Министерство образования Республики Беларусь

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет прикладной математики и информатики

**Кафедра ММАД**

Макаревич Анатолий Сергеевич

Отчет по лабораторным работам по курсу

“Имитационное и статистическое моделирование”

студента 4 курса 10 группы

|  |  |
| --- | --- |
| Работа сдана 20.11.2017г. | **Преподаватель** |
| зачтена \_\_\_\_\_\_\_\_\_\_ 2017 г.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись преподавателя) | *Лобач Виктор Иванович*  доцент кафедры ММАД,  канд. физ.-мат. наук |
|  |  |

*Минск 2017*

**Лабораторная работа 1.**

**Условие:**

Смоделировать БСВ (выборку из 100 элементов) мультипликативным конгруэнтным методом и методом Маклорена-Марсальи и реализовать проверку методами Колмогорова и Пирсона.

**Теория:**

**Мультипликативный конгруэнтный метод:**

Псевдослучайная последовательность  строится по следующим рекуррентным формулам:

где  - параметры датчика:  - множитель (*<M*), *M* – модуль,  - стартовое значение (нечетное число).

В данной работе брались значения: *M*=2147483648, ==65539.

**Метод Маклорена-Марсальи:**

Пусть  - псевдослучайные последовательности, порожденные независимо работающими датчиками;  - результирующая псевдослучайная последовательность реализация БСВ;

*V={V(0), V(1), …,V(K-1)}* – вспомогательная таблица *K* чисел.

Процесс вычисления  включает следующие этапы:

- первоначальное заполнение таблицы

*V*: 

- случайный выбор из таблицы:



-обновление табличных значений:

.

В данной работе в качестве  бралась последовательность (из 100 элементов), полученная мультипликативным конгруэнтным методом, описанным выше. В качестве , бралась последовательности (из 10000) элементов, полученная аналогичным способом с тем же M и . *K*=100.

** - критерий согласия Пирсона:**

Область возможных значений случайной величины разбивается на интервалы .

Рассматривается следующая статистика,

,

*n* – объем выборки,

 - количество элементов выборки, попавших в *k*-ый интервал,

 - вероятность попадания случайной величины в *k*-ый интервал.

Проверяется условие , где , *G* функция распределения распределения**,**  - уровень значимости (обычно =0.05).

В данной работе отрезок [0;1] разбивался на 10 интервалов.

**Критерий согласия Колмогорова:**

Рассматривается статистика:



где

,



Проверяется условие , где , *K* - функция распределенияраспределения Колмогорова**,**  - уровень значимости.

**Код программы: см по ссылке**

[**https://github.com/NowanIlfideme/statmodelling/blob/master/Lab1.py**](https://github.com/NowanIlfideme/statmodelling/blob/master/Lab1.py)

**Результаты:**

По всем тестам проходит.

**Лабораторная работа 2.**

**Условие:**

Смоделировать распределение Пуассона и реализовать проверку методом Пирсона.

**Теория:**

**Распределение Пуассона (с параметром ):**

Случайная величина  принимает только целые неотрицательные значения, причем 

В данной работе, сначала моделировалась последовательность БСВ, а потом по каждой БСВ строился соответствующий элемент выборки распределения Пуассона: отрезок [0;1] разбивался на интервалы длин  проверялось, в какой интервал попадает элемент последовательности БСВ.

**Код программы: см по ссылке**

[**https://github.com/NowanIlfideme/statmodelling/blob/master/Lab2.py**](https://github.com/NowanIlfideme/statmodelling/blob/master/Lab2.py)

**Результат:**

Generating 1000 of BernoulliGen

Test result: 0.527089256866

Generating 1000 of BinomialGen

Test result: 0.834326795774

Generating 1000 of GeometricGen

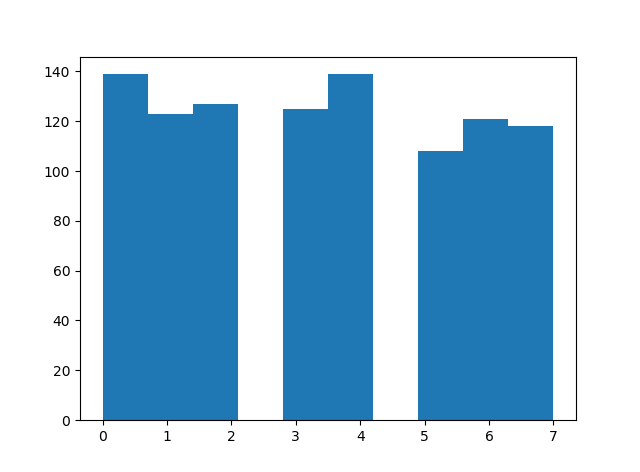
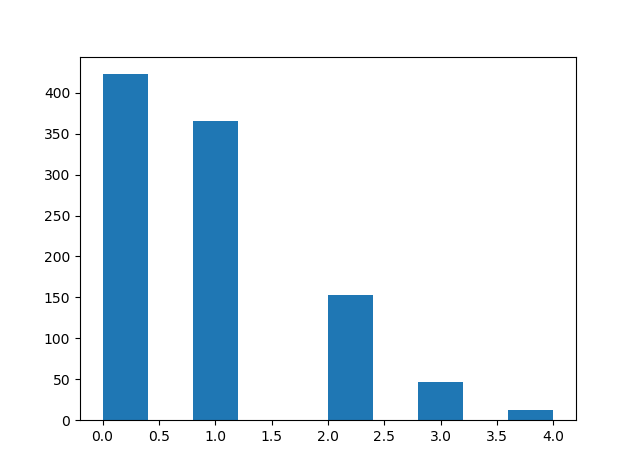
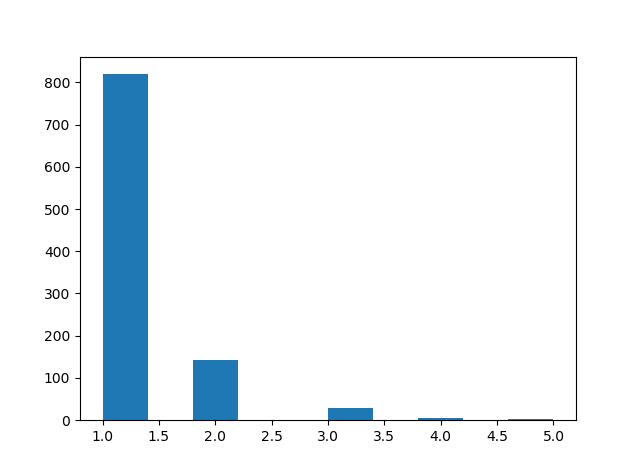
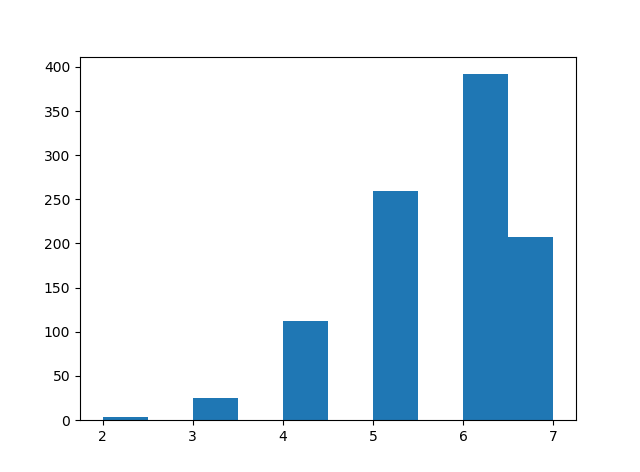
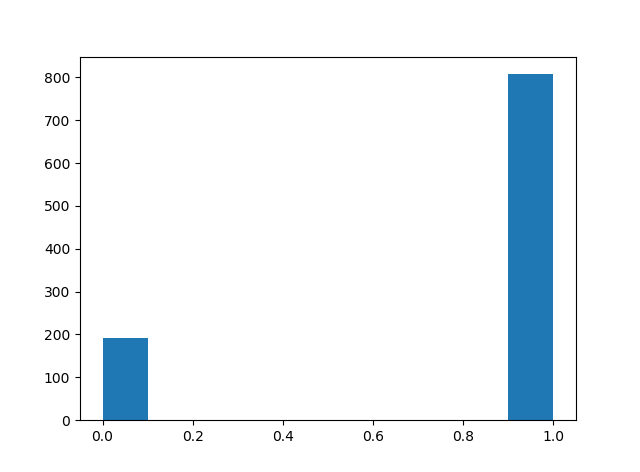
Test result: 0.554901100245

Generating 1000 of PoissonGen

Test result: 0.157394209549

Generating 1000 of DiscreteUniformGen

Test result: 0.536017885119

****

**Лабораторная работа 3.**

**Условие:**

Смоделировать распределение Вейбулла-Гнеденко и реализовать проверку методом Колмогорова.

**Теория:**

**Распределение Вейбулла-Гнеденко (с параметрами** *k,* **):**

Распределение имеет плотность



Распределение может быть смоделировано методом обратной функции:

,

*U* – БСВ.

**Код программы: см. по ссылке**

[**https://github.com/NowanIlfideme/statmodelling/blob/master/Lab3.py**](https://github.com/NowanIlfideme/statmodelling/blob/master/Lab3.py)

**Результат:**

Generating 1000 of ContinuousUniform

Test result: 1.0

Generating 1000 of NormalUnivariate

Test result: 0.701077186769

Generating 1000 of LogNormal

Test result: 0.638960751179

Generating 1000 of ExponentialDist

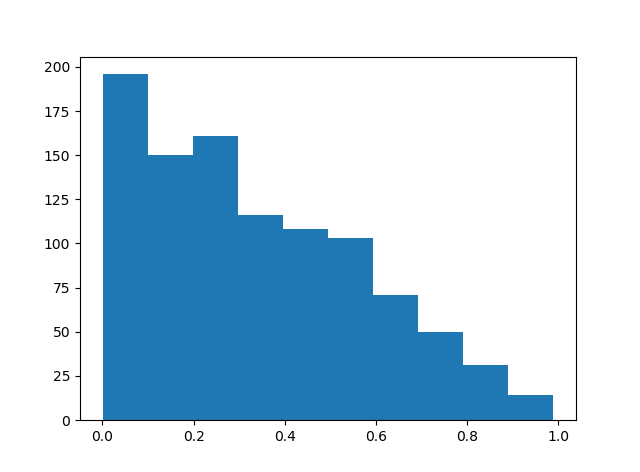
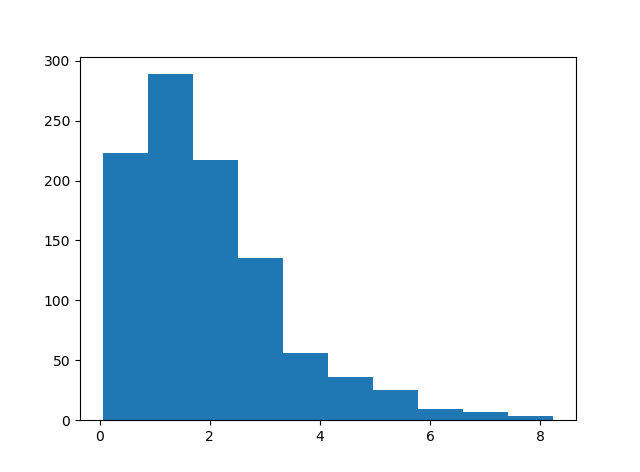
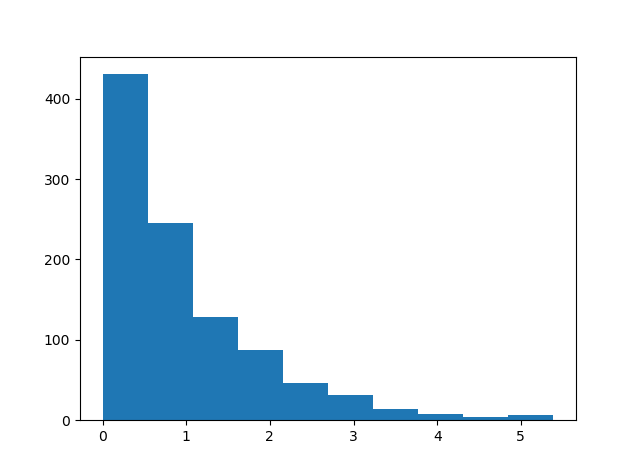
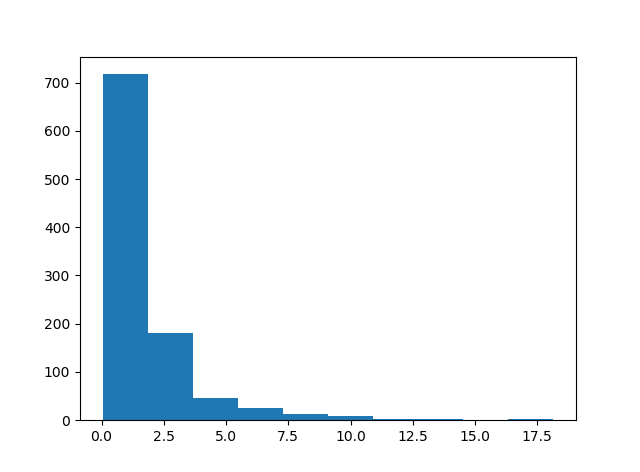
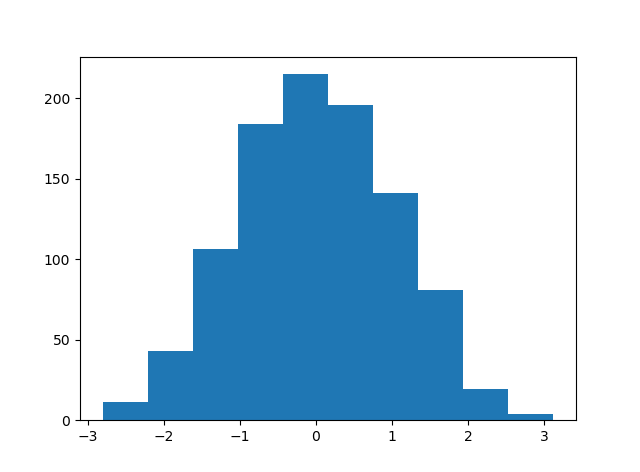
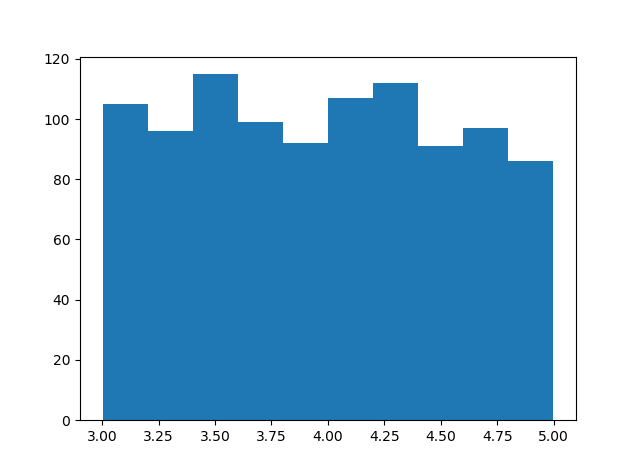
Test result: 1.0

Generating 1000 of GammaDist

Test result: 1.0

Generating 1000 of BetaDist

Test result: 0.956747667308



**Лабораторная работа 4.1.**

**Условие:**

Вычислить интеграл методом Монте-Карло:

 и 

**Теория:**

**Метод Монте-Карло приближенного вычисления интеграла:**

Необходимо вычислить .

Пусть  - произвольная случайная величина с плотностью распределения  имеющая конечный момент второго порядка.

Пусть  Тогда 

В качестве приближенного значения *a* можно взять



В данной работе в качестве  бралась случайная величина, равномерно распределенная на [0;π], [0;1], [0;2].

**Код программы: см. по ссылке** [**https://github.com/NowanIlfideme/statmodelling/blob/master/Lab4.py**](https://github.com/NowanIlfideme/statmodelling/blob/master/Lab4.py)Результат:





**Лабораторная работа 4.2.**

**Условие:**

Решить систему линейных алгебраических уравнений методом Монте-Карло:



**Теория:**

**Метод Монте-Карло приближенного решения системы линейных алгебраических уравнений:**

Необходимо решить систему, представленную в виде , где , собственные значения *A* по модулю меньше 1.

Наша цель – вычислить скалярное произведение вектора решения  с некоторым вектором .

Рассмотрим цепь Маркова с параметрами  такими что





 если 

 если 

Положим



Выберем некоторое натуральное *N* и рассмотрим случайную величину



Где 🡪🡪…🡪 - траекторая цепи Маркова.

*Qm* опряделяется как:



Тогда скалярное произведение вектором *h* и *x* приблизительно равно .

Можем найти *x*, скалярно умножая его на векторы *h* у которых в одной позиции стоит 1, а в остьльных – 0.

В данной работе выбиралось 

**Код программы: см. по ссылке** [**https://github.com/NowanIlfideme/statmodelling/blob/master/Lab4.py**](https://github.com/NowanIlfideme/statmodelling/blob/master/Lab4.py)

**Результат:**

****

**Задания 5 и 6**

**1. Постановка задачи**

**Разработать в системе GPSS имитационную модель вычислительной системы коллективного пользования** ([2], задание 78).

ЗАДАНИЕ 18. Моделирование вычислительной системы с удаленными терминалами

Вычислительная система представляет собой двухпроцессорный комплекс, который

обслуживает местных пользователей и три однотипных удаленных терминала.

На каждом терминале задача формируется в среднем через t сек, а время выполнения

задачи в процессорном блоке имеет математическое ожидание µ сек (экспоненциальное

распределение). После выполнения задача возвращается на соответствующий терминал,

инициируя тем самым формирование новой задачи. Время передачи данных по каналу связи

распределяется равномерно в пределах от a до b сек.

Разработать GPSSV- модель для анализа процесса функционирования вычислительной системы в течение одного часа.

Первоначальный перечень экспериментов: t=25, µ=20, a=10, b=30.

**2. Обоснование моделирующего алгоритма**

В данной системе – 2 процессора, 1 локальный и 3 удаленных терминала. Каждый терминал занимает процессор, и, если удаленный, то занимает еще время передачи данных. Затем симулируется работа, и возврат результата мгновенный. Далее процессор освобождается для повторного использования.

**3. Имитационная модель на GPSS/РС (текст программы)**

……………………………………………………………………………………

;\*\*\*\*\*\*\*

; 2 processors handling 1 local and 3 remote terminals

; Tasks come to terminal at time delays ~ Exp(t), t=25

; Tasks take time ~ Exp(mu), mu=20 to be finished on a processor.

; Data transfer from remote terminals takes time ~ R[a,b], a=10, b=30

;\*\*\*\*\*\*\*

; \*\* tottime TABLE M1,30,30,20 ; total time in system

proc STORAGE 2 ; two processors

GENERATE 25;(EXPONENTIAL(1,0,25)) ; t=25

TRANSFER ,work

GENERATE 25;(EXPONENTIAL(1,0,25)) ; t=25

TRANSFER ,datt

GENERATE 25;(EXPONENTIAL(1,0,25)) ; t=25

TRANSFER ,datt

GENERATE 25;(EXPONENTIAL(1,0,25)) ; t=25

TRANSFER ,datt

;\*\*\*\*\*\*\*

; simulate data transfer

datt QUEUE datatrans,1

ADVANCE 20,10 ;a=10 to b=30

DEPART datatrans

TRANSFER ,work

;\*\*\*\*\*\*\*

; simulate working

work QUEUE waittime,1

ENTER proc ; Use a processor

DEPART waittime

QUEUE worktime,1

ADVANCE (EXPONENTIAL(1,0,20)) ; mu=20, set to 5 to clear

DEPART worktime

LEAVE proc

TERMINATE

;\*\*\*\*\*\*\*

GENERATE 3600 ; 60 minutes

TERMINATE 1

**4. Анализ результатов имитационных экспериментов**

Результат имитационной модели:

QUEUE MAX CONT. ENTRY ENTRY(0) AVE.CONT. AVE.TIME AVE.(-0) RETRY

DATATRANS 6 4 432 0 2.376 19.797 19.797 0

WAITTIME 247 247 572 2 125.014 786.801 789.562 0

WORKTIME 2 2 325 0 1.981 21.941 21.941 0

STORAGE CAP. REM. MIN. MAX. ENTRIES AVL. AVE.C. UTIL. RETRY DELAY

PROC 2 0 0 2 325 1 1.981 0.990 0 247

Аналогичная задача была реализована в средствах Python с использованием библиотеки SimPy. Текст программы следующий (так же можно его увидеть по ссылке <https://github.com/NowanIlfideme/statmodelling/blob/master/Lab5.py>):

import simpy

import random

import numpy as np

import matplotlib.pyplot as plt

debug = False

def P(\*args):

if debug: print(\*args)

pass

class Lab18(object):

class Terminal(object):

"""Represents a single terminal for users."""

def \_\_init\_\_(self, obj, env, resource, t=25, mu=20, name=""):

self.obj = obj

self.env = env

self.resource = resource

self.action = env.process(self.run())

self.t = t

self.mu = mu

self.name = name

pass

def P(self, \*args):

P(self.name,\*args)

pass

def run(self):

"""Running process."""

while True:

# Generate problem

st = self.env.now

yield self.env.process(self.gen\_problem())

self.obj.gen\_times.append(self.env.now - st)

# Solution

st = self.env.now

yield self.env.process(self.request\_solve())

self.obj.solve\_times.append(self.env.now - st)

pass

pass

def gen\_problem(self):

"""Simulate generating a problem."""

self.P("@ %04.02f Generating problem " % self.env.now)

duration = random.expovariate(1.0/self.t)

yield self.env.timeout(duration) # Generate next problem

pass

def request\_solve(self):

"""Simulate a request to solve the problem."""

with self.resource.request() as req:

self.P("@ %04.02f Requesting terminal access " % self.env.now)

yield req # Request access

self.P("@ %04.02f Requesting solution" % self.env.now)

duration = random.expovariate(1.0/self.mu)

yield self.env.timeout(duration) # Do calculation

pass

pass

class RemoteTerminal(Terminal):

"""Represents a remote user."""

def \_\_init\_\_(self, obj, env, resource, t = 25, mu = 20, a=10, b=30, name=""):

super().\_\_init\_\_(obj, env, resource, t=t, mu=mu, name=name)

self.a = a

self.b = b

pass

def request\_solve(self):

"""Simulate a request to solve the problem.

Note that extra time is required to transfer data."""

with self.resource.request() as req:

self.P("@ %04.02f Requesting terminal access" % self.env.now)

yield req # Request access

self.P("@ %04.02f Moving data" % self.env.now)

transf\_time = random.randint(self.a, self.b)

yield self.env.timeout(transf\_time) # Transfer data

self.P("@ %04.02f Requesting solution" % self.env.now)

duration = random.expovariate(1.0/self.mu)

yield self.env.timeout(duration) # Do calculation

pass

pass

def \_\_init\_\_(self, t = 25, mu = 20, a=10, b=30, n\_remote=3, n\_processors=2):

# Options

self.a = a

self.b = b

self.t = t

self.mu = mu

self.n\_remote = 3

self.n\_processors = n\_processors

# Save data

self.gen\_times = []

self.solve\_times = []

pass

def run(self, T=3600):

"""Runs a simulation for T seconds."""

env = simpy.Environment()

procs = simpy.Resource(env, capacity=self.n\_processors)

home\_term = Lab18.Terminal(self, env, procs, name="home ")

remote\_terms = [Lab18.RemoteTerminal(self, env, procs, name="sess%d " % i) for i in range(3)]

env.run(until=3600) # Run for T = 60\*60 seconds = 60 minutes

return self

def output(self):

plt.title("Problem generation times.")

plt.hist(self.gen\_times)

plt.show()

plt.title("Problem solution times (incl. waiting).")

plt.hist(self.solve\_times)

plt.show()

return self

pass

debug = True

l18 = Lab18().run(3600)

print("Average gen time: %0.2f" % np.mean(l18.gen\_times))

print("Average solve time: %0.2f" % np.mean(l18.solve\_times))

l18.output()

Вывод программы:

home @ 0.00 Generating problem

sess0 @ 0.00 Generating problem

sess1 @ 0.00 Generating problem

sess2 @ 0.00 Generating problem

sess1 @ 0.75 Requesting terminal access

sess1 @ 0.75 Moving data

...

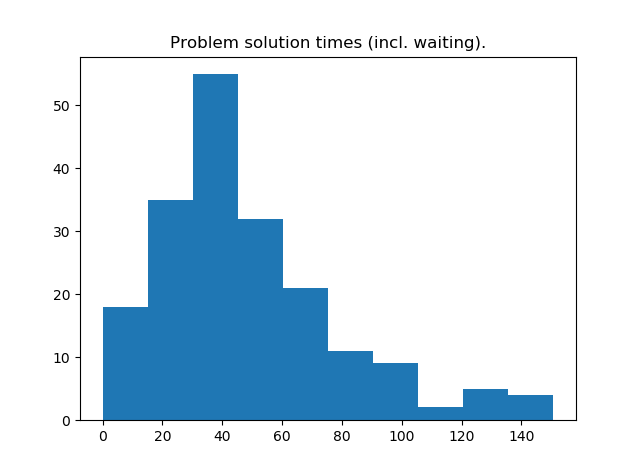
sess1 @ 3586.78 Generating problem

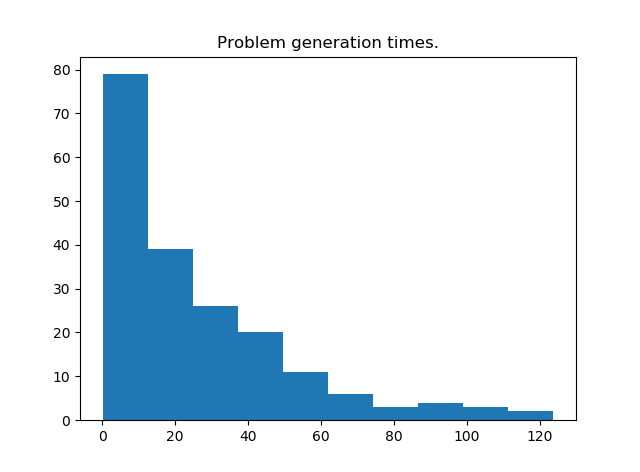
home @ 3586.78 Requesting solution

home @ 3590.77 Generating problem

Average gen time: 25.71

Average solve time: 48.40





**Литература**

1. Харин Ю.С., Малюгин В.И., Кирлица В.П., Лобач В.И., Хацкевич Г.А. Основы имитационного и статистического моделирования. Учебное пособие. Минск: ДизайнПРО, 1997 – 228 с.
2. Лобач В.И., Кирлица В.П., Малюгин В.И., Сталевская С.Н. Имитационное и статистическое моделирование. Практикум для студентов математических и экономических специальностей. Минск, БГУ, 2004 –189 с.