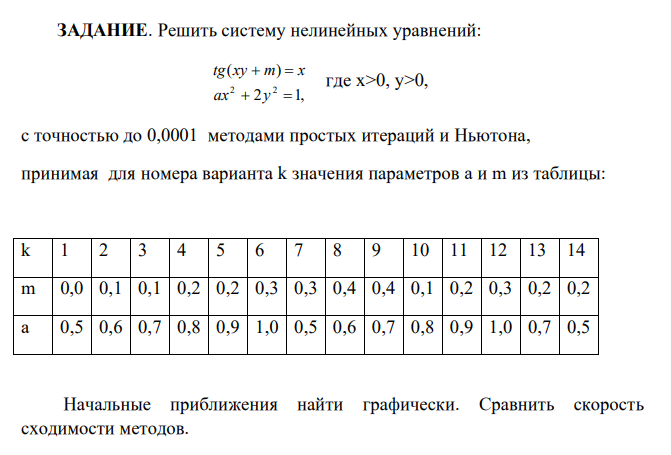
Минск 2023

**Содержание**

1. Цель работы
2. Задание
3. Программная реализация
4. Полученные результаты
5. Оценка полученных результатов
6. Вывод

**Цель работы**

* изучить метод простых итераций и метод Ньютона решения нелинейных уравнений;
* составить программу решения нелинейных уравнений указанными методами, применимую для организации вычислений на ЭВМ;
* выполнить тестовые примеры и проверить правильность работы программы

****

**Вариант 3**

**Программная реализация**

Для проверки решения подставим найденный корень в функцию и найдем ее значение.

*Исходные данные*

Система, полученная в результате подстановки в ***m*** и ***a***:



*Примечание*: x\*\*n – возведение x в степень n.

Код метода простых итераций:

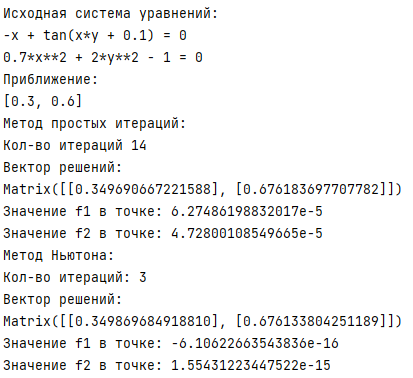
def simple\_method(phi\_x, phi\_y, X):  
 X\_solve = Matrix(X)  
 X\_next = Matrix([[0.0], [0.0]])  
 count = 0  
 while count < 2000:  
 X\_next[0] = phi\_x.subs([(x, X\_solve[0]),(y, X\_solve[1])])  
 X\_next[1] = phi\_y.subs([(x, X\_solve[0]),(y, X\_solve[1])])  
 if max(abs(X\_next - X\_solve)) < 10 \*\* (-4):  
 print(f"Кол-во итераций {count}")  
 return X\_next  
 X\_solve = X\_next.copy()  
 count += 1  
 return X\_solve

Код метода Ньютона:

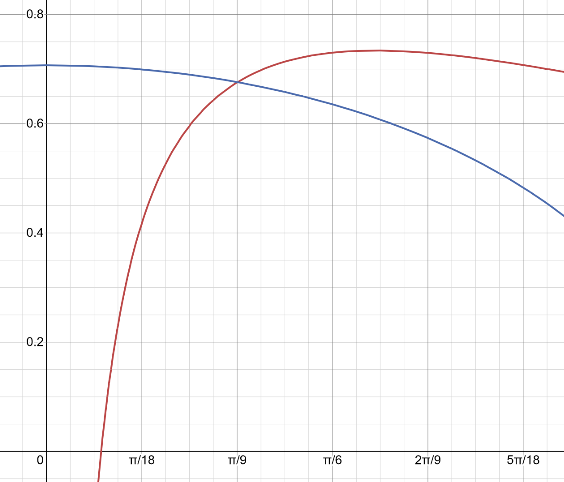
def newton\_method(f1, f2, X):  
 X\_solve = Matrix(X)  
 J = Matrix([[f1.diff(x), f1.diff(y)], [f2.diff(x), f2.diff(y)]])  
 F = Matrix([[f1], [f2]])  
 count = 0  
 while count < 2000:  
 NumJ = get\_numeric\_matrix(J, X\_solve)  
 NumF = get\_numeric\_matrix(F, X\_solve)  
 if(NumJ.det() == 0):  
 print("Определитель матрицы Якоби равен 0 :(")  
 exit()  
 X\_next = X\_solve - (NumJ \*\* (-1) \* NumF)  
 if(max(abs(X\_next - X\_solve)) < 10 \*\* (-4)):  
 print(f"Кол-во итераций: {count}")  
 return X\_next  
 X\_solve = X\_next.copy()  
 count += 1  
 return X\_solve

def get\_numeric\_matrix(Matr, X\_solve):  
 Numeric = Matr.copy()  
 tup = Matr.shape  
 for i in range(0, tup[0] \* tup[1]):  
 Numeric[i] = Matr[i].subs([(x, X\_solve[0]), (y, X\_solve[1])])  
 return Numeric

**Полученные результаты**



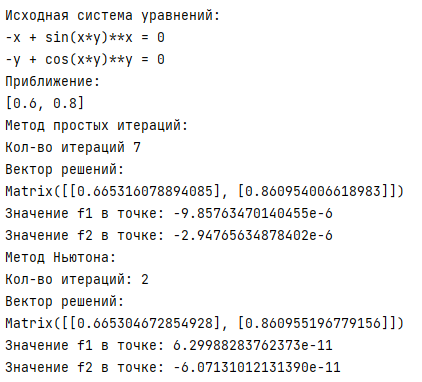
*График функций:*

**

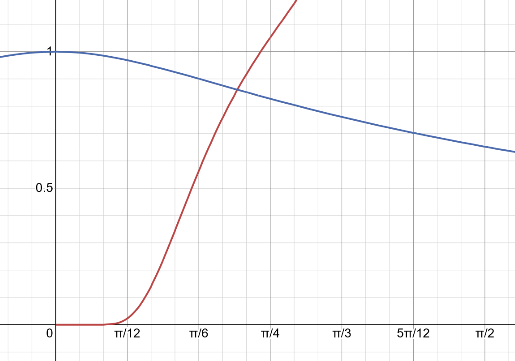
*Результаты вычислений:*

|  |  |  |
| --- | --- | --- |
|  | *Метод простых итераций* | *Метод Ньютона* |
| Решение найденное с точностью 10^-4 | x = 0.34969066  y = 0.67618370 | x = 0.34986968  y = 0.67613380 |
| Функция-1,  Функция-2 | 0.00006274  0.00004728 | -6.106e-16  1.554e-15 |
| Кол–во итераций | 14 | 3 |

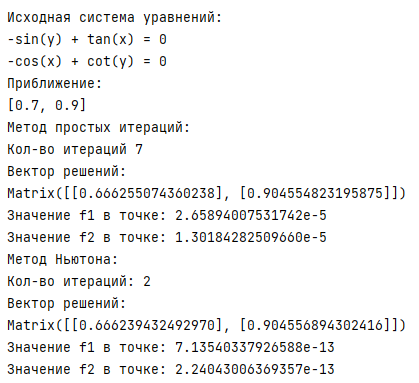
*Тестовый пример 1.*

**

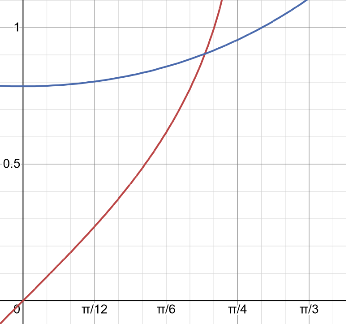
*График функций:*

**

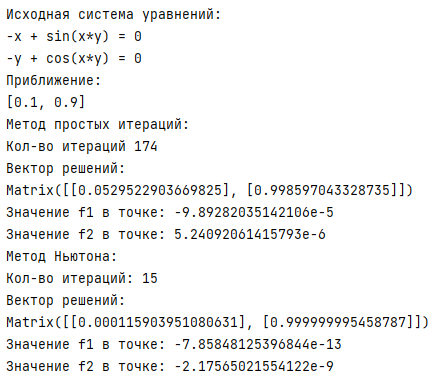
*Тестовый пример 2.*



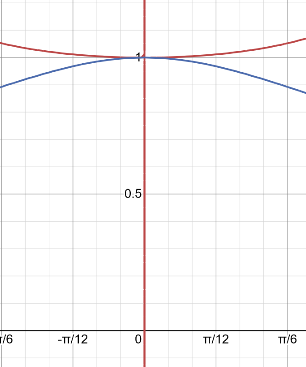
*График функций:*



*Тестовый пример 3.*



*График функций:*



**Вывод**

В ходе выполнения лабораторной работы я изучил метод простых итераций и метод Ньютона решения нелинейных уравнений, написал программу их реализации на языке Python, правильность работы программы проверил на тестовых примерах.

На основании тестов можно сделать следующие выводы:

* Программа позволяет получить решения системы с заданной точностью (заданная точность в условиях лабораторной работы 10^-4);
* Метод Ньютона эффективнее по сравнению с методом простых итераций, так как затрачивает меньшее число итераций;
* Оптимальным способом решения нелинейных уравнений является применение метода Ньютона, так скорость сходимости в этом методе почти всегда квадратичная.
* Основной недостаток методов – малая область сходимости (о должен быть достаточно близок к решению уравнения).