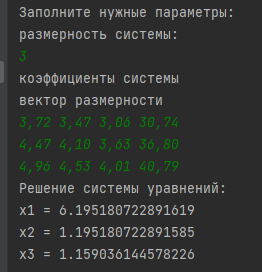
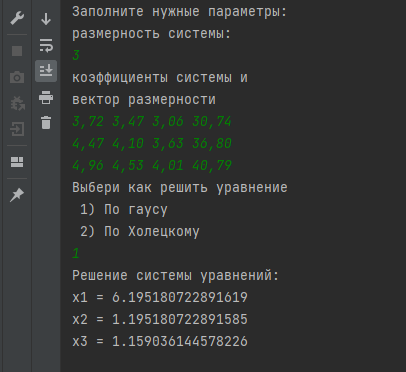
**Пример выполнения программы для 13 варианта:**

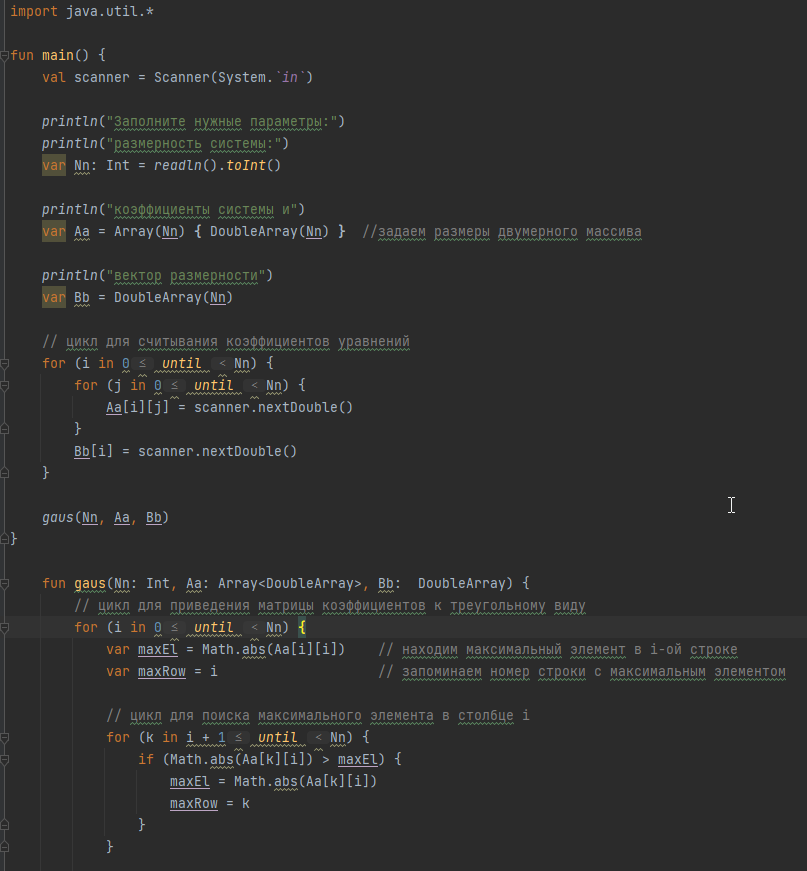
**Метод Гауса:**

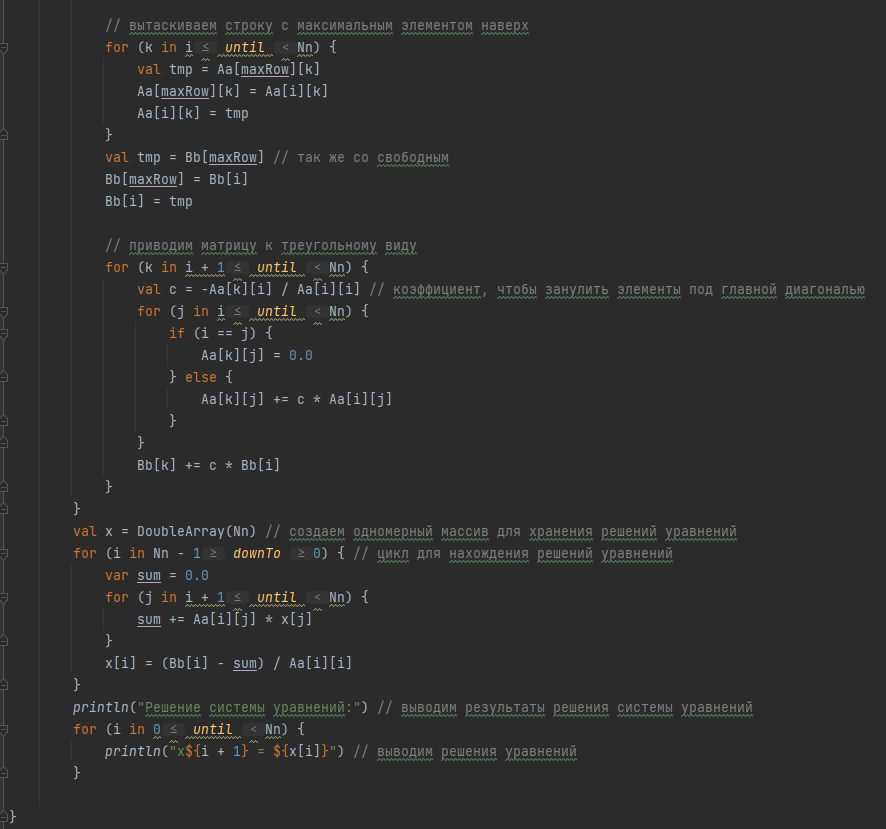


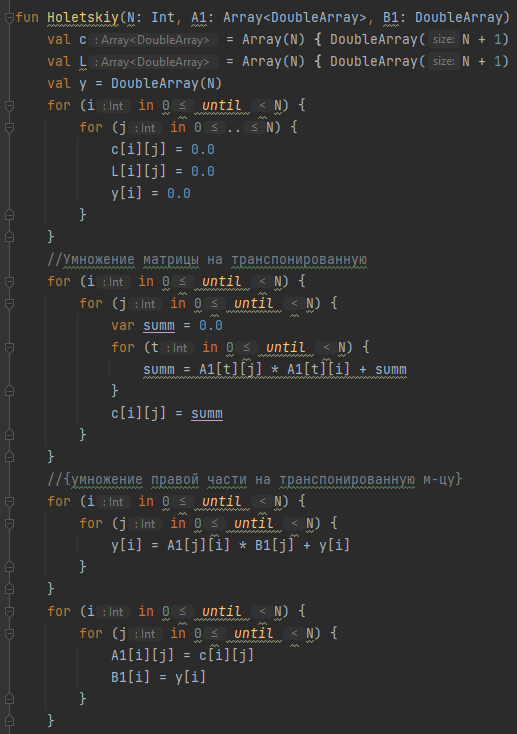
**Метод Холецкого:**

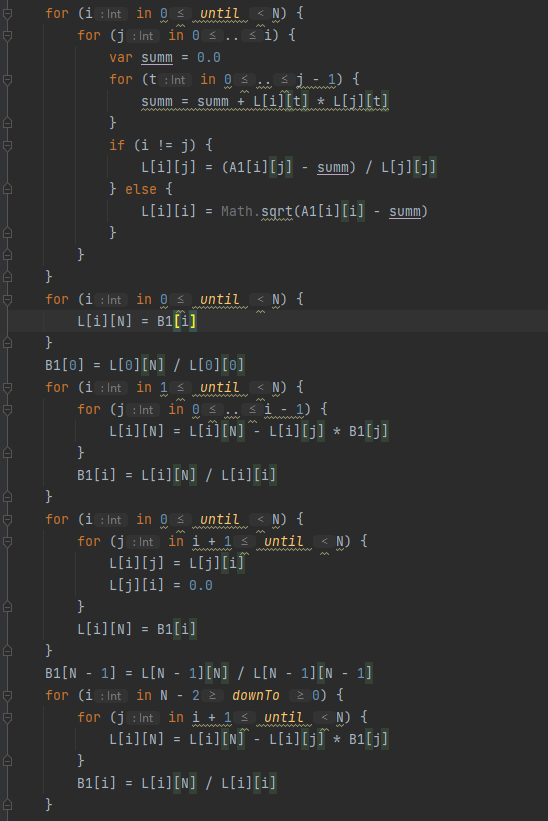


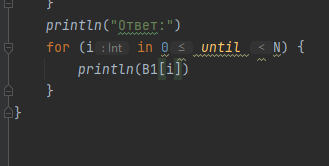
**Код программы:**











import java.lang.Math.abs  
import java.lang.Math.sqrt  
import java.util.\*  
import kotlin.math.\*  
  
fun main() {  
 val scanner = Scanner(System.*`in`*)  
  
 *println*("Заполните нужные параметры:")  
 *println*("размерность системы:")  
 var Nn: Int = *readln*().*toInt*()  
  
 *println*("коэффициенты системы и")  
 var Aa = Array(Nn) **{** DoubleArray(Nn) **}** //задаем размеры двумерного массива  
  
 *println*("вектор размерности")  
 var Bb = DoubleArray(Nn)  
  
 // цикл для считывания коэффициентов уравнений  
 for (i in 0 *until* Nn) {  
 for (j in 0 *until* Nn) {  
 Aa[i][j] = scanner.nextDouble()  
 }  
 Bb[i] = scanner.nextDouble()  
 }  
 *println*("Выбери как решить уравнение \n 1) По гаусу \n 2) По Холецкому")  
 var metod = *readln*().*toInt*()  
 if (metod == 1) *gaus*(Nn, Aa, Bb)  
 else *Holetskiy*(Nn, Aa, Bb)  
  
  
}  
  
 fun gaus(Nn: Int, Aa: Array<DoubleArray>, Bb: DoubleArray) {  
 // цикл для приведения матрицы коэффициентов к треугольному виду  
 for (i in 0 *until* Nn) {  
 var maxEl = Math.abs(Aa[i][i]) // находим максимальный элемент в i-ой строке  
 var maxRow = i // запоминаем номер строки с максимальным элементом  
  
 // цикл для поиска максимального элемента в столбце i  
 for (k in i + 1 *until* Nn) {  
 if (Math.abs(Aa[k][i]) > maxEl) {  
 maxEl = Math.abs(Aa[k][i])  
 maxRow = k  
 }  
 }  
  
 // вытаскиваем строку с максимальным элементом наверх  
 for (k in i *until* Nn) {  
 val tmp = Aa[maxRow][k]  
 Aa[maxRow][k] = Aa[i][k]  
 Aa[i][k] = tmp  
 }  
 val tmp = Bb[maxRow] // так же со свободным  
 Bb[maxRow] = Bb[i]  
 Bb[i] = tmp  
  
 // приводим матрицу к треугольному виду  
 for (k in i + 1 *until* Nn) {  
 val c = -Aa[k][i] / Aa[i][i] // коэффициент, чтобы занулить элементы под главной диагональю  
 for (j in i *until* Nn) {  
 if (i == j) {  
 Aa[k][j] = 0.0  
 } else {  
 Aa[k][j] += c \* Aa[i][j]  
 }  
 }  
 Bb[k] += c \* Bb[i]  
 }  
 }  
 val x = DoubleArray(Nn) // создаем одномерный массив для хранения решений уравнений  
 for (i in Nn - 1 *downTo* 0) { // цикл для нахождения решений уравнений  
 var sum = 0.0  
 for (j in i + 1 *until* Nn) {  
 sum += Aa[i][j] \* x[j]  
 }  
 x[i] = (Bb[i] - sum) / Aa[i][i]  
 }  
 *println*("Решение системы уравнений:") // выводим результаты решения системы уравнений  
 for (i in 0 *until* Nn) {  
 *println*("x${i + 1} = ${x[i]}") // выводим решения уравнений  
 }  
  
}  
  
fun Holetskiy(N: Int, A1: Array<DoubleArray>, B1: DoubleArray) {  
 val c = Array(N) **{** DoubleArray(N + 1) **}** val L = Array(N) **{** DoubleArray(N + 1) **}** val y = DoubleArray(N)  
 for (i in 0 *until* N) {  
 for (j in 0..N) {  
 c[i][j] = 0.0  
 L[i][j] = 0.0  
 y[i] = 0.0  
 }  
 }  
 //Умножение матрицы на транспонированную  
 for (i in 0 *until* N) {  
 for (j in 0 *until* N) {  
 var summ = 0.0  
 for (t in 0 *until* N) {  
 summ = A1[t][j] \* A1[t][i] + summ  
 }  
 c[i][j] = summ  
 }  
 }  
 //{умножение правой части на транспонированную м-цу}  
 for (i in 0 *until* N) {  
 for (j in 0 *until* N) {  
 y[i] = A1[j][i] \* B1[j] + y[i]  
 }  
 }  
 for (i in 0 *until* N) {  
 for (j in 0 *until* N) {  
 A1[i][j] = c[i][j]  
 B1[i] = y[i]  
 }  
 }  
 for (i in 0 *until* N) {  
 for (j in 0..i) {  
 var summ = 0.0  
 for (t in 0..j - 1) {  
 summ = summ + L[i][t] \* L[j][t]  
 }  
 if (i != j) {  
 L[i][j] = (A1[i][j] - summ) / L[j][j]  
 } else {  
 L[i][i] = Math.sqrt(A1[i][i] - summ)  
 }  
 }  
 }  
 for (i in 0 *until* N) {  
 L[i][N] = B1[i]  
 }  
 B1[0] = L[0][N] / L[0][0]  
 for (i in 1 *until* N) {  
 for (j in 0..i - 1) {  
 L[i][N] = L[i][N] - L[i][j] \* B1[j]  
 }  
 B1[i] = L[i][N] / L[i][i]  
 }  
 for (i in 0 *until* N) {  
 for (j in i + 1 *until* N) {  
 L[i][j] = L[j][i]  
 L[j][i] = 0.0  
 }  
 L[i][N] = B1[i]  
 }  
 B1[N - 1] = L[N - 1][N] / L[N - 1][N - 1]  
 for (i in N - 2 *downTo* 0) {  
 for (j in i + 1 *until* N) {  
 L[i][N] = L[i][N] - L[i][j] \* B1[j]  
 }  
 B1[i] = L[i][N] / L[i][i]  
 }  
 *println*("Ответ:")  
 for (i in 0 *until* N) {  
 *println*(B1[i])  
 }  
}