Отчёт по лабораторной работе №1

«Решение систем линейных алгебраических уравнений»

Выполнили:

Адаменко С. С.

Гневнов А. Е.

Суворов Р.М

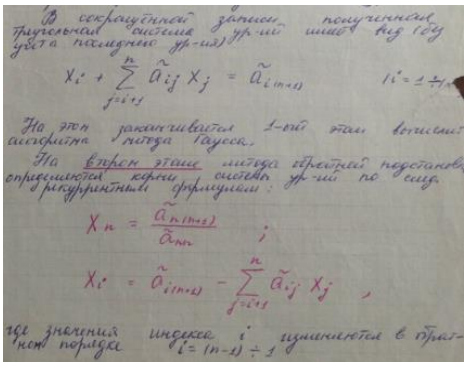
Отчёт Адаменко С.С.

Используемое оборудование: ПК, языки программирования: Python; используемые сторонние библиотеки: Numpy; среда разработки Visual Studio Code.

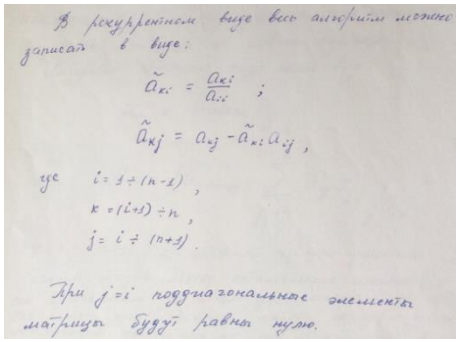
Постановка задачи: Ознакомиться с теорией вопроса (представлена в материалах папок: Теория_1_Классический_метод; Теория_2_Метод оптимального исключения; Теория_3_Гаусса-Жордана).

Математическая модель:

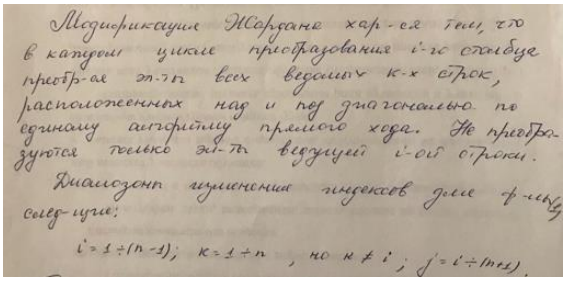
1. Классический метод:



2. Метод оптимального исключения:



3. Метод Гаусса-Жордана:



Код программы:

```
import numpy as np
from tabulate import tabulate


def gaussian_elimination(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])

    for i in range(num_rows):
        # Поиск максимального элемента в столбце
        max_row = i
        for j in range(i + 1, num_rows):
            if abs(matrix[j][i]) > abs(matrix[max_row][i]):
                max_row = j

        # Перестановка строк
        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        # Приведение к верхнетреугольному виду
        for j in range(i + 1, num_rows):
            ratio = matrix[j][i] / matrix[i][i]
            for k in range(i, num_cols):
                matrix[j][k] -= ratio * matrix[i][k]
```

```
    return matrix


def back_substitution(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])
    solution = np.zeros(num_rows)

    for i in range(num_rows - 1, -1, -1):
        solution[i] = matrix[i][num_cols - 1] / matrix[i][i]
        for j in range(i - 1, -1, -1):
            matrix[j][num_cols - 1] -= matrix[j][i] * solution[i]

    return solution


def gauss_jordan(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])

    for i in range(num_rows):
        # Поиск максимального элемента в столбце
        max_row = i
        for j in range(i + 1, num_rows):
            if abs(matrix[j][i]) > abs(matrix[max_row][i]):
                max_row = j

        # Перестановка строк
```

```
        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        # Приведение к единичной матрице
        pivot = matrix[i][i]
        for j in range(i, num_cols):
            matrix[i][j] /= pivot

        # Обновление остальных строк
        for j in range(num_rows):
            if j != i:
                ratio = matrix[j][i]
                for k in range(i, num_cols):
                    matrix[j][k] -= ratio * matrix[i][k]

    return matrix

def gauss_elimination(coeff_matrix, const_vector):
    n = len(coeff_matrix)
    # Создаем расширенную матрицу
    augmented_matrix = np.column_stack((coeff_matrix, const_vector))
    # Поиск максимального элемента в столбце
    for i in range(n):
        max_row = i
        for k in range(i + 1, n):
            if abs(augmented_matrix[k, i]) > abs(augmented_matrix[max_row, i]):
                max_row = k
        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]
    # Прямой ход
        for j in range(i + 1, n):
```

```
            ratio = augmented_matrix[j,i] / augmented_matrix[i, i]
            augmented_matrix[j, i:] -= ratio * augmented_matrix[i, i:]
    # Обратный ход
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = augmented_matrix[i, -1] / augmented_matrix[i, i]
        for j in range(i - 1, -1, -1):
            augmented_matrix[j, -1] -= augmented_matrix[j, i] * x[i]
    return x

# Ввод исходных данных
matrix = []
n = int(input("Введите размерность матрицы: "))
print("Введите элементы матрицы:")
for i in range(n):
    row = list(map(float, input().split()))
    matrix.append(row)

print("Исходная матрица:")
print(tabulate(matrix, tablefmt="fancy_grid"))

# Прямой ход метода Гаусса
reduced_matrix = gaussian_elimination(matrix)
print("Матрица после прямого хода:")
print(tabulate(reduced_matrix, tablefmt="fancy_grid"))

# Обратный ход метода Гаусса
solution = back_substitution(reduced_matrix)
print("Решение СЛУ:")
```

```
print(solution)


# Реализация метода Гаусса-Жордана
gauss_jordan_matrix = gauss_jordan(matrix)
print("Результат метода Гаусса-Жордана:")
print(tabulate(gauss_jordan_matrix, tablefmt="fancy_grid"))


# Пример использования
A = np.array(matrix, dtype = float)
B = np.array([23,32,33,31])
x = gauss_elimination(A, B)
print("Решение", x)
```

Результаты программы для контрольного примера:

A				*B*
5	7	6	5	23
7	10	8	7	32
6	8	10	9	33
5	7	9	10	31

```
Введите размерность матрицы: 4
Введите элементы матрицы:
5 7 6 5 23
7 10 8 7 32
6 8 10 9 33
5 7 9 10 31
Исходная матрица:
```

5	7	6	5	23
7	10	8	7	32
6	8	10	9	33
5	7	9	10	31

Матрица после прямого хода:

7	10	8	7	32
0	-0.571429	3.14286	3	5.57143
0	0	2.5	4.25	6.75
0	0	0	0.1	0.1

Решение СЛУ:

[1. 1. 1. 1.]

Результат метода Гаусса-Жордана:

1	0	0	0	-2.62857
0	1	0	0	2.4
0	0	1	0	-0.7
0	0	0	1	1

```
Решение [1. 1. 1. 1.]
```

и для значений элементов столбца свободных членов на

1) 0,1

```
Введите элементы матрицы:
5 7 6 5 23.1
7 10 8 7 32.1
6 8 10 9 33.1
5 7 9 10 31.1
Исходная матрица:
```

5	7	6	5	23.1
7	10	8	7	32.1
6	8	10	9	33.1
5	7	9	10	31.1

Матрица после прямого хода:

7	10	8	7	32.1
0	-0.571429	3.14286	3	5.58571
0	0	2.5	4.25	6.775
0	0	0	0.1	0.13

```
Решение СЛУ:
[ 3.  -0.2  0.5  1.3]
Результат метода Гаусса-Жордана:
```

1	0	0	0	7.62571
0	1	0	0	-2.78
0	0	1	0	-1.71
0	0	0	1	1.3

```
Решение [ 3.  -0.2  0.5  1.3]
```

2) 0,001

```
Введите размерность матрицы: 4
Введите элементы матрицы:
5 7 6 5 23.001
7 10 8 7 32.001
6 8 10 9 33.001
5 7 9 10 31.001
Исходная матрица:
```

5	7	6	5	23.001
7	10	8	7	32.001
6	8	10	9	33.001
5	7	9	10	31.001

Матрица после прямого хода:

7	10	8	7	32.001
0	-0.571429	3.14286	3	5.57157
0	0	2.5	4.25	6.75025
0	0	0	0.1	0.1003

```
Решение СЛУ:
[1.02  0.988 0.995 1.003]
Результат метода Гаусса-Жордана:
```

1	0	0	0	-2.52603
0	1	0	0	2.3482
0	0	1	0	-0.7101
0	0	0	1	1.003

3) 0,01

```
Введите размерность матрицы: 4
Введите элементы матрицы:
5 7 6 5 23.01
7 10 8 7 32.01
6 8 10 9 33.01
5 7 9 10 31.01
Исходная матрица:
```

5	7	6	5	23.01
7	10	8	7	32.01
6	8	10	9	33.01
5	7	9	10	31.01

Матрица после прямого хода:

7	10	8	7	32.01
0	-0.571429	3.14286	3	5.57286
0	0	2.5	4.25	6.7525
0	0	0	0.1	0.103

Решение СЛУ:

[1.2 0.88 0.95 1.03]

Результат метода Гаусса-Жордана:

1	0	0	0	-1.60314
0	1	0	0	1.882
0	0	1	0	-0.801
0	0	0	1	1.03

```
Решение [1.2  0.88 0.95 1.03]
```

Вывод:

Прямой ход метода Гаусса хорошо справляется с системой во всех трех случаях. Обратный ход метода Гаусса показывает, что при элементах 23 и 0,1 система становится не доопределённой что неудивительно, так как в этих случаях есть свободные переменные. Метод Гаусса-Жордана дает матрицу с единичными значениями на диагонали во всех случаях, но значения свободных членов могут быть с погрешностью из-за использования чисел с плавающей точкой

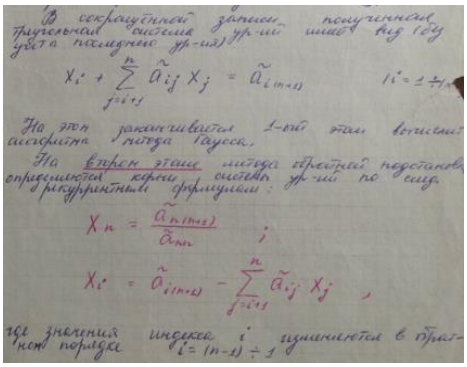
Отчёт Гневнов А. Е..

Используемое оборудование: ПК, языки программирования: Python; используемые сторонние библиотеки: Numpy; среда разработки Visual Studio Code.

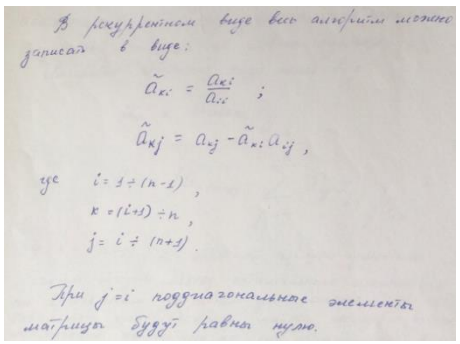
Постановка задачи: Ознакомиться с теорией вопроса (представлена в материалах папок: Теория_1_Классический_метод; Теория_2_Метод оптимального исключения; Теория_3_Гаусса-Жордана).

Математическая модель:

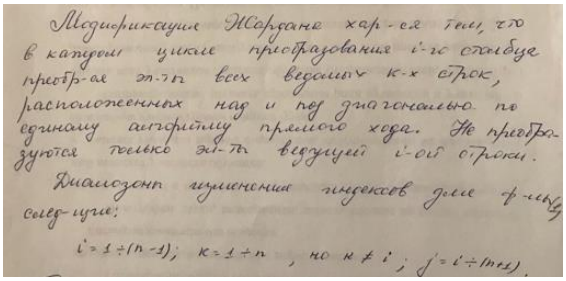
1. Классический метод:



2. Метод оптимального исключения:



3. Метод Гаусса-Жордана:



Код программы:

```
import numpy as np
from tabulate import tabulate


def gaussian_elimination(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])

    for i in range(num_rows):
        # Поиск максимального элемента в столбце
        max_row = i
        for j in range(i + 1, num_rows):
            if abs(matrix[j][i]) > abs(matrix[max_row][i]):
                max_row = j

        # Перестановка строк
        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        # Приведение к верхнетреугольному виду
        for j in range(i + 1, num_rows):
            ratio = matrix[j][i] / matrix[i][i]
            for k in range(i, num_cols):
                matrix[j][k] -= ratio * matrix[i][k]
```

```
    return matrix


def back_substitution(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])
    solution = np.zeros(num_rows)

    for i in range(num_rows - 1, -1, -1):
        solution[i] = matrix[i][num_cols - 1] / matrix[i][i]
        for j in range(i - 1, -1, -1):
            matrix[j][num_cols - 1] -= matrix[j][i] * solution[i]

    return solution


def gauss_jordan(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])

    for i in range(num_rows):
        # Поиск максимального элемента в столбце
        max_row = i
        for j in range(i + 1, num_rows):
            if abs(matrix[j][i]) > abs(matrix[max_row][i]):
                max_row = j

        # Перестановка строк
```

```
        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        # Приведение к единичной матрице
        pivot = matrix[i][i]
        for j in range(i, num_cols):
            matrix[i][j] /= pivot

        # Обновление остальных строк
        for j in range(num_rows):
            if j != i:
                ratio = matrix[j][i]
                for k in range(i, num_cols):
                    matrix[j][k] -= ratio * matrix[i][k]

    return matrix

def gauss_elimination(coeff_matrix, const_vector):
    n = len(coeff_matrix)
    # Создаем расширенную матрицу
    augmented_matrix = np.column_stack((coeff_matrix, const_vector))
    # Поиск максимального элемента в столбце
    for i in range(n):
        max_row = i
        for k in range(i + 1, n):
            if abs(augmented_matrix[k, i]) > abs(augmented_matrix[max_row, i]):
                max_row = k
        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]
    # Прямой ход
        for j in range(i + 1, n):
```

```
            ratio = augmented_matrix[j,i] / augmented_matrix[i, i]
            augmented_matrix[j, i:] -= ratio * augmented_matrix[i, i:]
    # Обратный ход
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = augmented_matrix[i, -1] / augmented_matrix[i, i]
        for j in range(i - 1, -1, -1):
            augmented_matrix[j, -1] -= augmented_matrix[j, i] * x[i]
    return x

# Ввод исходных данных
matrix = []
n = int(input("Введите размерность матрицы: "))
print("Введите элементы матрицы:")
for i in range(n):
    row = list(map(float, input().split()))
    matrix.append(row)

print("Исходная матрица:")
print(tabulate(matrix, tablefmt="fancy_grid"))

# Прямой ход метода Гаусса
reduced_matrix = gaussian_elimination(matrix)
print("Матрица после прямого хода:")
print(tabulate(reduced_matrix, tablefmt="fancy_grid"))

# Обратный ход метода Гаусса
solution = back_substitution(reduced_matrix)
print("Решение СЛУ:")
```

```
print(solution)


# Реализация метода Гаусса-Жордана
gauss_jordan_matrix = gauss_jordan(matrix)
print("Результат метода Гаусса-Жордана:")
print(tabulate(gauss_jordan_matrix, tablefmt="fancy_grid"))


# Пример использования
A = np.array(matrix, dtype = float)
B = np.array([23,32,33,31])
x = gauss_elimination(A, B)
print("Решение", x)
```

Результаты программы для контрольного примера:

A				*B*
5	7	6	5	23
7	10	8	7	32
6	8	10	9	33
5	7	9	10	31

```
Введите размерность матрицы: 4
Введите элементы матрицы:
5 7 6 5 23
7 10 8 7 32
6 8 10 9 33
5 7 9 10 31
Исходная матрица:
```

5	7	6	5	23
7	10	8	7	32
6	8	10	9	33
5	7	9	10	31

Матрица после прямого хода:

7	10	8	7	32
0	-0.571429	3.14286	3	5.57143
0	0	2.5	4.25	6.75
0	0	0	0.1	0.1

Решение СЛУ:

[1. 1. 1. 1.]

Результат метода Гаусса-Жордана:

1	0	0	0	-2.62857
0	1	0	0	2.4
0	0	1	0	-0.7
0	0	0	1	1

```
Решение [1. 1. 1. 1.]
```

и для значений элементов столбца свободных членов на

1) 0,1

```
Введите элементы матрицы:
5 7 6 5 23.1
7 10 8 7 32.1
6 8 10 9 33.1
5 7 9 10 31.1
Исходная матрица:
```

5	7	6	5	23.1
7	10	8	7	32.1
6	8	10	9	33.1
5	7	9	10	31.1

Матрица после прямого хода:

7	10	8	7	32.1
0	-0.571429	3.14286	3	5.58571
0	0	2.5	4.25	6.775
0	0	0	0.1	0.13

```
Решение СЛУ:
[ 3.  -0.2  0.5  1.3]
Результат метода Гаусса-Жордана:
```

1	0	0	0	7.62571
0	1	0	0	-2.78
0	0	1	0	-1.71
0	0	0	1	1.3

```
Решение [ 3.  -0.2  0.5  1.3]
```

2) 0,001

```
Введите размерность матрицы: 4
Введите элементы матрицы:
5 7 6 5 23.001
7 10 8 7 32.001
6 8 10 9 33.001
5 7 9 10 31.001
Исходная матрица:
```

5	7	6	5	23.001
7	10	8	7	32.001
6	8	10	9	33.001
5	7	9	10	31.001

Матрица после прямого хода:

7	10	8	7	32.001
0	-0.571429	3.14286	3	5.57157
0	0	2.5	4.25	6.75025
0	0	0	0.1	0.1003

```
Решение СЛУ:
[1.02  0.988 0.995 1.003]
Результат метода Гаусса-Жордана:
```

1	0	0	0	-2.52603
0	1	0	0	2.3482
0	0	1	0	-0.7101
0	0	0	1	1.003

3) 0,01

```
Введите размерность матрицы: 4
Введите элементы матрицы:
5 7 6 5 23.01
7 10 8 7 32.01
6 8 10 9 33.01
5 7 9 10 31.01
Исходная матрица:
```

5	7	6	5	23.01
7	10	8	7	32.01
6	8	10	9	33.01
5	7	9	10	31.01

Матрица после прямого хода:

7	10	8	7	32.01
0	-0.571429	3.14286	3	5.57286
0	0	2.5	4.25	6.7525
0	0	0	0.1	0.103

Решение СЛУ:

[1.2 0.88 0.95 1.03]

Результат метода Гаусса-Жордана:

1	0	0	0	-1.60314
0	1	0	0	1.882
0	0	1	0	-0.801
0	0	0	1	1.03

```
Решение [1.2  0.88 0.95 1.03]
```

Вывод:

Прямой ход метода Гаусса хорошо справляется с системой во всех трех случаях. Обратный ход метода Гаусса показывает, что при элементах 23 и 0,1 система становится не доопределённой что неудивительно, так как в этих случаях есть свободные переменные. Метод Гаусса-Жордана дает матрицу с единичными значениями на диагонали во всех случаях, но значения свободных членов могут быть с погрешностью из-за использования чисел с плавающей точкой

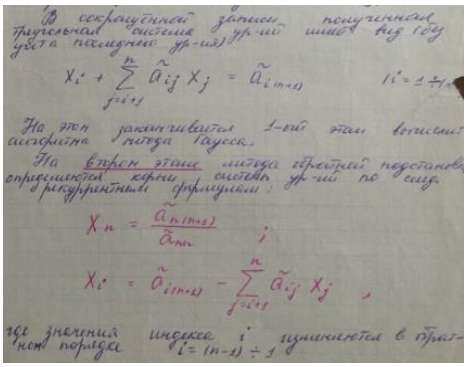
Отчёт Суворов Р.М

Используемое оборудование: ПК, языки программирования: Python; используемые сторонние библиотеки: Numpy; среда разработки Visual Studio Code.

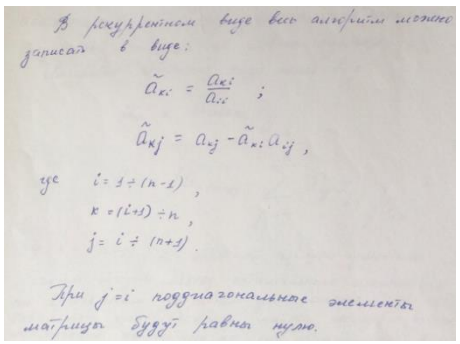
Постановка задачи: Ознакомиться с теорией вопроса (представлена в материалах папок: Теория_1_Классический_метод; Теория_2_Метод оптимального исключения; Теория_3_Гаусса-Жордана).

Математическая модель:

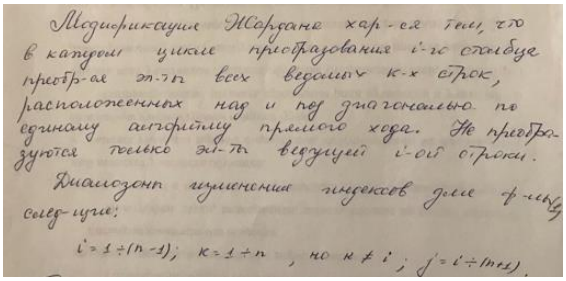
1. Классический метод:



2. Метод оптимального исключения:



3. Метод Гаусса-Жордана:



Код программы:

```
import numpy as np
from tabulate import tabulate


def gaussian_elimination(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])

    for i in range(num_rows):
        # Поиск максимального элемента в столбце
        max_row = i
        for j in range(i + 1, num_rows):
            if abs(matrix[j][i]) > abs(matrix[max_row][i]):
                max_row = j

        # Перестановка строк
        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        # Приведение к верхнетреугольному виду
        for j in range(i + 1, num_rows):
            ratio = matrix[j][i] / matrix[i][i]
            for k in range(i, num_cols):
                matrix[j][k] -= ratio * matrix[i][k]
```

```
    return matrix


def back_substitution(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])
    solution = np.zeros(num_rows)

    for i in range(num_rows - 1, -1, -1):
        solution[i] = matrix[i][num_cols - 1] / matrix[i][i]
        for j in range(i - 1, -1, -1):
            matrix[j][num_cols - 1] -= matrix[j][i] * solution[i]

    return solution


def gauss_jordan(matrix):
    num_rows = len(matrix)
    num_cols = len(matrix[0])

    for i in range(num_rows):
        # Поиск максимального элемента в столбце
        max_row = i
        for j in range(i + 1, num_rows):
            if abs(matrix[j][i]) > abs(matrix[max_row][i]):
                max_row = j

        # Перестановка строк
```

```
        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        # Приведение к единичной матрице
        pivot = matrix[i][i]
        for j in range(i, num_cols):
            matrix[i][j] /= pivot

        # Обновление остальных строк
        for j in range(num_rows):
            if j != i:
                ratio = matrix[j][i]
                for k in range(i, num_cols):
                    matrix[j][k] -= ratio * matrix[i][k]

    return matrix

def gauss_elimination(coeff_matrix, const_vector):
    n = len(coeff_matrix)
    # Создаем расширенную матрицу
    augmented_matrix = np.column_stack((coeff_matrix, const_vector))
    # Поиск максимального элемента в столбце
    for i in range(n):
        max_row = i
        for k in range(i + 1, n):
            if abs(augmented_matrix[k, i]) > abs(augmented_matrix[max_row, i]):
                max_row = k
        augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]
    # Прямой ход
        for j in range(i + 1, n):
```

```
            ratio = augmented_matrix[j,i] / augmented_matrix[i, i]
            augmented_matrix[j, i:] -= ratio * augmented_matrix[i, i:]
    # Обратный ход
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = augmented_matrix[i, -1] / augmented_matrix[i, i]
        for j in range(i - 1, -1, -1):
            augmented_matrix[j, -1] -= augmented_matrix[j, i] * x[i]
    return x

# Ввод исходных данных
matrix = []
n = int(input("Введите размерность матрицы: "))
print("Введите элементы матрицы:")
for i in range(n):
    row = list(map(float, input().split()))
    matrix.append(row)

print("Исходная матрица:")
print(tabulate(matrix, tablefmt="fancy_grid"))

# Прямой ход метода Гаусса
reduced_matrix = gaussian_elimination(matrix)
print("Матрица после прямого хода:")
print(tabulate(reduced_matrix, tablefmt="fancy_grid"))

# Обратный ход метода Гаусса
solution = back_substitution(reduced_matrix)
print("Решение СЛУ:")
```

```
print(solution)


# Реализация метода Гаусса-Жордана
gauss_jordan_matrix = gauss_jordan(matrix)
print("Результат метода Гаусса-Жордана:")
print(tabulate(gauss_jordan_matrix, tablefmt="fancy_grid"))


# Пример использования
A = np.array(matrix, dtype = float)
B = np.array([23,32,33,31])
x = gauss_elimination(A, B)
print("Решение", x)
```

Результаты программы для контрольного примера:

A				*B*
5	7	6	5	23
7	10	8	7	32
6	8	10	9	33
5	7	9	10	31

```
Введите размерность матрицы: 4
Введите элементы матрицы:
5 7 6 5 23
7 10 8 7 32
6 8 10 9 33
5 7 9 10 31
Исходная матрица:
```

5	7	6	5	23
7	10	8	7	32
6	8	10	9	33
5	7	9	10	31

Матрица после прямого хода:

7	10	8	7	32
0	-0.571429	3.14286	3	5.57143
0	0	2.5	4.25	6.75
0	0	0	0.1	0.1

Решение СЛУ:

[1. 1. 1. 1.]

Результат метода Гаусса-Жордана:

1	0	0	0	-2.62857
0	1	0	0	2.4
0	0	1	0	-0.7
0	0	0	1	1

```
Решение [1. 1. 1. 1.]
```

и для значений элементов столбца свободных членов на

1) 0,1

```
Введите элементы матрицы:
5 7 6 5 23.1
7 10 8 7 32.1
6 8 10 9 33.1
5 7 9 10 31.1
Исходная матрица:
```

5	7	6	5	23.1
7	10	8	7	32.1
6	8	10	9	33.1
5	7	9	10	31.1

Матрица после прямого хода:

7	10	8	7	32.1
0	-0.571429	3.14286	3	5.58571
0	0	2.5	4.25	6.775
0	0	0	0.1	0.13

```
Решение СЛУ:
[ 3.  -0.2  0.5  1.3]
Результат метода Гаусса-Жордана:
```

1	0	0	0	7.62571
0	1	0	0	-2.78
0	0	1	0	-1.71
0	0	0	1	1.3

```
Решение [ 3.  -0.2  0.5  1.3]
```

2) 0,001

```
Введите размерность матрицы: 4
Введите элементы матрицы:
5 7 6 5 23.001
7 10 8 7 32.001
6 8 10 9 33.001
5 7 9 10 31.001
Исходная матрица:
```

5	7	6	5	23.001
7	10	8	7	32.001
6	8	10	9	33.001
5	7	9	10	31.001

Матрица после прямого хода:

7	10	8	7	32.001
0	-0.571429	3.14286	3	5.57157
0	0	2.5	4.25	6.75025
0	0	0	0.1	0.1003

```
Решение СЛУ:
[1.02  0.988 0.995 1.003]
Результат метода Гаусса-Жордана:
```

1	0	0	0	-2.52603
0	1	0	0	2.3482
0	0	1	0	-0.7101
0	0	0	1	1.003

3) 0,01

```
Введите размерность матрицы: 4
Введите элементы матрицы:
5 7 6 5 23.01
7 10 8 7 32.01
6 8 10 9 33.01
5 7 9 10 31.01
Исходная матрица:
```

5	7	6	5	23.01
7	10	8	7	32.01
6	8	10	9	33.01
5	7	9	10	31.01

Матрица после прямого хода:

7	10	8	7	32.01
0	-0.571429	3.14286	3	5.57286
0	0	2.5	4.25	6.7525
0	0	0	0.1	0.103

Решение СЛУ:

[1.2 0.88 0.95 1.03]

Результат метода Гаусса-Жордана:

1	0	0	0	-1.60314
0	1	0	0	1.882
0	0	1	0	-0.801
0	0	0	1	1.03

```
Решение [1.2  0.88 0.95 1.03]
```

Вывод:

Прямой ход метода Гаусса хорошо справляется с системой во всех трех случаях. Обратный ход метода Гаусса показывает, что при элементах 23 и 0,1 система становится не доопределённой что неудивительно, так как в этих случаях есть свободные переменные. Метод Гаусса-Жордана дает матрицу с единичными значениями на диагонали во всех случаях, но значения свободных членов могут быть с погрешностью из-за использования чисел с плавающей точкой