Communication Networks: Theory
Homework Assignment 3
Discrete Event Simulation and Random Number Generators
(25 points)

Prof. Dr. Anna Förster
Dr. Andreas Könsgen
Dr. Asanga Udugama
Idrees Zaman MSc.
Dipl.-Ing. Jens Dede
Vishnupriya Kuppusammy MSc.

Release date: May 29th, 2019, 10:00 AM
Due date: June 19th, 2019, 10:00 AM (hard deadline!)

- All submissions **must** be uploaded through Stud.IP/DoIT.
- All late submissions are **rejected**.

Student Declaration: By handing in my submission, I assure, that this work has been done solely by me without any further help from others except for the official attendance by the teaching staff of the Chair of Communication Networks.

- Note that the format of your submission and its presentation quality is an essential part of the grading.
- Answers must be submitted in a PDF document (Portable Document Format) and programmed code, as Python code files.
- In case of Python code, submit the Python source files indicating the Python version used. All source files must contain comments to explain the code. Quality and the formatting of the Python code is an essential part of grading.
- Handwritten and scanned submissions are rejected.
- Indicate clearly your **First Name**, **Last Name** and your **Matriculation Number** in all submissions (PDF file and Python source files).

Exercise 1. Random Number Generators (16 points)

In class, we have seen how to form linear congruential RNGs and Lagged Fibonacci RNGs. We could also combine them into something we call a Mixed LBG (Mixed Linear Fibonacci Generator). For example, we could take the following LBG:

$v_i = (v_{i-a} - v_{i-b}) mod\ 2^{24}$
$c_i = (c_{i-1} - M) mod\ 2^{32}$
$d_i = (v_i - c_i) mod\ 2^a$

where

a is your age in years
b is the age of your youngest sibling (brother/sister), not including yourself. If you do not have any siblings, $b = 1$.
M is your matriculation number

Q1. (3 points) Do you agree to the following statement: *"The period length of my personalised generator depends only on a."* Explain why or why not. Be careful to consider the difference between period length and the number of possible values a generator creates.

Q2. (3 points) Which are the seeds of your generator and how do you plan to generate them? Make an online literature survey and cite the reference you decided for.

Q3. (5 points) Implement this RNG in the Python programming language. Make sure you define a, b and M as parameters, which can be changed from the code easily. Use the seeds from Q2.

Q4. (5 points) Implement the chi-square test in your program above. The number of generated random numbers should be a parameter (in the code). Test the frequency of your RNG against a uniform distribution at a confidence level of 90% in the code with 1000 numbers. Your program should output the complete set of generated random variables, nicely formatted, together with the final value of chi squared and the result of its comparison to the maximum accepted chi squared at confidence level of 90%. Use the example from the lecture, if needed.

Exercise 2. Discrete Event Simulation (9 points)

The following code of a simple discrete event simulation is given:

```
event A:
        if (stateA == 0)
                r = drawrng();
                schedule(eventB, simTime+r)
        else
                schedule(eventA, simTime+r)
        end
        stateA = ~stateA

event B:
        schedule(eventC, simTime + 10)

event C:
        schedule(eventA, simTime + 1)

event D:
        do nothing

main:
        stateA = 0
        schedule(eventA, 8)
        schedule(eventD, 13)
        while (scheduledQueue is not empty)
                fire nextEvent
```

The used random number generator is of the type:

$x_i = (7x_{i-1} + 11) mod\ 9$

with x_0 is your matriculation number.

Q1. (3 points) Which are the first 20 numbers produced by your random number generator (including x_0)?

Q3. (3 points) Which are the first 20 events, coming out of your discrete event simulation? Write them down together with their simulated execution time.

Q3. (3 points) Rewrite the above pseudo code to avoid the internal state variable stateA.

Exercise 1.

(A): Yes, I agree. "The period length depends only on *a*." The period of a generator is at most *mod* a or *mod* 2^a, and for some choices much less than that. With this, the generator is capable of producing good, distinct pseudorandom numbers; and here, this is extremely sensitive to *a*.

(B). We first make the LCG, and then use the values as the seeds to finally implement the mixed Fibonacci. One benefit of LCGs is that with an appropriate choice of parameters, the period is known and long. The Linear Congruential Generators are capable of producing pseudorandom numbers which can pass formal tests for randomness; in most cases, this is extremely sensitive to the choice of the parameters *m* and *a*.

Reference: *https://en.wikipedia.org/wiki/Linear_congruential_generator*

The LCG algorithm is: $d_{n+1} = (ad_n + c)$ *mod* m. a, d_n, c, and m values were chosen to follow Hull-Dobell Theorem.

(C). Please, see attached Python Code.

(D). Please, see attached Python Code.

Exercise 2.

(A) . Running the Python script. (see attached *.py* file also)

```
a2 = 7
c2 = 11
m2 = 9
## Xo == Xo2
Xo2 = 3061569
LCG_list = [Xo2]
for i in range(1,20):
    Xo2 = (a2*Xo2 + c2)% m2
    LCG_list.append (Xo2)
    print (LCG_list)

    ### To print the first 20 random numbers, X0 incluzive
LCGrn = LCG_list[0:19]
print ("then, the first 20 random numbers, Xo incluzive is: ", LCGrn)
```

Then, the first 20 sequence with X_o inclusive are: [3061569, 5, 1, 0, 2, 7, 6, 8, 4, 3, 5, 1, 0, 2, 7, 6, 8, 4, 3]

(B). The first 20 events from the DES are below:

	Event	Simulation Exectiton Time
1	A	simTime + 8
2	B	simTime + 18
3	C	simTime + 19
4	A	simTime + 27
5	B	simTime + 37
6	C	simTime + 38
7	A	simTime + 46
8	B	simTime + 56
9	C	simTime + 57
10	A	simTime + 65
11	B	simTime + 75
12	C	simTime + 76
13	A	simTime + 84
14	B	simTime + 94
15	C	simTime + 95
16	A	simTime + 103
17	B	simTime + 113
18	C	simTime + 114
19	A	simTime + 122
20	B	simTime + 132

(C).

```
Init:
        Schedule (eventA, simTime+8)
event A:
        r = drawrng();
        int u= (rand() % 50);
        if (u ==  5)
        schedule (eventA, simTime+r)
        else
        schedule (eventB, simTime+r)

        end

event B:
        schedule (eventC, simTime + 10)

event C: schedule (eventA, simTime + 1)

event D:
        do nothing

main:
        schedule (eventA, 8)
        schedule(eventD, 13)
        while (scheduledQueue is not empty)
                fire nextEvent
```

```
print ('  ' * 20)
print(' Aguboshim, Emmanuel | 3061569 | eagubosh@uni-bremen.de ' )
print(' Communication Network - Homework III - Discrete Event Simulation and Random Number
Generators')
print(' Python version 3.7')


# To implement this we first make the LCG, and then use the values as the seeds to finally implement
the mixed Fibonacci:
# LCG algorithm first is: dn+1 = (a*dn + c) mod m
# a, dn, c, and m values were chosen to follow Hull-Dobell Theorem:
# Compute LCG with these parameters
# Obtain (29-1) distinct random numbers.
# Use this as seed to obtain the next (k-a), (k-b), and (k-1) values?
# The result of this is appeneded onto a list called seed (seed here is the list of the 28 lcg
numbers + other newly generated ones),
# ...and yet is still appended to a (empty) list called FibRNG. The FibRNG will contain random values
with index 28 till 1028. i.e 1000 random numbers.
# Iteration is made from 29 to 1029 also.
# Here, the displayed values are just to make the print a lot easier to view. Prints on a 50 interval
scale.

# e = multiplier
# c = incrementor
# m = modulo

print ('__' * 20)
print ('#Exercise 1b & 1c')

import random
import numpy as np
import scipy.stats
import scipy
```

```
from scipy.stats import chisquare
import matplotlib.pyplot as plt

####take here:
a = 27 # your age &
b = 25, # age of younger sibling
viMod = pow(2,24)
ciMod = pow(2,32)
diMod = pow(2,a)
p = pow(2,18)
N = 10

#e, c, and p were choosen to follow Hull Dobell.
def lcg (e = 101427, c = 321, Xo = 123456789):
    LCG_list = []
    for i in range(0,28):  # length of seed is 28 (identified by index 0 to index 27)
        Xo = (e*Xo + c)% p
        LCG_list.append (Xo)
    return LCG_list

seed = lcg()

print(f"{'    '*20} \n Length of seed : {len(seed)}")
print("\n Seed :", seed)

def mFibRNG (a = 27, b = 25, M = 3061569):
    FibRN = []
    seed = lcg()
    for k in range(28,1028):
        vi = ((seed[k - a]) - (seed[k - b]))% viMod
        ci = ((seed[k - 1]) - M) % ciMod
        di = (vi - ci) % diMod
        FibRN.append (di)
        seed.append(di)
    return FibRN

fib = (mFibRNG()) # fib stores the values of the generated fibonacci sequence.

i = 0
inc = 50
print ('__' * 30)
print(f"{'  ' * 30} \n The random vlaues from the Mixed Fibonacci Generator are as below (grouped for
better view):")
print ('  ' * 30)
while i<len(fib)-1:
    print(f" Elements {i} to {inc} : {fib[i:inc]} \n")
    i = inc
    inc +=50


#**************************************
# Here, the Chi-Squared Test starts:
print ('__' * 20)
print ('#Exercise 1d')

r_ange = max(fib) - min(fib) # obtains the range of values so I can apply it to get the classwidth
ClassWidth = round(r_ange/N) # obtain the classwidth

print("\n Range :", r_ange)
print ("\n The Maximum number is :", max(fib))
print ("\n The Minimum number is :", min(fib))
print("\n The ClassWidth is :", ClassWidth)

total = 0
interval = []
interval2 = []
for i in range(0, 11):
    total = total + ClassWidth
    interval.append(total)
#print (interval)  #///To print the intervals expected

print ('__' * 30) #//Just to draw a line:
print ("\n The 10 Class intervals are listed as:")
x1 = [interval[0], interval[1] - 1]
```

```
x2 = [interval[1], interval[2] - 1]
x3 = [interval[2], interval[3] - 1]
x4 = [interval[3], interval[4] - 1]
x5 = [interval[4], interval[5] - 1]
x6 = [interval[5], interval[6] - 1]
x7 = [interval[6], interval[7] - 1]
x8 = [interval[7], interval[8] - 1]
x9 = [interval[8], interval[9] - 1]
x10 = [interval[9], max(fib)]

print (x1), print (x2), print (x3) , print (x4), print (x5), print (x6), print (x7), print (x8),
print (x9), print (x10)  # just to print the classwidths

#*******************************
def make_frequency_distribution(sequence):
    return np.histogram(sequence)

distribution = make_frequency_distribution(fib)
#print(distribution)  ##Uncomment to view.

print ('__' * 20)
Obs = distribution[0]
print ("\n The observed distribution is :", Obs)

expected = len(fib)/N
Exp = [expected, expected, expected, expected, expected, expected, expected, expected, expected,
expected]
print ("\n The expected distribution is :", Exp)


# Chi_Square formular implementation:
D = scipy.stats.chisquare(Obs, Exp, 9)
print ("\n The D statistics is:", D[0])


#***plt.hist(distribution, bins="auto")
#plt.show()
print ('__' * 20)
chisq = 14.68

print ("\n From the Chi-Sqaured distribution table, we see that at DegreeOfFreedom = 9, and P_Value =
0.10, the ChiSquared Value is:", chisq)

if D[0] < chisq:
    print (' Yes, accept at 90% confidence level')
else:
    print (' No, not accept at 90% confidence level')


print ('__' * 20)
print ('#Exercise 2a')
print ('  ' * 20)
#### Exercise 2a:

a2 = 7
c2 = 11
m2 = 9
## Xo == Xo2
Xo2 = 3061569
LCG_list = [Xo2]
for i in range(1,20):
    Xo2 = (a2*Xo2 + c2)% m2
    LCG_list.append (Xo2)
    print (LCG_list)

    ### To print the first 20 random numbers, X0 incluzive
LCGrn = LCG_list[0:19]
print ("then, the first 20 random numbers, Xo incluzive is: ", LCGrn)
```