Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Моделирование

ОТЧЕТ

по лабораторной работе № 8

на тему

МЕТОД МОНТЕ-КАРЛО,

РЕШЕНИЕ ЗАДАЧ МОДЕЛИРОВАНИЯ НА ОСНОВЕ АЛГОРИТМОВ ИМИТАЦИИ СЛУЧАЙНЫХ СОБЫТИЙ И ВЕЛИЧИН,

ВАРИАНТ № 6-6-1

Студенты: А.В. Гуринович

Проверила: Ю.О. Герман

МИНСК 2022

# 1. Цель работы

Изучить метод Монте–Карло и решение задач моделирования на основе алгоритмов имитации случайных событий и величин.

# 2. Задание

## 2.1 Имитация случайных событий

* разработать алгоритм имитации для решения задачи на основе метода Монте-Карло.
* выполнить два испытания алгоритма, используя случайные числа из учебного пособия (приложение 1).
* разработать имитационную модель на языке VBA для решения задачи.

Библиотека предоставляет платную услугу: выдачу книг из читального зала на дом. Читателю выдается не более четырех книг. Из опыта работы библиотеки известно, что примерно 60% читателей, пользующихся данной услугой, берут одну книгу, 20% - две, 15% - три, 5% - четыре книги.

Книги выдаются на срок не более десяти дней. Книги могут быть выданы только на целое количество дней (т.е. нельзя взять книгу, например, на полтора дня). Читатель берет все книги на один срок (т.е. по одному заказу нельзя, например, взять одну книгу на один день, а еще одну – на три дня). Из опыта работы библиотеки известно, что читатели берут книги на любой срок (от одного до десяти дней) одинаково часто.

Плата за пользование книгой из общего читального зала составляет 12 д.е. в день, из специализированного читального зала – 15 д.е. в день. Из всех книг, которые читатели берут на дом, примерно 40% составляют книги из общего читального зала, 60% - из специализированного. Читатель может брать книги как из одного читального зала, так и из разных.

Найти:

а) средний размер платы одного читателя;

б) вероятность того, что все книги, взятые читателем, будут из специализированного читального зала.

## 2.2 Имитация случайных величин: метод обратных функций, метод исключений

Некоторая непрерывная случайная величина *X* задана плотностью распределения.

Требуется:

1) решить задачу имитации заданной случайной величины методом обратных функций. Для этого:

* разработать алгоритм имитации заданной случайной величины на основе метода обратных функций;
* используя случайные числа из учебного пособия (приложение 1), выполнить имитацию двух значений заданной случайной величины;
* выполнить проверку разработанного алгоритма, используя построение гистограммы;

2) решить задачу имитации заданной случайной величины методом исключений, выполнив те же действия, что и в пункте 1);

3) используя один из разработанных алгоритмов, выполнить имитацию заданной случайной величины и найти для нее: а) среднее значение; б) вероятность того, что величина *X* превысит некоторое заданное значение (выбрать это значение самостоятельно).

200

400

800

1200

x

f(x)

0.001

0.0005

0.0015

## 2.3 Решение задач моделирования объектов на основе имитации случайных величин

* разработать алгоритм имитации для решения задачи на основе метода Монте-Карло.
* выполнить одно испытание алгоритма, используя случайные числа из учебного пособия (приложение 1).
* разработать имитационную модель на любом языке для решения задачи.

**2.3.1 Первая часть**

Предприятие выпускает комплекты инструментов по заказам. Количество инструментов в комплекте – от 3 до 6. Инструменты могут быть стандартными или высокоточными; комплект может содержать стандартные и высокоточные инструменты в любой комбинации. Время изготовления одного стандартного инструмента составляет от 30 мин до одного часа, высокоточного – от 40 мин до двух часов. Время испытания инструмента – экспоненциальная случайная величина со средним значением 10 мин для стандартных инструментов, 15 мин – для высокоточных.

Найти: а) среднее время изготовления и испытания комплекта; б) вероятность того, что комплект будет состоять только из высокоточных инструментов; в) вероятность того, что время изготовления и испытания комплекта превысит 24 часа.

**2.3.2 Вторая часть**

В мастерской выполняется проверка и ремонт некоторых приборов. Прибор состоит из пяти схем. Вероятность неисправности схемы - 0,1. Работа с прибором включает следующие операции:

* проверка: от 2 до 5 мин на каждую схему;
* замена неисправных схем: время замены одной схемы – гауссовская случайная величина со средним значением 5 мин и стандартным отклонением 0,5 мин;
* настройка. Время настройки – экспоненциальная случайная величина со средним значением 6 мин, если ни одна схема не заменялась, и 10 мин – если потребовалась замена хотя бы одной схемы.

Найти:

а) среднее время работы с прибором;

б) вероятность того, что в приборе потребуется замена хотя бы одной схемы;

в) среднее количество схем, заменяемых в одном приборе.

## 3 Ход работы

Общие для всех трёх задач переменные и функции:

import Foundation

extension Double {

var string: String {

String((self \* 100).rounded()/100)

}

}

extension Array<Double> {

func average() -> Double {

return self.reduce(0, +) / Double(self.count)

}

}

var firstTestProbabilities = [0.0795, 0.3780, 0.0593, 0.7602, 0.2847, 0.8197, 0.6133, 0.5766, 0.9595, 0.0981,0.2410, 0.5962, 0.2978, 0.6458, 0.9762, 0.0523, 0.4523, 0.8153, 0.4286, 0.8400,0.8754, 0.5900, 0.3421, 0.7919, 0.1204, 0.7492, 0.3792, 0.7198, 0.4822, 0.8215,0.6058, 0.5321, 0.8064, 0.0540, 0.9046, 0.5778, 0.0043, 0.6218, 0.0563, 0.5934,0.3716, 0.3549, 0.0112, 0.8438, 0.3233, 0.5052, 0.7647, 0.2125, 0.6382, 0.5239,0.0644, 0.7612, 0.5707, 0.2759, 0.8396, 0.9127, 0.7652, 0.5576, 0.2914, 0.5682,0.7195, 0.1977, 0.2245, 0.3979, 0.0652, 0.0136, 0.2148, 0.8663, 0.7802, 0.9190, 0.5994, 0.8691, 0.0564, 0.9681, 0.7945, 0.1137, 0.3537, 0.8137, 0.8484, 0.4385, 0.9561, 0.2147, 0.9299, 0.1944, 0.0147, 0.1935, 0.1453, 0.2754, 0.0517, 0.7645, 0.4403, 0.1689, 0.0453, 0.1344, 0.5162, 0.0540, 0.2084, 0.3725, 0.3741, 0.2042]

let secondTestProbabilities = [0.2058, 0.0720, 0.2383, 0.7248, 0.2661, 0.1888, 0.8946, 0.1636, 0.9949, 0.3229, 0.5869, 0.5937, 0.3465, 0.9940, 0.1708, 0.3062, 0.1309, 0.6818, 0.9790, 0.7400, 0.3931, 0.0718, 0.7680, 0.9890, 0.0435, 0.9811, 0.5677, 0.5516, 0.0583, 0.5912, 0.7759, 0.5792, 0.0555, 0.6824, 0.2781, 0.5160, 0.6977, 0.1059, 0.5570, 0.6978, 0.8090, 0.4302, 0.6869, 0.4293, 0.9700, 0.2145, 0.2849, 0.8945, 0.7895, 0.3605, 0.4983, 0.8773, 0.4004, 0.2839, 0.1111, 0.6815, 0.6258, 0.8406, 0.8576, 0.4549, 0.4122, 0.4560, 0.2445, 0.9876, 0.3876, 0.1894, 0.1342, 0.6081, 0.1433, 0.6191, 0.3483, 0.5023, 0.4718, 0.5838, 0.5971, 0.1554, 0.5865, 0.8211, 0.5267, 0.0068, 0.3178, 0.5673, 0.8289, 0.7917, 0.3653, 0.8804, 0.0996, 0.4143, 0.7519, 0.4857, 0.1757, 0.2741, 0.5951, 0.0243, 0.4258, 0.4118, 0.9968, 0.7917, 0.6760, 0.7910]

let thirdTestProbabilities = [0.6016, 0.2608, 0.2722, 0.4435, 0.3890, 0.8999, 0.8378, 0.0933, 0.0203, 0.5364, 0.5291, 0.7498, 0.4549, 0.8937, 0.9365, 0.1678, 0.6006, 0.1885, 0.5667, 0.5623, 0.4291, 0.9307, 0.8907, 0.4776, 0.1691, 0.6589, 0.9766, 0.8555, 0.2026, 0.5771, 0.9220, 0.9723, 0.3733, 0.3401, 0.8118, 0.0978, 0.0913, 0.7856, 0.6641, 0.5063, 0.3362, 0.8619, 0.7735, 0.7002, 0.4086, 0.6974, 0.6014, 0.3553, 0.5268, 0.4749, 0.2861, 0.1074, 0.8348, 0.1517, 0.0206, 0.2963, 0.6904, 0.0079, 0.5070, 0.3674, 0.7993, 0.6758, 0.1273, 0.5915, 0.0321, 0.0516, 0.6761, 0.0542, 0.5205, 0.2080, 0.2886, 0.2257, 0.8240, 0.6699, 0.2291, 0.5493, 0.5030, 0.6703, 0.3374, 0.8210, 0.1792, 0.4387, 0.6405, 0.7603, 0.1866, 0.4258, 0.3329, 0.2838, 0.8122, 0.2138, 0.6629, 0.2038, 0.0407, 0.4089, 0.0196, 0.8947, 0.4352, 0.1265, 0.2156, 0.4300]

var allProbabilities: [Double] = []

allProbabilities.append(contentsOf: firstTestProbabilities)

allProbabilities.append(contentsOf: secondTestProbabilities)

allProbabilities.append(contentsOf: thirdTestProbabilities)

### 3.1 Имитация случайных событий

enum BookType {

case regular

case specialised

init(\_ probability: Double) {

if probability <= 0.4 {

self = .regular

} else {

self = .specialised

}

}

var cost: Int {

switch self {

case .regular:

return 12

case .specialised:

return 15

}

}

}

class LibraryOrder {

let duration: Int

let books: [BookType]

init(probabilities: inout [Double]) {

self.duration = Int(probabilities.removeFirst() \* 10) + 1

switch probabilities.removeFirst() {

case 0..<0.6:

books = [BookType(probabilities.removeFirst())]

case 0.6..<0.8:

books = [BookType(probabilities.removeFirst()),

BookType(probabilities.removeFirst())]

case 0.8..<0.95:

books = [BookType(probabilities.removeFirst()),

BookType(probabilities.removeFirst()),

BookType(probabilities.removeFirst())]

default:

books = [BookType(probabilities.removeFirst()),

BookType(probabilities.removeFirst()),

BookType(probabilities.removeFirst()),

BookType(probabilities.removeFirst())]

}

}

func averageCost() -> Double {

let costs = self.books.map { $0.cost }

return Double(costs.reduce(0, +)) / Double(costs.count)

}

}

func calculateLibraryOrders(probabilities: [Double]) {

var probabilities = probabilities

var libraryOrders: [LibraryOrder] = []

while(probabilities.count > 6) {

libraryOrders.append(LibraryOrder(probabilities: &probabilities))

}

print("For \(libraryOrders.count) operations:")

let allCosts = libraryOrders.map { $0.averageCost() }

let averageCost = allCosts.reduce(0.0, +) / Double(allCosts.count)

print("Average cost: \(averageCost)")

let booksArrays = libraryOrders.map { $0.books }

let onlySpecialisedCount = booksArrays.filter { $0.allSatisfy { $0 == .specialised } }.count

print("Probability of only specialised books in order: \(Double(onlySpecialisedCount) / Double(booksArrays.count) )\n" )

}

print("10000 random values:")

calculateLibraryOrders(probabilities: (0...10000).map{ \_ in Double.random(in: 0...1)})

///For 82 operations:

///Average cost: 13.753048780487806

///Probability of only specialised books in order: 0.43902439024390244

///

print("\(allProbabilities.count) random values:")

calculateLibraryOrders(probabilities: allProbabilities)

///For 2742 operations:

///Average cost: 13.821207148067105

///Probability of only specialised books in order: 0.47957695113056165

## 3.2 Имитация случайных величин: метод обратных функций, метод исключений

Запишем уравнения плотности распределения:

Теперь проведём расчёт выражений для функции распределения на всех интервалах:

Таким образом, функция распределения имеет вид:

Запишем уравнения

Из этих уравнений выражаем x:

Диаграммы для методов обратных функций и исключений соответственно:



Рисунок 3.1 – диаграмма для метода обратных функций



Рисунок 3.2 – диаграмма для метода исключений

Относящиеся к этой части задания код программы:

enum ToolType: CaseIterable {

case standard

case precise

init() {

if Bool.random() {

self = .standard

} else {

self = .precise

}

}

}

class Tool {

var type = ToolType()

var assemblyTime: Double

var testingTime: Double

init() {

switch type {

case .standard:

assemblyTime = Double.random(in: 30.0...60.0)

case .precise:

assemblyTime = Double.random(in: 40.0...120.0)

}

switch type {

case .standard:

testingTime = -10 \* log(Double.random(in: 0.0...1.0))

case .precise:

testingTime = -15 \* log(Double.random(in: 0.0...1.0))

}

}

}

class Toolbox {

var tools: [Tool] = (0...Int.random(in: 3...6)).map { \_ in Tool() }

var assemblyTime: Double

var testingTime: Double

var allTime: Double

init() {

assemblyTime = tools.map { $0.assemblyTime }.reduce(0, +)

testingTime = tools.map { $0.testingTime }.reduce(0, +)

allTime = assemblyTime + testingTime

}

}

func calculateToolboxes() {

print("\nPart 3.1:")

let toolboxes = (1...100000).map { \_ in Toolbox() }

let averageAssemblyTime = toolboxes.map { $0.assemblyTime }.average()

print("Average toolbox assembly time: \(averageAssemblyTime)")

let averageTestingTime = toolboxes.map { $0.testingTime }.average()

print("Average toolbox testing time: \(averageTestingTime)")

let onlyPreciseInstrumentsProbability = Double(toolboxes.filter { $0.tools.allSatisfy { $0.type == .precise } }.count) / Double(toolboxes.count)

print("Probability of only precise instruments in toolbox: \(onlyPreciseInstrumentsProbability)")

let moreThenDayTimeProbability = Double(toolboxes.filter { $0.allTime >= 1440 }.count) / Double(toolboxes.count)

print("Probability of 1440 minutes and more of assembly and testing toolbox: \(moreThenDayTimeProbability)")

}

calculateToolboxes()

///For 100 000 toolboxes

///Average toolbox assembly time: 343.83379019630075

///Average toolbox testing time: 69.06121370956292

///Probability of only precise instruments in toolbox: 0.02893

///Probability of 1440 minutes and more of assembly and testing toolbox: 0.0

## 3.3 Решение задач моделирования объектов на основе имитации случайных величин

### 3.3.1 Первая часть

enum ToolType: CaseIterable {

case standard

case precise

init() {

if Bool.random() {

self = .standard

} else {

self = .precise

}

}

}

class Tool {

var type = ToolType()

var assemblyTime: Double

var testingTime: Double

init() {

switch type {

case .standard:

assemblyTime = Double.random(in: 30.0...60.0)

case .precise:

assemblyTime = Double.random(in: 40.0...120.0)

}

switch type {

case .standard:

testingTime = -10 \* log(Double.random(in: 0.0...1.0))

case .precise:

testingTime = -15 \* log(Double.random(in: 0.0...1.0))

}

}

}

class Toolbox {

var tools: [Tool] = (0...Int.random(in: 3...6)).map { \_ in Tool() }

var assemblyTime: Double

var testingTime: Double

var allTime: Double

init() {

assemblyTime = tools.map { $0.assemblyTime }.reduce(0, +)

testingTime = tools.map { $0.testingTime }.reduce(0, +)

allTime = assemblyTime + testingTime

}

}

func calculateToolboxes() {

print("\nPart 3.1:")

let toolboxes = (1...100000).map { \_ in Toolbox() }

let averageAssemblyTime = toolboxes.map { $0.assemblyTime }.average()

print("Average toolbox assembly time: \(averageAssemblyTime)")

let averageTestingTime = toolboxes.map { $0.testingTime }.average()

print("Average toolbox testing time: \(averageTestingTime)")

let onlyPreciseInstrumentsProbability = Double(toolboxes.filter { $0.tools.allSatisfy { $0.type == .precise } }.count) / Double(toolboxes.count)

print("Probability of only precise instruments in toolbox: \(onlyPreciseInstrumentsProbability)")

let moreThenDayTimeProbability = Double(toolboxes.filter { $0.allTime >= 1440 }.count) / Double(toolboxes.count)

print("Probability of 1440 minutes and more of assembly and testing toolbox: \(moreThenDayTimeProbability)")

}

calculateToolboxes()

///For 100 000 toolboxes

///Average toolbox assembly time: 343.83379019630075

///Average toolbox testing time: 69.06121370956292

///Probability of only precise instruments in toolbox: 0.02893

///Probability of 1440 minutes and more of assembly and testing toolbox: 0.0

### 3.3.2 Вторая часть

class Circuit {

var isDefective: Bool

var checkTime: Double = Double.random(in: 2.0...5.0)

var replaceTime: Double

init() {

if (0.0...0.1).contains(Double.random(in: 0...1)) {

isDefective = true

replaceTime = 5.0 \* 0.5 \* sqrt(2.0) \* ((1...6).map{ \_ in Double.random(in: 0.0...1.0)}).reduce(0.0, +) - 3.0

} else {

isDefective = false

replaceTime = 0.0

}

}

}

class Device {

var circuits: [Circuit] = (1...5).map { \_ in Circuit() }

var configurationTime: Double

var allTime: Double

var defectivePercent: Double

init() {

if let \_ = circuits.first(where: { $0.isDefective }) {

configurationTime = -10 \* log(Double.random(in: 0.0...1.0))

} else {

configurationTime = -6 \* log(Double.random(in: 0.0...1.0))

}

allTime = (circuits.map { $0.checkTime + $0.replaceTime }).reduce(0.0, +)

defectivePercent = Double(circuits.filter({ $0.isDefective }).count) / Double(circuits.count)

}

}

func calculateDevices() {

print("\nPart 3.2:")

let devices = (1...100000).map { \_ in Device() }

let averageTime = devices.map { $0.allTime }.average()

print("Average devices processing time: \(averageTime)")

let oneAndMoreDefectiveCircuitsProbability = Double(devices.filter { $0.circuits.contains { $0.isDefective == true } }.count) / Double(devices.count)

print("At least one defective circuit probability: \(oneAndMoreDefectiveCircuitsProbability)")

let averageDefectiveCircuitsPercent = devices.map { $0.defectivePercent }.average()

print("Average defective circuit percent \(averageDefectiveCircuitsPercent)")

}

calculateDevices()

///Part 3.2:

///100 000 devices

///Average devices processing time: 21.28127286809263

///At least one defective circuit probability: 0.40842

///Average defective circuit percent 0.09961799999999575

## 4. Вывод

Изучены метод Монте–Карло и решение задач моделирования на основе алгоритмов имитации случайных событий и величин.