Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы численного анализа

**ОТЧЁТ**

к лабораторной работе

на тему

Решение систем нелинейных уравнений

Выполнил: студент группы 253502

Шишко Виктор Викторович

Проверил: Анисимов Владимир Яковлевич

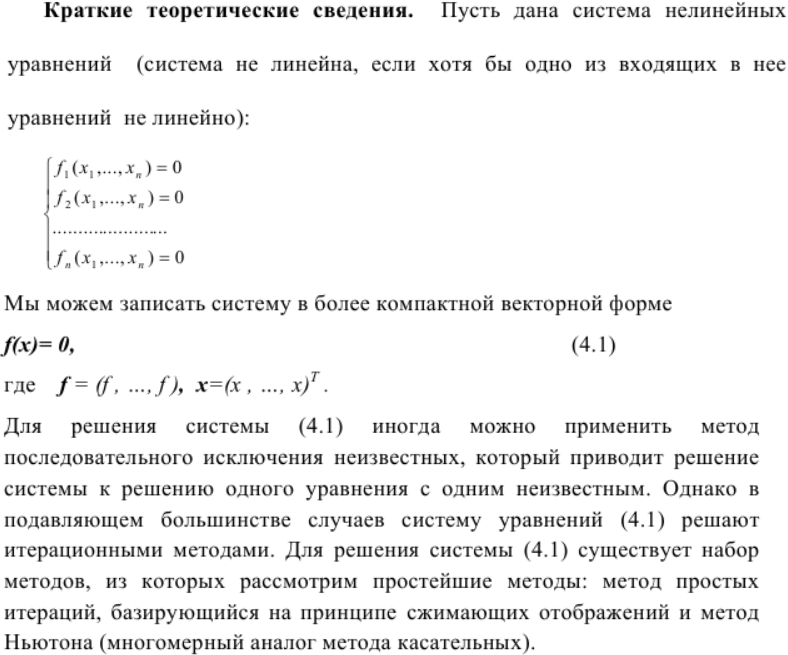
Минск 2023

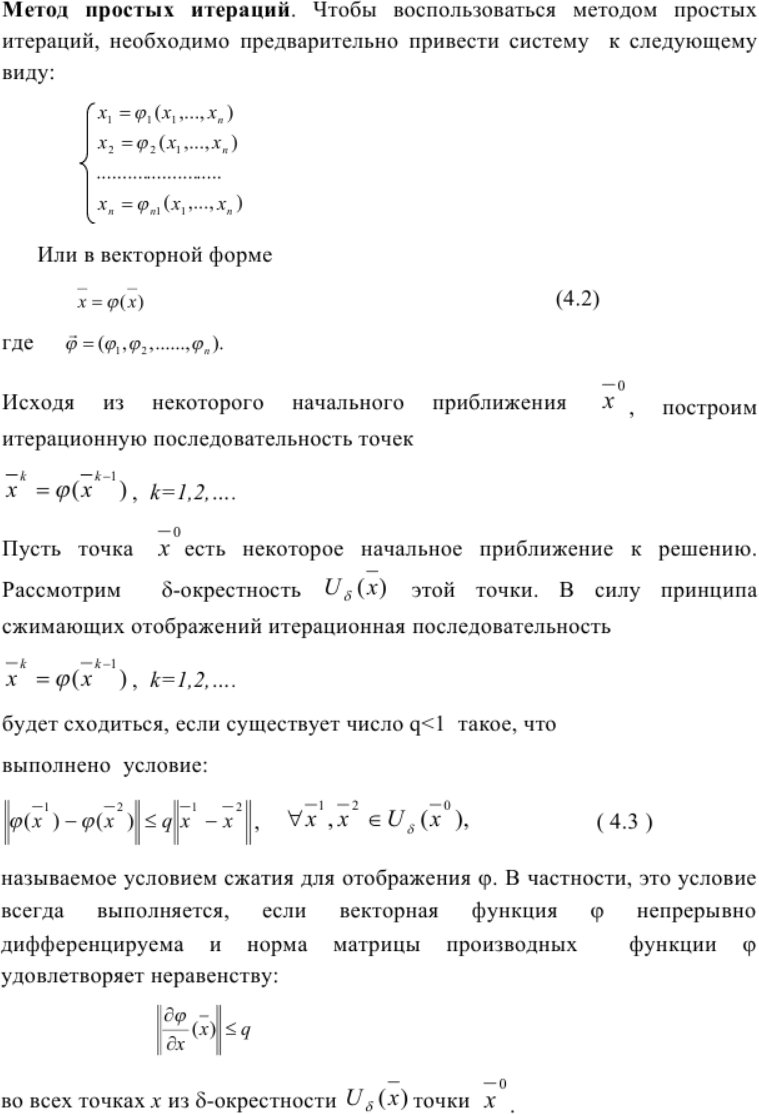
**Вариант 13**

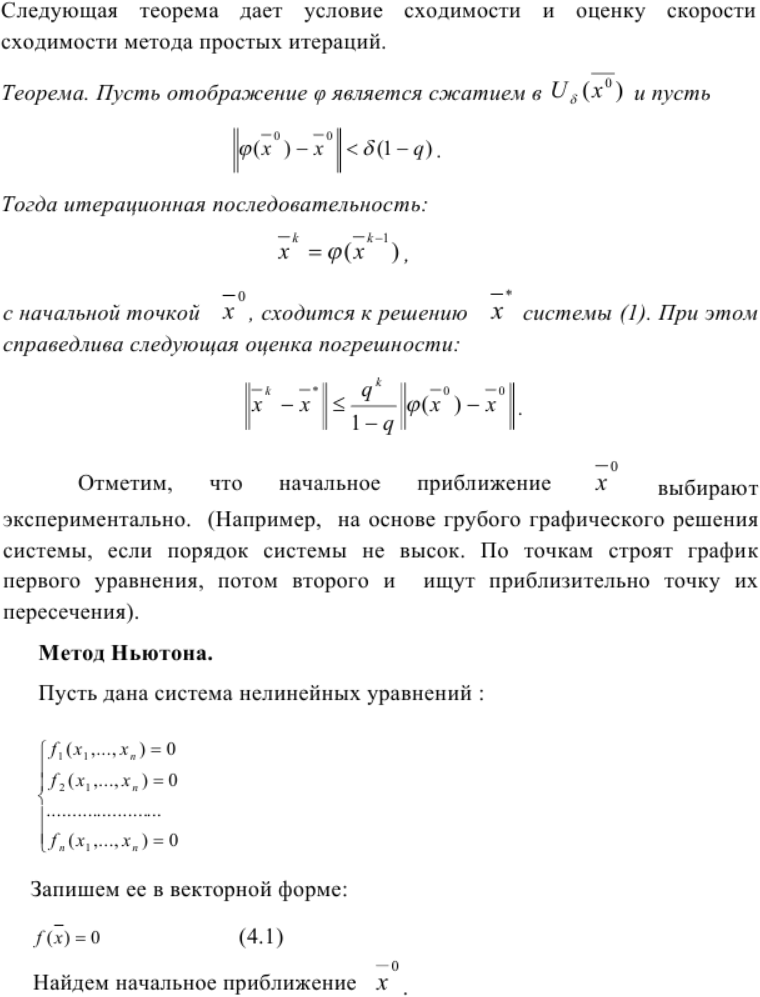
# **Цели выполнения задания**

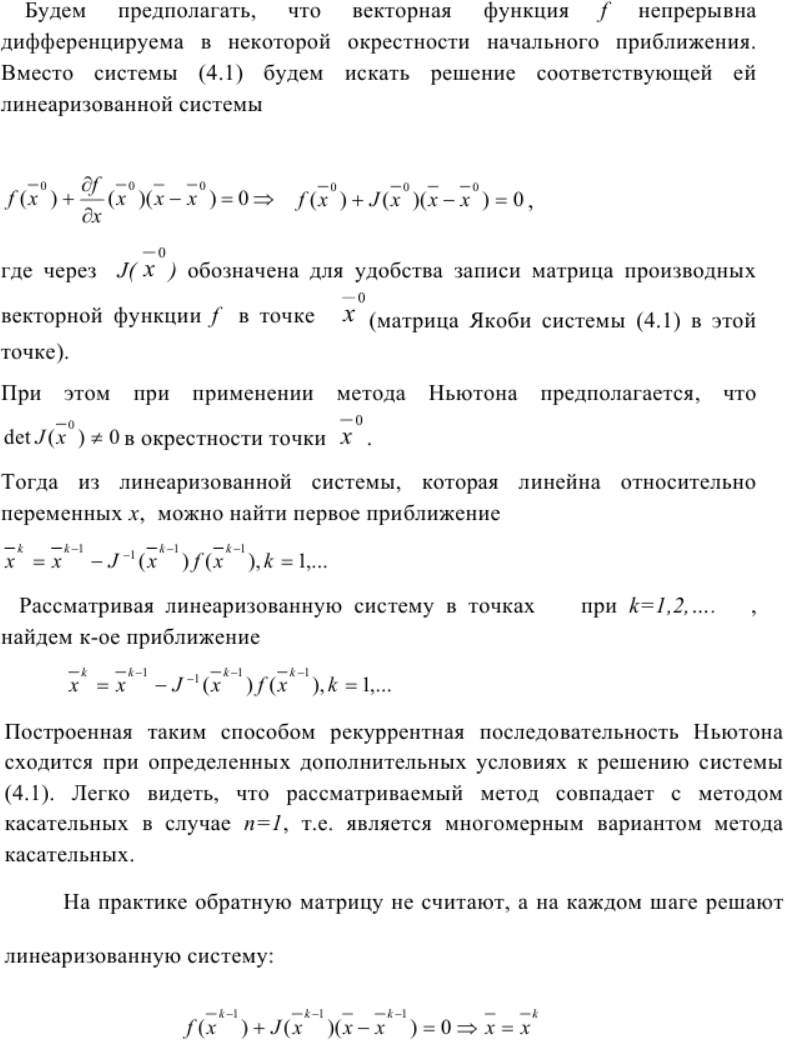
* Изучить численное решение систем нелинейных уравнений методами простых итераций и Ньютона.
* Построить и запрограммировать алгоритмы методов
* Численно решить тестовое задание.
* Сравнить число итераций, необходимого для достижения заданной точности вычисления разными методами.

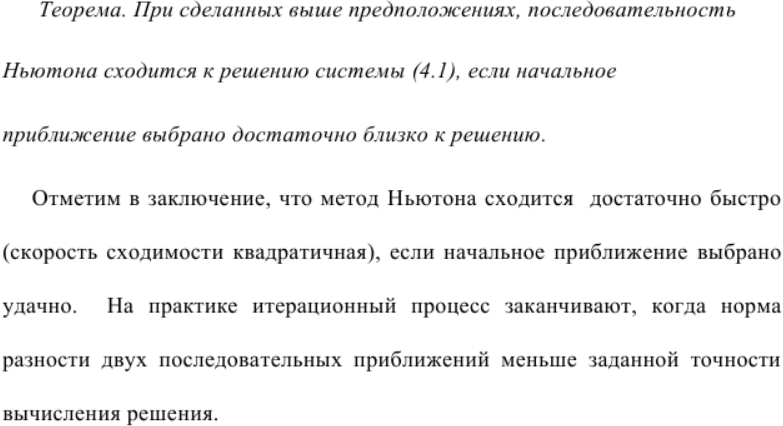
Краткие теоретические сведения









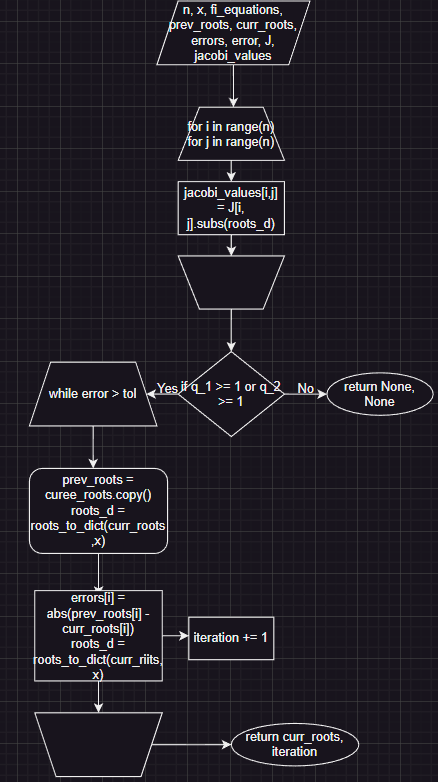


**Задание**

Решить систему нелинейных уравнений с точностью до 0,0001 методами простых итераций и Ньютона:

**Программная реализация**

1. iteration\_method

****

def get\_q(fi\_equation, approx, e=0.1):

    """

        Проверка условий сходимости системы уравнений.

        Для этого считаем матрицу якоби с частными производными и смотрим проверку на условие сходимости

    """

    x\_0 = approx[0]

    x\_fi = approx[1]

    x = symbols('x:2')

    e = 0.1

    x\_1 = random.uniform(x\_0 - e, x\_0 + e)

    x\_2 = random.uniform(x\_0 - e, x\_0 + e)

    q = (abs(fi\_equation.subs({x[0]: x\_1, x[1]: x\_fi}) - fi\_equation.subs({x[0]: x\_2, x[1]: x\_fi}))) / (abs(x\_1 - x\_2))

    return q

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)

    errors = np.zeros(shape=(n, ))

    error = 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:

        """

            Если отношение < q(q < 1), сходимость есть и далее считаем приближения до момента, пока отношение не > погрешности

        """

        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))

        if q\_1>=1 or q\_2>=1:

            print("q is greater than 1")

            return None, None

    iteration = 0

    while error > 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("some complex numbers")

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

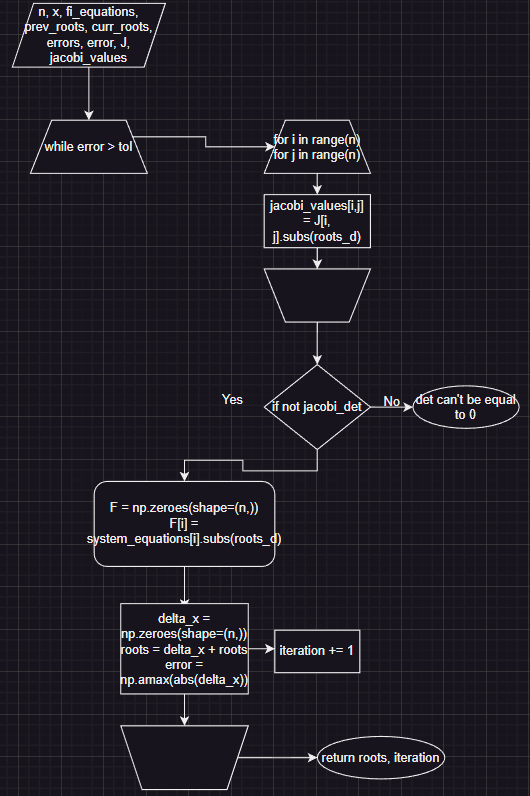
            roots\_d = roots\_to\_dict(curr\_roots, x)

        error = np.amax(errors)

        iteration += 1

    return curr\_roots, iteratio

1. newton\_method



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)

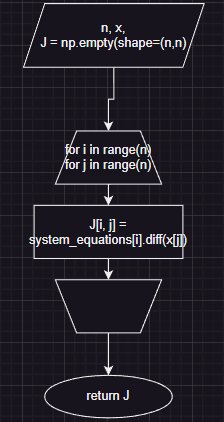
        delta\_x = np.linalg.solve(jacobi\_values, -1 \* F)

        roots = delta\_x + roots

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

    return roots, iteration

1. get\_jacobi



def get\_jacobi(system\_equations: np.array):

    n = system\_equations.shape[0]

    x = symbols(f'x:{n}')

    J = np.empty(shape=(n, n), dtype=core.add.Add)

    for i in range(n):

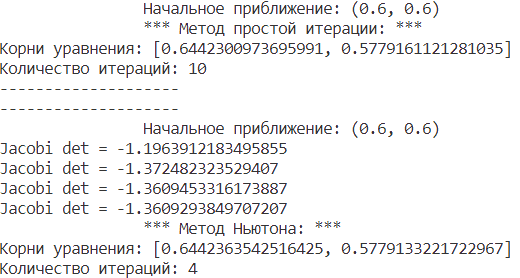
        for j in range(n):

            J[i, j] = system\_equations[i].diff(x[j])

    return J

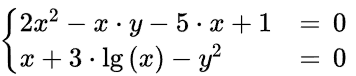
**Результаты работы программной реализации**

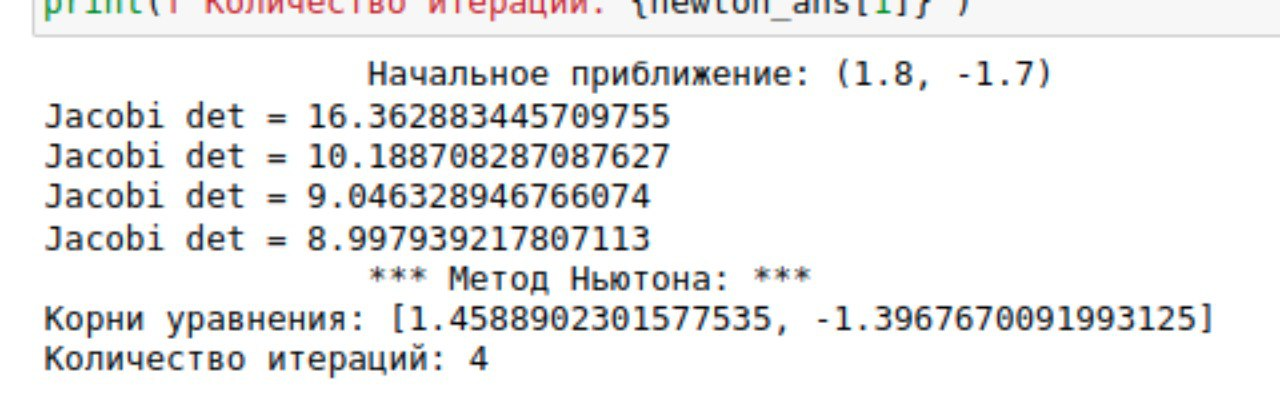
|  |  |
| --- | --- |
|  | |
| Начальное приближение: | |
| Метод простых итераций | Метод Ньютона |
|  |  |
| Количество итераций | |
| 10 | 4 |

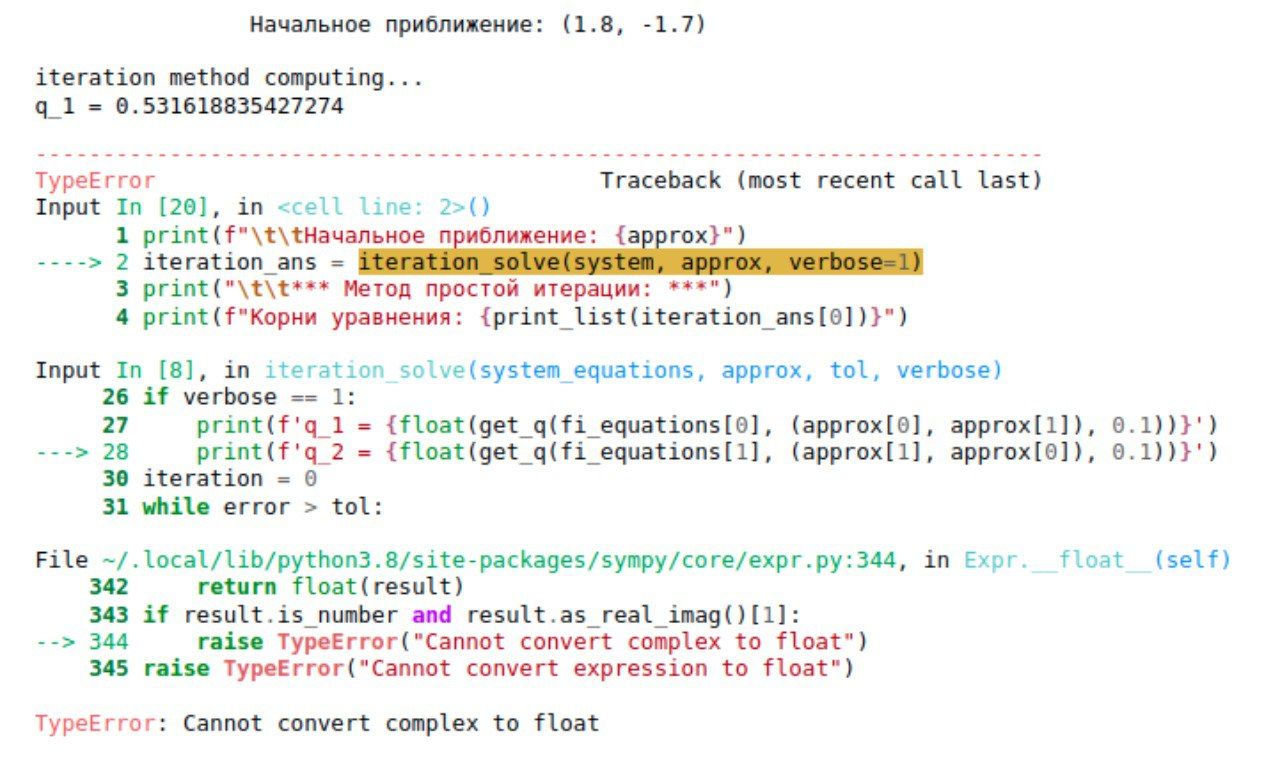


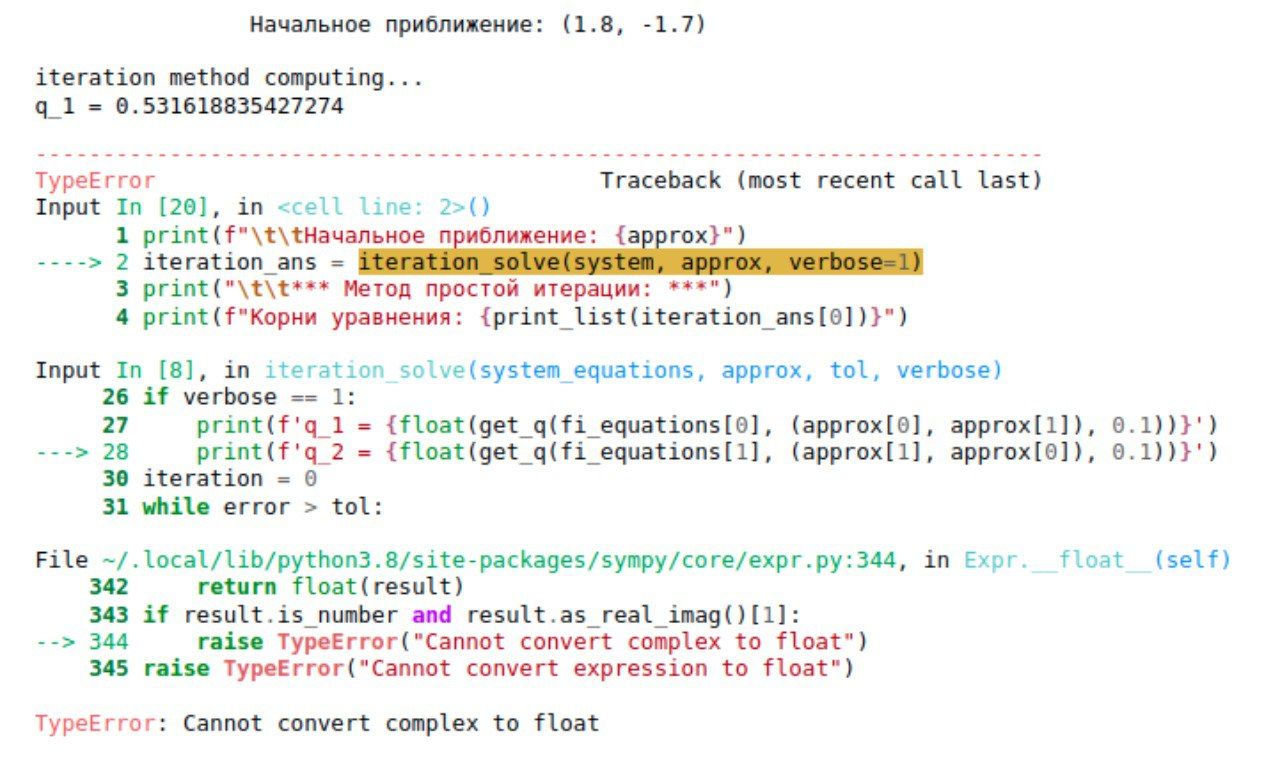
**Тестовые примеры**

1. Решить систему нелинейных уравнений с точностью до 0,0001 методами простых итераций и Ньютона:

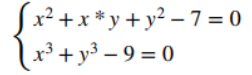
****

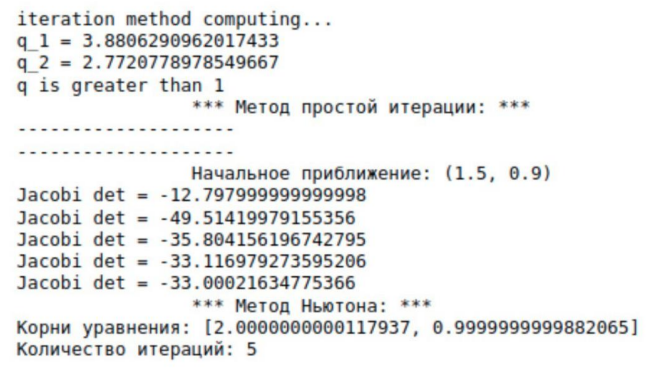






1. Решить систему нелинейных уравнений с точностью до 0,0001 методами простых итераций и Ньютона:

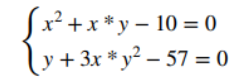




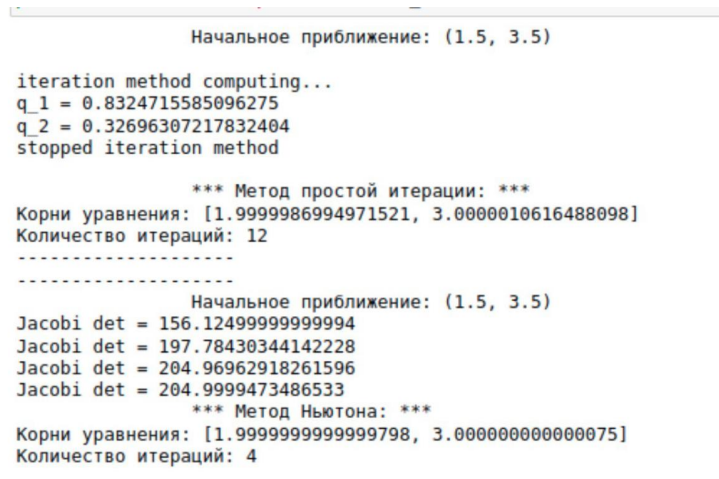
Значение q > 1при вычислении сходимости частных производных,

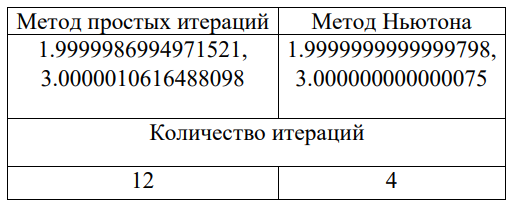
вследствсие чего решений нет

1. Решить систему нелинейных уравнений с точностью до 0,0001 методами простых итераций и Ньютона:

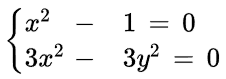


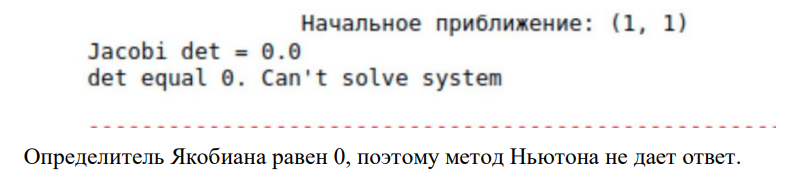
Начальное приближение для верхнего корня можно взять x = 1.5, y = 3.5





1. Решить систему нелинейных уравнений с точностью до 0,0001 методами простых итераций и Ньютона:



****

Так как графики функции накладываются друг на друга.

**Оценка погрешности**

0,35586656344112120,35584769202783595

**Выводы**

Таким образом, в ходе выполнения лабораторной работы были изучены методы численного решения систем нелинейных уравнений (метод простой итерации, метод Ньютона), составлен алгоритм и программа численного решения систем нелинейных уравнений методами простой итерации и Ньютона, проверена правильность работы программы на тестовых примерах, численно решена система уравнений заданного варианта.

Исходя из полученных результатов, можно сделать вывод о большей трудоемкости метода простых итераций. Метод простых итераций обладает линейной скоростью сходимости. Метод Ньютона сходится достаточно быстро (квадратичная скорость сходимости), если начальное приближение выбрано удачно