Міністерство освіти і науки України
Національний технічний університет України «КПІ» імені Ігоря Сікорського
Факультет обчислювальної техніки
Кафедра обчислювальної техніки

ЗВІТ
з лабораторної роботи 1
з навчальної дисципліни «Основи комп’ютерного моделювання»

Тема: Перевірка генератора випадкових чисел
на відповідність закону розподілу

Виконав
Студент 4 курсу ІП-94
Рекечинський Дмитро

Перевірив
Іваніщев Б. В.

Київ 2022

Завдання до роботи: Згенерувати 10000 випадкових чисел способом, указаним у варіанті. Побудувати гістограму частот, знайти середнє і дисперсію цих випадкових чисел. По виду гістограми частот визначити вид закону розподілу. Відповідність заданому закону розподілу перевірити за допомогою критерію згоди хі-квадрат.

Варіант завдання: 9423 mod 3 + 1 = 1

1) Згенерувати випадкове число по формулі $x_i = -\frac{1}{\lambda}\ln \xi_i$, де ξ_i - випадкове число, рівномірно розподілене в інтервалі (0;1). Числа ξ_i можна створювати за допомогою убудованого в мову програмування генератора випадкових чисел. Перевірити на відповідність експоненційному закону розподілу $F(x) = 1 - e^{-\lambda x}$. Перевірку зробити при різних значеннях λ.

Лістинг програми

Примітка: Програма написана за допомогою компільованої мови програмування Nim.

Команда для компіляції та запуску: nim c -r solution.nim

Вміст програми у файлі solution.nim:

```
# sequence.sorted()
import std/algorithm


# ln(), sequence.sum(), sqrt(), exp()
import std/math


# initRand(), rand()
import std/random


# sequence.mapIt()
import std/sequtils


##############################
# Types
##############################


type
  StatisticValues = tuple[average, variance: float]
  HistogramUnit = tuple[lowerBound, upperBound: float, occurencies:
int]
  Histogram = seq[HistogramUnit]


##############################
# Constants, global variables
##############################


const AMOUNT_OF_GENERATED_NUMBERS = 10000
```

```
    const NUMBER_OF_FRACTIONAL_DIGITS = 2
    const MINIMAL_NUMBER_OF_OCCURENCIES = 5

    # For the current task, there is only one parameter of distribution
law:
    # lambda
    const NUMBER_OF_DISTRIBUTION_LAW_PARAMETERS = 1

    # Recommended amount of intervals
    const AMOUNT_OF_INTERVALS = 20

    # Level of significance equals 0.05
    const PEARSONS_CHI_SQUARE_TEST = [
      3.8,
      6.0,
      7.8,
      9.5,
      11.1,
      12.6,
      14.1,
      15.5,
      16.9,
      18.3,
      19.7,
      21.0,
      22.4,
      23.7,
      25.0,
      26.3,
      27.6,
      28.9,
      30.1,
      31.4
    ]
```

```
      var randomizer = initRand()

      ##############################
      # Functions
      ##############################

      proc randomNumber(): float = randomizer.rand(1.0)

      proc givenFunction(lambd: float): float = (-1.0 / lambd) *
ln(randomNumber())

      proc calculateError(expected, actual: float): float =
        abs((expected - actual) / actual)

      proc generateSequence(lambd: float): seq[float] =
        for _ in 1..AMOUNT_OF_GENERATED_NUMBERS:
          let number = givenFunction(lambd)
          result.add(number)



      proc calculateStatisticValues(sequence: seq[float]): StatisticValues =
        let sumOfNumbers = sequence.sum()
        let amount = float(AMOUNT_OF_GENERATED_NUMBERS)

        let average = sumOfNumbers / amount

        let variancePart = sequence.mapIt((it - average)^2).sum()
        let squaredVariance = variancePart / (amount - 1.0)
        let variance = sqrt(squaredVariance)

        return (average, variance)
```

```
      proc calculateHistogram(sequence: seq[float], intervalAmount: int):
Histogram =
        let sortedSeq = sequence.sorted()

        let minimum = sortedSeq[sortedSeq.low]
        let maximum = sortedSeq[sortedSeq.high]
        let stepValue = (maximum - minimum) / float(intervalAmount)

        var intervalStep = minimum
        var index = sortedSeq.low

        for _ in 1..intervalAmount:
          let lowerBound = intervalStep
          intervalStep += stepValue
          let upperBound = intervalStep

          var occurencies = 0

          while true:
            var currentValue = sortedSeq[index]

            if currentValue <= intervalStep:
              inc occurencies
              inc index
              if index > sortedSeq.high: break
            else: break

          result.add((lowerBound: lowerBound, upperBound: upperBound,
occurencies: occurencies))

      proc joinIntervalsHelper (target: var HistogramUnit, source:
HistogramUnit) =
        let intervalBounds = @[
          target.lowerBound,
```

```
          target.upperBound,
          source.lowerBound,
          source.upperBound
        ]

        let lowerBound = intervalBounds.min()
        let upperBound = intervalBounds.max()

        target.lowerBound = lowerBound
        target.upperBound = upperBound
        target.occurencies += source.occurencies

      proc joinIntervals(histogram: var Histogram) =
        # Join intervals while any of them has a number of occurencies
        # lower than it's allowed
        while histogram.anyIt(it.occurencies <
MINIMAL_NUMBER_OF_OCCURENCIES):
          var index = histogram.high

          while index > histogram.low:
            let currentInterval = histogram[index]
            let occurencies = currentInterval.occurencies

            if occurencies < MINIMAL_NUMBER_OF_OCCURENCIES:
              var intervalIndex: int

              # Join the interval with one which has lower number of
occurencies
              if index == histogram.high:
                intervalIndex = index - 1
              else:
                let previousIndex = index - 1
                let nextIndex = index + 1
```

```
                intervalIndex = if histogram[previousIndex] >
histogram[nextIndex]:
                  nextIndex
                  else: previousIndex

              joinIntervalsHelper(histogram[intervalIndex], currentInterval)
              histogram.delete(index)
              break
            dec index



      proc hypoteticalDistributionLaw(x, lambd: float): float = 1 - exp(-
lambd * x)


      proc chiSquaredTest(histogram: Histogram, lambd: float): float =
        for interval in histogram:
          let lowerBound = interval.lowerBound
          let upperBound = interval.upperBound


          let theoreticalHitRate =
            hypoteticalDistributionLaw(upperBound, lambd) -
hypoteticalDistributionLaw(lowerBound, lambd)
          let theoreticalOccurencies = (theoreticalHitRate *
AMOUNT_OF_GENERATED_NUMBERS).round().int()


          result += (interval.occurencies - theoreticalOccurencies)^2 /
theoreticalOccurencies



      ##############################
      # Usage
      ##############################
      const lambdas = [0.05, 0.1, 0.2, 0.25]


      for lambd in lambdas:
```

```
        echo "\n============================"
        echo "Lambda: ", lambd

        let expectedAverage = 1.0 / lambd
        let expectedVariance = 1.0 / lambd

        let sequence = generateSequence(lambd)
        let statisticValues = calculateStatisticValues(sequence)

        echo "Expected average: ", expectedAverage
        echo "Actual average: ", statisticValues.average
        echo "Relative calculation error: ",
          calculateError(expectedAverage, statisticValues.average)

        echo "Expected variance: ", expectedVariance
        echo "Actual variance: ", statisticValues.variance
        echo "Relative calculation error: ",
          calculateError(expectedVariance, statisticValues.variance)

        var histogram = calculateHistogram(sequence, AMOUNT_OF_INTERVALS)
        histogram.joinIntervals()

        echo "Histogram sequence"
        for interval in histogram: echo interval.occurencies

        let chiSquaredTestValue = histogram.chiSquaredTest(lambd)
        echo "Chi-squared test: ", chiSquaredTestValue
        let freedomDegree = histogram.len - 1 -
NUMBER_OF_DISTRIBUTION_LAW_PARAMETERS
        let theoreticalChiSquare = PEARSONS_CHI_SQUARE_TEST[freedomDegree]

        echo "Theoretical value of chi-square test: ", theoreticalChiSquare
        echo "Passed: ", chiSquaredTestValue < theoreticalChiSquare
```

Перевірка роботи

Програма генерує числа при таких значеннях параметру “лямбда”: 0.05, 0.1, 0.2, 0.25.

Усі виводи програми, наведені у звіті, перекладені із англійської в українську. Програма при роботі виводить текст англійською мовою.

1. Значення середнього та дисперсії

Програма виводить на екран результати поточного значення середнього та дисперсії, а також обчислює відносну похибку:

==========

Лямбда: 0.05

Очікуване середнє: 20.0

Поточне середнє: 19.78919753727089

Відносна похибка: 0.01065240075208128

Очікувана дисперсія: 20.0

Поточна дисперсія: 19.80907987647805

Відносна похибка: 0.009638010685627897

==========

Лямбда: 0.1

Очікуване середнє: 10.0

Поточне середнє: 9.921100830208989

Відносна похибка: 0.007952662828581421

Очікувана дисперсія: 10.0

Поточна дисперсія: 10.01334316156644

Відносна похибка: 0.001332538129488402

==========

Лямбда: 0.2

Очікуване середнє: 5.0

Поточне середнє: 4.99084602850792

Відносна похибка: 0.001834152253904892

Очікувана дисперсія: 5.0

Поточна дисперсія: 4.978304310271349

Відносна похибка: 0.004358048117687039

==========

Лямбда: 0.25

Очікуване середнє: 4.0

Поточне середнє: 3.99634040823425

Відносна похибка: 0.0009157357461864848

Очікувана дисперсія: 4.0

Поточна дисперсія: 3.966554793468763

Відносна похибка: 0.008431802476624676

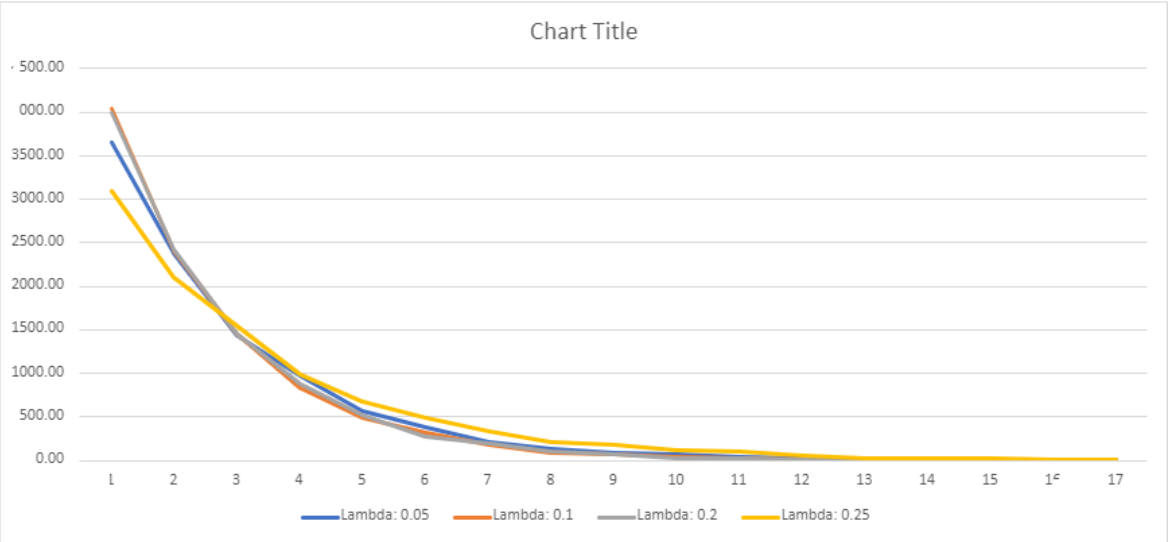
В усіх випадках похибка не перевищує 0.96%, що є задовільним у межах лабораторної роботи.

2. Зображення гістограми частот

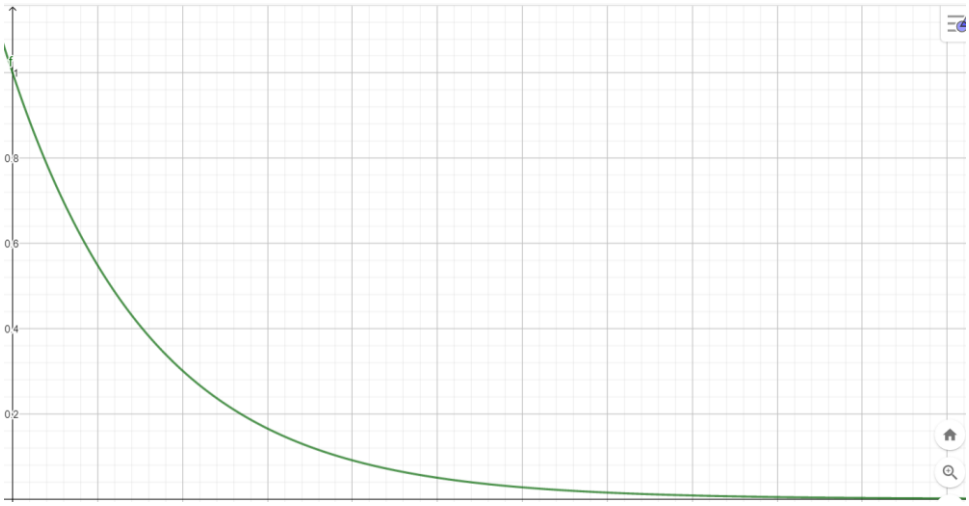
Графіки гістограм частот були побудовані за допомогою Excel:

Lambda: 0.05	Lambda: 0.1	Lambda: 0.2	Lambda: 0.25
3650.00	4045.00	3988.00	3091.00
2386.00	2418.00	2428.00	2104.00
1437.00	1447.00	1446.00	1544.00
963.00	831.00	874.00	990.00
573.00	497.00	527.00	674.00
375.00	323.00	277.00	493.00
217.00	172.00	196.00	332.00
138.00	94.00	106.00	218.00
94.00	72.00	64.00	177.00
67.00	38.00	32.00	121.00
33.00	20.00	30.00	96.00
23.00	14.00	9.00	54.00
16.00	16.00	9.00	26.00
10.00	7.00	7.00	23.00
7.00	5.00	6.00	24.00
5.00			7.00
6.00			16.00
			10.00

Зображення графіків:



Графік експоненційної функції e^{-3x} (функція взята приблизно для наглядності):



Таким чином, розподіл чисел ГВЧ набуває форми експоненійного закону розподілу.

3. Критерій згоди хі-квадрат

Згідно із теоретичними відомостями, наведеними у розділі 2.1, рівень значимості становить 0.05, а кількість степенів свободи рівна кількості інтервалів мінус 1 мінус кількість параметрів закону розподілу (згідно із завданням, їх всього один: “лямбда”).

Програма обчислила критерій згоди хі-квадрат та вивела результати тесту на екран:

==========

Лямбда: 0.05

Значення тесту хі-квадрат: 9.625836204946152

Теоретично допустиме значення тесту хі-квадрат: 26.3

Тест пройдено: true

==========

Лямбда: 0.1

Значення тесту хі-квадрат: 13.53670918328753

Теоретично допустиме значення тесту хі-квадрат: 23.7

Тест пройдено: true

==========

Лямбда: 0.2

Значення тесту хі-квадрат: 11.16480268934084

Теоретично допустиме значення тесту хі-квадрат: 23.7

Тест пройдено: true

==========

Лямбда: 0.25

Значення тесту хі-квадрат: 23.2971423337009

Теоретично допустиме значення тесту хі-квадрат: 27.6

Тест пройдено: true

В усіх випадках генератор випадкових чисел відповідає критерію згоди хі-квадрат відносно наведеного експоненційного закону розподілу.

Висновки про відповідність закону розподілу

Тест на середнє та дисперсію було пройдено із похибкою, що не перевищує 0.96%.

Графік гістограми розподілу чисел ГВЧ має форму, що нагадує експоненційний закон розподілу.

Критерій згоди хі-квадрат показав, що поточний закон розподілу відповідає наведеному експоненційному закону розподілу.

Вищенаведені фактори свідчать про те, що закон розподілу ГВЧ відповідає експоненційному закону розподілу.

Висновки

Під час виконання даної роботи було проведено аналіз закону розподілу ГВЧ. Усі критерії підтверджують припущення відповідності закону розподілу ГВЧ експоненційному. Для роботи було використано компільовану мову програмування Nim та програму для роботи із табличними даними Excel.