Lab Session-6: Pseudorandom Number Generation

MATH350 – Statistical Inference

Dr. Tanujit Chakraborty
Assistant Professor in Statistics at Sorbonne University
Mail me at tanujitisi@gmail.com
Course Webpage: https://www.ctanujit.org/SI.html
Code available at https://github.com/tanujit123/MATH350

Image from reddit.com

Random Number Generation

The goal is to generate a sequence of numbers that are distributed randomly according to a uniform probability distribution. Desired Property:

- Long Periodicity: e.g., if a 32-bit integer is used the period should be close to $2^{31} - 1 = 2147483647$.
- Best Randomness: The correlation among the generated numbers should be small, i.e., $\langle x_i x_{i+l} \rangle$ should have a uniform distribution for $l \neq 0$.

Two types of generators:

- True Random number generator: Based on the physical phenomena, such as radioactive decay, atmospheric noise, etc.
- Pseudorandom number generator (PRNG): Computational algorithms produce long sequences of apparently random results.

Pseudorandom number generator (PRNG)

- Not truly random $\rightarrow$ always produces the same sequence of numbers if the input is the same.
- The series of numbers generated by these computational algorithms is generally determined by a fixed number, called a seed.
- Can be used as random numbers if the sequence of numbers has good random properties.
- Advantage: Speed and reproducibility.
- Hence, are central in applications such as Monte Carlo simulations.

Pseudorandom Generation

- The building block of computational simulation is the generation of uniform random numbers. If we can draw from $U(0,1)$, then we can draw from most other distributions. Thus the construction of sampling from $U(0,1)$ requires special attention.
- Computers can generate numbers between $(0,1)$, which although are not exactly random (and in fact deterministic), have the appearance of being $U(0,1)$ random variables. These draw from $U(0,1)$ are pseudorandom draws.
- The goal of pseudorandom generation is to draw

$$x_1, \ldots, x_n \overset{\text{approx}}{\sim} U(0,1).$$

so that they are as uniformly distributed as possible.

Multiplicative congruential method

A common algorithm to generate a sequence $\{x_n\}$ is the *multiplicative congruential* method:

1. Set seed x_0, and positive integers a, m.
2. $x_n = ax_{n-1} \mod m$
3. Return sequence x_n/m.

x_n is one of $0, 1, \dots m-1$, and so x_n/m is between $(0, 1)$.

Also note that after some finite number of steps $< m$, the algorithm will repeat itself, since when a seed x_0 is set, a deterministic sequence of numbers follows.

Multiplicative congruential example

Set $a = 123$ and $m = 10$, and let $x_0 = 7$. Then

$$\begin{aligned} x_1 &= 123 * 7 \bmod 10 = 1 \\ x_2 &= 123 * 1 \bmod 10 = 3 \\ x_3 &= 123 * 3 \bmod 10 = 9 \\ x_4 &= 123 * 9 \bmod 10 = 7 \\ x_5 &= 123 * 7 \bmod 10 = 1 \\ &\vdots \end{aligned}$$

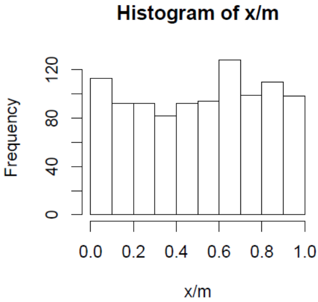
Thus, we see that the above choices of a, m, x_0 repeats itself. Naturally, both a and m should be chosen to be large so as to avoid repetition.

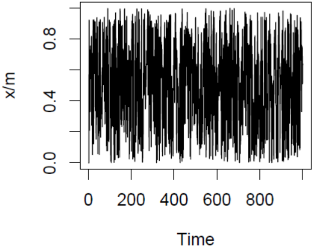
It is recommended to set $m = 2^{31} - 1$ and $a = 7^5$. Notice that both are large.

Multiplicative congruential implementation

```
m <- 2^(31) - 1
a <- 7^5
x <- numeric(length = 1e3)
x[1] <- 7
for(i in 2:1e3)
{
  x[i] <- (a * x[i-1]) %% m
}
par(mfrow = c(1,2))
hist(x/m) # looks close to uniformly distributed
plot.ts(x/m) # look like it's jumping around too
```

Multiplicative congruential results





Quality of PRNG

Any pseudorandom generation method should satisfy:

1. for any initial seed, the resultant sequence has the "appearance" of being IID from Uniform $[0, 1]$.
2. for any initial seed, the number of values generated before repetition begins is large
3. the values can be computed efficiently.

Typically m should be a large prime number

Mixed Congruential Generator

Another method is the mixed congruential generator:

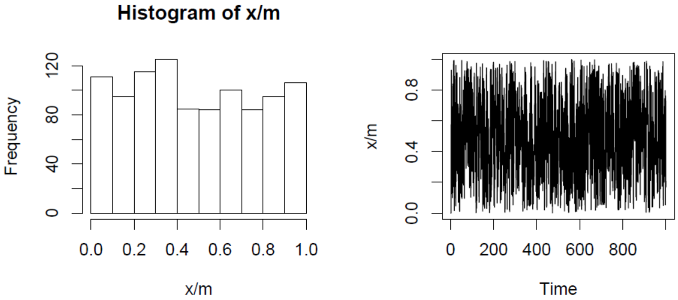
1. Set seed x_0, and positive integers a, c, m.
2. $x_n = (ax_{n-1} + c) \bmod m$
3. Return sequence x_n/m.

Implementation

```r
m <- 2^(31) - 1
a <- 7^5
c <- 2^(10) - 1
x <- numeric(length = 1e3)
x[1] <- 7

for(i in 2:1e3)
{
  x[i] <- (c + a * x[i-1]) %% m
}

par(mfrow = c(1,2))
hist(x/m) # looks close to uniformly distributed
plot.ts(x/m) # look like it's jumping around too
```



We must be cautious not to be happy with a just a histogram. A histogram shows that the empirical distribution of all samples is uniformly distributed. But we can still get a uniform looking histogram if we set $a = 1, m = 1e3$ and $c = 1$

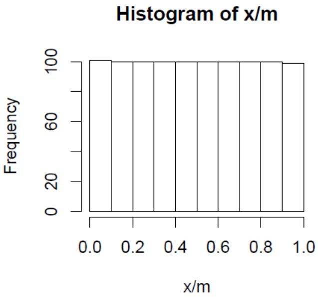
Implementation

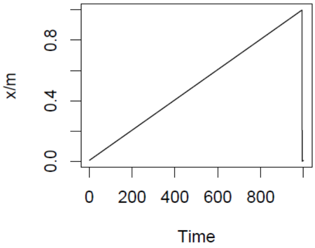
```r
m <- 1e3
a <- 1
c <- 1
x <- numeric(length = 1e3)
x[1] <- 7

for(i in 2:1e3)
{
  x[i] <- (c + a * x[i-1]) %% m
}

par(mfrow = c(1,2))
hist(x/m) # looks uniformly distributed
plot.ts(x/m) # look like it's jumping around too
```

Results


Histogram of x/m
Frequency
0
20
60
100
0.0
0.2
0.4
0.6
0.8
1.0
x/m


x/m
0.0
0.4
0.8
0
200
400
600
800
Time

- Although a histogram shows an almost perfect uniform distribution, the trace plot shows that the draws don't behave like they are independent.
- We could also use

$$x_n = (a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_k x_{n-k} + c) \mod m$$

  but this requires more flops from the computer, and so is not as computationally viable.
- We claim that these methods return "good" pseudosamples, in the sense of the three points. There are statistical hypothesis tests, like the Kolmogorov-Smirnov test, one can do to test whether a sample is truly random: independent and identically distributed.

Integrals continued

- Now that we know how to generate (pseudo) random numbers from Uniform $[0,1]$, we are equipped to estimate integrals. Consider a simple problem
$$\theta = \int_0^1 \left(3x^2 + 5x\right) dx$$
- We can now carry on a simple Monte Carlo procedure to estimate θ. What if for arbitrary a and b, interest is in
$$\int_5^{10} \left(3x^2 + 5x\right) dx?$$
- If we can draw from $U(5,10)$, then we estimate the integral. But we only know how to draw from $U(0,1)$. Note that if $U \sim U(0,1)$, then for any a, b,
$$(b-a) * U + a \sim U(a,b).$$

Integrals continued

- That means, we can draw $U \sim U(0,1)$ and set $X = (b-a) * U + a$. Then $X \sim U(a,b)$.

$$\theta = \int_5^{10} \left(3x^2 + 5x\right) dx = 5 \int_5^{10} \left(3x^2 + 5x\right) \frac{1}{5} = \mathrm{E}_{X \sim U(10,5)} \left(3X^2 + 5X\right)$$

```
set.seed(1)
repeats <- 1e4
b <- 10
a <- 5
U <- runif(repeats, min = 0, max = 1)
X <- (b - a) * U + a #R is vectorized
5* mean(3*X^2 + 5*X)

## [1] 1063.222
```

Higher dimensional integrals

- Consider estimating the integral

$$\theta = \int_{a_k}^{b_k} \dots \int_{a_1}^{b_1} g\left(x_1, x_2, \dots, x_k\right) dx_1, dx_2, \dots, dx_k$$

- The same rules apply; We want to find a distribution that is defined on the space $(a_1, b_1) \times \cdots \times (a_k, b_k)$. Independent uniforms would do the trick!
- Consider estimating

$$\theta = \int_2^3 \int_5^6 3x^2 y dx dy = \mathrm{E}\left[3x^2 y\right]$$

where that expectation is with respect to $U(5, 6) \times U(2, 3)$.

```r
set.seed(1)
repeats <- 1e4
U1 <- runif(repeats, min = 0, max = 1)
X <- (6 - 5) * U + 5
U2 <- runif(repeats, min = 0, max = 1) # have to generate different U2
Y <- (3 - 2) * U + 2
mean(3*X^2 * Y)

## [1] 230.3351
```