БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет КСИС

Специальность ИиТП

Контрольная работа №2

по дисциплине «Математическое моделирование»

Вариант (2520050 mod 8) + 1 = 3

3. Распределение Эрланга

Выполнил студент: Драгун О.В.

группа 893551

Зачетная книжка № 2520050

Минск 2022

# ИМИТАЦИЯ НЕПРЕРЫВНЫХ СЛУЧАЙНЫХ ВЕЛИЧИН (МЕТОД ОБРАТНЫХ ФУНКЦИЙ)

## Теоретическая функция распределения

Теоретическая функция распределения Эрланга выглядит как

Изображение выглядит как текст

Автоматически созданное описание

Реализация на kotlin в работе выглядит так:

fun calculateTheoretical(n: Int, mu: Double, x: Double): Double =  
 (mu.*pow*(n) \* x.*pow*(n - 1) \* *E*.*pow*(-mu \* x)) / factorial(n - 1)

Где n это shape-параметр, в формуле выше обозначен как k

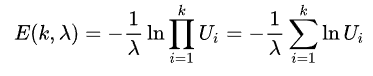
mu это rate-параметр, в формуле выше обозначен как λ

## Получение случайного числа из распределения Эрланга

Случайные величины, распределенные по Эрлангу, могут быть сгенерированы из равномерно распределенных случайных чисел



по следующей формуле:



Однако мы возьмем более эффективный с точки зрения вычисления алгоритм:

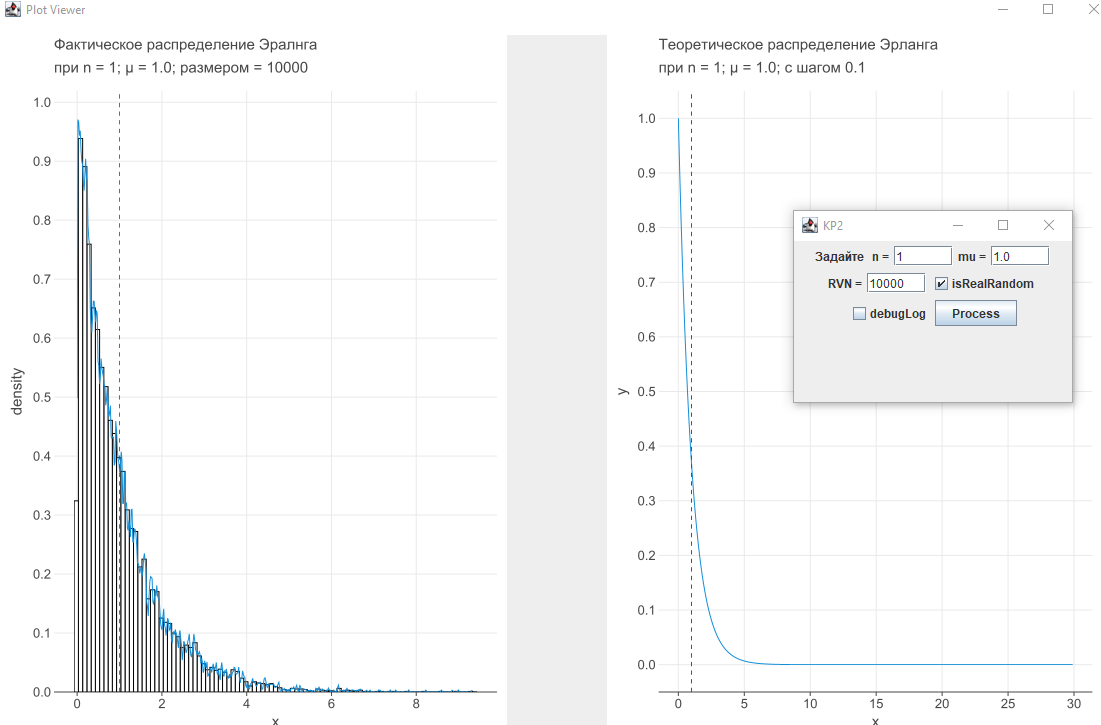
1. Сгенерировать n равномерных (0, 1) случайных величин U1, . . . , Un;
2. Установить X = − ln(U1U2 · · · Un)/µ.

Реализация на kotlin в работе выглядит так:

*/\*\*  
 \** ***@param*** *n = shape, в некоторых источниках обозначается k  
 \** ***@param*** *mu = rate или μ, в некоторых источниках не мю, а лямбда λ  
 \** ***@return*** *Случайная величина из распределения Эрланга  
 \*/*operator fun invoke(n: Int, mu: Double): Double {  
 var multiplicationResult = 1.0  
 val listU = *List*(n) **{** randomEventGenerator.invoke(Pa = 1.0).randomNumber **}** listU.*map* **{** multiplicationResult \*= **it }** val result = -*ln*(multiplicationResult) / mu  
 if (isDebug) {  
 listU.*forEachIndexed* **{** index, d **->** *println*("U${index + 1} = $d")  
 **}** *println*("multiplicationResult = $multiplicationResult")  
 *println*("Случайная величина из распределения Эрланга = $result")  
 }  
 return result  
}

## Оценки

Выведем на графики массив данных размером 10000, а также теоретическое распределение с n = 1; μ = 1.0



### Точечные и интервальные оценки

Подсчитанные точечные оценки выведем в консоль:

Мат.ожидание = 1.0

Несмещенная состоятельная оценка математического ожидания или выборочное среднее= 1.0034489272560065

Несмещенная состоятельная оценка дисперсии s02 = 1.0178310617313706

Смещенная состоятельная оценка дисперсии = 1.0177292786251975

Несмещенная состоятельная оценка дисперсии s12 = 1.0177411737244146

Состоятельная оценка среднеквадратичного отклонения = 1.0088761379532032

Выборочный начальный момент 1-го порядка: 1.0034489272560065

Выборочный центральный момент 1-го порядка: 2.2448709557920667E-17

Выборочный начальный момент 2-го порядка: 2.024639028236435

Выборочный центральный момент 2-го порядка: 1.0177292786251975

Выборочный начальный момент 3-го порядка: 6.141082428336711

Выборочный центральный момент 3-го порядка: 2.0669818616252646

Выборочный начальный момент 4-го порядка: 24.835155796019116

Выборочный центральный момент 4-го порядка: 9.376276426659864

Выборочный начальный момент 5-го порядка: 124.62673543096018

Выборочный центральный момент 5-го порядка: 45.47069832443826

По методу моментов разделим оценку математического ожидания на оценку дисперсии, получим

mu` = 0.9858698216078221

### Проверка соответствия экспериментального закона распределения теоретическому

Будем использовать Критерий хи-квадрат К. Пирсона

1. Выдвинем гипотезу

Изображение выглядит как текст

Автоматически созданное описание

1. Вычислим значения критерия xi2 = 96.12651167822523
2. Проверим выполнение контрольного соотношения Изображение выглядит как текст

   Автоматически созданное описание

mod (1- sumpi) = 0.008012971705627536

Это меньше 0.01?: true

1. Для динамической генерации табличного значения критерия будем использовать формулу Excel CHISQ.INV.RT или ХИ2.ОБР.ПХ

Для этого сгенерируем Excel файл с нашими параметрами  
Изображение выглядит как текст

Автоматически созданное описание

Количество степеней свободы будем считать как 

1. Так как фактическое значение не больше табличного, у нас нет оснований отклонять гипотезу H0.

# Листинг кода

## Главный файл

package kr\_two  
  
import kr\_one.RandomEventGenerator  
import org.jetbrains.letsPlot.GGBunch  
import org.jetbrains.letsPlot.geom.geomDensity  
import org.jetbrains.letsPlot.geom.geomHistogram  
import org.jetbrains.letsPlot.geom.geomLine  
import org.jetbrains.letsPlot.geom.geomVLine  
import org.jetbrains.letsPlot.label.ggtitle  
import org.jetbrains.letsPlot.letsPlot  
import java.awt.FlowLayout  
import java.math.BigDecimal  
import java.math.RoundingMode  
import javax.swing.\*  
import kotlin.math.\*  
  
*/\*\*  
 \* Получаем U1...Un случайных величин на интервале 0..1  
 \* Перемножаем их друг на друга  
 \* Случайная величина из распределения Эрланга равна  
 \* x = -ln(U1\*U2\*...\*Un)/mu  
 \** ***@see*** *<a href="https://www.win.tue.nl/~marko/2WB05/lecture8.pdf">математическое обоснование, страница 9</a>  
 \*/*class ErlangDistributionGenerator(  
 val isRealRandom: Boolean,  
 val isDebug: Boolean  
) {  
 //отсюда будем получать x (0-1)  
 private val randomEventGenerator = RandomEventGenerator(isRealRandom = isRealRandom, isDebug = false)  
  
 */\*\*  
 \** ***@param*** *n = shape, в некоторых источниках обозначается k  
 \** ***@param*** *mu = rate или μ, в некоторых источниках не мю, а лямбда λ  
 \** ***@return*** *Случайная величина из распределения Эрланга  
 \*/* operator fun invoke(n: Int, mu: Double): Double {  
 var multiplicationResult = 1.0  
 val listU = *List*(n) **{** randomEventGenerator.invoke(Pa = 1.0).randomNumber **}** listU.*map* **{** multiplicationResult \*= **it }** val result = -*ln*(multiplicationResult) / mu  
 if (isDebug) {  
 listU.*forEachIndexed* **{** index, d **->** *println*("U${index + 1} = $d")  
 **}** *println*("multiplicationResult = $multiplicationResult")  
 *println*("Случайная величина из распределения Эрланга = $result")  
 }  
 return result  
 }  
  
 */\*\*  
 \* Probability density function  
 \** ***@see*** *<a href="https://en.wikipedia.org/wiki/Erlang\_distribution"> erlang wiki </a>  
 \*/* fun calculateTheoretical(n: Int, mu: Double, x: Double): Double =  
 (mu.*pow*(n) \* x.*pow*(n - 1) \* *E*.*pow*(-mu \* x)) / factorial(n - 1)  
  
  
 private fun factorial(num: Int) = if (num == 0) {  
 1  
 } else {  
 (1..num).*reduce* **{** a, b **->** a \* b **}** }  
}  
  
class KrTwoInputGetter : JFrame("КР2") {  
 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 **{** KrTwoInputGetter() **}** }  
 }  
  
 private val button = JButton("Process")  
 private val label = JLabel("Задайте ")  
  
 private val labelN = JLabel("n =")  
 private val textFieldN = JTextField(KrTwoParamsSaver.loadKrTwoParams().n.toString(), 5)  
  
 private val labelMu = JLabel("mu =")  
 private val textFieldMu = JTextField(KrTwoParamsSaver.loadKrTwoParams().mu.toString(), 5)  
 private val labelRVN = JLabel("RVN =")  
 private val textFieldRVN = JTextField(KrTwoParamsSaver.loadKrTwoParams().RVN.toString(), 5)  
  
 private val cbRandom = JCheckBox("isRealRandom").*apply* **{** *isSelected* = KrTwoParamsSaver.loadKrTwoParams().realRandom  
 **}** private val cbDebug = JCheckBox("debugLog").*apply* **{** *isSelected* = KrTwoParamsSaver.loadKrTwoParams().debug  
 **}** init {  
 *layout* = FlowLayout()  
 *defaultCloseOperation* = *EXIT\_ON\_CLOSE* setLocationRelativeTo(null)  
 *isVisible* = true  
 setSize(295, 200)  
  
 button.addActionListener **{** processButtonClick()  
 **}** add(label)  
 add(labelN)  
 add(textFieldN)  
 add(labelMu)  
 add(textFieldMu)  
 add(labelRVN)  
 add(textFieldRVN)  
 add(cbRandom)  
 add(cbDebug)  
 add(button)  
 }  
  
 private fun processButtonClick() {  
 try {  
 val n = textFieldN.*text*.*toInt*()  
 val mu = textFieldMu.*text*.*toDouble*()  
 val RVN = textFieldRVN.*text*.*toInt*()  
 val isRealRandom = cbRandom.*isSelected* val isDebug = cbDebug.*isSelected* val erlangDistributionGenerator =  
 ErlangDistributionGenerator(isRealRandom = isRealRandom, isDebug = isDebug)  
  
 *drawGraphs*(n, mu, RVN, erlangDistributionGenerator)  
  
 KrTwoParamsSaver.saveKrTwoParams(  
 KrTwoParams(  
 n, mu, RVN, isRealRandom, isDebug  
 )  
 )  
  
  
 } catch (e: Exception) {  
 JOptionPane.showMessageDialog(  
 this@KrTwoInputGetter, """  
 Ошибка во время процессинга:  
 ${e.message}  
 """.*trimIndent*()  
 )  
 }  
 }  
}  
  
private fun drawGraphs(n: Int, mu: Double, RVN: Int, erlangDistributionGenerator: ErlangDistributionGenerator) {  
  
 val listTheoreticalSize = 300  
 val theoreticalXFrequency = 0.1  
 val listTheoreticalX = *List*(listTheoreticalSize) **{ it** \* theoreticalXFrequency **}** val listTheoreticalY = listTheoreticalX  
 .*map* **{** erlangDistributionGenerator.calculateTheoretical(n, mu, **it**) **}** val dataTheoretical = *mapOf*<String, List<\*>>(  
 "x" *to* listTheoreticalX,  
 "y" *to* listTheoreticalY  
 )  
 val pTheoretical = *letsPlot*(dataTheoretical) **{** x = "x"; y = "y" **}** val plotTheoretical = (pTheoretical +  
 geomLine **{** y = "y" **}** +  
 *ggtitle*(  
 "Теоретическое распределение Эрланга",  
 "при n = $n; μ = $mu; с шагом $theoreticalXFrequency"  
 ) +  
 //Среднее значение должно быть равно n/mu  
 geomVLine(  
 xintercept = (n / mu),  
 color = "red",  
 linetype = "dashed"  
 ))  
  
 val data = *mapOf*<String, List<Double>>(  
 "x" *to List*(RVN) **{** erlangDistributionGenerator.invoke(n, mu) **}** )  
  
 val p = *letsPlot*(data) **{** x = "x" **}** val plot = (p +  
 geomHistogram(binWidth = 0.1, color = "black", fill = "white") **{** y = "..density.." **}** +  
 //default adjust is 1  
 geomDensity(linetype = 1, adjust = 0.1) +  
 *ggtitle*(  
 "Фактическое распределение Эралнга",  
 "при n = $n; μ = $mu; размером = $RVN"  
 ) + geomVLine(xintercept = (data["x"] as List<Double>).*average*(), color = "red", linetype = "dashed"))  
  
 GGBunch()  
 .addPlot(  
 plot, 0, 0, 500, 700  
 )  
 .addPlot(  
 plotTheoretical, 600, 0, 500, 700  
 )  
 .show()  
  
 */\*\*  
 \* Все оценки посчитаны по твимс 2 часть 2012: https://www.bsuir.by/m/12\_100229\_1\_90172.pdf  
 \* ~~~~~~~~~~~~~~~~~~~~~~Точечные оценки~~~~~~~~~~~~~~~~~~~~~~~~~~  
 \*/* val mx = n / mu  
 *println*("Мат.ожидание = $mx")  
 val dx = data["x"]!!.*average*()  
 *println*("Несмещенная состоятельная оценка математического ожидания или выборочное среднее= $dx")  
 val s02 = 1.0 / (RVN - 1) \* data["x"]!!.*fold*(0.0) **{** sum, element **->** sum + (element - dx).*pow*(2) **}** *println*("Несмещенная состоятельная оценка дисперсии s02 = $s02")  
 val s2 = 1.0 / RVN \* data["x"]!!.*fold*(0.0) **{** sum, element **->** sum + (element - dx).*pow*(2) **}** *println*("Смещенная состоятельная оценка дисперсии = $s2")  
 val s12 = 1.0 / RVN \* data["x"]!!.*fold*(0.0) **{** sum, element **->** sum + (element - mx).*pow*(2) **}** *println*("Несмещенная состоятельная оценка дисперсии s12 = $s12")  
 val s0 = *sqrt*(s02)  
 *println*("Состоятельная оценка среднеквадратичного отклонения = $s0")  
 *repeat*(5) **{** k **->** val ak = 1.0 / RVN \* data["x"]!!.*fold*(0.0) **{** sum, element **->** sum + element.*pow*(k + 1) **}** val muk = 1.0 / RVN \* data["x"]!!.*fold*(0.0) **{** sum, element **->** sum + (element - dx).*pow*(k + 1) **}** *println*("Выборочный начальный момент ${k + 1}-го порядка: $ak")  
 *println*("Выборочный центральный момент ${k + 1}-го порядка: $muk")  
 **}** */\*\*  
 \* Метод моментов  
 \*/* val lambdaMoments = dx / s02  
 *println*("По методу моментов разделим оценку математического ожидания на оценку дисперсии, получим")  
 *println*("mu` = $lambdaMoments")  
 */\*\*  
 \* ~~~~~~~~~~~~~~~~~~~~~~Интервальные оценки~~~~~~~~~~~~~~~~~~~~~~~~~~  
 \*/* //стъюдент до 1000 http://old.exponenta.ru/educat/referat/XIkonkurs/student5/tabt-st.pdf  
 //для 99 это 2.6264055  
 // для 999 это 2.5807596  
 val left = dx - (s02 \* 2.5807596) / *sqrt*(RVN - 1.0)  
 val right = dx + (s02 \* 2.5807596) / *sqrt*(RVN - 1.0)  
 *println*(  
 "Доверительный интервал для математического ожидания при выборке размером 1000 и уровне значимости 0.99: \n" +  
 "$left <= $mx <= $right"  
 )  
  
 */\*\*  
 \* ~~~~~~~~~~~~~~~~~~~~~~Соответствие закона распределения распределению Эрланга~~~~~~~~~~~~~~~~~~~~~~~~~~  
 \*/* val roundedDataX: List<Double> = (plot.data!!["x"] as List<Double>)  
 ////округляем до 1 знака после запятой, при округлении до 2 график выглядит плохо  
 .*map* **{** BigDecimal(**it**).setScale(1, RoundingMode.*HALF\_EVEN*).toDouble() **}** //.map { it.toInt().toDouble() }  
 //Сюда будем сплюсовывать density при каждом совпадении  
 val calculatedDensity = listTheoreticalY.*map* **{** 0.0 **}**.*toMutableList*()  
  
 roundedDataX.*forEachIndexed* **{** index, d **->** calculatedDensity[listTheoreticalX.*closestIndex*(d)] += 1.0 / RVN  
 **}**// val dataWeird = mapOf<String, List<\*>>(  
// "x" to listTheoreticalX,  
// "y" to calculatedDensity  
// )  
// val pWeird = letsPlot(dataWeird) { x = "x"; y = "y" }  
// val plotWeird = (pWeird +  
// geomLine { y = "y" }).show()  
  
 //теперь посчитаем полный хи квадат. Нужно сравнить фактические calculatedDensity с теоретическими  
 var sum = 0.0  
 var sumControl = 0.0  
 //выравниваем теоретический скейл с подсчитанным  
 val listTheoreticalYAligned = listTheoreticalY.*map* **{ it** / 10 **}** //выведем самое большое несовпадение  
 var biggestViolation = 0 *to* 0.0  
 calculatedDensity.*forEachIndexed* **{** index, d **->** if (d != 0.0 && index != 0) {  
 val indexDiff = *abs*(d - listTheoreticalY[index])  
 if (indexDiff >= biggestViolation.second) {  
 biggestViolation = index *to* indexDiff  
 }  
 }  
 **}** *println*("biggest violation index = ${biggestViolation.first} diff = ${biggestViolation.second}")  
 *println*("теоретическое значение в этом индексе x = ${listTheoreticalX[biggestViolation.first]} y = ${listTheoreticalYAligned[biggestViolation.first]}")  
 *println*("экспериментальное значение после округлений = ${calculatedDensity[biggestViolation.first]}")  
  
 var M = 0  
 calculatedDensity.*forEachIndexed* **{** index, d **->** if (d != 0.0 && listTheoreticalYAligned[index] != 0.0 && index != 0) {  
 sum += (listTheoreticalYAligned[index] - d).*pow*(2) / listTheoreticalYAligned[index]  
 sumControl += listTheoreticalYAligned[index]  
 M += 1  
 }  
 **}** val xi2 = sum \* RVN  
 val sumControlCalculated = *abs*(1 - sumControl)  
 *println*("После вычисления всех вероятностей pi проверим, выполняется ли контрольное соотношение: ${1 - sumControl}")  
 *println*("mod (1- sumpi) = $sumControlCalculated")  
 *println*("Это меньше 0.01?: ${sumControlCalculated <= 0.01}")  
 *println*("xi2 = $xi2")  
 val degreesOfFreedom = M - 1 - 2  
 *println*("degrees of freedom = $degreesOfFreedom")  
 CriticalChiSquareExcelFileGenerator().invoke(0.001, degreesOfFreedom, xi2)  
}  
  
fun List<Double>.closestIndex(value: Double): Int {  
 val closest = *minBy* **{** *abs*(value - **it**) **}** val result = this.indexOf(closest)  
 return result  
}

## Сохранение параметров ввода

package kr\_two  
  
import java.util.prefs.Preferences  
  
object KrTwoParamsSaver {  
 private fun prefNode(): Preferences = Preferences.userRoot().node("KR\_TWO")  
  
 fun loadKrTwoParams(): KrTwoParams {  
 return prefNode().*run* **{** KrTwoParams(  
 n = getInt(KrTwoParams.N\_KEY, 2),  
 mu = getDouble(KrTwoParams.MU\_KEY, 2.0),  
 RVN = getInt(KrTwoParams.RVN\_KEY, 10000),  
 realRandom = getBoolean(KrTwoParams.RANDOM\_KEY, true),  
 debug = getBoolean(KrTwoParams.DEBUG\_KEY, false)  
 )  
 **}** }  
  
 fun saveKrTwoParams(params: KrTwoParams) {  
 prefNode().*apply* **{** putInt(KrTwoParams.N\_KEY, params.n)  
 putDouble(KrTwoParams.MU\_KEY, params.mu)  
 putInt(KrTwoParams.RVN\_KEY, params.RVN)  
 putBoolean(KrTwoParams.RANDOM\_KEY, params.realRandom)  
 putBoolean(KrTwoParams.DEBUG\_KEY, params.debug)  
 **}** }  
}  
  
data class KrTwoParams(  
 val n: Int,  
 val mu: Double,  
 val RVN: Int,  
 val realRandom: Boolean,  
 val debug: Boolean  
) {  
 companion object {  
 const val N\_KEY = "n"  
 const val MU\_KEY = "mu"  
 const val RVN\_KEY = "rvn"  
 const val RANDOM\_KEY = "realRandom"  
 const val DEBUG\_KEY = "debug"  
 }  
}

## Определение операционной системы

package kr\_two  
  
import java.awt.Desktop  
import java.io.File  
import java.util.\*  
  
  
object OSDetector {  
 val isWindows: Boolean  
 val isLinux: Boolean  
 val isMac: Boolean  
 init {  
 val os = System.getProperty("os.name").*lowercase*(Locale.getDefault())  
 isWindows = os.*contains*("win")  
 isLinux = os.*contains*("nux") || os.*contains*("nix")  
 isMac = os.*contains*("mac")  
 }  
  
 fun openWithSystem(file: File): Boolean {  
 return try {  
 if (isWindows) {  
 Runtime.getRuntime().exec(  
 *arrayOf*(  
 "rundll32", "url.dll,FileProtocolHandler",  
 file.*absolutePath* )  
 )  
 true  
 } else if (isLinux || isMac) {  
 Runtime.getRuntime().exec(  
 *arrayOf*(  
 "/usr/bin/open",  
 file.*absolutePath* )  
 )  
 true  
 } else {  
 // Unknown OS, try with desktop  
 if (Desktop.isDesktopSupported()) {  
 Desktop.getDesktop().open(file)  
 true  
 } else {  
 false  
 }  
 }  
 } catch (e: Exception) {  
 e.printStackTrace(System.*err*)  
 false  
 }  
 }  
}

## Генерация excel-файла для сравнения xi2

package kr\_two  
  
import org.apache.poi.hssf.usermodel.HSSFWorkbook  
import java.io.File  
import java.io.FileOutputStream  
  
class CriticalChiSquareExcelFileGenerator {  
 */\*\*  
 \** ***@param*** *alpha = уровень значимости  
 \** ***@param*** *degreesOfFreedom = число степеней свободы  
 \** ***@param*** *chiSqCalculated = подсчитанное хи квадрат  
 \** ***@result*** *запуск эксель файла с посчитанным/критическим значением  
 \*/* operator fun invoke(alpha: Double, degreesOfFreedom: Int, chiSqCalculated: Double) {  
 val workbook = HSSFWorkbook()  
 val sheet = workbook.createSheet(SHEET\_NAME)  
 sheet.setColumnWidth(0, 27\*256)  
  
 sheet.createRow(0).*apply* **{** createCell(0).setCellValue("Уровень значимости альфа =")  
 createCell(1).setCellValue(alpha)  
 **}** sheet.createRow(1).*apply* **{** createCell(0).setCellValue("Число степеней свободы k =")  
 createCell(1).setCellValue(degreesOfFreedom.toString())  
 **}** sheet.createRow(2).*apply* **{** createCell(0).setCellValue("Критическое хиКвадрат =")  
 //CHISQ.INV.RT == ХИ2.ОБР.ПХ в русском экселе  
 createCell(1).*cellFormula* = "\_xlfn.CHISQ.INV.RT(B1,B2)"  
 **}** sheet.createRow(3).*apply* **{** createCell(0).setCellValue("Подсчитанное хиКвадрат =")  
 createCell(1).setCellValue(chiSqCalculated)  
 **}** val path = String.*format*(PATH, System.currentTimeMillis().toString())  
 FileOutputStream(path).*use* **{** fileOut **->** workbook.write(fileOut)  
 workbook.close()  
 fileOut.close()  
 **}** OSDetector.openWithSystem(File(path))  
 }  
  
 companion object {  
 private const val PATH = "src/main/kotlin/kr\_two/experiments/%s\_JavaBooks.xls"  
 private const val SHEET\_NAME = "ChiSquareDistCalculator"  
 }  
}

# Источники

1. Распределение Эрланга <https://en.wikipedia.org/wiki/Erlang_distribution>
2. ТЕОРИЯ ВЕРОЯТНОСТЕЙ И МАТЕМАТИЧЕСКАЯ СТАТИСТИКА, часть 2 <https://www.bsuir.by/m/12_100229_1_90172.pdf>
3. Генерация случайных чисел <https://www.win.tue.nl/~marko/2WB05/lecture8.pdf>
4. Преобразование равномерно распределенной случайной величины в нормально распределенную <https://habr.com/ru/post/208684/>
5. Работа с excel в java/kotlin <https://poi.apache.org/>
6. Краткий гайд по poi <https://poi.apache.org/components/spreadsheet/quick-guide.html>