# CS 3430: S22: Scientific Computing
# Assignment 10
# Pseudorandom Numbers, Random Image Art, and Tests of Randomness

Vladimir Kulyukin
Department of Computer Science
Utah State University

April 02, 2022

## Learning Objectives

1. Pseudorandom Number Generation
2. Linear Congruential Generator
3. Xorshift
4. Random Image Art
5. Equidistribution Test of Sequence Randomness

## Introduction

In this assignment, we'll experiment with the Linear Congruential, 32-bit Xorshift, and Mersenne Twister pseudorandom number generators (PRNGs), do some random image art to visualize how these generators work, and implement the Equidistribution Test of number sequence randomness.

You'll code up your solutions to the three problems in this assignment in `prng.py`. Included in the zip is `cs3430_s22_hw10_uts.py` with my unit tests for this assignment. Remember not to run all unit tests at once. Try to work on one unit test at a time.

## Problem 0: (0 points)

Review the slides for Lectures 20 and 21 (or you class notes if you attend my F2F lectures) to become comfortable with the LCG, Xorshift, Mesenne Twister, $\chi^2$ (chi-square), mathematical expectation, and the Equidistribution Test of sequence randomness.

## Problem 1: LCG, XORSHIFT, Mersenne Twister (2 points)

Implement the static method `lcg(a, b, m, n, x0=0)` in `prng.py` that generates random numbers with the LCG method discussed on Slides 7–13 in the Lecture 20 PDF in Canvas. This method takes the parameters $a$, $b$, $m$ explained on the slides. The parameter $n$ specifies how many numbers will be generated and the keyword `x0` defines the seed. This method returns a generator (i.e., a Python generator) that can generates $n$ random numbers. Here's a trial run.

```
>>> from prng import prng
>>> a, b, m, n, seed = 214013, 2531011, 4294967296, 5, 0
```

We can now use LCG to generate 5 random numbers with the seed of 0. Here's how.

```
>>> lcgg = prng.lcg(a, b, m, n, x0=seed)()
>>> rns = [next(lcgg) for _ in range(n)]
>>> rns
[2531011, 505908858, 3539360597, 159719620, 2727824503]
```

Note that the call `prng.lcg(a, b, m, n, x0=seed)()` indicates that `prng.lcg(a, b, m, n, x0=seed)` returns a generator as a function, which is why we have to call () to turn it into a generator object that we can use with `next()`.

Let's run the second unit test for Problem 1 in `cs3430_s22_hw10_uts.py`.

```
def test_hw10_prob01_ut02(self):
    print('\n***** CS3430: S22: HW10: Problem 01: Unit Test 02 ************')
    a, b, m, n, seed = 214013, 2531011, 4294967296, 10, 1
    lcgg = prng.lcg(a, b, m, n, x0=seed)()
    rns = [next(lcgg) for _ in range(n)]
    print('LCG random numbers: {}'.format(rns))
    assert len(rns) == n
    if self.__check_uniqueness(rns):
       print('all random numbers are unique')
    print('CS 3430: S22: HW10: Problem 01: Unit Test 02: pass...')
```

This test generates $n = 10$ random numbers with LCG with $a = 214013$, $b = 2531011$, $m = 4294967296$, and $x0 = 1$. Here's my output. The private method `__check_uniqueness()` returns `True` when all generated numbers are unique. My output for the other unit tests for Problem 1 is in `cs3430_s22_hw10_output.txt`.

```
***** CS3430: S22: HW10: Problem 01: Unit Test 02 ************
LCG random numbers: [2745024, 3357800067, 415139642, 3884216597, 3403800452,
1030492215, 752224798, 1924036713, 1766988168, 3750785579]
all random numbers are unique
CS 3430: S22: HW10: Problem 01: Unit Test 02: pass...
```

Implement the static method `xorshift(a, b, c, n, x0=1)` in `prng.py` that generates random numbers with the 32 xorshit method discussed on Slides 14–20 in the Lecture 20 PDF in Canvas. This method takes the parameters $a$, $b$, $c$. The parameter $n$ specifies how many numbers will be generated and the keyword `x0` defines the seed. This method returns a Python generator that generates $n$ random numbers. Here's a trial run.

```
>>> a, b, c, n, seed = 1, 3, 10, 5, 1
>>> xsg = prng.xorshift(a, b, c, n, x0=seed)()
>>> rns = [next(xsg) for _ in range(n)]
>>> rns
[1, 3075, 5898885, 3488497534, 2316485042]
```

We can use use the following XOR, SHIFT, and AND commands to implement the 32-bit xorshift.

```
Xn = Xn ^ ((Xn << a) & 0xFFFFFFFF)
Xn = Xn ^ ((Xn >> b) & 0xFFFFFFFF)
Xn = Xn ^ ((Xn << c) & 0xFFFFFFFF)
```

Let's run the 11-th unit test for Problem 1 in `cs3430_s22_hw10_uts.py`.

```
def test_hw10_prob01_ut11(self):
    print('\n***** CS3430: S22: HW10: Problem 01: Unit Test 11 ************')
    a, b, c, n, seed = 1, 3, 10, 10, 3
    xsg = prng.xorshift(a, b, c, n, x0=seed)()
    rns = [next(xsg) for _ in range(n)]
    print('XORSHIFT random numbers: {}'.format(rns))
    assert len(rns) == n
    if self.__check_uniqueness(rns):
       print('all random numbers are unique')
    print('CS 3430: S22: HW10: Problem 01: Unit Test 11: pass...')
```

Here's my output.

```
***** CS3430: S22: HW10: Problem 01: Unit Test 11 ************
XORSHIFT random numbers: [3, 5125, 15598478, 1342456832, 3680195968, 559219024,
1812913038, 3868524800, 222651232, 3430023444]
all random numbers are unique
CS 3430: S22: HW10: Problem 01: Unit Test 11: pass...
```

Implement the static method `mersenne_twister(n, x0=1, start=0, stop=1000)` in `prng.py` that generates random numbers with Python's Mersenne Twister method. This method can be implemented with Python's `random.randint(start, stop)` that to generate random integers in the range `[start, stop]`. This method also takes the parameters $n$ (the number of random numbers to generate), `x0` (the seed), `start` (the lower bound of the range), `stop` (the upper bound for the range). This method returns a Python generator that generates $n$ random numbers. Here's a trial run.

```
>>> from prng import prng
>>> seed, start, stop, n = 1, 0, 1000, 5
>>> mtw = prng.mersenne_twister(n, x0=seed, start=start, stop=stop)()
>>> rns = [next(mtw) for _ in range(n)]
>>> rns
[137, 582, 867, 821, 782]
```

## Problem 2: Random Image Art (1 point)

One way to estimate the randomness of a PRNG is to turn the generated random numbers into an image and display it. If we don't see any patterns in the image, then the numbers are random. If we see a pattern, then the numbers are not random. We can forget about randomness and enjoy the generated images as works of art.

The method `gen_lcg_data(a, b, m, n, seed=0, option=1)` uses the LCG method to generate $n$ numbers for a PIL image.

```
def gen_lcg_data(a, b, m, n, seed=0, option=1):
    lcgg = prng.lcg(a, b, m, n, x0=seed)()
    ### option 1) generate n lcg numbers.
    if option == 1:
       return np.array([next(lcgg) for _ in range(n)])
    ### option 2) generate n lcg numbers by
    ### varying the seed from i upto n-1.
    elif option == 2:
```

```
        data = np.zeros(n)
        for i in range(n):
            data[i] = next(prng.lcg(a, b, m, 1, x0=i)())
        return data
    ### option 3) generate n numbers with numpy arange.
    elif option == 3:
        return np.arange(n)
    else:
        raise Exception('prng.get_lcg_data(): option must be 1,2,3')
```

When `option==1`, we generate $n$ numbers from one seed. When `option==2`, we generate $n$ numbers with the seed that ranges from 0 upto $n-1$. When `option==3`, we simply generate an array of numbers from 0 upto $n-1$.

Follow the method `gen_lcg_data(a, b, m, n, seed=0, option=1)` as an example to implement the methods `gen_xorshift_data()` and `gen_mersenne_twister_data()` that generate $n$ numbers with the xorshift and Mersenne Twister methods, respectively. Each should implement the same three options.

You can use the method `gen_pil_image()` in `prng.py` to turn an array of random numbers into a PIL image and display it or save it in the current directory.

```
def gen_pil_image(data, w, h, n, name=None, save_flag=False):
    img_data = np.zeros((n, 3))
    for i, j in enumerate(data):
        img_data[i] = prng.__int_to_rgb(j)
    img_data = img_data.reshape(h, w, 3)
    pil_img = Image.fromarray(img_data, 'RGB')
    ### save PIL image in the current directory
    if save_flag:
        pil_img.save(name + '.png')
    else:
        pil_img.show()
```

Let's run `test_hw10_prob02_ut01()`. We should see the left image on Slide 10 in the Lecture 20 PDF.

```
def test_hw10_prob02_ut01(self):
    print('\n***** CS3430: S22: HW10: Problem 02: Unit Test 01 ************')
    w, h = 600, 600
    n = h * w
    a, b, m, seed = 214013, 2531011, 4294967291, 0
    data = prng.gen_lcg_data(a, b, m, n, seed=seed, option=1)
    prng.gen_pil_image(data, w, h, n)
    print('CS 3430: S22: HW07: Problem 02: Unit Test 01: pass')
```

If we run `test_hw10_prob02_ut02()`, we should see the middle image on Slide 10 in the Lecture 20 PDF.

```
def test_hw10_prob02_ut02(self):
    print('\n***** CS3430: S22: HW10: Problem 02: Unit Test 02 ************')
    w, h = 600, 600
    n = h * w
    a, b, m, seed = 214013, 2531011, 4294967291, 0
    data = prng.gen_lcg_data(a, b, m, n, seed=seed, option=2)
    prng.gen_pil_image(data, w, h, n)
    print('CS 3430: S22: HW07: Problem 02: Unit Test 02: pass')
```

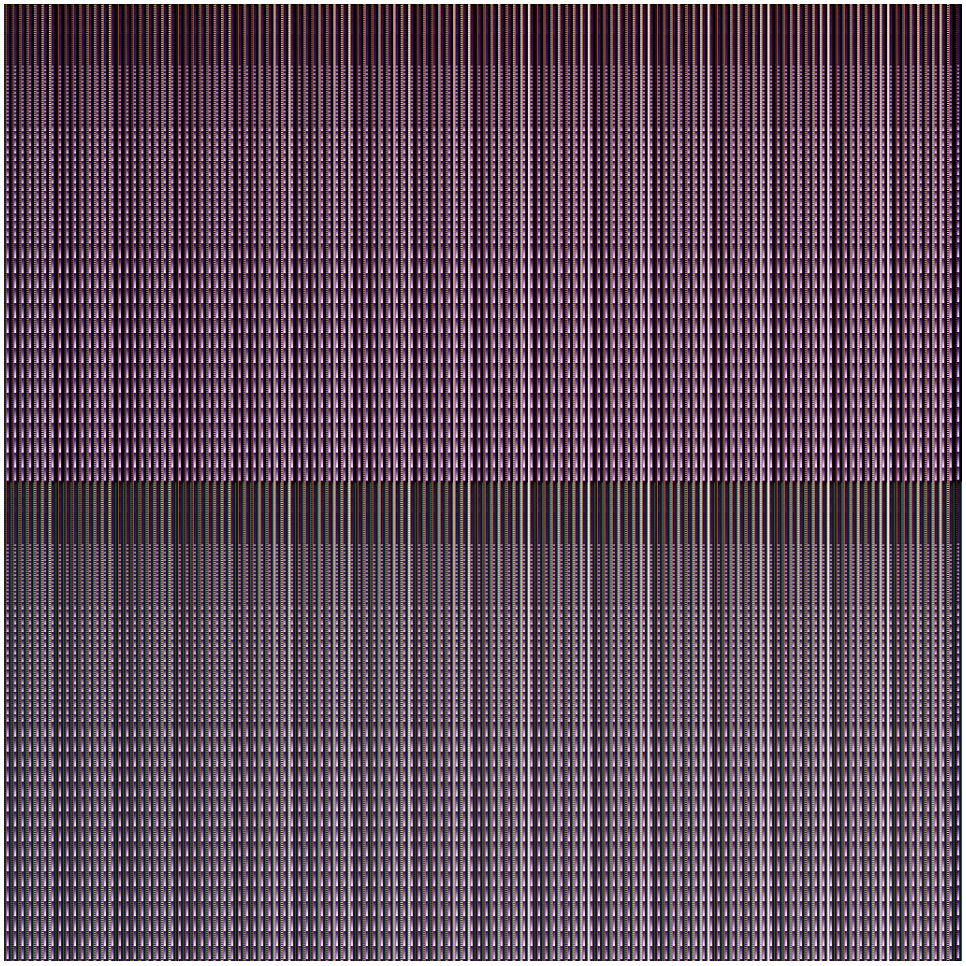

Figure 1: Image generated with test_hw10_prob02_ut08().

If we run `test_hw10_prob02_ut03()`, we should see the right image on Slide 10 in the Lecture 20 PDF.

```
def test_hw10_prob02_ut03(self):
    print('\n***** CS3430: S22: HW10: Problem 02: Unit Test 03 ************')
    w, h = 600, 600
    n = h * w
    a, b, m, seed = 214013, 2531011, 4294967291, 0
    data = prng.gen_lcg_data(a, b, m, n, seed=seed, option=3)
    prng.gen_pil_image(data, w, h, n)
    print('CS 3430: S22: HW07: Problem 02: Unit Test 03: pass')
```

My favorite image for this assignment is the one in Figure 1 generated with `test_hw10_prob02_ut08()`. It's not random, but the pattern is intricate.

## Problem 3: Equidistribution Test of Sequence Randomness (2 points)

Implement `equidistrib_test(seq, n, lower_bound, upper_bound)` that executes the Equidistribution Test on Slide 13 in the Lecture 21 PDF in Canvas. This method returns two values: the V statistic and its $p$ value. The argument `seq` is the sequence of $n$ random numbers in the range

[lower_bound, upper_bound]. Your implementation should use scipy.stats.chisquare(fobs, fexp), as shown on the slide. This scipy method returns the V statistic and its $p$ value.

The unit test test_hw10_prob03_ut01(self) runs the example on Slide 14.

```
def test_hw10_prob03_ut01(self):
    print('\n***** CS3430: S22: HW10: Problem 03: Unit Test 01 ***********')
    seq = [1, 5, 7, 8, 5, 1, 3, 4, 3, 3, 2, 2, 0, 7, 9, 8, 7, 4, 3, 1]
    n, lower_bound, upper_bound = 20, 0, 9
    v_stat, p_val = prng.equidistrib_test(seq, n, lower_bound, upper_bound)
    print('seq   = {}'.format(seq))
    print('V     = {}'.format(v_stat))
    print('p val = {}'.format(p_val))
    print('CS 3430: S22: HW10: Problem 03: Unit Test 01: pass')
```

Since the $p$-value for the V statistic (i.e., 6) is in [0.25, 0.75], the sequence is considered random. The module cs3430_s22_hw10_uts.py has a total of 12 tests for Problem 3. My output is in cs3430_s22_hw10_uts_output.txt.

## Quotes on Randomness and Probability Theory

A few famous quotes to think about while you work on this assignment.

```
In no other branch of mathematics is it so easy
for experts to blunder as in probability theory.

                                Martin Gardner

Probability is expectation founded upon partial
knowledge. A perfect acquaintance with all the
circumstances affecting the occurences of an
event would change expection into ceratinty
and leave neither room nor demand for a theory
of probabilities.

                                George Boole

No amount of experimentation can ever prove
me right; a single experiment can prove me
wrong.

                            Albert Einstein

God does not play dice.

                            Albert Einstein
```

## What To Submit

Submit your code in prng.py along with your favorite random image.

Happy Hacking!