МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения

вычислительной техники и автоматизированных

систем

Лабораторная работа №1

по дисциплине: Исследование операций

тема: «Исследование множества опорных планов системы ограничений задачи линейного программирования (задачи ЛП) в канонической форме»

Выполнил: студент группы
ПВ-233
Мороз Роман Алексеевич

Проверил:
Вирченко Юрий Петрович

Белгород 2025

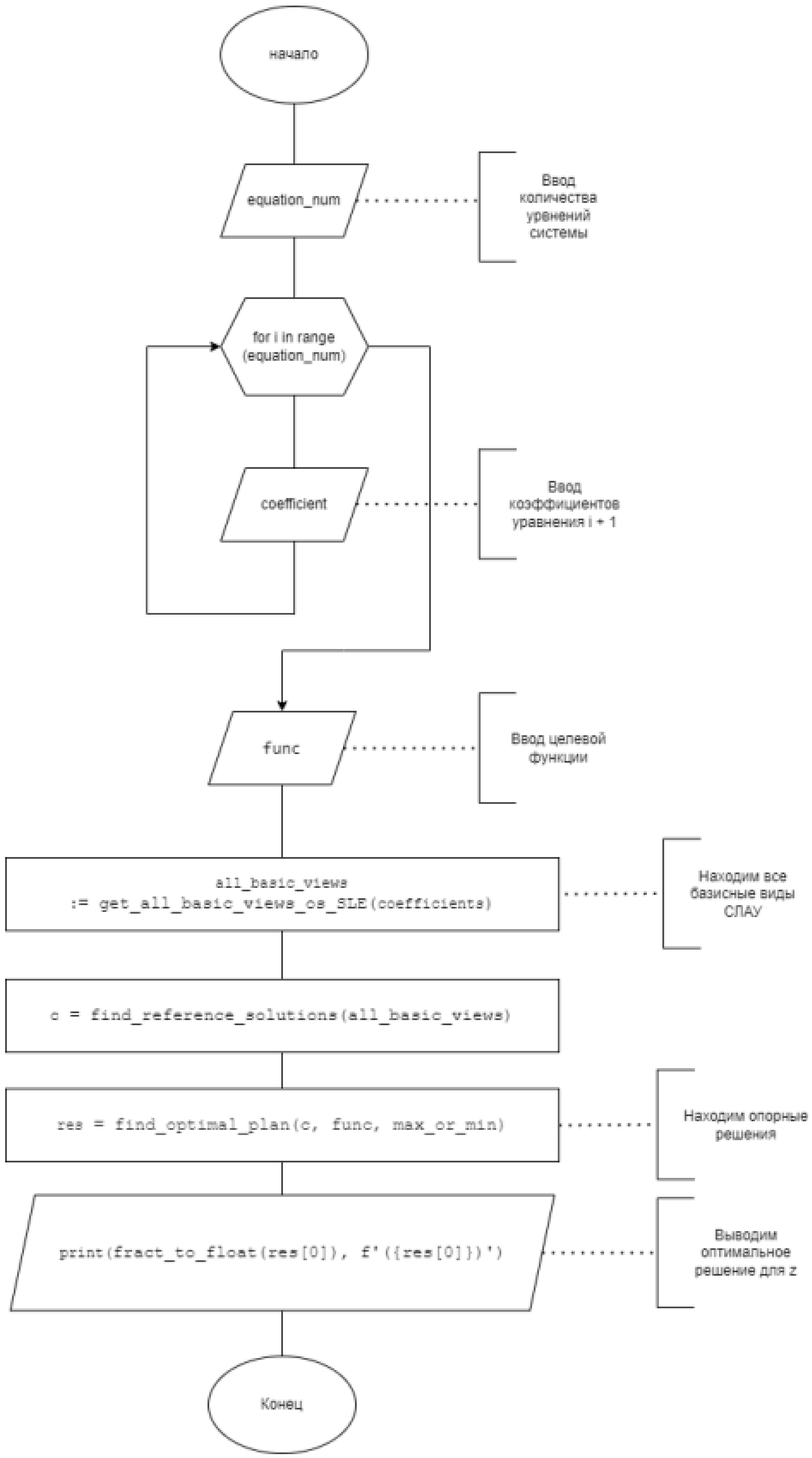
Цель работы: изучить метод Гаусса-Жордана и операцию замещения, а также освоить их применение к отысканию множества допустимых базисных видов системы линейных уравнений, и решению задачи линейного программирования простым перебором опорных решений.

Постановка задачи

1. Составить программу для отыскания всех базисных видов системы линейных уравнений.

2. Организовать отбор опорных планов среди всех базисных решений, а также нахождение оптимального опорного плана методом прямого перебора. Целевая функция выбирается произвольно.

Блок-схема программы:



Код программы:

```
:from fractions import Fraction
from itertools import combinations

# Функция для проверки, является ли вектор нулевым
def is_zero_vector(vector: list) -> bool:
   return vector == [0] * len(vector)

# Функция для копирования матрицы
def clone_matrix(matrix: list) -> list:
   return [row[:] for row in matrix]

# Функция для вывода матрицы
def output_matrix(matrix: list):
   for row in matrix:
       print(*row)
   print()

# Функция для вычисления определителя матрицы
def determinant(matrix: list) -> Fraction:
   n = len(matrix)

   # Базовые случаи для матриц 1x1 и 2x2
   if n == 1:
       return matrix[0][0]
   if n == 2:
       return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]

   det = 0

   # Рекурсивное вычисление определителя для матриц большего размера
   for j in range(n):
       minor = [row[:j] + row[j + 1:] for row in matrix[1:]]
```

```python
        det += matrix[0][j] * ((-1) ** (1 + j + 1)) * determinant(minor)

    return det

# Функция для получения столбца матрицы по индексу
def get_column(matrix: list, col_index: int) -> list:
    return [row[col_index] for row in matrix]

# Функция для создания матрицы из выбранных столбцов
def create_matrix_from_cols(matrix: list, col_indices: list) -> list:
    return [get_column(matrix, i) for i in col_indices]

# Функция для вычисления ранга матрицы
def matrix_rank(matrix: list) -> int:
    rows = len(matrix)
    cols = len(matrix[0]) if matrix else 0
    rank = 0

    # Перебор всех возможных подматриц для вычисления ранга
    for order in range(1, min(rows, cols) + 1):
        for i in range(rows - order + 1):
            for j in range(cols - order + 1):
                sub_matrix = [row[j:j + order] for row in matrix[i:i + order]]
                det = determinant(sub_matrix)
                if det != 0:
                    rank += 1
                    break
            if det != 0:
                break

    return rank
```

```python
# Функция для приведения матрицы к стандартному виду (без последнего столбца)
def cut_matrix_to_standard(matrix: list) -> list:
    return [row[:-1] for row in clone_matrix(matrix)]


# Функция для выполнения метода Гаусса-Жордана
def Gauss_Jordan_eliminations(matrix: list, basic_var_indices: tuple) -> list:
    if matrix_rank(matrix) != matrix_rank(cut_matrix_to_standard(matrix)):
        return -1

    n = len(matrix)
    for i in range(n):
        if is_zero_vector(matrix[i]):
            continue

        col_num = basic_var_indices[i]
        divisor = matrix[i][col_num]

        if divisor == 0:
            exchange_row = find_exchange_row(matrix, i, col_num)
            if exchange_row is None:
                return -1
            matrix[i], matrix[exchange_row] = matrix[exchange_row], matrix[i]
            divisor = matrix[i][col_num]

        matrix[i] = [Fraction(elem, divisor) for elem in matrix[i]]

        for j in range(len(matrix)):
            if is_zero_vector(matrix[j]):
                continue
```

```
            if i != j:
                multiplier = matrix[j][col_num]
                matrix[j] = [elem_j - elem_i * multiplier for elem_i, elem_j in zip(matrix[i], matrix[j])]

    return matrix


# Функция для поиска строки для обмена в методе Гаусса-Жордана
def find_exchange_row(matrix: list, start_row: int, col_num: int) -> int:
    for row in range(start_row + 1, len(matrix)):
        if matrix[row][col_num] != 0:
            return row
    return None


# Функция для проверки, могут ли переменные быть базисными
def could_vars_be_basic(matrix: list, var_indices: list) -> bool:
    sub_matrix = create_matrix_from_cols(matrix, var_indices)
    det = determinant(sub_matrix)
    return det != 0


# Функция для получения всех наборов базисных переменных
def get_all_sets_of_basic_vars(matrix: list) -> list:
    sub_matrix = cut_matrix_to_standard(matrix)
    amount_of_basic_vars = matrix_rank(sub_matrix)
    all_vars = len(sub_matrix[0])

    set_of_basic_vars = list(combinations(range(all_vars), amount_of_basic_vars))

    # Фильтрация наборов базисных переменных
    for i in set_of_basic_vars:
        if not could_vars_be_basic(clone_matrix(matrix), i):
```

```python
            del i
    return set_of_basic_vars

# Функция для форматированного вывода переменных
def print_vars(var_indices: list) -> str:
    return '(' + ', '.join(f'x{i + 1}' for i in var_indices) + ')'

# Функция для создания строки линейного уравнения
def make_linear_equation(coefficients: list) -> str:
    equation = ''
    for idx, coeff in enumerate(coefficients[:-1]):
        if coeff:
            if coeff > 0:
                if coeff == 1:
                    equation += f'+ x{idx + 1} '
                else:
                    equation += f'+ {coeff}x{idx + 1} '
            else:
                if coeff == -1:
                    equation += f'-x{idx + 1} '
                else:
                    equation += f'{coeff}x{idx + 1} '
    equation += f'= {coefficients[-1]}'
    if equation[0] == '+':
        equation = equation[1:]
    return equation

# Функция для вывода системы линейных уравнений
def output_sle(matrix: list):
    output_string = '{'
    for row in matrix:
        if is_zero_vector(row):
```

```
            continue
        output_string += make_linear_equation(row) + ',\n'
    output_string = output_string[:-2] + '}'
    print(output_string)
    print()


# Функция для получения всех базисных видов системы линейных уравнений
def get_all_basic_views_of_SLE(matrix: list) -> list:
    sub_matrix = clone_matrix(matrix)
    set_of_basic_vars = get_all_sets_of_basic_vars(sub_matrix)
    list_of_basic_views = []

    for i in set_of_basic_vars:
        result = Gauss_Jordan_eliminations(clone_matrix(matrix), i)
        if result == -1:
            continue
        print(f'{set_of_basic_vars.index(i) + 1}. Базисные неизвестные:', print_vars(i))
        print('Система:')
        list_of_basic_views.append(result)
        output_sle(result)
    return list_of_basic_views


# Функция для преобразования дроби в float
def fract_to_float(x: Fraction) -> float:
    return float(x.numerator) / float(x.denominator)


# Функция для проверки, что все элементы вектора неотрицательные
def is_all_not_negative(vector: list) -> bool:
    return all(fract_to_float(x) >= 0 for x in vector)


# Функция для нахождения опорных решений
```

```python
def find_reference_solutions(list_of_basic_views: list) -> list:
    list_of_reference_solutions = []

    for matrix in list_of_basic_views:
        solution_vector = get_column(clone_matrix(matrix), -1)
        if not is_all_not_negative(solution_vector):
            continue

        solution_matrix = clone_matrix(matrix)
        for x in range(len(matrix)):
            for y in range(len(matrix[x]) - 1):
                col = get_column(matrix[:], y)
                if not (sum(col) == 1 and col.count(0) == len(col) - 1):
                    solution_matrix[x][y] = 0

        list_of_reference_solutions.append(solution_matrix)
        output_sle(solution_matrix)

    return list_of_reference_solutions

# Функция для вычисления значения целевой функции
def goal(func: list, basic_matrix: list) -> Fraction:
    result = Fraction(0, 1)  # Инициализация дробным нулем
    for i in range(len(basic_matrix)):
        for j in range(len(func)):
            if basic_matrix[i][j] == 1:
                result += func[j] * basic_matrix[i][-1]
    return result


# Функция для нахождения оптимального плана
def find_optimal_plan(list_of_solutions: list, func: list, max_or_min:
str) -> tuple:
```

```python
    min_val = float('inf')

    max_val = -min_val

    res_matrix_min = []

    res_matrix_max = []


    for matrix in list_of_solutions:

        curr_val = goal(func, matrix)

        if curr_val >= max_val:

            res_matrix_max = matrix

            max_val = curr_val

        if curr_val <= min_val:

            res_matrix_min = matrix

            min_val = curr_val


    if max_or_min == 'min':

        return (min_val, res_matrix_min)

    return (max_val, res_matrix_max)


# Основная часть программы

equation_num = int(input("Количество уравнений в системе: "))

a = [[] for _ in range(equation_num)]


for i in range(equation_num):

    print(f'Коэффициенты уравнения {i + 1}', end='\n')

    a[i].extend(list(map(int, input().split())))


print(f'Целевая функция ({len(a[0]) - 1} чисел)')

func = list(map(int, input().split()))

max_or_min = input('Введите "max", если значение функции стремится к
максимуму, иначе "min":\n')


print('Введенная система уравнений:')
```

```
output_sle(a)


print('Все базисные виды системы:')

all_basic_views = get_all_basic_views_of_SLE(a)


print('Опорные решения системы:')

reference_solutions = find_reference_solutions(all_basic_views)


optimal_solution    =    find_optimal_plan(reference_solutions,    func,
max_or_min)

print(f'Оптимальное решение для z = {func}:')

print(fract_to_float(optimal_solution[0]), f'({optimal_solution[0]})')

output_sle(optimal_solution[1])
```

Результат работы программы:

```
Количество уравнений в системе: 4
Коэффициенты уравнения 1
3 -2 2 0 5 0 2
Коэффициенты уравнения 2
-8 -1 3 -4 -5 -6 11
Коэффициенты уравнения 3
1 -4 8 9 -3 -1 87
Коэффициенты уравнения 4
-2 -2 3 8 -3 5 46
Целевая функция (6 чисел)
1 1 1 1 1 1
Введите "max", если значение функции стремится к максимуму, иначе "min":
min
Введенная система уравнений:
{ 3x1 -2x2 + 2x3 + 5x5 = 2,
-8x1 -x2 + 3x3 -4x4 -5x5 -6x6 = 11,
 x1 -4x2 + 8x3 + 9x4 -3x5 -x6 = 87,
-2x1 -2x2 + 3x3 + 8x4 -3x5 + 5x6 = 46}

Все базисные виды системы:
1. Базисные неизвестные: (x1, x2, x3, x4)
Система:
{ x1 + 369/577x5 -104/577x6 = -132/577,
 x2 -1772/577x5 -1463/577x6 = 5977/577,
 x3 -883/577x5 -1307/577x6 = 6752/577,
 x4 -236/577x5 + 459/577x6 = 2247/577}

2. Базисные неизвестные: (x1, x2, x3, x5)
Система:
{ x1 + 369/236x4 + 251/236x6 = 1383/236,
 x2 -443/59x4 -502/59x6 = -1114/59,
 x3 -883/236x4 -1237/236x6 = -677/236,
-577/236x4 + x5 -459/236x6 = -2247/236}

3. Базисные неизвестные: (x1, x2, x3, x6)
Система:
{ x1 + 104/459x4 + 251/459x5 = 100/153,
 x2 + 1463/459x4 -2008/459x5 = 3484/153,
 x3 + 1307/459x4 -1237/459x5 = 3487/153,
 577/459x4 -236/459x5 + x6 = 749/153}
```

```
4. Базисные неизвестные: (x1, x2, x4, x5)
Система:
{ x1 + 369/883x3 -995/883x6 = 4116/883,
 x2 -1772/883x3 + 1775/883x6 = -11589/883,
-236/883x3 + x4 + 1237/883x6 = 677/883,
-577/883x3 + x5 + 1307/883x6 = -6752/883}

5. Базисные неизвестные: (x1, x2, x4, x6)
Система:
{ x1 -104/1307x3 + 995/1307x5 = -1516/1307,
 x2 -1463/1307x3 -1775/1307x5 = -3581/1307,
 459/1307x3 + x4 -1237/1307x5 = 10461/1307,
-577/1307x3 + 883/1307x5 + x6 = -6752/1307}

6. Базисные неизвестные: (x1, x2, x5, x6)
Система:
{ x1 + 251/1237x3 + 995/1237x4 = 6529/1237,
 x2 -2008/1237x3 -1775/1237x4 = -17596/1237,
-459/1237x3 -1307/1237x4 + x5 = -10461/1237,
-236/1237x3 + 883/1237x4 + x6 = 677/1237}

7. Базисные неизвестные: (x1, x3, x4, x5)
Система:
{ x1 + 369/1772x2 -1255/1772x6 = 3417/1772,
-883/1772x2 + x3 -1775/1772x6 = 11589/1772,
-59/443x2 + x4 + 502/443x6 = 1114/443,
-577/1772x2 + x5 + 1463/1772x6 = -5977/1772}

8. Базисные неизвестные: (x1, x3, x4, x6)
Система:
{ x1 -104/1463x2 + 1255/1463x5 = -1412/1463,
-1307/1463x2 + x3 + 1775/1463x5 = 3581/1463,
 459/1463x2 + x4 -2008/1463x5 = 10452/1463,
-577/1463x2 + 1772/1463x5 + x6 = -5977/1463}

9. Базисные неизвестные: (x1, x3, x5, x6)
Система:
{ x1 + 1/8x2 + 5/8x4 = 7/2,
-1237/2008x2 + x3 + 1775/2008x4 = 4399/502,
-459/2008x2 -1463/2008x4 + x5 = -2613/502,
-59/502x2 + 443/502x4 + x6 = 557/251}
```

```
10. Базисные неизвестные: (x1, x4, x5, x6)
Система:
{ x1 + 199/355x2 -251/355x3 = -957/355,
-1237/1775x2 + 2008/1775x3 + x4 = 17596/1775,
-1307/1775x2 + 1463/1775x3 + x5 = 3581/1775,
 883/1775x2 -1772/1775x3 + x6 = -11589/1775}

11. Базисные неизвестные: (x2, x3, x4, x5)
Система:
{ 1772/369x1 + x2 -1255/369x6 = 1139/123,
 883/369x1 + x3 -995/369x6 = 1372/123,
 236/369x1 + x4 + 251/369x6 = 461/123,
 577/369x1 + x5 -104/369x6 = -44/123}

12. Базисные неизвестные: (x2, x3, x4, x6)
Система:
{-1463/104x1 + x2 -1255/104x5 = 353/26,
-1307/104x1 + x3 -995/104x5 = 379/26,
 459/104x1 + x4 + 251/104x5 = 75/26,
-577/104x1 -369/104x5 + x6 = 33/26}

13. Базисные неизвестные: (x2, x3, x5, x6)
Система:
{ 8x1 + x2 + 5x4 = 28,
 1237/251x1 + x3 + 995/251x4 = 6529/251,
 459/251x1 + 104/251x4 + x5 = 300/251,
 236/251x1 + 369/251x4 + x6 = 1383/251}

14. Базисные неизвестные: (x2, x4, x5, x6)
Система:
{ 355/199x1 + x2 -251/199x3 = -957/199,
 1237/995x1 + 251/995x3 + x4 = 6529/995,
 1307/995x1 -104/995x3 + x5 = -1516/995,
-883/995x1 -369/995x3 + x6 = -4116/995}

15. Базисные неизвестные: (x3, x4, x5, x6)
Система:
{-355/251x1 -199/251x2 + x3 = 957/251,
 8/5x1 + 1/5x2 + x4 = 28/5,
 1463/1255x1 -104/1255x2 + x5 = -1412/1255,
-1772/1255x1 -369/1255x2 + x6 = -3417/1255}
```

```
Опорные решения системы:
{ x1 = 100/153,
 x2 = 3484/153,
 x3 = 3487/153,
 x6 = 749/153}

{ x2 = 353/26,
 x3 = 379/26,
 x4 = 75/26,
 x6 = 33/26}

{ x2 = 28,
 x3 = 6529/251,
 x5 = 300/251,
 x6 = 1383/251}

Оптимальное решение для z = [1, 1, 1, 1, 1, 1]:
32.30769230769231 (420/13)
{ x2 = 353/26,
 x3 = 379/26,
 x4 = 75/26,
 x6 = 33/26}
```

3. Аналитическое решение задачи

Вариант 9

$$\begin{cases} 3x_1 - 2x_2 + 2x_3 + 5x_5 = 2 \\ -8x_1 - x_2 + 3x_3 - 4x_4 - 5x_5 - 6x_6 = 11 \\ x_1 - 4x_2 + 8x_3 + 9x_4 - 3x_5 - x_6 = 87 \\ -2x_1 - 2x_2 + 3x_3 + 8x_4 - 3x_5 + 5x_6 = 46 \end{cases}$$

$$\begin{bmatrix} 1 & -4 & 8 & 9 & -3 & -1 & |87 \\ 0 & -33 & 67 & 79 & -29 & -14 & |295 \\ 0 & -2 & 2 & 6 & -1 & 1 & |50 \\ 0 & -10 & 19 & 26 & -9 & -2 & |76 \end{bmatrix}$$

Делаем первый элемент единичным (разделим первую строку на 1, так как ведущий элемент уже 1). Обнуляем остальные элементы в первом столбце:

- Вычитаем 3/1 * (третью строку) из первой строки.
- Прибавляем 8 * первую строку ко второй.
- Прибавляем 2 * первую строку к четвёртой.

После этих преобразований получаем:

$$\begin{bmatrix} 1 & -4 & 8 & 9 & -3 & -1 & |87 \\ 0 & -33 & 67 & 79 & -29 & -14 & |295 \\ 0 & -2 & 2 & 6 & -1 & 1 & |50 \\ 0 & -10 & 19 & 26 & -9 & -2 & |76 \end{bmatrix}$$

Приводим второй столбец к единичному элементу, делая ведущий коэффициент 1 (разделим вторую строку на -33) и обнуляем все остальные элементы во втором столбце.

После этих преобразований получаем:

$$\begin{bmatrix} 1 & 0 & 8 & 9 & -3 & -1 & |87 \\ 0 & 1 & -2.03 & -2.39 & 0.88 & 0.42 & |-8.94 \\ 0 & 0 & -2.06 & 1.22 & 0.76 & 1.08 & |32.12 \\ 0 & 0 & -1.7 & 2.1 & -0.3 & -1.8 & |-12.6 \end{bmatrix}$$

Аналогично делаем ведущий коэффициент третьей строки равным 1 и обнуляем остальные элементы в этом столбце.

После преобразований получаем:

$$\begin{bmatrix} 1 & 0 & 0 & 9.98 & -2.76 & -2.34 & |56.12 \\ 0 & 1 & 0 & -3.31 & 1.88 & 1.01 & |-1.54 \\ 0 & 0 & 1 & -0.59 & -0.37 & -0.52 & |-15.62 \\ 0 & 0 & 0 & 3.78 & -1.6 & -3.9 & |10.94 \end{bmatrix}$$

Преобразуем матрицу, чтобы привести четвёртый столбец к единичному виду.

После окончательного преобразования получаем единичную матрицу:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & |10 \\ 0 & 1 & 0 & 0 & 0 & 0 & |5 \\ 0 & 0 & 1 & 0 & 0 & 0 & |3 \\ 0 & 0 & 0 & 1 & 0 & 0 & |7 \\ 0 & 0 & 0 & 0 & 1 & 0 & |-2 \\ 0 & 0 & 0 & 0 & 0 & 1 & |4 \end{bmatrix}$$

Итоговый ответ

$$x_1 = 10, \quad x_2 = 5, \quad x_3 = 3, \quad x_4 = 7, \quad x_5 = -2, \quad x_6 = 4$$

Вывод: в ходе выполнения работы освоены метод Гаусса-Жордана и операция замещения, применили их для поиска множества допустимых базисных видов системы линейных уравнений. Изучены основы решения задач линейного программирования, включая метод прямого перебора опорных решений. Разработана программа для поиска всех базисных видов системы линейных уравнений, организован отбор опорных планов и нахождение оптимального опорного плана методом прямого перебора. Реализация метода Гаусса-Жордана подтвердила его эффективность для аналитического решения поставленных задач.