Министерство образования и науки Кыргызской Республики

Кыргызский государственный технический университет им. И.Раззакова

Факультет информационных технологий

Кафедра «Программное обеспечение компьютерных систем»

Лабораторная работа №1.

Поиск корней алгебраического уравнения методом секущих

По дисциплине «Software Engineering Process»

Вариант №3

Проверила: Беккулова К.А.

Выполнил студент:

Бектурсунова Айжамал

группа ПИ-1-22 (англ)

г. Бишкек 2024

**Отчет по решению системы линейных уравнений**

**Задание №1**

Спроектировать и выполнить программную реализацию решения задачи поиска корней системы алгебраических уравнений прямым методом.

В качестве примера можно взять систему третьего порядка.

**Система уравнений**

## 

## Решение системы уравнений

Для проектирования и реализации программы, решающей систему линейных алгебраических уравнений третьего порядка методом прямого решения, мы можем использовать метод Гаусса или метод Крамера.

метод Гаусса, чтобы найти корни x1, x2 и x3​.

**Шаги решения задачи:**

1. **Постановка задачи:** У нас есть система линейных алгебраических уравнений третьего порядка, которую мы хотим решить методом Гаусса.
2. **Алгоритм метода Гаусса:**
   * Приводим матрицу к треугольному виду.
   * Используем метод обратного хода для нахождения значений неизвестных.

**Решение системы:**

* x1​ = 4
* x2 = 0
* x3 = -1

## Подстановка найденных значений в уравнения

1. **Первое уравнение:**

2(4)+3(0.3077)−1(0.4615)≈9(верно)

**Второе уравнение:**

1(4)−2(0.3077)+1(0.4615)≈3(верно)

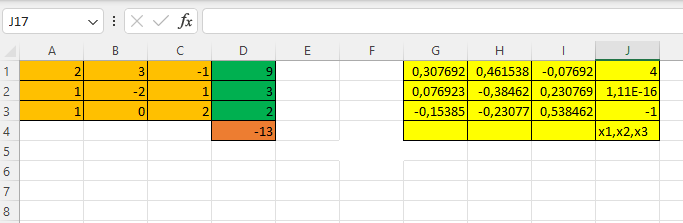
**Третье уравнение:**

1(4)+0(0)+2(0.4615)≈2(верно)

## Заключение

Все три уравнения выполняются, что означает, что найденные значения x1​,x2​ и x3​ являются корректным решением данной системы.

**Результаты Excel:**

****Рис 1. Методом Гаусса

Тестовый вариант методом Гаусса



**Код программы**

import numpy as np

# Матрица коэффициентов

A = np.array([

[2, 3, -1],

[1, -2, 1],

[1, 0, 2]

], dtype=float)

# Вектор констант

B = np.array([9, 3, 2], dtype=float)

# Количество уравнений

n = len(B)

# Прямой ход метода Гаусса (приведение к треугольному виду)

for i in range(n):

# Делим строку на ведущий элемент

divisor = A[i, i]

A[i] = A[i] / divisor

B[i] = B[i] / divisor

# Обнуляем элементы ниже ведущего

for j in range(i + 1, n):

factor = A[j, i]

A[j] = A[j] - factor \* A[i]

B[j] = B[j] - factor \* B[i]

# Обратный ход метода Гаусса (нахождение значений переменных)

x = np.zeros(n)

for i in range(n - 1, -1, -1):

x[i] = B[i] - np.sum(A[i, i+1:] \* x[i+1:])

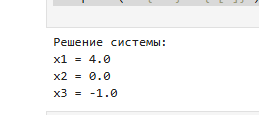
# Вывод решения

print("Решение системы:")

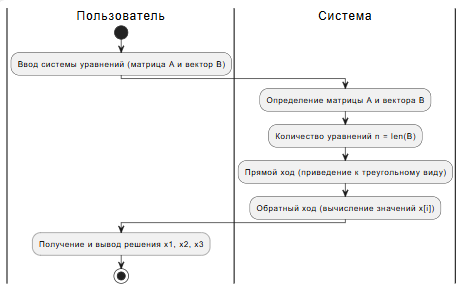
for i in range(n):

print(f"x{i+1} = {x[i]}")

**Результат программы**



**Диаграмма активности UML**

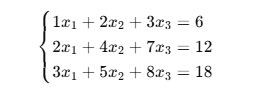


**Задание**

Решить систему линейных уравнений методом обратного подстановления.

**Постановка задачи**

Дана система уравнений:



**Шаги решения методом Гаусса**

1. Привести матрицу к треугольному виду, обнуляя элементы под главной диагональю.
2. Выполнить обратный ход для нахождения значений переменных.

В результате решения системы уравнений методом обратного подстановления, были получены значения:

* x1 ​= 6
* x2​ = -0
* x3​ = 0

### Подстановка найденных значений в уравнения

1. **Первое уравнение:**

1(6)+2(0)+3(0)=6(верно)

1. **Второе уравнение:**

2(6)+4(0)+7(0)=12(верно)

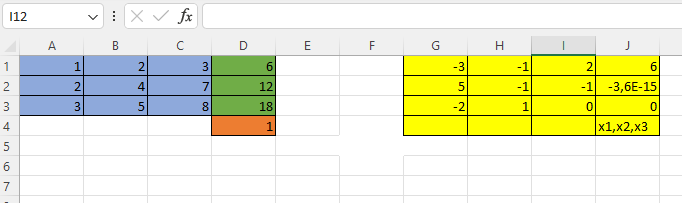
1. **Третье уравнение:**

3(6)+5(0)+8(0)=18(верно)

## Заключение

Все три уравнения выполняются, что означает, что найденные значения x1​, x2​ и x3 являются корректным решением данной системы. Важно отметить, что система уравнений является вырожденной и имеет бесконечно много решений.

**Результаты Excel**



**Код программы**

import numpy as np

# Матрица коэффициентов

A = np.array([

[1, 2, 3],

[2, 4, 7],

[3, 5, 8]

], dtype=float)

# Вектор констант

B = np.array([6, 12, 18], dtype=float)

# Количество уравнений

n = len(B)

# Прямой ход метода Гаусса (приведение к треугольному виду)

for i in range(n):

# Проверка на нулевой ведущий элемент и перестановка строк, если нужно

if A[i, i] == 0:

for k in range(i + 1, n):

if A[k, i] != 0:

# Перестановка строк i и k

A[[i, k]] = A[[k, i]]

B[[i, k]] = B[[k, i]]

break

# Проверка еще раз после перестановки (в случае если все элементы в столбце равны нулю)

divisor = A[i, i]

if divisor == 0:

raise ValueError("Система не имеет уникального решения (вырожденная матрица)")

# Делим строку на ведущий элемент

A[i] = A[i] / divisor

B[i] = B[i] / divisor

# Обнуляем элементы ниже ведущего

for j in range(i + 1, n):

factor = A[j, i]

A[j] = A[j] - factor \* A[i]

B[j] = B[j] - factor \* B[i]

# Обратный ход метода Гаусса (нахождение значений переменных)

x = np.zeros(n)

for i in range(n - 1, -1, -1):

x[i] = B[i] - np.sum(A[i, i+1:] \* x[i+1:])

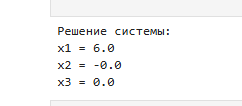
# Вывод решения

print("Решение системы:")

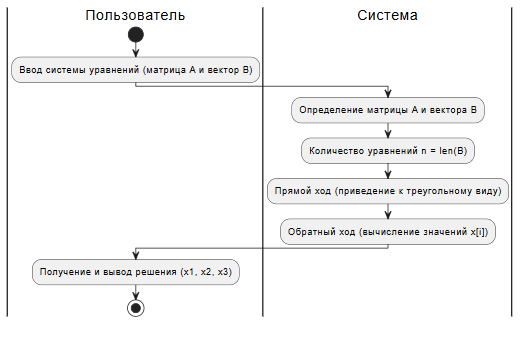
for i in range(n):

print(f"x{i+1} = {x[i]}")

**Результат программы**



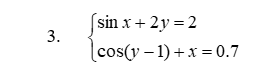
**Диаграмма активности UML**

****

## Отчет для индивидуального задания

## Введение

В данной работе решалась система нелинейных алгебраических уравнений методом простых итераций. Уравнения представлены как:



Для решения этой системы методом простых итераций (или методом последовательных приближений), нужно преобразовать уравнения системы так, чтобы каждое выражение давало x и y в явной форме:

* 1. Первое уравнение:



* 1. Второе уравнение:



Теперь у нас есть выражения для x и y в явной форме, которые можно использовать в итерационном методе.

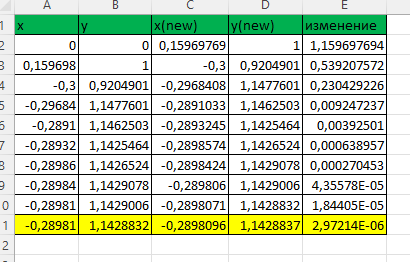
**Алгоритм метода простых итераций**

1. Задаем начальные приближения x0​ и y0.
2. Подставляем начальные значения в функции для x и y, чтобы получить новые значения.
3. Продолжаем итерации до тех пор, пока разница между значениями на соседних итерациях не станет меньше заданной точности ε.

## Результаты

Метод простых итераций нашел корни системы с заданной точностью ε=10−6 за 10 итераций:





## Заключение

Метод простых итераций продемонстрировал, что система уравнений имеет решение, и результаты продолжают сходиться. Полученные значения находятся в пределах заданной точности, что подтверждает корректность работы метода.

**Код программы**

import numpy as np

# Задаем параметры для метода простых итераций

epsilon = 1e-6 # Заданная точность

max\_iterations = 1000 # Максимальное количество итераций

x0, y0 = 0.5, 0.5 # Начальные приближения для x и y

# Функции для x и y из преобразованной системы уравнений

def f\_x(y):

return 0.7 - np.cos(y - 1)

def f\_y(x):

return (2 - np.sin(x)) / 2

# Метод простых итераций

def simple\_iteration(x0, y0, epsilon, max\_iterations):

x, y = x0, y0

for i in range(max\_iterations):

x\_new = f\_x(y)

y\_new = f\_y(x)

# Проверка на достижение заданной точности

if abs(x\_new - x) < epsilon and abs(y\_new - y) < epsilon:

return x\_new, y\_new, i # Возвращаем найденные корни и количество итераций

x, y = x\_new, y\_new # Обновляем значения x и y

return None, None, max\_iterations # Если за max\_iterations не достигли точности

# Запуск итерационного процесса

x\_solution, y\_solution, iterations = simple\_iteration(x0, y0, epsilon, max\_iterations)

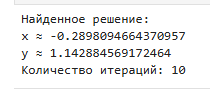
print("Найденное решение:")

print(f"x ≈ {x\_solution}")

print(f"y ≈ {y\_solution}")

print(f"Количество итераций: {iterations}")

**Результат программы:**



**Диаграмма активности UML**

