Statistical Computing

Master Data Science

Winter Semester 2019/20

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Prof. Tim Downie

Workshop 9 — Simulation, Sampling and the Central Limit Theorem *(with solutions)*

Introduction

Simulation, in the statistical sense, means using computer generated *pseudo-random numbers* to mimic a real event or a real data set. This is a whole research area in its own right; in one workshop we are only able to scratch the surface of this subject.

Contents

- The first section gives an introduction to simulation and how one simulates random numbers from the uniform distribution in R.
- Section 2 contains exercises which demonstrate the central limit theorem by using simulation.
- A deeper look into simulation is given in Section 3.
- Two theory exercises are given in section 4

It is planned that you work through the first two sections in the workshop and sections 3 and 4 at home.

1 Simulation

1.1 Simulating a uniform $U[0,1]$ random number in R

A *random number generator* (RNG) simulates a random number which comes from the standard uniform distribution $U[0,1]$. The function in R which does this is called `runif` (pronounced 'r-unif'). The `r` means random.

```
> runif(1)
[1] 0.2088658
```

Gives a value between 0 and 1. Your output will be different from the example because it is *random*.

To simulate a sample $x_1, x_2, \dots x_n$of size n

```
> n<-10
> runif(n)
[1] 0.2347135 0.9871401 0.7752655 0.7920699 0.3962449 0.0443566
[7] 0.3701889 0.2743758 0.3264861 0.5437145
```

We need to consider what is meant by "simulated from a distribution" actually means.

1.2 Empirical distribution function

Suppose $X \sim U[0,1]$ is a standard uniform distributed random variable. X takes values between 0 and 1, and each value in that range is equally likely. X has the *cumulative distribution function* (cdf)

$$F_X(x) = P(X \leqslant x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } 0 \leqslant x \leqslant 1 \\ 1 & \text{for } x > 1 \end{cases}$$

And the density function (pdf)

$$f_X(x) = F'_X(x) = \begin{cases} 1 & \text{for } 0 \leqslant x \leqslant 1 \\ 0 & \text{otherwise.} \end{cases}$$

In order to *simulate* from this distribution, we implement a RNG, which gives values $x_1, x_2, \dots x_n$. The numbers are not actually random, because they are produced by a computer program (hence *pseudo-random numbers*), but we can consider the output as random provided the RNG satisfies the following conditions:

i) The proportion of observations $\leqslant x$ is approximately equal to $F_X(x)$ for all $x \in \mathbb{R}$.

ii) Predicting x_{n+1} is not easier, when we know the values of $x_1, x_2, \dots x_n$. This mimics the property that all X_is are independent.

Lets look at point i) in more detail.

The **empirical distribution function** (ECDF) $F_n(x)$ is defined by the observed data $x_1, x_2, \dots x_n$ with sample size n.

The empirical distribution function $F_n(x)$ is the proportion of the values $x_1, x_2, \dots x_n$, which are less than or equal to x. As a mathematical formula, this is

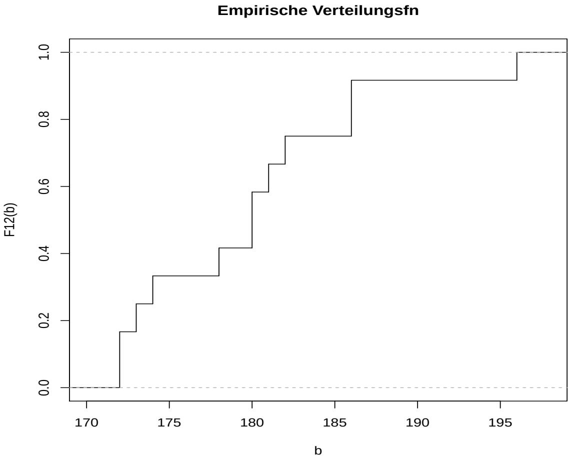
$$F_n(x) := \frac{\#\left\{x_{(i)} | x_{(i)} \leqslant x\right\}}{n} \qquad (\#\text{ means "the number of"}).$$

Example: for the following data $172, 172, 173, 174, 178, 180, 180, 181, 182, 186, 186, 196$

$$F_{12}(175) = \text{Proportion of values} \leqslant 175 = \frac{(\#x_i \leqslant 175)}{12} = \frac{4}{12} = 0.3\overline{3}$$

$$F_{12}(183) = \text{Proportion of values} \leqslant 183 = \frac{(\#x_i \leqslant 183)}{12} = 0.75$$

A plot of $F_{12}(x)$ against x gives the following step function.



The Glivenko–Cantelli Theorem states for a independent and identically distributed random sample $x_1, x_2, \ldots x_n$, arising from a cumulative distribution function $F_X(x)$

$$\lim_{n\to\infty} P\left(\sup_{x\in\mathbb{R}} \left|\widehat{F}_n(x) - F_X(x)\right| \leqslant \varepsilon\right) = 1 \qquad \text{for an arbitrarily small } \varepsilon > 0\,.$$

This theorem says that the empirical distribution function tends to the underlying cdf $F_X(x)$ as n tends to infinity.

If we have a good RNG then $F_n(x) \to x$ for $0 \leqslant x \leqslant 1$ as $n \to \infty$

Lets take a look if this happens with `runif`:

```
> n<-10
> xsamp<-runif(n)
> curve(punif(x),0,1)
> plot(ecdf(xsamp),verticals=TRUE,do.points=FALSE,add=TRUE)
```

‘p-unif’ is the R function for the cdf of the $U[0,1]$ distribution.
Repeat the above code for $n = 50, 100$ and 200. The step function is clearly tending towards the ideal function.

1.3 Simulating from the general uniform distribution

We would like to be able to generate random numbers from other distributions as well. There are many varied techniques to do this which are not within the scope of this course, but we will look at a simple example of modifying a $U[0,1]$ distributed random variable.

If X has a general uniform distribution with minimum value a and maximum value b: $X \sim U[a,b]$, the cdf is

$$F_X(x) = P(X \leqslant x) = \frac{x-a}{b-a}, \text{ for } a \leqslant x \leqslant b$$

As long as we have a good method of generating $U \sim U[0,1]$ from a standard uniform distribution then we can transform U into any uniform random variable by "stretching by the width of the interval $[a,b]$ and adding a".

$$X = (b-a)U + a$$

We can show mathematically that this transformation gives the right distribution using the known cdf of U:

$$\begin{aligned} P(X \leqslant x) &= P\Big((b-a)U + a \leqslant x\Big) \\ &= P\left(U \leqslant \frac{x-a}{b-a}\right) \\ &= \frac{x-a}{b-a} = F_X(x) \qquad \square \end{aligned}$$

The R-functions `runif` and `punif` take two optional arguments as the a and b parameters:

```
> n<-10
> xsamp<-runif(n)*25-5
> curve(punif(x,-5,20),-5,20)
> plot(ecdf(xsamp),verticals=TRUE,do.points=FALSE,add=TRUE)
```

Again increase the value of n.

Usually you would generate such a sample using `runif(n,a,b)`. Note that for a $U[0,1]$ distribution you can just use `runif(n)` as the default values of a and b are 0 and 1.

1.4 Continuous distributions in R

R has built in functions which generate random numbers from a wide variety of distributions, which follow the same syntax
`r`+dist.name+`(n,`optional parameters`)`
E.g. `runif` and `pnorm`.

For each distribution there are four R functions available specified by their prefix:

`r`	random
`d`	density (pdf)
`p`	probability (cdf)
`q`	quantile

The normal distribution has two parameters; expectation μ and variance σ^2, notation: $X \sim N(\mu, \sigma^2)$. To generate 5 random normal $N(50, 5^2)$ numbers, use

```
> rnorm(5,100,5)
[1]  95.30848 103.22888 107.98139 101.84953  98.55983
```

Watch out: that the mathematical notation uses the variance σ^2 but R expects the **standard deviation**!

`dnorm` is the probability density function (pdf) $f_X(x)$ of the normal distribution

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \qquad x \in \mathbb{R}.$$

For $X \sim N(100, 5^2)$, the density at $x = 97$ is $f_X(97) = \frac{1}{\sqrt{50\pi}} \exp\left(-\frac{(x-100)^2}{50}\right) \approx 0.067$

```
> dnorm(97,100,5)
[1] 0.06664492
```

`pnorm` is the cumulative density function (cdf) $F_X(x) = P(X \leqslant x)$ of the normal distribution.

```
> pnorm(97,100,5)
[1] 0.2742531
```

`qnorm` is the inverse function of `pnorm`. The p-quantile q_p is defined so that $F_X(q_p) = P(X \leqslant q_p) = p$. From the previous example we can see that the 0.2742531-quantile of the $N(100, 5^2)$ distribution is 97

```
> qnorm(0.2742531,100,5)
[1] 97
> qnorm(0.975,100,5)
[1] 109.7998
```

Other common continuous distributions

Distribution	Notation	R Abbreviation	Example	Support
Exponential	$X \sim \text{Exp}(\lambda)$	`exp`	`dexp(4,1.5)`	$[0, \infty)$
Chi-squared	$X \sim \chi^2_m$	`chisq`	`dchisq(2.5,4)`	$[0, \infty)$
Gamma	$X \sim \Gamma(\alpha, \lambda)$	`gamma`	`dgamma(3.6,2.1,3.4)`	$[0, \infty)$
t	$X \sim t_m$	`t`	`dt(1.96,10)`	$(-\infty, \infty)$
f	$X \sim f_{m,n}$	`f`	`df(1.96,4,6)`	$[0, \infty)$

The support is the range of x for which $f(x) > 0$ and so corresponds to possible values for the random variable.

Exercise 1
For each of the continuous distributions in the previous table use the function curve to plot the density functions (pdf) and distribution functions (cdf) using the parameters given. You will need to find appropriate x-axis limits in each case.

1.5 Discrete distributions in R

Below are the most common *discrete* distributions. All three are integer valued distributions.

- **Binomial** $X \sim \text{Bin}(m, p)$: The model is: X is the number of successes from p independent Bernoulli trials each with a success probability of p. For example, a football player practising penalty shots takes $m = 10$ penalties. The success of each attempt has the probability $p = 0.9$. The number of successful penalties fits a $\text{Bin}(10, 0.9)$ distribution.

  The command to generate a generate sample of size n is

  ```
  > rbinom(n,m,p)
  ```

- **Poisson** $X \sim \text{Poi}(\lambda)$. The model is: X the number of events in a time interval. The parameter λ is the expectation of X: $\mathrm{E}(X) = \lambda$. For example X is the number of ”strong” earthquakes on a Pacific island has a $\text{Poi}(0.4)$ distribution: the expected number of Earthquakes per year is 0.4

  ```
  > rpois(n, lambda)
  ```

- **Geometric** $X \sim \text{Geo}_0(p)$: The model is the number of Bernoulli p trial failures before a success is observed. *The drunken porter example* A porter has 8 similar looking keys on his keyring and attempts to unlock an office door. Because he is drunk he forgets immediately which keys he has tried. The number of failed attempts to open the door has a $\text{Geo}_0(\frac{1}{8})$ distribution.

  Note: there are two versions of the geometric distribution. One corresponds to the number of attempts in order to obtain a success (minimum number of attempts is 1) and the number of failures *before* a success is obtained and has 0 as its smallest value.
  R uses the second if these Geo_0, with: $X \in \{0, 1, 2, \ldots\}$.

  ```
  > rgeom(n, p)
  ```

Examples:

```
> dbinom(3,4,0.2)
[1] 0.0256
> pbinom(3,4,0.2)
[1] 0.9984
> qbinom(0.5,4,0.2)
[1] 1
> dpois(3,1.5)
[1] 0.1255107
> dgeom(0,0.9)
[1] 0.9
```

Important: discrete distributions do not have a density function, instead they have a probability mass function (pmf) . However to keep the syntax consistent with continuous distributions R uses `d` as the prefix for the pmf in discrete distributions.

Exercise 2
For each of the discrete distributions above plot the probability functions (pmf)using the parameters in the scenarios above. You will need to specify appropriate x-axis values each case. For example

```
> plot(0:4, dbinom(0:4,?,?),type="h")
> points(0:4,dbinom(0:4,?,?),pch=16)
```

Also plot the distribution functions (cdf) for each distribution.

```
> plot(-1:5,  ?binom(-1:5,?,?),type="s")
```

2 Central Limit Theorem: Exercises

Sections 3 and 4 continue the theme of simulation, but these can be left until later. Below are some exercises simulating the Central Limit Theorem. You should refer to the lecture notes to help answer the following exercises.

Exercise 3 A random sample with normally distributed values
Let $X_1, X_2, \ldots, X_{10}$ be a sample of $n = 10$ random numbers from a $N(50, 5^2)$ distribution.

(a) What is the distribution of $\overline{X}$?

(b) Use `pnorm` to calculate the probability that $P(\overline{X} \leqslant 51)$

(c) Simulate $X_1, X_2, \ldots, X_{10}$ using `rnorm` and calculate the mean value `xmean`.

(d) Repeat part (c) calculating 100 “replicates” of `xmean`. Do this by using a `for` loop and store the value of `xmean` in a vector of length 100 called `xsamp`. Each element is the mean of ten $N(50, 5^2)$ random numbers.

(e) Calculate the proportion of these 100 replicates that are $\leqslant 51$ and compare this with your theoretical answer in part

(f) Plot the empirical distribution function of `xmean` and compare it with the theoretical cdf.

```
> curve(pnorm(x,???,???),???,???,col=2)
> plot(ecdf(xsamp),verticals=TRUE,do.points=FALSE,add=TRUE)
```

Exercise 4 A random sample from the standard uniform distribution

(a) Repeat Exercise 3 parts (d) and (f) but using a $U[0,1]$ as the population distribution.

(b) Consider now a sample size of $n = 30$. Repeat all parts of Exercise 1 using the central limit theorem for the distribution of $\overline{X}$.
Hints $E(X) = 0.5$ and $\mathrm{Var}(X) = \sigma^2 = \frac{1}{12}$. For the probability questions use $P(\overline{X} \leqslant 0.55)$.

Exercise 5 Binomial distribution

You will see that, for large m, a Binomial(m, p) distribution tends to a normal distribution with expectation $\mu = mp$ and variance $\sigma^2 = mp(1-p)$

(a) Simulate $n = 100$ replications of a $B(m, 0.25)$ distribution with $m = 10$.

(b) Plot the resulting ecdf and the cdf of the $N(\mu, \sigma^2)$ distribution.

(c) Repeat for $m = 20, 50, 100$ and 200.

3 Homework Reading: Types of random number generator

By default R uses a "very good" RNG called *Mersenne-Twister*. The personal computers in the 1980's used notoriously bad RNGs, typically this was a *congruential generator*. Given a, b, m and y_0, a sequence of N pseudo random numbers can be obtained using

$$y_n = ay_{n-1} + b \,(\text{mod } m)^1 \quad \text{for } n = 1, \ldots, N.$$

Each y_n is divided by m to give a pseudo random number from the $U[0,1]$ distribution. The value y_0 is called the "seed" as it is used as a starting point of the sequence.

The quality of a congruential generator is highly dependent on the choice of a, b and m. The early computers were poor in the sense that

- The *period* of the RNG could be small. If N exceeds the period then the sequence $y_0, y_1, \ldots$ eventually starts repeating itself and the numbers are now predictable.
- Knowing the last few values can help to predict the next value, ie. the sequence is not independent.

[1] mod= modulo operator. $n(\text{mod } m)$ is the remainder when dividing n by m. E.g. $8 \text{ (mod 3)} = 2$

The Mersenne-Twister RNG has a period of $2^{19937} - 1$, which is about 10^{6000}. If you ask R to calculate this number it returns `Inf`!

Several types of RNG are available in R, all of which are more advanced than a congruential generators, but rely on broadly similar principles.

To specify another RNG use the function `RNGkind` with the name of the RNG you want to use. For example

```
> RNGkind("Wichmann-Hill")
```

The names of all available RNGs can be found using

```
> help(RNGkind)
```

Usually there is no need to change the default setting. You can also force `runif` to use your own RNG, but you would only want to do this if you think you have a better RNG than any of the ones in R!

3.1 Random number generator seed

A potential problem with simulation, is that every time you run your code you get different results. Sometimes this is desirable, but not always. Suppose you are writing up your Master's thesis which includes some simulated results. Your supervisor thinks you need to calculate just one more statistic. If you have not set the seed in advance, you will be unable to get exactly the same simulation results to replicate your results. You will need to update all of your existing results just to calculate one more number.

The answer is to specify an arbitrary starting point for the random number generator. If you need to recreate the same sequence then re-set the seed to the original value.

In the congruential generator in the last section y_0 is the seed. Most advanced RNGs use several numbers as the seed. We rarely need to know the exact form of the seed as the function `set.seed()`, does the hard work for us. `set.seed()` takes a single integer as its argument and uses this number to set the seed required for the random number generator. You should choose an arbitrary number such as

```
> set.seed(2017)
```

An example of this was given in Workshop 1.

4 Homework Questions:

Exercise 6 Congruential Generator

A trivial congruential generator is

$$y_n = 5y_{n-1} + 1 \bmod 16,$$

If we start with the value $y_0 = 9$ we get

$$y_1 = 5 \cdot 9 + 1 \bmod 16 = 14$$

and $x_1 = \frac{14}{16} = 0.875$

(a) Calculate y_2, y_3 and y_4 and hence x_2, x_3, a_4 *7,4,5 and 0.4375, 0.2500, 0.3125*.

(b) $y_{15} = 8$ calculate y_{16} and y_{17} hence x_{16} and x_{17}. *9, 14 and 0.5625, 0.8750*

Notice that the last $y_{16} = y_0$ and that $y_{17} = y_1$. This congruential generator has a period of 16. Most RNGs repeat after a fixed number of observations have been generated, however good generators have a period much much greater than 16 (for example 7×10^{12}).

Exercise 7 Central limit theorem

The monthly rainfall for June in Berlin has expectation 75 mm and variance 85 mm^2, but the distribution is not known. Let $\overline{X}$ be the mean June rainfall for 50 years.

Note: take care with this exercise not to mix up the variance and the standard deviation.

(a) Use the central limit theorem to show that $\overline{X}$ has a $N(75, \sigma^2)$ Distribution, specifying σ^2. $\sigma^2 = \frac{85}{50}$

(b) Transform the probability $P(74 \leqslant \overline{X} \leqslant 76)$ into a probability of the form $P(-k \leqslant Z \leqslant k)$, where Z has a standard normal distribution (you need to find the value of k).

$$P\left(\frac{74-75}{\sqrt{\frac{85}{50}}} \leqslant Z \leqslant \frac{76-75}{\sqrt{\frac{85}{50}}}\right) = P(-0.767 \leqslant Z \leqslant 0.767)$$

(c) Use `pnorm(k)` and `pnorm(-k)` in R to calculate the probability in part (b).

$0.7784592 - 0.2215408 = 0.557$