Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра информатики

**Лабораторная работа № 1**

**«Решение СЛАУ, вычисление определителей и обратных матриц методом Гаусса и его модификациями»** по учебной дисциплине «Методы численного анализа»

**Выполнили:**

студент гр. 153504 Климкович Н. В.

студент гр. 153504 Тиханёнок И. А.  
 студент гр. 153504 Тарасенко Ф. П.

**Проверила:**

ст. преподаватель кафедры информатики Стройникова Е. Д.

Минск 2022

Содержание

[1. Цель работы 3](https://docs.google.com/document/d/1B2lPiVYRg2lOXK3-2uB3Y39RHP3WV6CL/edit#heading=h.30j0zll)

[2. Теоретические сведения 3](https://docs.google.com/document/d/1B2lPiVYRg2lOXK3-2uB3Y39RHP3WV6CL/edit#heading=h.1fob9te) [3. Программная реализация 7](https://docs.google.com/document/d/1B2lPiVYRg2lOXK3-2uB3Y39RHP3WV6CL/edit#heading=h.26in1rg) [4. Выводы 9](https://docs.google.com/document/d/1B2lPiVYRg2lOXK3-2uB3Y39RHP3WV6CL/edit#heading=h.lnxbz9)

**1. Цель работы:**

1. Изучить теоретические сведения о методе Гаусса и его модификациях.

2. Составить алгоритм решения СЛАУ указанными методами, применимый для организации вычислений на ЭВМ.

3. Составить программу решения СЛАУ по разработанному алгоритму. 4. Продемонстрировать работу программы на примере СЛАУ и

проверить правильность работы программы.

**2. Теоретические сведения**

Задача отыскать решения СЛАУ с n неизвестными является одной из наиболее часто встречающихся вычислительных задач. Хотя задача решения СЛАУ сравнительно редко представляет самостоятельней интерес для приложений, от умения эффективно решать такие системы часто зависит сама возможность математического моделирования с применением ЭВМ разнообразных процессов. Значительная часть численных методов решения различных по своей природе задач (в особенности – нелинейных) включает в себя решение систем линейных уравнений как элементарный шаг соответствующего алгоритма.

СЛАУ обычно записывается в виде 𝑛

∑ 𝑎 𝑥 = 𝑏 ; 𝑖≤1≤𝑛, или коротко *Ax=****b***, где 𝑗=1

*A* = [𝑎11 𝑎12 ... 𝑎1𝑛 ... 𝑎 𝑛1 ... 𝑎 𝑛2 ... ... ... 𝑎 𝑛𝑚 ]  
 a = [𝑥1 ... 𝑥𝑛 ];  
 b = 𝑏1 ... 𝑏𝑛 ].

Здесь *А* и ***b*** заданы и требуется найти x.

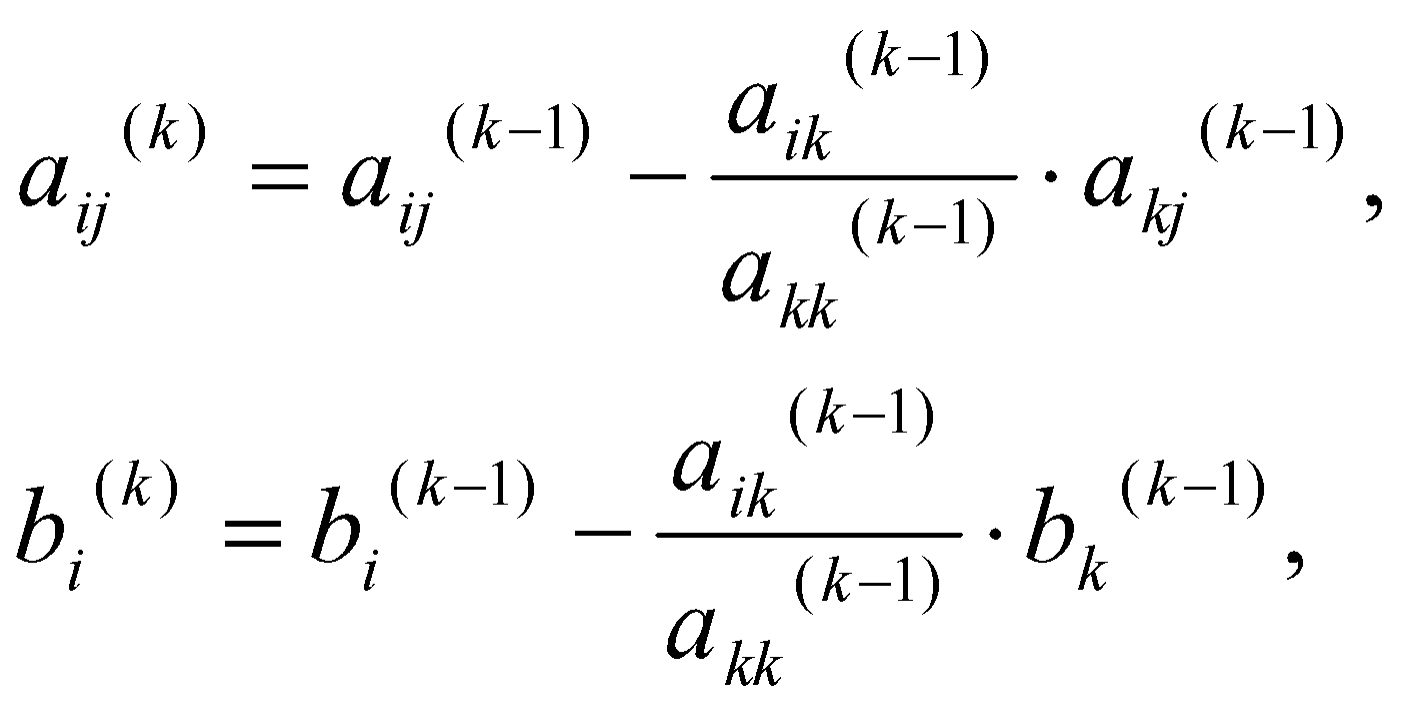
Методы решения СЛАУ делятся на прямые и итерационные.

Прямые методы дают в принципе точное решение за конечное число арифметических операций. Они просты и наиболее универсальны. Для хорошо обусловленной системы необходимого порядка n ≤ 200 применяются практически только прямые методы.

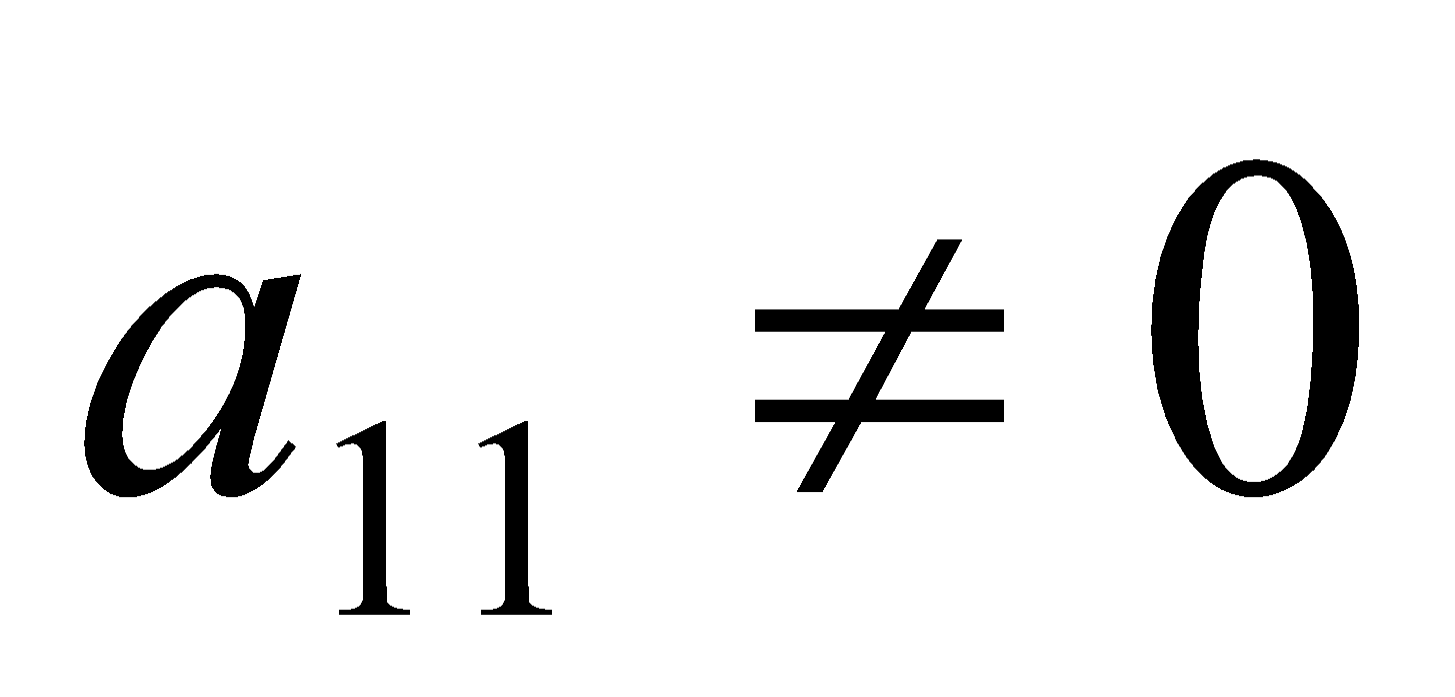
Наибольшее распространение среди методов получил метод Гаусса и его модификации.

**Метод Гаусса**

Метод Гаусса является одним из самых распространенных методов решения систем линейных уравнений. Этот метод, который также называют *методом последовательного исключения неизвестных*, известен в различных вариантах. Он состоит из прямого и обратного ходов.



Вычисления с помощью метода Гаусса на этапе прямого хода заключаются в последовательном исключении неизвестных из системы для преобразования ее к равносильной системе с верхней треугольной (трапециевидной) матрицей. Вычисления значений неизвестных производят на этапе обратного хода.

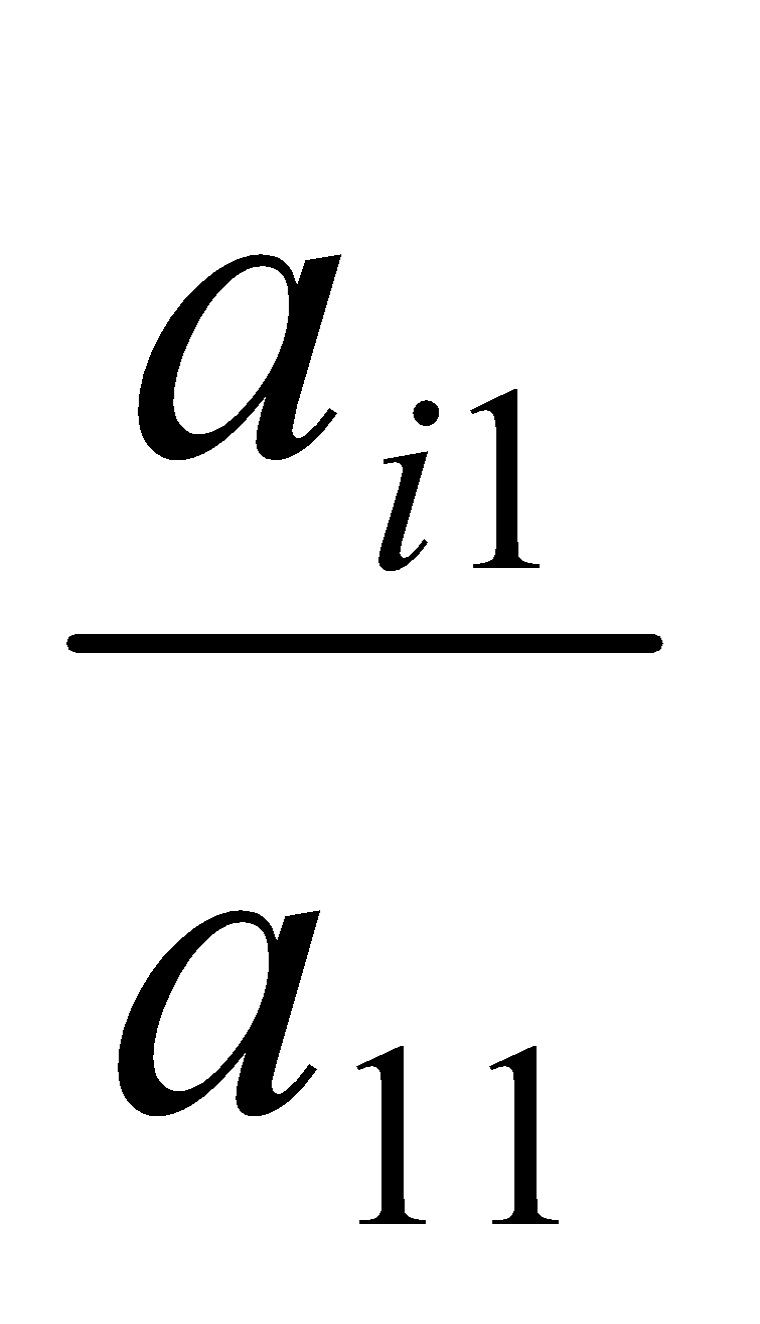
Пусть система (2.1) содержит *n* уравнений с *n* неизвестными. На первом шаге надо исключить неизвестное *x*1 из уравнений с номерами *i* =

2, 3, …, *n*. Пусть элемент первого шага.

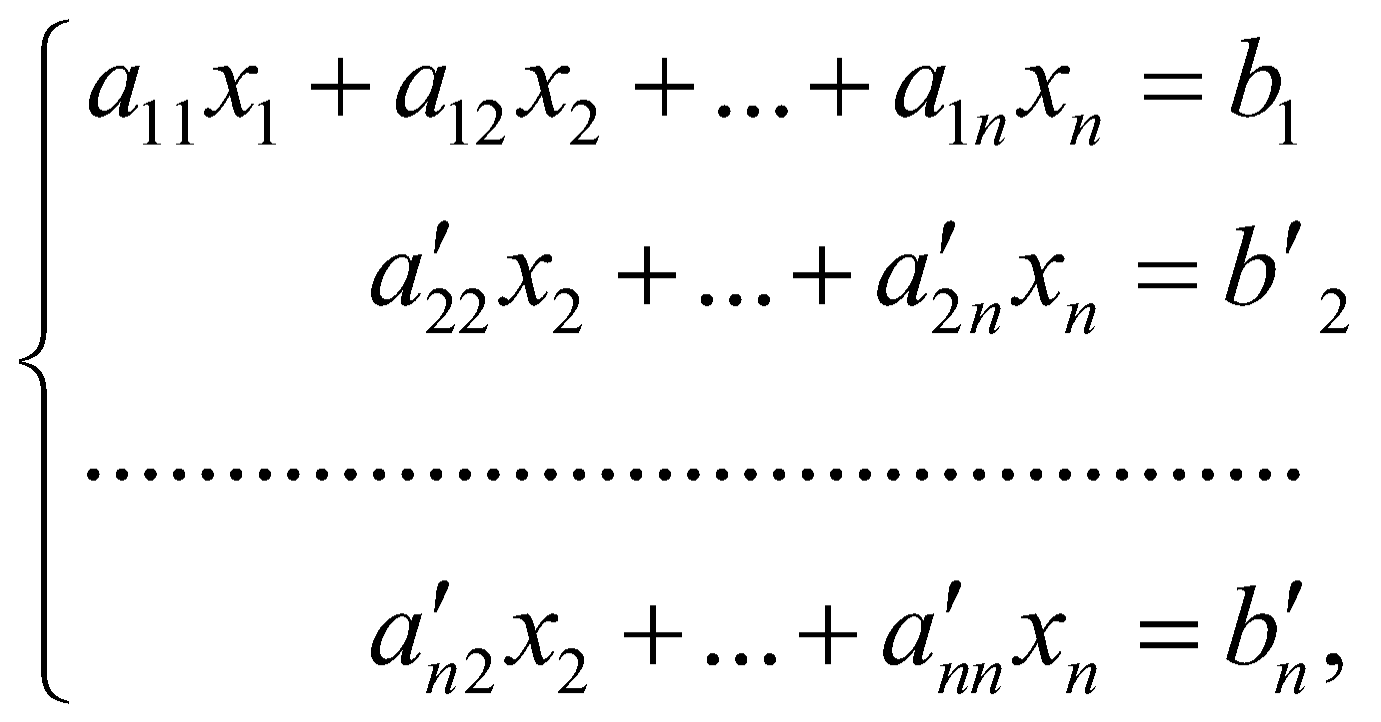
Вычтем последовательно из

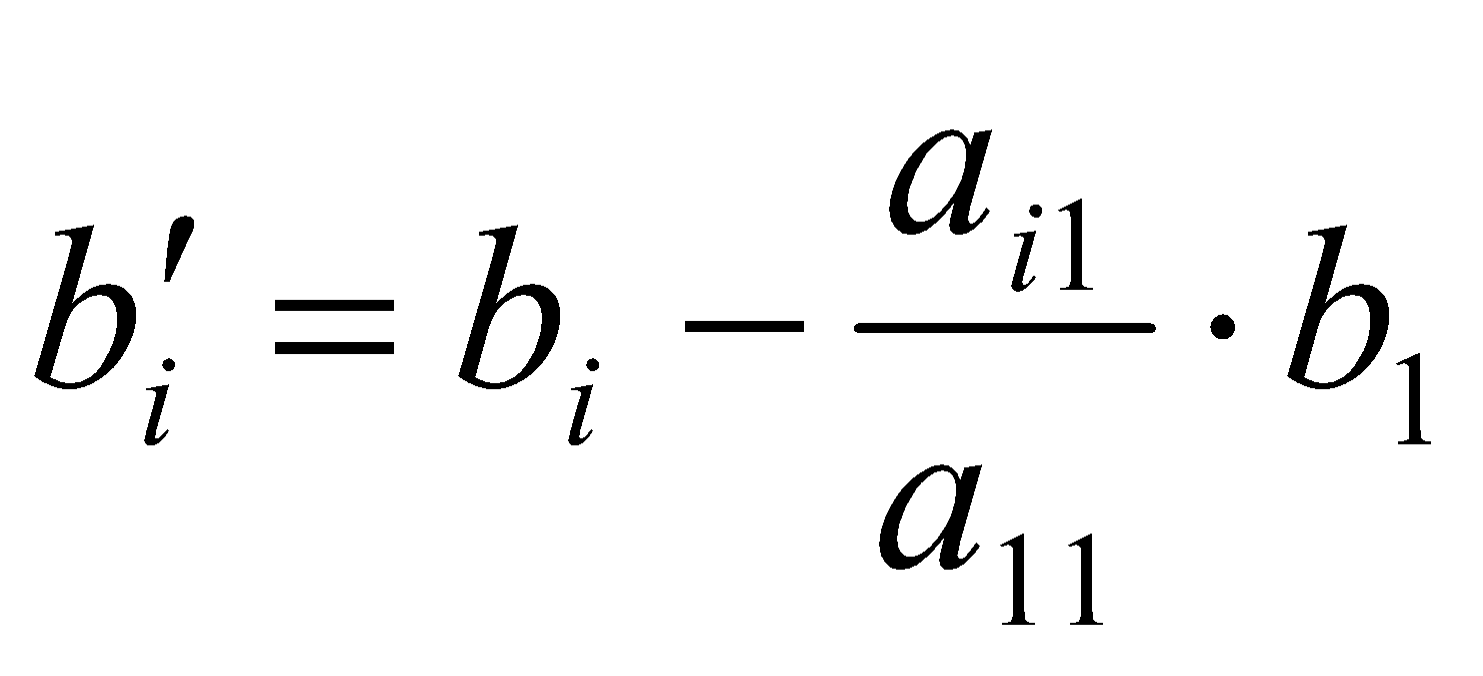
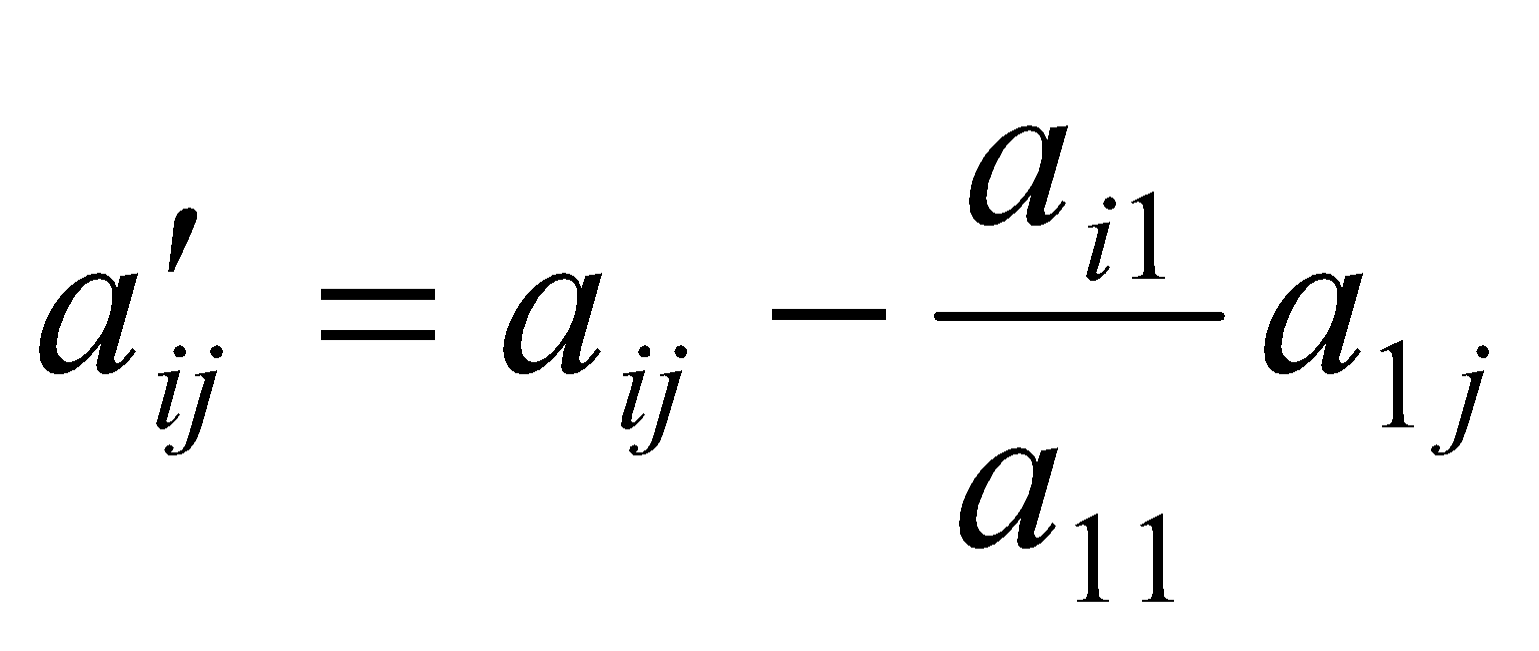
. Он называется *ведущим элементом*

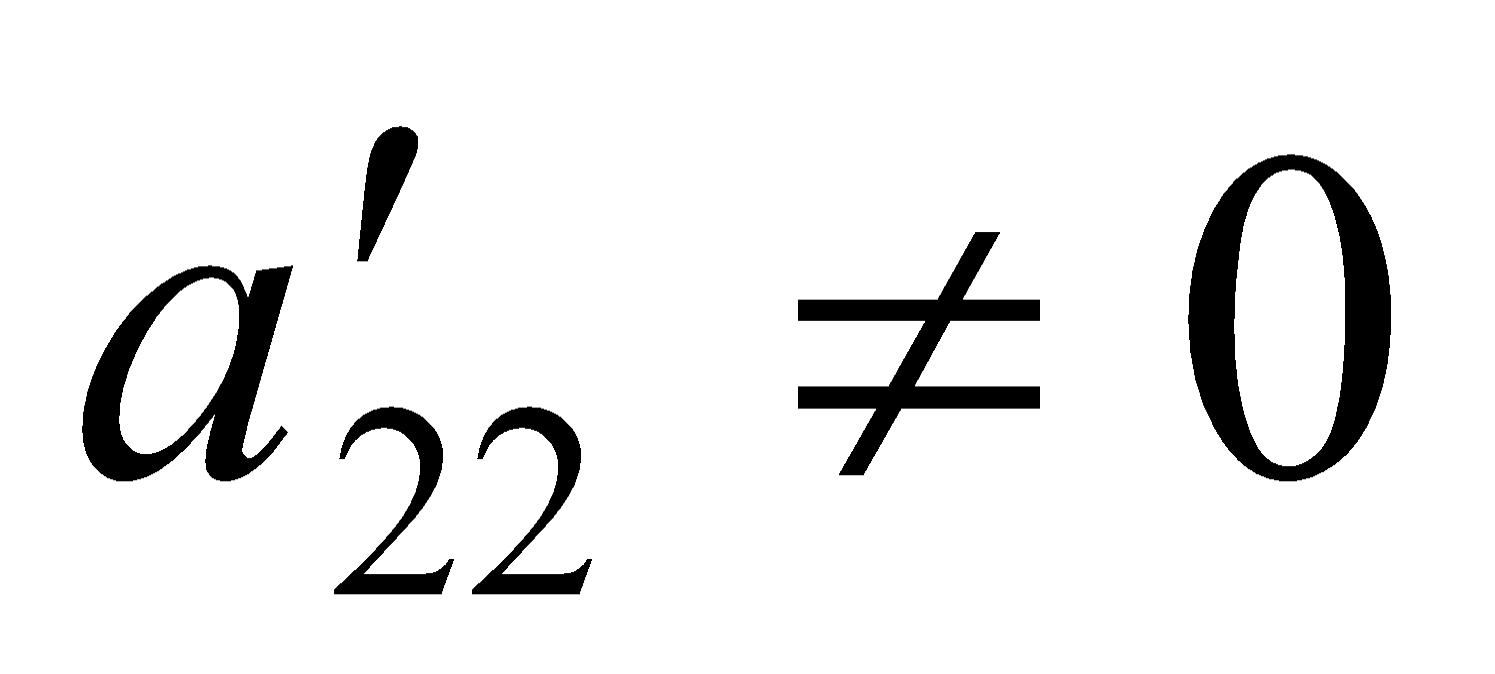
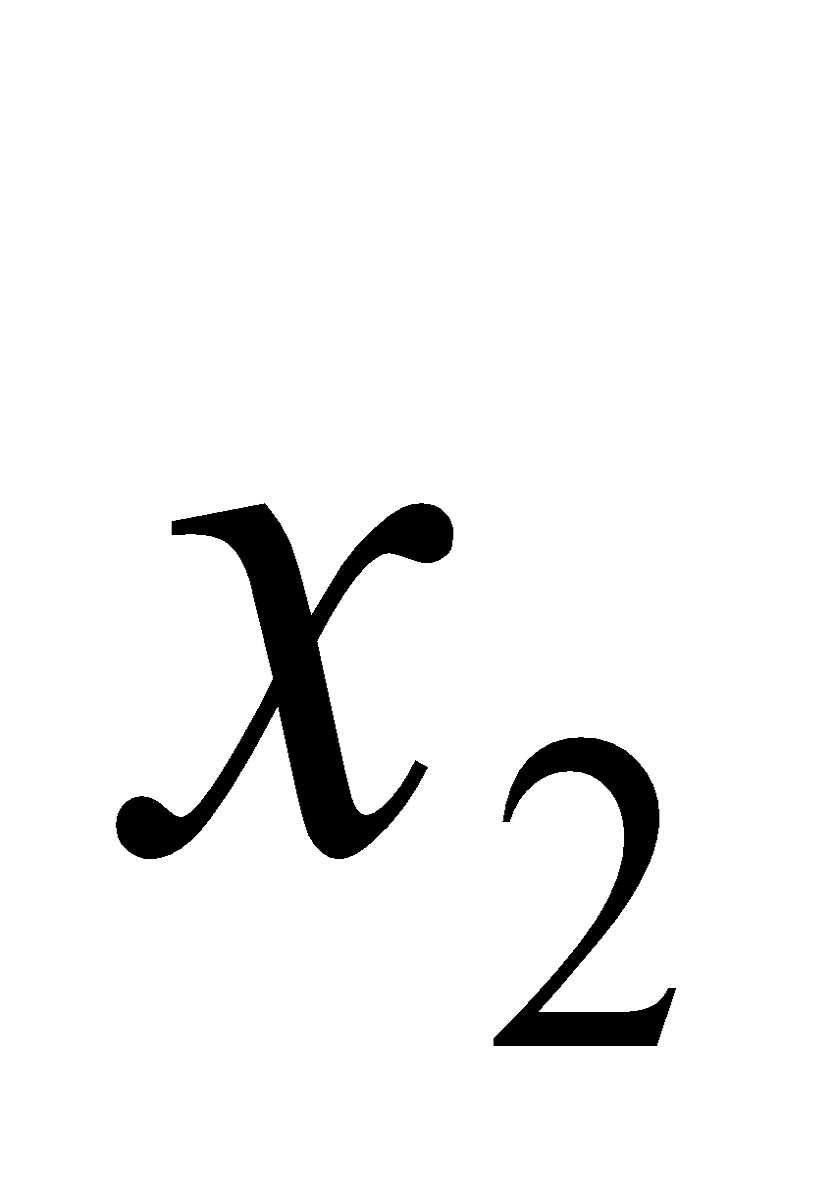
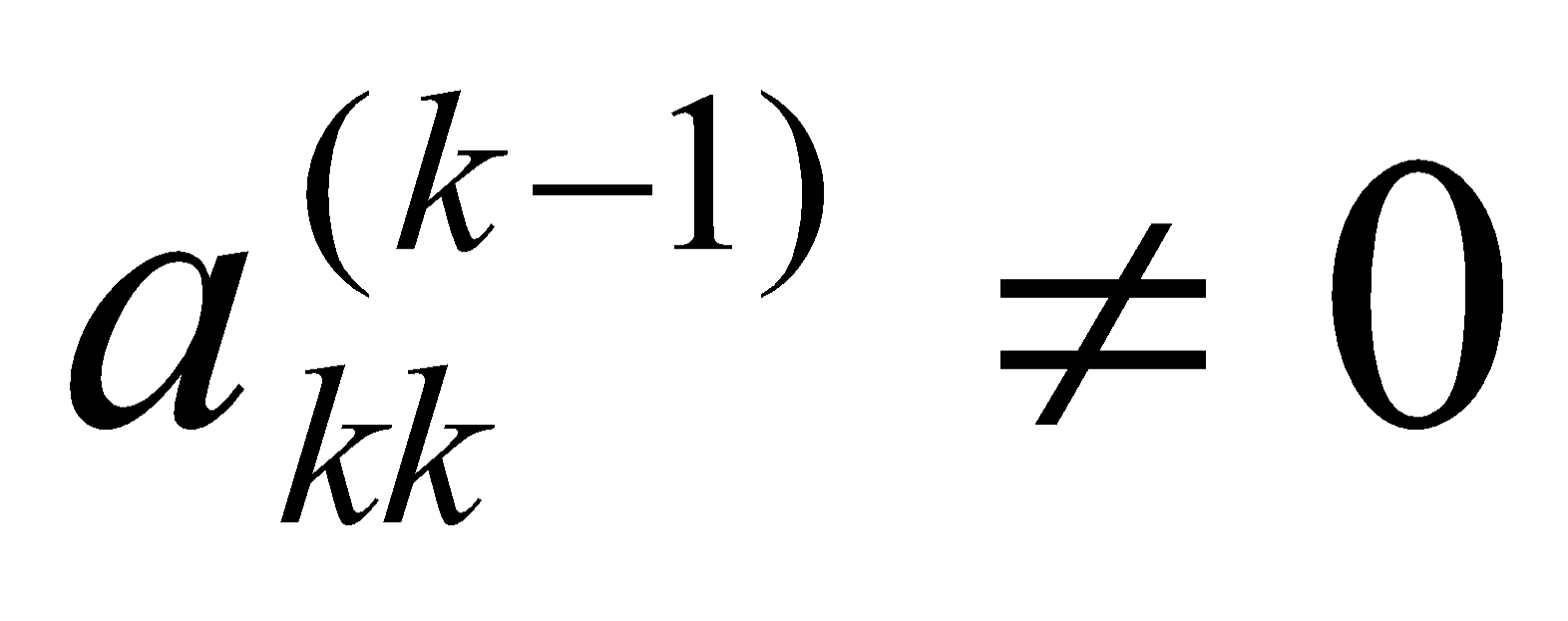
второго, третьего, …, *n-*го уравнений

системы первое уравнение, умноженное соответственно на .

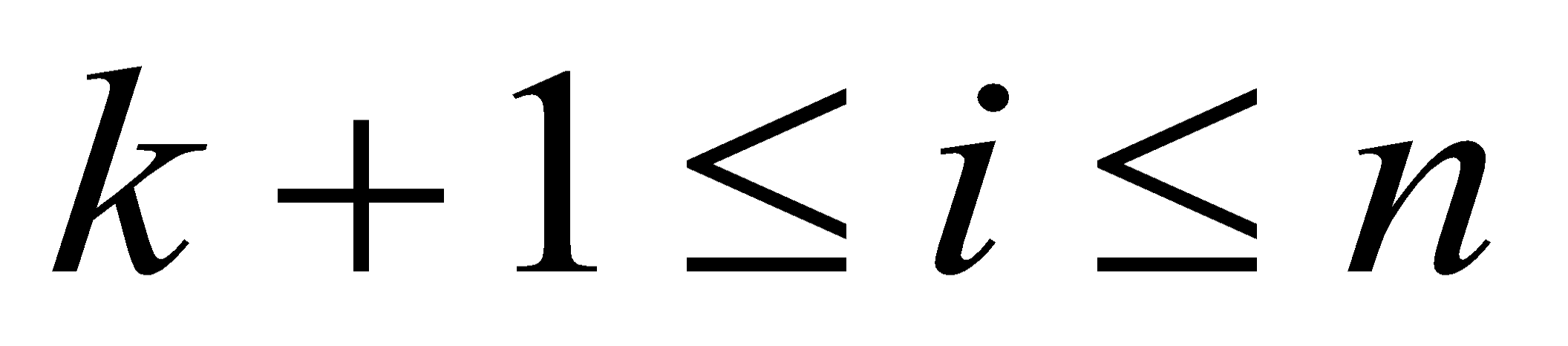
Это позволит обратить в нуль коэффициенты при *x*1 во всех уравнениях, кроме первого. В результате получим равносильную систему:



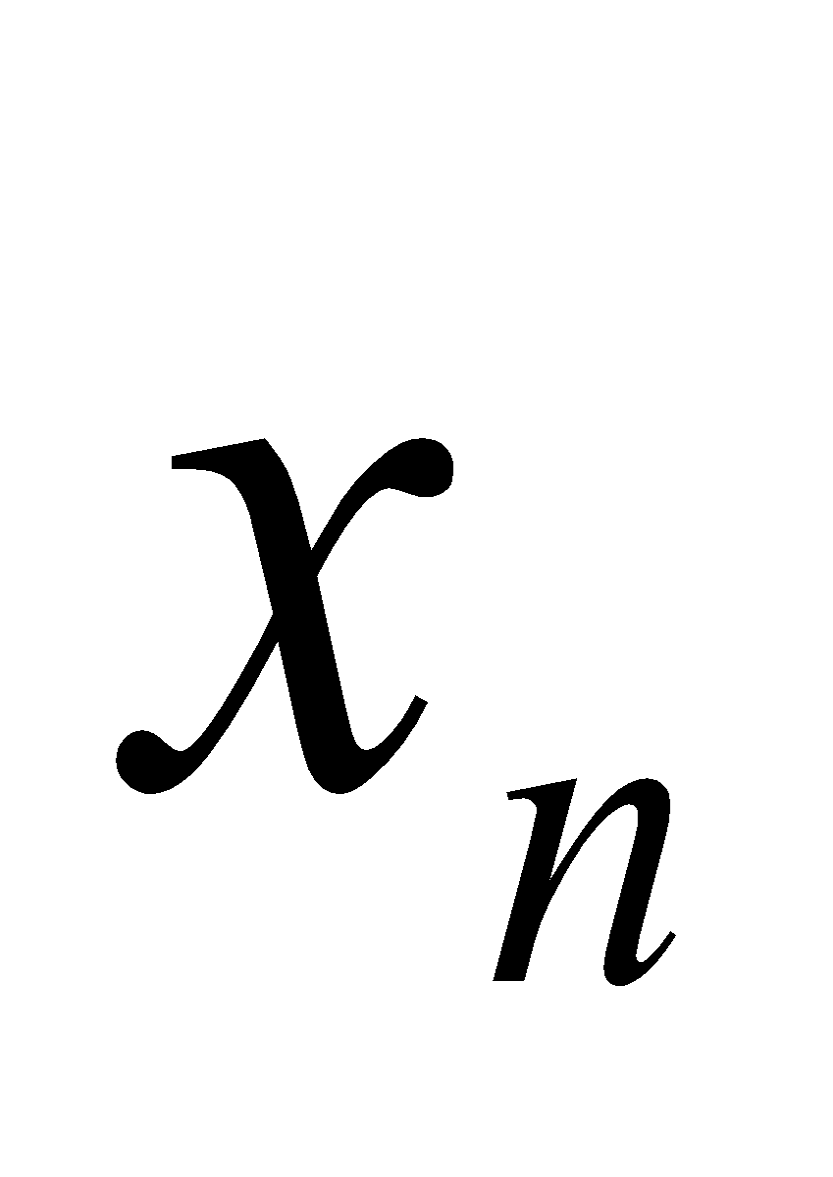
где , , *i* = 2, 3, …, *n*, *j* = 1, 2, …, *n*.

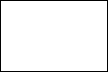
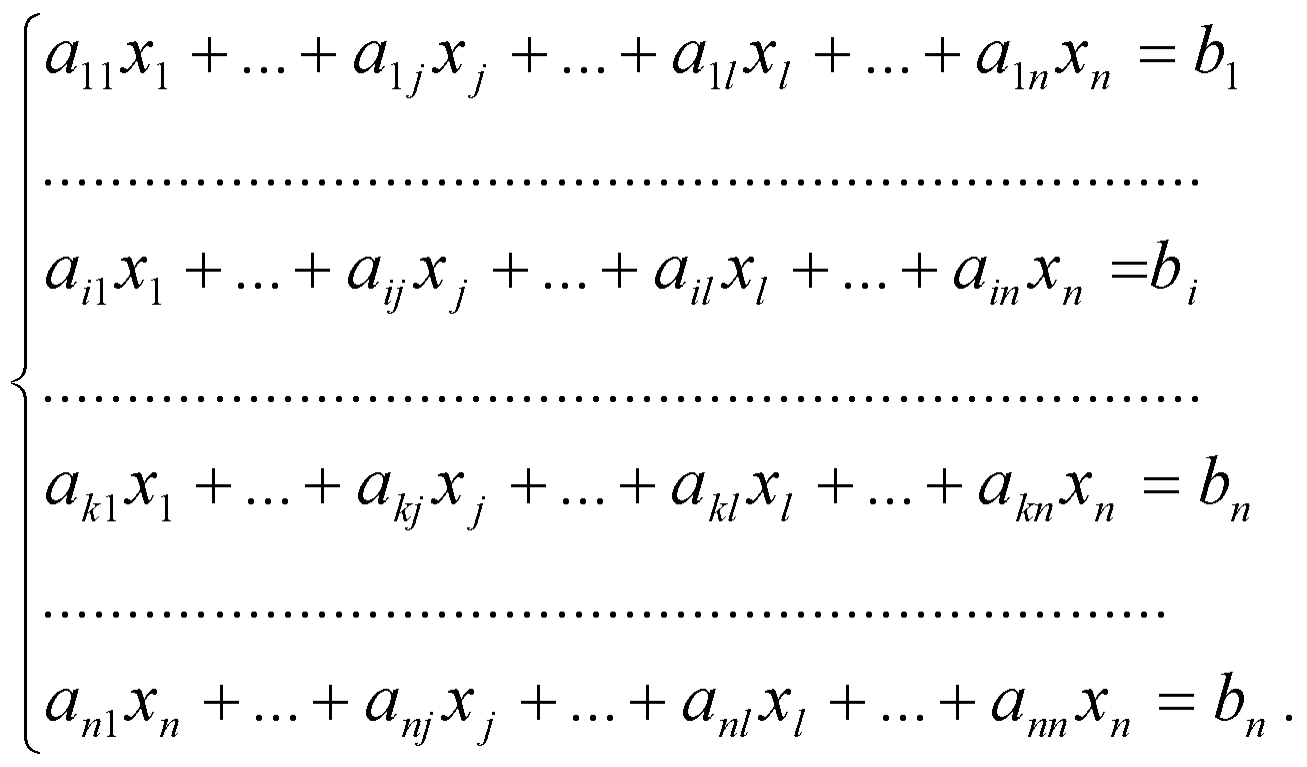
Продолжим дальше, предполагая, что и исключим неизвестное из уравнений, начиная с третьего и так далее. На *k*-м шаге

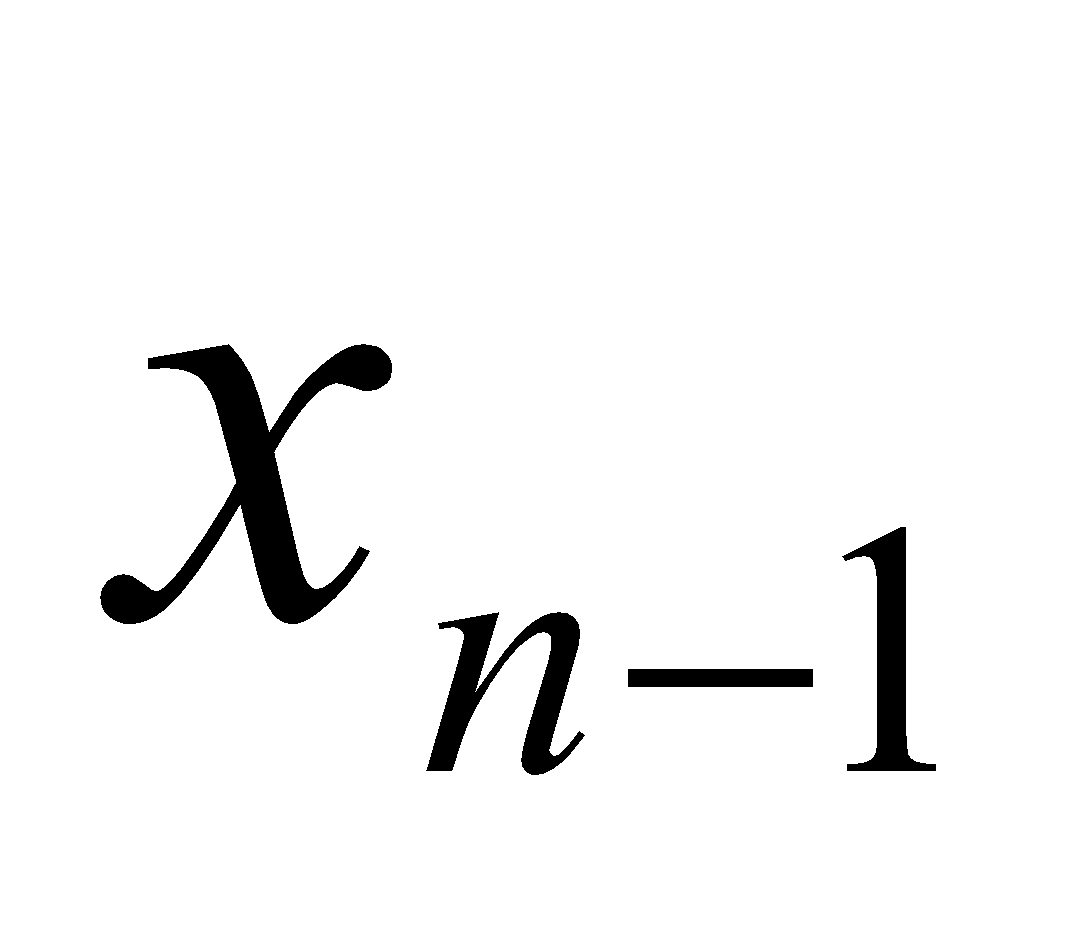
предполагаем, что ведущий элемент *k*-го шага , и, продолжая процесс, получаем формулы для преобразования элементов матрицы на данном шаге:

, *k* ≤ *j* ≤ *n*.

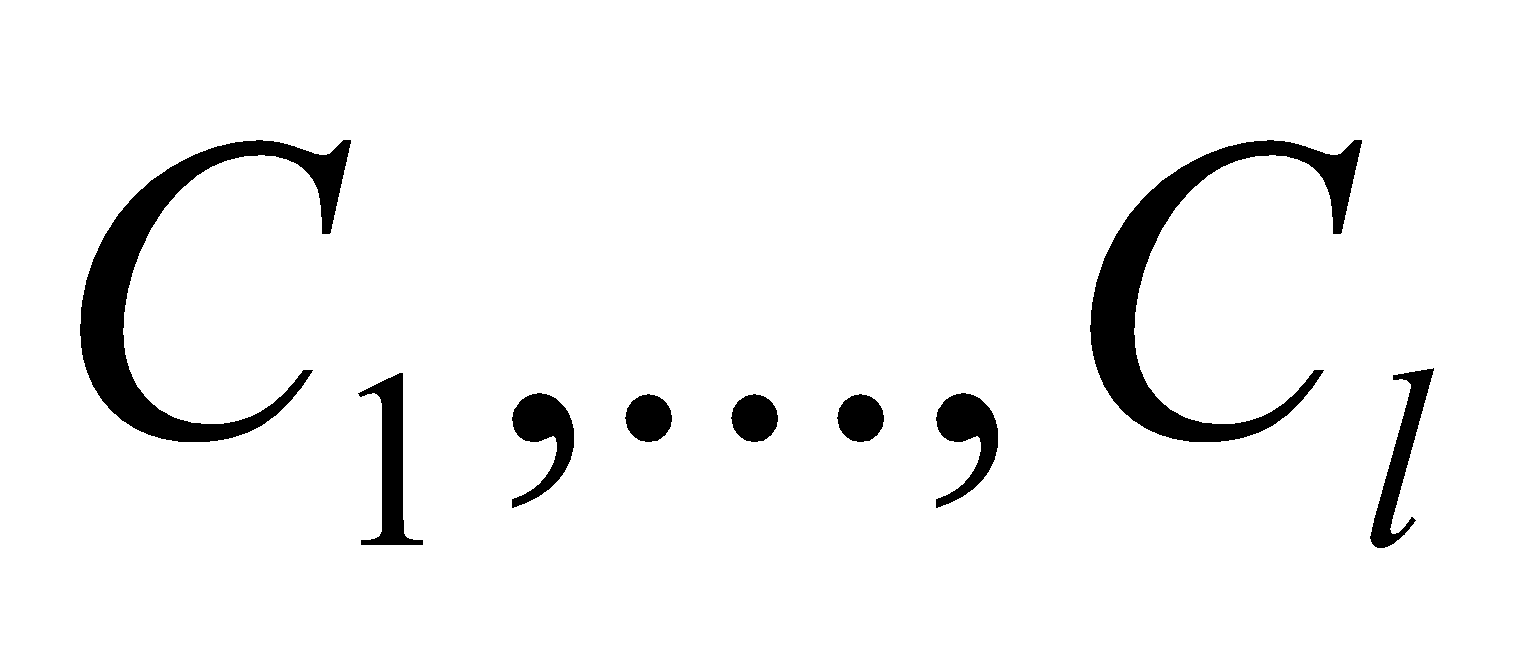
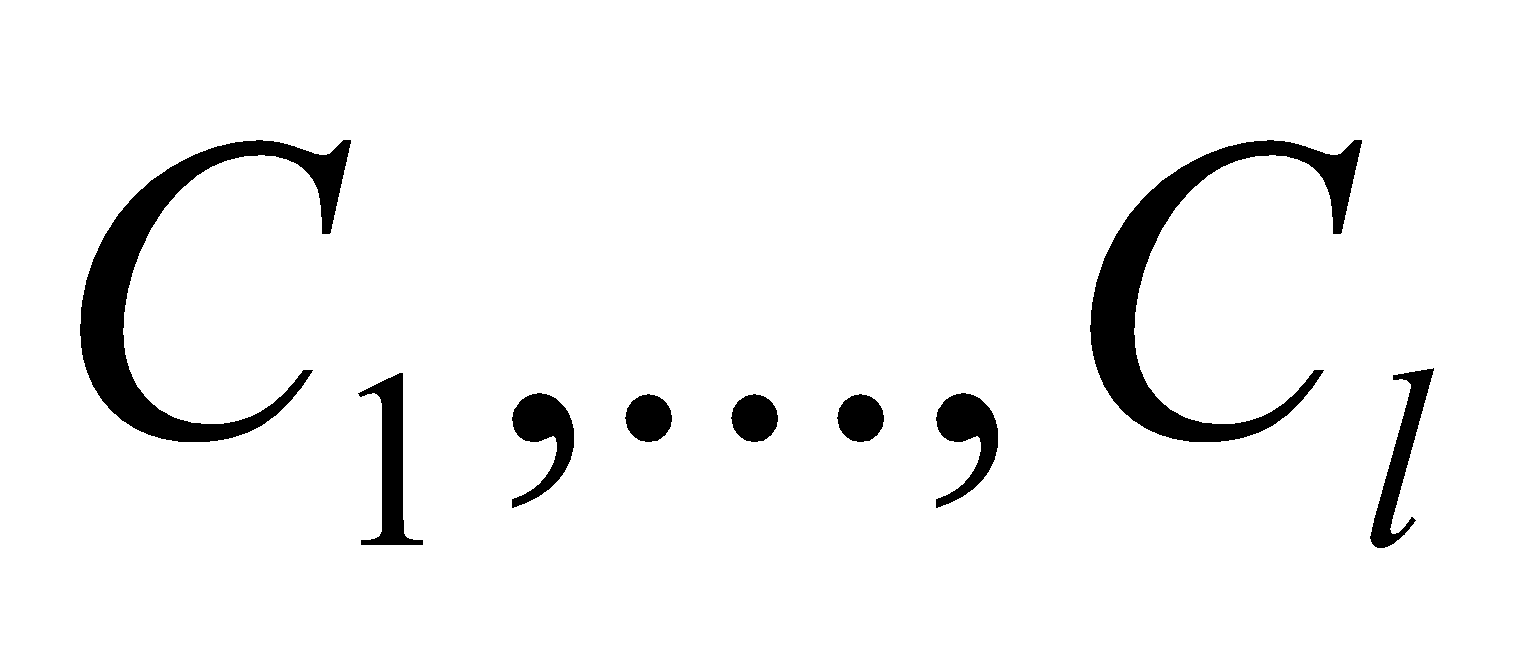
Это так называемый *прямой ход* метода Гаусса. Если после него какое-то уравнение представляет собой невыполнимое равенство, это означает, что система не имеет решений. В противном случае после (*n–*1)*-*го шага исключений можем получить треугольную систему, из

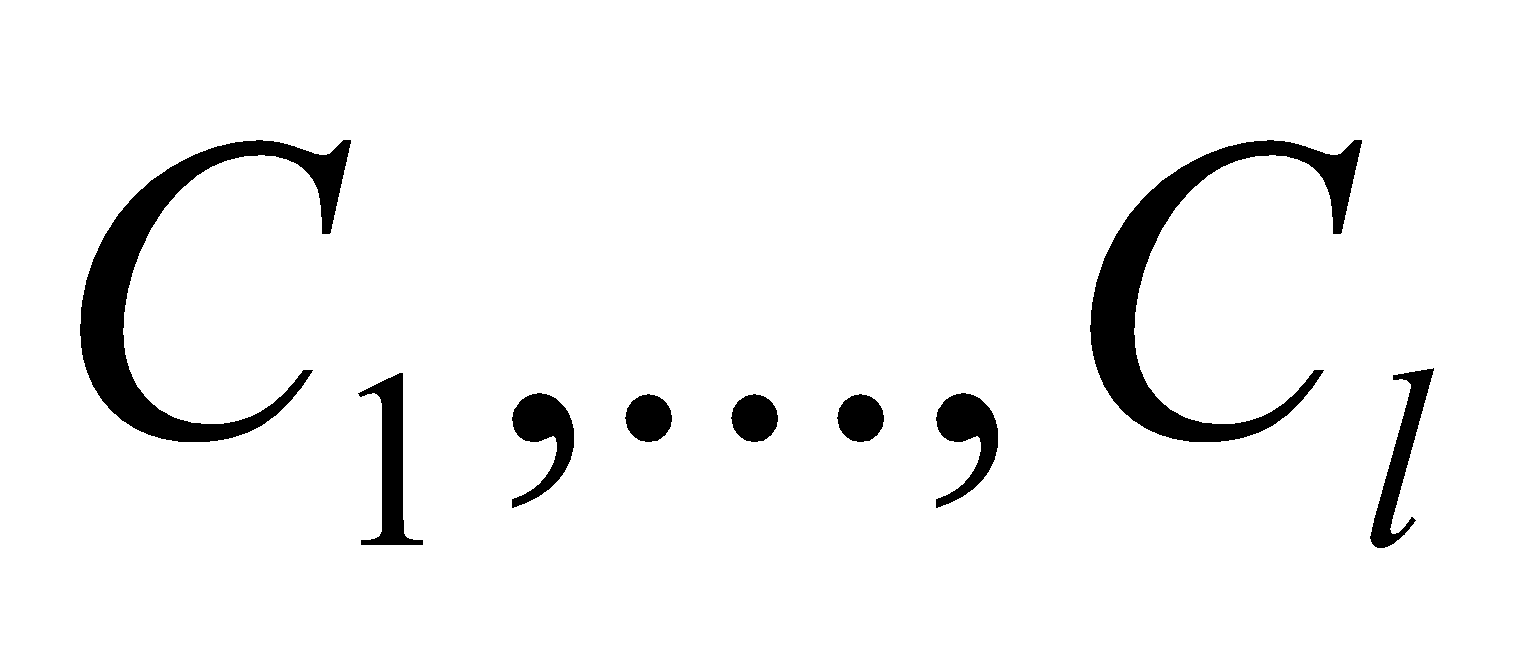
последнего уравнения которой мы найдем . Подставляя его в



предпоследнее уравнение, найдем . И так далее. Этот процесс называется *обратным ходом* метода Гаусса.

Возможна также ситуация, когда в процессе прямого хода получается трапециевидная система, где в последнем оставшемся уравнении имеется более одной переменной. В этом случае придаем всем оставшимся в последнем уравнении неизвестным, кроме первого, произвольные

значения , затем из последнего уравнения через выражается оставшееся неизвестное и подставляется в предыдущие уравнения. Постепенно все неизвестные выражаются через параметры

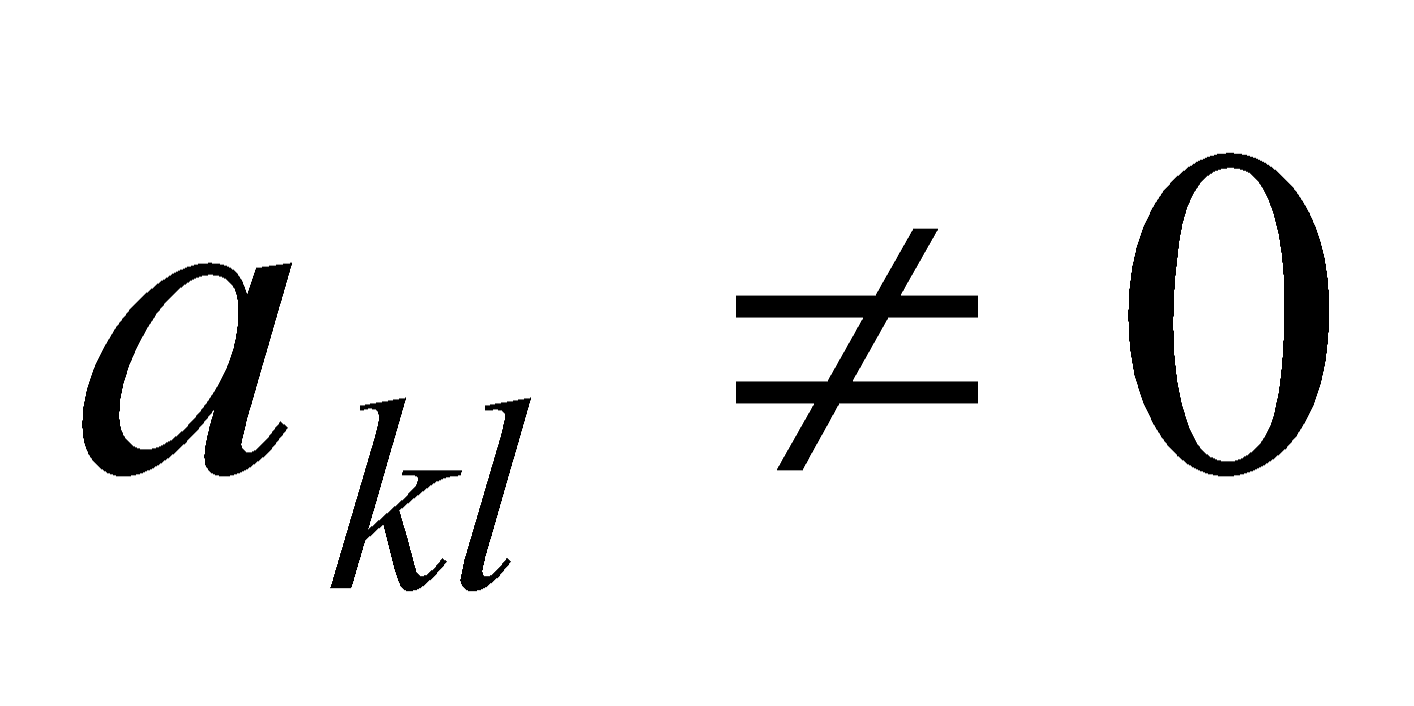
, которые могут иметь произвольные числовые значения. Таким образом, в последнем случае система имеет бесконечно много решений.

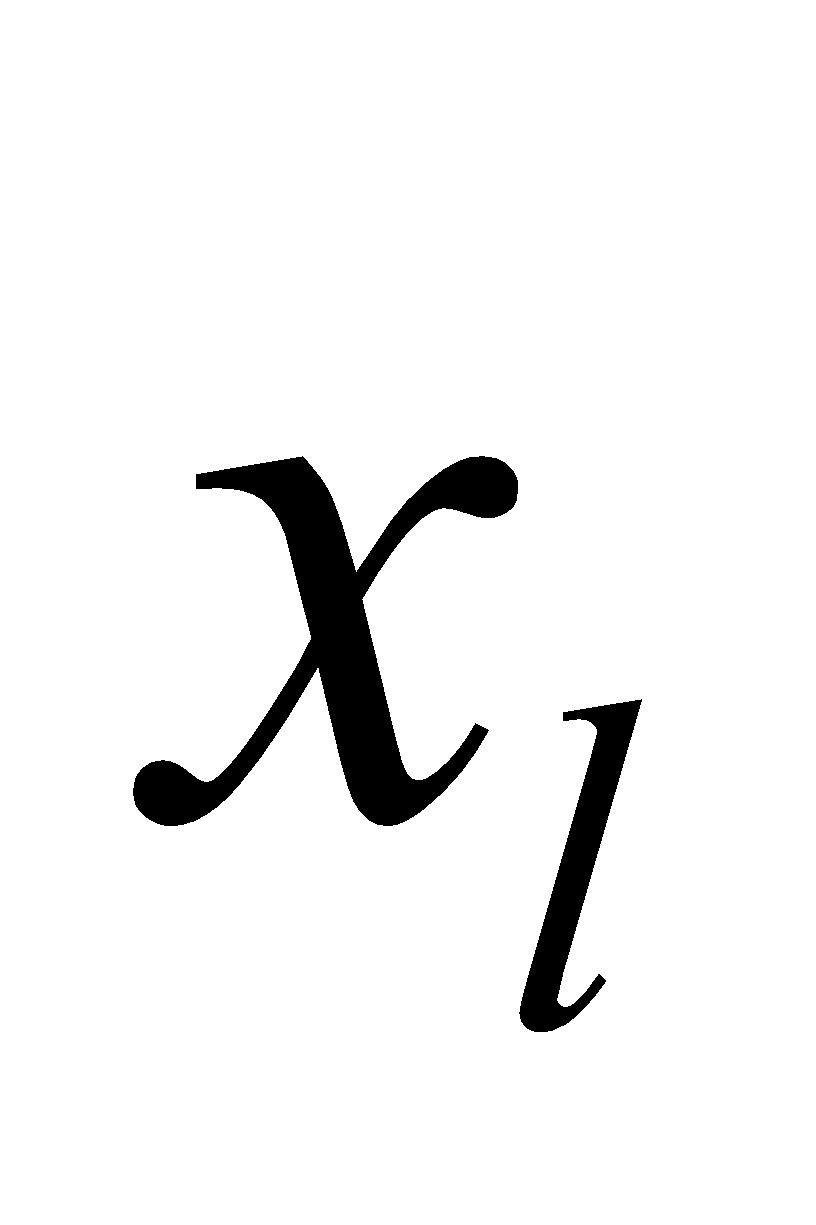
*Замечание.* При реализации метода Гаусса на каждом шаге производится деление на соответствующий ведущий элемент, поэтому предполагается, что эти элементы не должны быть равными нулю. В противном случае проводится перенумерация уравнений (при перестановке строк) и неизвестных (при перестановке столбцов).

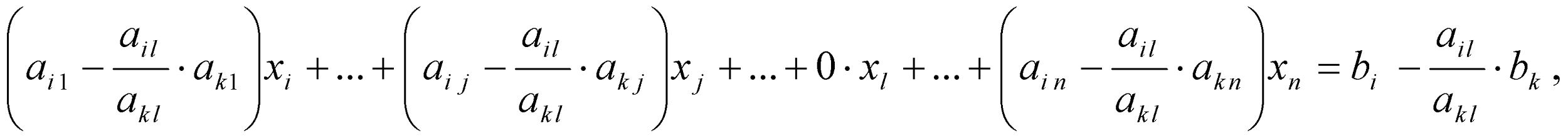
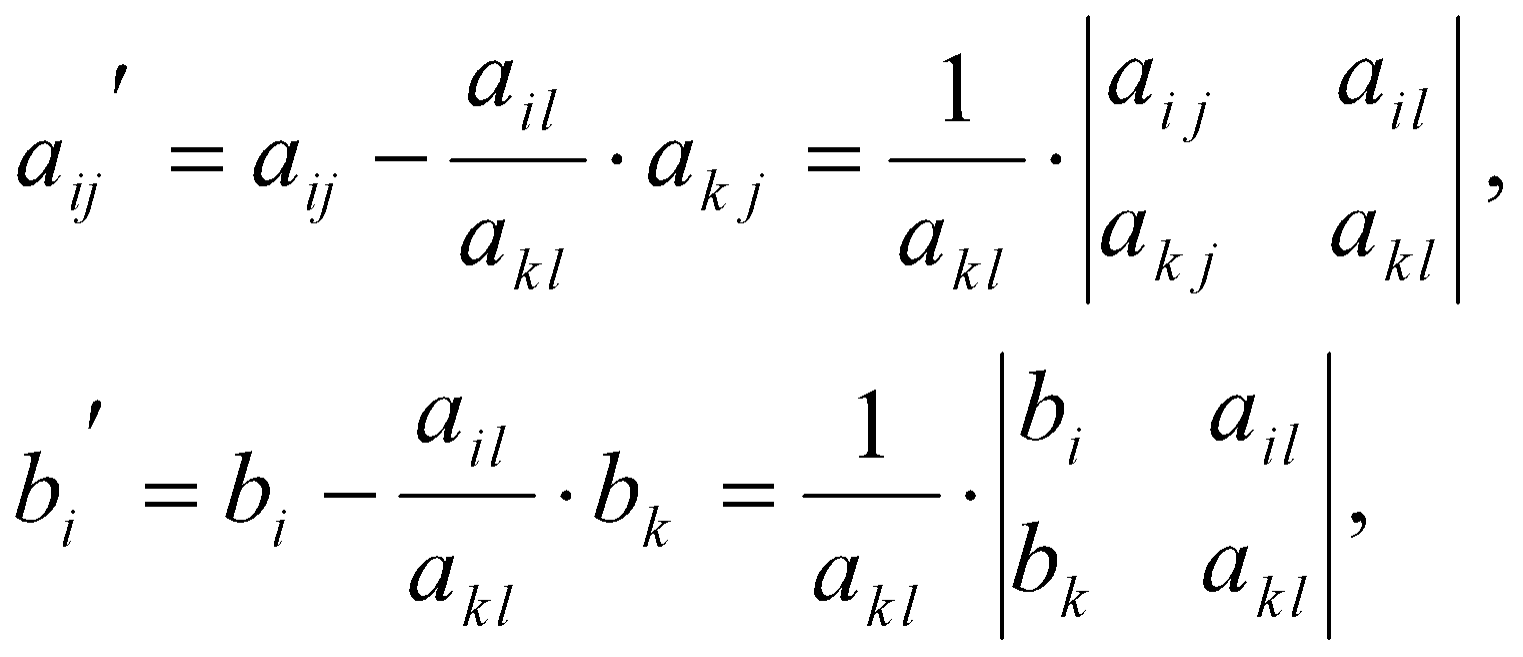
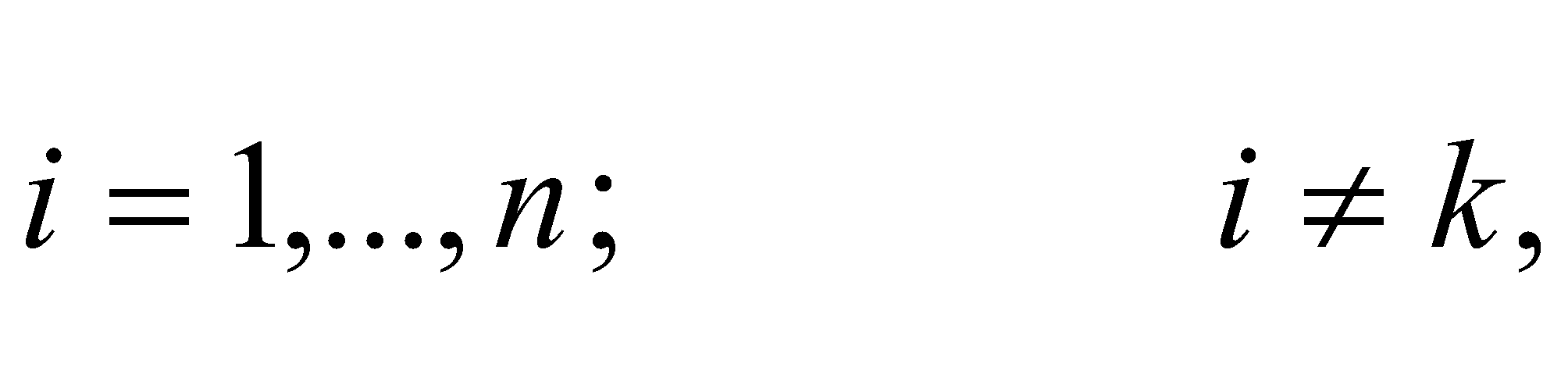
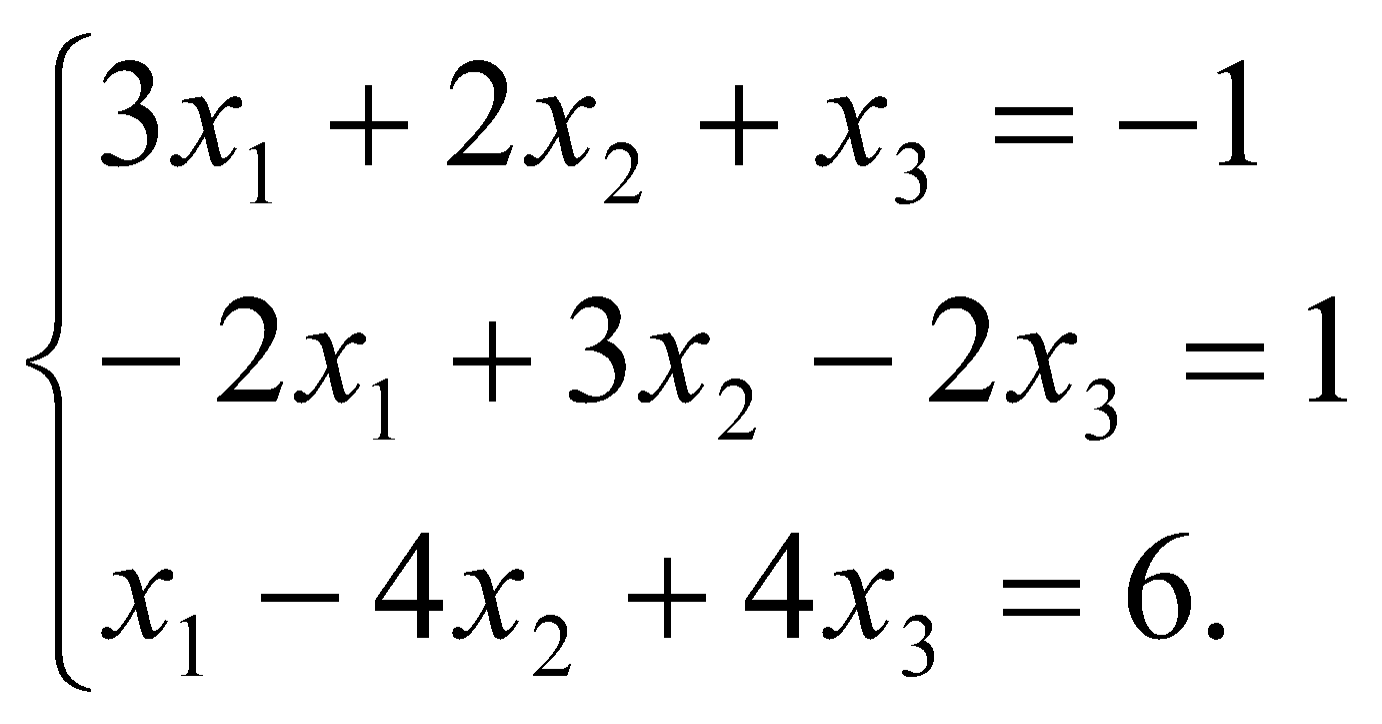
**Метод ведущего элемента**

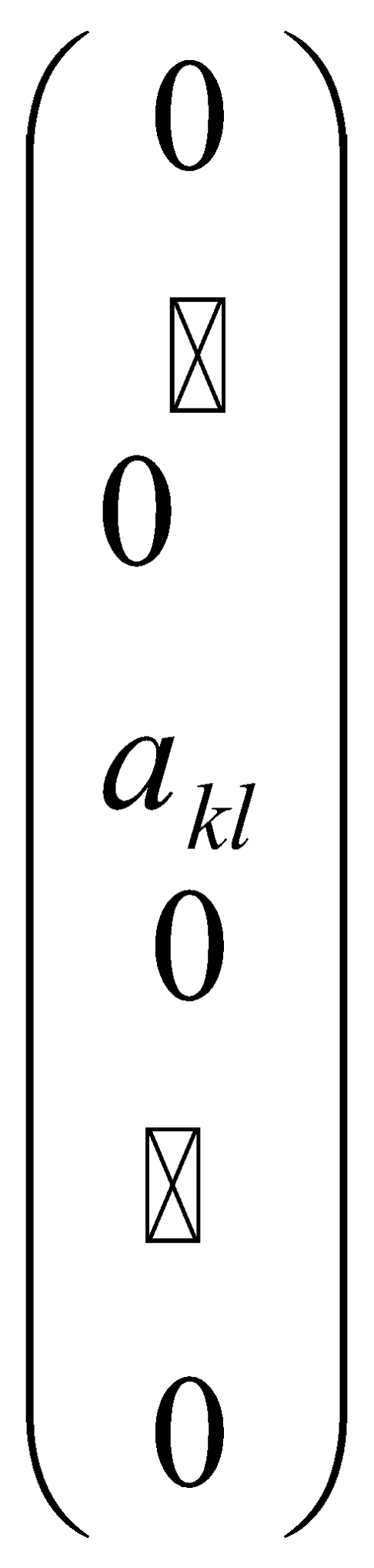
*Метод ведущего элемента* представляет собой модификацию метода Гаусса.

Рассмотрим систему уравнений (2.1) с одинаковым числом *n* неизвестных и уравнений:

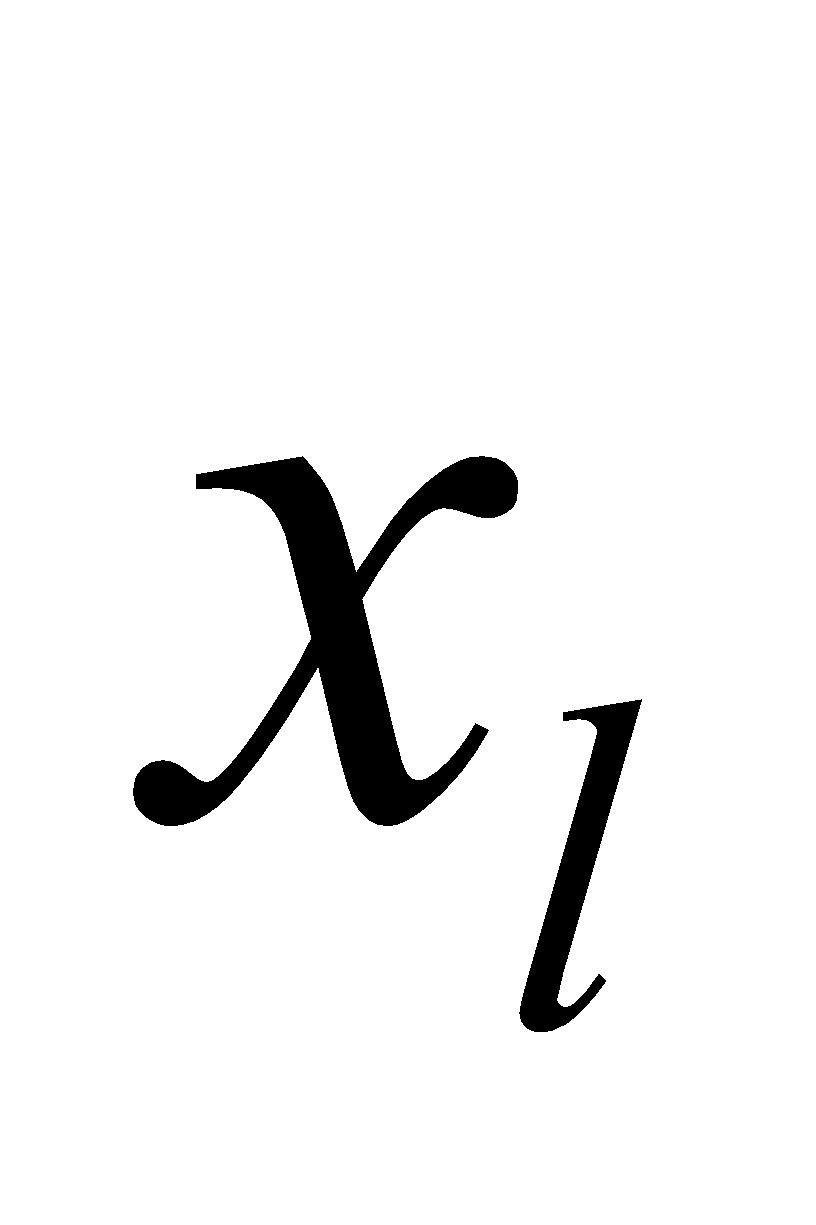
Выбираем в матрице *А* любой не равный нулю элемент , который называют *ведущим элементом*. Строка и столбец, в котором находится этот элемент, называют *рабочими*. То есть рабочими будут *l*-й

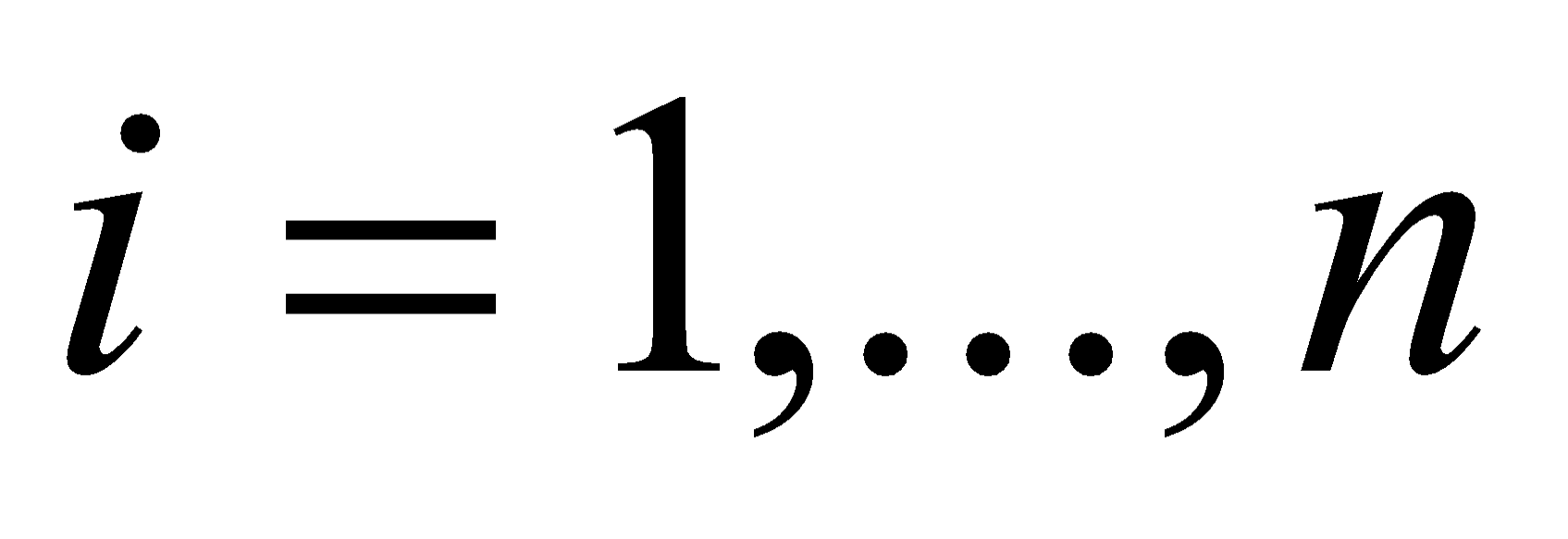
столбец и *k*-я строка. Исключаем неизвестное из всех уравнений, кроме *k-*го. В матрице новой системы *l-*й столбец принимает вид:

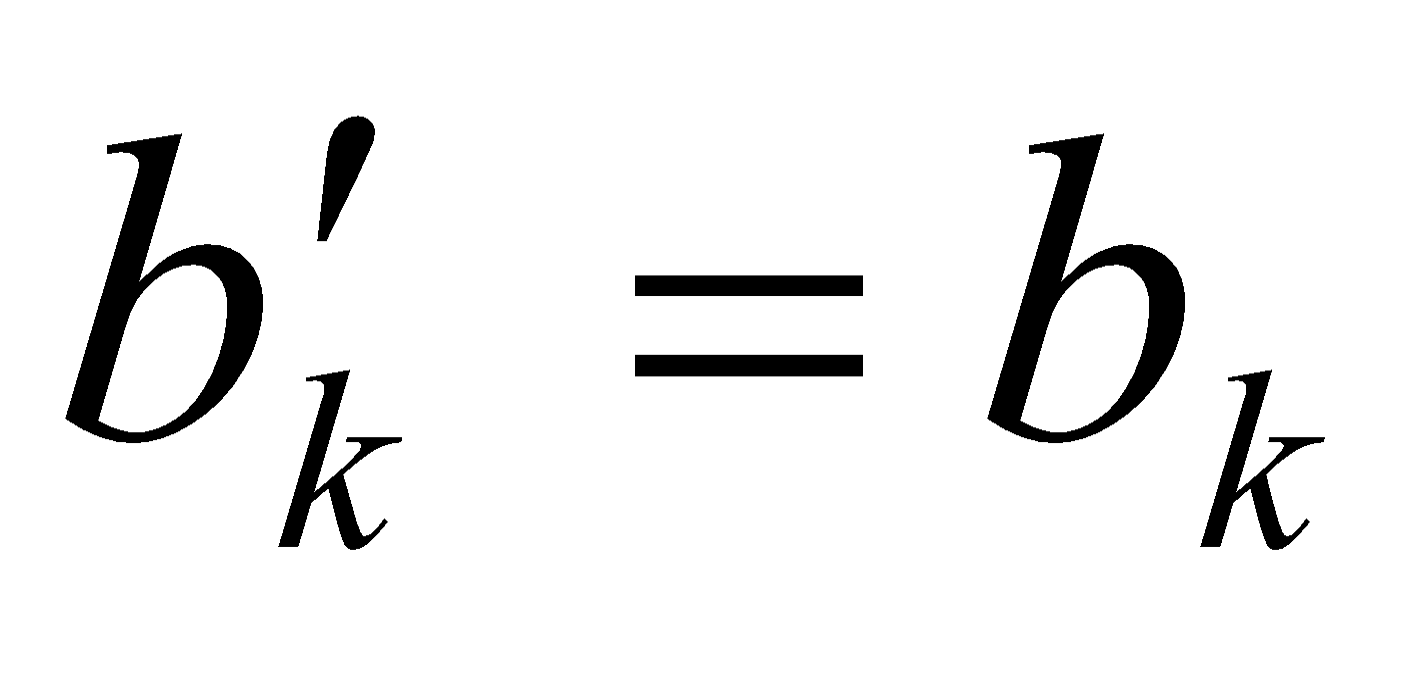
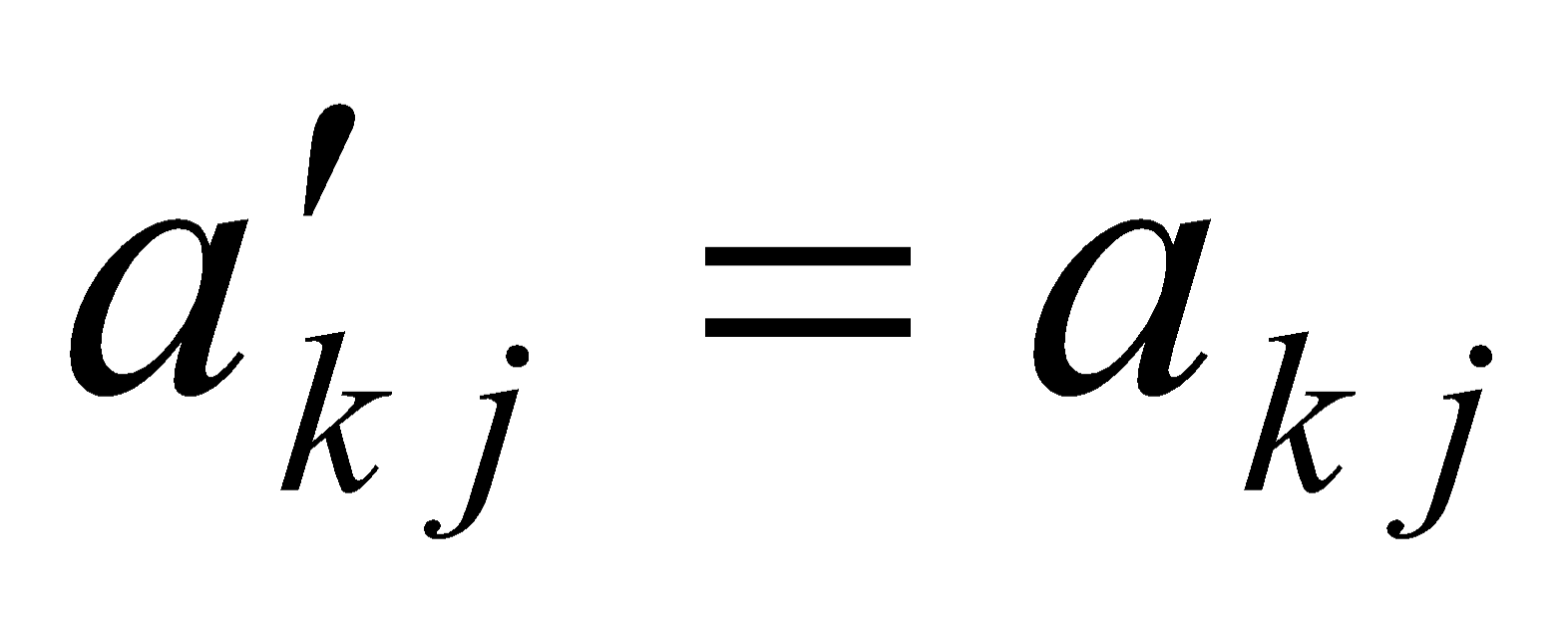
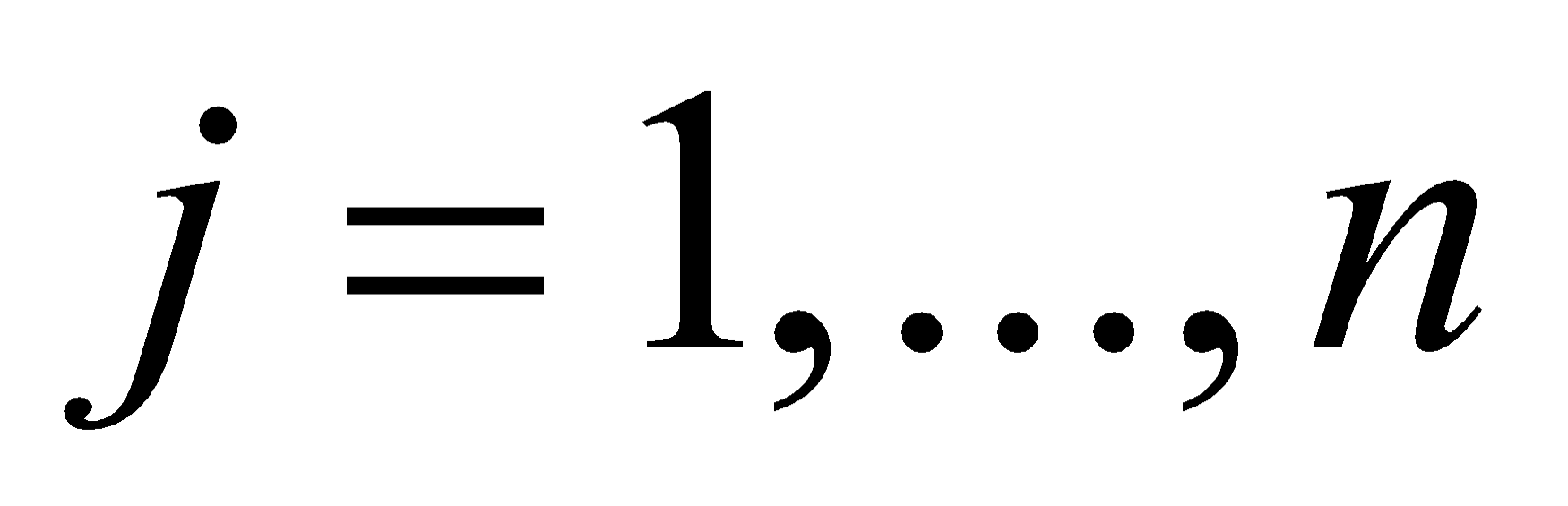


.

Помечаем рабочие строку и столбец. В последующих действиях они больше не могут являться рабочими.

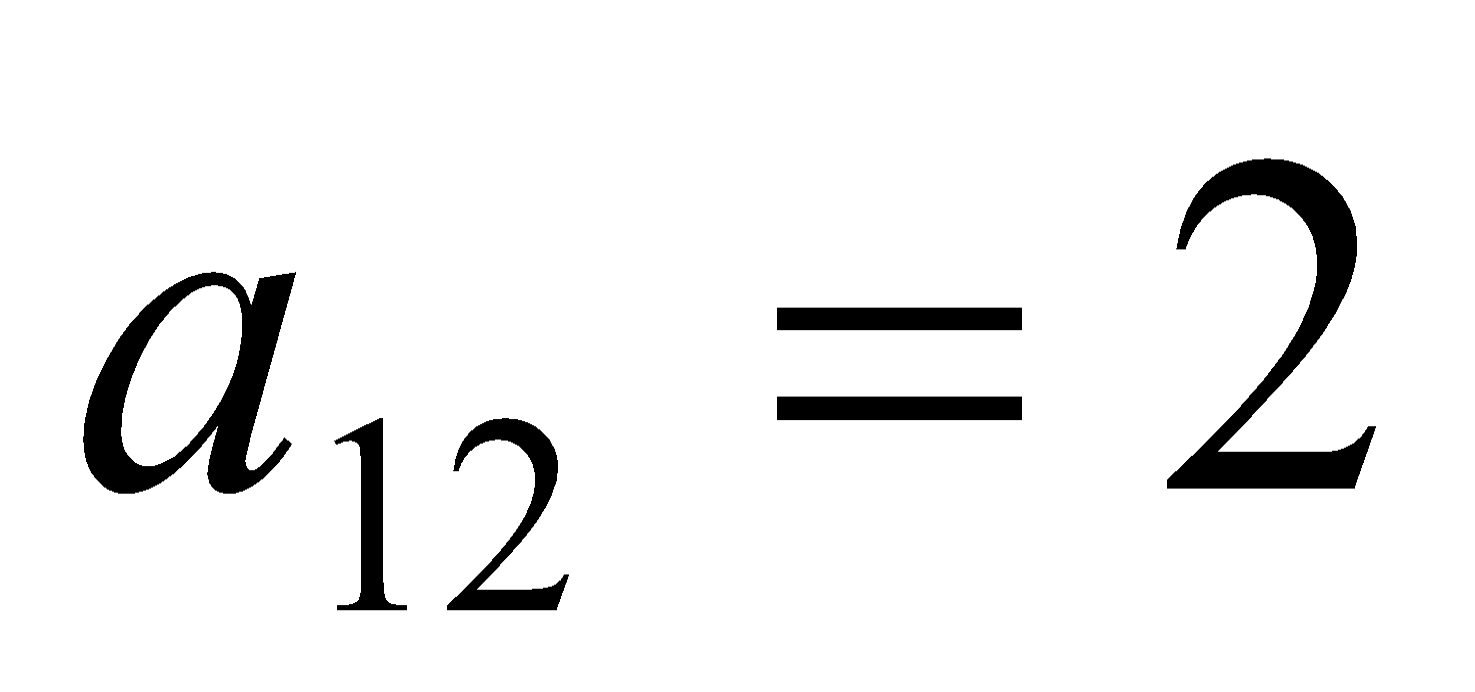
После исключения неизвестного система принимает вид

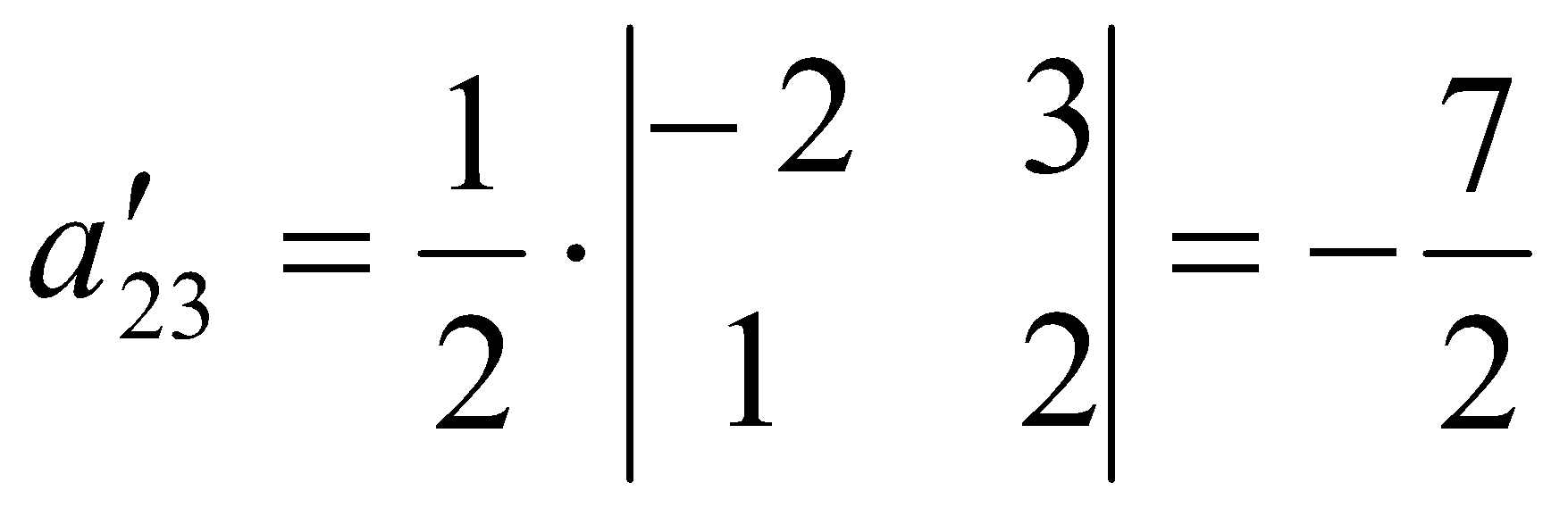
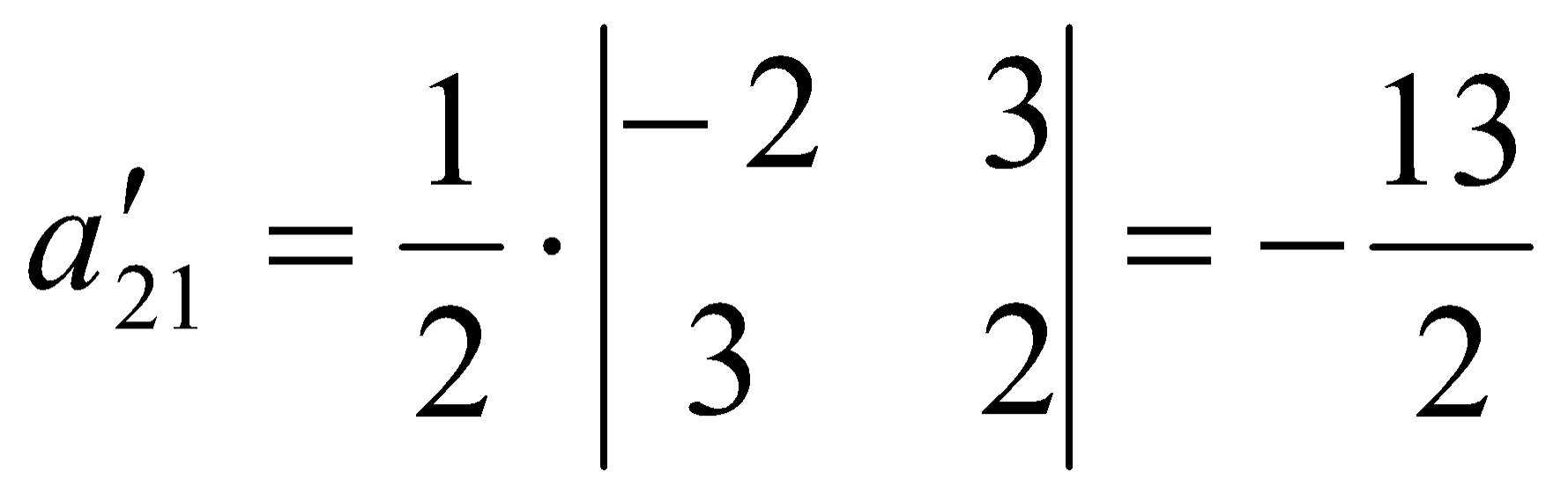
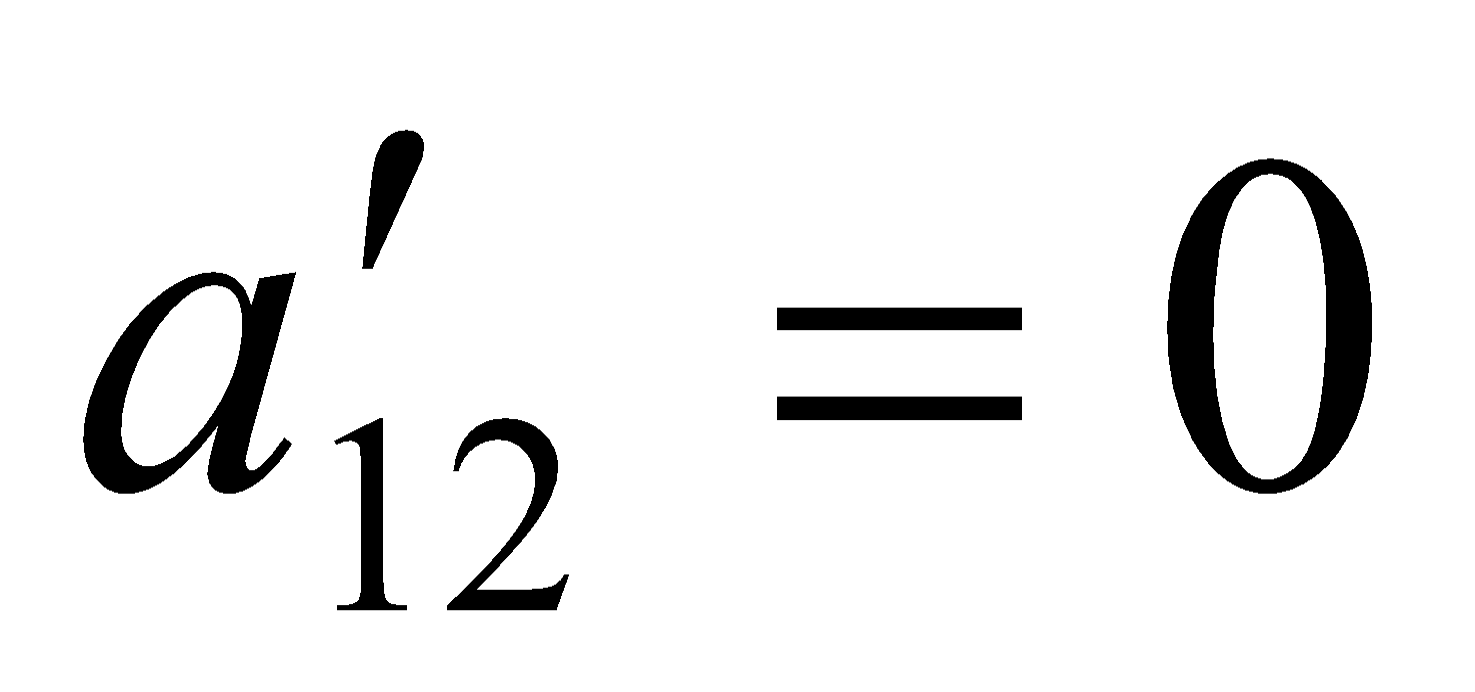
где (кроме *i=k*), т. е. коэффициенты пересчитываются по формулам:

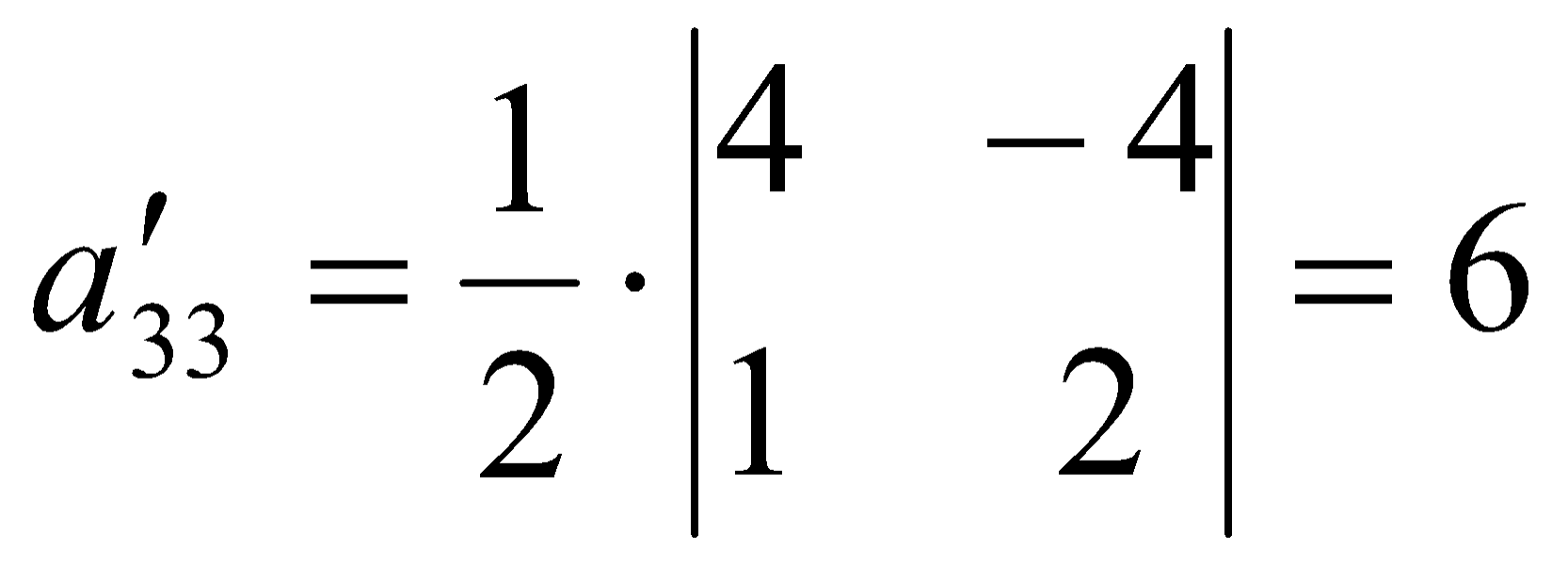
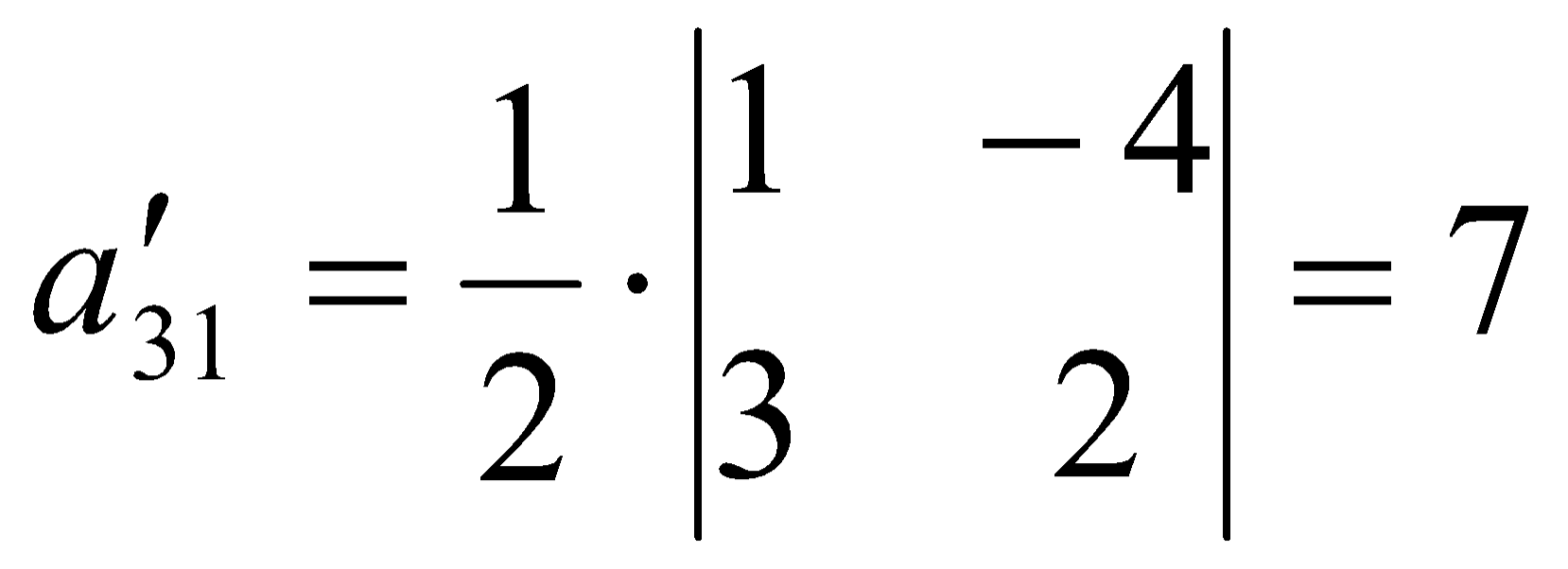
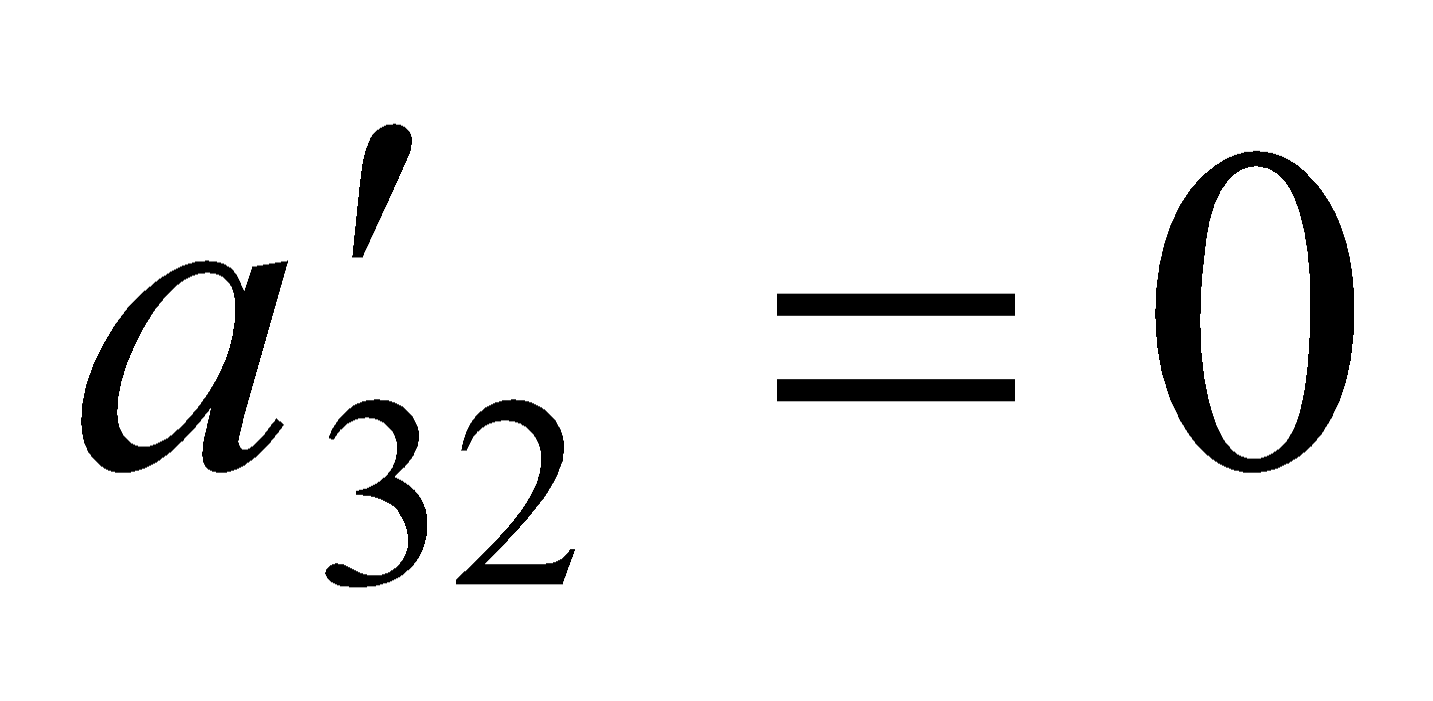
, , .

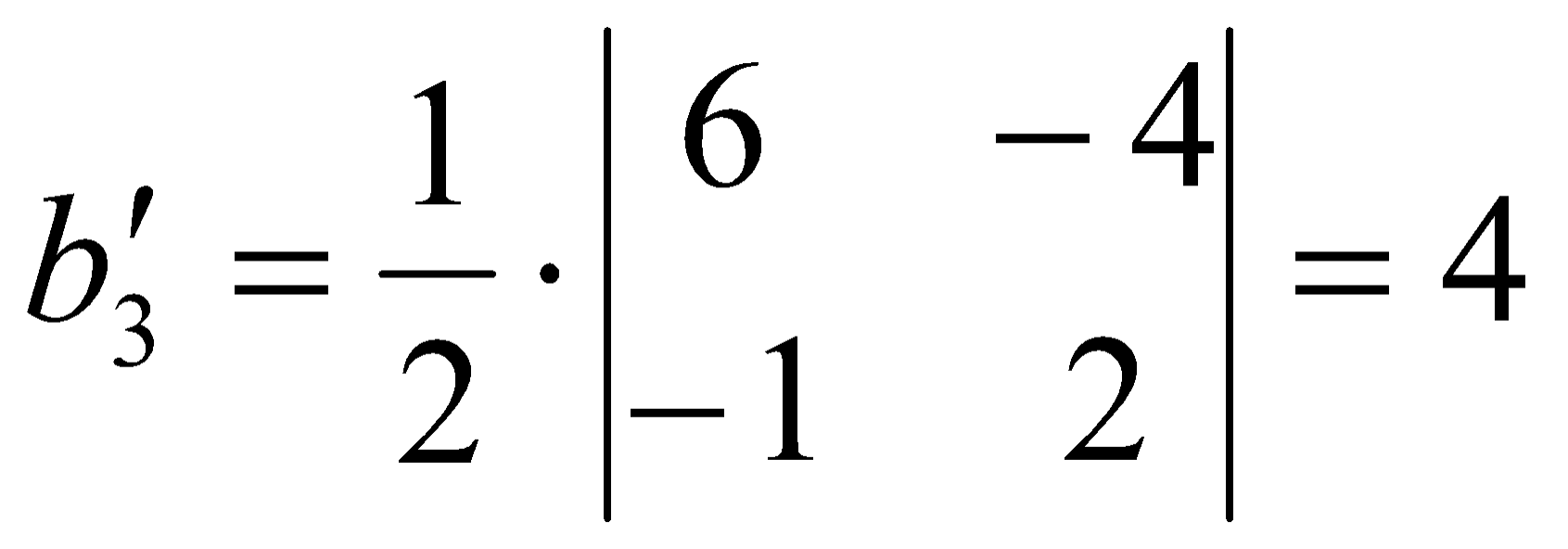
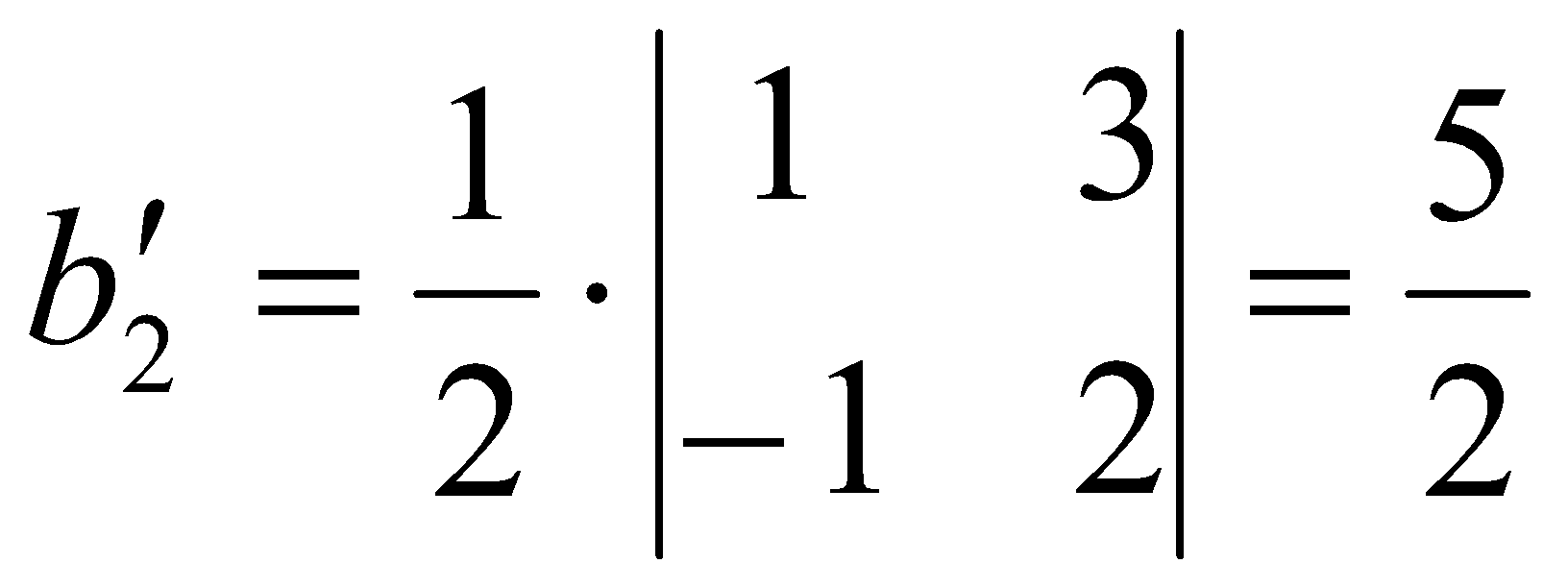
Опять ищем ведущий элемент и исключаем соответствующую переменную. Так продолжаем до тех пор, пока не получится уравнение с одним неизвестным. Из этого уравнения находим значение неизвестного и подставляем его в предыдущие уравнения для обратного хода. Продолжаем процесс, пока не получим значения всех неизвестных.

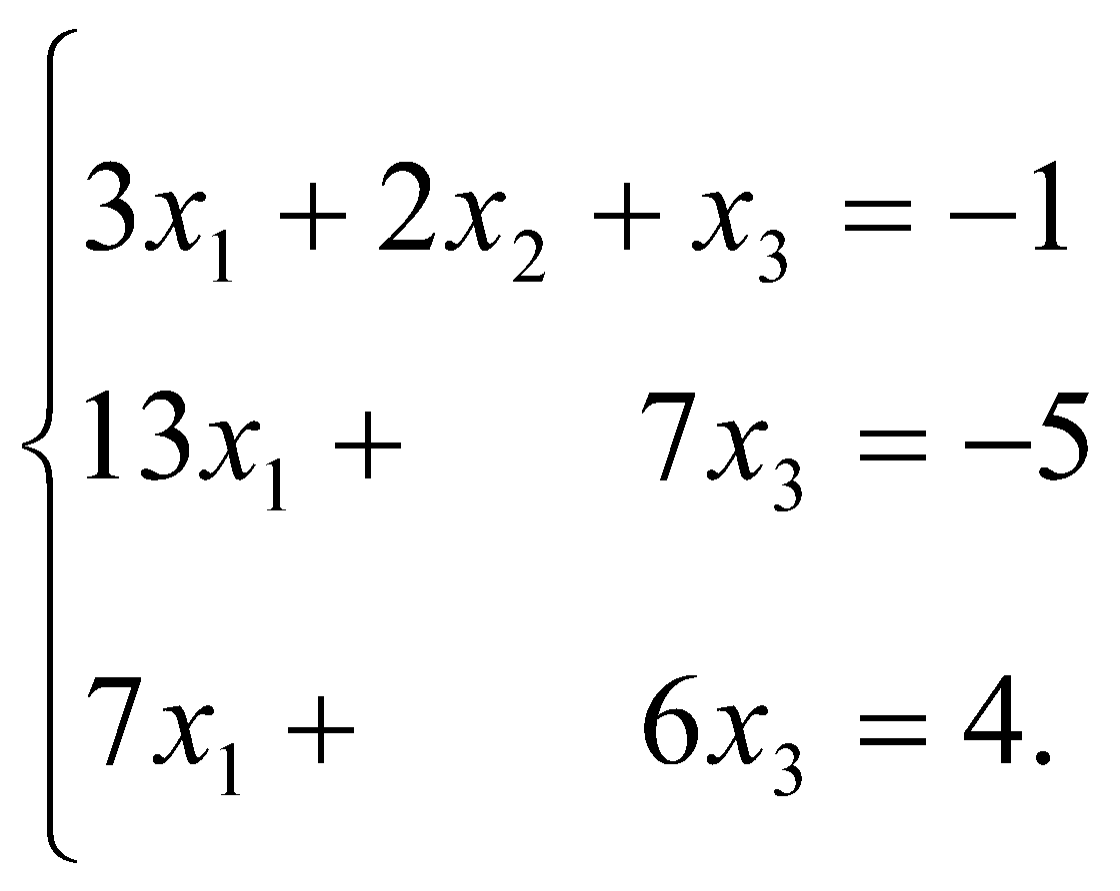
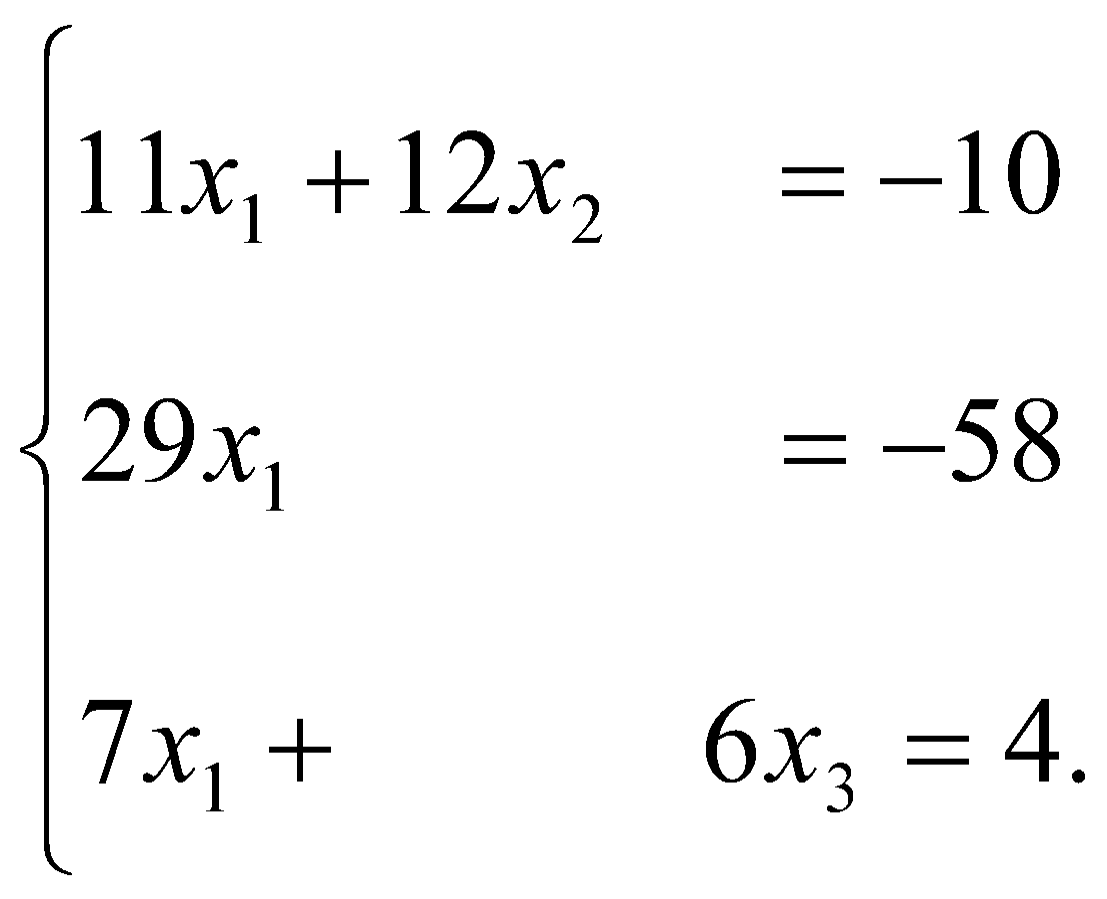
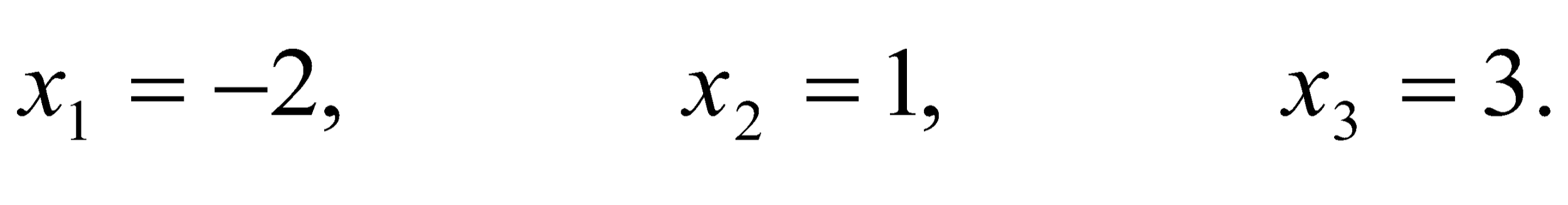
*Пример*. Пусть

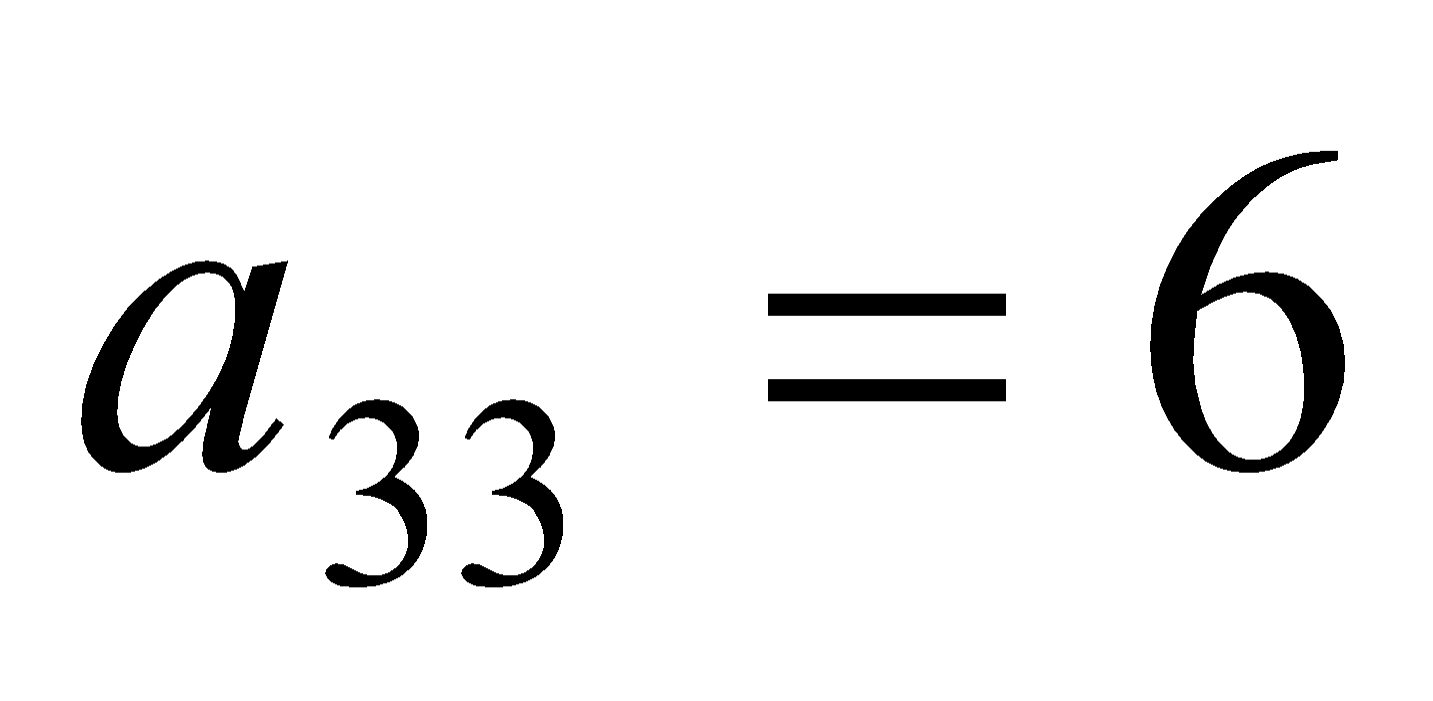
Выберем ведущим элементом . Пересчитываем коэффициенты. Находим:

, , ;

, , ;

, . Получаем:



Выбираем ведущим элементом и получаем:

Откуда

**Метод Гаусса** – **Жордана**

*Метод Гаусса* – *Жордана* представляет собой одну из разновидностей метода Гаусса.

Краткий алгоритм:

Выбирают первый слева столбец [матрицы,](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) в котором есть хоть одно отличное от нуля значение.

Если самое верхнее число в этом столбце ноль, то меняют всю первую строку матрицы с другой строкой матрицы, где в этой колонке нет нуля.

Все элементы первой строки делят на верхний элемент выбранного столбца.

Из оставшихся строк вычитают первую строку, умноженную на первый элемент соответствующей строки, с целью получить первым элементом каждой строки (кроме первой) ноль.

Далее проводят такую же процедуру с матрицей, получающейся из исходной матрицы после вычёркивания первой строки и первого столбца.

После повторения этой процедуры n-1 раз получают [верхнюю треугольную матрицу](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D1%85%D0%BD%D1%8F%D1%8F_%D1%82%D1%80%D0%B5%D1%83%D0%B3%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0)

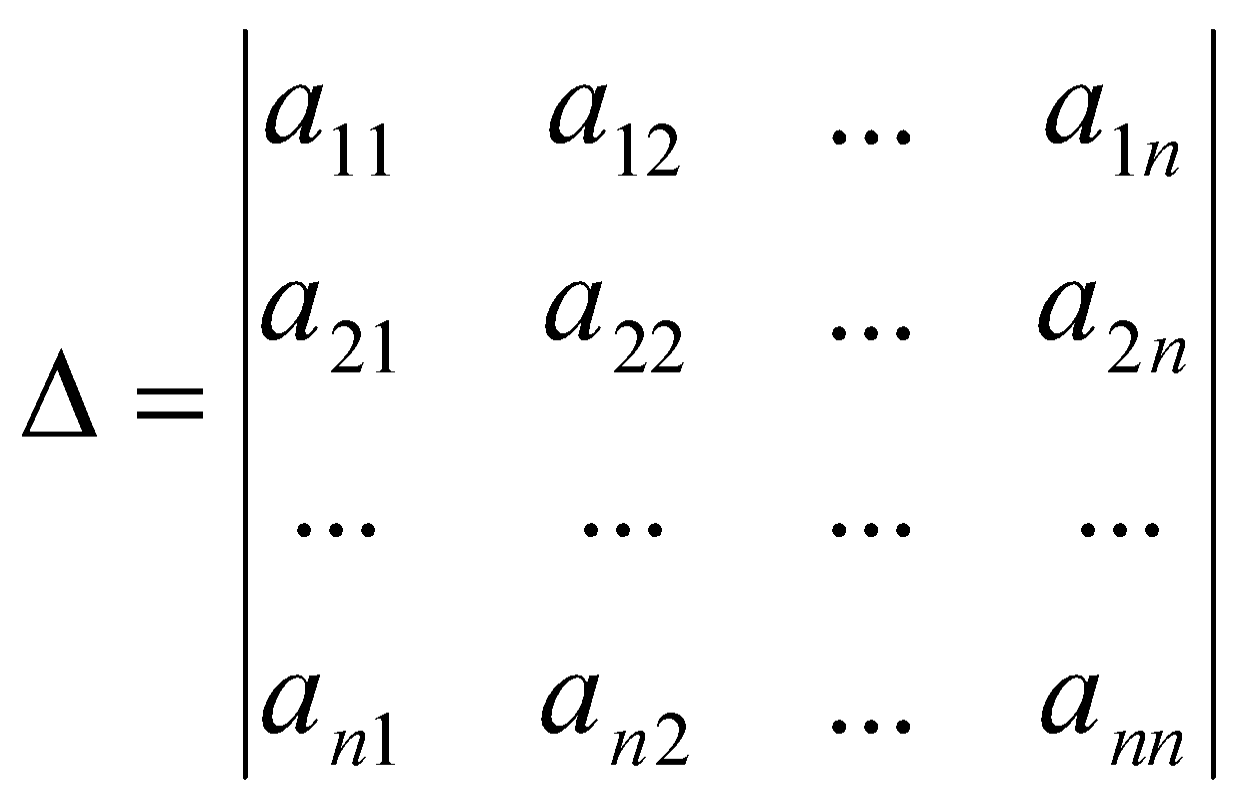
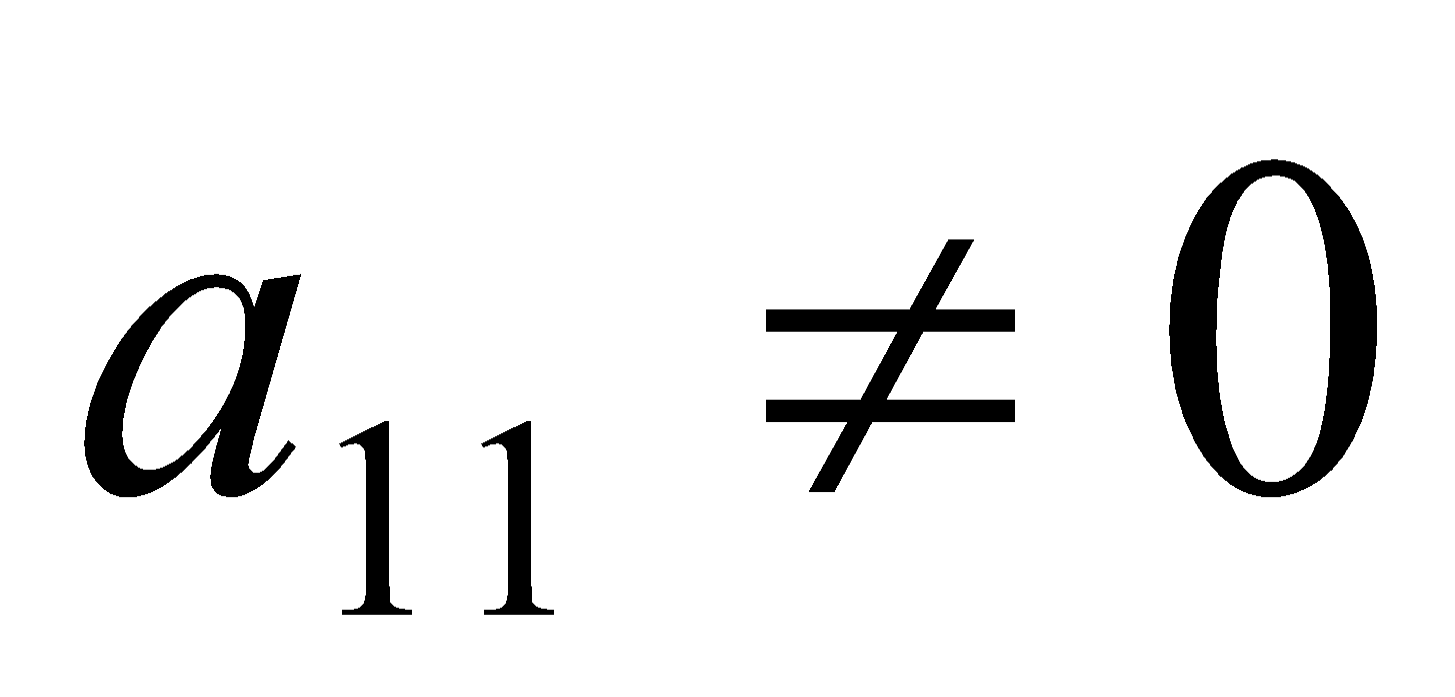
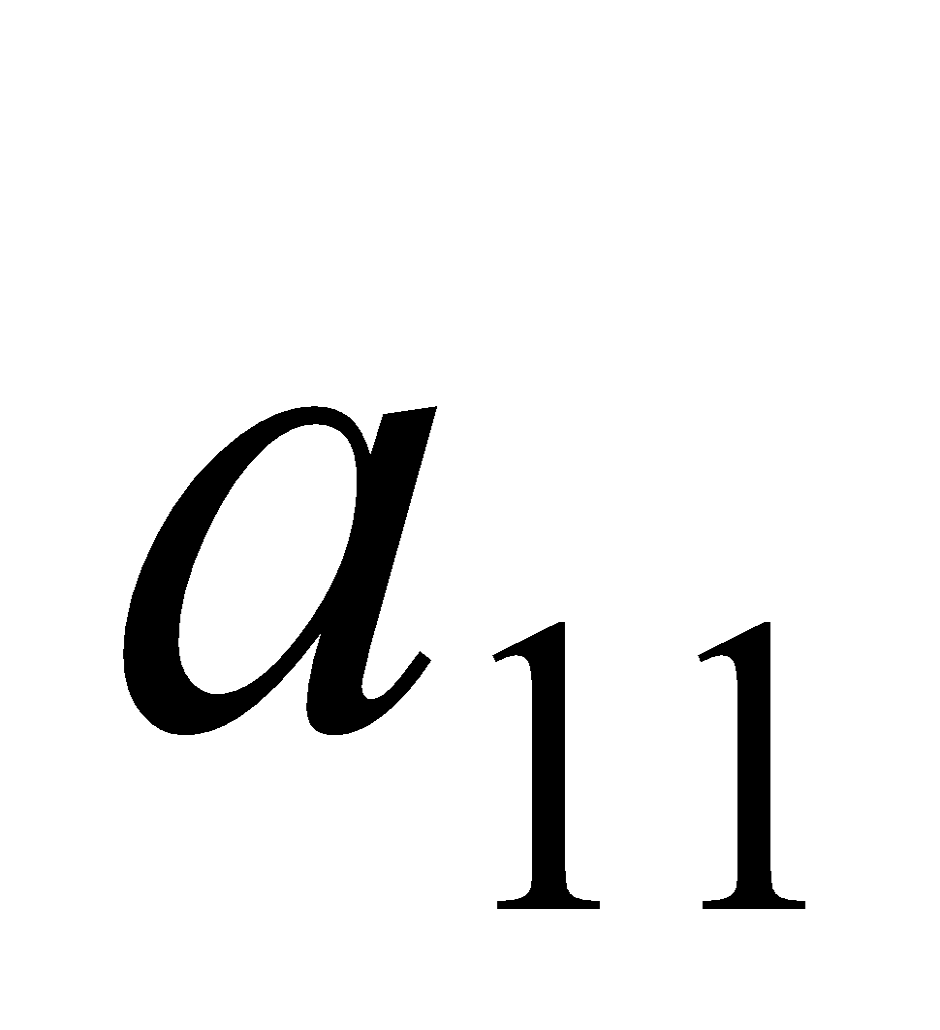
Вычитают из предпоследней строки последнюю строку, умноженную на соответствующий коэффициент, с тем, чтобы в предпоследней строке осталась только 1 на [главной диагонали](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B4%D0%B8%D0%B0%D0%B3%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C).

Повторяют предыдущий шаг для последующих строк. В итоге получают [единичную матрицу](https://ru.wikipedia.org/wiki/%D0%95%D0%B4%D0%B8%D0%BD%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0) и решение на месте свободного вектора (с ним необходимо проводить все те же преобразования).

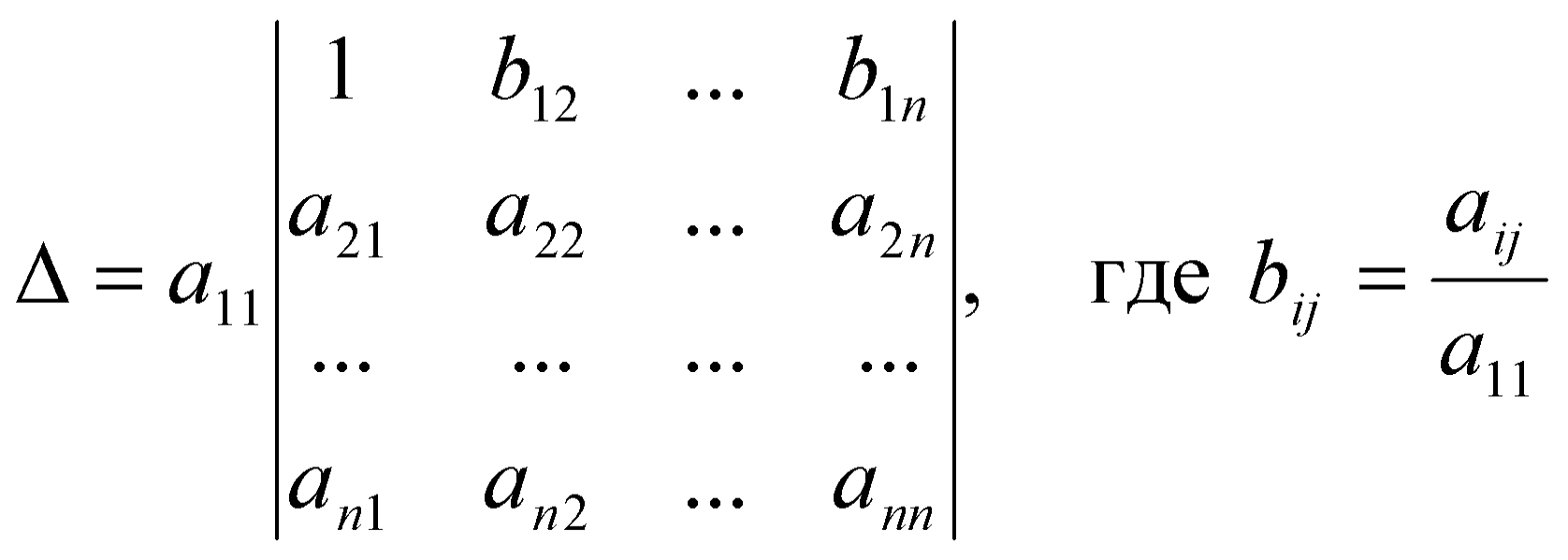
**Вычисление определителей**

Для упрощения процесса вычисления определителей пользуются формулами так называемой схемы единственного деления.

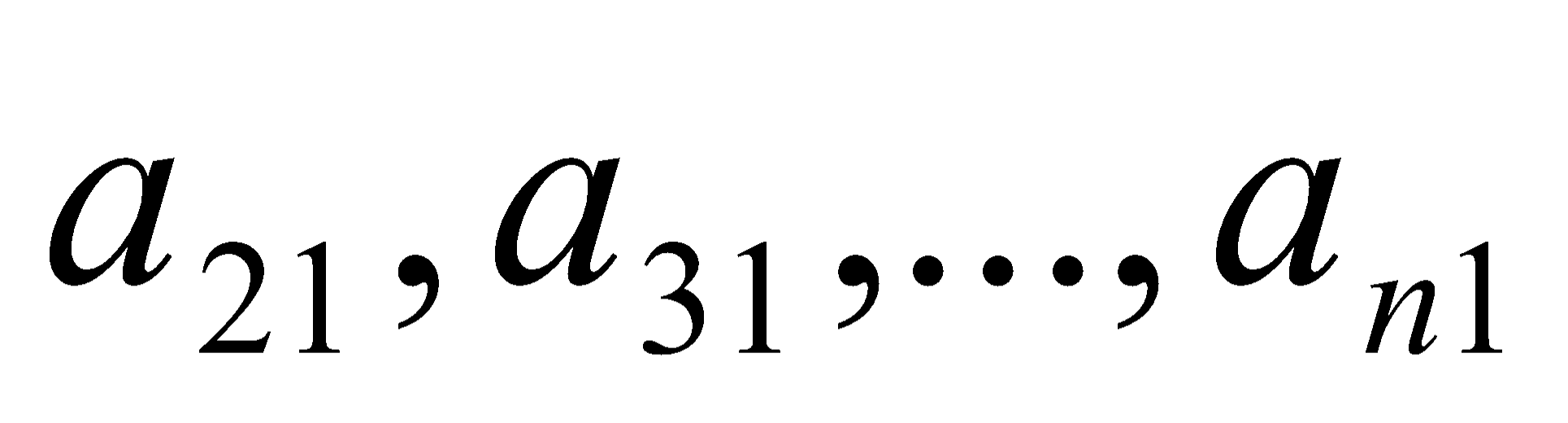
*Схема единственного деления* состоит в следующем. Пусть необходимо вычислить определитель

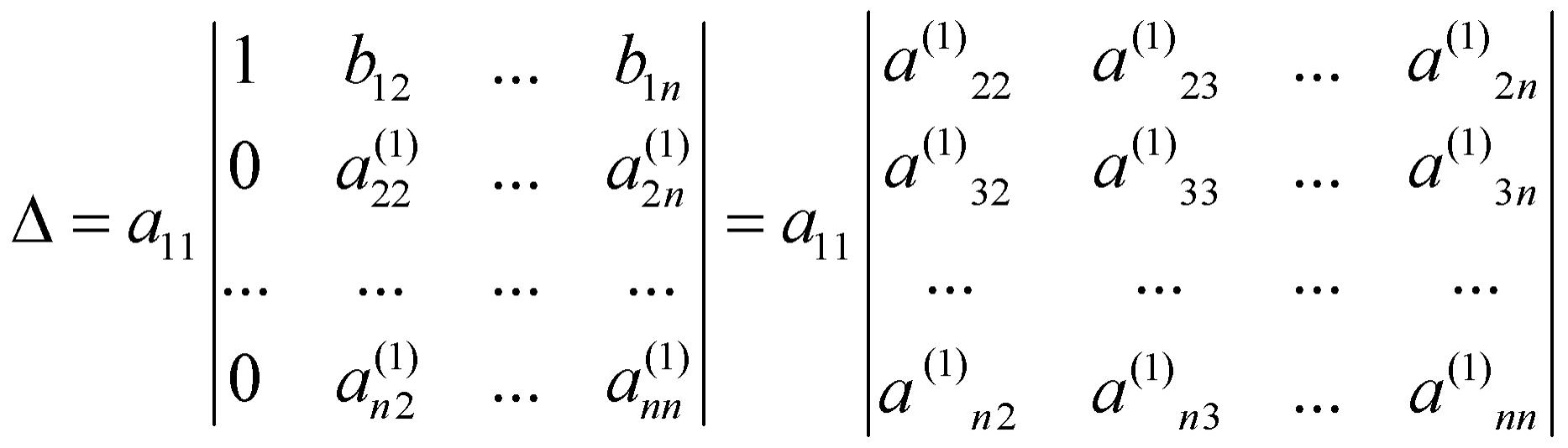
, причем .

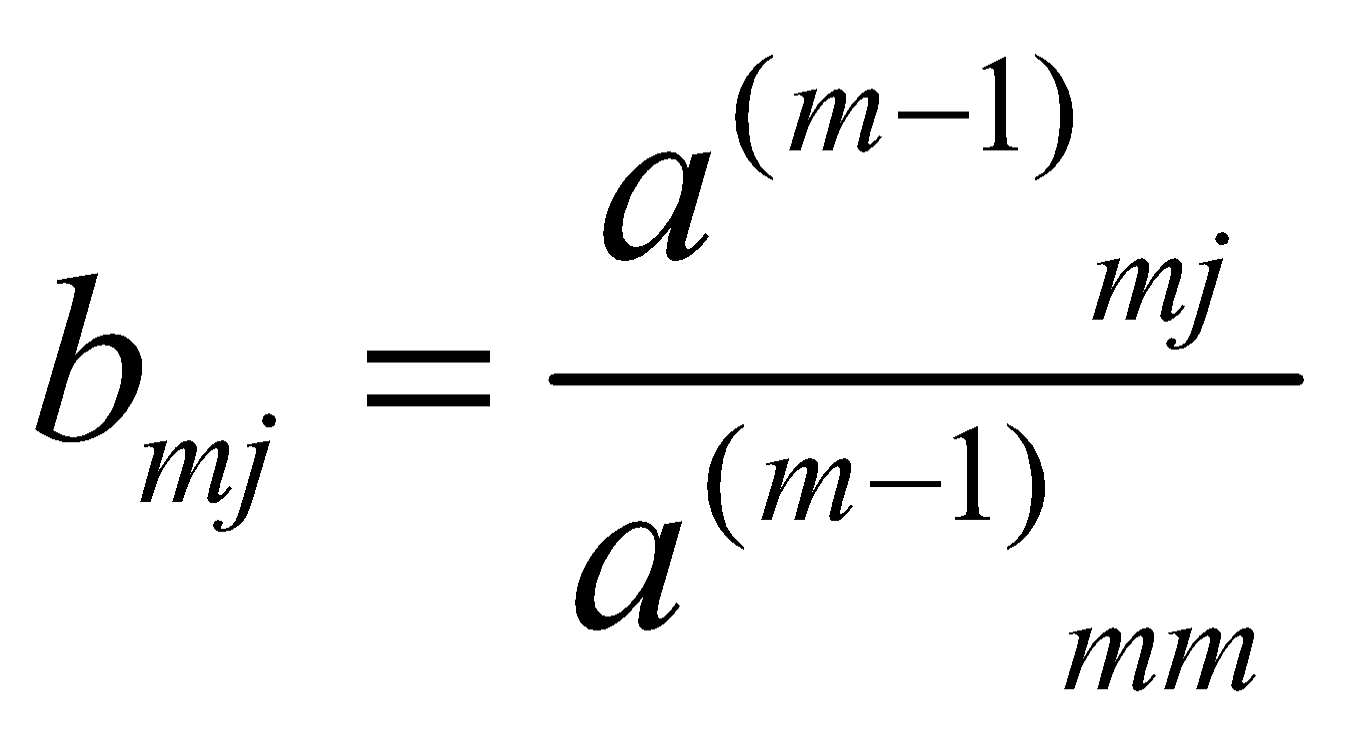
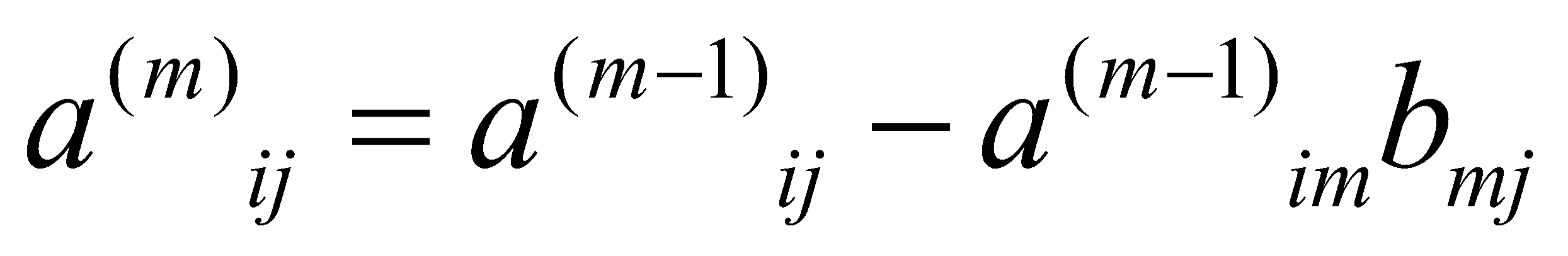
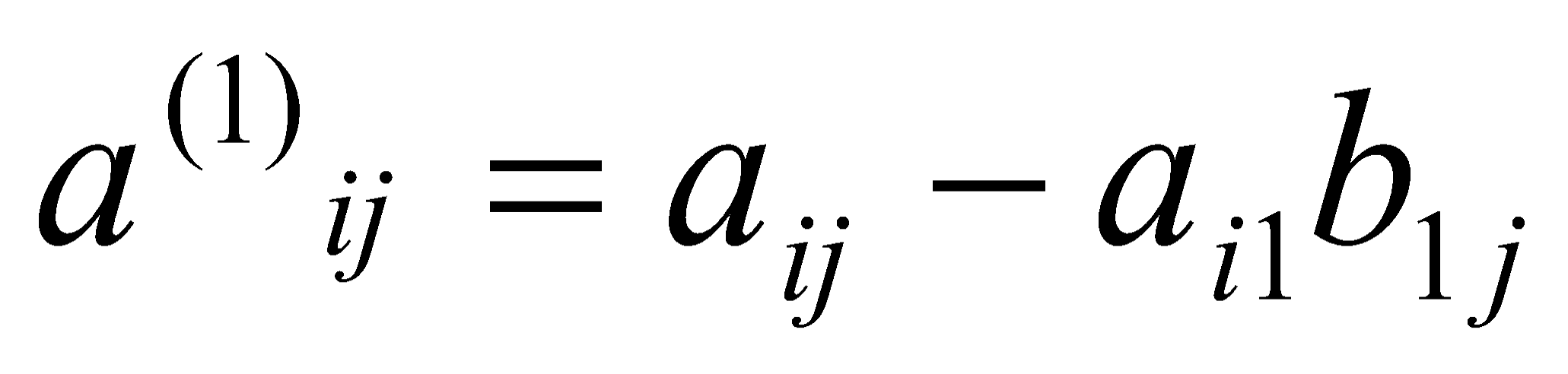
Выносим элемент из первой строки и получим

.

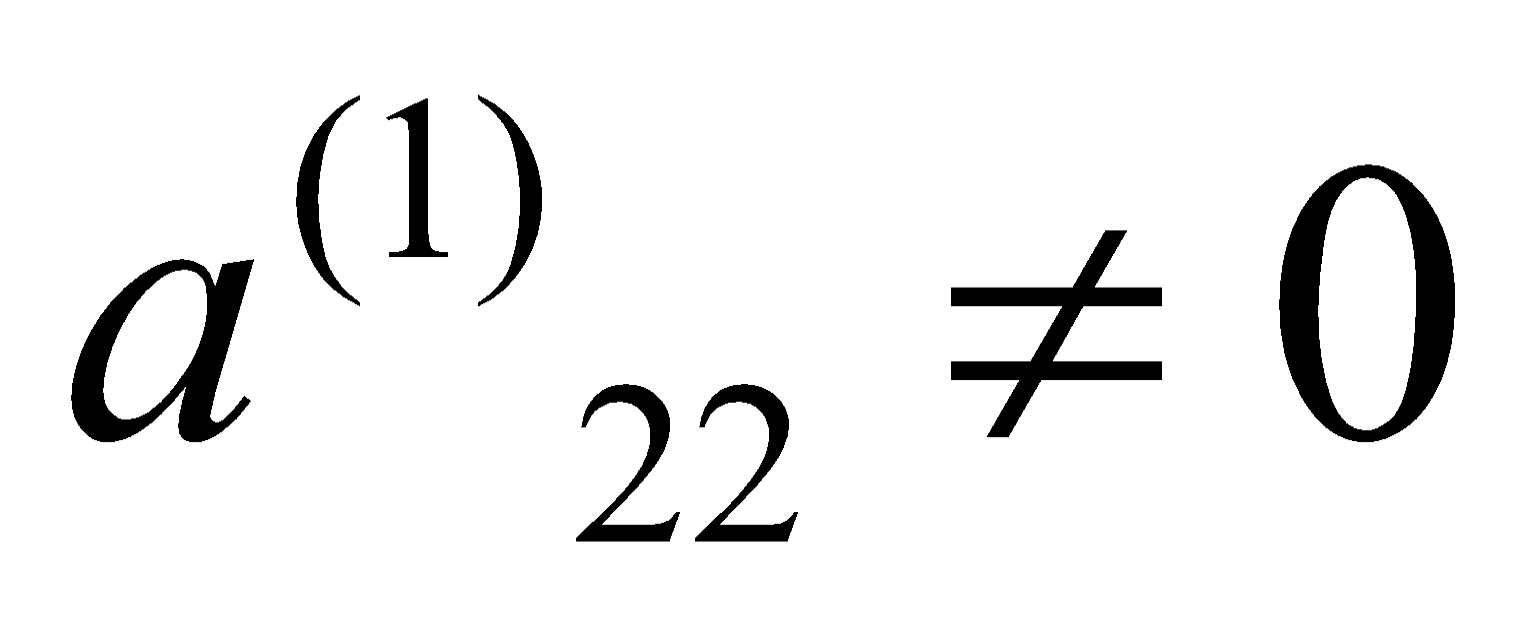
Вычитая из каждой строки, начиная со второй, первую строку,

умноженную соответственно на , получим

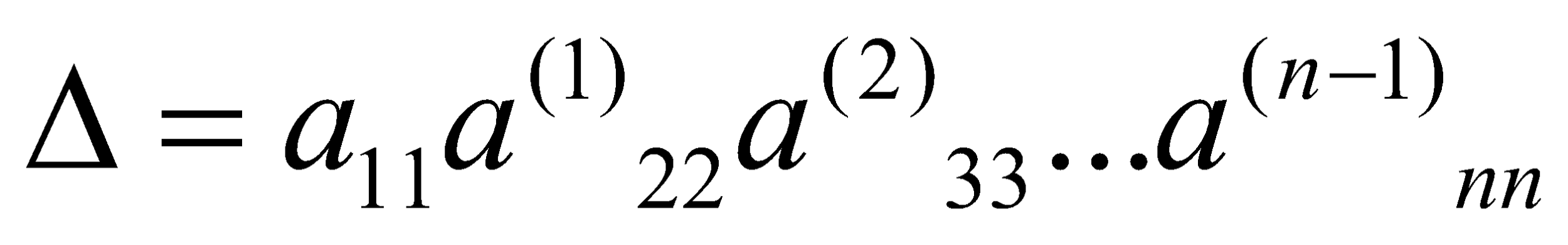
,

где и соответственно , .

С образовавшимся определителем (*n–*1)-го порядка поступаем таким

же образом, если . В противном случае перенумеруем и поменяем местами строки или столбцы определителя, при этом его знак изменится на противоположный.

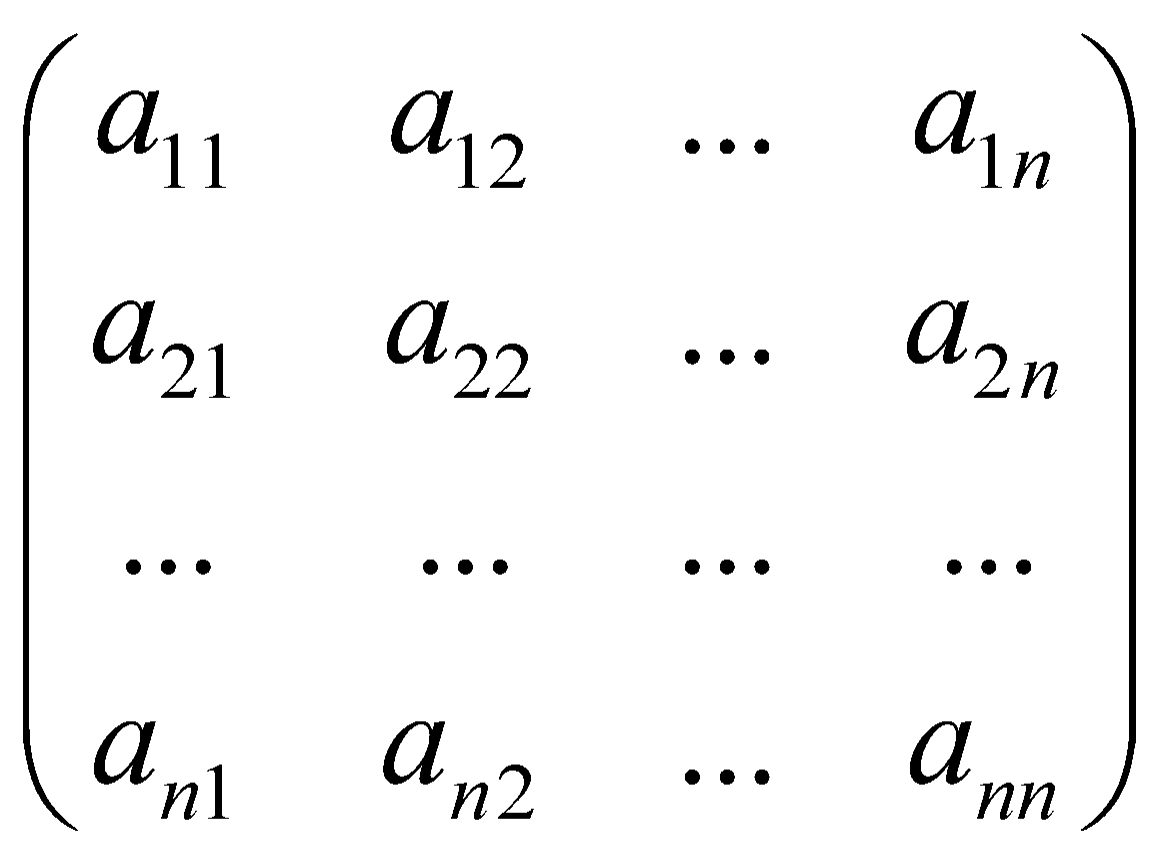
Продолжая процесс, получим, что искомый определитель равен произведению ведущих элементов:

.

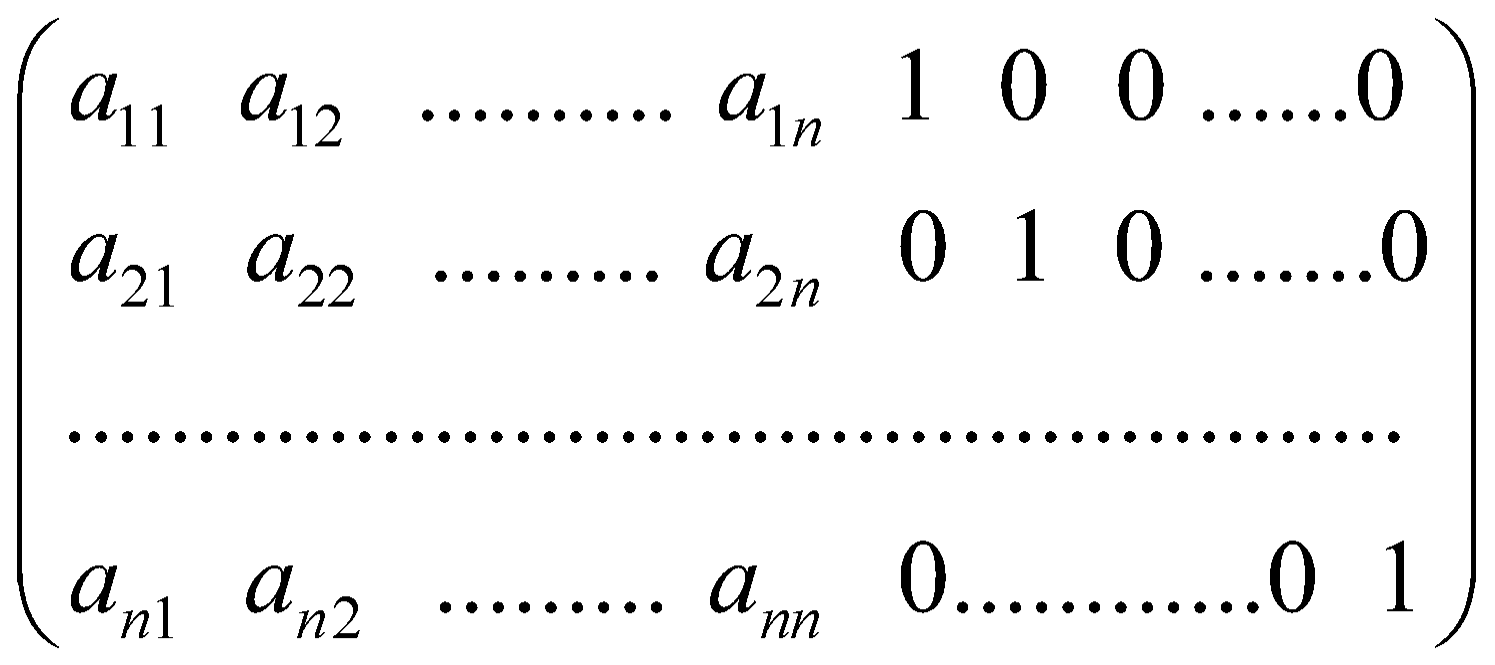
**Обращение матриц**

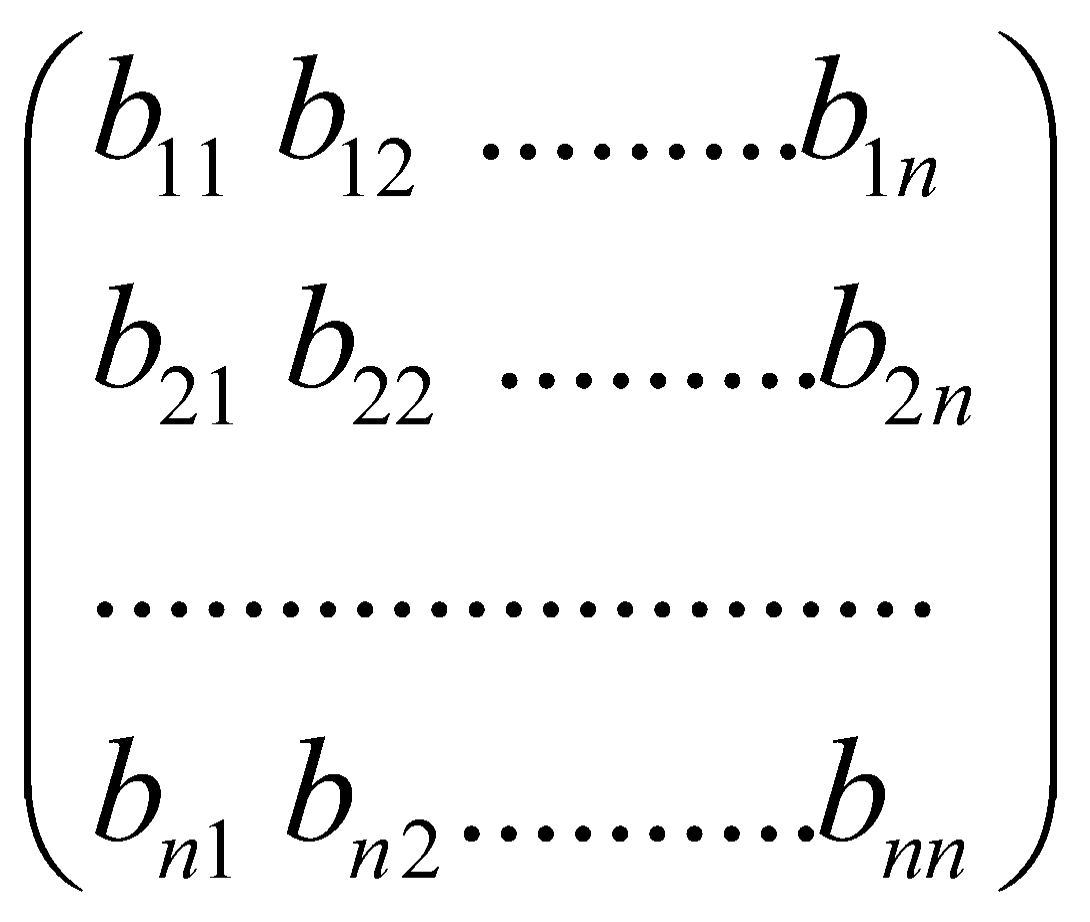
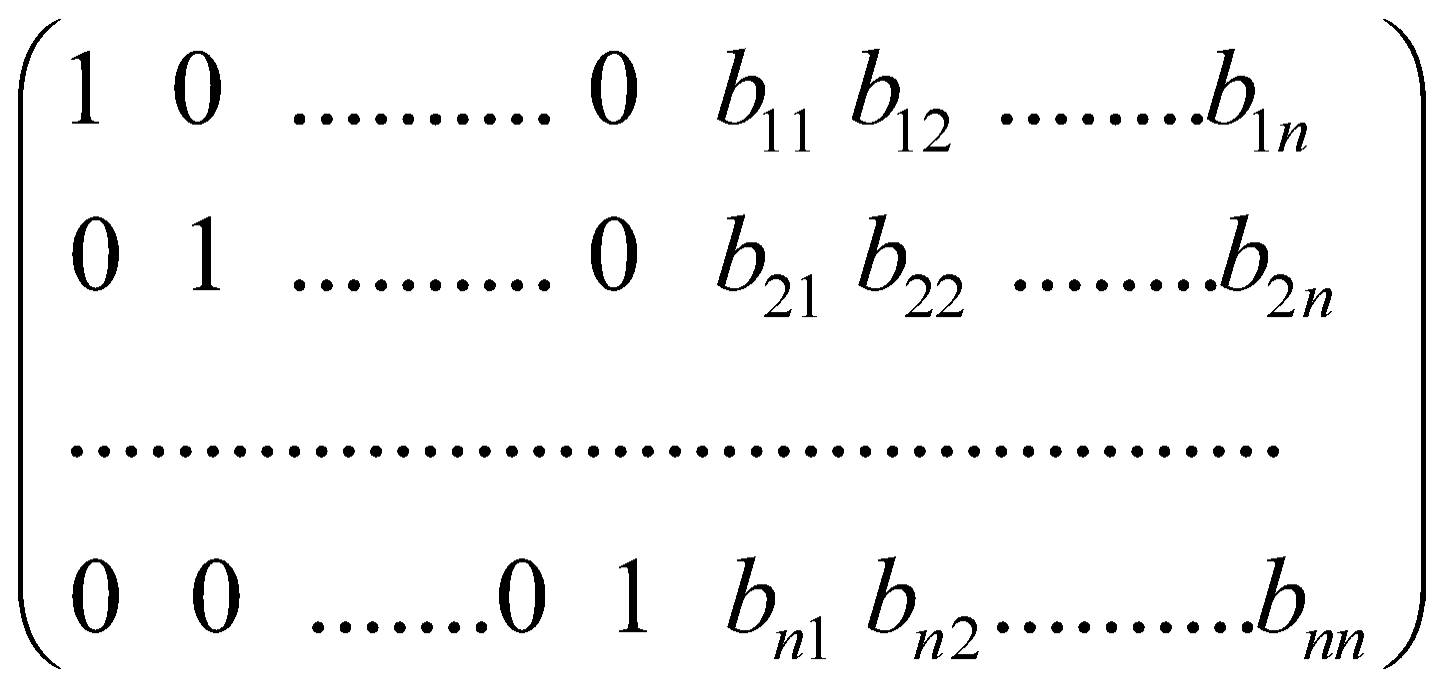
Часто возникает необходимость обратить квадратную матрицу

*A*= .



Поскольку вычисление матрицы *А*-1 обычным способом достаточно трудоемко, поступаем следующим образом. Построим прямоугольную матрицу:

,



которая получается, если к *A* приписать справа единичную матрицу *Е*. Применим к этой матрице метод исключения Гаусса – Жордана, не

заботясь о правых столбцах. По окончании исключения получим

.

Если матрица *А* не вырождена, то матрица

*В*=

будет обратной к *A*. Чтобы убедиться в этом, заметим, что каждый шаг процесса исключения эквивалентен умножению слева на некоторую матрицу. Произведение всех этих левых матриц есть некоторая матрица *С*, умножение на которую слева приводит *А* к единичной матрице *Е*, то есть *СА=Е*, а будучи применено к правым столбцам, это произведение делает из единичной матрицы матрицу *В*: *СЕ=В*. В таком случае из существования обратной матрицы *A*-1 следует, что *С= A*-1 и *С=В*, следовательно, *В= A*-1.

**3. Программная реализация**

import numpy as np

import sys

def GaussElimination(num, augmented\_matrix, A, b):

solution\_vector = np.zeros(num)

try:

np.linalg.solve(A.copy(), b.copy())

except:

return "no solutions or infinity amount of solutions"

for i in range(num):

if augmented\_matrix[i][i] == 0.0:

return 'Divide by zero detected!'

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

ratio = augmented\_matrix[j][i] / augmented\_matrix[i][i]

for k in range(num + 1):

augmented\_matrix[j][k] = augmented\_matrix[j][k] - ratio \* augmented\_matrix[i][k]

global UpperTriangleMatrix

UpperTriangleMatrix = augmented\_matrix

# back substitution

solution\_vector[num - 1] = augmented\_matrix[num - 1][num] / augmented\_matrix[num - 1][num - 1]

for i in range(num - 2, -1, -1):

solution\_vector[i] = augmented\_matrix[i][num]

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

solution\_vector[i] = solution\_vector[i] - augmented\_matrix[i][j] \* solution\_vector[j]

solution\_vector[i] = solution\_vector[i] / augmented\_matrix[i][i]

return solution\_vector

def UpperTriangleMatrix(num, augmented\_matrix, A, b):

try:

np.linalg.solve(A.copy(), b.copy())

except:

return "no solutions or infinity amount of solutions"

for i in range(num):

if augmented\_matrix[i][i] == 0.0:

return 'Divide by zero detected!'

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

ratio = augmented\_matrix[j][i] / augmented\_matrix[i][i]

for k in range(num + 1):

augmented\_matrix[j][k] = augmented\_matrix[j][k] - ratio \* augmented\_matrix[i][k]

print("Upper triangle matrix:")

print(augmented\_matrix)

return augmented\_matrix

def GaussEliminationWithPivoting(a, vector\_b, A, b):

n = len(b)

# array for solution vector

solution\_vector = np.zeros(n)

try:

np.linalg.solve(A.copy(), b.copy())

except:

return "no solutions or infinity amount of solutions"

# first loop specifies the fixed row

for k in range(n - 1):

if np.fabs(a[k, k]) < 1.0e-12:

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

if np.fabs(a[i, k]) > np.fabs(a[k, k]):

a[[k, i]] = a[[i, k]]

vector\_b[[k, i]] = vector\_b[[i, k]]

break

# applies the elimination below the fixed row

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

if a[i, k] == 0:

continue

factor = a[k, k] / a[i, k]

for j in range(k, n):

a[i, j] = a[k, j] - a[i, j] \* factor

# we also calculate the b vector of each row

vector\_b[i] = vector\_b[k] - vector\_b[i] \* factor

# print(a)

# print(b)

solution\_vector[n - 1] = vector\_b[n - 1] / a[n - 1, n - 1]

for i in range(n - 2, -1, -1):

sum\_ax = 0

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

sum\_ax += a[i, j] \* solution\_vector[j]

solution\_vector[i] = (vector\_b[i] - sum\_ax) / a[i, i]

return solution\_vector

def GaussJordanElimination(num, augmented\_matrix, A, b):

# array for solution vector

solution\_vector = np.zeros(num)

try:

np.linalg.solve(A.copy(), b.copy())

except:

return "no solutions or infinity amount of solutions"

for i in range(num):

if augmented\_matrix[i][i] == 0.0:

return 'Divide by zero detected!'

for j in range(num):

if i != j:

ratio = augmented\_matrix[j][i] / augmented\_matrix[i][i]

for k in range(num + 1):

augmented\_matrix[j][k] = augmented\_matrix[j][k] - ratio \* augmented\_matrix[i][k]

# Obtaining Solution

for i in range(num):

solution\_vector[i] = augmented\_matrix[i][num] / augmented\_matrix[i][i]

return solution\_vector

def CalculateDeterminant(matrix):

if matrix.shape[0] == 2:

return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]

determinant = 0

for i in range(matrix.shape[0]):

temp\_matrix = matrix[1:, :].copy()

temp\_matrix = np.delete(temp\_matrix, i, 1)

determinant = determinant + ((-1) \*\* i) \* matrix[0][i] \* CalculateDeterminant(temp\_matrix)

return determinant

def InvertMatrix(num, matrix, A):

if np.linalg.det(A.copy()) == 0:

return "zero determinant -- can't invert matrix"

else:

for i in range(num):

if matrix[i][i] == 0.0:

return 'Divide by zero detected!'

for j in range(num):

if i != j:

ratio = matrix[j][i] / matrix[i][i]

for k in range(2 \* num):

matrix[j][k] = matrix[j][k] - ratio \* matrix[i][k]

# row operation to make principal diagonal element to 1

for i in range(num):

divisor = matrix[i][i]

for j in range(2 \* num):

matrix[i][j] = matrix[i][j] / divisor

inverted\_matrix = np.zeros((num, num))

for i in range(num):

for j in range(num, 2 \* num):

inverted\_matrix[i][j - num] += matrix[i][j]

return inverted\_matrix

def MultiplyMatrix(inverted\_A, vector\_b, A):

if np.linalg.det(A.copy()) == 0:

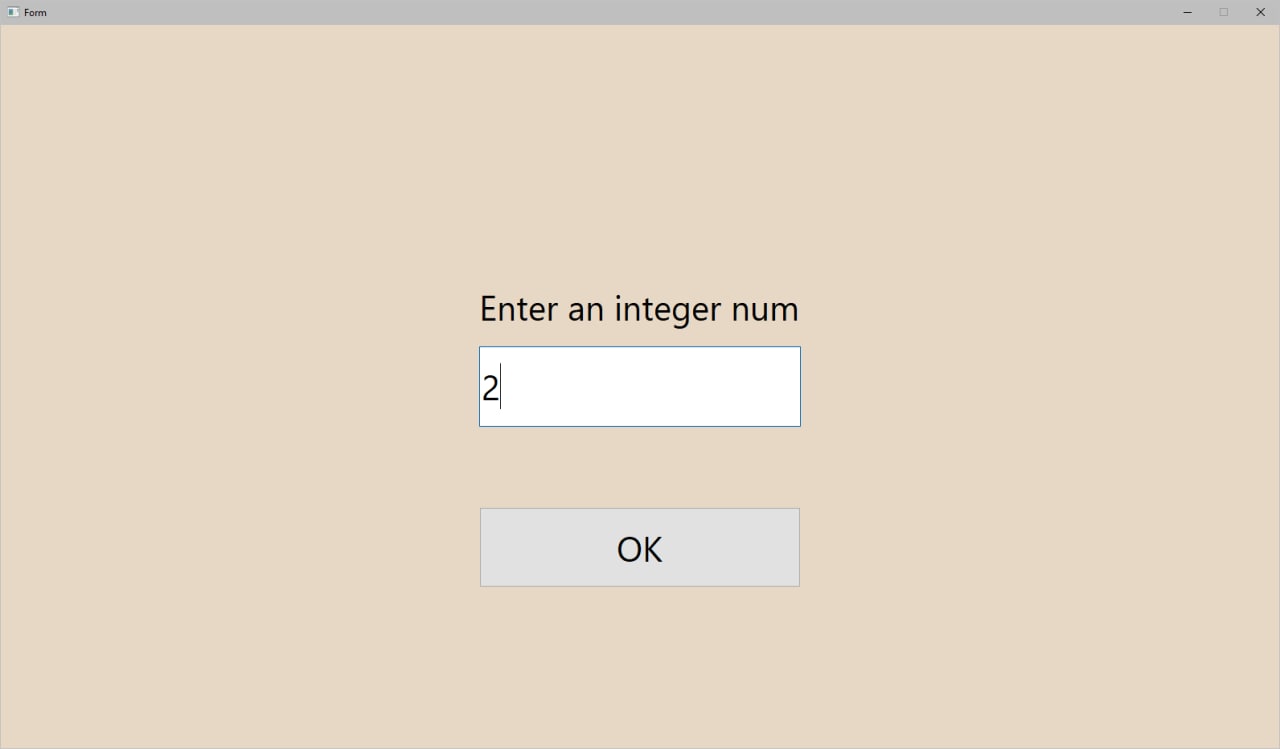
return "zero determinant -- can't invert matrix"

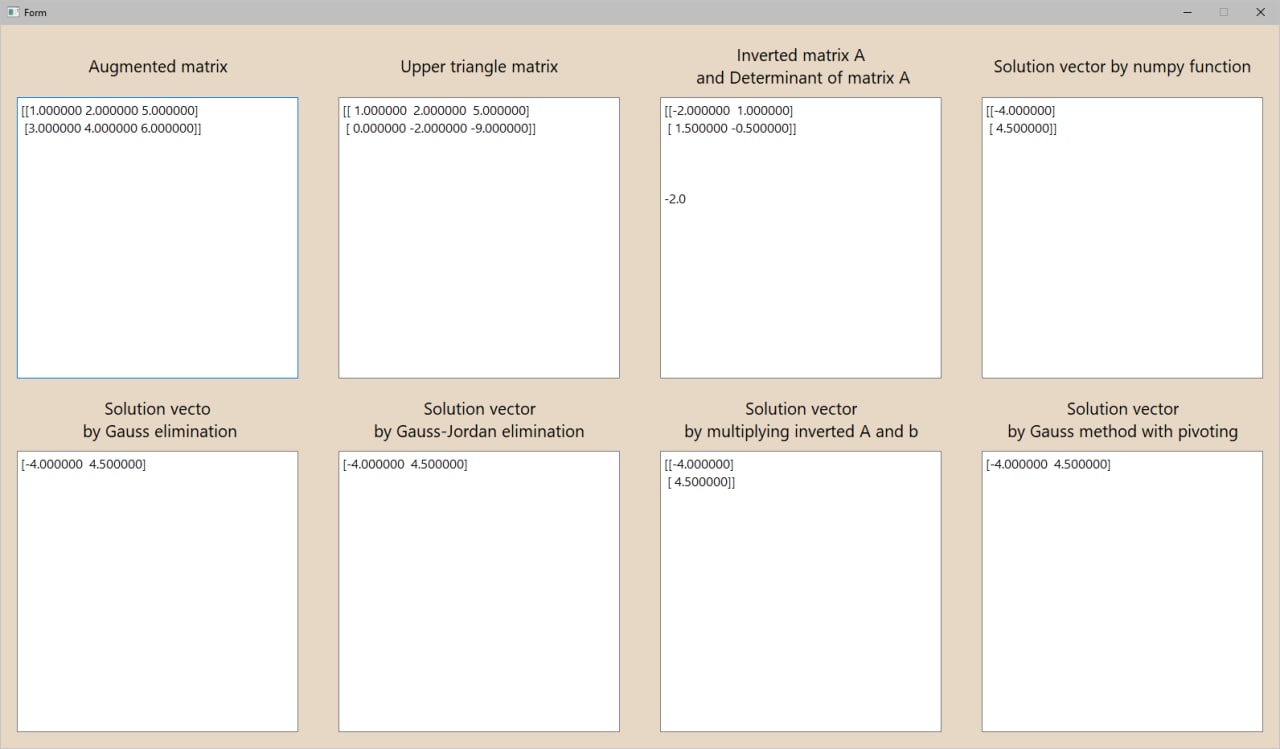
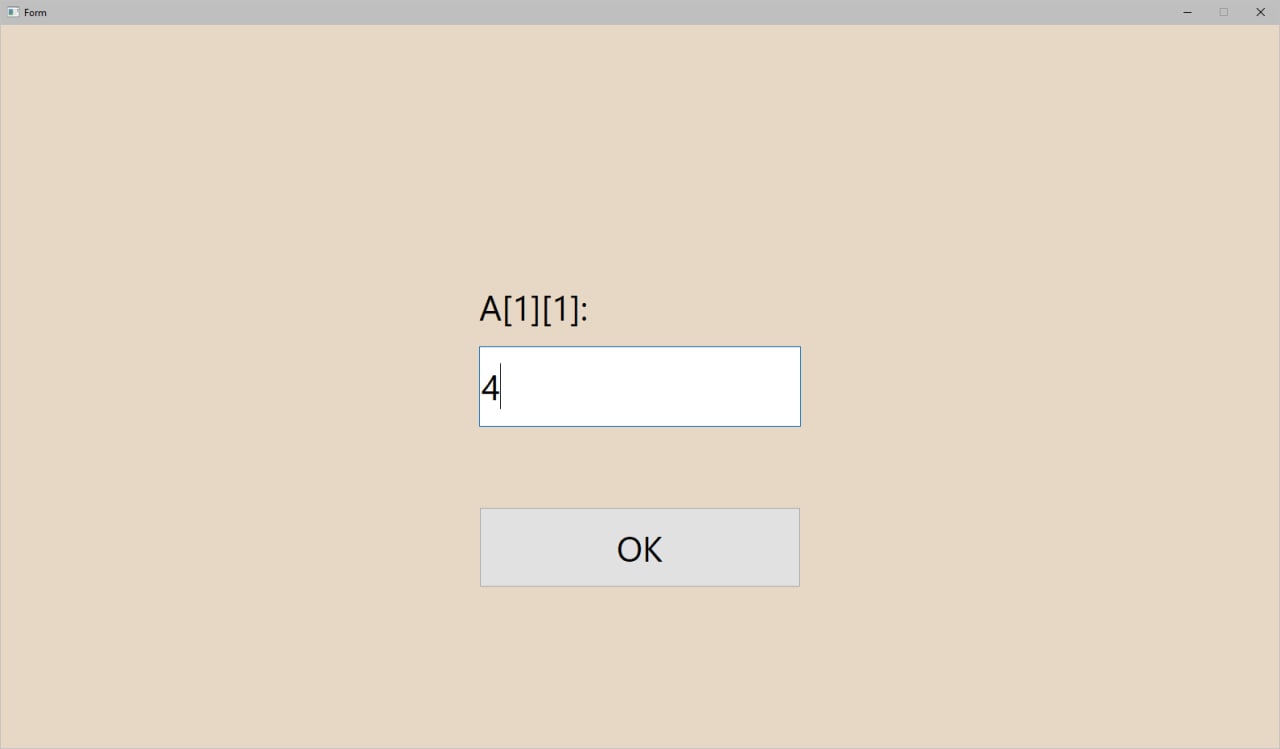
else:

solution\_vector = np.dot(inverted\_A, vector\_b)

return solution\_vector

**Пример работы программы:**

****

****

**4. Выводы**

Таким образом, в ходе выполнения лабораторной работы был изучен метод Гаусса и его модификации, составлен алгоритм метода и программа его реализации, получено численное решение заданной СЛАУ, составлен алгоритм решения СЛАУ указанными методами, применимый для организации вычислений на ЭВМ, составлена программу решения СЛАУ по разработанному алгоритму, выполнены тестовые примеры и проверена правильность работы программы.