Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Факультет информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

**Курсовая работа по дисциплине  
 Алгоритмы и вычислительные методы оптимизации**

**Метод искусственного базиса**

Вариант 8

Выполнил:

студент гр.ИП-814 Краснов И.В.

«14» апреля 2021 г. Подпись студента\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

ассистент кафедры ПМиК Новожилов Д. И.

«14» апреля 2021 г. Подпись преподавателя\_\_\_\_\_\_\_\_\_\_

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2021 г.

Содержание

[Задание 3](#_Toc72781401)

[Переход к канонической форме записи 4](#_Toc72781402)

[Решение задачи линейного программирования методом искусственного базиса 5](#_Toc72781403)

[Графическое решение исходной задачи 7](#_Toc72781404)

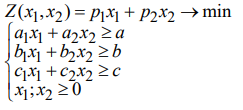
[Решение двойственной задачи 8](#_Toc72781405)

[Код программы 10](#_Toc72781406)

[Примеры решения задач 23](#_Toc72781407)

# Задание

1. Перейти к канонической форме записи задачи линейного программирования.

****

2. Написать программу, решающую задачу линейного программирования в канонической форме (с выводом всех промежуточных таблиц) методом искусственного базиса.

3. Решить исходную задачу графически и отметить на чертеже точки, соответствующие симплексным таблицам, полученным при выполнении программы из п.2.

4. Составить двойственную задачу к исходной и найти ее решение на основании теоремы равновесия.

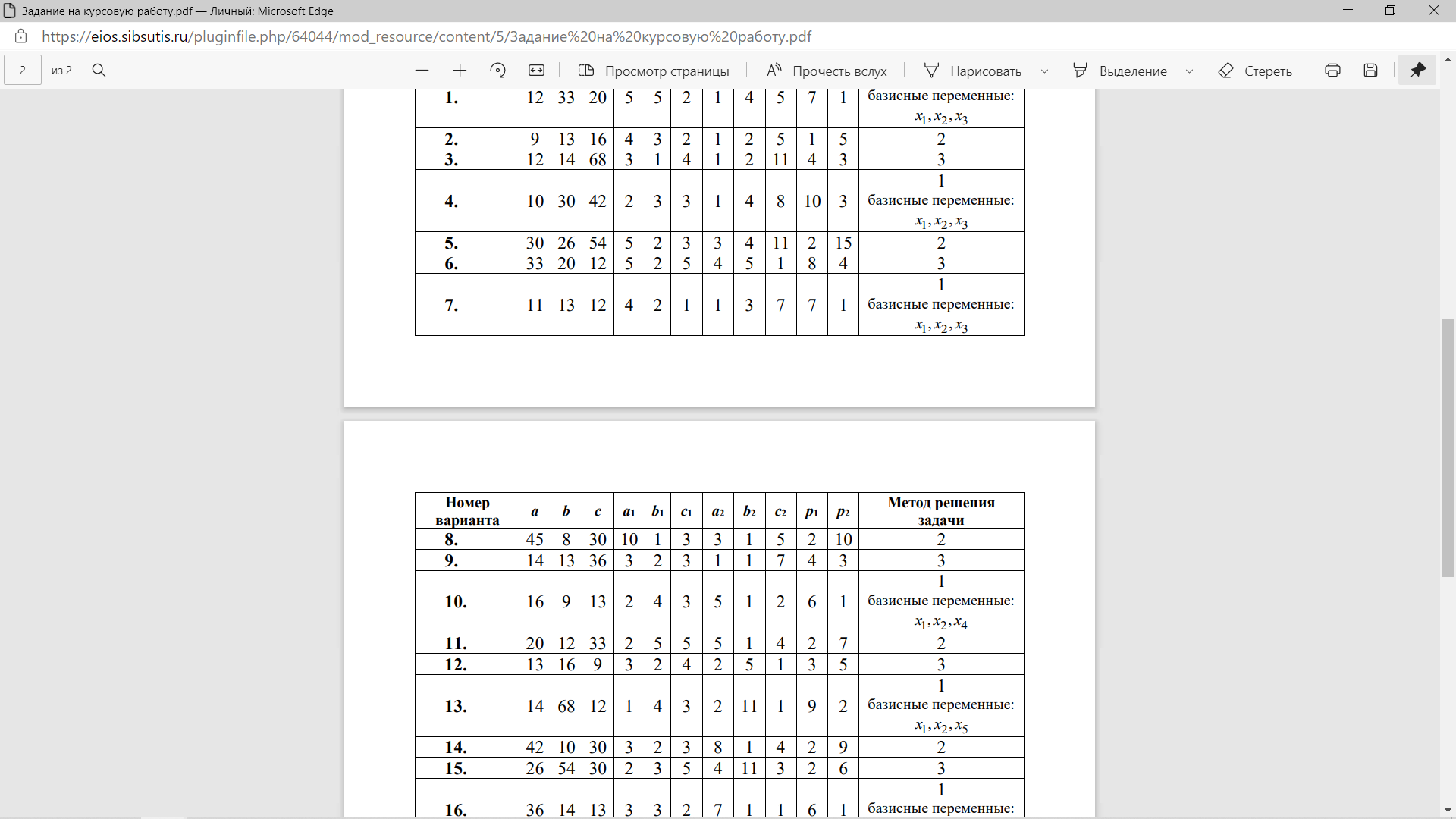


Рис. 1 – Исходные данные для варианта №8.

# Переход к канонической форме записи

Для того, чтобы перейти к канонической форме записи исходной ЗЛП (задачи линейного программирования), необходимо добавить балансовые переменные соответствующего неравенству знака, а также сменить условие минимизации целевой функции на условие максимизации. Так как все знаки в неравенствах – это «≥», то необходимо в каждое неравенство добавить по одной уникальной балансовой переменной со знаком минус, тогда уравнение можно будет записать со знаком «=».

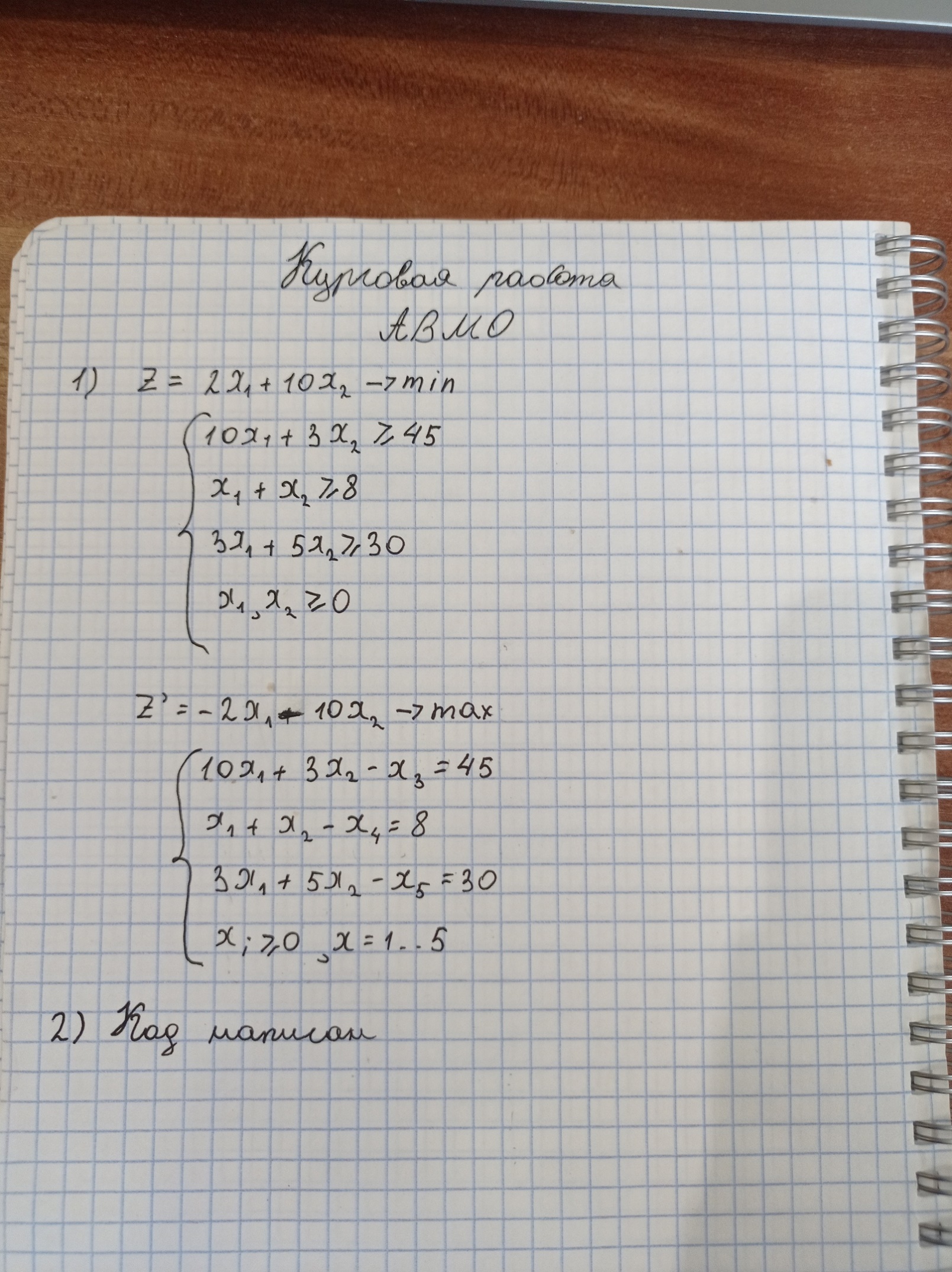


Рис. 2 – Каноническая форма записи исходной ЗЛП.

# Решение задачи линейного программирования методом искусственного базиса

Далее будет описан алгоритм работы программы. Сначала пользователь вводит входные данные, т.е. саму ЗЛП в канонической форме записи.

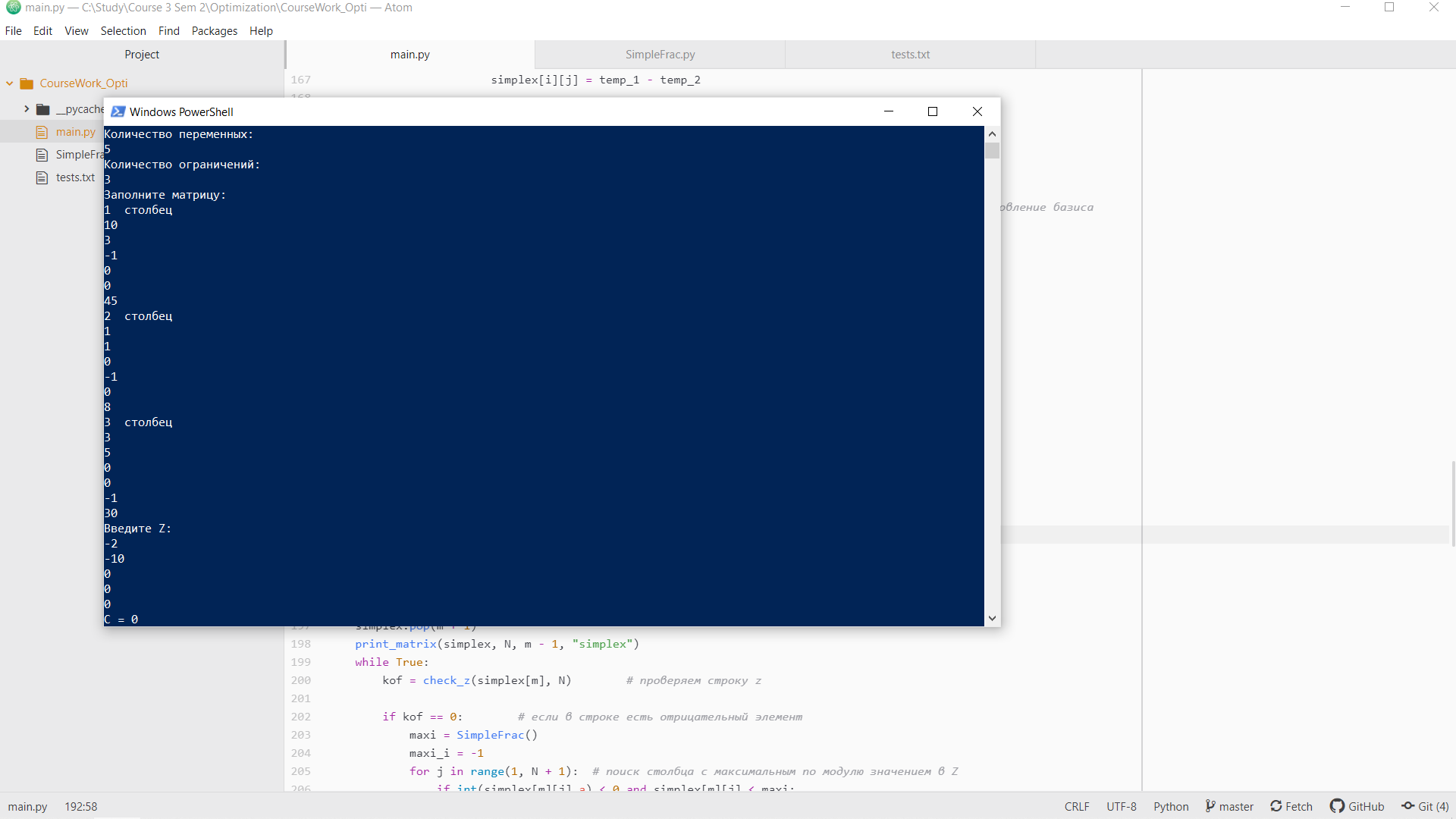


Рис. 3 – Входные данные.

После того, как данные введены, вызывается функция artificial\_basis(), в котором выполняется сам метод искусственного базиса. Сначала происходит добавление нового базиса (в каждую строку добавляется новая переменная). После этого добавляются последовательно строки Z и M. Таким образом сформировалась симплекс-таблица, над которой будут производится дальнейшие вычисления.

После того, как сформировалась симплекс-таблица, начинается первый цикл, обрабатывающий строку M. В этом цикле из таблицы последовательно выводятся базисные переменные. В функции *check\_m()* проверяется строка M, если в переменных есть отрицательное значение, цикл продолжается, если вся строка обнулилась, цикл завершается, иначе программа завершается со словами «Система не совместна». После завершения цикла M-строка удаляется.

Затем следует цикл симплекс-метода, выход из цикла происходит, когда все отрицательные значения в строке Z перестали быть отрицательными (проверка происходит в функции check\_z()).

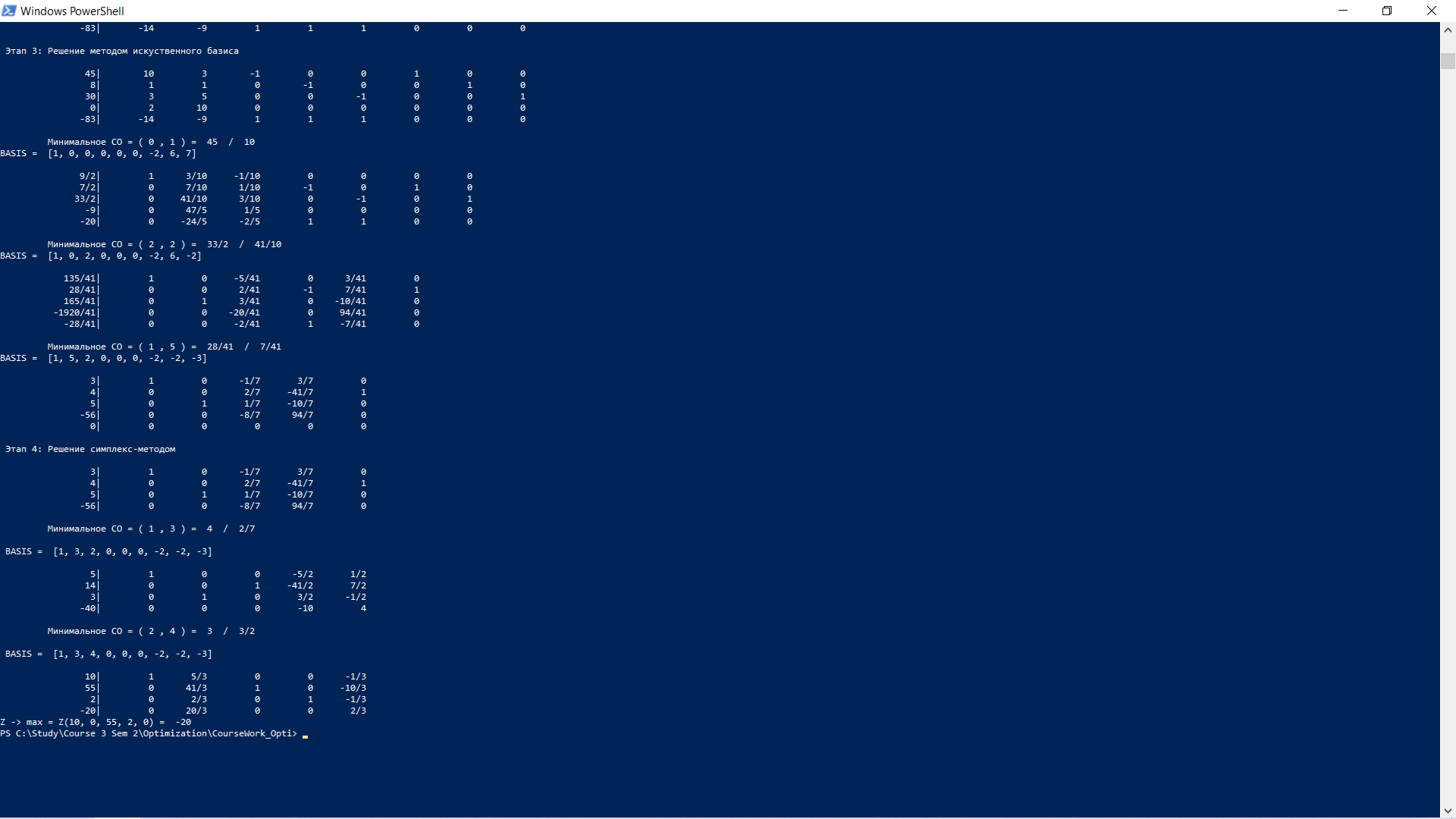
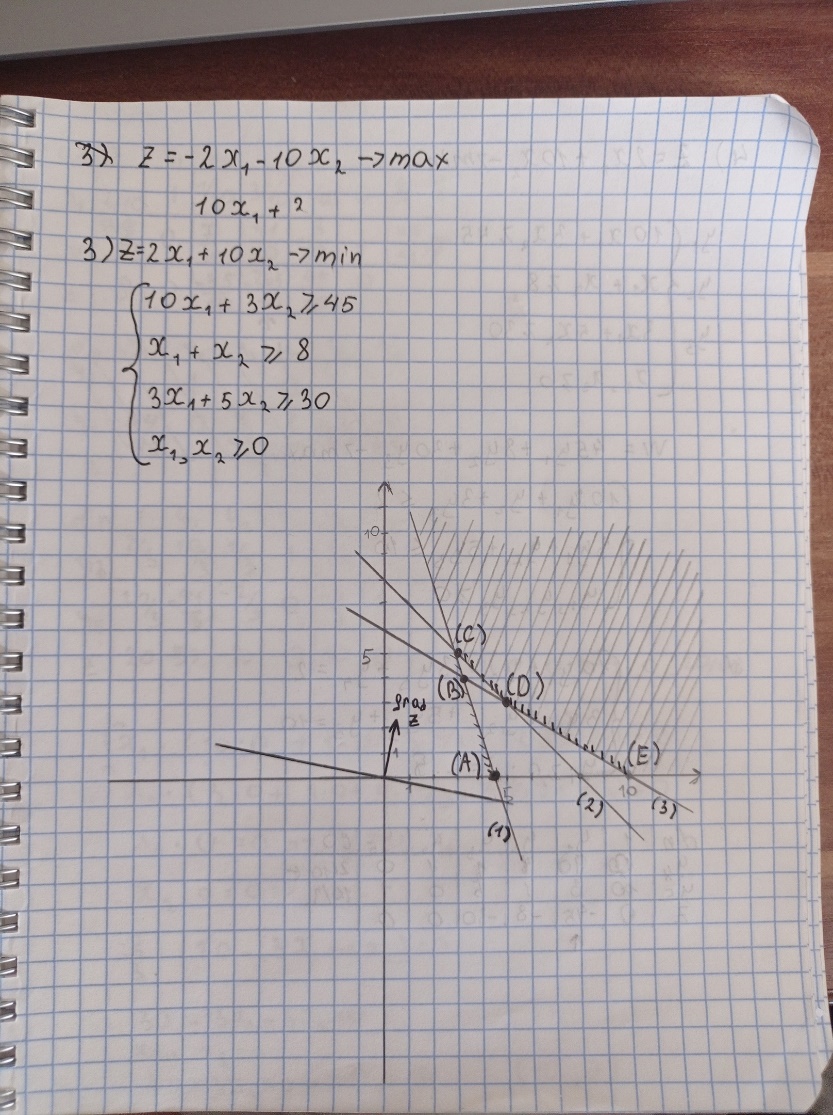
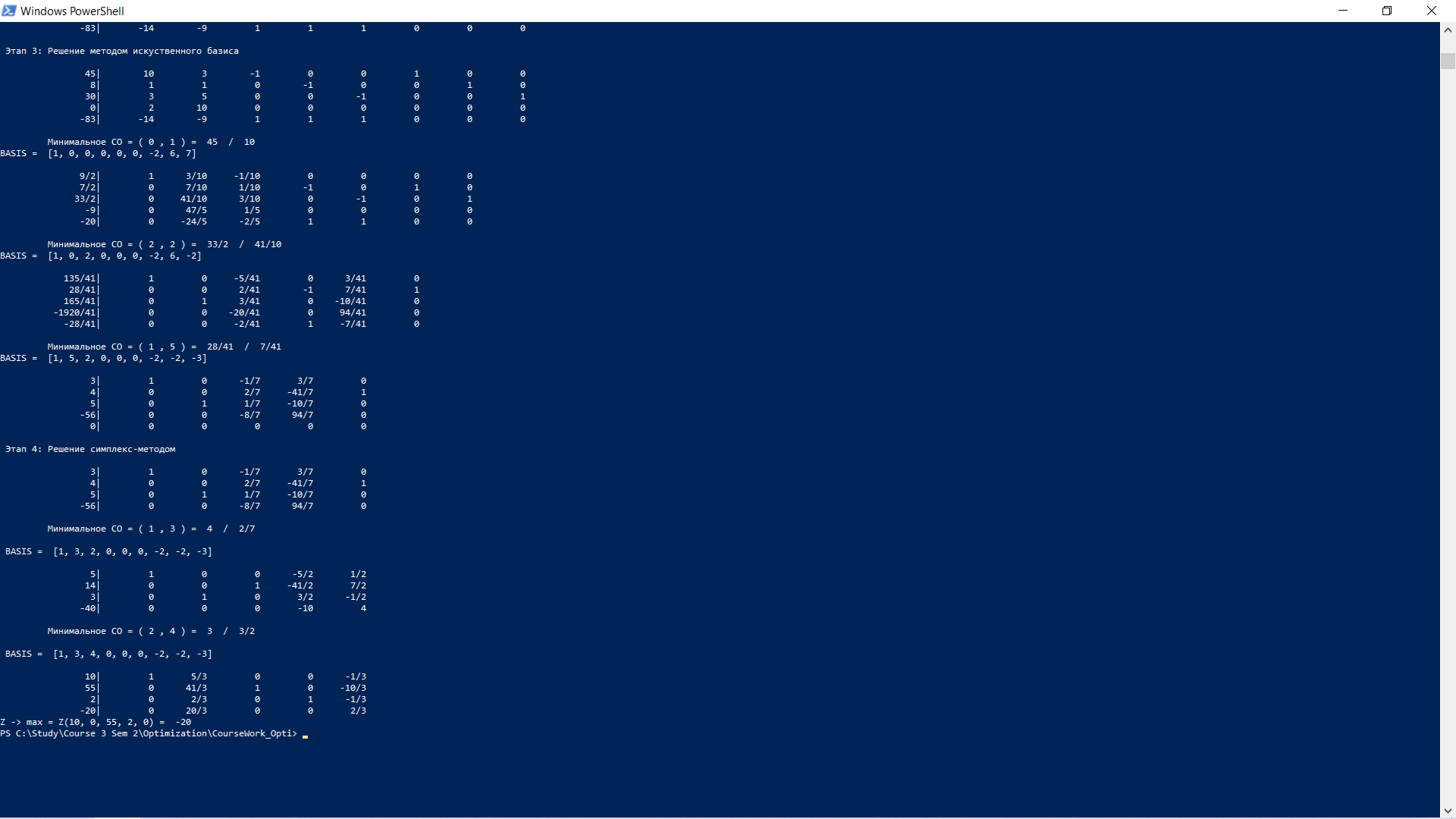


Рис. 4 – Промежуточные таблицы работы алгоритма

# Графическое решение исходной задачи

Для графического решения исходной ЗЛП мы построили линии, соответствующие уравнениям из исходной системы ограничений, определили, где находится область, которую мы ищем, и построили градиент, по которому, с помощью линий уровня, нашли точку минимума нашей функции. Это точка с координатами (8;0).

E

D

C

B

A

Рис. 5 – Графическое решение исходной ЗЛП.

# Решение двойственной задачи

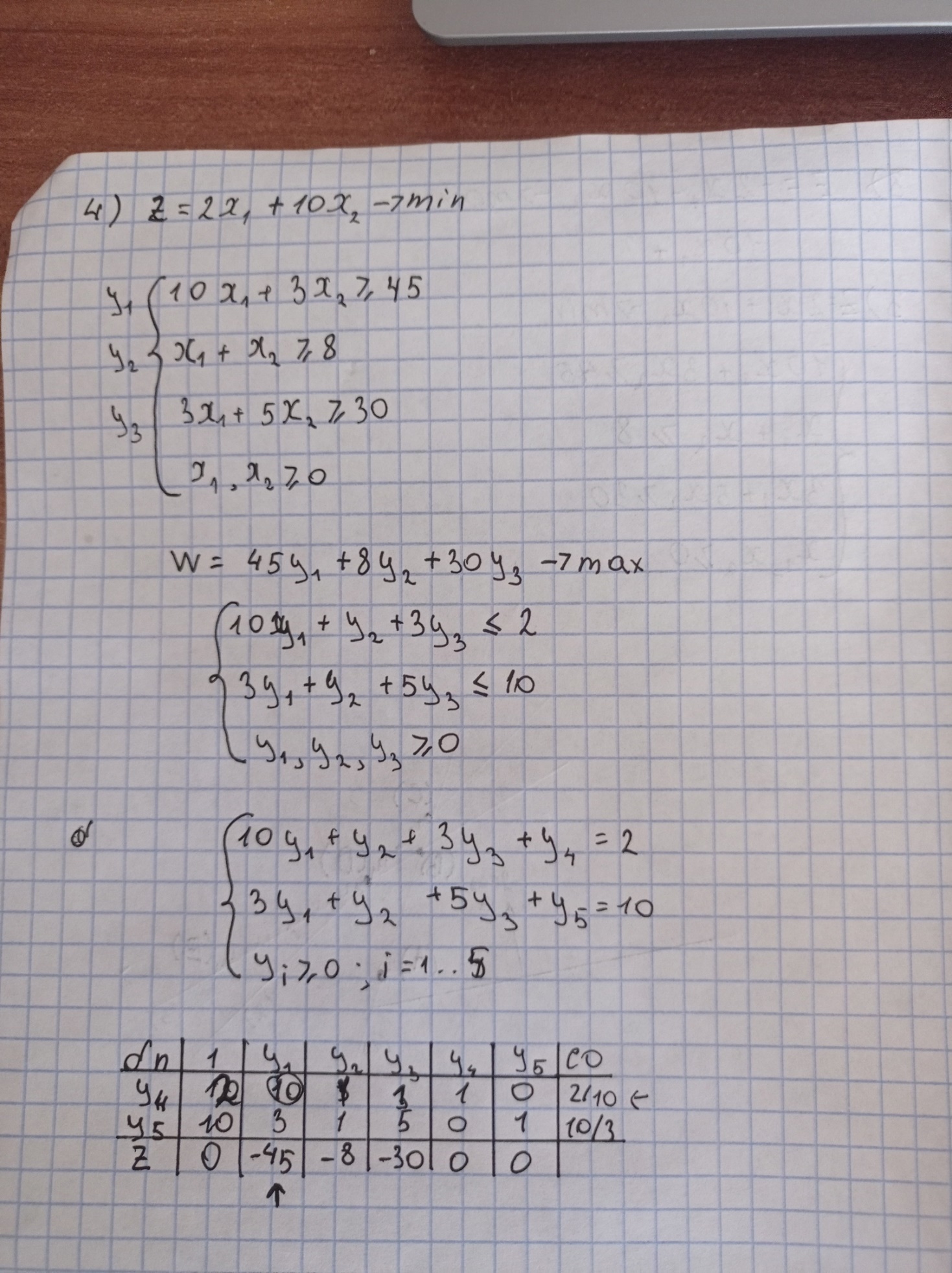


Рис. 6 – Составление двойственной задачи и симплексной таблицы.

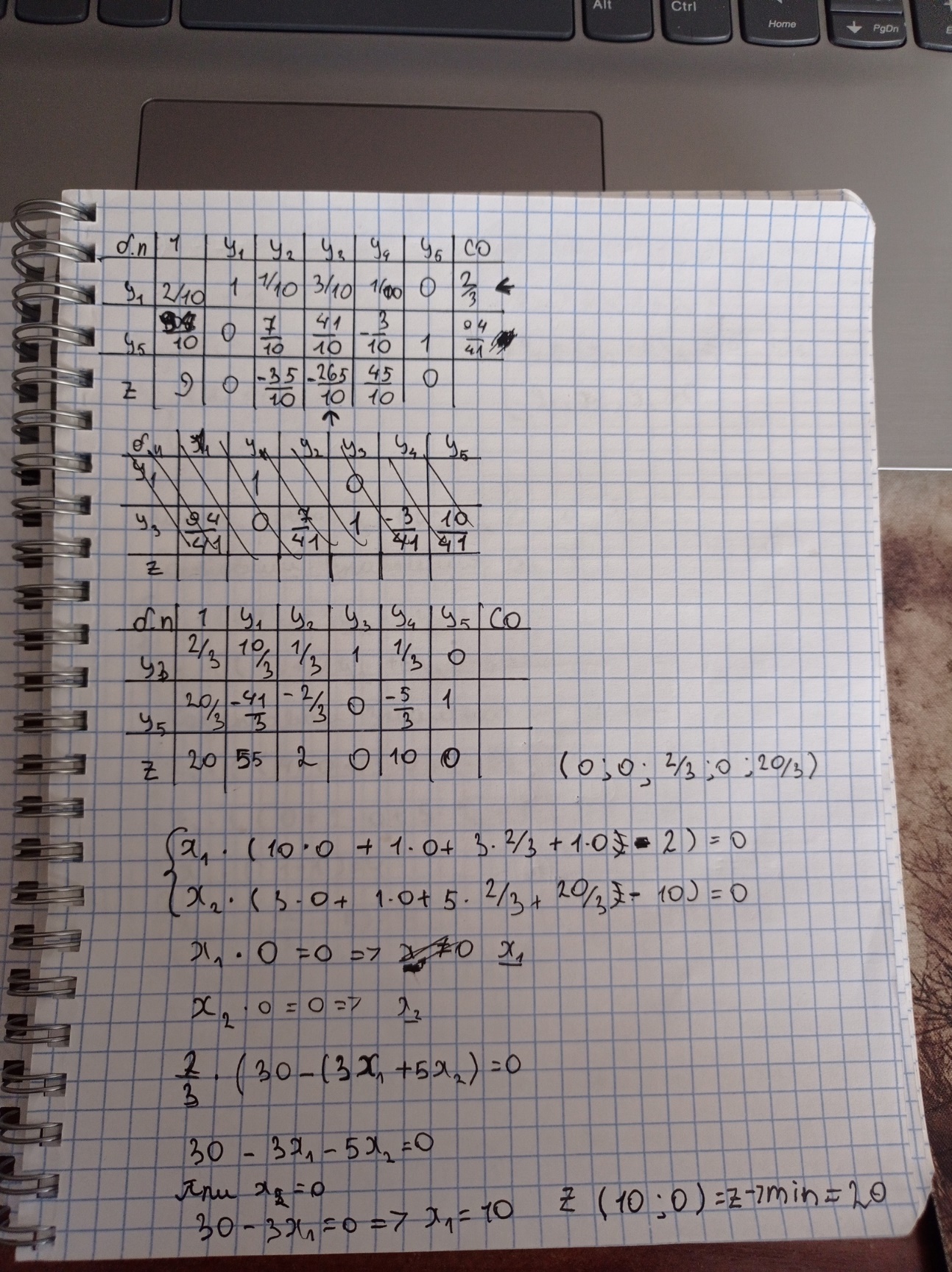


Рис. 7 –Решение двойственной задачи на основании теоремы равновесия.

# Код программы

SimpleFrac.py – файл с описанием класса дроби

class SimpleFrac:

a = 0

b = 1

def \_\_init\_\_(self):

self.a = 0

self.b = 1

def write(self, x: int, y: int):

self.a = x

self.b = y

def \_\_str\_\_(self):

if self.b != 1:

return '{}/{}'.format(int(self.a), int(self.b))

else:

return '{}'.format(int(self.a))

def sf\_reduce(self):

temp\_a = abs(self.a)

temp\_b = abs(self.b)

if self.a == 0:

return self

while temp\_a != temp\_b:

if temp\_a > temp\_b:

tmp = temp\_a

temp\_a = temp\_b

temp\_b = tmp

temp\_b = temp\_b - temp\_a

self.a /= temp\_a

self.b /= temp\_a

return self

def \_\_add\_\_(self, other):

result = SimpleFrac()

result.a = self.a \* other.b + other.a \* self.b

result.b = self.b \* other.b

if result.b < 0:

result.a \*= -1

result.b \*= -1

if result.a == 0:

result.b = 1

result.sf\_reduce()

return result

def \_\_sub\_\_(self, other):

result = SimpleFrac()

result.a = self.a \* other.b - other.a \* self.b

result.b = self.b \* other.b

if result.b < 0:

result.a \*= -1

result.b \*= -1

if result.a == 0:

result.b = 1

result.sf\_reduce()

return result

def \_\_mul\_\_(self, other):

result = SimpleFrac()

temp1 = SimpleFrac()

temp2 = SimpleFrac()

temp1.a = self.a

temp1.b = other.b

temp2.a = other.a

temp2.b = self.b

temp1.sf\_reduce()

temp2.sf\_reduce()

result.a = temp1.a \* temp2.a

result.b = temp1.b \* temp2.b

if result.b < 0:

result.a \*= -1

result.b \*= -1

if result.a == 0:

result.b = 1

result.sf\_reduce()

return result

def \_\_truediv\_\_(self, other):

result = SimpleFrac()

temp1 = SimpleFrac()

temp2 = SimpleFrac()

temp1.a = self.a

temp1.b = other.a

temp2.a = other.b

temp2.b = self.b

temp1.sf\_reduce()

temp2.sf\_reduce()

result.a = temp1.a \* temp2.a

result.b = temp1.b \* temp2.b

if result.b < 0:

result.a \*= -1

result.b \*= -1

if result.a == 0:

result.b = 1

result.sf\_reduce()

return result

def \_\_gt\_\_(self, other):

result = self - other

if result.a > 0:

return True

else:

return False

def \_\_ge\_\_(self, other):

result = self - other

if result.a >= 0:

return True

else:

return False

def \_\_lt\_\_(self, other):

result = self - other

if result.a < 0:

return True

else:

return False

def \_\_le\_\_(self, other):

result = self - other

if result.a <= 0:

return True

else:

return False

def \_\_abs\_\_(self):

if self.a < 0:

self.a \*= -1

return self

main.py – главный исполнительный файл

from SimpleFrac import SimpleFrac

import math

def print\_matrix(matrix, n, m, k):

print()

if k == "matrix":

for i in range(m):

for j in range(n + 1):

if j == n:

print("|%10s" % matrix[i][j], end='')

else:

print("%10s" % matrix[i][j], end='')

print()

elif k == "simplex":

for i in range(m + 2):

print("\t", end='')

for j in range(n + 1):

if j == 0:

print("%10s|" % matrix[i][j], end='')

else:

print("%10s" % matrix[i][j], end='')

print()

else:

print("\nERROR: UNKNOWN SITUATION\n")

def check\_m(simplex\_list, n):

k = 0

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

if simplex\_list[i].a > 0:

k = 1

elif simplex\_list[i].a < 0:

return 0

if k == 0 and simplex\_list[0].a == 0:

return 1

else:

return 2

def check\_z(simplex\_list, n):

k = 0

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

if simplex\_list[i].a < 0:

return 0

return 1

def artificial\_basis(matrix, n, m, z):

N = n

simplex = []

basis = []

for i in range(n + m + 1):

basis.append(i)

for i in range(n + 1):

basis[i] = 0

print("\n", "Этап 1: Расширение матрицы")

for i in range(m):

k = 0

for j in range(m):

matrix[i].insert(N + k, SimpleFrac())

if j == i:

matrix[i][N + k].write(1, 1)

k += 1

N += m

print\_matrix(matrix, N, m, "matrix")

print("\n", "Этап 2: Построение Симплекс-таблицы")

for i in range(m):

simplex.append([])

for j in range(N + 1):

if j == 0:

simplex[i].append(matrix[i][N])

else:

simplex[i].append(matrix[i][j - 1])

simplex.append([])

for j in range(n + 1):

if j == 0:

simplex[m].append(z[n])

else:

temp = z[j - 1]

temp.a \*= -1

simplex[m].append(temp)

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

simplex[m].append(SimpleFrac())

simplex.append([])

for j in range(n + 1):

if j == 0:

temp = SimpleFrac()

for i in range(m):

temp = temp + matrix[i][N]

temp.a \*= -1

simplex[m + 1].append(temp)

else:

temp = SimpleFrac()

for i in range(m):

temp = temp + matrix[i][j - 1]

temp.a \*= -1

simplex[m + 1].append(temp)

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

simplex[m + 1].append(SimpleFrac())

print\_matrix(simplex, N, m, "simplex")

print("\n", "Этап 3: Решение методом искуственного базиса")

print\_matrix(simplex, N, m, "simplex")

# print("\n\t", "BASIS = ", basis)

while True:

stop\_kof = check\_m(simplex[m + 1], n) # проверка строки M

if stop\_kof == 0: # если есть отрицательный элемент в строке M

maxi = SimpleFrac()

maxi\_i = -1

for j in range(1, N + 1): # поиск столбца с максимальным по модулю значением в M

if int(simplex[m + 1][j].a) < 0 and simplex[m + 1][j] < maxi:

maxi = simplex[m + 1][j]

maxi\_i = j

mini = SimpleFrac()

mini\_i = -1

k = -1

for i in range(m): # поиск опорного элемента, чьё симплексное отношение будет взято как минимальное

if simplex[i][maxi\_i].a > 0 and simplex[i][0].a > 0:

mini = simplex[i][0] / simplex[i][maxi\_i]

k = i

break

if k == -1:

print("Нет решений")

return 1

for i in range(k, m): # поиск строки с минимальным сиплексным отношением

if simplex[i][maxi\_i].a > 0 and simplex[i][0].a > 0 and simplex[i][0] / simplex[i][maxi\_i] <= mini:

mini = simplex[i][0] / simplex[i][maxi\_i]

mini\_i = i

print("\n\t", "Минимальное СО = (", mini\_i, ",", maxi\_i, ") = ", simplex[mini\_i][0], " / ", simplex[mini\_i][maxi\_i])

mini = simplex[mini\_i][maxi\_i]

for i in range(N + 1): # делим строку с минимальным сиплексным отношением на опорный элемент

simplex[mini\_i][i] = simplex[mini\_i][i] / mini

for i in range(m + 2): # шаг Гаусса

for j in range(N + 1):

if i != mini\_i and j != maxi\_i:

temp\_1 = simplex[i][j] \* simplex[mini\_i][maxi\_i]

temp\_2 = simplex[i][maxi\_i] \* simplex[mini\_i][j]

simplex[i][j] = temp\_1 - temp\_2

for i in range(m + 2): # обнуляем столбец с опорным элементом

if i != mini\_i:

simplex[i][maxi\_i].a = 0

simplex[i][maxi\_i].b = 1

if basis[n + 1 + mini\_i] >= 0: # удаление столбца с искусственной переменной и обновление базиса

# print("\n\t", "DEL = ", basis[n + 1 + mini\_i])

temp\_k = basis[n + 1 + mini\_i]

for i in range(m + 2):

simplex[i].pop(temp\_k)

N -= 1

basis[n + 1 + mini\_i] = -1

for i in range(n + 1 + mini\_i, n + m + 1):

basis[i] -= 1

basis[mini\_i] = maxi\_i

print("BASIS = ", basis)

print\_matrix(simplex, N, m, "simplex")

elif stop\_kof == 1: # если строка M обнулилась

break

elif stop\_kof == 2: # если матрица не разрешима

print("\n", "Система ограничений не совместна")

return 1

print("\n", "Этап 4: Решение симплекс-методом")

simplex.pop(m + 1)

print\_matrix(simplex, N, m - 1, "simplex")

while True:

kof = check\_z(simplex[m], N) # проверяем строку z

if kof == 0: # если в строке есть отрицательный элемент

maxi = SimpleFrac()

maxi\_i = -1

for j in range(1, N + 1): # поиск столбца с максимальным по модулю значением в Z

if int(simplex[m][j].a) < 0 and simplex[m][j] < maxi:

maxi = simplex[m][j]

maxi\_i = j

mini = SimpleFrac()

mini\_i = -1

k = -1

for i in range(m): # поиск опорного элемента, чьё симплексное отношение будет взято как минимальное

if simplex[i][maxi\_i].a > 0 and simplex[i][0].a > 0:

mini = simplex[i][0] / simplex[i][maxi\_i]

k = i

break

if k == -1:

print("Начинайте паниковать")

print("Но сперва проверьте правильно ли записана строка z")

return 1

# print("Cur mini = ", mini)

for i in range(k, m): # поиск строки с минимальным сиплексным отношением

if simplex[i][maxi\_i].a > 0 and simplex[i][0].a > 0 and simplex[i][0] / simplex[i][maxi\_i] <= mini:

mini = simplex[i][0] / simplex[i][maxi\_i]

mini\_i = i

print("\n\t", "Минимальное СО = (", mini\_i, ",", maxi\_i, ") = ", simplex[mini\_i][0], " / ",

simplex[mini\_i][maxi\_i])

mini = simplex[mini\_i][maxi\_i]

for i in range(N + 1): # делим строку с минимальным сиплексным отношением на опорный элемент

simplex[mini\_i][i] = simplex[mini\_i][i] / mini

for i in range(m + 1): # шаг Гаусса

for j in range(N + 1):

if i != mini\_i and j != maxi\_i:

temp\_1 = simplex[i][j] \* simplex[mini\_i][maxi\_i]

temp\_2 = simplex[i][maxi\_i] \* simplex[mini\_i][j]

simplex[i][j] = temp\_1 - temp\_2

for i in range(m + 1): # обнуляем столбец с опорным элементом

if i != mini\_i:

simplex[i][maxi\_i].a = 0

simplex[i][maxi\_i].b = 1

basis[mini\_i] = maxi\_i # записываем новый элемент в базис

print("\n", "BASIS = ", basis)

print\_matrix(simplex, N, m - 1, "simplex")

if kof == 1:

print("Z -> max = Z(", end='')

for i in range(n):

k = -1

for j in range(n):

if i + 1 == basis[j] and j <= m:

print(simplex[j][0], end='');

k = 0

break;

if k != 0:

print(0, end='')

if i != n - 1:

print(", ", end='')

print(") = ", simplex[m][0])

return 0

return 1

if \_\_name\_\_ == '\_\_main\_\_':

matrix = []

z = []

print("Количество переменных: ")

n = int(input())

print("Количество ограничений: ")

m = int(input())

for i in range(m):

matrix.append([])

for j in range(n + 1):

matrix[i].append(SimpleFrac())

print("Заполните матрицу: ")

for i in range(m):

print(i + 1, " столбец")

for j in range(n + 1):

matrix[i][j].write(int(input()), 1)

print('Введите Z: ')

for i in range(n + 1):

if i == n:

print("C = ", end='')

z.append(SimpleFrac())

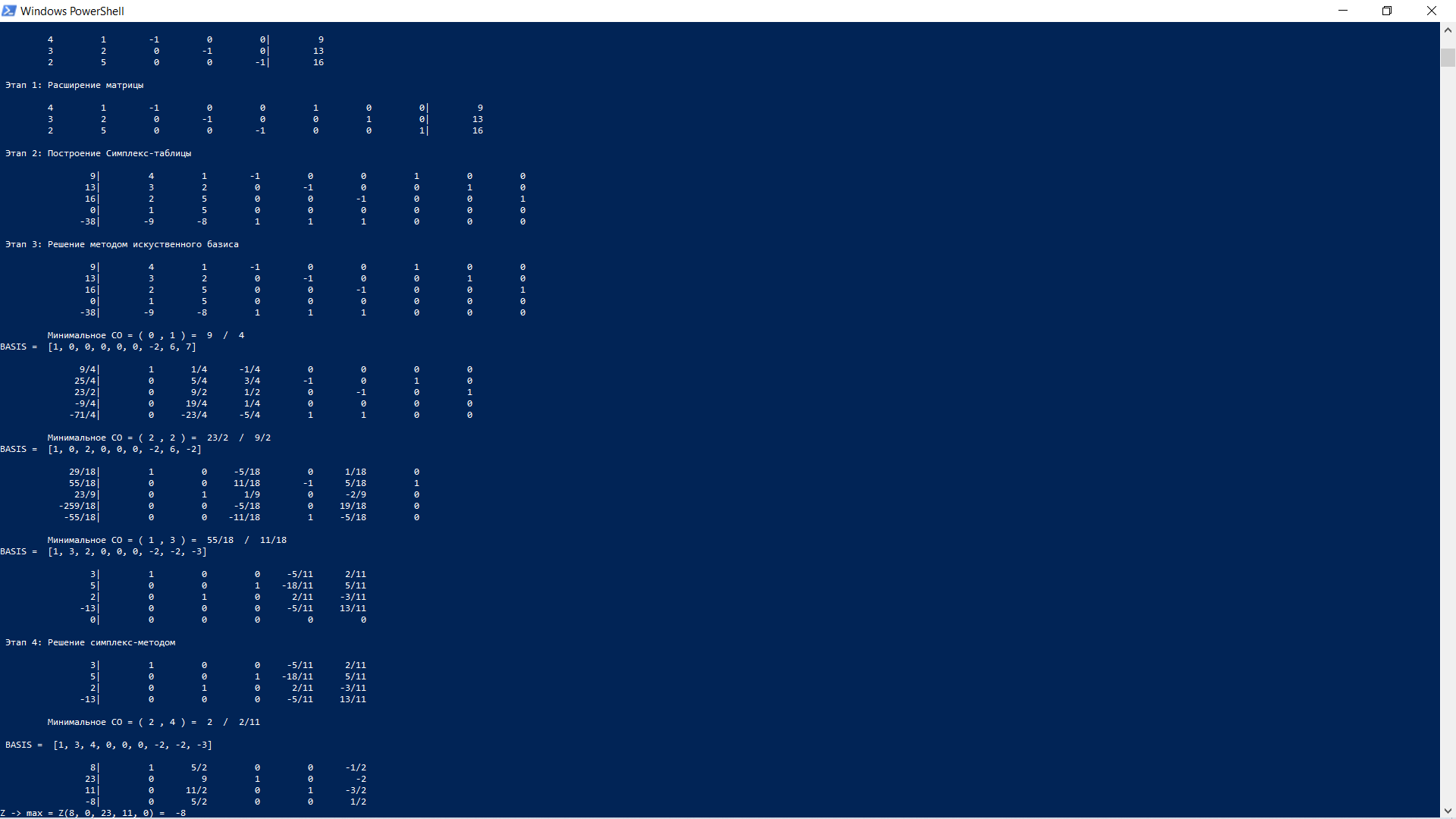
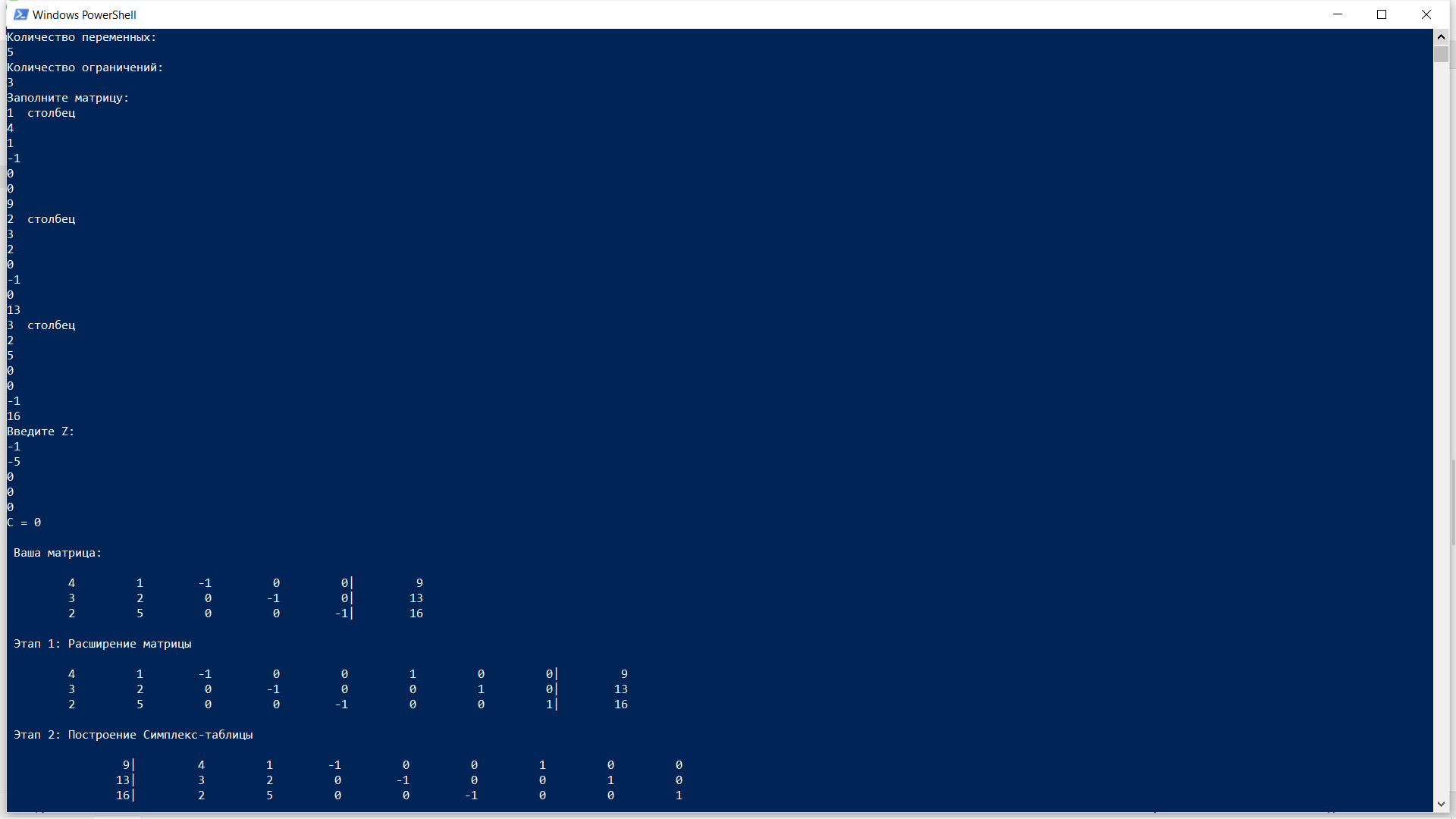
z[i].write(int(input()), 1)

print("\n", "Ваша матрица:")

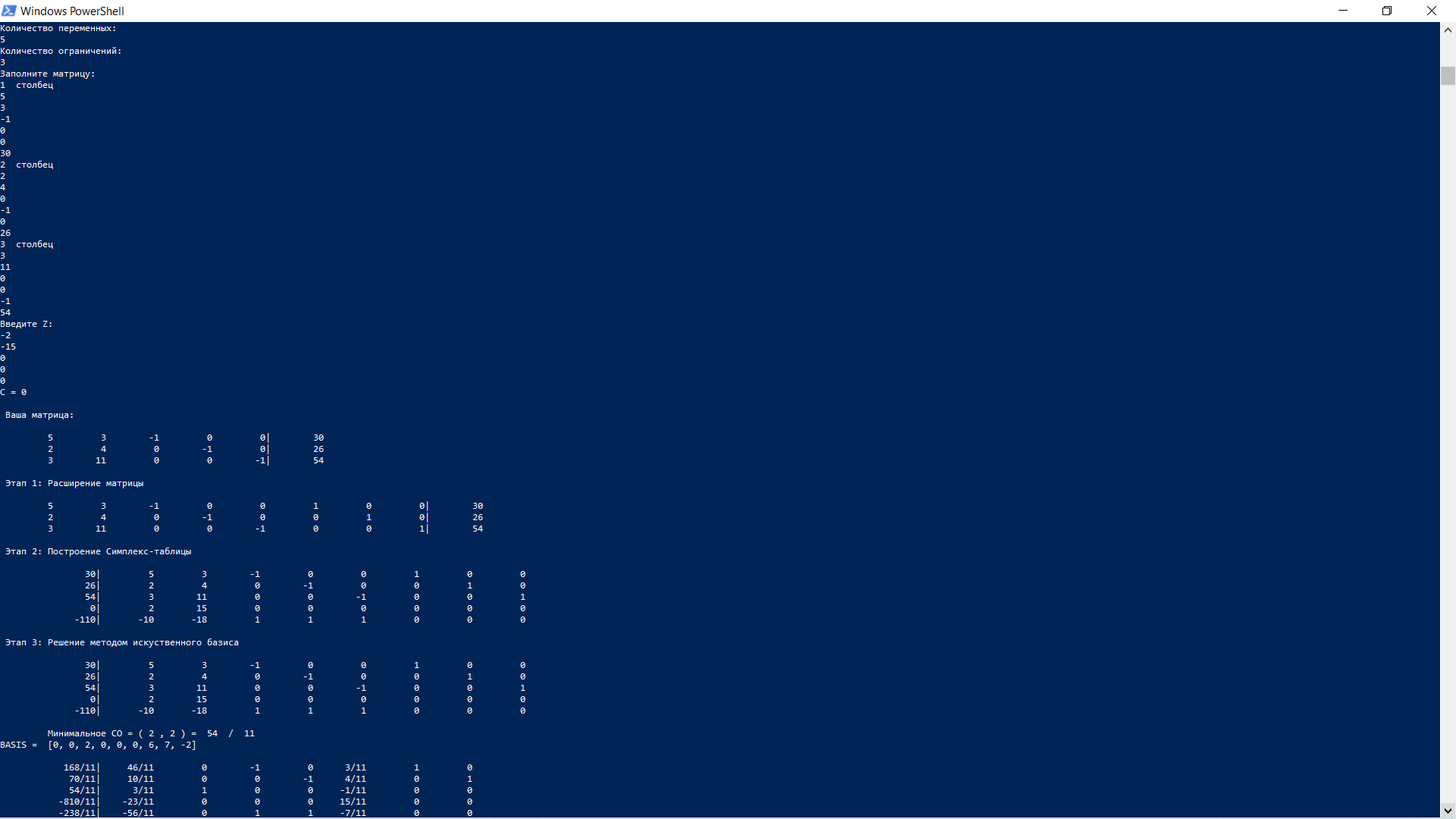
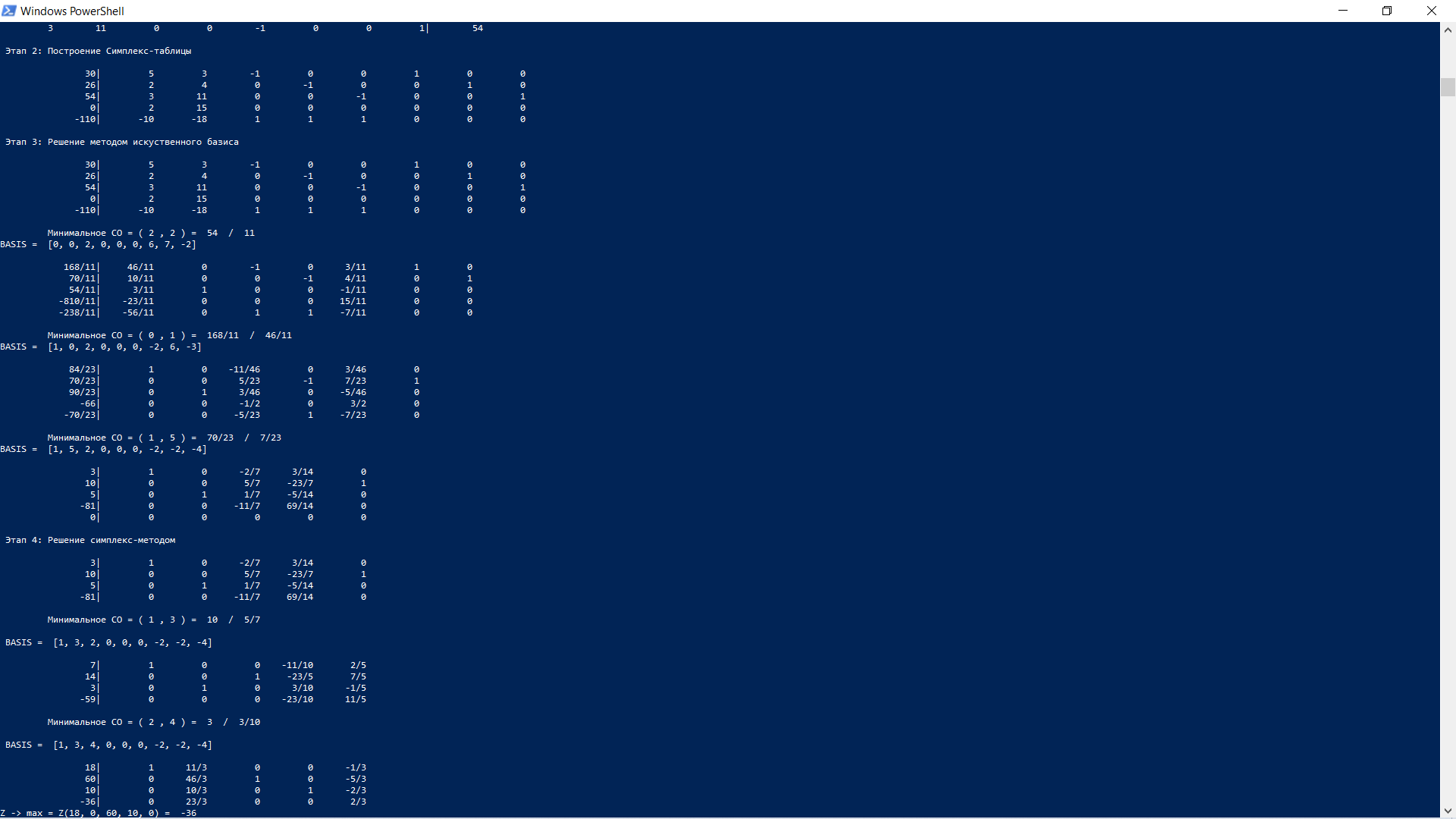
print\_matrix(matrix, n, m, "matrix")

artificial\_basis(matrix, n, m, z)

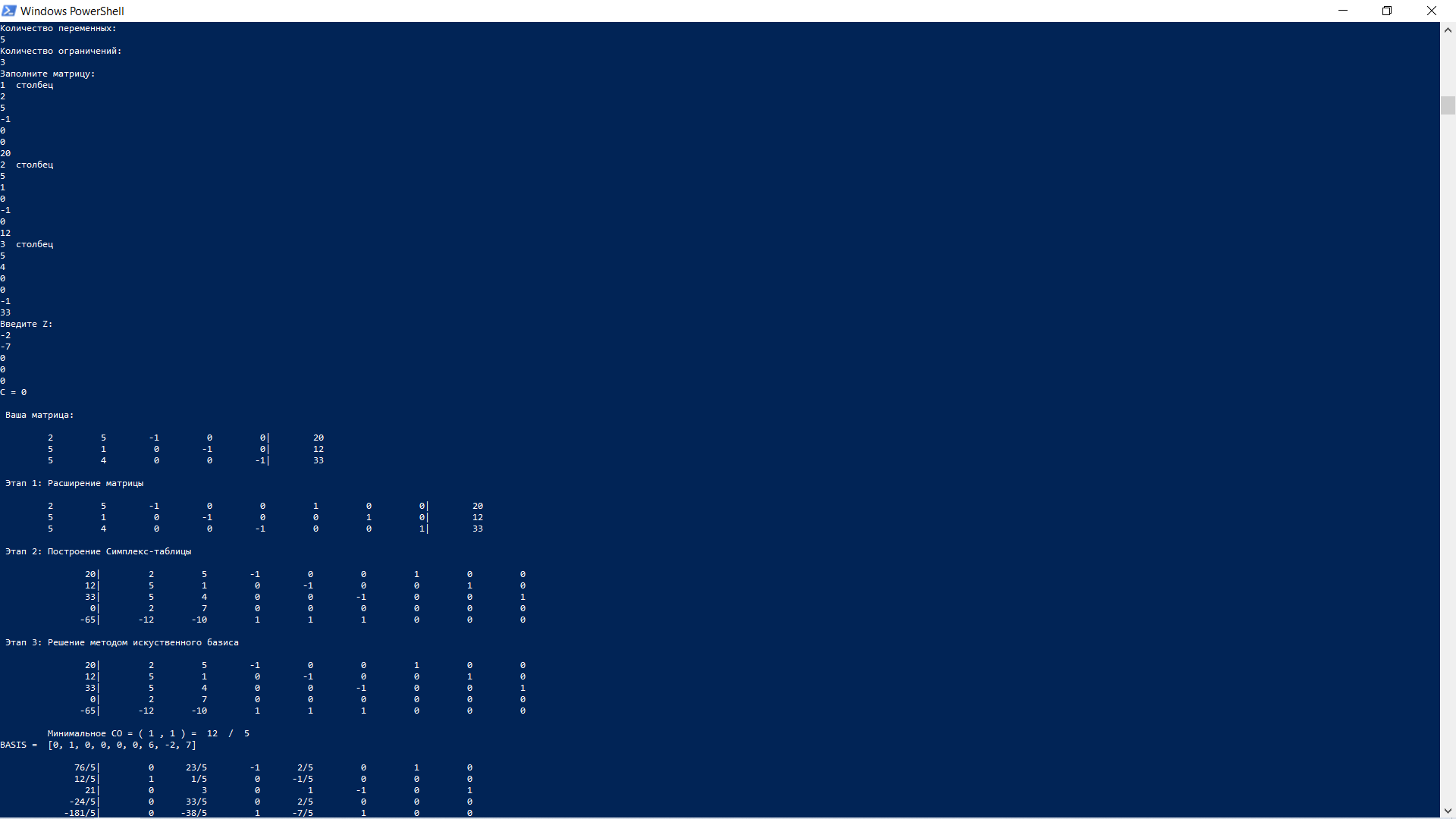
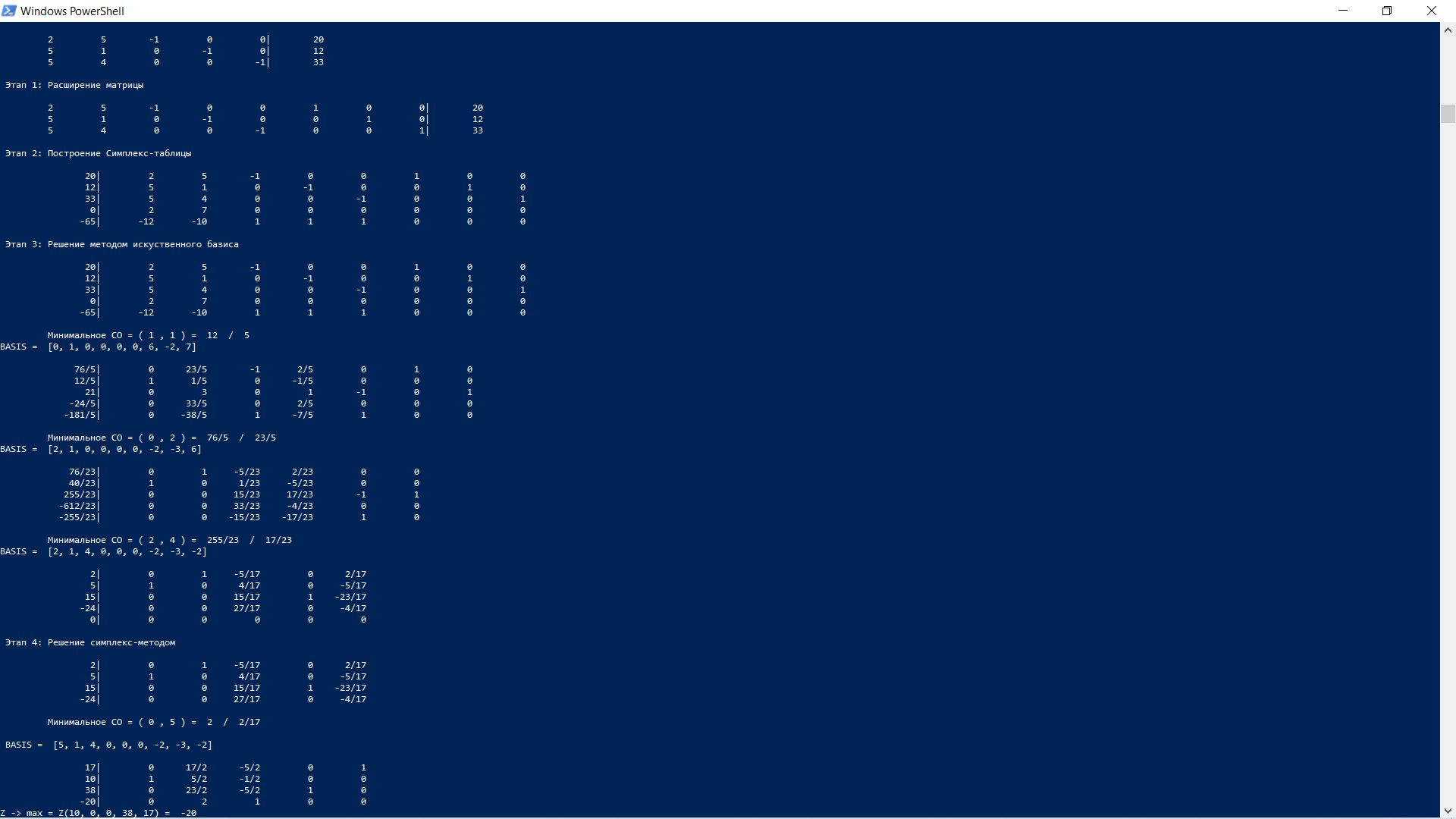
# Примеры решения задач



*Рис. 8,9 Решение задачи из варианта 2*

*Рис. 10,11 Решение задачи из варианта 5*

*Рис. 12, 13 Решение задачи из варианта 11*