Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский Авиационный Институт»
(Национальный Исследовательский Университет)

Факультет №8 «Информационные технологии и прикладная математика»
Кафедра 805 «Математическая кибернетика»

Реферат

по теме

«Генерация случайных чисел из динамического хаоса»

2 семестр

Автор работы:

студент 1 курса, гр. М8О-103Б-21

Тысячный В. В.

Руководитель проекта:

Севастьянов В. С.

Дата сдачи:

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Случайность имеет множество применений в области науки, искусства, статистики, криптографии, игр и других областях. Например, случайное распределение в исследованиях помогает ученым проверять гипотезы, а также случайные и псевдослучайные числа находят применение в видеоиграх, они помогают разнообразить игровой процесс.

Случайные числа имеют применение в физике, например в исследованиях электронного шума, в инженерном деле и исследовании операций. Многие методы статистического анализа, требуют случайных чисел, например методы Монте-Карло[1].

Основное же применение последовательности случайных чисел (ПСЧ) находят в криптографии, например, при потоковом шифровании[2]. Одной из главных областей применения генераторов случайных чисел является формирование уникальных ключей для шифрования. В любой системе передачи секретных данных требуется множество ключей для всех пользователей системы. Однако, если нежелательный человек вдруг узнает ключ, использовавшийся для генерации псевдослучайных ключей, он сможет сгенерировать точно такие же ключи и вскрыть все передаваемые в системе данные. Следовательно, секретные ключи должны быть действительно случайными. Поэтому задача генерирования последовательностей настоящих случайных чисел представляет большой интерес для разработчиков криптосистем.

---

[1] Метод Монте-Карло URL: https://ru.wikipedia.org/wiki/Метод_Монте-Карло

[2] Лекция 8: Поточные шифры и генераторы псевдослучайных чисел. Часть 1URL: https://intuit.ru/studies/courses/691/547/lecture/12383

РАЗДЕЛ I

ТЕОРЕТИЧЕСКИЕ АСПЕКТЫ ИССЛЕДОВАНИЯ СЛУЧАЙНЫХ ЧИСЕЛ, ТЕОРИИ ХАОСА

Криптология - наука, занимающаяся методами шифрования и расшифровывания. Криптология состоит из двух частей - криптографии и криптоанализа. Криптография занимается разработкой методов шифрования данных, в то время как криптоанализ занимается оценкой сильных и слабых сторон методов шифрования, а также разработкой методов, позволяющих взламывать криптосистемы[3].

Исторически, криптография появилась сильно раньше цифровых информационных систем и использовалась в основном для обеспечения тайны переписки. Однако исторические алгоритмы шифрования полагаются в основном на неизвестность алгоритма третьей стороне, что делает их непригодными в современных реалиях. Современные шифры не полагаются на тайну алгоритма - все широко используемые алгоритмы опубликованы во множестве открытых источников. Вместо этого, современные криптосистемы используют ключ - некоторое значение, выбранное из множества возможных значений, называемого пространством ключей. Современная криптография образует отдельное научное направление, объединяющее математику и информатику. Практическое применение криптографии стало неотъемлемой частью жизни современного общества - её используют в таких отраслях как электронная коммерция, электронный документооборот, телекоммуникации и других.

Последовательность случайных чисел – это такая последовательность, что числа в ней не зависят друг от друга, иными словами, их нельзя предугадать. Генератор случайных чисел – алгоритм, который создает такие

---

[3] Криптология URL: https://ru.wikipedia.org/wiki/Криптология

последовательности. Стоит отличать ГСЧ от ГПСЧ[4] (генератор псевдослучайных чисел). В последовательности псевдослучайных чисел элементы ПОЧТИ не зависят друг от друга.

Генерация случайных чисел является неотъемлемой частью криптографии, например, ключ должен генерироваться настолько случайно, чтобы его значения невозможно было предугадать. В настоящее время криптографии есть достаточно много различных генераторов, но у каждого есть свои недостатки. Так среди ГПСЧ выделяют такие недостатки, как слишком короткий период, большая “случайность” некоторых элементов, чем других, обратимость и др. В связи с этим в криптографии есть определенные требования к генераторам[5]:

1. Однородное распределение: распределение чисел в последовательности должно быть однородным; это означает, что частота появления каждого числа должна быть приблизительно одинаковой.
2. Независимость: ни одно значение в последовательности не должно зависеть от других.
3. Невозможность предугадать следующие элементы последовательности.

Источники действительно случайных чисел найти трудно. Физические генераторы шумов, такие как, газовые разрядные трубки и имеющий течь конденсатор могут быть такими источниками, так же могут быть использованы системы, которым присуще свойство динамического хаоса. Динамический хаос - явление в теории динамических систем, при котором поведение нелинейной системы выглядит случайным; нерегулярное, непредсказуемое изменение состояния полностью детерминированной системы. Причиной появления хаоса

[4] Генераторы псевдослучайных чисел и их аспекты URL: https://ru.wikipedia.org/wiki/Генератор_псевдослучайных_чисел

[5] Требования к качественному гпсч URL: https://studfile.net/preview/1042727/page:3/

в таких системах является неустойчивость (чувствительность) по отношению к начальным условиям и параметрам: малое изменение начального условия со временем приводит к сколь угодно большим изменениям динамики системы[6]. К примеру, мы знаем траекторию движения механической системы, если даны начальные условия. Если бы система была устойчива, не хаотична, то изменив немного начальные условия, из которых начнется движение, то и новая траектория бы не сильно отличалась от прежней. Но если система была бы хаотичной, неустойчивой, то поначалу старая и новая траектории могли бы и быть близки, однако со временем траектории стали бы совершенно различны, то есть система проявила бы высокую чувствительность к начальным данным задачи о движении. Примером системы с динамическим хаосом может служить вынужденный осциллятор Ван-дер-Поля[7].

Осциллятор Ван-дер-Поля - осциллятор с нелинейным затуханием, подчиняющийся уравнению:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0$$

Для вынужденных колебаний:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = A\cos(\omega t),$$

где μ - коэффициент, характеризующий нелинейность и силу затухания колебаний, x - координата точки, зависящая от времени t, A - амплитуда внешнего гармонического сигнала, ω - его угловая частота.

Преобразуем дифференциальное уравнение второго порядка к системе из двух дифференциальных уравнений первого порядка при помощи алгебраической подстановки $\dot{x} = y$:

$$\begin{cases} \dot{x} = y \\ \dot{y} = \mu(1 - x^2)y - x + A\cos(\omega t) \end{cases}$$

[6] Аспекты теории хаоса URL: https://ru.wikipedia.org/wiki/Теория_хаоса#Чувствительность_к_начальным_условиям

[7] Назимов А. И. Колебания. Колебательные системы. Модели колебательных систем на примере дифференциальных уравнений URL: https://www.numamo.org/HTML/Articles/Oscillator.html

Запишем систему вынужденных осцилляторов Ван-дер-Поля:

$$\begin{cases} \dot{x}_1 = y_1 \\ \dot{x}_2 = y_2 \\ \dot{y}_1 = \mu_1(1 - {x_1}^2)y_1 - x_1 + A\cos(\omega t) \\ \dot{y}_2 = \mu_2(1 - {x_2}^2)y_2 - x_2 + A\cos(\omega t + \varphi) \end{cases}$$

с определенными константами $A, \omega, \varphi, \mu_1, \mu_2$ и начальными условиями

$$x_1(0) = x_{10}, \qquad x_2(0) = x_{20}$$

$$y_1(0) = y_{10}, \qquad y_2(0) = y_{20}$$

Моделирование движения этой системы может быть проиллюстрировано графиками, построенными для функций x_1, x_2, y_1, y_2 в области $t \in [0, T]$. В качестве средства для моделирования и построения может быть взята процедура *solve_ivp*[8] из библиотеки *scipy.integrate* в *python*. Она использует метод Рунге-Кутты 4-го порядка[9] и является весьма точной.

При моделировании получаем графики вида $x_1 = x_1(t)$ вида, представленного на рис. 1, и фазовый портрет $y_1 = y_1(x_1)$ первого осциллятора, представленный на рис. 2.

---

[8] Scipy.integrate.solve_ivp() URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

[9] Метод Рунге – Кутты URL: http://espressocode.top/runge-kutta-4th-order-method-solve-differential-equation/

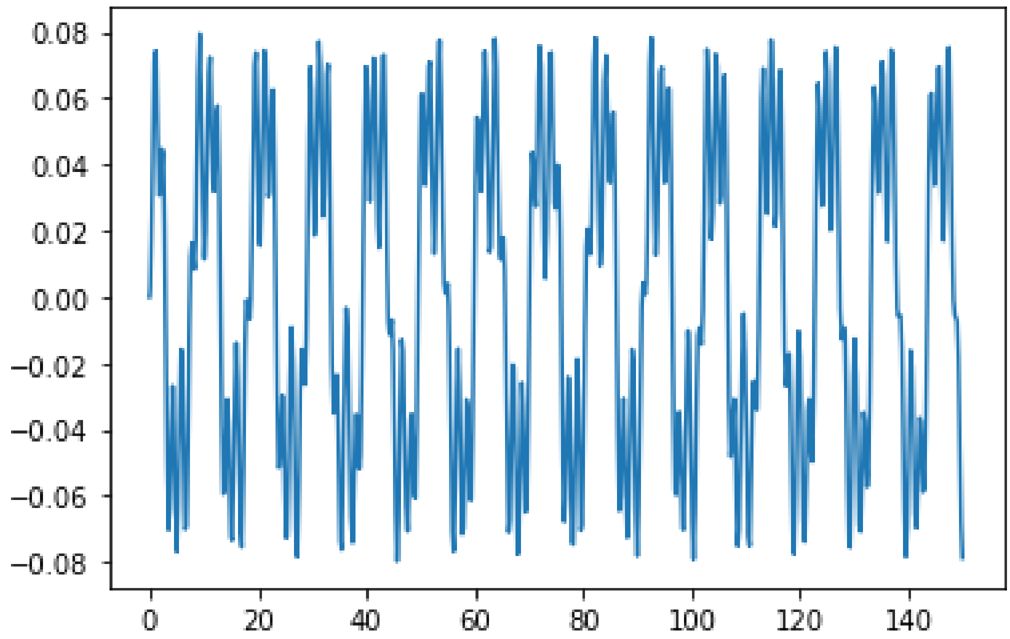


Рис. 1. Движение осциллятора Ван-дер-Поля

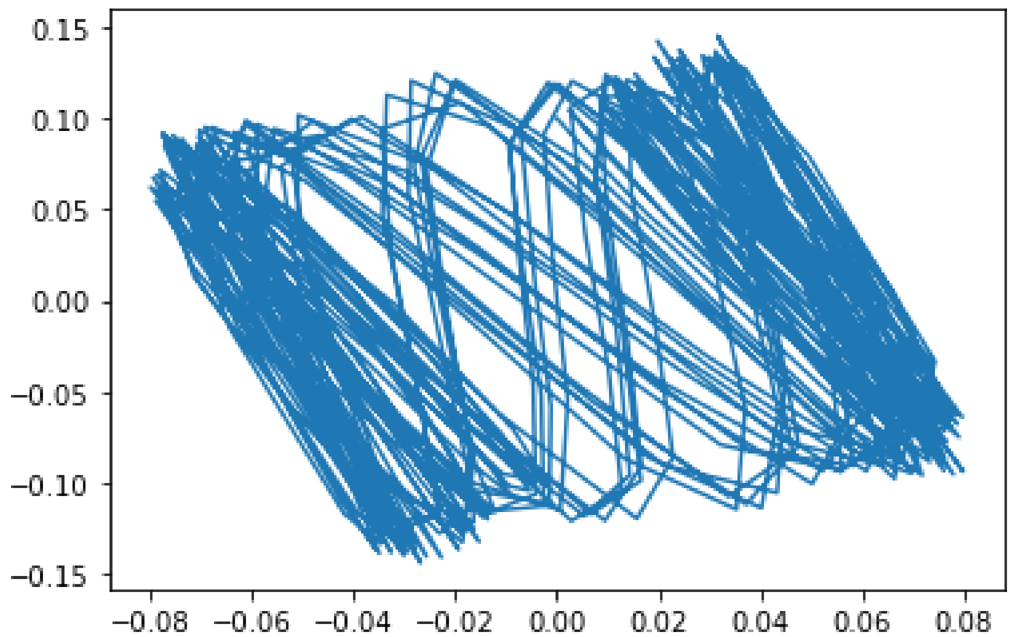


Рис. 2. Фазовый портрет осциллятора Ван-дер-Поля.

Из рис. 2 видно, что области внутри четырехугольников

$$(-0.03, -0.12) - (-0.06, 0) - (-0.05, 0.05) - (-0.013, -0.1)$$

$$(0.06, -0.07) - (-0.03, 0.07) - (0.055, 0.09) - (0.075, -0.03)$$

являются областями высокой степени смешивания и в них можно выбирать случайное число. Эта область будет соответствовать значениям по оси y из рис. 1, по модулю большим 0,02. Поэтому процедура выбора предлагается следующей:

1. Генерируются псевдослучайные числа $\varphi \in \left(0, \frac{\pi}{2}\right)$ и $T \in \left[\frac{100}{\omega}, \frac{500}{\omega}\right]$ (диапазон T можно менять).
2. Осуществляется моделирование осцилляторов
3. Находится значение $x_1 = x_1(T)$
4. Проверяется принадлежность значения к нужной области, т.е. проверяется, больше ли модуль этого числа 0,02.
5. Значение добавляется в список.
6. Полученный список нормируется.
7. Получен список случайных чисел.

Его случайность определяется хаотичностью осциллятора вкупе с приближенностью методов интегрирования ОДУ.

Для тестирования данной последовательности на случайность будут использоваться статистические тесты NIST, а точнее частотный побитовый тест[10] и тест на самую длинную последовательность единиц в блоке[11]. Этот пакет тестов, разработанный Лабораторией информационных технологий, специально для определения меры случайности двоичных последовательностей. Так как они работают с двоичными последовательностями, то уже имеющийся список десятичных чисел в промежутке от 0 до 1 требуется перевести в список двоичных чисел. Тестирование будет проводиться на первых четырех тестах: частотном побитовом, частотном блочном, тесте на одинаковые идущие подряд биты и

[10] A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications / A. Rukhin, J. Soto, J. Nechvatal // NIST Special Publi-cation 800-22 Revision 1a. p. 2-2 – 2-3

[11] A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications / A. Rukhin, J. Soto, J. Nechvatal // NIST Special Publi-cation 800-22 Revision 1a. p. 2-7 – 2-10

тесте на самую длинную последовательность единиц в блоке.

Суть первого теста заключается в нахождении отношения количества 0 в последовательности к количеству 1. В истинно случайных последовательностях оно должно быть равно 1. Для проверки гипотезы для заданной последовательности вычисляется сумма S таким образом, что каждая единица принимается за 1, а ноль за -1. По полученному значению вычисляется $s_{obs} = \frac{S}{\sqrt{n}}$, где n – длина последовательности. С помощью него вычисляется значение $P_{value} = erfc(\frac{s_{obs}}{\sqrt{2}})$, $erfc(x) = \frac{2}{\sqrt{\pi}}\int_x^{\infty} e^{-t^2}dt$, на основе чего делается вывод об истинности гипотезы.

Для частотного блочного теста последовательность делится на 64 подпоследовательности длины 100. Для каждого блока высчитывается отношение единиц к нулям и вычисляется критерий Хи-квадрат с N – количество блоков степенями свободы и ожиданием в ½. P-value здесь вычисляется через неполную верхнюю гамма-функцию: $P_{value} = igamc\left(\frac{N}{2}, \frac{\chi^2_{obs}}{2}\right), igamc(a, x) = \frac{1}{\Gamma(a)}\int_x^{\infty} e^{-t}t^{a-1}dt$, где Г – стандартная гамма-функция.

Тест на одинаковые идущие подряд биты проверяет «редкость» смены битов в последовательности. Для начала вычисляется доля единиц в общей массе и проверяется условие $|p - 0.5| < \frac{2}{\sqrt{n}}$, где p эта доля. Если данная проверка не удовлетворяется, то тест считается проваленным. Далее вычисляется количество знакоперемен V. По полученному числу вычисляется $P_{value} = erfc(\frac{|V-2np(1-p)|}{2\sqrt{2n}p(1-p)})$.

Последний тест определяет длину наибольшей последовательности 1 в блоке длины M и сравнивает с длиной для абсолютно случайной последовательности. Отклонение так же будет считаться с помощью Хи-квадрат по формуле $\chi^2 = \sum_{i=0}^{K}\frac{(\upsilon_i - R\pi_i)^2}{R\pi_i}$, где значения K, R, υ_i, π_i берутся из таблиц:

M	K	R
8	3	16
128	5	49
10000	6	75

ν_i, π_i	**M=8**	**M=128**	**M=10000**
ν_0, π_0	≤ 1, 0.2148	≤ 4, 0.1174	≤ 10, 0.0882
ν_1, π_1	2, 0.3672	5, 0.2430	11, 0.2092
ν_2, π_2	3, 0.2305	6, 0.2493	12, 0.2483
ν_3, π_3	≥ 4, 0.1875	7, 0.1752	13, 0.1933
ν_4, π_4		8, 0.1027	14, 0.1208
ν_5, π_5		≥ 9, 0.1124	15, 0.0675
ν_6, π_6			≥ 16, 0.0727

K в данном уравнении будет являться степенью свободы. По найденному значению Хи-квадрат находится $P_{value} = igamc(\frac{K}{2}, \frac{\chi^2_{obs}}{2})$.

Если полученные P_{value} будут больше 0,01, то последовательность, действительно, можно считать случайной.

РАЗДЕЛ II

ЭКСРПЕРИМЕНТАЛЬНАЯ ГЕНЕРАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТИ ЧИСЕЛ И ПРОВЕРКА НА СЛУЧАЙНОСТЬ

Сначала обозначим библиотеки, которые будут использоваться в коде:

```
from scipy.integrate import solve_ivp
from matplotlib import pyplot as plt
import random
```

Из библиотеки *scipy* раздела *integrate* импортируем функцию *solve_ivp*, которая численно интегрирует систему дифференциальных уравнений, для того чтобы сгенерировать нужную нам последовательность чисел из исходной функции. Из библиотеки *matplotlib* импортируем раздел *pyplot*, отвечающий за построение графиков для визуализации результатов. Библиотека random будет использоваться для случайного выбора значений из последовательности.

Введем константы, которые будут использоваться в построении функции как параметры (их можно задать любыми):

```
M = [5, 7, 1]
k2 = [100, 70, 120]
```

Далее пропишем функцию, описывающую работу осциллятора Ван-дер-Поля, которая и будет в дальнейшем интегрироваться:

```
def ab_sys(t, Y):
    X = [Y[0] * abs(Y[0]), Y[1] * abs(Y[1]), Y[2] * abs(Y[2])]
    return [Y[3],
            Y[4],
            Y[5],
            -k2[0] / M[0] * X[0] + k2[1] / M[0] * (X[1] - X[0]) + k2[2] / M[0] * (X[2] - X[0]),
            -k2[1] / M[1] * (X[1] - X[0]),
            -k2[2] / M[2] * (X[2] - X[0])]
```

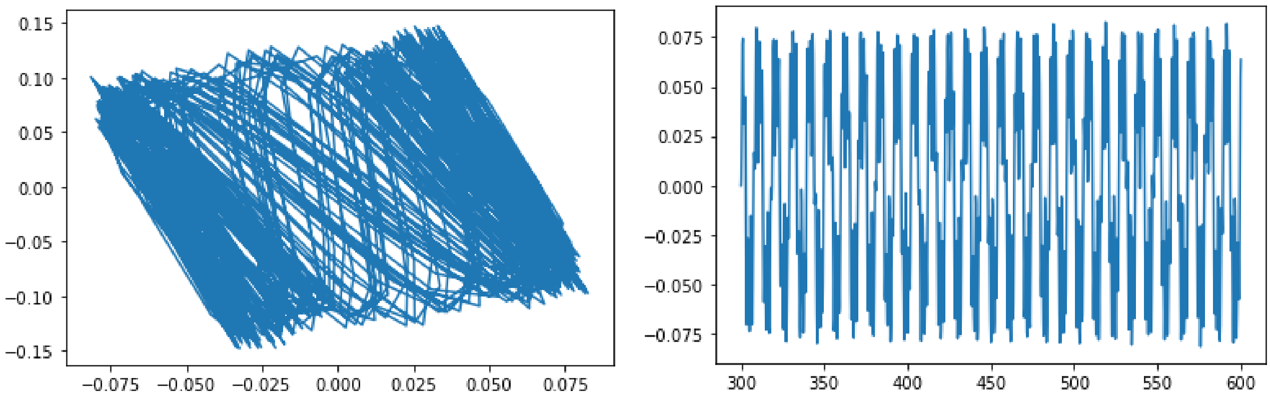
Проинтегрируем данную функцию с помощью *solve_ivp* на промежутке от 300 до 600 (его так же можно изменить):

```
sol = solve_ivp(ab_sys, [300, 600], [0, 0.1, 0.1, 0, 0, 0])

print(sol)
```

Для наглядности с помощью функций *plot* и *show* построим и выведем фазовый портрет полученной функции и график ее зависимости от времени:

```
plt.plot(sol.y[0], sol.y[2])
plt.show()

plt.plot(sol.t, sol.y[0])
plt.show()
```

Приданных параметрах графики будут выглядеть таким образом:



Выберем из значений функции случайные 202 числа с помощью функции *chois* из библиотеки *random*, которые будут по модулю больше 0,02:

```
Arr = []
i = 1
while i <= 202:
    OY = random.choice(sol.y[2])
    if abs(OY) > 0.02:
        Arr.append(abs(OY))
        i+=1
print(Arr)
```

Для того, чтобы полученную последовательность можно было считать случайной в диапазоне (0; 1) нормализуем ее, вычев из всех ее значений минимальное из них и поделив на максимальное. Полученные значения все будут в требуемом диапазоне, но будет заведомо известно, что в последовательности есть 0 и 1 (то минимальное и максимальное число после нормализации), поэтому удалим их. Таким образом, получим последовательность из 200 случайных чисел в диапазоне от 0 до 1:

```
Min = min(Arr)
PRN = []
k=0
for i in range(len(Arr)):
    PRN.append(Arr[i]-Min)

Max = max(PRN)

for i in range(len(Arr)):
    PRN[i] /= Max
    if PRN[i]<0.5: k+=1

PRN.remove(1)
PRN.remove(0)

n=len(PRN)
print(PRN)
```

Далее, чтобы протестировать, требуется перевести последовательность случайных чисел в двоичную. Для этого домножим каждое число на $2^{32} - 1$, так как с 32 битными двоичными числами будет удобнее работать, и добавим ведущие нули, чтобы все числа имели ровно 32 бита:

```
Bit=[]
for i in range(n):
    b=bin(int(PRN[i] * 2**32-1))[2:]
    while len(b)!=32:
        b='0'+b
    Bit.append(b)
print(Bit)
```

Теперь в переменной *Bit* хранится последовательность случайных чисел в двоичном виде, которую остается протестировать.

В тестах часто используется распределение Хи-квадрат, поэтому пропишем для него отдельную функцию, в которую подается список *o* с входными данными в количестве равном степени свободы и список *e* с эталонными значениями:

```python
def X2(o, e):
    x2=0
    for i in range(len(o)):
        x2+=(o[i] - e[i])**2/e[i]
    return x2
```

Пропишем первый частотный побитовый тест, который посчитает количество 1 во всей последовательности и посчитает распределение Хи-квадрат для полученных значений:

```python
import math

Sum = 0
for i in range(len(Bit)):
    for j in range(len(Bit[i])):
        if Bit[i][j] == '1':
            Sum += 1
        else:
            Sum -= 1

n = len(Bit) * len(Bit[0])
Sum = abs(Sum) / math.sqrt(len(Bit) * len(Bit[0]))
p_value = 1 - math.erf(Sum / math.sqrt(2))
p_value
```

Для второго теста производится разбиение последовательности *Bit* на блоки длины 100, соединив изначально все в единую последовательность S, и для каждого блока считается доля единиц, которая хранится в *cnt*, по ним считается Хи-квадрат и P-value:

```python
import scipy

S = ''
Arr = []
S = ''.join(Bit)
m = 100
for i in range(0, len(S), m):
    Arr.append(S[i:i + m])
if len(S) % m != 0:
    Arr = Arr[:-1]
cnt = []
for i in Arr:
    cnt.append(i.count('1'))
x2 =  2 * X2(cnt, [m / 2] * len(cnt))
p_value = 1 - scipy.special.gammainc(len(cnt) / 2, x2 / 2)
p_value
```

Третий тест вычисляет P-value по количеству смен бита, хранящемуся в V, в последовательности S:

```
n = len(S)
p = S.count('1') / n
if abs(p - 1 / 2) >= 2 / math.sqrt(n):
    print("Fall")
else:
    V = 1
    for i in range(n - 1):
        if S[i] != S[i + 1]:
            V += 1
    p_value = 1 - math.erf(abs(V - 2 * n * p * (1 - p))
      / (2 * math.sqrt(2 * n) * p * (1 - p)))
    print(p_value)
```

Для теста на самую длинную последовательность единиц в блоке нам понадобится подфункция, вычисляющая максимальную последовательность единиц в подаваемом блоке, в которую подается блок, и возвращается искомая длина:

```
def Max_Seq_1(S):
    k = 0
    Maxk = 0
    for i in range(len(S)):
        if S[i] == '1':
            k += 1
        else:
            if Maxk < k:
                Maxk = k
            k = 0
    if Maxk < k:
        Maxk = k
    return Maxk
```

Сам тест будет разбивать последовательность *Bit* на блоки длиной 128 символов, т.е. по 4 числа, для каждого такого блока будет вызываться функция *MaxSeq1*. Так как длинна блока 128, то M = 128, следовательно из приведенных ранее таблиц считаем соответствующие значения для R, ν_i, π_i и вычисляем Хи-квадрат:

```
Seq1 = []
for i in range(int(n // 128)):
    Str = S[128 * i:128 * i + 128]
    Seq1.append(Max_Seq_1(Str))
V = []
k = 0
V = []
k = 0
for i in range(5):
    k += Seq1.count(i)
V.append(k)
for i in range(5, 9):
    V.append(Seq1.count(i))
V.append(int(n // 128 - sum(V)))
x2 = X2(V, [49 * 0.1174, 49 * 0.2430, 49 * 0.2493, 49 * 0.1752, 49 * 0.1027, 49 * 0.1124])
p_value = 1 - scipy.special.gammainc(5 / 2, x2 / 2)
p_value
```

Полученные в результате значения P-value являются характеристиками полученной последовательности, на их основе уже можно сделать вывод, является полученная последовательность, действительно, псевдослучайной.

ЗАКЛЮЧЕНИЕ

В результате тестирования было получено, что сгенерированная данным методом ПСЧ имеет среднее значение p-value около 0,4, что отвечает поставленной задаче в достижении результата большего 0,01. Но в настоящее время для реального использования на практике, в криптографии требуется гораздо больший результат, в частности хотя бы 0,95. Иногда написанная программа может выдать такой результат, но он будет побочным в связи с использованием библиотеки *random*, поэтому данные случаи не будут являться отражением действительности.

Таким образом, в ходе проекта был создан ГПСЧ, основанный на одномерной нелинейной простейшей динамической системе с хаосом, которая удовлетворяет наименьшим криптографическим требованиям, но не отражает в себе реальных требований для использования.

СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Petrosjan, L.A. and Murzov, N.V. (1966). Game-theoretic problems of mechanics. Litovsk. Mat. Sb. 6, 423–433.
2. Фомичёв В. М. Дискретная математика и криптология: Курс лекций / под ред. Н. Д. Подуфалов — М.: Диалог-МИФИ, 2013. — 397 с. — ISBN 978-5-86404-185-7
3. Дональд Э. Кнут. Глава 3. Случайные числа // Искусство программирования = The Art of Computer Programming. — 3-е изд. — М.: Вильямс, 2000. — Т. 2. Получисленные алгоритмы. — 832 с. — 7000 экз. — ISBN 5-8459-0081-6 (рус.) ISBN 0-201-89684-2 (англ.).
4. Юрий Лифшиц. Лекция 9: Псевдослучайные генераторы // Современные задачи криптографии. — Курс лекций.
5. Жельников В. Псевдослучайные последовательности чисел // Криптография от папируса до компьютера. — М.: ABF, 1996. — 335 с. — ISBN 5-87484-054-0.
6. Соболь И. М. Метод Монте-Карло. — М.: Наука, 1968. — 64 с. — (Популярные лекции по математике). — 79 000 экз.
7. L’Ecuyer, Pierre. Random Number Generation // Springer Handbooks of Computational Statistics: Глава. — 2007. — С. 93—137. — doi:10.1002/9780470172445.ch4.
8. Van der Pol, B. On "relaxation-oscillations"/B. Van der Pol // Phylosophical Magazine and Journal of Science S7. -1926. -Vol.2. -№11. -P.978
9. Van der Pol, B. Forced Oscillations in a Circuit with non-linear Resistance.(Reception with reactive Triode.) / B. Van der Pol // Phylosophical Magazine and Journal of Science S7. -1927. -Vol.3 -№13. -P.65

10. Van der Pol, B. The heartbeat considered as a relaxation oscillation, and an electrical model of the heart/ B. Van der Pol, J. Van der Mark // Phylosophical Magazine and Journal of Science S7. -1928. -Vol.6. -№38. -P.763

11. Турчак, Л.И. Основы численных методов / Л.И. Турчак, П.В. Плотников. - М.: Физматлит, 2005. –304с.

12. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications / A. Rukhin, J. Soto, J. Nechvatal // NIST Special Publication 800-22 Revision 1a

13. Гринвуд, Синди; Никулин, М.С. (1996), Руководство по тестированию хи-квадрат, Нью-Йорк: Wiley, ISBN 0-471-55779-X

14. Никулин, М.С. (1973), "Критерий хи-квадрат на нормальность", Труды Международной Вильнюсской конференции по теории вероятностей и математической статистике, 2, стр. 119–122