LFSR Documentation

Release latest

May 21, 2021

Contents

Linear Feedback Shift Register. This package includes following components:

CHAPTER 1

LFSR

1.1 Links:

- **Github Page** - http://nikeshbajaj.github.io/Linear_Feedback_Shift_Register
- **Documentation** - https://lfsr.readthedocs.io
- **Github** - https://github.com/Nikeshbajaj/Linear_Feedback_Shift_Register
- **PyPi-project** - https://pypi.org/project/pylfsr
- **Installation:** *pip install pylfsr*

1.2 Installation

Requirement : *numpy*, *matplotlib*

With pip:

```
pip install pylfsr
```

Build from source

Download the repository or clone it with git, after cd in directory build it from source with

```
python setup.py install
```

CHAPTER 2

Examples

2.1 Basic Examples:

2.2 Example 1: 5-bit LFSR with feedback polynomial: $x^5 + x^2 + 1$

default feedback polynomial is $p(x) = x^5 + x^2 + 1$

```
# import LFSR
import numpy as np
from pylfsr import LFSR

L = LFSR()

# print the info
L.info()

5 bit LFSR with feedback polynomial  x^5 + x^2 + 1
Expected Period (if polynomial is primitive) =  31
Current :
State        :  [1 1 1 1 1]
Count        :  0
Output bit   : -1
feedback bit : -1
```

```
L.next()
L.runKCycle(10)
L.runFullCycle()
L.info()
```

2.3 Example 2: 5-bit LFSR with custum state and feedback polynomial

```
state = [0,0,0,1,0]
fpoly = [5,4,3,2]
L = LFSR(fpoly=fpoly,initstate =state, verbose=True)
L.info()
tempseq = L.runKCycle(10)
L.set(fpoly=[5,3])
```

2.4 Example 3: 23-bit LFSR: $x^{23}+ x^{18}+1$

```
L = LFSR(fpoly=[23,18],initstate ='random',verbose=True)
L.info()
L.runKCycle(10)
L.info()
seq = L.seq
```

2.5 Example 4: 23-bit LFSR: $x^{23}+ x^{5}+1$

```
fpoly = [23,5]
L1 = LFSR(fpoly=fpoly,initstate ='ones', verbose=False)
L1.info()
```

```
23 bit LFSR with feedback polynomial  x^23 + x^5 + 1
Expected Period (if polynomial is primitive) =  8388607
Current :
 State        :  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 Count        :  0
 Output bit   :  -1
 feedback bit :  -1
```

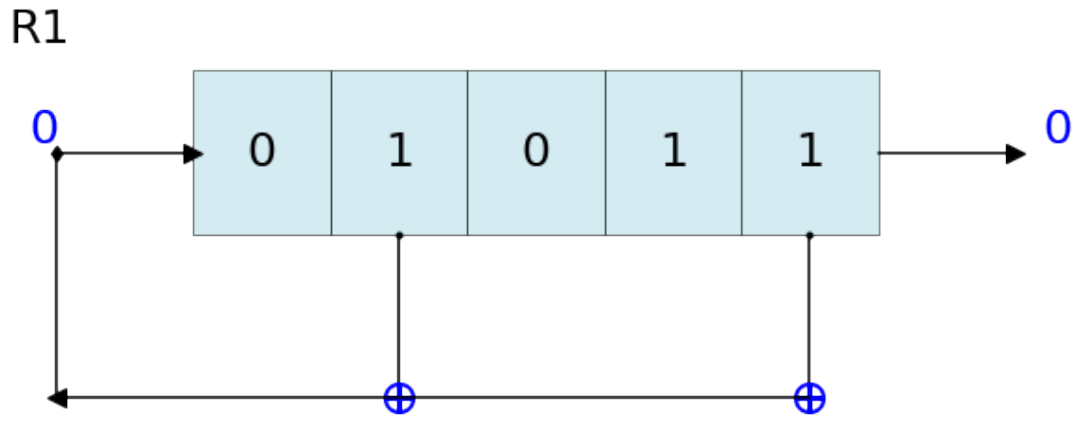
2.6 +

2.7 Poltting & Visualizations:

2.8 Example 5: Plotting LFSR with pylsr

Each LFSR can be visualize as it in current state by using *.Viz()* method

```
L = LFSR(initstate=[1,1,0,1,1],fpoly=[5,2])
L.runKCycle(15)
L.Viz(title='R1')
```



Output seq = 1101100011111100

2.9 Example 6: Dynamic visualization of LFSR - Animation*

```
%matplotlib notebook
L = LFSR(initstate=[1,0,1,0,1],fpoly=[5,4,3,2],counter_start_zero=False)
```

```
fig, ax = plt.subplots(figsize=(8,3))
for _ in range(35):
  ax.clear()
  L.Viz(ax=ax, title='R1')
  plt.ylim([-0.1,None])
  #plt.tight_layout()
  L.next()
  fig.canvas.draw()
  plt.pause(0.1)
```

2.10 +

2.11 Setting clock start:

Initial output bit An argument *counter_start_zero* can be used to initialize the output bit. * If *counter_start_zero=True* (default), the output bit is initialize by -1, to illustrate that No clock is provided yet.

In this case, *cout* (counter) starts with 0. The first output is not computed until first cylce is executed, such as by executing .next(), .runFullCycle, etc

- If *counter_start_zero=False*, the output bit is initialize by the last bit of register. In one sense, first clock cycle is executed. This is why, in this case, *cout* (counter) starts with 1.

In both cases counter_start_zero =True or False, the L.seq will be same, the only difference is the total number of output bits produced after N-cycles, i.e. when setting *counter_start_zero = False*, there will be one extra bit, since first bit was already computed. To understand this, look at following two examples. *counter_start_zero=True* can be seen as dealyed response by one bit.

2.12 Example 7.1: Visualize, 3-bit LFSR at each step, with default *counter_start_zero = True*

```
state = [1,1,1]
fpoly = [3,2]
L = LFSR(initstate=state,fpoly=fpoly)
print('count \t state \t\toutbit \t seq')
print('-'*50)
for _ in range(15):
    print(L.count,L.state,'',L.outbit,L.seq,sep='\t')
    L.next()
print('-'*50)
print('Output: ',L.seq)
```

```
count           state           outbit   seq
--------------------------------------------------
0               [1 1 1]         -1       [-1]
1               [0 1 1]         1        [1]
2               [0 0 1]         1        [1 1]
3               [1 0 0]         1        [1 1 1]
4               [0 1 0]         0        [1 1 1 0]
5               [1 0 1]         0        [1 1 1 0 0]
6               [1 1 0]         1        [1 1 1 0 0 1]
7               [1 1 1]         0        [1 1 1 0 0 1 0]
8               [0 1 1]         1        [1 1 1 0 0 1 0 1]
9               [0 0 1]         1        [1 1 1 0 0 1 0 1 1]
10              [1 0 0]         1        [1 1 1 0 0 1 0 1 1 1]
11              [0 1 0]         0        [1 1 1 0 0 1 0 1 1 1 0]
12              [1 0 1]         0        [1 1 1 0 0 1 0 1 1 1 0 0]
13              [1 1 0]         1        [1 1 1 0 0 1 0 1 1 1 0 0 1]
14              [1 1 1]         0        [1 1 1 0 0 1 0 1 1 1 0 0 1 0]
--------------------------------------------------
Output:  [1 1 1 0 0 1 0 1 1 1 0 0 1 0 1]
```

2.13 Example 7.2: Visualize, 3-bit LFSR at each step, with *counter_start_zero = False*

```
state = [1,1,1]
fpoly = [3,2]
L = LFSR(initstate=state,fpoly=fpoly,counter_start_zero=False)
print('count \t state \t\toutbit \t seq')
print('-'*50)
for _ in range(15):
    print(L.count,L.state,'',L.outbit,L.seq,sep='\t')
    L.next()
```

(continues on next page)

(continued from previous page)

```
print('-'*50)
print('Output: ',L.seq)
```

```
count          state          outbit   seq
--------------------------------------------------
1      [1 1 1]          1        [1]
2      [0 1 1]          1        [1 1]
3      [0 0 1]          1        [1 1 1]
4      [1 0 0]          0        [1 1 1 0]
5      [0 1 0]          0        [1 1 1 0 0]
6      [1 0 1]          1        [1 1 1 0 0 1]
7      [1 1 0]          0        [1 1 1 0 0 1 0]
8      [1 1 1]          1        [1 1 1 0 0 1 0 1]
9      [0 1 1]          1        [1 1 1 0 0 1 0 1 1]
10     [0 0 1]          1        [1 1 1 0 0 1 0 1 1 1]
11     [1 0 0]          0        [1 1 1 0 0 1 0 1 1 1 0]
12     [0 1 0]          0        [1 1 1 0 0 1 0 1 1 1 0 0]
13     [1 0 1]          1        [1 1 1 0 0 1 0 1 1 1 0 0 1]
14     [1 1 0]          0        [1 1 1 0 0 1 0 1 1 1 0 0 1 0]
--------------------------------------------------
Output:  [1 1 1 0 0 1 0 1 1 1 0 0 1 0 1]
```

2.14 +

2.15 LFSR Properties:: Test 3+1 properties of LFSR

Using *test_properties(verbose=1)* method, it we can test if LSFR set be state and polynomial setisfies the following properites in addition to periodicity (period T = 2^M -1) for M-bit LFSR * (1) Balance Property * (2) Runlength Property * (3) Autocorrelation Property

2.16 Example 8.1: test [5,3], for 5-bit LFSR, which we know is a primitive polynomial

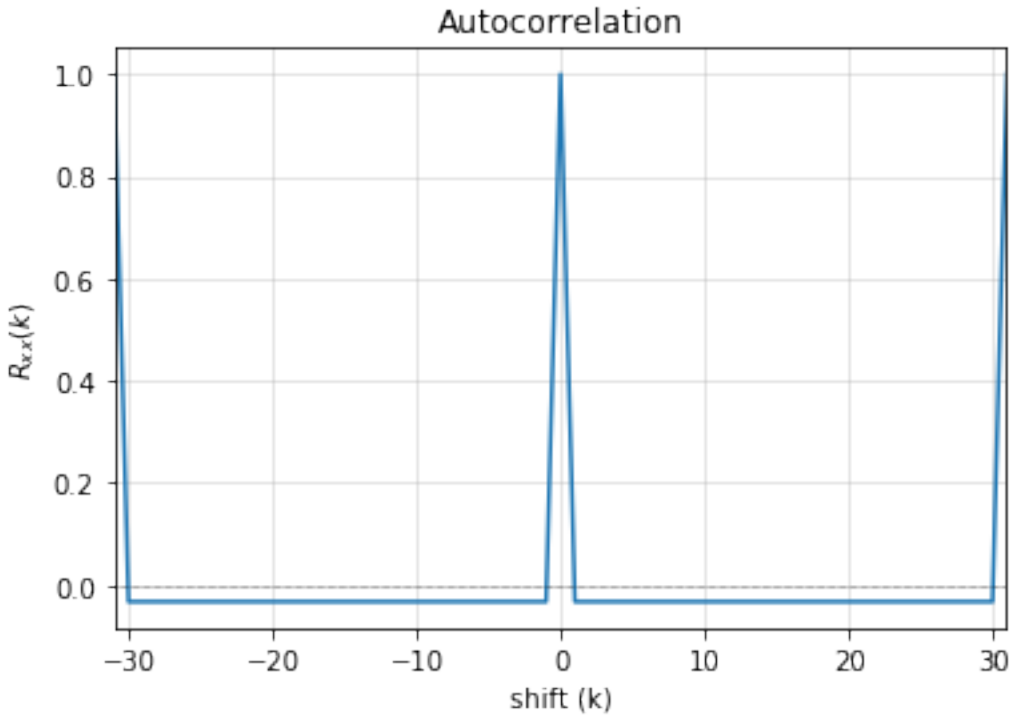
```
state = [1,1,1,1,0]
fpoly = [5,3]
L = LFSR(initstate=state,fpoly=fpoly)
result  = L.test_properties(verbose=2)
```

```
1. Periodicity
------------------
 - Expected period = 2^M-1 = 31
 - Pass?:  True

2. Balance Property
-------------------
 - Number of 1s = Number of 0s+1 (in a period): (N1s,N0s) =  (16, 15)
 - Pass?:  True

3. Runlength Property
```

(continues on next page)

(continued from previous page)

```
-------------------
 - Number of Runs in a period should be of specific order, e.g. [4,2,1,1]
 - Runs:  [8 4 2 1 1]
 - Pass?:  True

4. Autocorrelation Property
-------------------
 - Autocorrelation of a period should be noise-like, specifically, 1 at k=0, -1/m␣
↪everywhere else
 - Pass?:  True


==================
Passed all the tests
==================
```



2.17 Example 8.2: test [5,1], for 5-bit LFSR, which we know is *NOT* a primitive polynomial

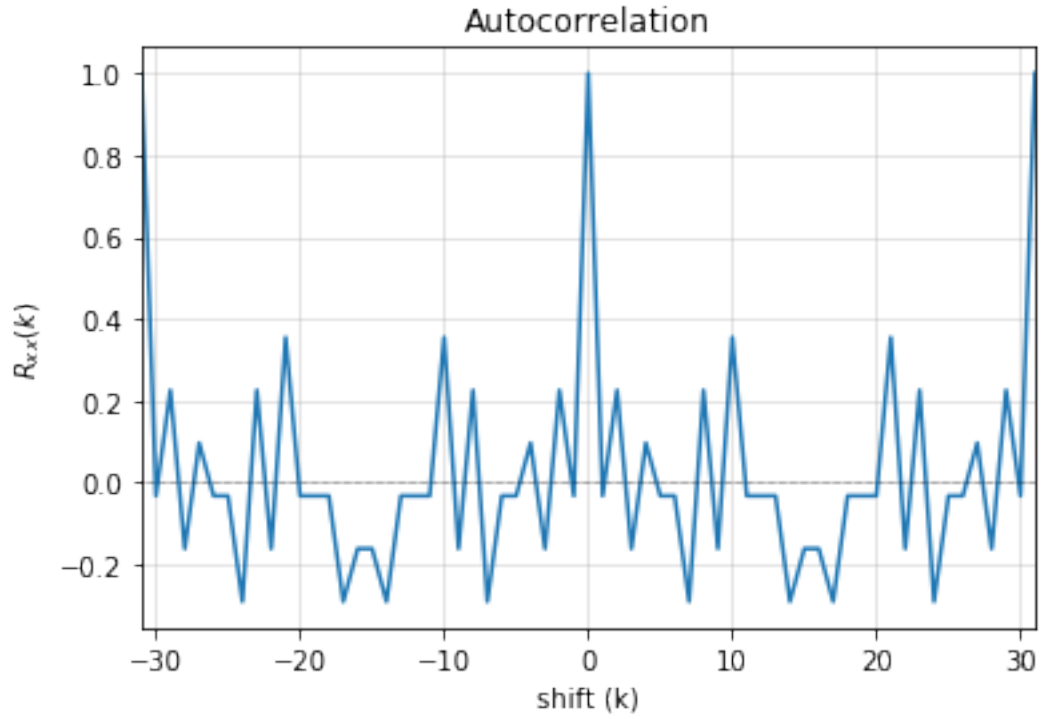
```
state = [1,1,1,1,0]
fpoly = [5,1]
L = LFSR(initstate=state,fpoly=fpoly)
result  = L.test_properties(verbose=2)
```

```
1. Periodicity
------------------
 - Expected period = 2^M-1 = 31
```

(continues on next page)

(continued from previous page)

```
 - Pass?:  False

2. Balance Property
-------------------
 - Number of 1s = Number of 0s+1 (in a period): (N1s,N0s) =  (17, 14)
 - Pass?:  False

3. Runlength Property
---------------------
 - Number of Runs in a period should be of specific order, e.g. [4,2,1,1]
 - Runs:  [10  2  1  1  2]
 - Pass?:  False

4. Autocorrelation Property
---------------------
 - Autocorrelation of a period should be noise-like, specifically, 1 at k=0, -1/m␣
↪everywhere else
 - Pass?:  False

==================
Failed one or more tests, check if feedback polynomial is primitive polynomial
==================
```



2.18 Example 8.3: test individual properties

```
state = [1,1,1,1,1]
fpoly = [5,4,3,2]
L = LFSR(initstate=state,fpoly=fpoly)
```

(continues on next page)

(continued from previous page)

```
# get one full period
p = L.getFullPeriod()

L.balance_property(p.copy())
L.runlength_property(p.copy())
L.autocorr_property(p.copy())
```

2.19 +

2.20 Feedback (Primitive) Polynomials:

A primitive polynomial is is irreducible, and not trivial to derive. A list of primitive polynomials upto 32 degree can be found at Ref, which is not an exhaustive list. Since for each primitive polynomial, an image replica (which is also primitive) can be computed easily list include half of polynomials for each degree and other half can be compputed by *get_Ifpoly()* method, see example 7.2

Ref : http://www.partow.net/programming/polynomials/index.html

2.21 Example 9.1: Get a list of feedback polynomials for a m-bit LFSR

```
L = LFSR()
# list of 5-bit feedback polynomials
fpolys = L.get_fpolyList(m=5)
[[5, 2], [5, 4, 2, 1], [5, 4, 3, 2]]

# list of all feedback polynomials as a dictionary
fpolyDict = L.get_fpolyList()
```

2.22 Example 9.2: Get a image replica of a feedback polynomial

Image replica of a primitive polynomial is a primitive polynomial, hence a valid feedback polynomial for LFSR For m-bit primitive polynomial p(x) = x^m + x^k + .. + 1, a image replica is ip(x) = x^(-m)p(x) where 0 < k < m

```
L = LFSR()
L.get_Ifpoly([5,3])
[5, 2]
```

```
L.get_Ifpoly([5,4,3,2])
[5, 3, 2, 1]
```

2.23 Example 9.3: Changing feedback polynomial in between

After generating some bits from an LFSR, a feedback polynomial can be changed keeping the current state as intial state and generate the new sequece.

```
L = LFSR(fpoly=[23,18],initstate ='ones')
seq0 = L.runKCycle(10)

# Change after 10 clocks
L.changeFpoly(newfpoly =[23,14],reset=False)
seq1 = L.runKCycle(20)

# Change after 20 clocks
L.changeFpoly(newfpoly =[23,9],reset=False)
seq2 = L.runKCycle(20)
```

2.24 +

CHAPTER 3

Generators

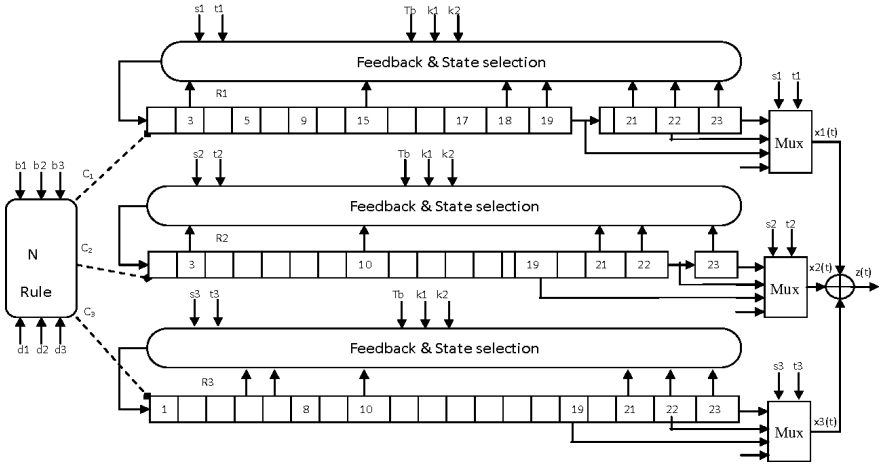
3.1 A5/1 GSM Stream cipher generator

Ref: https://en.wikipedia.org/wiki/A5/1

```
import numpy as np
import matplotlib.pyplot as plt
from pylfsr import A5_1

A5 = A5_1(key='random')
print('key: ',A5.key)
A5.R1.Viz(title='R1')
A5.R2.Viz(title='R2')
A5.R3.Viz(title='R3')

print('key: ',A5.key)
print()
print('count \t cbit\t\tclk\t R1_R2_R3\toutbit \t seq')
print('-'*80)
for _ in range(15):
    print(A5.count,A5.getCbits(),A5.clock_bit,A5.getLastbits(),A5.outbit,A5.getSeq(),
↪sep='\t')
    A5.next()
print('-'*80)
print('Output: ',A5.seq)

A5.runKCycle(1000)
A5.getSeq()
```

3.2 Enhanced A5/1

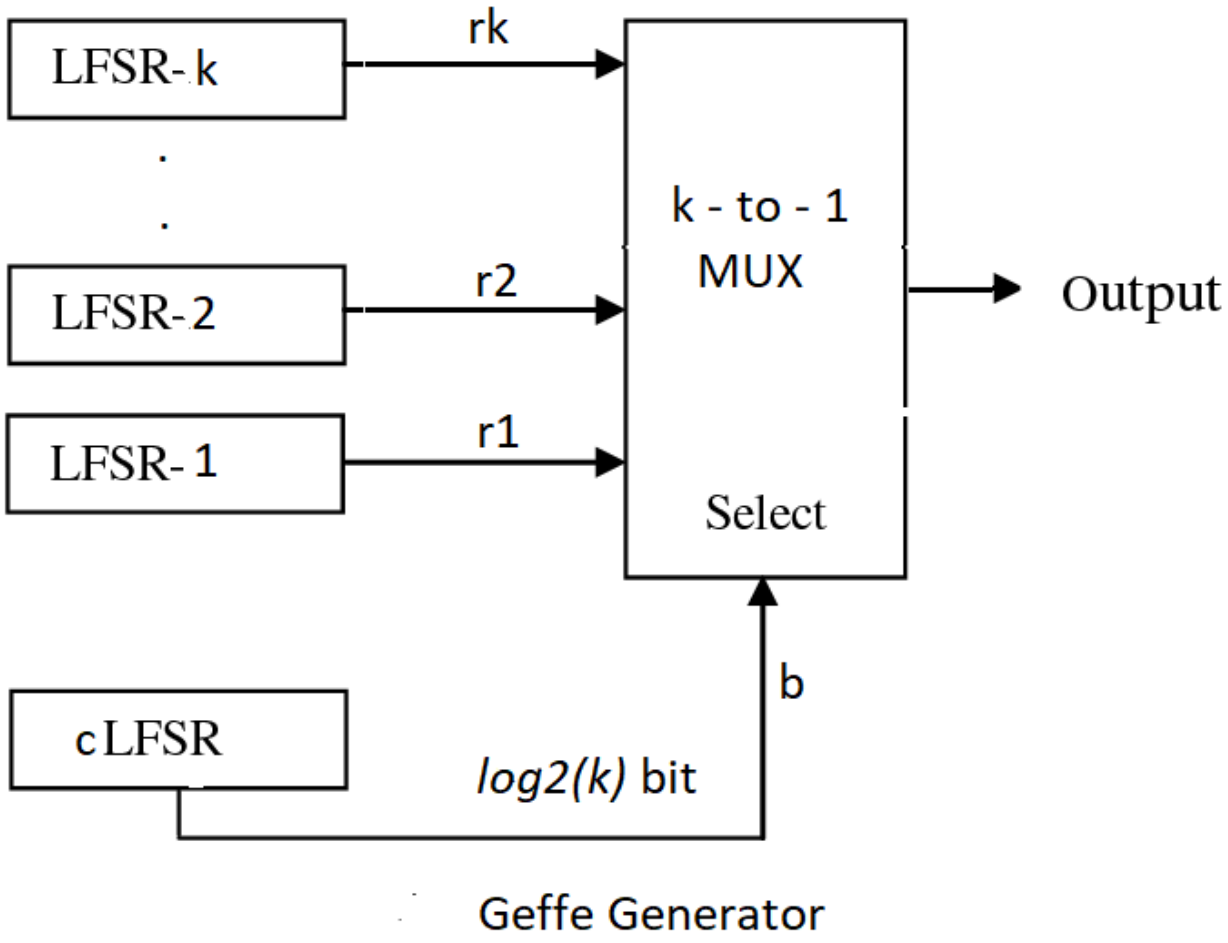
Reference Article: **Enhancement of A5/1**: https://doi.org/10.1109/ETNCC.2011.5958486




```
# Three LFSRs initialzed with 'ones' though they are intialized with encription key
R1 = LFSR(fpoly = [19,18,17,14])
R2 = LFSR(fpoly = [23,22,21,8])
R3 = LFSR(fpoly = [22,21])

# clocking bits
b1 = R1.state[8]
b2 = R3.state[10]
b3 = R3.state[10]
```

3.3 Geffe Generator

Ref: Schneier, Bruce. Applied cryptography: protocols, algorithms, and source code in C. john wiley & sons, 2007. Chaper 16



Geffe Generator

```
import numpy as np
import matplotlib.pyplot as plt
from pylfsr import Geffe, LFSR

kLFSR = [LFSR(initstate='random') for _ in range(8)]  # List of 8 5-bit LFSRs with␣
↪default feedback polynomial and random initial state
cLFSR = LFSR(initstate='random')                      # A 5-bit LFSR with for␣
↪selecting one of 8 output at a time

GG = Geffe(kLFSR_list=kLFSR, cLFSR=cLFSR)

print('key: ',GG.getState())
print()
for _ in range(50):
    print(GG.count,GG.m_count,GG.outbit_k,GG.sel_k,GG.outbit,GG.getSeq(),sep='\t')
    GG.next()

GG.runKCycle(1000)
GG.getSeq()
```

CHAPTER 4

+

CHAPTER 5

MATLAB CODE

Folder : https://github.com/Nikeshbajaj/Linear_Feedback_Shift_Register/tree/master/matlabfiles

5.1 Description

Generate randon binary sequence using LFSR for any given feedback taps (polynomial), This will also check three fundamental properties of LFSR:

1. Balance Property
2. Runlength Property
3. Autocorrelation Property

This MATLAB Code work for any length of LFSR with given taps (feedback polynomial) -Universal, There are three files LFSRv1.m an LFSRv2.m, LFSRv3.m

5.2 LFSRv1

This function will return all the states of LFSR and will check Three fundamental Property of LFSR (1) Balance Property (2) Runlength Property (3) Autocorrelation Property

Example:

```
s=[1 1 0 0 1]
t=[5 2]
[seq c] =LFSRv1(s,t)
```

5.3 LFSRv2

This function will return only generated sequence will all the states of LFSR, no verification of properties are done here. Use this function to avoid verification each time you execute the program.

Example:

```
s=[1 1 0 0 1]
t=[5 2]
[seq c] =LFSRv2(s,t)
```

5.4 LFSRv3 (faster)

seq = LFSRv3(s,t,N) this function generates N bit sequence only. This is faster then other two functions, as this does not gives each state of LFSR

Exmple:

```
s=[1 1 0 0 1]
t=[5 2]
seq =LFSRv3(s,t,50)
```

Tips

- If you want to use this function in middle of any program, use LFSRv2 or LFSRv1 with verification =0.
- If you want to make it fast for long length of LFSR,use LFSRv3.m

CHAPTER 6

Contacts

If any doubt, confusion or feedback please contact me

- *n.bajaj@qmul.ac.uk*
- *nikkeshbajaj@gmail.com*

Nikesh Bajaj: http://nikeshbajaj.in

PhD Student: **Queen Mary University of London** & **University of Genoa**

CHAPTER 7

API LFSR

class(fpoly=[5,2], initstate='ones', verbose=False)

help doc

Linear Feedback Shift Register

#Parameters ————————————

- **initstate** [binary np.array (row vector) or str ='ones' or 'random', optional (default = 'ones')) Initial state of LFSR. default ='ones'] Initial state is intialized with ones and length of register is equal to degree of feedback polynomial if state='rand', initial state is intialized with random binary sequence of length equal to degree of feedback polynomial
- **fpoly** [List, optional (default=[5,2])] Feedback polynomial, it has to be primitive polynomial of GF(2) field, for valid output of LFSR to get the list of feedback polynomials check method 'get_fpolyList' or check Refeferece: Ref: List of some primitive polynomial over GF(2)can be found at http://www.partow.net/programming/polynomials/index.html http://www.ams.org/journals/mcom/1962-16-079/S0025-5718-1962-0148256-1/S0025-5718-1962-0148256-1.pdf http://poincare.matf.bg.ac.rs/~ezivkovm/publications/primpol1.pdf
- **verbose** [boolean, optional (default=False)] if True, state of LFSR will be printed at every cycle(iteration)

#Attributes ————————————

- **count** [int] Count the cycle
- **seq** [np.array shape =(count,)] Output sequence stored in seq since first cycle. If -1, no cycle has been excecuted, count =0
- **outbit** [binary bit] Current output bit, Last bit of current state if -1, no cycle has been excecuted, count =0
- **feedbackbit** [binary bit] If -1, no cycle has been excecuted, count =0
- **M** [int] length of LFSR, M-bit LFSR,
- **expectedPeriod** [int] Expected period of sequence if feedback polynomial is primitive and irreducible (as per reference), period will be 2^M -1
- **feedpoly** [str] feedback polynomial

#Methods ————————————————

- **next()** run one cycle on LFSR with given feedback polynomial and update the count, state, feedback bit, output bit and seq

  return: binary bit output bit : binary

- **runKCycle(k)** run k cycles and update all the Parameters

  return tempseq : shape =(k,)

  output binary sequence of k cycles

- **runFullCycle()** run full cycle (= 2^M-1)

  return seq : binary output sequence since start: shape = (count,)

- **set(fpoly,state='ones')** set feedback polynomial and state

  fpoly : list feedback polynomial like [5,4,3,2]

  state : np.array, like np.array([1,0,0,1,1]), default ='ones' Initial state is intialized with ones and length of register is equal to degree of feedback polynomial if state='rand', initial state is intialized with random binary sequence of length equal to degree of feedback polynomial

- **reset()** Reseting LFSR to its initial state and count to 0

- **changeFpoly(newfpoly, reset=False)** Changing Feedback polynomial newfpoly : list like, [5,4,2,1], changing the feedback polynomial

  reset : boolean default=False if True, reset all the Parameters : count=0, seq=-1.. if False, leave the LFSR as it is only change the feedback polynomial as used in *'Enhancement of A5/1: Using variable feedback polynomials of LFSR'* ref: 10.1109/ETNCC.2011.5958486

- **check()** check if -degree of feedback polynomial <= length of LFSR >=1 -given intistate of LFSR is correct

- **info()** display the information about LFSR with current state of variables

- **get_fpolyList(m=None)** Get the list of primitive polynomials as feedback polynomials for m-bit LFSR if m is None, list of feedback polynomials for $1 < m < 32$ is return as a dictionary

- **get_Ifpoly(*fpoly*)** Get the image of primitive polynomial *fpoly*, which is also a valid primitive polynomial