МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»
КАФЕДРА №51

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доц., к.т.н.		Окатов А.В
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

ГЕНЕРАТОРЫ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ XORSHIFT И ЛЕМЕРА

по курсу: Программно-аппаратные средства защиты информации в инфокоммуникационных системах связи

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	5711М		Пятаков В.С.
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2017

Цель работы

Изучение генераторов псевдослучайных чисел.

Порядок выполнения работы

1. Реализовать алгоритмы генераторов псевдослучайных чисел xorshift и Лемера.
2. Провести тест на решетчатость для каждого алгоритма .
3. Построить функцию автокорреляции для каждого алгоритма.
4. Выделить у каждого из алгоритмов 1,2 и 3 младших разряда и для каждого провести тесты на решетчастость и построить автокорреляционную функцию.
5. Изменить параметры у алгоритма Лемера и посмотреть как будут меняться результаты тестов.
6. Сделать выводы по полученным результатам в ходе исследования двух алгоритмов..

Описание алгоритмов

Линейный конгруэнтный датчик Лемера

Одними из наиболее часто используемых методов генерации псевдослучайных чисел являются различные модификации так называемого **линейного конгруэнтного метода**, схема которого предложена Д.Г. Лемером (Derrick Henry Lehmer) в 1949 г.:

$X_{n+1} = (aX_n + c) \bmod m$, где m – модуль, a – множитель, c – приращение, mod – операция взятия остатка от деления.

Причем $m > 0$, $0 < a \leq m$, $0 < c \leq m$, также задается начальное значение X_0: $0 < X_0 \leq m$.

Выбор параметров m, a, c, X_0 не может быть произвольным. Пример: при m=7, X_0=1, a=2, c=4 получится следующая последовательность: 1,6,2,1,6,2,1,… Таким образом, напрашиваются два вывода:

- Числа m, a, c, X_0 не могут быть случайными.
- Линейный конгруэнтный метод даёт нам повторяющиеся последовательности – конгруэнтная последовательность ВСЕГДА образует «петли». Этот цикл (период), повторяющийся бесконечное число раз – свойство всех последовательностей вида $X_n+1 = f(_n)$

Таким образом, необходимо выбирать параметры m, a, c, X_0 так, чтоб период был максимальным. Модуль m должен быть достаточно большим, т.к. период не больше m. Удобно связать m с длиной слова компьютера и использовать $2^e - 1$ либо $2^e + 1$ для e-разрядной машины, а еще лучше – m наибольшее простое, меньшее 2^e. При этом длина периода равна m в следующем случае: c и m – взаимно простые числа, b = a – 1 кратно p для любого p, являющегося множителем m, b кратно 4, если m кратно 4.

В реализованной программе взяты следующие числа :

- m = 2^31-1;
- a = 7^5;
- c = 0;

XORShift

Джорджем Марсаглией был разработан алгоритм ,который генерирует следующее число последовательности с помощью неоднократного сдвига текущего числа. В данном алгоритме не используется умножение, а используется только битовые сдвиги и сложение по модулю 2(XOR).

Как самый оптимальный ГПСЧ для большинства приложений Джордж Марсаглия рекомендует «Xorshift». Если нужны очень большие периоды, тогда нужно использовать ГПСЧ базирующиеся на MWC или CMWC.

Использование данных алгоритмов позволяет достигнуть периода 2^113102. Если же достаточно периодов: 2^160, 2^128, 2^96, тогда оптимальным вариантом будет «Xorshift». Данный алгоритм доказал свою стойкость пройдя тесты DIEHARD и может использоваться практически во всех областях.

Реализация на языке Python 2.7 выглядит следующим образом:

На вход подаются одно десятичное x, задающие период генерации чисел. В основе алгоритма лежит логический сдвиг и битовая операция XOR(исключающее ИЛИ).
Процедура генерации:

1. входное число складывается по модулю 2 с самим собой, сдвинутый на 11 позиций влево.
2. полученное складывается по модулю 2 с собственной копией, сдвинутый на 19 позиций вправо.
3. полученное число складывается по модулю для с собственной копией, сдвинутый на 8 позиций вправо.
4. Полученное значение является результатом генератора.

Листинг на python:

```
MAX32 =0xffffffff
X = 123456789
Y = 362436069
Z = 521288629
W = 88675123
```

```
def xor_shift():
    global X, Y, Z, W
    t = X ^ (X << 11)
    X = Y
    Y = Z
    Z = W
    W = W^ (W >>  19) ^ t ^ (t >> 8)
    return W
'''
W1 = xor_shift() % MAX32
W2 = xor_shift() % MAX32
W3 = xor_shift()% MAX32
'''
my_file = open("chet.txt", "w")
my_file1 = open("nechet.txt", "w")
my_file2=open("some.txt", "w")
n=0
while n<1000:
    print(n)
    z=xor_shift() % MAX32
    n+=1
    print(z)
    my_file2.write(str(z))
    my_file2.write("\n")
   # n % 2 ==0 if (my_file.write( str(z) )) else my_file.write("\n")
    if n % 2 ==1:
        my_file.write(str(z))
        my_file.write("\n")
    else:
        my_file1.write(str(z))
        my_file1.write("\n")
my_file.close()
my_file1.close()
my_file2.close()
```

Полученные результаты

Алгоритм Лемера для p=2^31-1

Тест на решетчатость

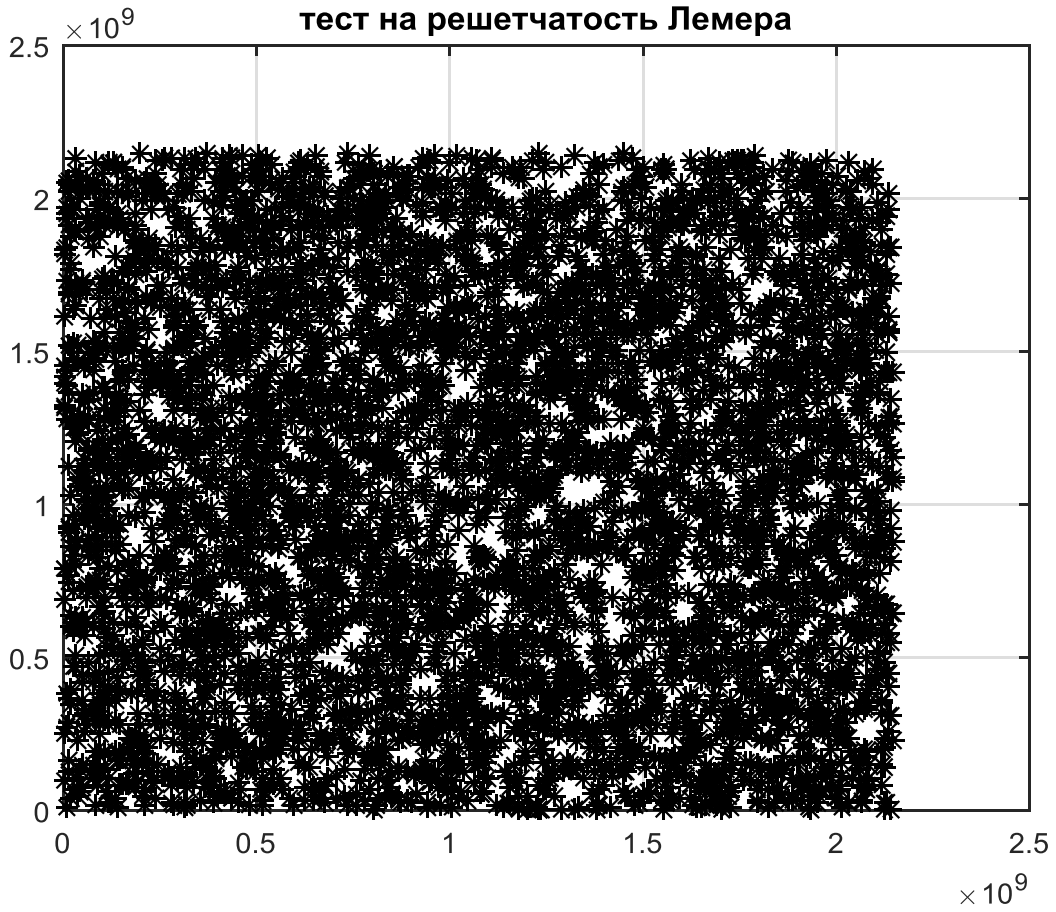


рисунок 1 - Тест на решетчастость для алгоритма Лемера.

Автокорреляционная функция

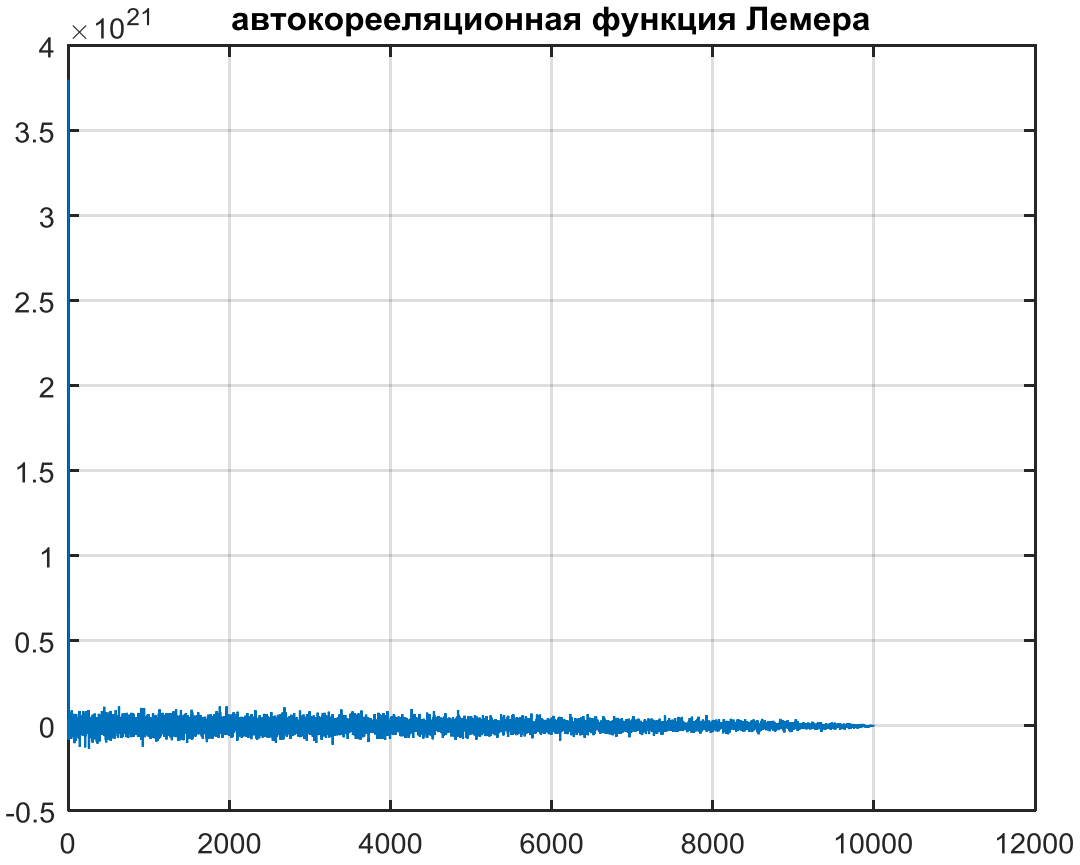
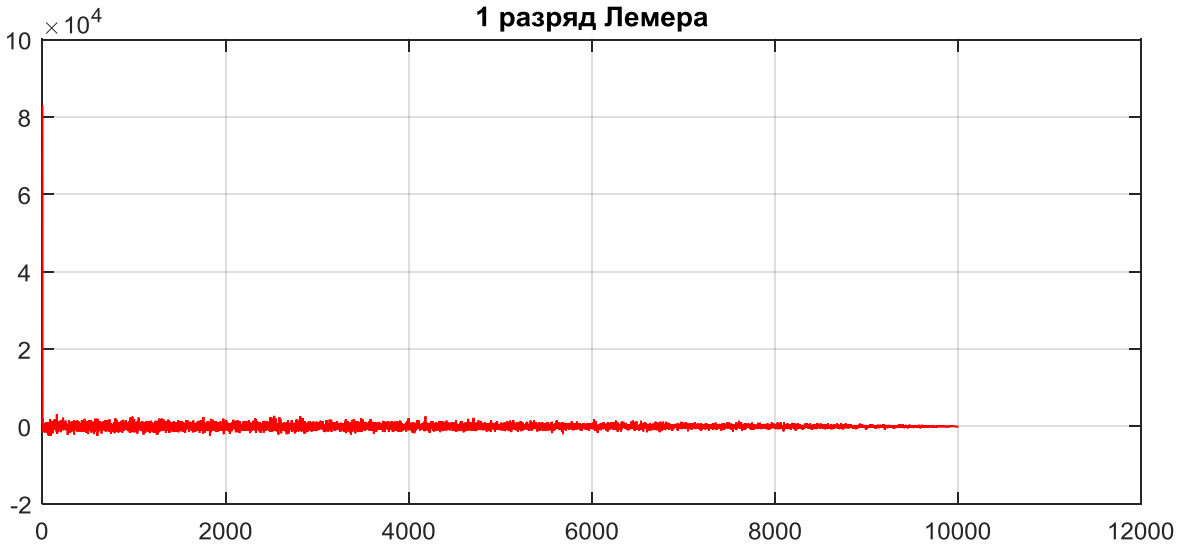


рисунок 2 - график автокорреляционной функции для алгоритма Лемера

Результаты тестов для 1, 2, 3 разрядов



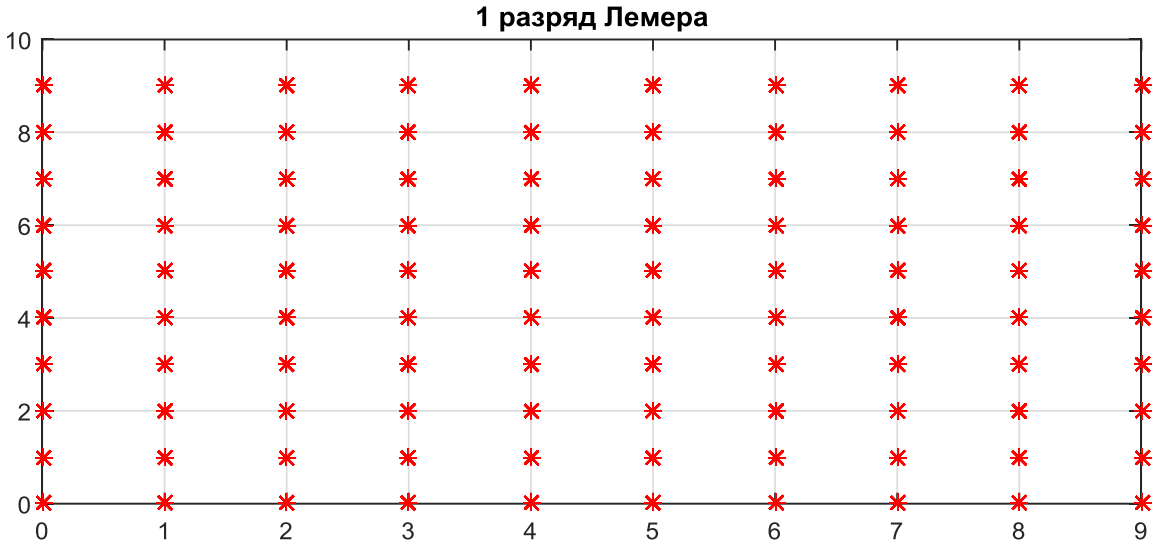
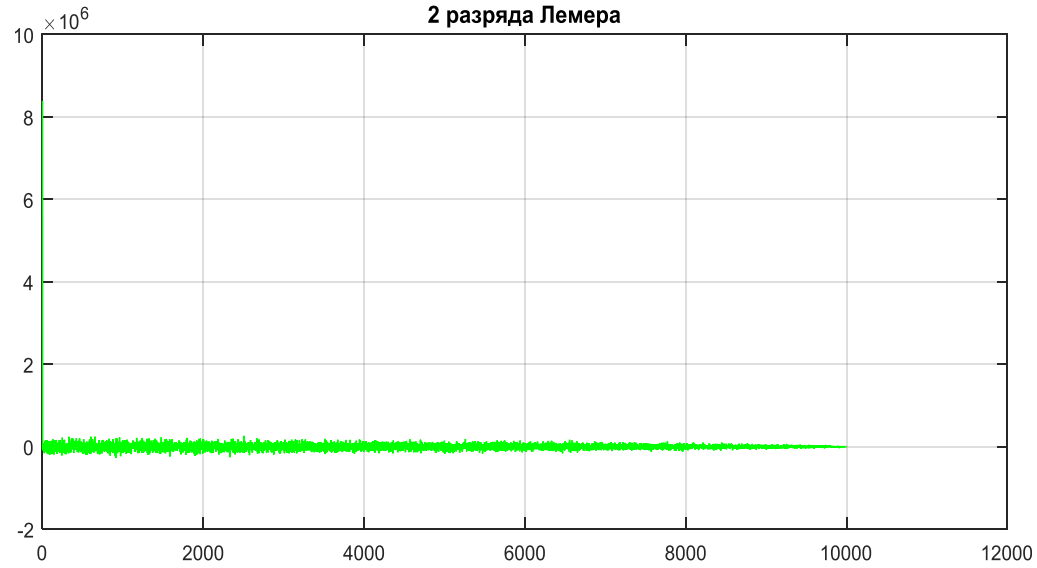


рисунок 3 - график автокорреляционной функции и тест на решетчатость для 1 разряда алгоритма Лемера.



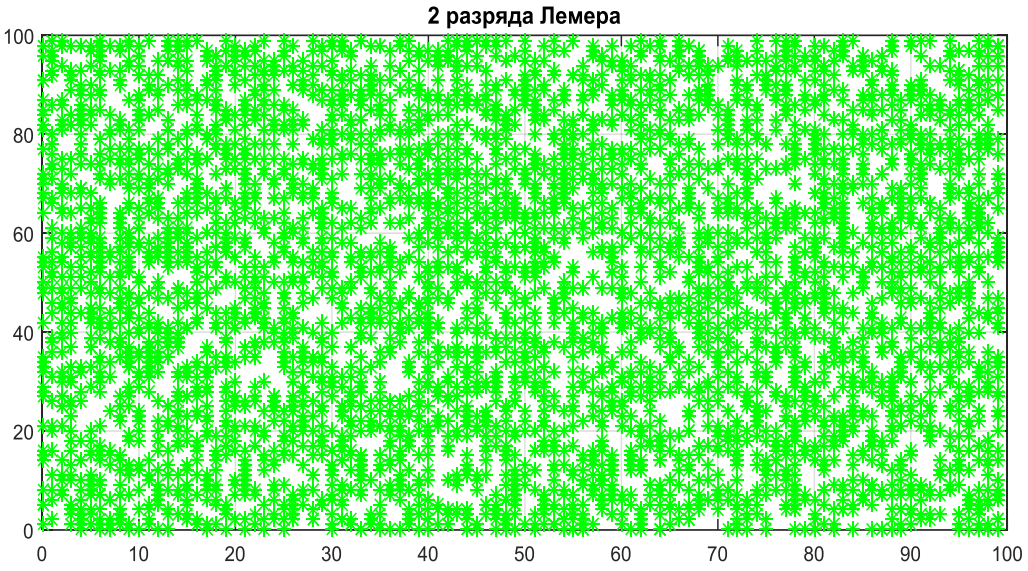


рисунок 4 - график автокорреляционной функции и тест на решетчатость для 2 разрядов алгоритма Лемера.

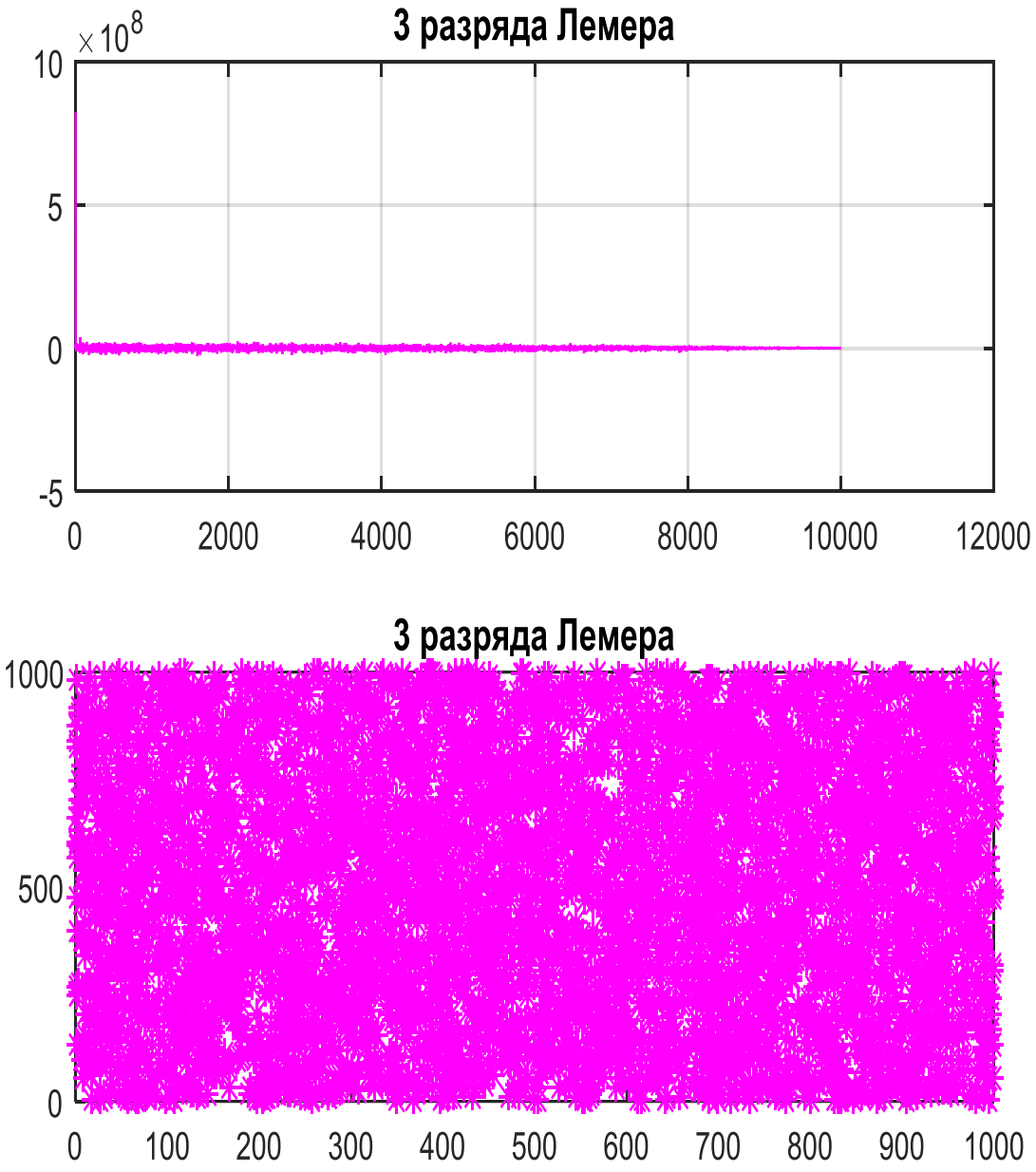


рисунок 5 - график автокорреляционной функции и тест на решетчатость для 3 разрядов алгоритма Лемера.

Алгоритм xorshift

Тест на решетчатость

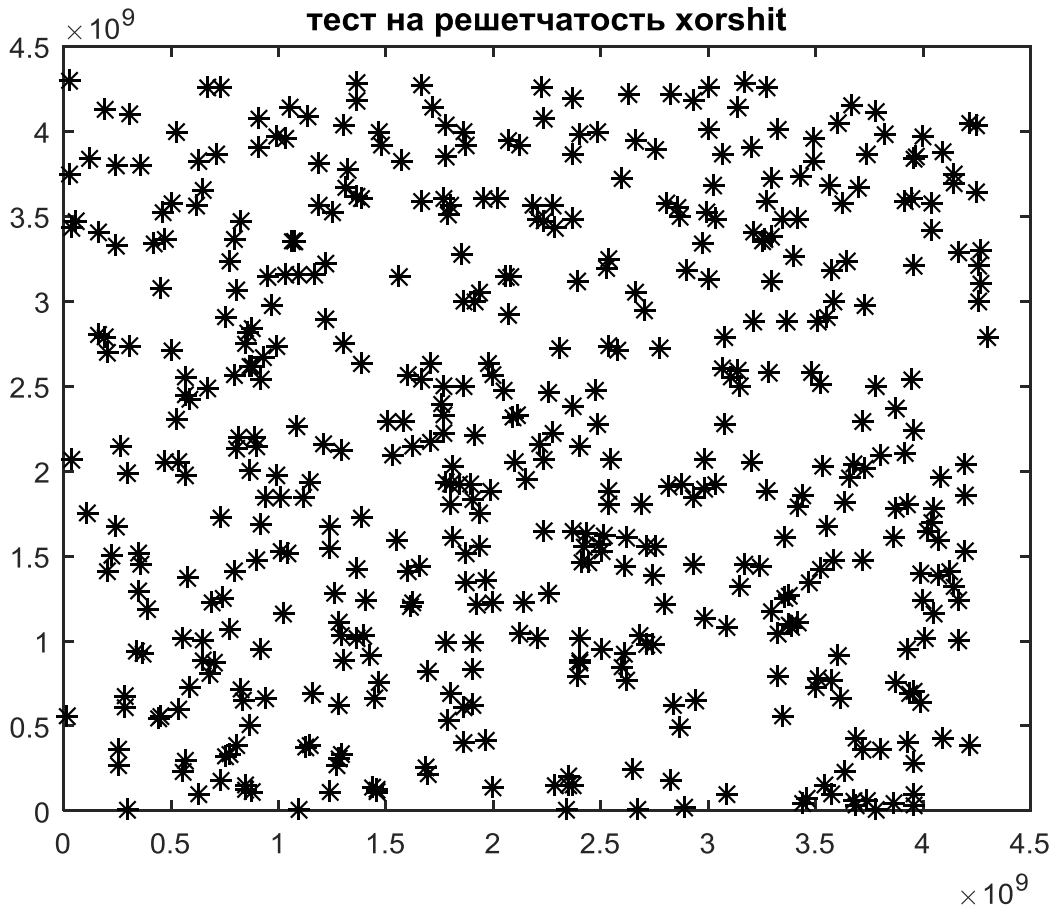


рисунок 6 - Тест на решетчастость для алгоритма xorshift

Автокорреляционная функция

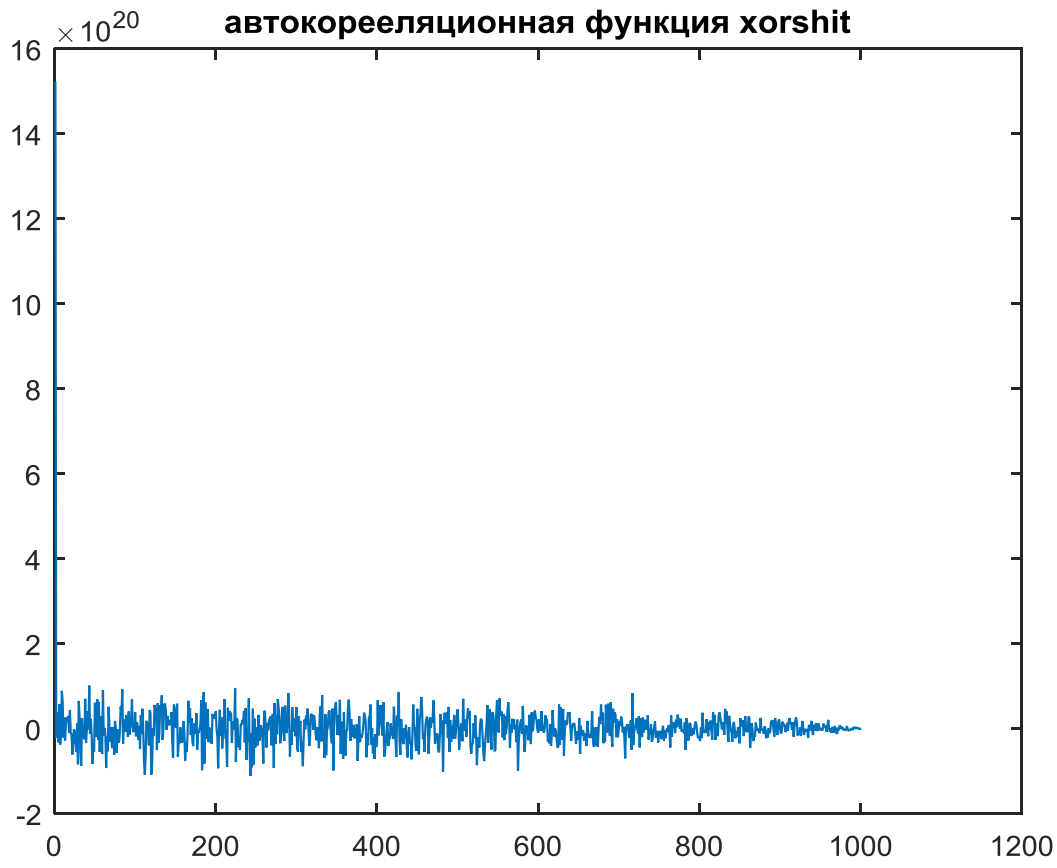


рисунок 7 - график автокорреляционной функции для алгоритма xorshift

Результаты тестов для 1, 2, 3 разрядов

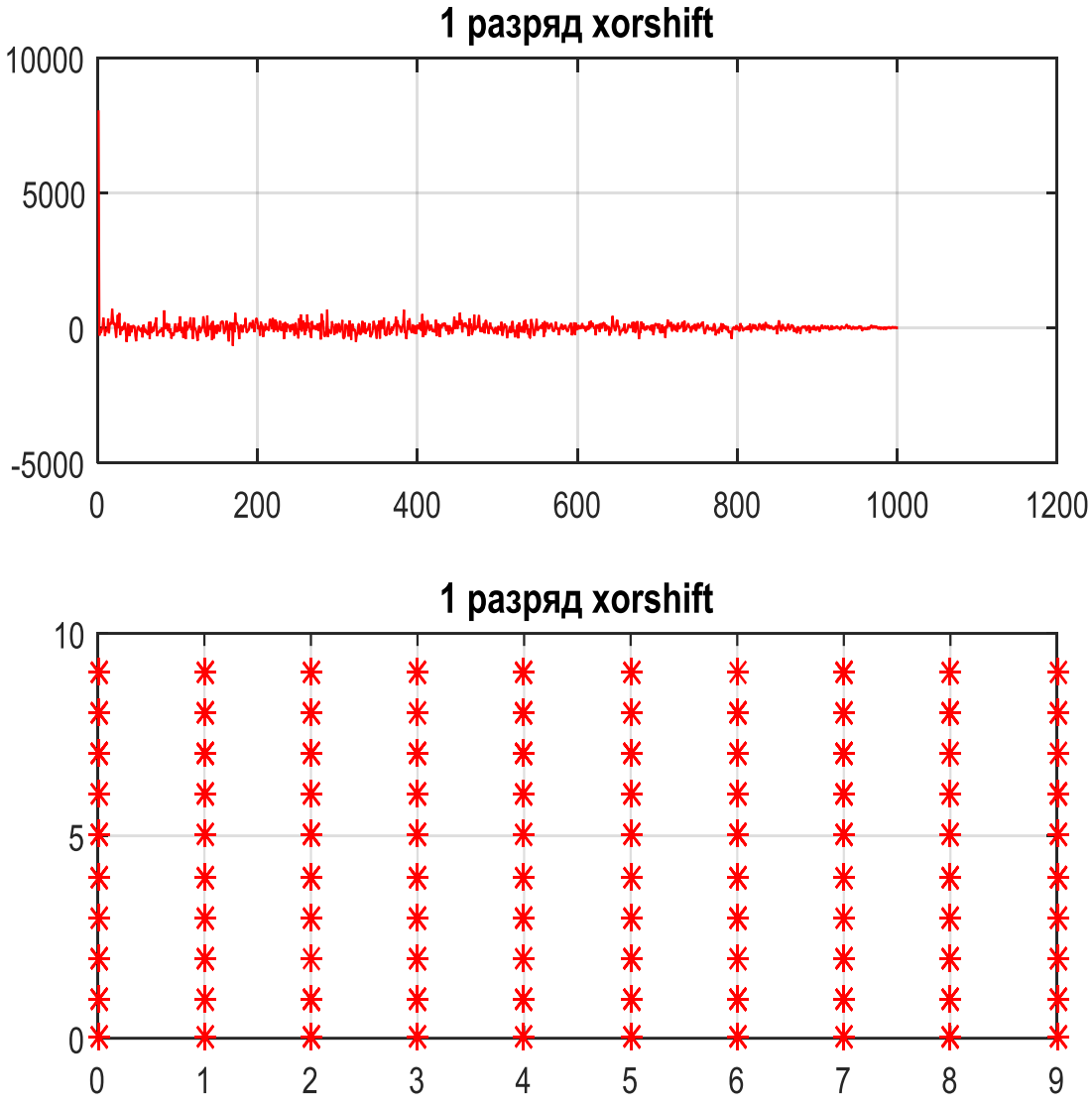


рисунок 8 - график автокорреляционной функции и тест на решетчатость для 1 разряда алгоритма xorshift

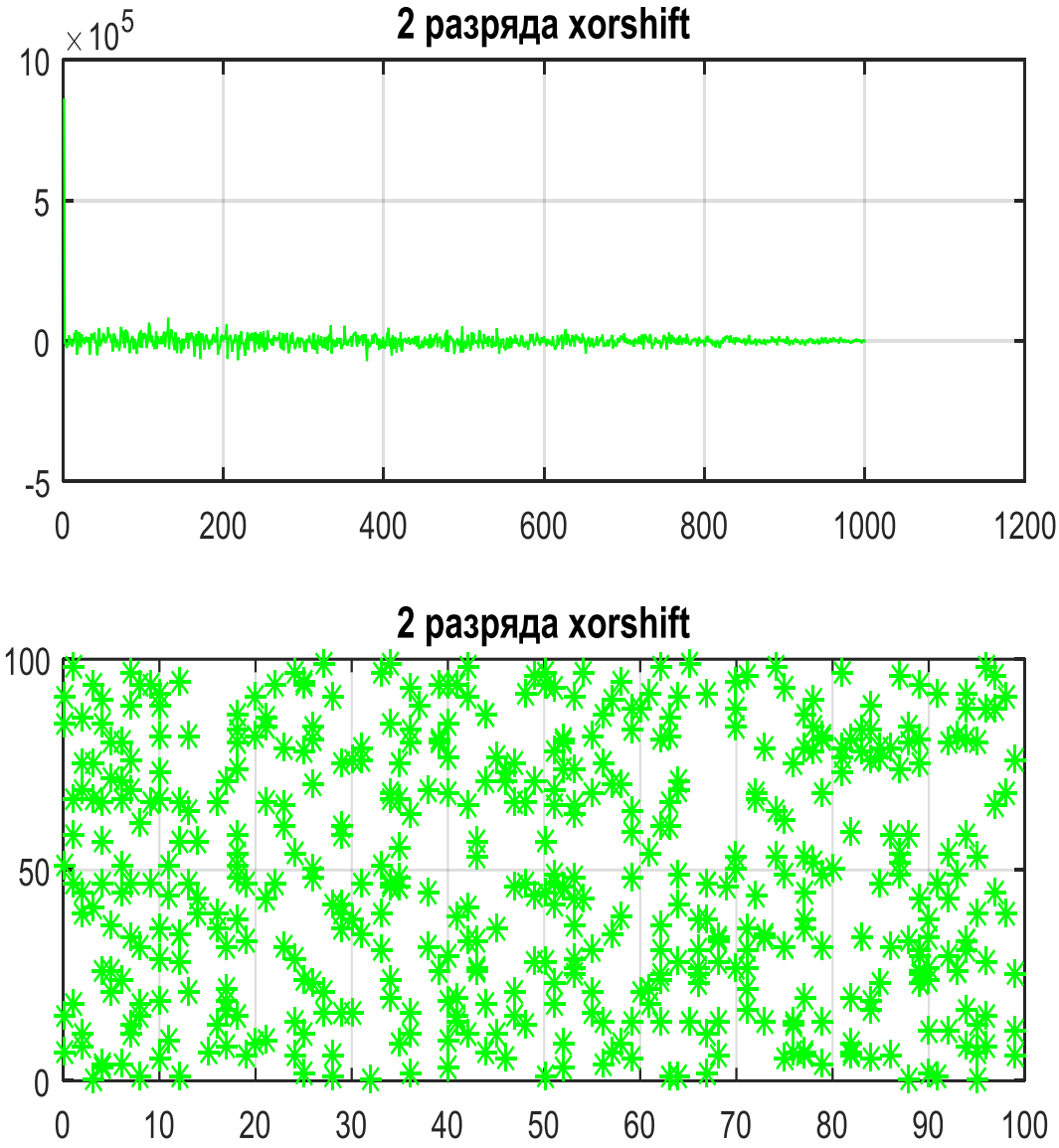


рисунок 9 - график автокорреляционной функции и тест на решетчатость для 2 разрядов алгоритма xorshift

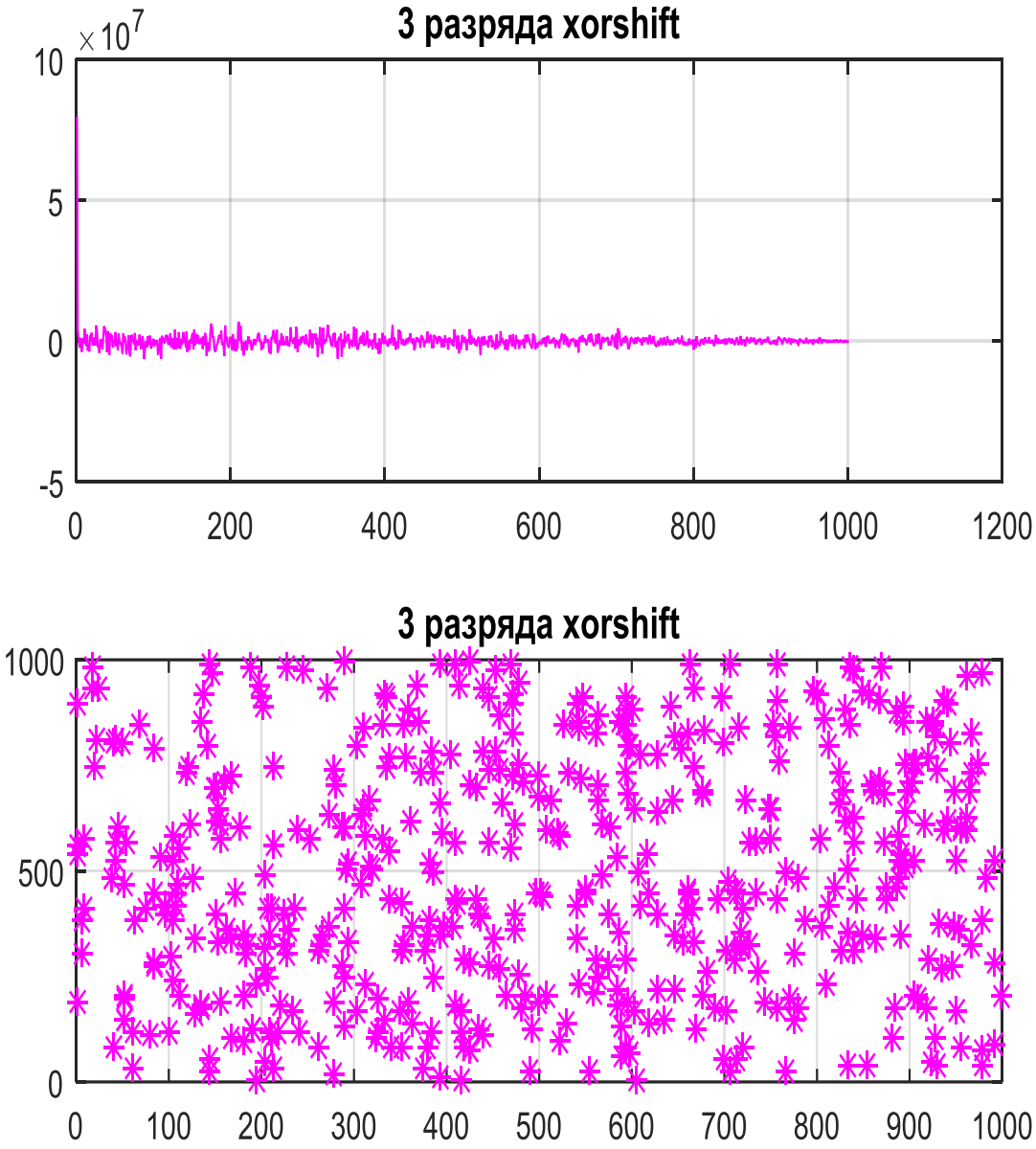


рисунок 10 - график автокорреляционной функции и тест на решетчатость для 3 разрядов алгоритма xorshift

Алгоритм Лемера для p=2^32,b= 2^13+1109 3^10+9752

Тест на решетчатость

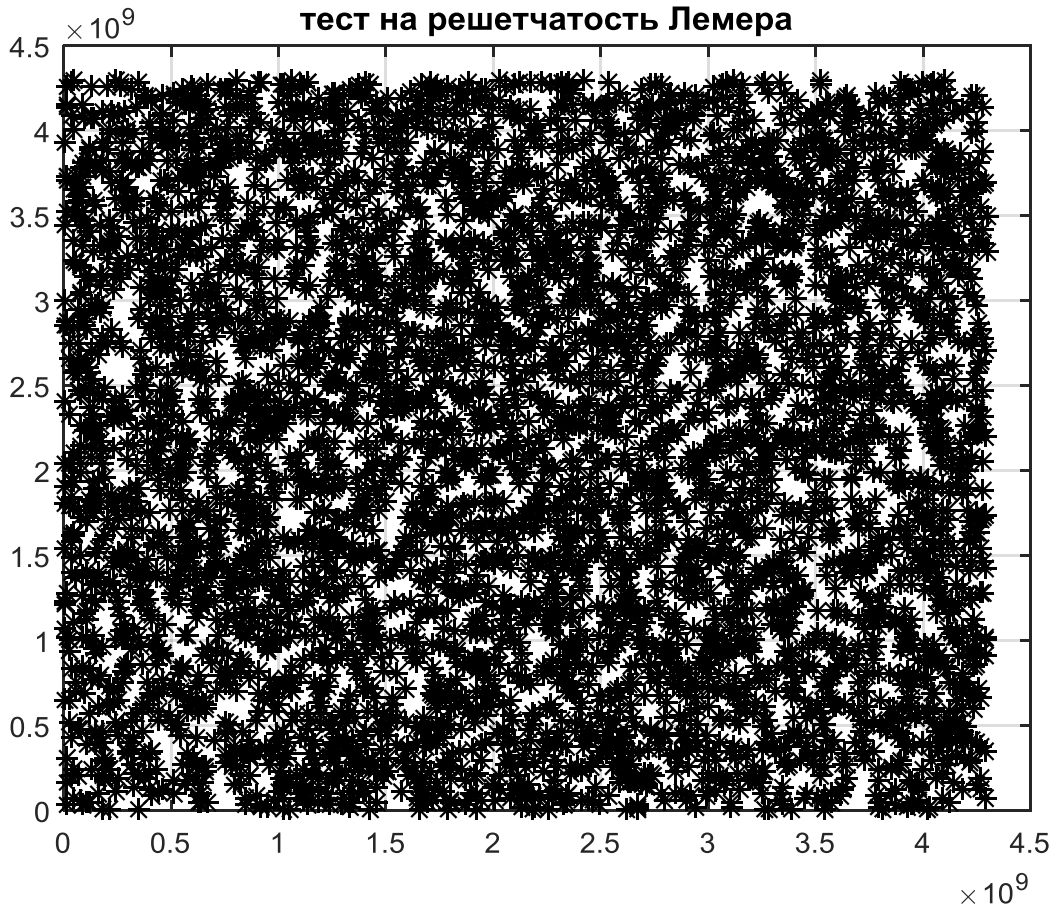


рисунок 11 - Тест на решетчастость для алгоритма Лемера.

Автокорреляционная функция

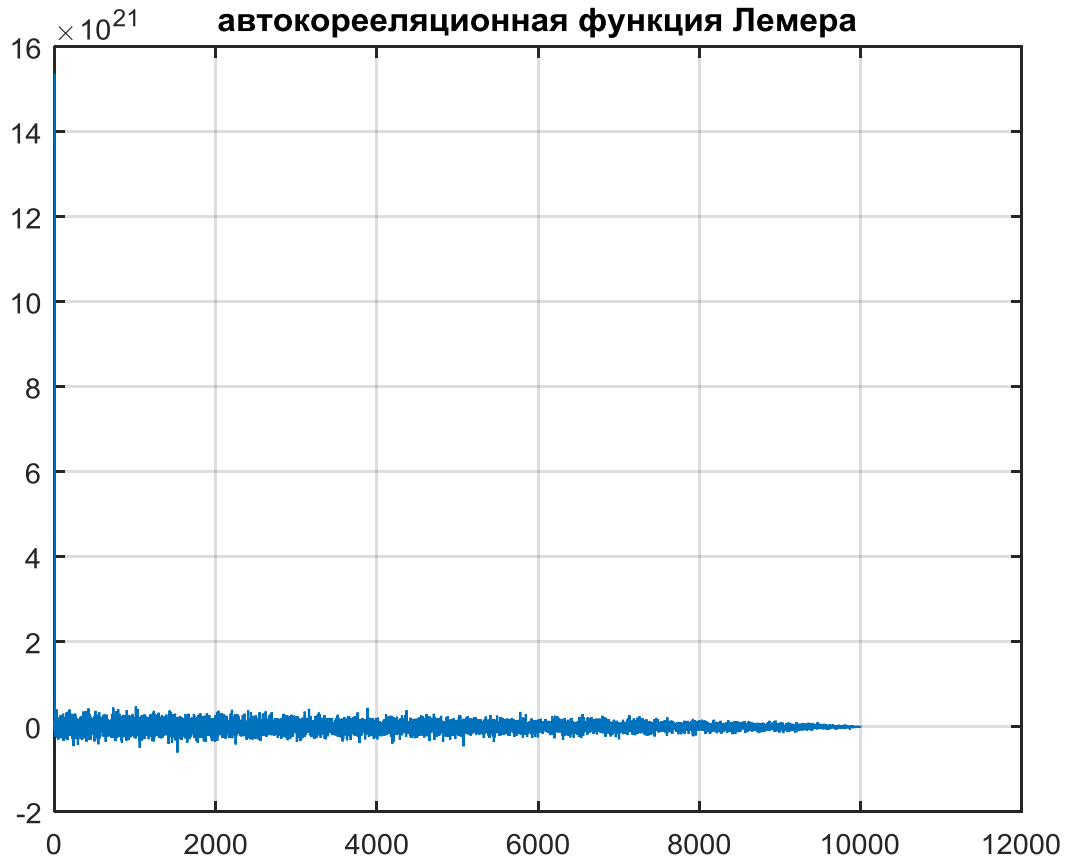


рисунок 12 - график автокорреляционной функции для алгоритма Лемера

Результаты тестов для 1, 2, 3 разрядов

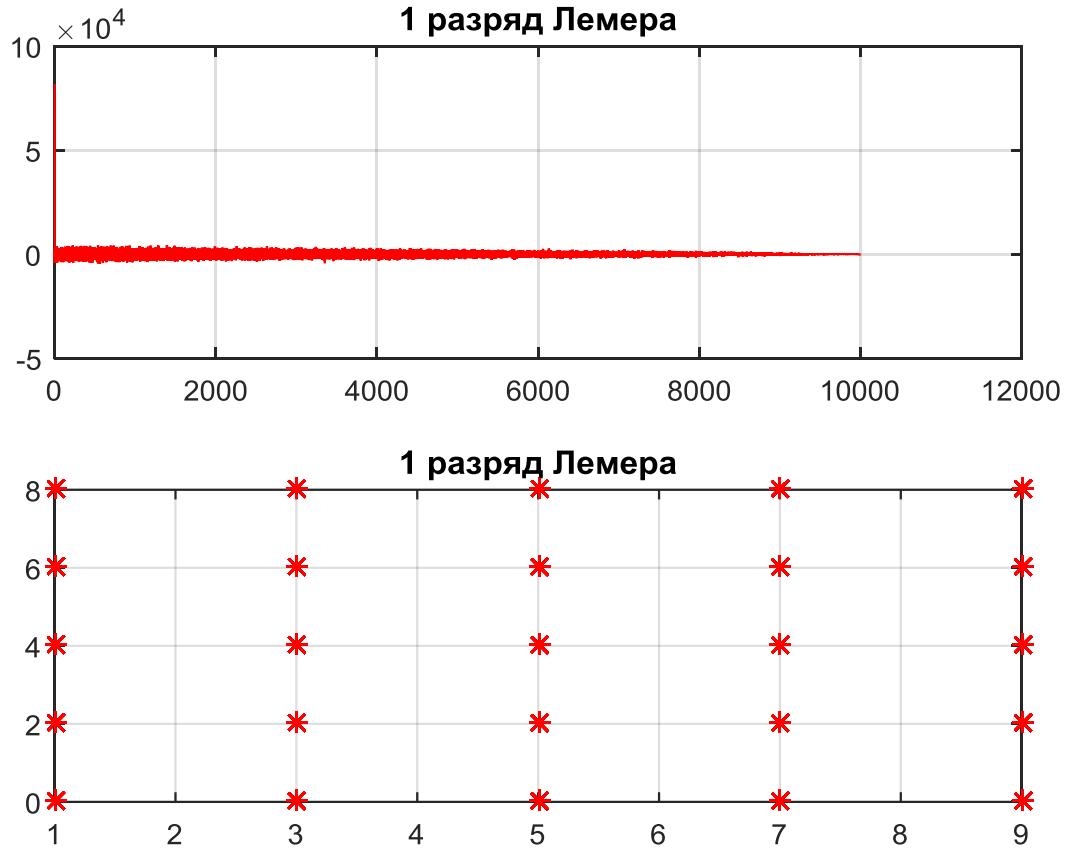


рисунок 13 - график автокорреляционной функции и тест на решетчатость для 1 разряда алгоритма Лемера.

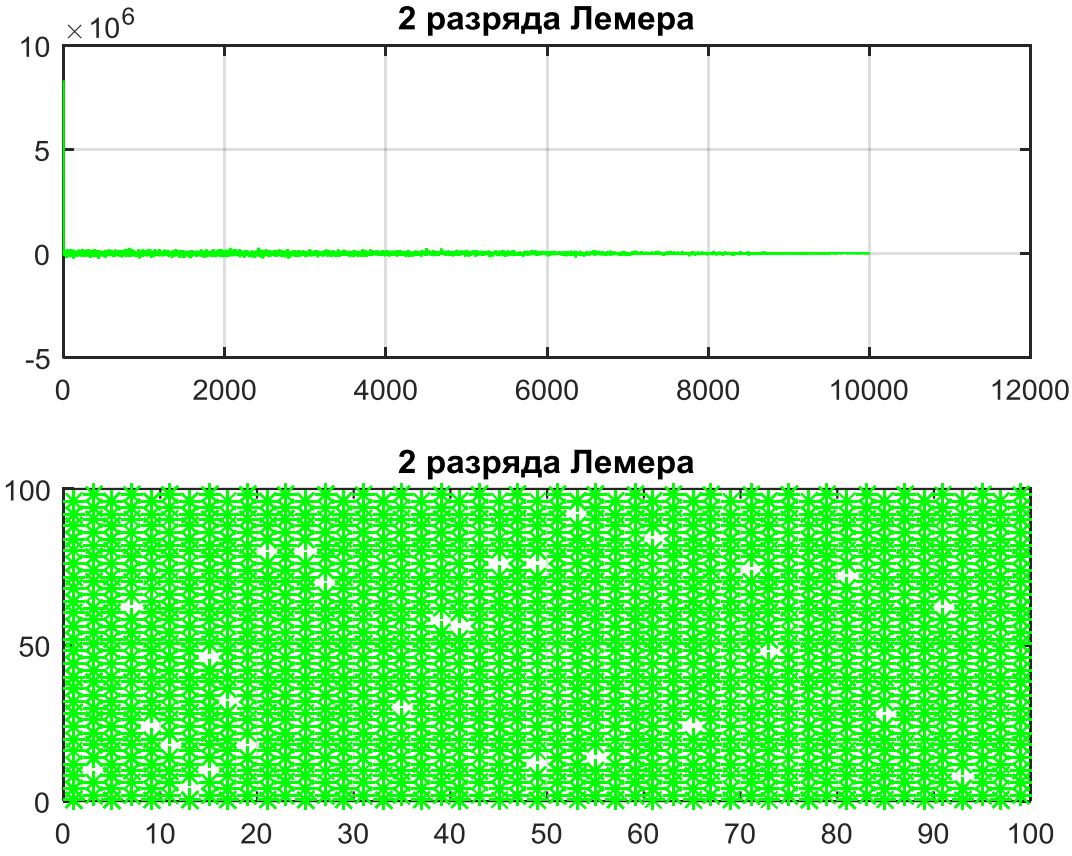


рисунок 14 - график автокорреляционной функции и тест на решетчатость для 2 разрядов алгоритма Лемера.

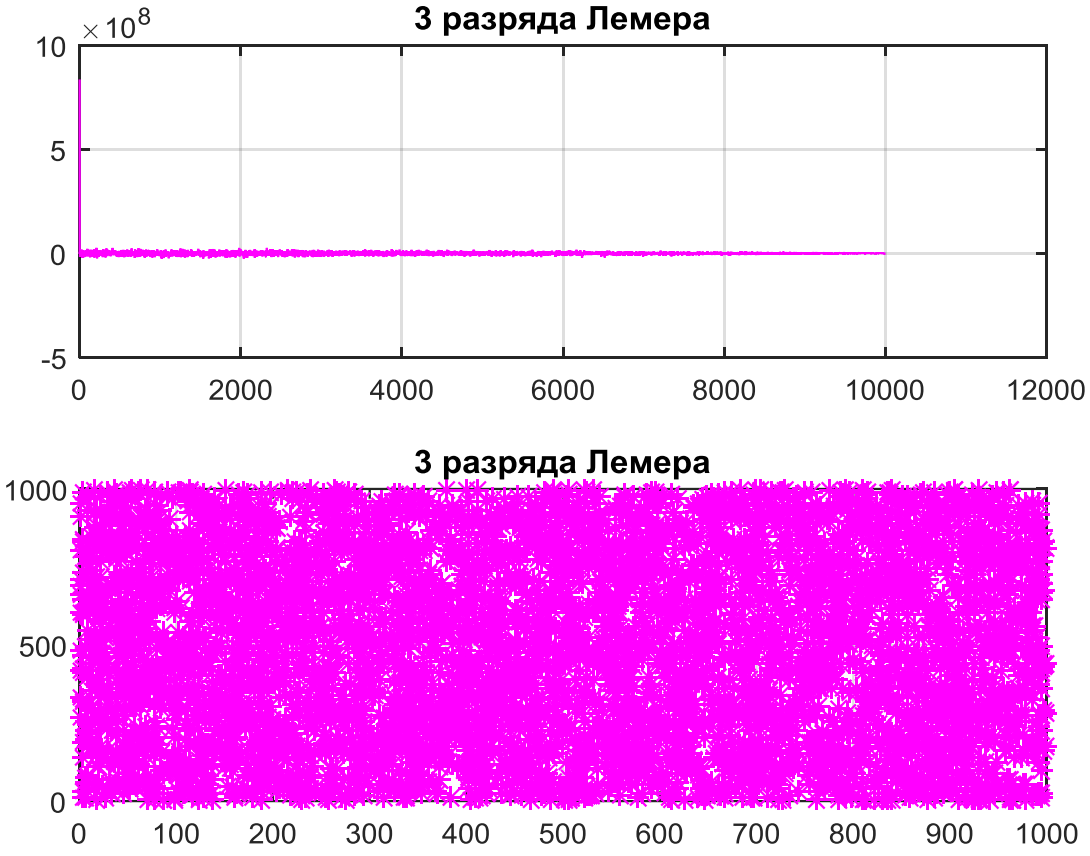


рисунок 15 - график автокорреляционной функции и тест на решетчатость для 3 разрядов алгоритма Лемера.

Выводы

В ходе выполнения лабораторной работы были реализованы два алгоритма псевдослучайных генераторов xorshift и Лемера. Были проведены тесты на решетчастость и построены автокорреляционные функции. Так же были протестированы младшие разряды алгоритмов.

По полученным графикам можно сделать следующие выводы:

Тест на решетчатость для алгоритма Лемера где p=2^31-1 показывает, что числа которые он генерирует, можно назвать псевдослучайными, так как все точки разбросаны и функциональных зависимостей не наблюдается.

Для младших разрядов такая зависимость ярко выражена для 1 разряда, а начиная с 2 отсутствует . Отсюда можно сделать вывод, что применение данного генератора возможно, если он будет выдавать числа с 2 разрядами и более.

По графику автокорреляционной функции можно сделать вывод о том, что генерируемые числа слабо коррелируют между собой.

Тест на решетчатость для алгоритма xorshift показывает, что числа которые он генерирует можно назвать псевдослучайными, так как все точки разбросаны и функциональных зависимостей не наблюдается.

Для младших разрядов такая зависимость ярко выражена для 1 разряда, а начиная с 2 отсутствует . Отсюда можно сделать вывод, что применение данного генератора возможно, если он будет выдавать числа с 2 разряда и более.

По графику автокорреляционной функции можно сделать вывод о том, что генерируемые числа слабо коррелируют между собой.

Тест на решетчатость для алгоритма Лемера где p=2^32 показывает, что числа которые он генерирует, можно назвать псевдослучайными, так как все точки разбросаны и функциональных зависимостей не наблюдается.

Для младших разрядов такая зависимость ярко выражена для 1 разряда, менее ярко для 2, а начиная с 3 отсутствует . Отсюда можно сделать вывод, что применение данного генератора возможно, если он будет выдавать числа с 3 разрядами и более.

По графику автокорреляционной функции можно сделать вывод о том, что генерируемые числа слабо коррелируют между собой.

По полученным результам оценки тестов можно сделать вывод о том, что оба генератора работоспособны и хорошо справляются со своей задачей. Результаты тестов схожи, по этому для использования подойдет любой из них. Однако генератор XorShift работает быстрее, так как реализован на простых операциях сдвига и сложения по модулю 2, в отличии от генератора Лемера, который требует более трудоемких операций, как умножение и сложение.

Проверяя функцией времени получили следующие результаты времени работы программ для генерации 100 000 чисел: Xorshift=1,1503 сек, Лемер=1,5257 сек. Полученные результаты показывают, что алгоритм Xorshift работает быстрее.

ПРИЛОЖЕНИЕ

Листинг программ реализующих данные алгоритмы

xorshift в среде python

```
MAX32 =0xffffffff
X = 123456789
Y = 362436069
Z = 521288629
W = 88675123

def xor_shift():
    global X, Y, Z, W
    t = X ^ (X << 11)
    X = Y
    Y = Z
    Z = W
    W = W^ (W >>  19) ^ t ^ (t >> 8)
    return W
'''
W1 = xor_shift() % MAX32
W2 = xor_shift() % MAX32
W3 = xor_shift()% MAX32
'''
my_file = open("chet.txt", "w")
my_file1 = open("nechet.txt", "w")
my_file2=open("some.txt", "w")
n=0
while n<1000:
    print(n)
    z=xor_shift() % MAX32
    n+=1
    print(z)
    my_file2.write(str(z))
    my_file2.write("\n")
   # n % 2 ==0 if (my_file.write( str(z) )) else my_file.write("\n")
    if n % 2 ==1:
        my_file.write(str(z))
        my_file.write("\n")
    else:
        my_file1.write(str(z))
        my_file1.write("\n")
my_file.close()
my_file1.close()
my_file2.close()


with open("chet.txt") as file:
    array = [row.strip() for row in file]
print(array)
```

Лемера в среде matlab

```
close all;
clear all;
clc;
%% генертаор Лемера
N=10000;
x=zeros(1,N-1);
x1=12345678;
x=[x1,x];
c=0;
b=7^5;
p=(2^31)-1;%2^31 по словам

for i=2:1:N

    x(1,i)=mod((x(1,i-1)*b+c),p);

end

%% тест на решетчатость
figure(1);
ne_chet=x(1,1:2:end);
chet=x(1,2:2:end);

plot(chet,ne_chet,'black*');
 title('тест на решетчатость Лемера');
 grid on

%% автокорреляция
 G  = autokorr( x );
 figure(2)
 plot(G);
 title('автокорееляционная функция Лемера');
 grid on

 %% младшие биты
 x1=mod(x,10);
 G1  = autokorr( x1 );
 ne_chet1=x1(1,1:2:end);
 chet1=x1(1,2:2:end);
 x2=mod(x,100);
 G2  = autokorr( x2 );
  ne_chet2=x2(1,1:2:end);
 chet2=x2(1,2:2:end);
 x3=mod(x,1000);
 G3  = autokorr( x3 );
 ne_chet3=x3(1,1:2:end);
 chet3=x3(1,2:2:end);
```