БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет КСИС

Специальность ИиТП

Индивидуальная практическая работа №1

по дисциплине «Математическое моделирование»

Выполнил студент: Драгун О.В.

группа 893551

Зачетная книжка № 2520050

Минск 2022

# Формирования двумерной случайной величины

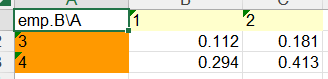
Формировать двумерную случайную величину будем согласно алгоритму, описанному в задаче.  
Для формирования эксперимента будем использовать следующую форму:  
Изображение выглядит как текст

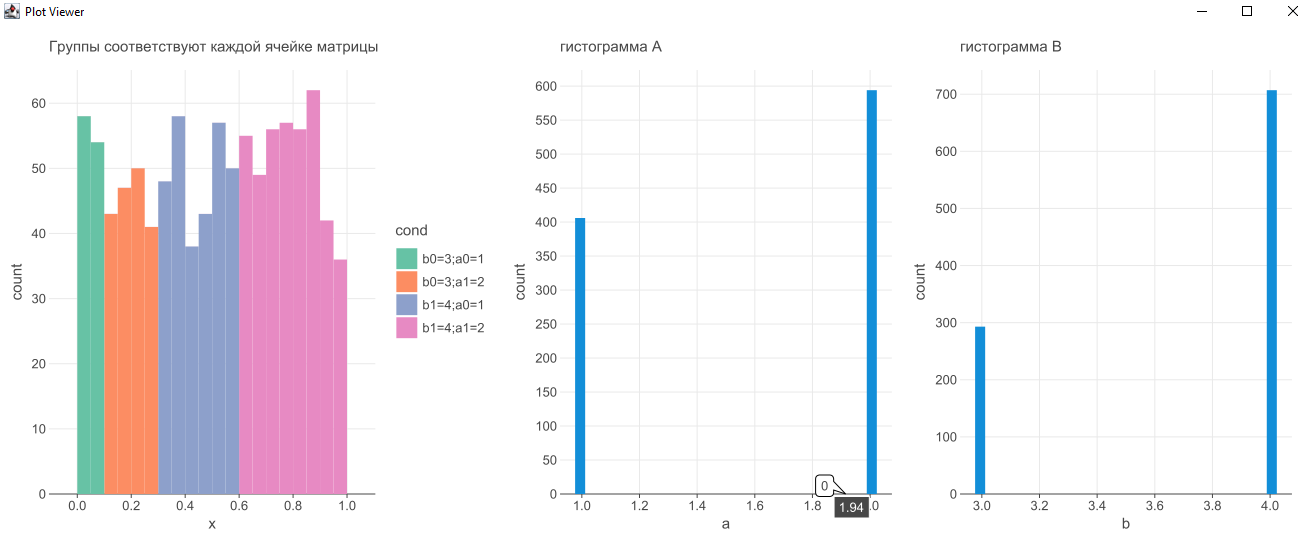
Автоматически созданное описание  
где из весов матрицы сформируем теоретическую матрицу вероятностей.  
Используем полученные ранее знания о генерации excel файлов для записи отчетов по экспериментам.

Теоретическая матрица вероятностей и значения векторов:

Изображение выглядит как стол

Автоматически созданное описание

Эмпирическая матрица вероятностей для 1000 случайных двумерных величин:  
  
Группы по ячейкам матрицы, а также гистограммы A и B



# Статистическое исследование

Все оценки полученных двумерных случайных величин выведем в таблицу

|  |  |
| --- | --- |
| (A)выборочное среднее: | 1.594 |
| (A)Несмещенная состоятельная оценка дисперсии S02: | 0.241405405 |
| (A)Cмещенная состоятельная оценка дисперсии S2: | 0.241164 |
| (A)Состоятельная оценка среднеквадратичного отклонения: | 0.491330241 |
| (A)Выборочный начальный момент 1-го порядка: | 1.594 |
| (A)Выборочный центральный момент 1-го порядка: | 0.001 |
| (A)Выборочный начальный момент 2-го порядка: | 2.782 |
| (A)Выборочный центральный момент 2-го порядка: | 241.165 |
| (A)Выборочный начальный момент 3-го порядка: | 5.158 |
| (A)Выборочный центральный момент 3-го порядка: | -45.337832 |
| (A)Выборочный начальный момент 4-го порядка: | 9.91 |
| (A)Выборочный центральный момент 4-го порядка: | 66.68477531 |
| (A)Выборочный начальный момент 5-го порядка: | 19.414 |
| (A)Выборочный центральный момент 5-го порядка: | -23.46964384 |
| (B)выборочное среднее | 3.707 |
| (B)Несмещенная состоятельная оценка дисперсии S02: | 0.207358358 |
| (B)Cмещенная состоятельная оценка дисперсии S2: | 0.207151 |
| (B)Состоятельная оценка среднеквадратичного отклонения: | 0.455366181 |
| (B)Выборочный начальный момент 1-го порядка: | 3.707 |
| (B)Выборочный центральный момент 1-го порядка: | 0.001 |
| (B)Выборочный начальный момент 2-го порядка: | 13.949 |
| (B)Выборочный центральный момент 2-го порядка: | 207.152 |
| (B)Выборочный начальный момент 3-го порядка: | 53.159 |
| (B)Выборочный центральный момент 3-го порядка: | -85.759514 |
| (B)Выборочный начальный момент 4-го порядка: | 204.725 |
| (B)Выборочный центральный момент 4-го порядка: | 78.4173896 |
| (B)Выборочный начальный момент 5-го порядка: | 795.167 |
| (B)Выборочный центральный момент 5-го порядка: | -50.22876153 |
| Несмещенная состоятельная оценка корреляционного момента kab: | -0.006964965 |
| Состоятельная оценка коэффициента корреляции rab | -0.031130398 |

По последним двум оценкам, приближающимся к 0 можно утверждать, что генерация двумерной случайной величины корректна.

Подсчет доверительных интервалов также выполняется автоматически:  
Изображение выглядит как стол

Автоматически созданное описание  
Теоретические ma и mb находятся в пределах интервала, значит эксперимент корректен

# Листинг кода

## Главный файл

package ipr\_one  
  
import java.awt.FlowLayout  
import java.awt.event.KeyAdapter  
import java.awt.event.KeyEvent  
import javax.swing.\*  
  
class IprOneInputGetter : JFrame("VectorsGetter") {  
 companion object {  
 @JvmStatic  
 fun main(args: Array<String>) {  
 //remove annoying warning "Graphics2D from BufferedImage lacks BUFFERED\_IMAGE hint", was actual for 1/2 PC  
 System.setProperty("org.apache.batik.warn\_destination", "false")  
  
 SwingUtilities.invokeLater **{** IprOneInputGetter() **}** }  
 val excelReportGenerator = ExcelReportGenerator()  
 val random2DValueGenerator = Random2DValueGenerator(isRealRandom = true, isDebug = false)  
 }  
 var button = JButton("Process")  
 val label = JLabel("Задайте целочисленные значения")  
  
 val labelN = JLabel("n =")  
 val textFieldN = JTextField(ParamsSaver.loadIprOneParams().n.toString(), 4)  
  
 val labelM = JLabel("m =")  
 val textFieldM = JTextField(ParamsSaver.loadIprOneParams().m.toString(), 4)  
  
 val labelMatrix = JLabel("Веса матрицы =")  
 val textFieldMatrix = JTextField(ParamsSaver.loadIprOneParams().matrix, 23)  
  
 val labelA = JLabel("Значения вектора А =")  
 var textFieldA = JTextField(ParamsSaver.loadIprOneParams().vectorAString, 20)  
  
 val labelB = JLabel("Значения вектора B= ")  
 var textFieldB = JTextField(ParamsSaver.loadIprOneParams().vectorBString, 20)  
  
 val labelRVN = JLabel("Число случайных величин:")  
 var textFieldRVN = JTextField(ParamsSaver.loadIprOneParams().RVN.toString(), 10)  
  
 init {  
 *layout* = FlowLayout()  
 *defaultCloseOperation* = *EXIT\_ON\_CLOSE* setLocationRelativeTo(null)  
 *isVisible* = true  
 setSize(420, 200)  
  
 val keyListener = object : KeyAdapter() {  
 override fun keyTyped(e: KeyEvent) {  
 val c = e.*keyChar* if (!((c in '0'..'9')  
 || c.*code* == KeyEvent.*VK\_BACK\_SPACE* || c.*code* == KeyEvent.*VK\_SPACE* || c.*code* == KeyEvent.*VK\_DELETE*)) {  
 *toolkit*.beep()  
 e.consume()  
 }  
 }  
 }  
 textFieldN.addKeyListener(keyListener)  
 textFieldM.addKeyListener(keyListener)  
 textFieldMatrix.addKeyListener(keyListener)  
 textFieldA.addKeyListener(keyListener)  
 textFieldB.addKeyListener(keyListener)  
 textFieldRVN.addKeyListener(keyListener)  
  
 button.addActionListener **{** processButtonClick()  
 **}** add(label)  
 add(labelN)  
 add(textFieldN)  
 add(labelM)  
 add(textFieldM)  
 add(labelMatrix)  
 add(textFieldMatrix)  
 add(labelA)  
 add(textFieldA)  
 add(labelB)  
 add(textFieldB)  
 add(labelRVN)  
 add(textFieldRVN)  
 add(button)  
 }  
  
 private fun processButtonClick() {  
 try {  
 val n = textFieldN.*text*.*toInt*().*also* **{** *println*("n = $**it**") **}** val m = textFieldM.*text*.*toInt*().*also* **{** *println*("m = $**it**") **}** val matrixString = textFieldMatrix.*text*.*also* **{** *println*("matrix string = $**it**") **}** val vectorAString = textFieldA.*text*.*also* **{** *println*("Vector A = $**it**") **}** val vectorBString = textFieldB.*text*.*also* **{** *println*("Vector B = $**it**") **}** val RVN = textFieldRVN.*text*.*toInt*().*also* **{** *println*("число случайных величин = $**it**") **}** val vectorA = vectorAString.*trim*().*split*(" ").*map* **{ it**.*toInt*() **}** val vectorB = vectorBString.*trim*().*split*(" ").*map* **{ it**.*toInt*() **}** InputValidator.validateInput(n, m, matrixString, vectorAString, vectorBString, RVN)  
 val probMatrix = excelReportGenerator.generateProbabilityMatrix(n, m, matrixString, vectorA, vectorB)  
 val listToAnalyze = random2DValueGenerator.invoke(  
 probMatrix = probMatrix,  
 vectorA = vectorA,  
 vectorB = vectorB,  
 RVN = RVN  
 )  
 val empiricalMatrix = excelReportGenerator.generateEmpiricalDistributionMatrix(  
 n = n,  
 m = m,  
 list = listToAnalyze,  
 vectorA = vectorA,  
 vectorB = vectorB  
 )  
 random2DValueGenerator.drawVectorsHist(listToAnalyze, empiricalMatrix, vectorA, vectorB)  
 excelReportGenerator.generateEstimatesSheet(listToAnalyze, RVN, probMatrix, vectorA, vectorB)  
 excelReportGenerator.generateDataListSheet(listToAnalyze)  
 ParamsSaver.saveIprOneParams(IprOneParams(  
 n = n,  
 m = m,  
 matrix = matrixString,  
 vectorAString = vectorAString,  
 vectorBString = vectorBString,  
 RVN = RVN  
 ))  
 } catch (e:Exception) {  
 JOptionPane.showMessageDialog(  
 this@IprOneInputGetter,  
 """  
 Ошибка во время процессинга:  
 ${e.message}  
 """.*trimIndent*()  
 )  
 }  
 }  
}

## Валидация ввода

package ipr\_one  
  
object InputValidator {  
  
 fun validateInput(n: Int, m: Int, matrixString: String, vectorAString: String, vectorBString: String, RVN:Int) {  
 val matrixWeights = matrixString.*trim*().*split*(" ")  
 if (n == 0) throw Exception("n не может быть 0")  
 if (m == 0) throw Exception("m не может быть 0")  
 if (n\*m != matrixWeights.size) throw Exception("размер матрицы не равен n\*m")  
 val vectorAList = vectorAString.*trim*().*split*(" ")  
 val vectorBList = vectorBString.*trim*().*split*(" ")  
 if (vectorAList.size != n) throw Exception("размер вектора А не равен n")  
 if (vectorBList.size != m) throw Exception("размер вектора B не равен m")  
 if (RVN == 0) throw Exception("Число случайных величин должно быть хотя бы 1")  
 }  
}

## Сохранение введенных параметров

package ipr\_one  
  
import java.util.prefs.Preferences  
  
object ParamsSaver {  
 private fun prefNode(): Preferences = Preferences.userRoot().node("MMOD\_STORAGE")  
  
 fun loadIprOneParams(): IprOneParams {  
 return prefNode().*run* **{** IprOneParams(  
 n = getInt(IprOneParams.N\_KEY, 2),  
 m = getInt(IprOneParams.M\_KEY, 2),  
 matrix = get(IprOneParams.MATRIX\_KEY, "1 2 3 4"),  
 vectorAString = get(IprOneParams.VECTOR\_A\_KEY,"1 2"),  
 vectorBString = get(IprOneParams.VECTOR\_B\_KEY," 3 4"),  
 RVN = getInt(IprOneParams.RVN\_KEY,200)  
 )  
 **}** }  
 fun saveIprOneParams(params: IprOneParams) {  
 prefNode().*apply* **{** putInt(IprOneParams.N\_KEY, params.n)  
 putInt(IprOneParams.M\_KEY, params.m)  
 put(IprOneParams.MATRIX\_KEY, params.matrix)  
 put(IprOneParams.VECTOR\_A\_KEY, params.vectorAString)  
 put(IprOneParams.VECTOR\_B\_KEY, params.vectorBString)  
 putInt(IprOneParams.RVN\_KEY, params.RVN)  
 **}** }  
}  
  
data class IprOneParams(  
 val n: Int,  
 val m: Int,  
 val matrix: String,  
 val vectorAString: String,  
 val vectorBString: String,  
 val RVN: Int  
) {  
 companion object{  
 const val N\_KEY = "n"  
 const val M\_KEY = "M"  
 const val MATRIX\_KEY = "matrix"  
 const val VECTOR\_A\_KEY = "vectorA"  
 const val VECTOR\_B\_KEY = "vectorB"  
 const val RVN\_KEY = "RVN"  
 }  
}

## Генерация двумерной случайно величины

package ipr\_one  
  
import kr\_one.RandomEventGenerator  
import org.jetbrains.letsPlot.GGBunch  
import org.jetbrains.letsPlot.geom.geomHistogram  
import org.jetbrains.letsPlot.intern.Plot  
import org.jetbrains.letsPlot.label.ggtitle  
import org.jetbrains.letsPlot.letsPlot  
  
fun main() {  
 //remove annoying warning "Graphics2D from BufferedImage lacks BUFFERED\_IMAGE hint", was actual for 1/2 PC  
 System.setProperty("org.apache.batik.warn\_destination", "false")  
 val random2DValueGenerator = Random2DValueGenerator(isRealRandom = true, isDebug = true)  
 val result = random2DValueGenerator.invoke(  
 probMatrix = *arrayOf*(  
 *doubleArrayOf*(1 / 8.0, 2 / 8.0, 2 / 8.0),  
 *doubleArrayOf*(0.0, 0.0, 3 / 8.0)  
 ),  
 vectorA = *listOf*(10, 20, 30),  
 vectorB = *listOf*(40, 50),  
 RVN = 10  
 )  
 *println*("size=" + result.size)  
  
}  
  
class Random2DValueGenerator(  
 val isRealRandom: Boolean,  
 val isDebug: Boolean  
) {  
 private val randomEventGenerator = RandomEventGenerator(isRealRandom = isRealRandom, isDebug = false)  
  
 operator fun invoke(  
 probMatrix: Array<DoubleArray>,  
 vectorA: List<Int>,  
 vectorB: List<Int>,  
 RVN: Int  
 ): List<Result> {  
 val result: MutableList<Result> = *mutableListOf*()  
 if (isDebug) {  
 probMatrix.*forEachIndexed* **{** index, doubles **->** *println*("строка $index значения:")  
 doubles.*forEach* **{** *print*("$**it**;")  
 **}** *print*("\n")  
 **}** }  
 *repeat*(RVN) **{** var randomNumber = randomEventGenerator.invoke(0.5).randomNumber  
 var tempSumToCompare = 0.0  
  
 probMatrix.*forEachIndexed* **{** row, doubles **->** doubles.*forEachIndexed* **{** column, d **->** tempSumToCompare += d  
 if (randomNumber <= tempSumToCompare) {  
 result += Result(  
 a = vectorA[column],  
 b = vectorB[row],  
 row = row,  
 column = column,  
 randomNumber = randomNumber  
 )  
 if (isDebug) {  
 *println*("число $randomNumber")  
 *println*("строка $row")  
 *println*("столбец $column")  
 }  
 //to not add more numbers  
 randomNumber = 2.0  
 }  
 **}  
 }  
 }** *println*("Фактическое кол-во элементов в списке для анализа: ${result.size}")  
 return result  
 }  
  
 data class Result(  
 val a: Int,  
 val b: Int,  
 val row: Int,  
 val column: Int,  
 val randomNumber: Double  
 )  
  
 fun drawVectorsHist(list: List<Result>, probMatrix: Array<DoubleArray>,vectorA: List<Int>, vectorB: List<Int>) {  
 GGBunch().*apply* **{** addPlot(groupsPlot(list, probMatrix, vectorA, vectorB), 0, 0, width = 500, height = 500)  
 addPlot(vectorAPlot(list),500, 0, width = 400, height = 500)  
 addPlot(vectorBPlot(list), 900, 0, width = 400, height = 500)  
 show()  
 **}** }  
  
 private fun groupsPlot(list: List<Result>, probMatrix: Array<DoubleArray>,vectorA: List<Int>, vectorB: List<Int>) : Plot {  
 val possibleEventSublist: MutableList<Triple<Int, Int, String>> = *mutableListOf*()  
 probMatrix.*forEachIndexed* **{** row, doubles **->** doubles.*forEachIndexed* **{** column, d **->** possibleEventSublist += Triple(row, column, "b$row=${vectorB[row]};a$column=${vectorA[column]}")  
 **}  
 }** val resultCondList: MutableList<String> = *mutableListOf*()  
 val resultXList: MutableList<Double> = *mutableListOf*()  
  
 possibleEventSublist.*forEach* **{** possibleEvent **->** val resultXListIteration = list  
 .*filter* **{ it**.row == possibleEvent.first && **it**.column == possibleEvent.second **}** .*map* **{ it**.randomNumber **}** resultCondList += list  
 .*filter* **{ it**.row == possibleEvent.first && **it**.column == possibleEvent.second **}** .*map* **{** possibleEvent.third **}** resultXList += resultXListIteration  
 *println*("event ${possibleEvent.third}: size = ${resultXListIteration.size}")  
 **}** val data = *mapOf*<String, Any>(  
 "cond" *to* resultCondList,  
 "x" *to* resultXList  
 )  
 val p = *letsPlot*(data) **{**x = "x"; fill = "cond"**}** return p+ geomHistogram(boundary = 0.0, binWidth = 0.05) + *ggtitle*("Группы соответствуют каждой ячейке матрицы")  
 }  
  
 private fun vectorAPlot(list: List<Result>): Plot {  
 val data = *mapOf*(  
 "a" *to* list.*map* **{ it**.a **}** )  
 val p = *letsPlot*(data) **{**x = "a"**}** return p+ geomHistogram() + *ggtitle*("гистограмма А")  
 }  
  
 private fun vectorBPlot(list: List<Result>): Plot {  
 val data = *mapOf*(  
 "b" *to* list.*map* **{ it**.b **}** )  
 val p = *letsPlot*(data) **{**x = "b"**}** return p+ geomHistogram() + *ggtitle*("гистограмма B")  
 }  
}

## Генерация excel-отчета

package ipr\_one  
  
import kr\_two.OSDetector  
import org.apache.poi.hssf.usermodel.HSSFWorkbook  
import org.apache.poi.ss.usermodel.FillPatternType  
import org.apache.poi.ss.usermodel.IndexedColors  
import java.io.File  
import java.io.FileOutputStream  
import kotlin.math.pow  
import kotlin.math.sqrt  
  
  
class ExcelReportGenerator {  
 private val workbook = HSSFWorkbook()  
 private val vectorACellStyle = workbook.createCellStyle().*apply* **{** *fillForegroundColor* = IndexedColors.*LEMON\_CHIFFON*.getIndex()  
 *fillPattern* = FillPatternType.*SOLID\_FOREGROUND* **}** private val vectorBCellStyle = workbook.createCellStyle().*apply* **{** *fillForegroundColor* = IndexedColors.*LIGHT\_ORANGE*.getIndex()  
 *fillPattern* = FillPatternType.*SOLID\_FOREGROUND* **}** fun generateProbabilityMatrix(  
 n: Int,  
 m: Int,  
 matrixString: String,  
 vectorA: List<Int>,  
 vectorB: List<Int>  
 ): Array<DoubleArray> {  
 //for 2nd process button click  
 *repeat*(workbook.*numberOfSheets*) **{** workbook.removeSheetAt(0)  
 **}** val matrixWeights = matrixString.*trim*().*split*(" ").*map* **{ it**.*toInt*() **}** val matrixSum = matrixWeights.*sum*().*also* **{** *println*("Сумма весов матрицы: $**it**") **}** val matrix = Array(m) **{** row **->** DoubleArray(n) **{** column **->** matrixWeights[row \* n + column] / matrixSum.toDouble()  
 **}  
 }** val sheet = workbook.createSheet(MATRIX\_SHEET\_NAME)  
 var summ = 0.0  
 //shift n m to +1 to set vector values to table also  
 *repeat*(m + 1) **{** rowIndex **->** if (rowIndex == 0) {  
 sheet.createRow(0).*apply* **{** createCell(0).setCellValue("B\\A")  
 *repeat*(n) **{** columnIndex **->** createCell(columnIndex + 1).*apply* **{** setCellValue(vectorA[columnIndex].toString())  
 setCellStyle(vectorACellStyle)  
 **}  
 }  
 }** } else {  
 sheet.createRow(rowIndex).*apply* **{** *repeat*(n + 1) **{** columnIndex **->** if (columnIndex == 0) {  
 createCell(0).*apply* **{** setCellValue(vectorB[rowIndex - 1].toString())  
 setCellStyle(vectorBCellStyle)  
 **}** } else {  
 val value = matrix[rowIndex - 1][columnIndex - 1]  
 summ += value  
 createCell(columnIndex).setCellValue(value)  
 }  
 **}  
 }** }  
 **}** //https://stackoverflow.com/questions/15625556/adding-and-subtracting-doubles-are-giving-strange-results  
 //if (summ != 1.0) throw Exception("Сумма вероятностей не равна 1. Степень двойки в знаменателе поправит проблему")  
 *repeat*(n) **{** sheet.setColumnWidth(**it**, 13 \* 256)  
 **}** *println*("Сумма всех ячеек: $summ")  
 return matrix  
 }  
  
 fun generateEmpiricalDistributionMatrix(  
 n: Int,  
 m: Int,  
 list: List<Random2DValueGenerator.Result>,  
 vectorA: List<Int>,  
 vectorB: List<Int>  
 ): Array<DoubleArray> {  
 val empiricalMatrix = Array(m) **{** row **->** DoubleArray(n) **{** column **->** list.*filter* **{ it**.row == row && **it**.column == column **}**.size / list.size.toDouble()  
 **}  
 }** val sheet = workbook.createSheet(EMPIRICAL\_MATRIX\_SHEET\_NAME)  
 var summ = 0.0  
 //shift n m to +1 to set vector values to table also  
 *repeat*(m + 1) **{** rowIndex **->** if (rowIndex == 0) {  
 sheet.createRow(0).*apply* **{** createCell(0).setCellValue("emp.B\\A")  
 *repeat*(n) **{** columnIndex **->** createCell(columnIndex + 1).*apply* **{** setCellValue(vectorA[columnIndex].toString())  
 setCellStyle(vectorACellStyle)  
 **}  
 }  
 }** } else {  
 sheet.createRow(rowIndex).*apply* **{** *repeat*(n + 1) **{** columnIndex **->** if (columnIndex == 0) {  
 createCell(0).*apply* **{** setCellValue(vectorB[rowIndex - 1].toString())  
 setCellStyle(vectorBCellStyle)  
 **}** } else {  
 val value = empiricalMatrix[rowIndex - 1][columnIndex - 1]  
 summ += value  
 createCell(columnIndex).setCellValue(value)  
 }  
 **}  
 }** }  
 **}** *repeat*(n) **{** sheet.setColumnWidth(**it**, 13 \* 256)  
 **}** *println*("Сумма всех ячеек эмпирически: $summ")  
 return empiricalMatrix  
 }  
  
 fun generateEstimatesSheet(  
 list: List<Random2DValueGenerator.Result>,  
 RVN: Int,  
 theoreticalMatrix: Array<DoubleArray>,  
 vectorA: List<Int>,  
 vectorB: List<Int>  
 ) {  
 val reportList = *mutableListOf*<Pair<String, String>>()  
  
 */\*\*  
 \*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Анализируем a\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 \*/* val aList = list.*map* **{ it**.a **}** val dA = aList.*average*().*also* **{** reportList.add("(A)выборочное среднее:" *to* **it**.toString())  
 **}** val aS02 = 1.0 / (RVN - 1) \* aList.*fold*(0.0) **{** sum, element **->** sum + (element - dA).*pow*(2)  
 **}** reportList.add("(A)Несмещенная состоятельная оценка дисперсии S02:" *to* aS02.toString())  
 val aS2 = 1.0 / RVN \* aList.*fold*(0.0) **{** sum, element **->** sum + (element - dA).*pow*(2)  
 **}** reportList.add("(A)Cмещенная состоятельная оценка дисперсии S2:" *to* aS2.toString())  
 val aS0 = *sqrt*(aS02)  
 reportList.add("(A)Состоятельная оценка среднеквадратичного отклонения:" *to* aS0.toString())  
 *repeat*(5) **{** k **->** val aak = 1.0 / RVN \* aList.*fold*(0.0) **{** sum, element **->** sum + element.toDouble().*pow*(k + 1)  
 **}** reportList.add("(A)Выборочный начальный момент ${k + 1}-го порядка:" *to* aak.toString())  
 val amuk = 1.0 / RVN + aList.*fold*(0.0) **{** sum, element **->** sum + (element - dA).*pow*(k + 1)  
 **}** reportList.add("(A)Выборочный центральный момент ${k + 1}-го порядка:" *to* amuk.toString())  
 **}** */\*\*  
 \*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Анализируем b\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 \*/* val bList = list.*map* **{ it**.b **}** val dB = bList.*average*()  
 reportList.add("(B)выборочное среднее" *to* dB.toString())  
 val bS02 = 1.0 / (RVN - 1) \* bList.*fold*(0.0) **{** sum, element **->** sum + (element - dB).*pow*(2)  
 **}** reportList.add("(B)Несмещенная состоятельная оценка дисперсии S02:" *to* bS02.toString())  
 val bS2 = 1.0 / RVN \* bList.*fold*(0.0) **{** sum, element **->** sum + (element - dB).*pow*(2)  
 **}** reportList.add("(B)Cмещенная состоятельная оценка дисперсии S2:" *to* bS2.toString())  
 val bS0 = *sqrt*(bS02)  
 reportList.add("(B)Состоятельная оценка среднеквадратичного отклонения:" *to* bS0.toString())  
 *repeat*(5) **{** k **->** val bak = 1.0 / RVN \* bList.*fold*(0.0) **{** sum, element **->** sum + element.toDouble().*pow*(k + 1)  
 **}** reportList.add("(B)Выборочный начальный момент ${k + 1}-го порядка:" *to* bak.toString())  
 val bmuk = 1.0 / RVN + bList.*fold*(0.0) **{** sum, element **->** sum + (element - dB).*pow*(k + 1)  
 **}** reportList.add("(B)Выборочный центральный момент ${k + 1}-го порядка:" *to* bmuk.toString())  
 **}** */\*\*  
 \*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Посчитаем ma и mb из теоретической таблицы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 \*/* val mb = vectorB.*foldIndexed*(0.0) **{** index, sum, element **->** sum + (theoreticalMatrix[index].*sum*() \* element)  
 **}** val ma = vectorA.*foldIndexed*(0.0) **{** index, sum, element **->** sum + (theoreticalMatrix.*map* **{ it**[index] **}**.*sum*() \* element)  
 **}** */\*\*  
 \*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Анализируем a b\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 \*/* val kab = 1.0 / (RVN - 1) \* list.*fold*(0.0) **{** sum, element **->** sum + (element.a - dA) \* (element.b - dB)  
 **}** reportList.add("Несмещенная состоятельная оценка корреляционного момента kab:" *to* kab.toString())  
 val rab = list.*fold*(0.0) **{** sum, element **->** sum + (element.a - dA) \* (element.b - dB)  
 **}** / *sqrt*(  
 aList.*fold*(0.0) **{** sum, element **->** sum + (element - dA).*pow*(2) **}** \*  
 bList.*fold*(0.0) **{** sum, element **->** sum + (element - dB).*pow*(2) **}** )  
 reportList.add("Состоятельная оценка коэффициента корреляции rab" *to* rab.toString())  
  
 reportEstimates(reportList,ma, mb, RVN)  
 }  
  
 private fun reportEstimates(  
 list: List<Pair<String, String>>,  
 ma: Double,  
 mb: Double,  
 RVN: Int  
 ) {  
 val sheet = workbook.createSheet(ESTIMATES\_SHEET\_NAME)  
  
  
 list.*forEachIndexed* **{** index, pair **->** *println*(pair.first + " " + pair.second)  
 sheet.createRow(index).*apply* **{** createCell(0).setCellValue(pair.first)  
 pair.second.*toDoubleOrNull*()?.*let* **{** createCell(1).setCellValue(**it**)  
 **}** ?: createCell(1).setCellValue(pair.second)  
 **}  
 }** sheet.setColumnWidth(0, 60 \* 256)  
 sheet.setColumnWidth(1, 13 \* 256)  
 sheet.setColumnWidth(3, 21 \* 256)  
 sheet.setColumnWidth(4, 13 \* 256)  
 sheet.setColumnWidth(5, 27 \* 256)  
 sheet.setColumnWidth(6, 13 \* 256)  
  
 sheet.getRow(0).*apply* **{** createCell(3).setCellValue("RVN-1")  
 createCell(4).setCellValue(RVN - 1.0)  
 **}** sheet.getRow(1).*apply* **{** createCell(3).setCellValue("1-уровень значимости")  
 createCell(4).setCellValue(0.01)  
 **}** sheet.getRow(2).*apply* **{** createCell(3).setCellValue("t")  
 //СТЬЮДЕНТ.ОБР.2Х  
 createCell(4).*cellFormula* = "\_xlfn.T.INV.2T(E2,E1)"  
 **}** sheet.getRow(3).*apply* **{** createCell(5).setCellValue("Доверительный интервал ma")  
 **}** sheet.getRow(4).*apply* **{** createCell(4).*cellFormula* = "B1-(B2\*E3)/SQRT(E1)"  
 createCell(5).setCellValue("<=ma<=")  
 createCell(6).*cellFormula* = "B1+(B2\*E3)/SQRT(E1)"  
  
 **}** sheet.getRow(5).*apply* **{** createCell(5).setCellValue("Доверительный интервал mb")  
 **}** sheet.getRow(6).*apply* **{** createCell(4).*cellFormula* = "B15-(B16\*E3)/SQRT(E1)"  
 createCell(5).setCellValue("<=ma<=")  
 createCell(6).*cellFormula* = "B15+(B16\*E3)/SQRT(E1)"  
 **}** sheet.getRow(8).*apply* **{** createCell(5).setCellValue("Теоретическое ma:")  
 createCell(6).setCellValue(ma)  
 **}** sheet.getRow(9).*apply* **{** createCell(5).setCellValue("Теоретическое mb:")  
 createCell(6).setCellValue(mb)  
 **}** }  
  
 fun generateDataListSheet(list: List<Random2DValueGenerator.Result>) {  
 val sheet = workbook.createSheet(DATA\_SHEET\_NAME)  
 sheet.createRow(0).*apply* **{** createCell(0).setCellValue("a")  
 createCell(1).setCellValue("b")  
 createCell(2).setCellValue("randomNumber")  
 **}** list.*forEachIndexed* **{** index, result **->** sheet.createRow(index + 1).*apply* **{** createCell(0).setCellValue(result.a.toDouble())  
 createCell(1).setCellValue(result.b.toDouble())  
 createCell(2).setCellValue(result.randomNumber)  
 **}  
 }** sheet.setColumnWidth(0, 13 \* 256)  
 sheet.setColumnWidth(1, 13 \* 256)  
 sheet.setColumnWidth(2, 13 \* 256)  
 writeWorkbookToFile()  
 }  
  
 private fun writeWorkbookToFile() {  
 FileOutputStream(PATH).*use* **{** fileOut **->** workbook.write(fileOut)  
 workbook.close()  
 fileOut.close()  
 **}** OSDetector.openWithSystem(File(PATH))  
 }  
  
 companion object {  
 private const val PATH = "src/main/kotlin/ipr\_one/CDCB\_report.xls"  
 private const val MATRIX\_SHEET\_NAME = "Матрица вероятностей"  
 private const val EMPIRICAL\_MATRIX\_SHEET\_NAME = "Эмпирическая матрица вероятностей"  
 private const val ESTIMATES\_SHEET\_NAME = "Оценки"  
 private const val DATA\_SHEET\_NAME = "Данные"  
 }  
}