RNG_challenge

August 23, 2019

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
```

From Wikipedia https://en.wikipedia.org/wiki/Linear_congruential_generator

A linear congruential generator (LCG) is an algorithm that yields a sequence of pseudo-randomized numbers calculated with a discontinuous piecewise linear equation. The method represents one of the oldest and best-known pseudorandom number generator algorithms. The theory behind them is relatively easy to understand, and they are easily implemented and fast, especially on computer hardware which can provide modulo arithmetic by storage-bit truncation.

The generator is defined by recurrence relation:

$$X_{n+1} = (aX_n + c) \, mod \, m \tag{1}$$

where X is the sequence of pseudorandom values, and

$$m, 0 < m - the\ modulus \tag{2}$$
$$a, 0 < a < m - the\ multiplier \tag{3}$$
$$c, 0 \leqslant c < m - the\ increment \tag{4}$$
$$X_0, 0 \leqslant X_0 < m - the\ seed \tag{5}$$

are integer constants that specify the generator.

```
In [2]: # helper function
        def int_to_bin_seq(x, num_digit):
            '''
            converts integer to binary sequence list
            of specified length by num_digit

            e.g. x=4, num_digit=3  to [1,0,0]
            e.g. x=4, num_digit=4  to [0,1,0,0]
            '''
            # convert int to binary string
            # e.g. 4 to '100'
            x_bin = '{0:b}'.format(x)
            # convert binary string to sequence of integers
            # e.g. '100' to [1, 0 ,0]
```

```
        x_bin_seq = list(map(int,x_bin))

        # add zeros in front if needed
        x_bin_seq = [0]*(num_digit-len(x_bin_seq)) + x_bin_seq

        return x_bin_seq
```

In [3]:
```
def lcg_generator(m, a, c, seq_len, seed = 'random'):
    '''
    For a given m and coeffs a and c,
    generates binary numbers of length (seq_len),
    based on linear congruential generator algorithm.
    '''
    # calculate the num of binary digits
    # necessary to represent nums.
    num_digit = int(np.ceil(np.log2(m)))

    # generate random seed
    if seed == 'random':
        x_init = np.random.randint(0,m)
    else:
        x_init = seed

    # convert int to binary sequence
    x_init_bin_seq = int_to_bin_seq(x_init, num_digit)

    # initialize sequence
    sequence = []
    sequence += x_init_bin_seq

    # initialize recurrence relation
    x = x_init

    while len(sequence) < seq_len:
        # recurrence relation
        x_next = (a*x + c) % m
        # convert int to binary sequence
        x_next_bin_seq = int_to_bin_seq(x_next, num_digit)
        # add this to sequence
        sequence += x_next_bin_seq
        # prepare for the following loop
        x = x_next

    # crop to fixed size
    sequence = sequence[0:seq_len]

    return sequence
```

In [4]:
```
# 10 example RNs generated by RNG
```

```
        RNs = lcg_generator(m=15, a=5, c=10, seq_len=10, seed = 'random')
        RNs
```

Out[4]: [1, 1, 0, 1, 0, 0, 0, 0, 1, 0]

```
In [ ]: def dataset_generator():
            # create a dataset generator
            # which should take the RNs
            # and arranges them for your chosen ML model
            return np.array(X), np.array(y)
```

```
In [ ]: # given a fixed length sequence
        # generated by the lcg_generator
        # predict the next number by building an ML model
```

```
In [ ]: # Jeroen Random Number Generator (JRNG)
        f = open("JRNG.txt", "r") # open
        JRNG = list(f.read()) # read
        JRNG = list(map(int, JRNG)) # convert to integers
        JRNG = np.array(JRNG) # convert list to np array
```

```
In [ ]: # do the same for JRNG
```