**Лабораторная работа № 2**

**«Численное решение систем нелинейных уравнений»**

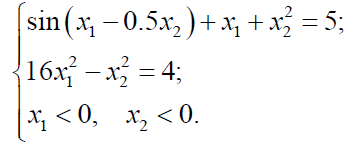
**Чеботаревский Никита, 2 курс, 7 группа**

**Постановка задачи**

Написать программу, которая решает данную систему нелинейных уравнений

*f* (*x*) = 0 c точностью ε = с помощью метода Ньютона, метода секущих. Начальное

приближение выбрать графически. Провести сравнительный анализ полученных результатов.



В лабораторной работе привести следующую информацию:

График для нахождения начального приближения *x*(0).

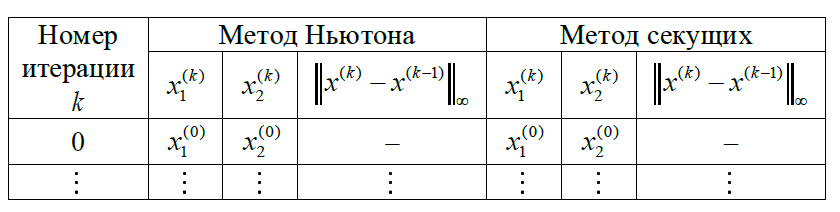
Алгоритм метода Ньютона.

Алгоритм метода секущих.

Результаты вычислительного эксперимента в виде таблицы 1.

4) Для каждого метода указать значение , где – полученное решение.

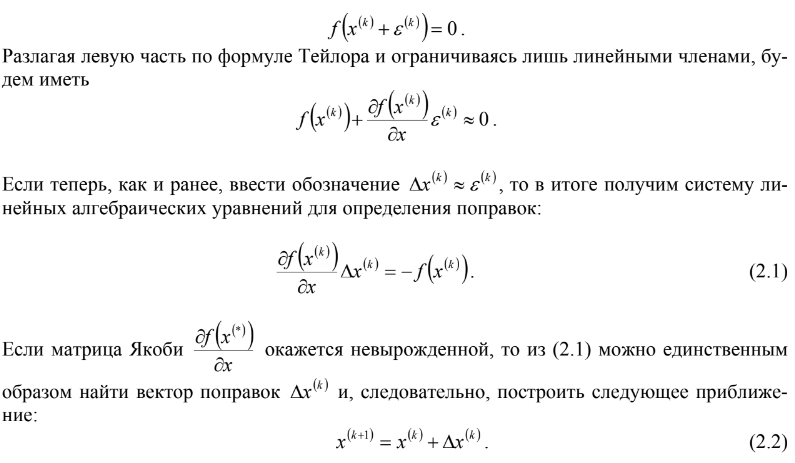
5) Листинг программы с комментариями.

**

**Теоретические сведения**

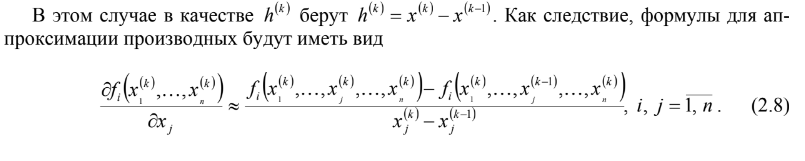
1. **Метод Ньютона**:

Основная идея метода Ньютона – линеаризация. Как и в случае одного уравнения введём вектор ошибки . Тогда имеем задачу вида:



Решение системы (2.1) можно искать любым способом решения СЛАУ.

1. **Метод секущих**:

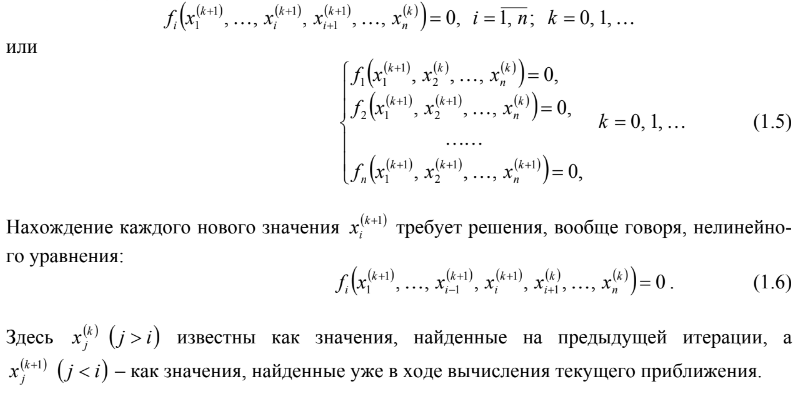


При этом на каждом шаге метода секущих решается система типа:



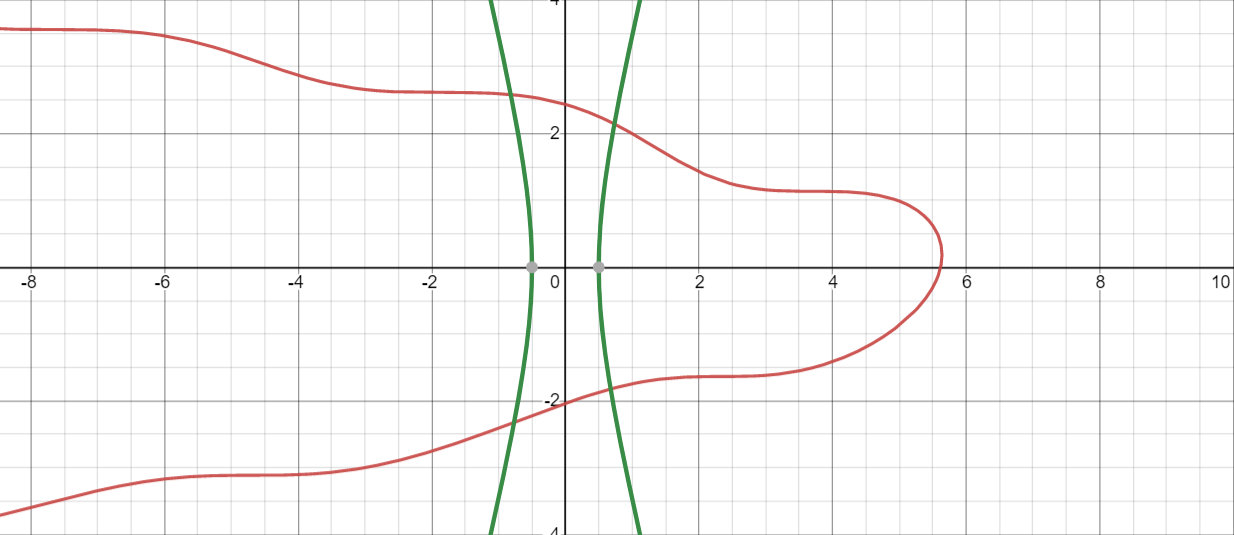
1. **Метод Гаусса-Зейделя**:

Метод Гаусса-Зейделя не требует предварительного преобразования системы к каноническому виду и имеет вид:



Для решения уравнения (1.6) можно применить любой из известных итерационных процессов (в нашем случае метод Ньютона). В результате имеем да вложенных итерационных процесса: один – вложенный, а другой – внутренний.

**График для нахождения начального приближения**



**Листинг программы**

*from* math *import* \*  
*import* numpy *as* np  
  
  
*def* count\_f1(x, y):  
 *return* sin(x - 0.5 \* y) + x + y \*\* 2 - 5  
  
  
*def* count\_f2(x, y):  
 *return* 16 \* x \*\* 2 - y \*\* 2 - 4  
  
  
*def* count\_partial\_derivative\_f1\_x(x, y):  
 *return* cos(x - 0.5 \* y) + 1  
  
  
*def* count\_partial\_derivative\_f1\_y(x, y):  
 *return* 2 \* y - 0.5 \* cos(x - 0.5 \* y)  
  
  
*def* count\_partial\_derivative\_f2\_x(x):  
 *return* 32 \* x  
  
  
*def* count\_partial\_derivative\_f2\_y(y):  
 *return* -2 \* y  
  
  
*def* newtons\_method(array\_x):  
 eps = 10 \*\* (-6)  
 matrix = np.array([[count\_partial\_derivative\_f1\_x(array\_x[0], array\_x[1]),  
 count\_partial\_derivative\_f1\_y(array\_x[0], array\_x[1])],  
 [count\_partial\_derivative\_f2\_x(array\_x[0]), count\_partial\_derivative\_f2\_y(array\_x[1])]])  
  
 f = np.array([-count\_f1(array\_x[0], array\_x[1]), -count\_f2(array\_x[0], array\_x[1])])  
 delta\_x = np.linalg.inv(matrix).dot(f)  
 x1 = array\_x + delta\_x  
 number\_of\_iteration = 1  
 print(f"{number\_of\_iteration}: {x1[0]}, {x1[1]}, {max(abs(delta\_x))}")  
 *while* max(abs(x1 - array\_x)) > eps:  
 array\_x = x1  
 matrix = np.array([[count\_partial\_derivative\_f1\_x(array\_x[0], array\_x[1]),  
 count\_partial\_derivative\_f1\_y(array\_x[0], array\_x[1])],  
 [count\_partial\_derivative\_f2\_x(array\_x[0]), count\_partial\_derivative\_f2\_y(array\_x[1])]])  
 f = np.array([-count\_f1(array\_x[0], array\_x[1]), -count\_f2(array\_x[0], array\_x[1])])  
 delta\_x = np.linalg.inv(matrix).dot(f)  
 x1 = array\_x + delta\_x  
 number\_of\_iteration += 1  
 print(f"{number\_of\_iteration}: {x1[0]}, {x1[1]}, {max(abs(delta\_x))}")  
  
 print(f"||f(x, y)|| = {max(abs(count\_f1(x1[0], x1[1])), abs(count\_f2(x1[0], x1[1])))}")  
  
  
*def* secant\_method(array\_x0):  
 eps = 10 \*\* (-6)  
 array\_x1 = np.array([-0.9, -2.9])  
 matrix = np.array([[(count\_f1(array\_x1[0], array\_x1[1]) - count\_f1(array\_x0[0], array\_x1[1])) /  
 (array\_x1[0] - array\_x0[0]), (count\_f1(array\_x1[0], array\_x1[1]) -  
 count\_f1(array\_x1[0], array\_x0[1])) / (  
 array\_x1[1] - array\_x0[1])],  
 [(count\_f2(array\_x1[0], array\_x1[1]) - count\_f2(array\_x0[0], array\_x1[1])) /  
 (array\_x1[0] - array\_x0[0]),  
 (count\_f2(array\_x1[0], array\_x1[1]) - count\_f2(array\_x1[0], array\_x0[1])) / (  
 array\_x1[1] - array\_x0[1])]])  
  
 f = np.array([-count\_f1(array\_x1[0], array\_x1[1]), -count\_f2(array\_x1[0], array\_x1[1])])  
 delta\_x = np.linalg.inv(matrix).dot(f)  
 x1 = array\_x1 + delta\_x  
 number\_of\_iterations = 1  
 print(f"{number\_of\_iterations}: {x1[0]}, {x1[1]}, {max(abs(delta\_x))}")  
 *while* max(abs(x1 - array\_x1)) > eps:  
 array\_x0 = array\_x1  
 array\_x1 = x1  
 matrix = np.array([[(count\_f1(array\_x1[0], array\_x1[1]) - count\_f1(array\_x0[0], array\_x1[1])) /  
 (array\_x1[0] - array\_x0[0]), (count\_f1(array\_x1[0], array\_x1[1]) -  
 count\_f1(array\_x1[0], array\_x0[1])) / (  
 array\_x1[1] - array\_x0[1])],  
 [(count\_f2(array\_x1[0], array\_x1[1]) - count\_f2(array\_x0[0], array\_x1[1])) /  
 (array\_x1[0] - array\_x0[0]),  
 (count\_f2(array\_x1[0], array\_x1[1]) - count\_f2(array\_x1[0], array\_x0[1])) / (  
 array\_x1[1] - array\_x0[1])]])  
  
 f = np.array([-count\_f1(array\_x1[0], array\_x1[1]), -count\_f2(array\_x1[0], array\_x1[1])])  
 delta\_x = np.linalg.inv(matrix).dot(f)  
 x1 = array\_x1 + delta\_x  
 number\_of\_iterations += 1  
 print(f"{number\_of\_iterations}: {x1[0]}, {x1[1]}, {max(abs(delta\_x))}")  
  
 print(f"||f(x, y)|| = {max(abs(count\_f1(x1[0], x1[1])), abs(count\_f2(x1[0], x1[1])))}")  
  
  
*def* gauss\_seidel\_method(x, y):  
 eps = 1e-6  
 number\_of\_iteration = 0  
 print(f"{number\_of\_iteration}: {x}, {y}, {max(abs(count\_f1(x, y)), abs(count\_f2(x, y)))}")  
 *while* max(abs(count\_f1(x, y)), abs(count\_f2(x, y))) > eps *or* number\_of\_iteration == 0:  
 x\_prev = x  
 x = x\_prev - count\_f2(x\_prev, y) / count\_partial\_derivative\_f2\_x(x\_prev)  
 *while* abs(x - x\_prev) > eps:  
 x\_prev = x  
 x = x\_prev - count\_f2(x\_prev, y) / count\_partial\_derivative\_f2\_x(x\_prev)  
  
 y\_prev = y  
 y = y\_prev - count\_f1(x, y\_prev) / count\_partial\_derivative\_f1\_y(x, y\_prev)  
 *while* abs(y - y\_prev) > eps:  
 y\_prev = y  
 y = y\_prev - count\_f1(x, y\_prev) / count\_partial\_derivative\_f1\_y(x, y\_prev)  
  
 number\_of\_iteration += 1  
 print(f"{number\_of\_iteration}: {x}, {y}, {max(abs(count\_f1(x, y)), abs(count\_f2(x, y)))}")  
  
 print(f"||f(x, y)|| = {max(abs(count\_f1(x, y)), abs(count\_f2(x, y)))}")  
  
  
x0 = np.array([-1, -3])  
print("Newton method:")  
newtons\_method(x0)  
print()  
print("Secant method:")  
secant\_method(x0)  
print()  
print("Gauss Seidel method:")  
print(gauss\_seidel\_method(x0[0], x0[1]))

**Результаты программы**

