Lecture 5: Statistics Basics in R

Cheng Lu

Overview

- Coin Filps
- Normal Distribution
- Other Distributions
- Generate Random Number
- Homework

Coin Flips

Before we go to generate random numbers lets make a function.

- Suppose you are flipping a fair coin 1000 times. Estimate the probability of heads after each flip.
- Make a 2-D graph for that probability.
- On your graph, x should be the number of flips and y should be the probability of heads.
- We are expecting the curve converges to 1/2 since the coin is fair.

Coin Flips

Analysis:

- We would keep generating a logical variable, using 1 for heads and 0 for tails.
- We also need a variable, recording how many heads we got after each iteration.
- Another vector can be used for recording 1000 probabilities.
- Functions will be used: sample(), plot()

Coin Flips

sample() function take samples of the specified size from the elements of x using either with or without replacement.

- sample(x, size, replace = FALSE, prob = NULL)

Example

```
# take samples from population with replacement
> sample(x=c(1,2,3),2,replace = T)
[1] 1 1
# take samples from population without replacement
> sample(x=c(1,2,3),2,replace = F)
[1] 1 3
```

Coin Flips

Example

```
No.heads <- 0
result.Vec <- NULL # vector contains all probabilities
for (flips in 1:1000){
  # x = c(1, 0), 1 means head, 0 means tail
  tmp <- sample(x = c(1, 0), size = 1, replace = T, prob = c(0.5, 0.5))
  # add tmp to number of head
  No.heads <- No.heads + tmp
  result.Vec <- c(result.Vec, No.heads/flips)
}

# tmp <- sample(x = c(0,1), size = 1000, replace = T, prob = c(0.5,0.5))
# No.heads <- cumsum(tmp)
# result.Vec <- No.heads/1:1000
```

Coin Flips

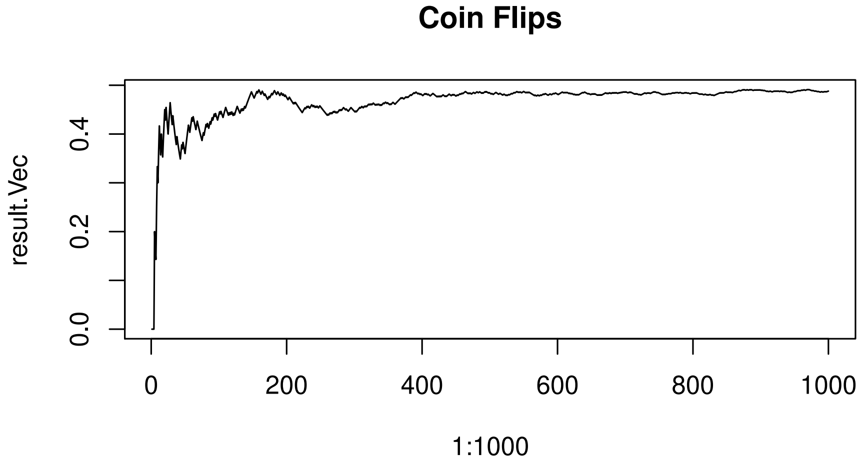
plot() is a generic function for plotting of R objects. For more details about the graphical parameter arguments.

- plot(x, y, type, col, main)
- x, y are the coordinates of points in the plot
- **type** will determine plot type. "p" for points, "l" for lines.
- **col** will determine the color for points or lines
- **main** will determine the title for this plot

Example

```
> plot(1:1000, result.Vec, type="l", main=c("Coin Flips"))
```

Coin Flips



Coin Flips

Example

```
coinFlip <- function(headProb) {
No.heads <- 0
result.Vec <- NULL
for (flips in 1:1000){
  tmp <- sample(x=c(1, 0), size=1, replace=T,prob=c(headProb, 1-headProb))
  No.heads <- No.heads + tmp
  result.Vec <- c(result.Vec, No.heads/flips)
}
plot(1:1000, result.Vec, type="l")
}
```

Now we have defined a function, to call it in the correct way you need to pass parameters with right type. In this case it has to be a real number between 0 and 1.

Example

```
coinFlip(0.5)
coinFlip(0.7)
coinFlip(0.9)
coinFlip(1)
```

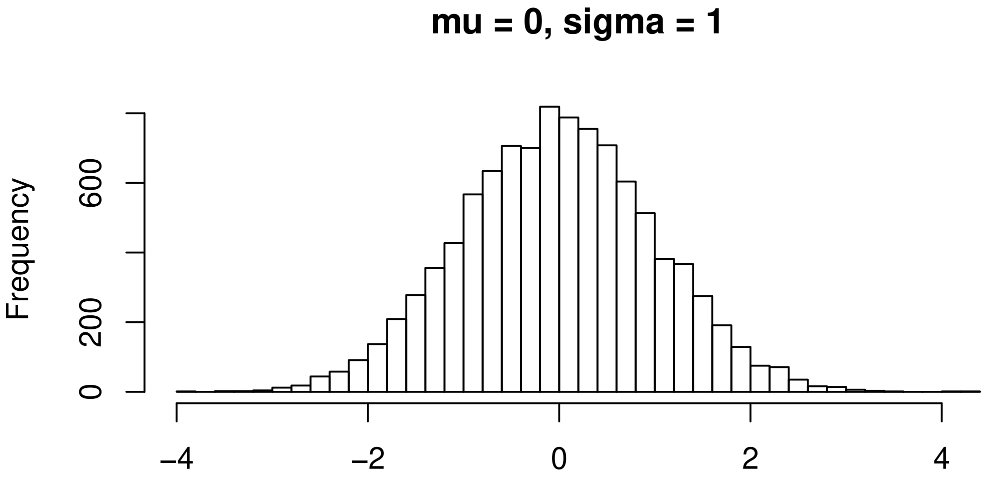
Normal Distribution

Now let's move on how to generate random variables.

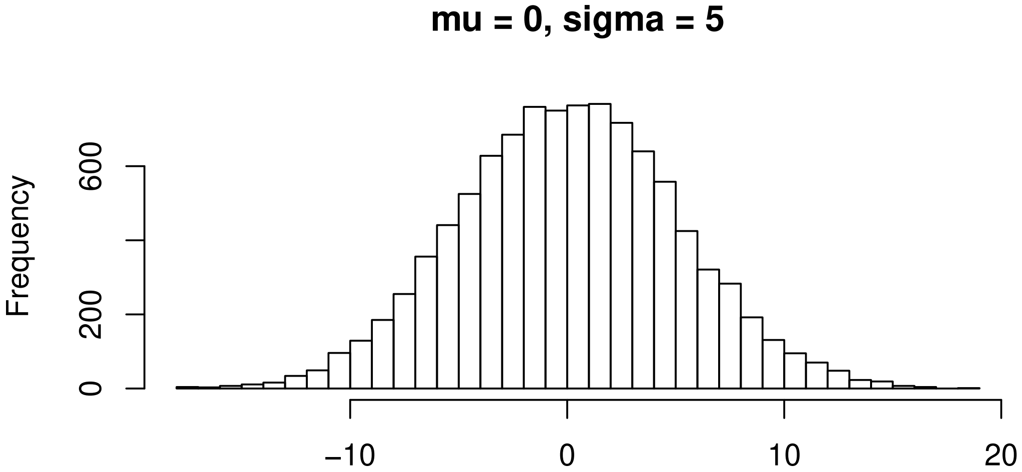
Example

```
> # rnorm(n = , mean = , sd = )
> x <- rnorm(n = 10000, mean = 0, sd = 1)
> hist(x) # histogram
> hist(x, nclass = 40, main = "mu = 0, sigma = 1")
> # another sigma
> x <- rnorm(n = 10000, mean = 0, sd = 5)
> hist(x, nclass = 40, main = "mu = 0, sigma = 5")
```

Normal Distribution



Normal Distribution



set.seed()

Any random numbers R gives you aren't really random. They're pseudo-random. To do this it needs some inputs, then random numbers can be generated by recursive formulas. The first input is called 'seed'.

Example

```
> set.seed(1)
> rnorm(5)
[1] -0.6264538 0.1836433 -0.8356286 1.5952808 0.3295078
> rnorm(5)
[1] -0.8204684 0.4874291 0.7383247 0.5757814 -0.3053884
> set.seed(1)
> rnorm(5)
[1] -0.6264538 0.1836433 -0.8356286 1.5952808 0.3295078
```

Normal Distribution

- rnorm: random number from normal distribution
- dnorm: probability density function of normal distribution

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- pnorm: cumulative distribution function of normal distribution

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)dt$$

- qnorm: quantile function of normal distribution

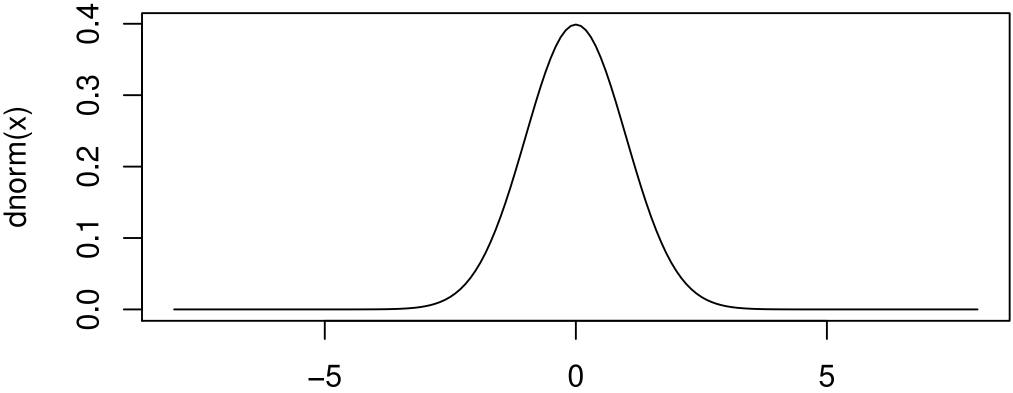
$$x = F^{-1}(p)$$

Normal Distribution

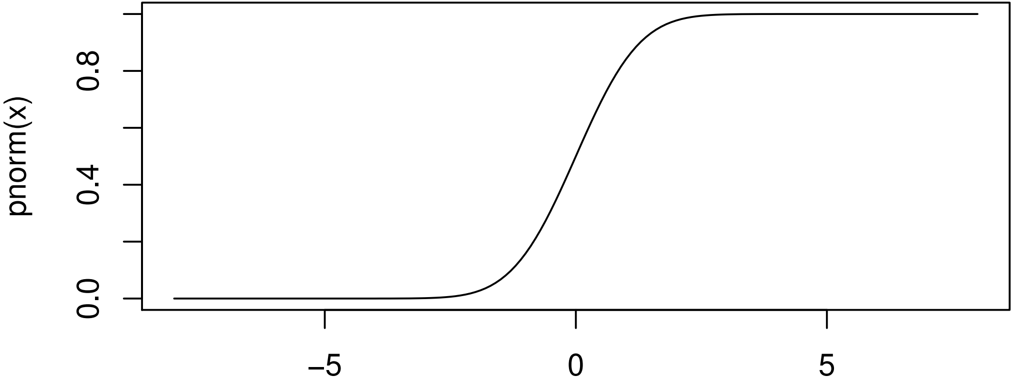
Example

```
> # Density function
> dnorm(x = 0) # f(0) = 1/sqrt(2*pi), mean = 0, sd = 1
[1] 0.3989423
> dnorm(x = 1) # f(1) = 1/sqrt(2*pi)*exp(-0.5)
[1] 0.2419707
> # cumulative distribution function
> pnorm(q = 0) # F(0) = P(X <= 0) = 0.5
[1] 0.5
> pnorm(q = 5)
[1] 0.9999997
```

Normal Distribution



Normal Distribution



Other Distributions

- rt() - random number from t distribution
- rpois() - random number from Poisson distribution
- runif() - random number from uniform distribution
- rexp() - random number from exponential distribution
- rgeom() - random number from geometric distribution
- rbinom() - random number from binomial distribution

Guess what is **dt()**, and what is **qpois()**? Check your answer by typing **?dt** and **?qpois**.

Other Distributions

Example

```
> x <- rpois(1000, lambda = 2)
> hist(x, nclass = 40)
>
> x <- rexp(1000)
> hist(x, nclass = 40)
>
> x <- rt(1000, df = 10)
> hist(x, nclass = 40)
```

Generate Random Number

Linear Congruential Generator (LCG) is an algorithm which generates pseudo-random number. The generator satisfies the following recursive relation

$$X_{n+1} = (a * X_n + b) \ mod \ m$$

where m is modulus, a is multiplier, b is increment, and $0 \leq X_0 < m$ is seed. e.g. Park and Miller suggests $m = 2^{31} - 1, a = 7^5, b = 0$.

Since each element of vector X is a number between 0 and m, then X/m is a vector of numbers between 0 and 1. We can use the algorithm generate pseudo-random number from uniform distribution.

Generate Random Number

Example

```
> seed <- 1 # let the seed be 1
> n <- 5 # quantity of random numbers
> m <- 2 ^ 31 - 1
> a <- 7 ^ 5
> b <- 0
> x <- rep(NA, n)
> x[1] <- (a * seed + b) %% m
> for(i in 1:(n-1)){
+   x[i + 1] <- (a * x[i] + b) %% m
+ }
> seed <- x[n] # change the seed
> x/m
[1] 7.826369e-06 1.315378e-01 7.556053e-01 4.586501e-01 5.327672e-01
```

Generate Random Number

Example

```
> # generate random number from uniform distribution
> seed <- 1 # let the seed be 1 in global environment
> rnd <- function(n){
+   m <- 2 ^ 31 - 1
+   a <- 7 ^ 5
+   b <- 0
+   x <- rep(NA, n)
+   x[1] <- (a * seed + b) %% m
+   for(i in 1:(n-1)){
+     x[i + 1] <- (a * x[i] + b) %% m
+   }
+   seed <<- x[n] # change the seed in global environment
+   return(x/m)
+ }
```

Generate Random Number

Example

```
> rnd(5) # the first few numbers are usually bad, we can discard them
[1] 7.826369e-06 1.315378e-01 7.556053e-01 4.586501e-01 5.327672e-01
> rnd(5)
[1] 0.21895919 0.04704462 0.67886472 0.67929641 0.93469290
> seed <- 1
> rnd(5) # same number for seed = 1
[1] 7.826369e-06 1.315378e-01 7.556053e-01 4.586501e-01 5.327672e-01
> seed <- 100 # set seed be 100
> rnd(5)
[1] 0.0007826369 0.1537788143 0.5605322195 0.8650131923 0.2767237412
> seed <- as.numeric(Sys.time()) # set seed be the current system time
> rnd(5)
[1] 0.41016256 0.60217068 0.68257667 0.06610620 0.04691248
```

Generate Random Number

Proposition

Suppose U is a random variable follows uniform distribution in $[0, 1]$, then $X = F^{-1}(U)$ has cumulative distribution function (CDF) F.

In fact

$$P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x)$$

Example

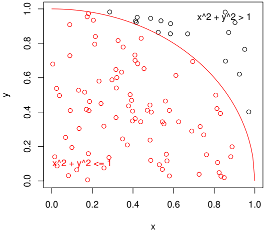
```
> U <- rnd(10000) # 10000 uniform r.v.
> # qnorm() is the quantile function (inverse CDF) of normal distribution
> X <- qnorm(U) # X follows normal distribution
> hist(X, nclass = 40) # histogram of normal distribution
> hist(rnorm(10000), 40) # equivalent to the above statement
> hist(qexp(rnd(10000), rate = 1), nclass = 40) # exponential distribution
```

Generate Random Number

Suppose we want to calculate the area of a unit circle (which is just π) by simulation. Consider the following figure, denote the area within the red line be the red area. Then the red area is the area of the $\frac{1}{4}$ unit circle. Suppose we draw N random points (which are 2 dimensional) in the unit square, let the points inside red area be red points, then

$$\frac{\pi}{4} = \text{area of } \frac{1}{4} \text{ unit circle} = \text{red area} = \frac{\text{red area}}{\text{area of square}} \approx \frac{\text{number of red points}}{N}$$

where N is the total number of points



Generate Random Number

Example

```
> N <- 10000
> x <- rnd(N)
> y <- rnd(N)
> #n_red <- 0 # number of red points
> #for (i in 1:N) {
> #  if(x[i]^2 + y[i]^2 <=1){
> #    n_red <- n_red + 1
> #  }
> #}
> n_red <- sum(x^2 + y^2 <= 1) # vectorized calculation
> area_quarter_circle <- n_red/N
> (Pi <- 4 * area_quarter_circle) # force output
[1] 3.132
```

Generate Random Number

We can also use system time as seed.

Example

```
> seed <- as.numeric(Sys.time()) # similar to runif(N)
> x <- rnd(N)
> y <- rnd(N)
> n_red <- sum(x^2 + y^2 <= 1)
> (Pi <- 4 * n_red/N)
[1] 3.1268
```

This is similar to

Example

```
> n_red <- sum(runif(N)^2 + runif(N)^2 <= 1)
> (Pi <- 4 * n_red/N)
[1] 3.126
```