MONTE CARLO METHODS

Probability distributions in R, Monte Carlo simulations

Probability distributions in R

Probability distributions in R

Universiti Kebangsaan Malaysia
The National University of Malaysia

- For most probability distributions, R has built-in functions to
  1. Calculate the pdf/pmf of a distribution.
  2. Calculate the cdf of the distribution.
  3. Calculate the quantile function of the distribution.
  4. Generate random numbers based on the distribution.

- All the functions related to above have the same "form" for all distributions.

Probability distributions in R

- For example, the functions related to a normal distribution are `dnorm()`, `pnorm()`, `qnorm()` and `rnorm()`.

- Functions related to a binomial distribution are `dbinom()`, `pbinom()`, `qbinom()` and `rbinom()`.

- Note:
  - "`d`" is for density. Used to calculate pdf/pmf.
  - "`p`" is for probability. Used to calculate cdf.
  - "`q`" is for quantile. Used to calculate quantile function given probability
  - "`r`" is for random. Used to generate random samples from the distribution.

Normal distribution

- Calculate pdf:
  - `dnorm(x, mean = 0, sd = 1, log = FALSE)`
- Calculate cdf:
  - `pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`
- Calculate quantile function:
  - `qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`
- Generate random numbers:
  - `rnorm(n, mean = 0, sd = 1)`

Example

- Let X be a random variable distributed following a normal distribution with mean 3 and standard deviation 1.5. Calculate the following probabilities.
  - $P(X < 2.54)$
  - $P(X > 3.5)$
  - $P(2.54 \leq X < 3.5)$

- Find the value of x such that
  - $P(X \leq x) = 0.32$
  - $P(X > x) = 0.75$

Example

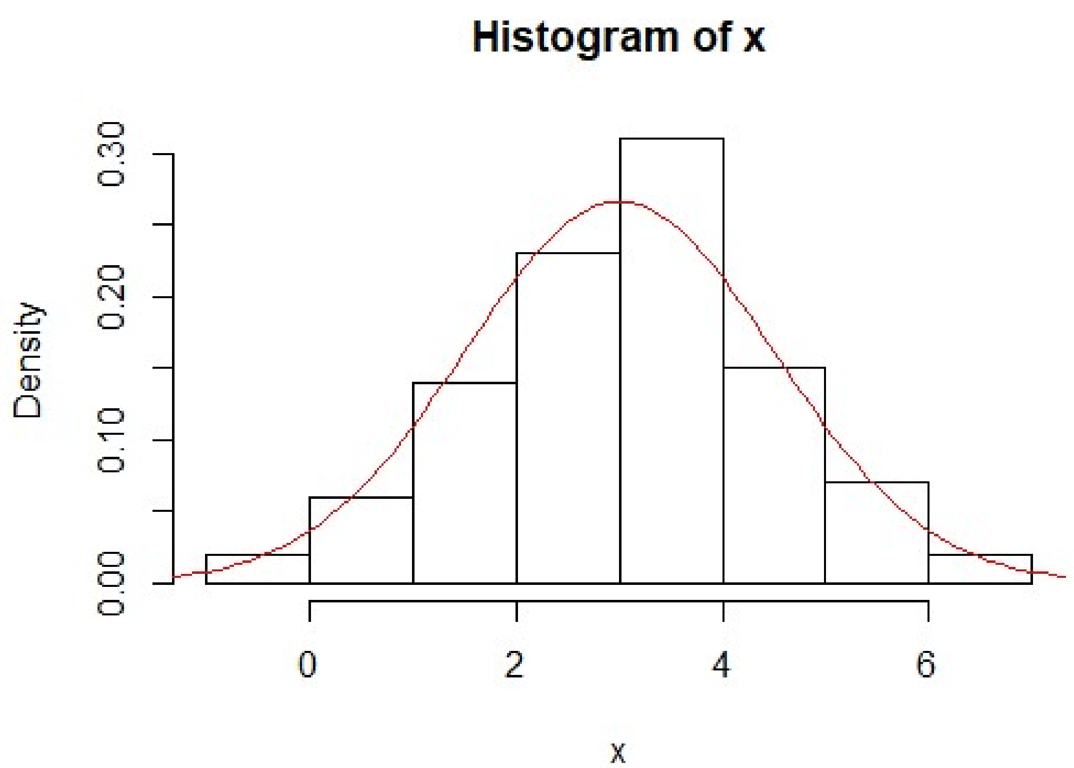
```
> pnorm(2.54, mean=3, sd=1.5)
[1] 0.3795486
> pnorm(3.5, mean=3, sd=1.5, lower.tail=FALSE)
[1] 0.3694413
> pnorm(3.5, mean=3, sd=1.5) - pnorm(2.54, mean=3,
        sd=1.5)
[1] 0.2510101
> qnorm(0.32, mean=3, sd=1.5)
[1] 2.298452
> qnorm(0.75, mean=3, sd=1.5, lower.tail=FALSE)
[1] 1.988265
```

Example

- Let X be a random variable distributed following a normal distribution with mean 3 and standard deviation 1.5.
  - Generate 100 samples of X.
  - Draw a histogram of the generated samples where the height of the histogram represent the probability of each bin.
  - Add a line showing the pdf of X onto the drawn histogram.

Example

```
> x <- rnorm(100, mean=3, sd=1.5)
> hist(x, freq=FALSE)
> curve(dnorm(x, mean=3, sd=1.5), from=-2, to=8,
        col="red", add=TRUE)
```



Common probability distributions in R

Distributions	Functions			
Beta	pbeta	qbeta	dbeta	rbeta
Binomial	pbinom	qbinom	dbinom	rbinom
Chi-Square	pchisq	qchisq	dchisq	rchisq
Exponential	pexp	qexp	dexp	rexp
F	pf	qf	df	rf
Gamma	pgamma	qgamma	dgamma	rgamma
Log Normal	plnorm	qlnorm	dlnorm	rlnorm
Normal	pnorm	qnorm	dnorm	rnorm
Poisson	ppois	qpois	dpois	rpois
Student t	pt	qt	dt	rt
Uniform	punif	qunif	dunif	runif

Sampling from a vector

- The `sample()` function can be used to randomly pick a sample (whether it be character, numbers, etc) from a given vector.

- Syntax: `sample(x, size, replace = FALSE, prob = NULL)`
  - `x`: a vector from which the elements are to choose from
  - `size`: how many samples we want
  - `replace`: should the sampling done with replacement? (can a sample be picked more than once?)
  - `prob`: a vector of the probability of picking each elements of `x`.

Example

```
> sample(c(1,2,3), 5, replace=TRUE)
[1] 3 1 2 3 1
> sample(c("A", "B", "C", "D"), 2, prob = c(0.1, 0.2,
0.5, 0.2))
[1] "B" "C"
```

- In the first example above, we want to get 5 values from the vector (1, 2, 3) where each element of the vector has the same probability of being picked. We set `replace=TRUE` because an element can be picked more than once.

- In the second example above, we want to get two values from the vector (A, B, C, D), where the probability of A being picked is 0.1, B is 0.2, C is 0.5, and D is 0.2. Since we do not specify replace argument, by default, `replace` is `FALSE` and a value cannot be picked more than once.

Random number generating process

- Note that the numbers generated in computer are not exactly random. They are “pseudorandom”.
- But they are good enough for most applications.
- You can set the “seed” of your randomly generated numbers using `set.seed()` function

Example

```
> set.seed(1)
> rnorm(10)
 [1] -0.6264538  0.1836433 -0.8356286  1.5952808
 [5]  0.3295078 -0.8204684  0.4874291  0.7383247
 [9]  0.5757814 -0.3053884
> set.seed(1)
> rnorm(10)
 [1] -0.6264538  0.1836433 -0.8356286  1.5952808
 [5]  0.3295078 -0.8204684  0.4874291  0.7383247
 [9]  0.5757814 -0.3053884
```

- Notice the randomly generated numbers are exactly the same.

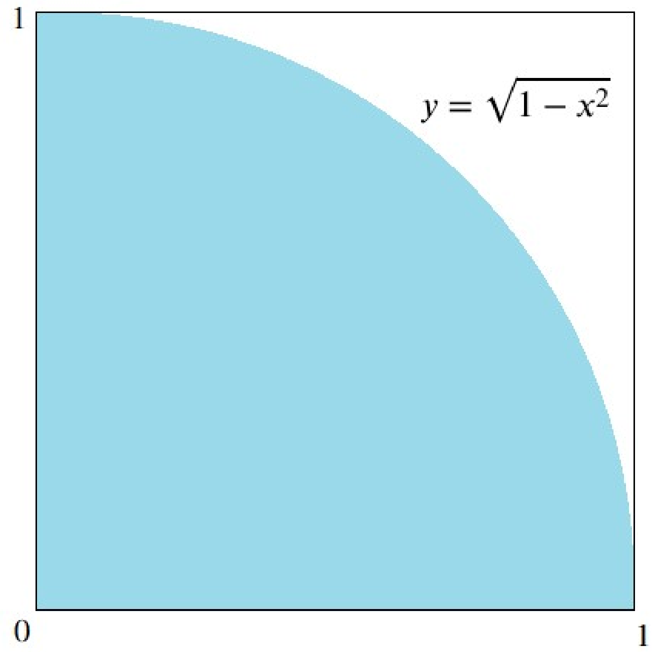
Monte Carlo methods

What are Monte Carlo methods?

- Basically, solving problems by simulations.
  - Estimating solution to problems cannot be solved analytically.
  - Testing methods or algorithms on simulated data.
  - Mimics experimental lab.

- Why use simulations?
  - For problems that cannot be solved analytically, sometimes simulations give easier solution
  - Able to perform analysis on a “perfect” data.

Example 1: Estimating the value of π



- Area of a unit square $= 1$
- Area of the quadrant $= \dfrac{\pi}{4}$
- Proportion of the area of quadrant over the area of square $= \dfrac{\pi}{4}$

Example 1: Estimating the value of π

- Algorithm:
  - Simulate the X and Y values for the coordinates of points in the unit square ($X \sim \text{Unif}(0,1)$ and $Y \sim \text{Unif}(0,1)$).
  - Count the proportion of simulated data that is inside the quadrant.
  - Estimate the value of π by

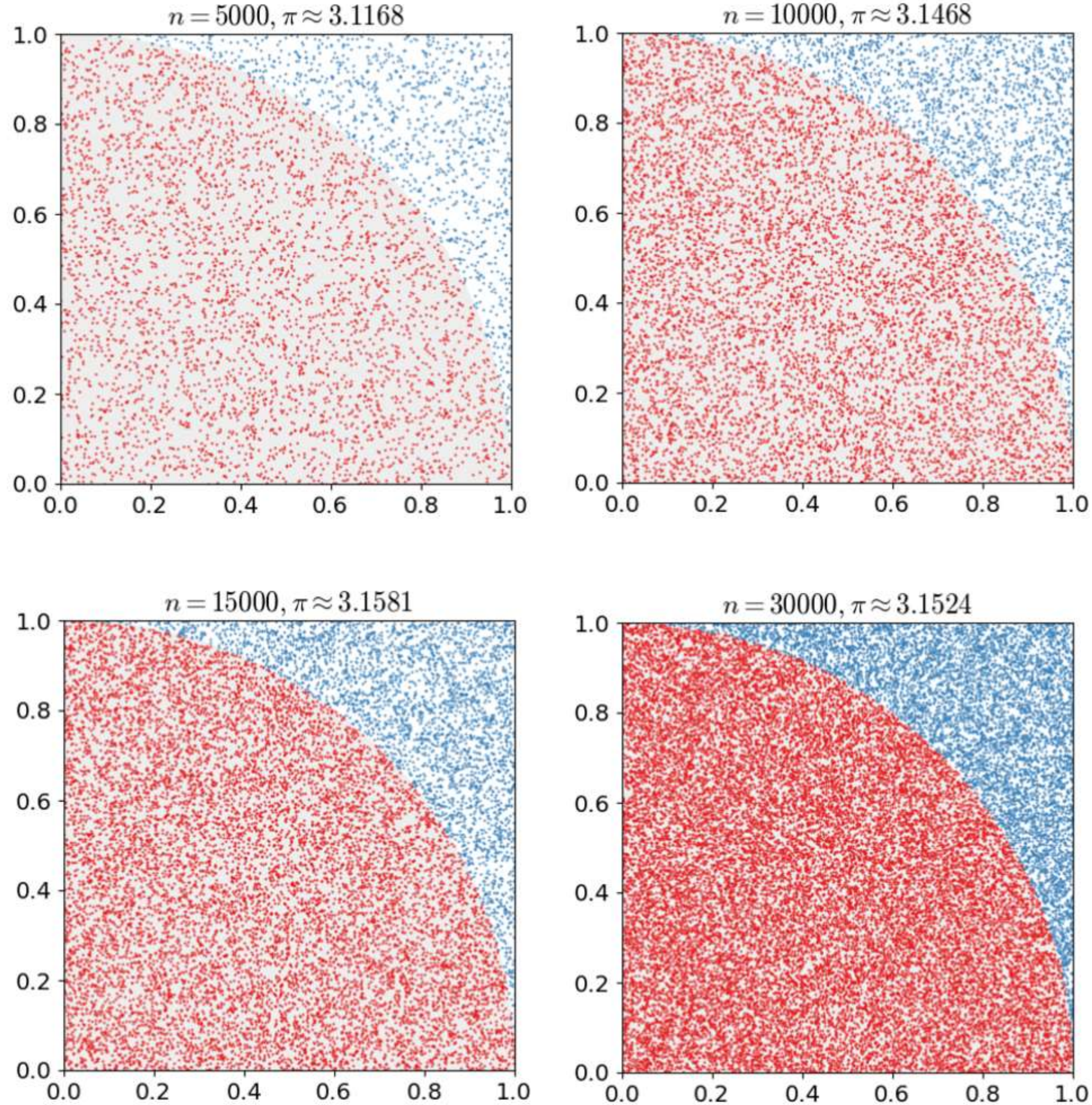
$$\pi \approx 4 \times (\text{proportion data inside quadrant})$$

Example 1: Estimating the value of π

```r
### Estimating pi

estimate_pi <- function(n){
    x <- runif(n=n, min=0, max=1)
    y <- runif(n=n, min=0, max=1)
    number_inside <- sum(y <= sqrt(1-x^2))
    proportion <- number_inside/n
    pi_est <- 4*proportion
    return(pi_est)
}
```

```r
> estimate_pi(100)
[1] 3.2
> estimate_pi(1000)
[1] 3.16
> estimate_pi(1e6)
[1] 3.140148
> pi
[1] 3.141593
```


$n = 5000, \pi \approx 3.1168$
$n = 10000, \pi \approx 3.1468$
$n = 15000, \pi \approx 3.1581$
$n = 30000, \pi \approx 3.1524$
0.0
0.2
0.4
0.6
0.8
1.0

Example 2: Inverse transform sampling

- Suppose we have a random variable X with cdf $F(x)$.

- Then it can be shown that the random variable $U = F(X)$ is uniformly distributed between 0 and 1.

- Theorem: Let $U \sim \mathrm{Unif}(0,1)$ and F be a cdf. Then $F^{-1}(U)$ has cdf F.

- We can use this property to randomly generate the variable X.

Example 2: Inverse transform sampling

- Example: Suppose

$$f(x) = 6x^2(1 - x^3), x \in [0,1]$$

$$F(x) = 1 - (1 - x^3)^2$$

- Algorithm:
  - Randomly generate U from Unif(0,1).
  - Calculate $X = F^{-1}(U)$

- First, we have to calculate $F^{-1}(U)$.

$$F^{-1}(U) = \left(1 - \sqrt{1 - U}\right)^{\frac{1}{3}}$$

Example 2: Inverse transform sampling

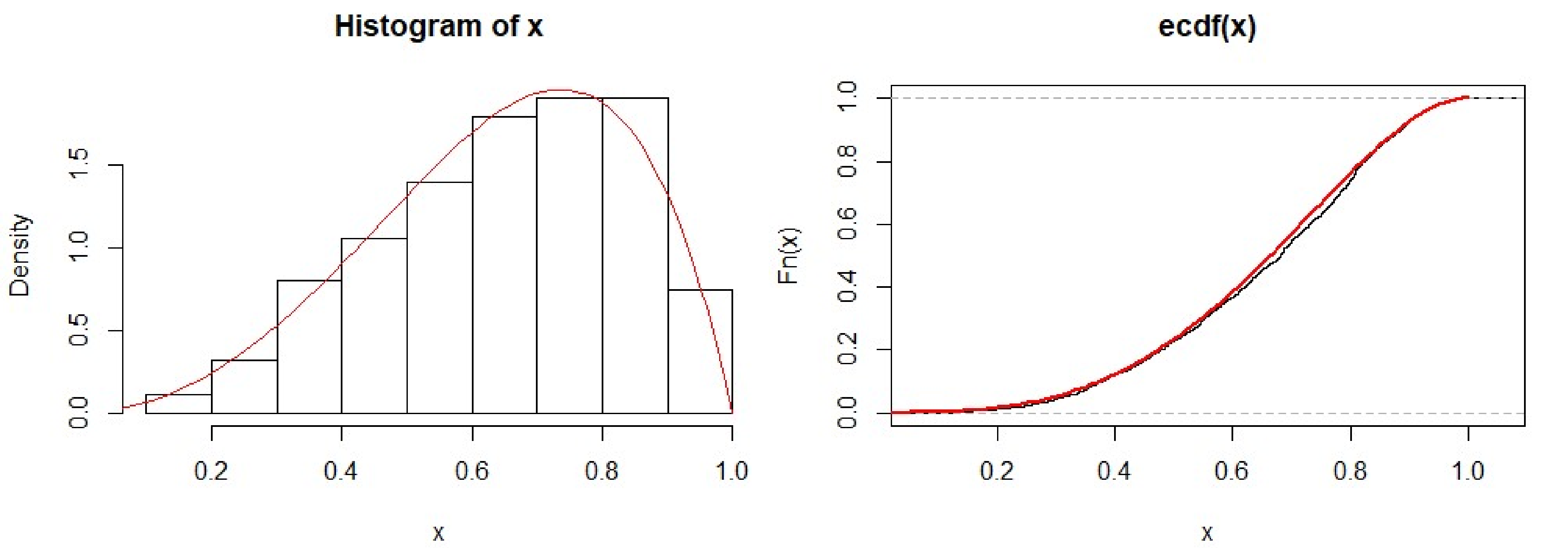
```
### Inverse transform sampling

generate_x <- function(n){
    u <- runif(n, min=0, max=1)
    x <- (1-sqrt(1-u))^(1/3)
    return(x)
}

# Compare histogram and pdf
x <- generate_x(1000)
hist(x, freq=FALSE)
curve(6*x^2*(1-x^3), from=0, to=1, add=TRUE,
      col="red")

# Compare empirical cdf and cdf
plot(ecdf(x))
curve(1-(1-x^3)^2, from=0, to=1, add=TRUE, col="red",
      lwd=2)
```

Example 2: Inverse transform sampling



Functions & descriptions

Function	Description
`dnorm()`	Used to calculate pdf of normal distribution.
`pnorm()`	Used to calculate cdf of normal distribution.
`qnorm()`	Used to calculate quantile function of normal distribution.
`rnorm()`	Used to generate random samples from the normal distribution.
`sample()`	Used to get samples from a given list

- Note: Other distributions are also available and follow similar notations for the functions – `d`, `p`, `q`, `r`