3D Random Walk

LYU Liuke

PHYS3061: Lab Report 1

Contents

1 Introduction

1.1 Random Number Generator

The random number generator implemented here is a simple Linear Congruential Generator(LCG), which is generally a recursive mapping within a certain integer domain. for a set of parameters (a,c,m):

$$x_{N+1} = (x_N * a + c) \mod m$$
$$\xi \colon \{1, 2, 3, \ldots, m-1\} \to \{1, 2, 3, \ldots, m-1\}$$

Some special sets of (a,c,m) give seemingly very random and uniform distribution of results in recursive mapping. Based on this feature, we consider this recursive method as a random number generator.

In this experiment (a=1559, c=647, m=13229) are chosen. The main characteristic of these parameters is that they are all primes, in order to try to avoid periodicity behaviors in the random sequance. This special set is chosen by testing different combinations of (a,c,m) in a small parameter space.

1.2 Random Walk

A random walk, in simple terms, is a random sequance representing the accumulative sum of a random variable. In a simple 3D case:

$$\vec{X_N} = \sum_{i=1}^{n} \vec{s_i}$$
$$\text{where} \quad \vec{s} \in V = \{(1,0,0), (-1,0,0), (0,1,0), (0,-1,0), (0,0,1), (0,0,-1)\}$$
$$\text{Assigning} \quad P(\vec{v}) \equiv 1/6 \quad \text{for} \quad \vec{v} \in V$$

Probability analysis give this famous relationship:

$$< |\vec{X_N}|^2 >= \sqrt{N}$$

This relationship can be verified by a Monte Carlo Method, averaging the results over a large number of samples. Alternatively, we can use it to evaluate the randomness of our LCG generator.

2 Code

2.1 Linear Congruential Generator

```
def LCG():
    '''LCG Random Number generator'''
    # seed, a, c, m are all prime numbers, to avoid periodicity
    a = 1559
    c = 647
    m = 13229

    t = time.time() #Using current time to set the seed
    state = int(   (t - int(t)) * 10000   ) % m

    logging_dict(locals()) #Recording the parameters in results.log

    '''LCG is a generator object'''
    while True:
        yield state / m
        state = (a*state+c) % m
```

LCG is the linear congruential generator with parameters a=1559, c=647, m=13229. The seed of the generator is determined by decimal time. The structure of it is a python generator, which is a sequance of predefined operations. This gives a good structure to continuously generate any length of random numbers and also avoids defining a global state variable. The outputs of this function (after "yield") is normalized by dividing it by m.

2.2 3D Random Walk

```
def random_choice(choices, size=1):
    '''Using LCG to randomly choose from a list of objects'''
    def hash(p):
        if type(choices) == int:
            return int(p * choices)
        else:
            return choices[int(p * len(choices))]
    vhash = np.vectorize(hash) # vectorizing hash function to make it operate on a numpy arra

    random_ns = np.fromiter(LCG, dtype=float, count=size) #from generator LCG draw (size) ran
    return vhash(random_ns)

def RandomWalk(steps, RNG='npRNG', dim=3):
    '''Using two random choices, determine the dimension and direction of a (steps) steps wal
    walks = np.zeros((steps,dim), dtype=int)
    if RNG=='LCG':
        dims = random_choice(dim, size=steps)
        ahead = random_choice([-1,1], size=steps)
    elif RNG=='npRNG':
        dims = np.random.choice(dim, size=steps)
        ahead = np.random.choice([-1,1], size=steps)

    walks[np.arange(steps), dims] = ahead

    pos = np.array(walks).cumsum(axis=0) # cumulative sum of walks in every step
    # returning a 2D array of size (steps, 3)
    return pos
```

3 Application

3.1 Pólya's Random Walk Problem

4 Results and Analysis

5 Conclusion