レポートタイトル

学科　学籍番号　氏名

2023 年?? 月?? 日

1　はじめに

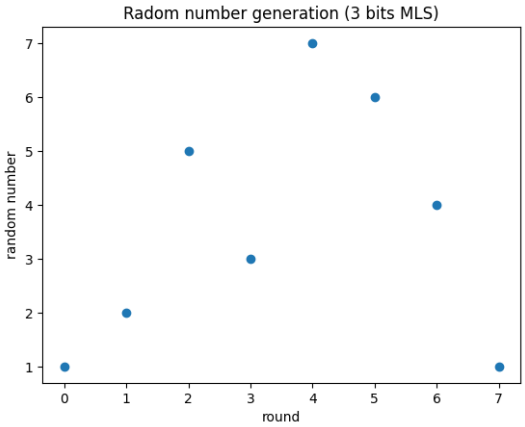


図1　図の貼り方

2　実験 1-5

2.1　目的

これからランダムウォーク実験を行うにあたって、ランダムな数を生成することが必要であるが、パソコンはランダムな数を生成できない。そこでコンピューター上では数式を用いて擬似乱数列ー乱数のように思えるが、初期状態が決定すれば未来の数列が決定してしまうので真にランダムではない数列ーを生成するのだが、このとき C 言語における srand(time(NULL)) や、python における rand() のようないわゆる組み込み関数を用いずに擬似乱数列を生成するのが今回の実験の目的である。

2.2　理論

擬似乱数列の生成を行う。このとき LFSR 法を用いた。LFSR とは次の数を直前の数に基づきシフト演算を使って漸化式的に決定した数列 (次の数が直前の数の線形写像になっているシフトレジスタ) のことで、このとき直前の数から次の数を求める漸化式をうまく設定することで、全ビットが 0 という状態以外のすべての

取る整数列を作ることができることが分かっており、これを最長 LFSR と呼ぶ。今回はこの最長 LSFR を作ることで擬似乱数列を生成した。

ここで、最長 LSFR を生成できる漸化式を次に示す。このとき、下に示す漸化式は 2 進数である

$$a_{n+1} = b_{n+1} + c_{n+1}\&1$$

但し

$$b_{n+1} = (a_n << 1)\&(10^{bit} - 1)$$

(しつこいようだが、10 はいわゆる 2 進数の 10 であり、つまり 10 進数における 2 である)

$$c_{n+1} = \sum_i (a_n >> k_i)$$

このとき、c_{n+1} を以下のように設定することで、最長 LFSR を計算することができる:

n	XNOR from	n	XNOR from	n	XNOR from	n	XNOR from
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	10,7	52	52,49	94	94,73	136	136,135,11,10
11	11,9	53	53,52,38,37	95	95,84	137	137,116
12	12,6,4,1	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,94	143	143,142,123,122
18	18,11	60	60,59	102	102,101,36,35	144	144,143,75,74
19	19,6,2,1	61	61,60,46,45	103	103,94	145	145,93
20	20,17	62	62,61,6,5	104	104,103,94,93	146	146,145,87,86
21	21,19	63	63,62	105	105,89	147	147,146,110,109
22	22,21	64	64,63,61,60	106	106,91	148	148,121
23	23,18	65	65,47	107	107,105,44,42	149	149,148,40,39
24	24,23,22,17	66	66,65,57,56	108	108,77	150	150,97
25	25,22	67	67,66,58,57	109	109,108,103,102	151	151,148
26	26,6,2,1	68	68,59	110	110,109,98,97	152	152,151,87,86
27	27,5,2,1	69	69,67,42,40	111	111,101	153	153,152
28	28,25	70	70,69,55,54	112	112,110,69,67	154	154,152,27,25
29	29,27	71	71,65	113	113,104	155	155,154,124,123
30	30,6,4,1	72	72,66,25,19	114	114,113,33,32	156	156,155,41,40
31	31,28	73	73,48	115	115,114,101,100	157	157,156,131,130
32	32,22,2,1	74	74,73,59,58	116	116,115,46,45	158	158,157,132,131
33	33,20	75	75,74,65,64	117	117,115,99,97	159	159,128
34	34,27,2,1	76	76,75,41,40	118	118,85	160	160,159,142,141
35	35,33	77	77,76,47,46	119	119,111	161	161,143
36	36,25	78	78,77,59,58	120	120,113,9,2	162	162,161,75,74
37	37,5,4,3,2,1	79	79,70	121	121,103	163	163,162,104,103
38	38,6,5,1	80	80,79,43,42	122	122,121,63,62	164	164,163,151,150
39	39,35	81	81,77	123	123,121	165	165,164,135,134
40	40,38,21,19	82	82,79,47,44	124	124,87	166	166,165,128,127
41	41,38	83	83,82,38,37	125	125,124,18,17	167	167,161
42	42,41,20,19	84	84,71	126	126,125,90,89	168	168,166,153,151
43	43,42,38,37	85	85,84,58,57	127	127,126		
44	44,43,18,17	86	86,85,74,73	128	128,126,101,99		

但し、漸化式中の bit、k_i は、表にある表現を用いて

$$bit = n$$

$$k_i = XNORfrom$$

と表せる

2.3 実験方法

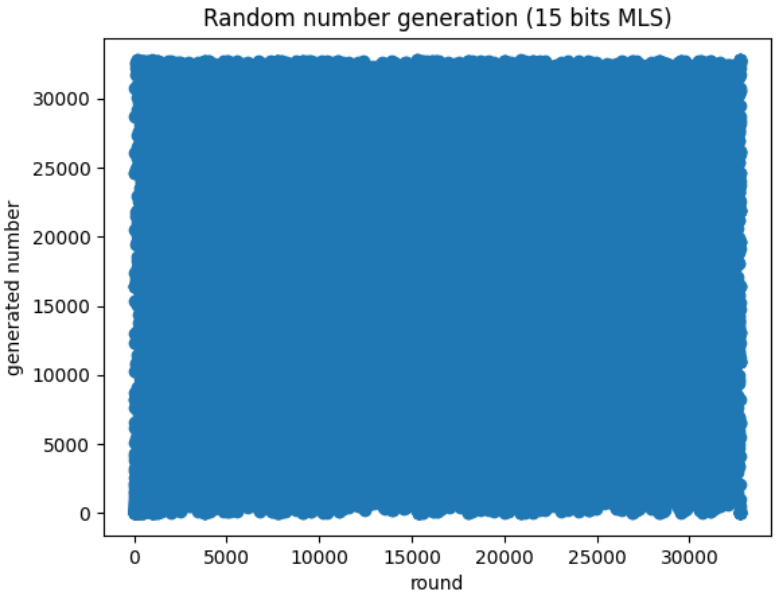
実験に用いた python プログラムを次に示す

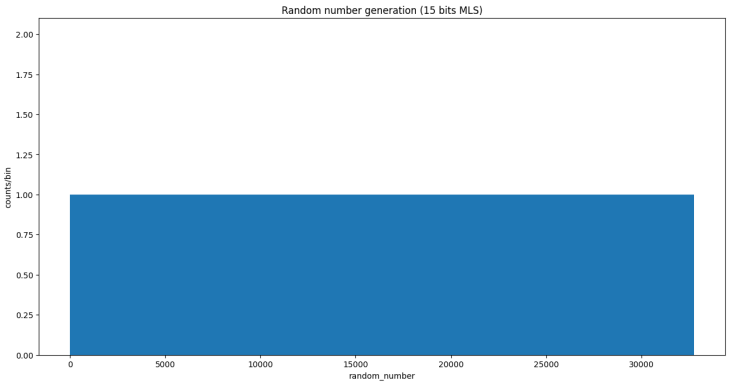
Listing 1　キャプション 2

```
import matplotlib.pyplot as plt

feedbacks = [[],[],[],[3,2],[4,3],
             [5,3],[6,5],[7,6],[8,6,5,4],[9,5],
             [10,7],[11,9],[12,11,10,4],[13,12,11,8],[14,13,12,2],
             [15,14],[16,14,13,11],[17,14],[18,11],[19,18,17,14],
             [20,17],[21,19],[22,21],[23,18],[24,23,22,17]
            ]

def calculate_lfsr(initNumber, bits):
    num = initNumber # initial number
    maxrounds = 2**bits
    formater = "0" + str(bits) + "b"

    rands = []

    for i in range(maxrounds):
        # print(num, "(" , format(num, formater), ")")
        rands.append(num)

        a = (num << 1) & (maxrounds-1)

        #seems both acceptable, but b=1 would become more complex and difficult to estimate
        # b = 1
        b = 0

        for j in range(len(feedbacks[bits])):
            target = feedbacks[bits][j] - 1
            b = ((b & 1) ^ (num >> target) & 1) & 1

        num = a+(b&1)

    return rands

def plot_results_a(rands, bits):
    nums = range(2**bits)
    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.scatter(nums, rands)
    plt.title(title_str)
    plt.xlabel("round")
    plt.ylabel("generated number")
    plt.show()

def plot_results_b(rands, bits):
    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.figure()
    plt.hist(rands, bins=2**bits, range=[0,2**bits])
    plt.title(title_str)
    plt.xlabel("random_number")
    plt.ylabel("counts/bin")
    plt.show()

def main():
    init = 3 # change variants here
    bits = 9 # change variants here
    # this program can calculate from 3 bits to 24 bits, but calculating 24 bits never finishes ! (due to
        the amount of calculation)
    local_rands = calculate_lfsr(init, bits)
    plot_results_a(local_rands, bits)
    plot_results_b(local_rands, bits)

if __name__ == "__main__":
    main()
```

2.4 実験結果

このプログラムを 15bit で実行した結果を次に示す



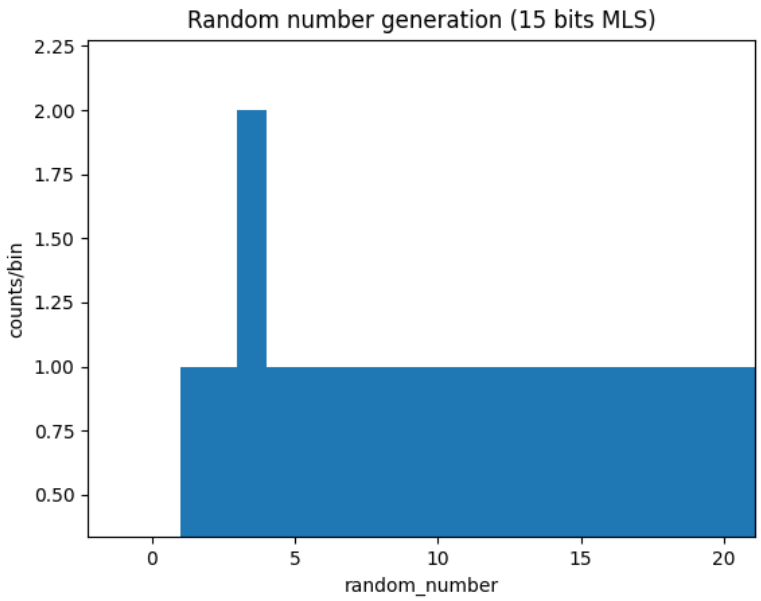


ここに示したヒストグラムのように、初期値である 3 (ソースコード 1.py の 54 行目で指定した) を除いた全ての数が 1 から 2^{15} まで 1 回ずつのみ現れ、3 のみ 2 回現れている。

3 実験 2

3.1 目的

$Z^d(\in j = (j_1, j_2..., j_d))$ と表現される d 次元格子において、毎回 2d 個の隣接点から 1 点を等確率で選んで進んでいく運動を d 次元単純ランダムウォークと呼ぶ。これはブラウン運動などと共に、統計力学・量子力



学・数理ファイナンスのようなランダムな運動を数学的に記述するモデルのなかで、もっとも基本的なものの一つとして知られている。今回はそのなかでもっとも基礎的な 1 次元単純ランダムウォークを実行し、粒子の動きを調べることで、確率過程モデルの基礎を確認する。

3.2 理論

t 回目の試行における粒子の座標を x_t とおくと、

$$P(x_{t+1} - x_t = 1) = 1/2$$

$$P(x_{t+1} - x_t = -1) = 1/2$$

を満たすとする (単純ランダムウォーク)

3.3 実験方法

3.3.1 実験 2-4

実験に用いた Python プログラムを次に示す

Listing 2　キャプション 2

```
import matplotlib.pyplot as plt
import time


feedbacks = [[],[],[],[3,2],[4,3],
             [5,3],[6,5],[7,6],[8,6,5,4],[9,5],
             [10,7],[11,9],[12,11,10,4],[13,12,11,8],[14,13,12,2],
             [15,14],[16,14,13,11],[17,14],[18,11],[19,18,17,14],
             [20,17],[21,19],[22,21],[23,18],[24,23,22,17]
            ]


def calculate_lfsr(initNumber, bits):
```

```
    num = initNumber # initial number
    maxrounds = 2**bits
    formater = "0" + str(bits) + "b"

    rands = []

    for i in range(maxrounds):
        # print(num, "(" , format(num, formater), ")")
        rands.append(num)

        a = (num << 1) & (maxrounds-1)

        #seems both acceptable, but b=1 would become more complex and difficult to estimate
        # b = 1
        b = 0

        for j in range(len(feedbacks[bits])):
            target = feedbacks[bits][j] - 1
            b = ((b & 1) ^ (num >> target) & 1) & 1

        num = a+(b&1)

    return rands


def plot_results_a(rands, bits):
    nums = range(2**bits)
    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.scatter(nums, rands)
    plt.title(title_str)
    plt.xlabel("round")
    plt.ylabel("generated number")
    plt.show()


def plot_results_b(rands, bits):
    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.figure()
    plt.hist(rands, bins=2**bits, range=[0,2**bits])
    plt.title(title_str)
    plt.xlabel("random_number")
    plt.ylabel("counts/bin")
    plt.show()


def random_walk(rands, bits, checkpoints):
    x=0
    pos_at_checkpoints = [0]

    for t in range(1, max(checkpoints)+1):
        if rands[(t-1)%(2**bits)] & 1 == 0: # this is the equivalent of "rands[] % 2 == 0"
            x = x+1
        else:
            x = x-1

        if t in checkpoints:
            pos_at_checkpoints.append(x)

    # print(checkpoints)
    # print(pos_at_checkpoints)

    return pos_at_checkpoints


def main():
    # this program can calculate from 3 bits to 24 bits, but calculating 24 bits never finishes ! (due to
        the amount of calculation)

    # change variants under here
    bits = 12
    cycle = 2**bits # but DO NOT CHANGE HERE!
    # trials = 5
    trials = 1
    checkpoints = range(cycle)
    # checkpoints = [0,200,350,int(0.3*cycle),int(0.5*cycle),int(0.7*cycle),cycle]
    # checkpoints.sort()
    markers = ['+','x','D','d']
    # change variants above here

```

```
    pos_at_checkpoints = []

    for i in range(trials):
        init = int(time.time()*100)**2 % (2**bits)
        rands = calculate_lfsr(init, bits)
        pos_at_checkpoints = random_walk(rands, bits, checkpoints)
        plt.plot(checkpoints, pos_at_checkpoints, label = "inital number = {:d}".format(init))
        # plt.scatter(checkpoints, pos_at_checkpoints, alpha=0.6, marker = markers[i%len(markers)], label =
              "inital number = {:d}".format(init))

    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.title(title_str)
    plt.xlabel("time")
    plt.ylabel("x position")
    plt.legend()
    plt.show()

    # plot_results_a(local_rands, bits)
    # plot_results_b(local_rands, bits)


if __name__ == "__main__":
    main()
```

3.3.2 実験 2-5

実験に用いた Python プログラムを次に示す

Listing 3　キャプション 2

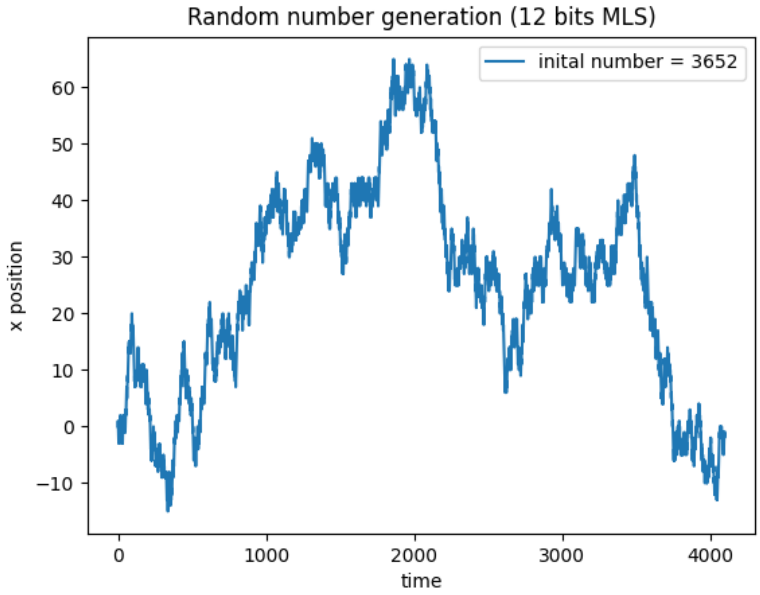
```
import matplotlib.pyplot as plt
import time


feedbacks = [[],[],[],[3,2],[4,3],
             [5,3],[6,5],[7,6],[8,6,5,4],[9,5],
             [10,7],[11,9],[12,11,10,4],[13,12,11,8],[14,13,12,2],
             [15,14],[16,14,13,11],[17,14],[18,11],[19,18,17,14],
             [20,17],[21,19],[22,21],[23,18],[24,23,22,17]
            ]


def calculate_lfsr(initNumber, bits):
    num = initNumber # initial number
    maxrounds = 2**bits
    formater = "0" + str(bits) + "b"

    rands = []

    for i in range(maxrounds):
        # print(num, "(" , format(num, formater), ")")
        rands.append(num)

        a = (num << 1) & (maxrounds-1)

        #seems both acceptable, but b=1 would become more complex and difficult to estimate
        # b = 1
        b = 0

        for j in range(len(feedbacks[bits])):
            target = feedbacks[bits][j] - 1
            b = ((b & 1) ^ (num >> target) & 1) & 1

        num = a+(b&1)

    return rands


def plot_results_a(rands, bits):
    nums = range(2**bits)
    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.scatter(nums, rands)
    plt.title(title_str)
    plt.xlabel("round")
    plt.ylabel("generated number")
    plt.show()
```

```
47
48
49  def plot_results_b(rands, bits):
50      title_str = "Random number generation ({:d} bits MLS)".format(bits)
51      plt.figure()
52      plt.hist(rands, bins=2**bits, range=[0,2**bits])
53      plt.title(title_str)
54      plt.xlabel("random_number")
55      plt.ylabel("counts/bin")
56      plt.show()
57
58
59  def random_walk(rands, bits, checkpoints):
60      x=0
61      pos_at_checkpoints = [0]
62
63      for t in range(1, max(checkpoints)+1):
64          if rands[(t-1)%(2**bits)] & 1 == 0: # this is the equivalent of "rands[] % 2 == 0"
65              x = x+1
66          else:
67              x = x-1
68
69          if t in checkpoints:
70              pos_at_checkpoints.append(x)
71
72      # print(checkpoints)
73      # print(pos_at_checkpoints)
74
75      return pos_at_checkpoints
76
77
78  def main():
79      # this program can calculate from 3 bits to 24 bits, but calculating 24 bits never finishes ! (due to
            the amount of calculation)
80
81      # change variants under here
82      bits = 12
83      cycle = 2**bits # but DO NOT CHANGE HERE!
84      trials = 5
85      # trials = 1
86      # checkpoints = range(cycle)
87      checkpoints = [0,200,350,int(0.3*cycle),int(0.5*cycle),int(0.7*cycle),cycle]
88      checkpoints.sort()
89      markers = ['+','x','D','d']
90      # change variants above here
91
92      pos_at_checkpoints = []
93
94      for i in range(trials):
95          init = int(time.time()*100)**2 % (2**bits)
96          rands = calculate_lfsr(init, bits)
97          pos_at_checkpoints = random_walk(rands, bits, checkpoints)
98          # plt.plot(checkpoints, pos_at_checkpoints, label = "inital number = {:d}".format(init))
99          plt.scatter(checkpoints, pos_at_checkpoints, alpha=0.6, marker = markers[i%len(markers)], label = "
                inital number = {:d}".format(init))
100     title_str = "Random number generation ({:d} bits MLS)".format(bits)
101     plt.title(title_str)
102     plt.xlabel("time")
103     plt.ylabel("x position")
104     plt.legend()
105     plt.show()
106
107     # plot_results_a(local_rands, bits)
108     # plot_results_b(local_rands, bits)
109
110
111 if __name__ == "__main__":
112     main()
```

3.4 実験結果

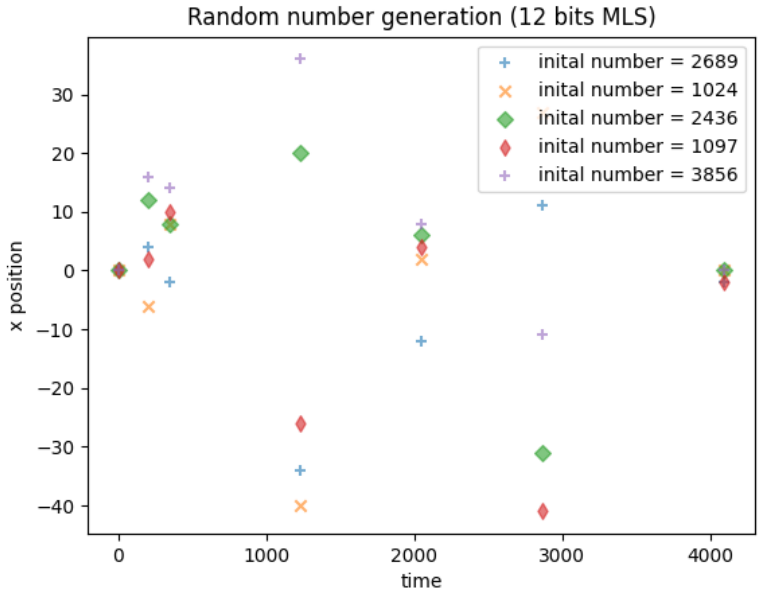
3.4.1 実験 2-4

実験の結果を次に示す



3.4.2 実験 2-5

実験の結果を次に示す



4　実験 3

4.1　目的

4.2　理論

4.3　実験方法

4.3.1　実験 3-1

実験に用いた Python プログラムを次に示す

Listing 4　キャプション 2

```
import matplotlib.pyplot as plt
import time
import numpy as np


feedbacks = [[],[],[],[3,2],[4,3],
             [5,3],[6,5],[7,6],[8,6,5,4],[9,5],
             [10,7],[11,9],[12,11,10,4],[13,12,11,8],[14,13,12,2],
             [15,14],[16,14,13,11],[17,14],[18,11],[19,18,17,14],
             [20,17],[21,19],[22,21],[23,18],[24,23,22,17]
            ]


def calculate_lfsr(initNumber, bits):
    num = initNumber # initial number
    maxrounds = 2**bits
    formater = "0" + str(bits) + "b"

    rands = []

    for i in range(maxrounds):
        # print(num, "(" , format(num, formater), ")")
        rands.append(num)

        a = (num << 1) & (maxrounds-1)

        #seems both acceptable, but b=1 would become more complex and difficult to estimate
        b = 1
        # b = 0

        for j in range(len(feedbacks[bits])):
            target = feedbacks[bits][j] - 1
            b = ((b & 1) ^ (num >> target) & 1) & 1

        num = a+(b&1)

    return rands


def plot_results_a(rands, bits):
    nums = range(2**bits)
    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.scatter(nums, rands)
    plt.title(title_str)
    plt.xlabel("round")
    plt.ylabel("generated number")
    plt.show()


def plot_results_b(rands, bits):
    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.figure()
    plt.hist(rands, bins=2**bits, range=[0,2**bits])
    plt.title(title_str)
    plt.xlabel("random_number")
    plt.ylabel("counts/bin")
    plt.show()

```

```
59
60  def random_walk(rands, bits, checkpoints):
61      x=0
62      pos_at_checkpoints = [0]
63
64      for t in range(1, max(checkpoints)+1):
65          if rands[(t-1)%(2**bits)] & 1 == 0: # this is the equivalent of "rands[] % 2 == 0"
66              x = x+1
67          else:
68              x = x-1
69
70          if t in checkpoints:
71              pos_at_checkpoints.append(x)
72
73      # print(checkpoints)
74
75      return pos_at_checkpoints
76
77
78  def main():
79      # this program can calculate from 3 bits to 24 bits, but calculating 24 bits never finishes ! (due to
            the amount of calculation)
80
81      # change variants under here
82      bits = 12
83      cycle = 2**bits # but DO NOT CHANGE HERE!
84      # trials = int(cycle*0.7)
85      trials = 300
86      # checkpoints = range(cycle)
87      checkpoints = [0,20,35,int(0.3*cycle),int(0.5*cycle),int(0.7*cycle),cycle]
88      checkpoints.sort()
89      checkpoint = 2
90      markers = ['+','x','D','d']
91      # change variants above here
92      # initialize the variants
93      all_pos_at_checkpoints = []
94      inits = []
95      # initializing ends here
96
97
98
99      for i in range(trials):
100         init = int(time.time()*10000)**2 % (2**bits)
101         rands = calculate_lfsr(init, bits)
102         pos_at_checkpoints = random_walk(rands, bits, checkpoints)
103         all_pos_at_checkpoints = np.append(all_pos_at_checkpoints, pos_at_checkpoints, axis=0)
104
105     all_pos_at_checkpoints = np.reshape(all_pos_at_checkpoints, [trials, len(checkpoints)]).astype(np.int64
            )
106
107     calculating_set = all_pos_at_checkpoints[:, checkpoint]
108     plt.scatter(list(range(trials)), calculating_set)
109     plt.xlabel("trial id")
110     plt.ylabel("x position (@ t = {:d})".format(checkpoints[checkpoint]))
111     plt.title("x position vs. trials as of {:d} bits".format(bits))
112     plt.show()
113     # plot_results_a(local_rands, bits)
114     # plot_results_b(local_rands, bits)
115
116
117 if __name__ == "__main__":
118     main()
```

4.3.2 実験 3-2,3-3

実験に用いた Python プログラムを次に示す

Listing 5　キャプション 2

```
1  import matplotlib.pyplot as plt
2  import time
3  import numpy as np
4
5
6  feedbacks = [[],[],[],[3,2],[4,3],
7               [5,3],[6,5],[7,6],[8,6,5,4],[9,5],
```

```
             [10,7],[11,9],[12,11,10,4],[13,12,11,8],[14,13,12,2],
             [15,14],[16,14,13,11],[17,14],[18,11],[19,18,17,14],
             [20,17],[21,19],[22,21],[23,18],[24,23,22,17]
            ]


def calculate_lfsr(initNumber, bits):
    num = initNumber # initial number
    maxrounds = 2**bits
    formater = "0" + str(bits) + "b"

    rands = []

    for i in range(maxrounds):
        # print(num, "(" , format(num, formater), ")")
        rands.append(num)

        a = (num << 1) & (maxrounds-1)

        #seems both acceptable, but b=1 would become more complex and difficult to estimate
        b = 1
        # b = 0

        for j in range(len(feedbacks[bits])):
            target = feedbacks[bits][j] - 1
            b = ((b & 1) ^ (num >> target) & 1) & 1

        num = a+(b&1)

    return rands


def plot_results_a(rands, bits):
    nums = range(2**bits)
    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.scatter(nums, rands)
    plt.title(title_str)
    plt.xlabel("round")
    plt.ylabel("generated number")
    plt.show()


def plot_results_b(rands, bits):
    title_str = "Random number generation ({:d} bits MLS)".format(bits)
    plt.figure()
    plt.hist(rands, bins=2**bits, range=[0,2**bits])
    plt.title(title_str)
    plt.xlabel("random_number")
    plt.ylabel("counts/bin")
    plt.show()


def random_walk(rands, bits, checkpoints):
    x=0
    pos_at_checkpoints = [0]

    for t in range(1, max(checkpoints)+1):
        if rands[(t-1)%(2**bits)] & 1 == 0: # this is the equivalent of "rands[] % 2 == 0"
            x = x+1
        else:
            x = x-1

        if t in checkpoints:
            pos_at_checkpoints.append(x)

    # print(checkpoints)

    return pos_at_checkpoints


def main():
    # this program can calculate from 3 bits to 24 bits, but calculating 24 bits never finishes ! (due to
         the amount of calculation)

    # change variants under here
    bits = 12
    cycle = 2**bits # but DO NOT CHANGE HERE!
    trials = int(cycle*0.7)
    # trials = 1
```

```
    # checkpoints = range(cycle)
    checkpoints = [0,int(0.0375*cycle),int(0.075*cycle),int(0.15*cycle),int(0.3*cycle),int(0.5*cycle),int
        (0.7*cycle),int(0.85*cycle),int(0.925*cycle),int(0.9625*cycle),cycle]
    checkpoints.sort()
    markers = ['+','x','D','d']
    # change variants above here

    all_pos_at_checkpoints = []
    means = []
    stds = []

    for i in range(trials):
        init = int(time.time()*1000000) % (2**bits)
        rands = calculate_lfsr(init, bits)
        pos_at_checkpoints = random_walk(rands, bits, checkpoints)
        all_pos_at_checkpoints = np.append(all_pos_at_checkpoints, pos_at_checkpoints, axis=0)

    all_pos_at_checkpoints = np.reshape(all_pos_at_checkpoints, [trials, len(checkpoints)]).astype(np.int64
        )

    for i in range(len(checkpoints)):
        calculating_set = all_pos_at_checkpoints[:, i]
        plt.hist(calculating_set, histtype = "step", label = "steps = {:d}".format(checkpoints[i]))

        mean = np.mean(calculating_set)
        means = np.append(means, mean)
        std = np.std(calculating_set)
        stds = np.append(stds, std)

        print(means, stds)

    plt.title("x position counts vs. x position")
    plt.xlabel("x positions at certain times")
    plt.ylabel("counts/bin")
    plt.yscale("log")
    plt.legend()
    plt.show()

    plt.clf()
    plt.close()

    plt.title("times vs. means at that times")
    plt.xlabel("times")
    plt.ylabel("means of x positions")
    plt.scatter(checkpoints, means)
    plt.show()

    plt.clf()
    plt.close()

    plt.title("times vs. standard deviations at that times")
    plt.xlabel("times")
    plt.ylabel("standard deviations of x positions")
    plt.scatter(checkpoints, stds)
    plt.show()


if __name__ == "__main__":
    main()
```

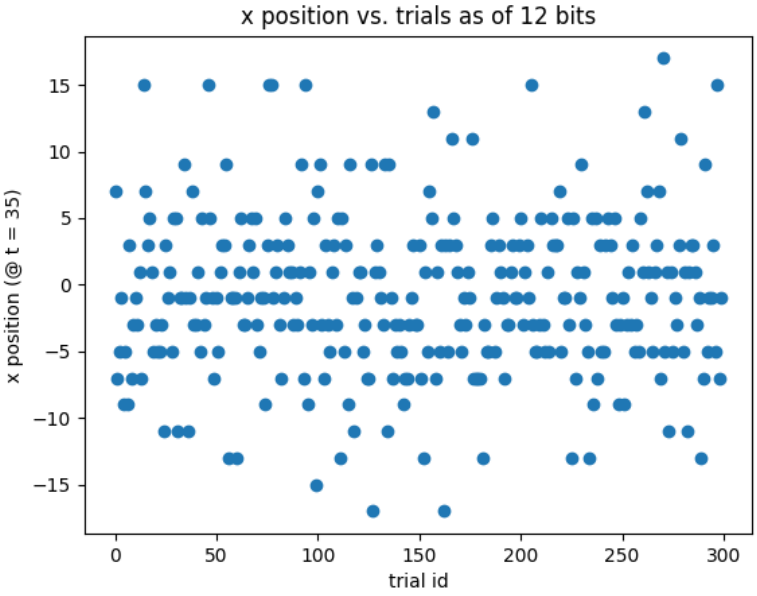
4.4 実験結果

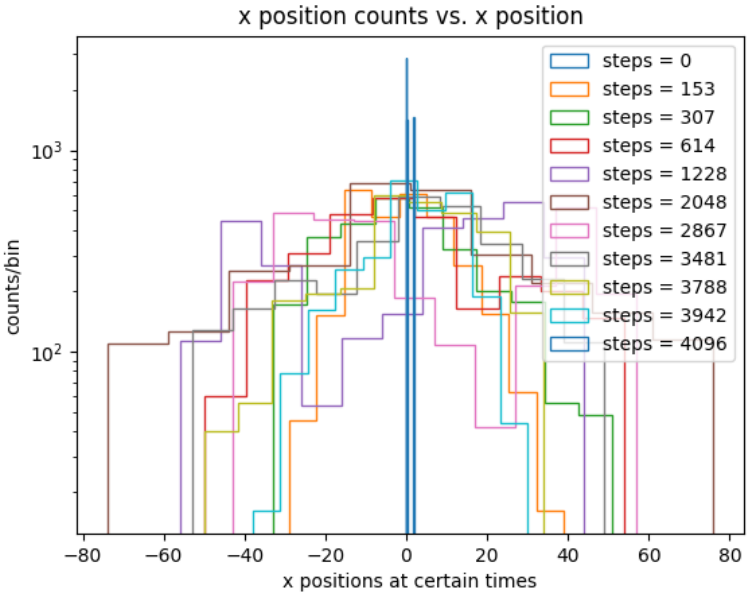
4.4.1 実験 3-1

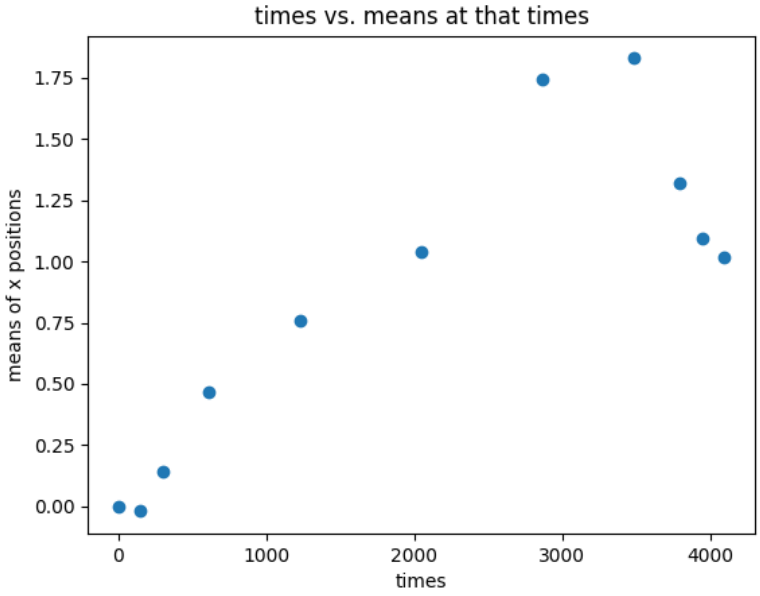
実験の結果を次に示す

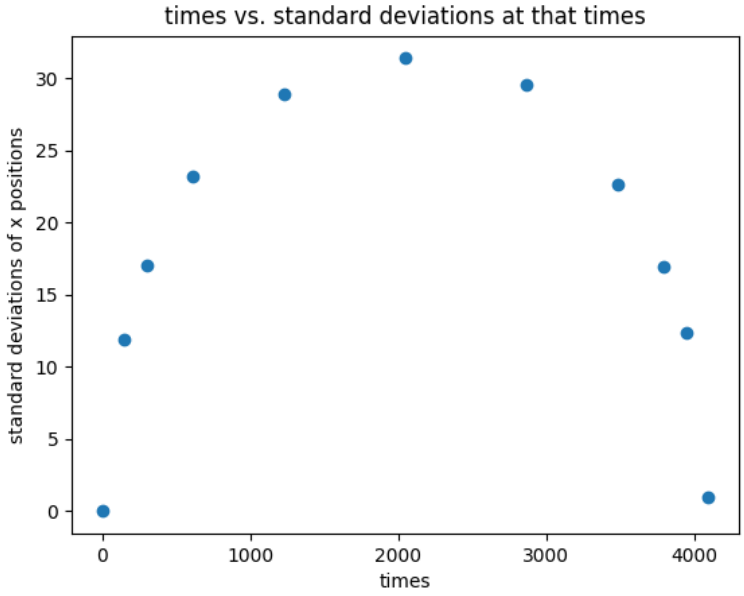
4.4.2 実験 3-2,3-3

実験の結果を次に示す


x position vs. trials as of 12 bits
x position (@ t = 35)
trial id
15
10
5
0
−5
−10
−15
0
50
100
150
200
250
300


x position counts vs. x position
counts/bin
x positions at certain times
steps = 0
steps = 153
steps = 307
steps = 614
steps = 1228
steps = 2048
steps = 2867
steps = 3481
steps = 3788
steps = 3942
steps = 4096
10^3
10^2
−80
−60
−40
−20
0
20
40
60
80


times vs. means at that times
means of x positions
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1000
2000
3000
4000
times


times vs. standard deviations at that times
standard deviations of x positions
30
25
20
15
10
5
0
0
1000
2000
3000
4000
times