Computer Networks

CSE-433

Practical Assignment - 4 (PA4)

Name - Chinmay Somani

Roll No.- 18074004

Branch - Computer Science and Engineering

Year - 4 (IDD)

Github Link for this Assignment-
https://github.com/ChinmaySomani/Assignment-4

The 2 PRNGs we are using are -

1. Blum Blum Shub -

PRNG which was devised by Lenore Blum, Manuel Blum and Michael Shub in 1986,
It generates number x[n + 1] by the equation -

X[n + 1] = (x[n] * x[n]) % M, for n >= 1

Where, M = p * q, where p, q are safe prime numbers and gcd((p - 3) / 2, (q - 3) / 2) is as minimum as possible

X[0] is selected such that it is coprime with p and q and is not 0 or 1.

In our case -
P = 809
Q = 911
X[0] = 18
N = 1000
Python Code stub for Blum Blum Shub implementation is -

```
# Blum Blum Shub implementation
def BlumBlumShub(num_generations, p, q, seed):
    ret = []
    ret.append(seed)
    M = p * q
    for i in range(num_generations - 1):
        ret.append((ret[-1] * ret[-1]) % M)
    return ret
```

2. Lagged Fibonacci Generator -

It is an extension to Fibonacci sequence(x[n] = x[n - 1] + x[n - 2]) .
The random number generated by this generator follow the equation -

X[n] = (x[n - i] + x[n - j]) % Mod, where i and j are already defined and x is pre-populated with j elements, assuming j > i.

Parameters used -
I = 3
J = 7
Mod = 1145
We have initialised x with 12 random values for a better random generation of numbers using the above equation.
For every random number that is created, all the elements of the list (lfg) are shifted one position to the left and the new number generated is stored at the end of the list.

Python Code stub for Blum Blum Shub implementation is -

```
# Lagged Fibonacci Generator implementation
def gen_LFG(i, j, n, lfg):
    lfg_li = []
    lfg_mod = 1145
    for i1 in range(n):
        lfg_li.append((lfg[i - 1] + lfg[j - 1]) % lfg_mod)
        lfg = lfg[1:]
        lfg.append(lfg_li[-1])
    return lfg_li
```

The two Randomness tests used are -

1. **Runs test -**

Null and alternative hypothesis -

- H_0 (null): Data is produced randomly
- H_1 (alternative): Data is not produced randomly

Implemented using runstest_1samp() function available in statsmodels library.

- Input - list of random numbers produced
- Output - z-test statistic and p-value.

Result Verification - Whenever the p-value is greater than or equal to 0.05(a threshold), we have to accept the null hypothesis and therefore numbers are randomly generated.

2. **Chi-Square test -**

Null and alternative hypothesis -

- H_0 (null): Data is produced randomly
- H_1 (alternative): Data is not produced randomly

Implemented using chisquare() function available in scipy.stats library.

We first need to calculate the frequency of each random number that is generated and that list of frequencies is taken as input by this chi-square function.

- Input - List containing the frequency count of the unique entries in randomly generated result
- Output - z-test statistic and the p-value.

Result Verification - Whenever the p-value is greater than or equal to 0.05(a threshold) we have to accept the null hypothesis and therefore numbers are randomly generated.

Code Implementation -
We have first implemented Blum Blum Shub followed by Lagged Fibonacci generator.
We first select p and q by iterating to the primes in till 2000 and selecting the best possible pair according to the 3 requirements as explained in the above explantation. On evaluating the output of the code, p and q turns out ot be 809 and 911. Then, we choose the mod and seed value and evaluate the answer.
Fro the Lagged Fibonacci part, pre-populate the required lfg list and set i and j to 3 and 7 respectively and Mod to 1145 and evaluate the answer.
The number of generated random numbers are 1000.

```
from statsmodels.sandbox.stats.runs import runstest_1samp
from scipy.stats import chisquare, kstest, wilcoxon
import math

########------------Uitility Functions----------########
# Selects 2 safe prime numbers p and q having lowest possible
# gcd((p1 - 3) / 2, (p2 - 3) / 2) for Blum Blum Shub
def prime(p):
    for i in range(2, int(1e9)):
        if i * i > p:
            break
        if p % i == 0:
            return False
    return True


def selectpq(l=800, r=2000):
    gcd, p = int(1e8), (-1, -1)
    for p1 in range(l, r + 1):
        if not prime(p1) or not prime(2 * p1 + 1):
            continue
        for p2 in range(p1 + 1, r + 1):
            if not prime(p2) or not prime(2 * p2 + 1) or (p1 * p2) % 4 != 3:
                continue
            if math.gcd((p1 - 3) // 2, (p2 - 3) // 2) < gcd:
                gcd = math.gcd((p1 - 3) // 2, (p2 - 3) // 2)
                p = (p1, p2)
    return p
```

```python
# Pretty prints a list li
def print_list(li):
    for i in li[:-1]:
        print(i, end=", ")
    print(li[-1])


# Returns frequency numbers without the key as a list
def count_freq(li):
    f = {}
    for i in li:
        if i not in f:
            f[i] = 1
        else:
            f[i] += 1
    r = []
    for k, v in f.items():
        r.append(v)
    return r


# Blum Blum Shub implementation
def BlumBlumShub(num_generations, p, q, seed):
    ret = []
    ret.append(seed)
    M = p * q
    for i in range(num_generations - 1):
        ret.append((ret[-1] * ret[-1]) % M)
    return ret


# Lagged Fibonacci Generator implementation
def gen_LFG(i, j, n, lfg):
    lfg_li = []
    lfg_mod = 1145
    for i1 in range(n):
        lfg_li.append((lfg[i - 1] + lfg[j - 1]) % lfg_mod)
        lfg = lfg[1:]
```

```
        lfg.append(lfg_li[-1])
    return lfg_li


# Select p, q, seed and number of elements to generate
p, q = selectpq()
print(p, q)
seed = 18
n = 1000
# Calculate Blum Blum Shub and print the list as the answer
li_bl = BlumBlumShub(n, p, q, seed)
freq_bl = count_freq(li_bl)
print_list(li_bl)
print("Runs test - p value for Blum Blum Shub : ", runstest_1samp(li_bl)[1])
print("Chisquare test - p value for Blum Blum Shub : ", chisquare(freq_bl))
# Lagged Fibonacci Generator
# Calculate LFG list and print the list as the answer
i, j, lfg = 3, 7, [5, 63, 72, 31, 421, 141, 54, 323, 26, 412, 14, 241]
li_lfg = gen_LFG(i, j, n, lfg)
print_list(li_bl)
freq_lfg = count_freq(li_lfg)
print("Runs test; p value for LFG : ", runstest_1samp(li_lfg)[1])
print("Chisquare test; p value for LFG : ", chisquare(freq_lfg))
```

Output -

```
(base) D:\Downloads\Assignments\Practical Assignment-4 (PA4)>C:/Users/Asus/anaconda3/python.exe "d:/Downlo
ads/Assignments/Practical Assignment-4 (PA4)/BlumBlumShub.py"
18, 324, 104976, 351528, 49453, 236527, 164638, 321822, 404212, 558636, 60935, 73263, 640451, 689951, 3063
07, 320554, 255339, 125385, 472556, 687133, 720329, 40277, 101930, 249997, 247810, 91424, 42117, 622095, 3
29130, 232883, 209277, 697154, 128179, 674333, 296884, 188049, 477382, 217142, 400140, 460848, 614273, 359
512, 626515, 536818, 323133, 602364, 92820, 34090, 617676, 631647, 575963, 546482, 263538, 439680, 716704,
 641583, 64409, 688909, 682237, 27713, 57411, 163393, 220673, 101003, 65851, 589084, 294911, 719929, 27029
5, 676155, 46359, 67797, 507445, 347415, 329993, 92804, 12102, 532602, 608295, 667091, 88095, 129555, 8279
9, 109703, 291538, 732768, 213385, 623006, 374680, 58882, 246628, 105915, 125446, 296268, 357921, 65064, 1
840, 437604, 599649, 59097, 554147, 157270, 166460, 717196, 75341, 636982, 112862, 277327, 734284, 1235, 5
1227, 489089, 284491, 109898, 367791, 686222, 281227, 525840, 420780, 642638, 313402, 219875, 92222, 66582
3, 628849, 248370, 103601, 250764, 355018, 633338, 157501, 652659, 458251, 116932, 287176, 603875, 121422,
 374088, 461624, 89517, 640161, 22968, 574739, 492323, 653205, 44963, 83112, 449916, 261716, 51594, 637447
, 175151, 289426, 103136, 664928, 597088, 394481, 131508, 672529, 443539, 438450, 320339, 282157, 466671,
91739, 252540, 243135, 675434, 600367, 118754, 36651, 483623, 151485, 504361, 391477, 458472, 37990, 19605
8, 556519, 622596, 418167, 309153, 73091, 525529, 672577, 152715, 274869, 251675, 400568, 459337, 194852,
61420, 455518, 475866, 348213, 480890, 382879, 594550, 661133, 424765, 580035, 557725, 114684, 672701, 405
413, 79581, 103154, 693153, 378324, 158181, 112711, 117758, 310379, 510353, 316015, 641727, 611299, 705438
, 411072, 321465, 694441, 376821, 153706, 294492, 654737, 648825, 51825, 206269, 685090, 75937, 147793, 33
1486, 102291, 273878, 348660, 232544, 147310, 37544, 409848, 45022, 233234, 202566, 565031, 171150, 297245
, 201909, 144596, 78585, 287604, 452049, 585671, 130656, 619498, 282734, 655220, 275915, 38521, 288454, 73
4013, 72208, 464338, 720794, 230381, 422176, 421811, 432138, 233427, 354261, 244407, 275700, 98135, 112292
, 177373, 167817, 339701, 413977, 368062, 375656, 546811, 301423, 62207, 466099, 134575, 154198, 698465, 5
53170, 223093, 207180, 730640, 638935, 185144, 477246, 99558, 632812, 409697, 109559, 408767, 258006, 6093
57, 380270, 173108, 324, 104976, 351528, 49453, 236527, 164638, 321822, 404212, 558636, 60935, 73263, 6404
51, 689951, 306307, 320554, 255339, 125385, 472556, 687133, 720329, 40277, 101930, 249997, 247810, 91424,
42117, 622095, 329130, 232883, 209277, 697154, 128179, 674333, 296884, 188049, 477382, 217142, 400140, 460
848, 614273, 359512, 626515, 536818, 323133, 602364, 92820, 34090, 617676, 631647, 575963, 546482, 263538,
 439680, 716704, 641583, 64409, 688909, 682237, 27713, 57411, 163393, 220673, 101003, 65851, 589084, 29491
```

```
 546482, 263538, 439680, 716704, 641583, 64409, 688909, 682237, 27713, 57411, 163393, 220673, 101003, 6585
1, 589084, 294911, 719929, 270295, 676155, 46359, 67797, 507445, 347415, 329993, 92804, 12102, 532602, 608
295, 667091, 88095, 129555, 82799, 109703, 291538, 732768, 213385, 623006, 374680, 58882, 246628, 105915,
125446, 296268, 357921, 65064, 1840, 437604, 599649, 59097, 554147, 157270, 166460, 717196, 75341, 636982,
 112862, 277327, 734284, 1235, 51227, 489089, 284491, 109898, 367791, 686222, 281227, 525840, 420780, 6426
38, 313402, 219875, 92222, 665823, 628849, 248370, 103601, 250764, 355018, 633338, 157501, 652659, 458251,
 116932, 287176, 603875, 121422, 374088, 461624, 89517, 640161, 22968, 574739, 492323, 653205, 44963, 8311
2, 449916, 261716, 51594, 637447, 175151, 289426, 103136, 664928, 597088, 394481, 131508, 672529, 443539,
438450, 320339, 282157, 466671, 91739, 252540, 243135, 675434, 600367, 118754, 36651, 483623, 151485, 5043
61, 391477, 458472, 37990, 196058, 556519, 622596, 418167, 309153, 73091, 525529, 672577, 152715, 274869,
251675, 400568, 459337, 194852, 61420, 455518, 475866, 348213, 480890, 382879, 594550, 661133, 424765, 580
035, 557725, 114684, 672701, 405413, 79581, 103154, 693153, 378324, 158181, 112711, 117758, 310379, 510353
, 316015, 641727, 611299, 705438, 411072, 321465, 694441, 376821, 153706, 294492, 654737, 648825, 51825, 2
06269, 685090, 75937, 147793, 331486, 102291, 273878, 348660, 232544, 147310, 37544, 409848, 45022, 233234
, 202566, 565031, 171150, 297245, 201909, 144596, 78585, 287604, 452049, 585671, 130656, 619498, 282734, 6
55220, 275915, 38521, 288454, 734013, 72208, 464338, 720794, 230381, 422176, 421811, 432138, 233427, 35426
1, 244407, 275700, 98135, 112292, 177373, 167817, 339701, 413977, 368062, 375656, 546811, 301423, 62207, 4
66099, 134575, 154198, 698465, 553170, 223093, 207180, 730640, 638935, 185144, 477246, 99558, 632812, 4096
97, 109559, 408767, 258006, 609357, 380270, 173108, 324, 104976, 351528, 49453, 236527, 164638, 321822, 40
4212, 558636, 60935, 73263, 640451, 689951, 306307, 320554, 255339, 125385, 472556, 687133, 720329, 40277,
 101930, 249997, 247810, 91424, 42117, 622095, 329130, 232883, 209277, 697154, 128179, 674333, 296884, 188
049, 477382, 217142, 400140, 460848, 614273, 359512, 626515, 536818, 323133, 602364, 92820, 34090, 617676,
 631647, 575963, 546482, 263538, 439680, 716704, 641583, 64409, 688909, 682237, 27713, 57411, 163393, 2206
73, 101003, 65851, 589084, 294911, 719929, 270295, 676155, 46359, 67797, 507445, 347415, 329993, 92804, 12
102, 532602, 608295, 667091, 88095, 129555, 82799, 109703, 291538, 732768, 213385, 623006, 374680, 58882,
246628, 105915, 125446, 296268, 357921, 65064, 1840, 437604, 599649, 59097
Runs test - p value for Blum Blum Shub :  0.33614682139644303
Chisquare test - p value for Blum Blum Shub : 0.9962944321907298
```

```
 546482, 263538, 439680, 716704, 641583, 64409, 688909, 682237, 27713, 57411, 163393, 220673, 101003, 6585
1, 589084, 294911, 719929, 270295, 676155, 46359, 67797, 507445, 347415, 329993, 92804, 12102, 532602, 608
295, 667091, 88095, 129555, 82799, 109703, 291538, 732768, 213385, 623006, 374680, 58882, 246628, 105915,
125446, 296268, 357921, 65064, 1840, 437604, 599649, 59097, 554147, 157270, 166460, 717196, 75341, 636982,
 112862, 277327, 734284, 1235, 51227, 489089, 284491, 109898, 367791, 686222, 281227, 525840, 420780, 6426
38, 313402, 219875, 92222, 665823, 628849, 248370, 103601, 250764, 355018, 633338, 157501, 652659, 458251,
 116932, 287176, 603875, 121422, 374088, 461624, 89517, 640161, 22968, 574739, 492323, 653205, 44963, 8311
2, 449916, 261716, 51594, 637447, 175151, 289426, 103136, 664928, 597088, 394481, 131508, 672529, 443539,
438450, 320339, 282157, 466671, 91739, 252540, 243135, 675434, 600367, 118754, 36651, 483623, 151485, 5043
61, 391477, 458472, 37990, 196058, 556519, 622596, 418167, 309153, 73091, 525529, 672577, 152715, 274869,
251675, 400568, 459337, 194852, 61420, 455518, 475866, 348213, 480890, 382879, 594550, 661133, 424765, 580
035, 557725, 114684, 672701, 405413, 79581, 103154, 693153, 378324, 158181, 112711, 117758, 310379, 510353
, 316015, 641727, 611299, 705438, 411072, 321465, 694441, 376821, 153706, 294492, 654737, 648825, 51825, 2
06269, 685090, 75937, 147793, 331486, 102291, 273878, 348660, 232544, 147310, 37544, 409848, 45022, 233234
, 202566, 565031, 171150, 297245, 201909, 144596, 78585, 287604, 452049, 585671, 130656, 619498, 282734, 6
55220, 275915, 38521, 288454, 734013, 72208, 464338, 720794, 230381, 422176, 421811, 432138, 233427, 35426
1, 244407, 275700, 98135, 112292, 177373, 167817, 339701, 413977, 368062, 375656, 546811, 301423, 62207, 4
66099, 134575, 154198, 698465, 553170, 223093, 207180, 730640, 638935, 185144, 477246, 99558, 632812, 4096
97, 109559, 408767, 258006, 609357, 380270, 173108, 324, 104976, 351528, 49453, 236527, 164638, 321822, 40
4212, 558636, 60935, 73263, 640451, 689951, 306307, 320554, 255339, 125385, 472556, 687133, 720329, 40277,
 101930, 249997, 247810, 91424, 42117, 622095, 329130, 232883, 209277, 697154, 128179, 674333, 296884, 188
049, 477382, 217142, 400140, 460848, 614273, 359512, 626515, 536818, 323133, 602364, 92820, 34090, 617676,
 631647, 575963, 546482, 263538, 439680, 716704, 641583, 64409, 688909, 682237, 27713, 57411, 163393, 2206
73, 101003, 65851, 589084, 294911, 719929, 270295, 676155, 46359, 67797, 507445, 347415, 329993, 92804, 12
102, 532602, 608295, 667091, 88095, 129555, 82799, 109703, 291538, 732768, 213385, 623006, 374680, 58882,
246628, 105915, 125446, 296268, 357921, 65064, 1840, 437604, 599649, 59097
Runs test; p value for LFG :  0.4118560797612164
Chisquare test; p value for LFG : 0.9672389419082345
```

Results -

On running Blum Blum Shub PRNG,

Runs Test p val = 0.33614682139644303, which is greater than 0.05, Thus it fails to reject Null Hypothesis

Chi-Square test p val = 0.9962944321907298, which is also greater than 0.05, It also fails to reject the Null hypothesis.

Hence the numbers are randomly generated.

On running Lagged Fibonacci Generator PRNG,

Runs Test p val = 0.4118560797612164, which is greater than 0.05, Thus it fails to reject Null Hypothesis

Chi-Square test p val = 0.9672389419082345, which is also greater than 0.05, It also fails to reject the Null hypothesis.

Hence the numbers are randomly generated.