Computational Statistics

Lecture notes

Mauricio B. García Tec

Instituto Tecnológico Autónomo de México

September 23, 2015

Contents

Chapter 1

Introduction

Computational statistics is a discipline lying within the interface of statistics and numerical analysis. The idea is to develop algorithms to compute, estimate or interact with statistical or mathematical objects of interest that would be hard to deal with analytically.

This course is not about learning how to programme or how to use a particular language. For expository purposes, however, we will give several examples using the software `R`. I assume the reader has some familiarity with this language, so in the examples I will often comment about efficient and advanced ways of using the tools available in `R` for implementing our algorithms. However, the reader can apply the algorithms we will study here on the programming language of their choice.

The core mechanism behind all the methods we shall cover is the use of randomness to our advantage. In the next chapter, we will learn how to simulate random numbers, and what we precisely mean by them. But suppose for the moment, that we know how to generate a sequence of *independent* random numbers that are taken uniformly from $(0,1)$[1]. The following example will show how to use these sequences to compute an approximation of the constant π.

Example 1.1 (Our first computation) The idea is the following: if you draw a quarter circle of radius 1 in the square $[0,1]\times[0,1]$, the area enclosed by this quarter circle is $\pi/4$. If we were to produce a sequence of random numbers uniformly distributed over this square, the fraction of points that would lie below the curve $f(x)=\sqrt{1-x^2}$ should be approximately $\pi/4$ after enough points have were considered.

In the following example we generate two sequences of uniform random numbers in $(0,1)$ using the `R` function `runif` and compute the fraction of points that lie below the quarter circle. Our final estimate of π is given by multipliying this fraction by four.

[1] Since we are working with computers, there is actually no such thing as a continuous uniform distribution in $(0,1)$; the machine precision constraints imply that there is actually only a finite quantity of numbers in $(0,1)$

```r
set.seed(110104) # Good practice!
nsim <- 1000 # The number of points we will generate
xcoord <- runif(nsim) # We store a random sequence for each coordinate
ycoord <- runif(nsim)
dist.orig <- sqrt(xcoord^2+ycoord^2) # Distance to (0,0)
hits <- dist.orig < 1 # Number of points lying below the circle of radius 1
area <- sum(hits)/nsim # Fraction below
4*area # Final Estimate

## [1] 3.112
```

Figure 1.1 shows an illustration of the method. The command to produce the figure is shown below.

```r
qplot(xcoord, ycoord, colour=hits)
```

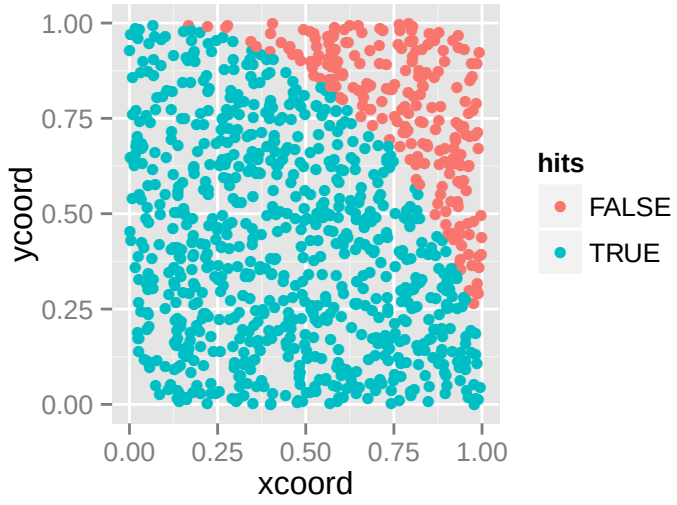

Figure 1.1: Hit and miss method for computing π

Later in this course we will look for ways of making these computations more efficient.■

Let us now give a different example to motivate our study of Computational Statistics. In fact, an example like this is what motivated the Monte-Carlo method, which will be one of the topics covered in this course. We will see a little bit of its history later, but these statistical algorithms were one of the first uses of modern computers dating back to the times of the ENIAC. The method was said to be devised by the great mathematician Stanislaw Ulam when failing to analytically compute the probability of winning a Solitaire game [Eck87].

Example 1.2 (Computing a hard probability) Imagine the following game. We start with five dice. We throw them and then we count the number of sixes observed. If there are no sixes at all, then we have lost. If there is at least one six, we take away the dice having a six and throw again the remaining dice. Once more, if there are no sixes, we lose, otherwise, we keep throwing the remaining dice until either we run out of dice and win or we throw no sixes and lose. Analytically obtaining the exact probability of winning can be a tedious exercise. Instead, the following code in R simulates several runs of the game and calculates the fraction of games that are won. This fraction is an approximation of the probability of winning.

```
set.seed(110104)
simulate.game <- function(...){ # 1) Simulate one run of the game
  continue <- TRUE # Dummy to know if we must throw again
  win <- TRUE # Dummmy that will change to false if one throws no 6 at any point
  dice.left <- 6 # Dice to throw at next round
  while(continue){
    throw <- sample.int(6, dice.left, replace=TRUE)
    hits <- sum(throw == 6)
    if(hits==0){
      win <- FALSE
      continue <- FALSE
    } else{
      dice.left <- dice.left - hits
      if(dice.left==0) continue <- FALSE
    }
  }
  return(win)
}
compute.prob <- function(nsim){ # 2) Run several times and average
  results <- sapply(1:nsim, FUN=simulate.game)
  return(sum(results)/nsim)
}
c(compute.prob(10), compute.prob(100), compute.prob(1000),
compute.prob(1000), compute.prob(10000)) # Different approx. of the prob.

## [1] 0.0000 0.0000 0.0130 0.0200 0.0192
```

So the probability of winning is very low and its approximate value is 2%. Notice the use of the function `sapply` in the code. Using `sapply` instead of a usual `for` makes the computations in R a lot faster. The strategy is called vectorisation. ■

Chapter 2

Random Number Generation

As we have discussed. The idea of computational statistics is to use randomness to compute, estimate or sample from a statistical/mathematical object of interest. The first problem we must address is to define what do we actually mean by randomness and how we can obtain a sequence of numbers that qualify as random.

In the first section we will learn how to generate a sequence of uniformly distributed random numbers in $(0, 1)$. Later we shall see how we can sample other probability distributions from our uniform sequences.

There are basically two strategies to produce the desired sequences: *Pseudo-Random Number Generation* and *True-Random Number Generation*.

As the name suggests, pseudo-random numbers are not entirely random, but they are produced by following a mathematical 'recipe' that guarantees that they behave similar enough to how observations of a true-random numbers do. True-random number generation is a very interesting topic in itself. The usual way in which true random numbers can be obtained is by taking into advantage the randomness of some physical phenomena; some classical examples include tossing a coin or throwing dice. Modern methods utilise quantum effects, thermal noise in electric circuits, the timing of radioactive decay, etc. An example of a provider of true-random numbers is `random.org`.

Example 2.1 The package `random` can be used in `R` to connect to `random.org` and obtain true random numbers. The outcome is not precisely uniformly in $(0, 1)$. One must specify a minimal and a maximal integer value and the outcome is a number or a sequence of numbers sampled from this range of integers. Evidently, one can obtain a sequence in $(0, 1)$ by dividing into the upper range from which the numbers were taken from. Unfortunately, I cannot run these example for these notes. But open an `R` console and type them yourself and try to figure out how they work!

1) To use the package `random` one must use the following instructions.

```
library(random)
rseq <- randomNumbers(n=10, min=1, max=10000) # Generate 10 true random numbers
# in between min and max
```

2) If, instead, you would like to use the connectivity properties of `R` and build your own program that extracts numbers from `random.org`, you can use the following code, which uses the simplest form of `html` syntax for this purpose.

```
library(XML)
library(RCurl)
true.random <- function(nsim, digits = 5){
  # Put together a valid URL address.
  url <- paste0('https://www.random.org/integers/',
          '?num=', nsim,'&max=', format(10^digits, scientific=FALSE),
          '&min=1&col=1&base=10&format=html&rnd=new')

  # Compile the code to extract the tag containing the numbers.
  # Take a look at the value of url to know why this works.
  html.code <- getURL(url)
  html <- htmlTreeParse(html.code, useInternal = TRUE)
  numbers.text <- xpathApply(html, "//pre[@class='data']", xmlValue)[[1]]
  numbers <- strsplit(numbers.text, split="\\n")[[1]]
  return(as.numeric(numbers)*10^(-digits)) # Normalise to (0,1)
}
```

For this course, it will not be necessary to use true random numbers. However, if you want to compare the performance of any algorithm we study, you can use one of this sequences.■

There is a vast amount of research focused on finding efficient ways of generating true random numbers. In applications like cryptography, it is very important to have unpredictable sequences: if someone was to decipher the algorithm that is being used to encrypt data, then there could be really bad consequences. On the contrary, for many of the algorithms we will study in this course; all we need is that the sequence of numbers generated behave 'reasonably' similar to a sequence of random numbers in $(0, 1)$. When working with true random numbers, the result are unreproducible. When working with pseudo-random numbers instead, one often specifies a seed and constructs the rest of the pseudo-random sequence based on this number; so the results are always repeatable. Reproducibility can be both an advantage or a disadvantage depending on the particular application.

2.1 Pseudo-Random Uniform Generation

The low efficiency of generating true random numbers is what motivates pseudo-random numbers. We will not do a deep survey of methods for generating pseudo-random numbers. In fact, it is not even advisable that you use the methods presented here for your algorithms. There is a vast amount of research that has taken place in this field. So it is better to use well-proven methods. Having said this, the method we will describe now is surprisingly good given its simplicity.

Here is a short list of some things we will be looking for in a good random number generator:

1. It must have the moments and quantiles as the uniform distribution: mean 0.5, median 0.5, standard deviation $\sqrt{1/12}$, etc.

2. We would like to test its conformity with the uniform distribution by using some goodness-of-fit test (e.g., the χ^2 test).

3. Since we want to simulate from a sample of uniformly distributed independent variables in $(0,1)$, the produced numbers should satisfy independence. For instance, we will not want that a number is correlated with its predecessors in the sequence.

Do not underestimate the last point above, as it will fail with our method if we do not choose adequate initial parameters. The *linear congruency method*, introduced by David Lehmer in 1949, is defined as follows [Leh43]:

Definition 2.2 A *linear congruential generator* (LCG) with modulus $M \in \mathbb{N}$ (meant to be large), multiplier $a \in \{1, 2, \ldots, M\}$, increment $c \in \{0, 1 \ldots, M\}$ and seed $x_0 \in \{0, 1, \ldots, M-1\}$ is the sequence $(r_i)_{i=1}^{\infty}$ obtained by considering the sequence $(x_i)_{i=1}^{\infty}$ defined as

$$x_i = ax_{i-1} + c \mod M, \qquad i \geq 1,$$

and then setting $r_i := x_i/M$.

Observe the general framework for generating random numbers. We have a state space $S = \{0, 1, \ldots, M-1\}$, which are the possible values our procedure output may take. Then we define a rule $T : S \to S$ which indicates how to produce the next number from the current number. We input the x_0 and produce a sequence $(T^i(x_0))_{i=1}^{\infty}$. Finally, since the numbers take values on S and we want them to be uniformly distributed in $(0,1)$, we transform $(T^i(x_0))_{i=1}^{\infty}$ to meet our requirements. In this case, we simply divide by M.

One example of choice of parameters is the NAG Fortran generator `G05CAF` uses $M = 259$ and $a = 1310$ and $c = 0$. More modern software uses much modern parameters. For example Borland C/C++ uses $M = 2^{32}$, $a = 22695477$ and increment $c = 1$. A poor choice of parameters can be really bad, as our later examples will show. It is not a good idea to

use your own parameters. If you want to use the LCG for your future algorithms, stick to well-proven initial paremeters. IBM's RANDU is a very famous example of a poor choice of initial parameters in the 70s.

Example 2.3 (Linear congruential generator) Below we will see how to implement a function using a linear congruential generator to create random numbers.

```
LCG <- function(nsim, M = 2^32, a = 22695477, c = 1, seed = 110104){
    X = c(seed, numeric(nsim-1)) # Preallocate space
    for(i in 1:(nsim-1)) X[i+1] <- ((a*X[i] + c)%% M) # Apply LCG rule
    return(X/M) # Apply transform
}
rseq <- LCG(1000)
head(rseq) # The first values

## [1] 2.563559e-05 8.118340e-01 6.844589e-01 2.424782e-01 9.725499e-01
## [6] 9.661459e-01

summary(rseq) # Quantiles and mean look good.

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000256 0.2470000 0.5128000 0.5038000 0.7552000 0.9995000

sd(rseq) # Theoretical value is srt(12)~~0.2887

## [1] 0.2924752
```

The period of the method. The way the linear congruential method is designed, there is exactly M possible values which come out of the sequence $(T^i(x_0))_{i=1}^{\infty}$. So eventually, a number will repeat. We say that the method is *periodic*. A poor choice of parameters will give short periods for many seeds. A good choice of parameters is that which guarantees the maximum period possible for every seed. This is the content of the Hull-Dobell theorem [HD62] (a result which you really don't need to memorise).

Theorem 2.4 (Hull-Dobell) *Provided that the increment c is nonzero. Then the LCG has maximum period possible for every seed if and only if the three folowing conditions hold:*

1. *c and m are relative prime.,*
2. *$a-1$ divides all the prime factors of m,*
3. *4 divides $a-1$ if and only if 4 divides m.*

Example 2.5 (Short period for the LCG) Below we will see how to implement a function using a linear congruential generator to create random numbers.

```
LCG(25, M = 384, a = 5, c = 32, seed = 56)
```

```
##  [1] 0.1458333 0.8125000 0.1458333 0.8125000 0.1458333 0.8125000 0.1458333
##  [8] 0.8125000 0.1458333 0.8125000 0.1458333 0.8125000 0.1458333 0.8125000
## [15] 0.1458333 0.8125000 0.1458333 0.8125000 0.1458333 0.8125000 0.1458333
## [22] 0.8125000 0.1458333 0.8125000 0.1458333
```

Goodness-of-fit A goodness-of-fit test can be implemented using the χ^2 test. This test works like this. Suppose you have a theoretical distribution that can take the k values $x_1, x_2, \ldots, x_k$ each one with probability $p_1, p_2, \ldots, p_k$. Then e_i is the expected number of occurrences in the i-th category. Define the $\hat{\chi}^2$-statistic as

$$\hat{\chi}^2 = \sum_{i=1}^{k} \frac{(o_i - e_i^2)}{e_i}.$$

It is shown in a basic course in statistics that $\hat{\chi}$ approximately follows the distribution of a χ^2 random variable with $k-1$ degrees of freedom. To implement this in our case, we must first divide $[0,1]$ into k equally sized bins (k is arbitrary). Then e_i will be n/k where n is the number of simulations and o_i will be the number of observations in the k-th bin. We must then compare the obtained value in χ^2 for $k-1$ degrees of freedom. This is done in R using the function `pchisq`.

Example 2.6 We now show an implementation of the χ^2 goodness-of-fit test.

```
nsim <- 50
k = 5 # number of bins
rseq <- LCG(nsim, seed=110104)
ei <- rep(nsim/k, times=k)
oi <- table(trunc(k*rseq)/k) # Easy way to compute the oi
chi2 <- sum((oi-ei)^2/ei)
pchisq(chi2, df = k-1)
```

```
## [1] 0.9008146
```

The way in which the last number must be interpreted is as the probability that the generated sequence truly comes from a uniform distribution. In this case it is of about 90%, which is considered high. ■

Failure of independence Another important thing to check in random number generation, is that desired sequences truly seem like independently generated. There are number of things that can be done. The easiest of which is to check the correlation with output values and its predecessors. Here we show with two examples what can go wrong and how independent things would look like.

Example 2.7 (Failure of independence) We will do a simple plot comparing the output term r_i on the horizontal axis and r_{i+1} on the vertical axis. If the number were independent now pattern should be recognisable.

1) This is an example of something that looks reasonably independent.

```
library(ggplot2)
nsim <- 1000
rseq <- LCG(nsim)
qplot(rseq[-nsim], rseq[-1])
```

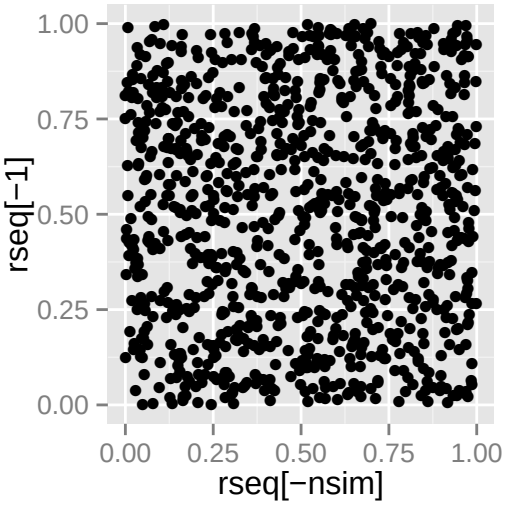

Figure 2.1: r_i independent of r_{i+1}

2) This is an example of something that definitely does not show independence.

```
library(ggplot2)
nsim <- 1000
rseq <- LCG(nsim, M = 574, a = 29, c = 466, seed = 11)
qplot(rseq[-nsim], rseq[-1])
```

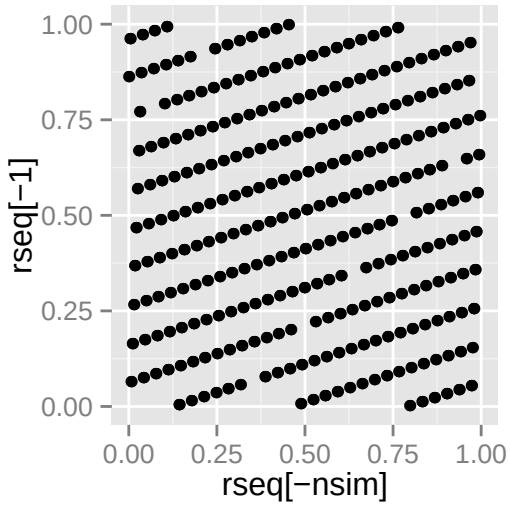

Figure 2.2: r_i extitnot independent of r_{i+1}

As a conclusion from this section it should be clear that is not a very good idea to come out with our own methods for creating random numbers. The method used by R is the *Mersenne Twister*. It is not really worth it to go into the details of this more complicated method for the purpose of this book. But one must know that it is a widely accepted and reliable method.

2.2 The Inverse Function Method

Now that we now how to simulate a sequence of pseudo-random numbers following the uniform distribution in $[0,1]$ we will now see how to simulate an observation of another probabilistic distribution. What we will want to do is to design methods that allow us to obtain random sequences of other distributions by using random uniform sequences.

Example 2.8 (Simulating a Bernoulli distribution) The simplest case of such a method is the simulation of a Bernoulli random variable with parameter p. Observe that if $U \sim \text{Unif}[0,1]$, then $\mathbb{P}(U \in [0,p]) = p$ and $\mathbb{P}(U \in (p,1]) = 1-p$. Hence the if we define a function $G(U)$ as

$$X = G(U) = \begin{cases} 1 & \text{if } U \in (0,p) \\ 0 & \text{if } U \in [p,1), \end{cases}$$

then X is has Bernoulli law with parameter p. ■

General case for discrete variables. Let us generalise the above example. Let $U \sim \text{Unif}[0,1]$ and suppose we would like a random variable X to take the values $x_1 < \ldots < x_n, \ldots$ with corresponding probabilities $p_1, \ldots, p_n, \ldots$. Define $F_n = \sum_{i=1}^{n} p_i$ and set $P_1 = [0, F_1]$ and $P_i = (F_{i-1}, F_i]$ for $i \geq 2$. Then $\mathbb{P}(U \in P_i) = p_i$. If we now set

$$X = G(U) = \begin{cases} x_i & \text{if } U \in P_i, \end{cases}$$

then $\mathbb{P}(X = x_i) = p_i$, so that X has the desired distribution. Let F_X be the cumulative distribution function of X, i.e., $F_X(x) = \mathbb{P}(X \leq x)$. Observe that G is like an 'inverse' of F_X in the sense that $F_X(G(F_i)) = F_i$.

Example 2.9 (Simulating a Geometric distribution) The geometric distribution with parameter p can be characterised as the number of attempts before a success with occurs of probability p in a sequence on identical independent events. If $X \sim \text{Geom}(p)$, then $\mathbb{P}(X = n) = (1-p)^{n-1}p$. We will show in R how use the method just described above to simulate observations of a geometric distribution out of observations of uniform random variables obtained using the function `runif`.

```
set.seed(110104)
# Function to simulate ------
rGeom.aux <- function(p){ # Simulate one draw
  U <- runif(1)
  # We now use a while loop to find to which partition U belongs to.
  k <- 1
  max.iter <- 10e5 # Security parameter to avoid infinite loops.
  # Good practice even when not necessary.
  Fk <- p
  while(U > Fk  &  k <= max.iter){
    k <- k + 1
    Fk <- Fk + (1-p)^(k-1)*p
  }
  return(k-1)
}
rGeom <- function(nsim, p){ # Simulate several draws
  replicate(nsim, rGeom.aux(p=p))
}
# Comparison with theoretical values ------
nsim <- 1000
p <- 0.45
geom.seq <- rGeom(nsim, p) # nsim simulations of Geom(p)
```

```r
max.obs <- max(geom.seq) # Max value to print in tables
observed <- table(factor(geom.seq, levels=0:max.obs)) # Observed frequencies
probs <- (1-p)^(0:max.obs)*p
expected <- round(probs*nsim)
data.frame(data.frame(observed), Expected=expected) # Observed vs expected
```

```
##    Var1 Freq Expected
## 1     0  426      450
## 2     1  255      248
## 3     2  148      136
## 4     3   69       75
## 5     4   51       41
## 6     5   24       23
## 7     6   12       12
## 8     7   10        7
## 9     8    1        4
## 10    9    2        2
## 11   10    1        1
## 12   11    1        1
```

General case for continuous variables. What made the previous easy cases work was that we found a function G such that $G(F_X(x)) = x$. Recall that $F(x) = \mathbb{P}(X \leq x)$ is always a non-decreasing function. In order to be invertible, we only need that F is strictly increasing. When F_X is invertible, we have the following result which can be used to generalize the previous cases.

Proposition 2.10 (The inverse function method) *Let $U \sim \text{Unif}[0,1]$ and X be a continuous random variable with cumulative distribution function F_X. Suppose that F_X is invertible. Then $F_X^{-1}(U)$ has the same law as X.*

Proof. In order to prove that X and $F_X^{-1}(U)$ have the same probability law it is sufficient to show that $\mathbb{P}(X \leq x) = \mathbb{P}(F_X^{-1}(U) \leq x)$, as this entirely determines the probability law. We thus compute

$$\mathbb{P}(F_X^{-1}(U) \leq x) = \mathbb{P}(U \leq F_X(x)) = F_X(x) = \mathbb{P}(X \leq x),$$

where we used that F_X is strictly increasing and hence that $F_X(x_1) \leq F_X(x_2)$ if and only if $x_1 \leq x_2$. This proves the result. □

The proof above works whenever we can find a function G solve the problem $F_X(G(u)) = u$. If we define G as $G(u) := \inf\{x \mid F(x) \geq u\}$, then we recover the previous discrete cases.

Example 2.11 (Simulating an exponential distribution) Suppose we want to simulate a random variable having exponential distribution with parameter $\lambda > 0$. We say that $X \sim \text{Exp}\lambda$ if its cumulative distribution function is $F_X(x) = 1 - e^{-\lambda x}$. This function is easy to invert. Its inverse is $F_X^{-1}(u) = -\lambda^{-1}\ln(1-U)$.

```
set.seed(110104)
library(ggplot2)
nsim <- 1000
lambda <- 2
rExp <- function(nsim, lambda){
  return((-1/lambda)*log(1-runif(nsim)))
}
dat <- data.frame(Value=rExp(nsim, lambda))
ggplot(dat, aes(x=Value)) +
  geom_histogram(aes(y=..density..), binwidth= .2, colour="black", fill="white") +
  stat_function(fun = function(x) lambda*exp(-lambda*x),colour = "blue")
```

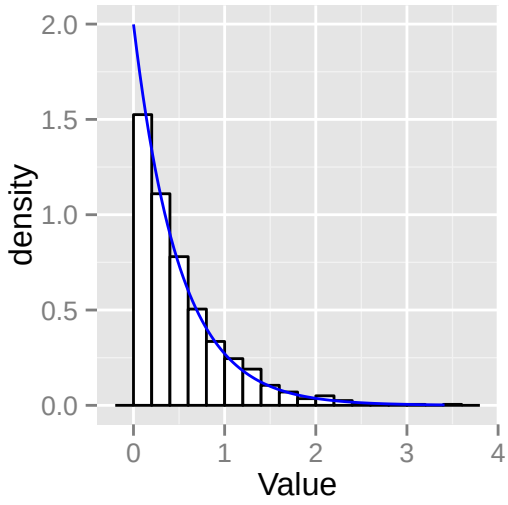

Figure 2.3: Density histogram of observations of simulated exponential random variables vs theoretical density Exp(λ) (in blue).

Observe that U and $1-U$ have identical distributions if $U \sim \text{Unif}(0,1)$. The above method would have works just as well if we had taken $G(u) = -\lambda^{-1}\ln(u)$ instead. ■

In practice, we rarely have a closed mathematical expression for the inverse of F_X (for instance, if we wanted to simulate the normal distribution). One solution is to try to use a

numerical software to estimate the inverse. Other approaches, sometimes more efficient, are given in the next sections.

2.3 The Acceptance-Rejection Method

The *acceptance-rejection rejection method* is an alternative to sample/simulate from probability distribution when we do not have an easy way to compute F_X^{-1}. It is a little bit complicated to grasp the intuition of the method at first sight. For that reason, let us first give a simpler case.

Uniform rejection sampling. A uniform density function on a (measurable) set E is a function f such that $f(x) = c \geq 0$ is constant for all $x \in E$ and $c \int_E \mathrm{d}x = 1$. Evidently, the function $f_X(x) = \mathbb{1}_{[0,1]}$, the indicator function of the interval $[0,1]$, defines a uniform density on $[0,1]$.

Proposition 2.12 *Let $D \subset [0,1]$ be a subset with $\int_D \mathrm{d}x > 0$. Define a random variable X as follows.*

1. *Generate $U \sim \mathrm{Unif}[0,1]$.*
2. *If $U \in D$, set $X = U$, otherwise, go to the previous step and generate a new independent value of U.*

Then X is distributed uniformly on D.

Proof. Here is a short simple argument. We first observe that the above recipe is well-defined. For this, we must only check that the method eventually concludes. Let N be the minimal iteration such that $U \in D$. Observe that $\mathbb{P}(N > n) = (1 - \mathbb{P}(U \in E))^n$, which converges to zero as $n \to \infty$ since $\mathbb{P}(U \in E) > 0$.

Let now $E \subset D$ be any (measurable) set. Then by construction we have that $\mathbb{P}(X \in E) = \mathbb{P}(U \in E \mid U \in D)$ since each draw of U is independent from the past and we set $X = U$ until we know that $U \in D$. But this implies that X has density function $(1/\mathbb{P}(U \in D))\mathbb{1}_D$, since

$$\mathbb{P}(X \in E) = \mathbb{P}(U \in E \mid U \in D) = \frac{\mathbb{P}(\{U \in E\} \cap \{U \in D\})}{\mathbb{P}(U \in D)} = \int_E \frac{1}{\mathbb{P}(U \in D)} \mathrm{d}x,$$

and this concludes the proof. □

Acceptance-Rejection method. Suppose we want to simulate from a probability law determined by a density function f but we only know how draw from a density g with the property that $f(y) \leq Mg(y)$ for all y with $M > 0$ some real number (such g is called a *majorising* function for f). Then there is a marvellous trick to draw from f.

Suppose for the moment that $M = 1$, here is the intuition of the method. Suppose that we are 'situated' at a train station y so that $g(y)$ measures the density of arrivals of passengers

at the station y. Suppose further that each arriving passenger is accepted with a probability $p(y)$ depending on y. The density of accepted passengers is now $p(y)g(y)$. If we want to simulate acceptances according to the density f, then we better had $p(y) = f(y)/g(y)$, so that the density of accepted passengers at y becomes $f(y)$. How can we achieve this? We can simply generate a uniform random variable U in $[0,1]$ and accept an arrival if $U \leq f(y)/g(y)$ (here we use that $f(y) \leq g(y)$) and reject it otherwise.

Proposition 2.13 *Let f and g be density function such that there exists $M \geq 1$ such that $f \leq Mg$. Define X according to the following algorithm*

1. *Generate $U \sim \mathrm{Unif}[0,1]$ and Y independently according to the density g,*
2. *If $U \leq f(Y)/(Mg(Y))$ then accept and set $X = Y$. Otherwise, go back to the previous step and start again.*

Then the resulting X is distributed according to the density function f.

Proof. The proof is just a computation that uses the law of total probability.

$$\begin{aligned}\mathbb{P}(X \leq x) &= \mathbb{P}\Big(Y \leq x \mid U \leq \tfrac{f(Y)}{Mg(Y)}\Big) \\ &= \frac{\mathbb{P}\Big(Y \leq x, U \leq \tfrac{f(Y)}{Mg(Y)}\Big)}{\mathbb{P}\Big(U \leq \tfrac{f(Y)}{Mg(Y)}\Big)}\end{aligned}$$

where the first equality holds by construction, the second by the definition of conditional probability. We now use the law of total probability two compute the term in the numerator.

$$\begin{aligned}\mathbb{P}\Big(Y \leq x, U \leq \tfrac{f(Y)}{Mg(Y)}\Big) &= \int_{\infty}^{\infty} \mathbb{P}\Big(Y \leq x, U \leq \tfrac{f(Y)}{Mg(Y)} \mid Y = y\Big) g(y)\,\mathrm{d}y \\ &= \int_{\infty}^{x} \mathbb{P}\Big(U \leq \tfrac{f(Y)}{Mg(Y)} \mid Y = y\Big) g(y)\,\mathrm{d}y \\ &= \int_{\infty}^{x} \mathbb{P}\Big(U \leq \tfrac{f(y)}{Mg(y)}\Big) g(y)\,\mathrm{d}y \\ &= \frac{1}{M}\int_{-\infty}^{x} \frac{f(y)}{g(y)} g(y)\,\mathrm{d}y \\ &= \frac{1}{M}\int_{-\infty}^{x} f(y)\,\mathrm{d}y.\end{aligned}$$

An almost identical computation for the denominator yields

$$\begin{aligned}\mathbb{P}\left(U \leq \frac{f(Y)}{Mg(Y)}\right) &= \int_{\infty}^{\infty} \mathbb{P}\left(U \leq \frac{f(Y)}{Mg(Y)} \mid Y = y\right) g(y)\,\mathrm{d}y \\ &= \int_{-\infty}^{\infty} \frac{f(y)}{Mg(y)} g(y)\,\mathrm{d}y \\ &= \frac{1}{M}\int_{-\infty}^{\infty} f(y)\,\mathrm{d}y \\ &= \frac{1}{M}.\end{aligned}$$

Combining the two simplifications above, we conclude that $\mathbb{P}(X \leq x) = \int_{-\infty}^{x} f(y)\,\mathrm{d}y$, which is to say that f is a density function associated to the law of X. This concludes the proof. □

Example 2.14 (Generating a Beta distribution) The Beta distribution is a probability law that has applications for Bayesian inference as well models in which a probabilistic phenomena is supported in an interval of finite length. We say that X has is distributed Beta with parameters α, β if it has density function

$$f_X(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}\mathbb{1}_{[0,1]}$$

where $\Gamma(x) = \int_0^{\infty} x^{t-1}e^{-x}\,\mathrm{d}x$ is the gamma function. One can show that the maximum valued of f is attained at $(\alpha-1)/(\alpha+\beta-2)$. Hence we can use set $g(x) = \mathbb{1}_{[0,1]}$ and $M = f((\alpha-1)/(\alpha+\beta-2))$.

```
alpha = 10 # Some random values for this example.
beta = 5
f <- function(x, alpha, beta){
  gamma(alpha + beta)/(gamma(alpha)*gamma(beta))*x^(alpha-1)*(1-x)^(beta-1)
}
g <- function(x) {1}
M <- f((alpha-1)/(alpha+beta-2), alpha, beta)
ggplot(data.frame(x=c(0,1)), aes(x)) +
  stat_function(fun = f, arg=list(alpha=alpha, beta=beta), aes(colour = "f")) +
  stat_function(fun = function(x) g(x)*M, aes(colour = "M*g")) +
  scale_colour_manual("Function", values = c("red", "blue")) + ylab("density")
```

```
set.seed(999)
nsim <- 1000
rBeta.aux <- function(alpha, beta, verbose=FALSE){
```

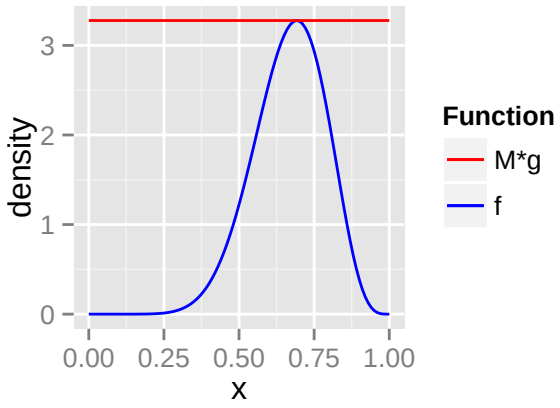

Figure 2.4: Acceptance-rejection for a Beta distribution: comparison between f and a majorising function with parameters $(\alpha, \beta) = (10, 5)$.

```
  d <- data.frame() # To keep track of acceptance and rejection.
  accepted <- FALSE
  max.iter <- 10e6 # When we use whiles, we set a max.iter paramater for security.
  iter <- 1
  while(!accepted & iter<=max.iter){
    U <- runif(1)
    Y <- runif(1) # In our case, g is uniform [0,1]. So this is sampling from g.
    if(U <= f(Y, alpha, beta)/(g(Y)*M)){
      accepted <- TRUE
      X <- Y
    }
    d <- rbind(d, data.frame(Y=round(Y,4), U=round(U,4),
        f.over.Mg=round(f(Y, alpha, beta)/(g(Y)*M),4), Accepted=accepted))
    iter <- iter + 1
  }
  if(verbose==TRUE){
    return(d) # If we indicate verbose we return a table with outcomes.
  } else{
    return(X)
  }
}
rBeta <- function(nsim, alpha, beta){
  replicate(nsim, rBeta.aux(alpha=alpha, beta=beta))
```

```
}
hist(rBeta(nsim, alpha, beta), breaks=20, main="", prob=TRUE)
```

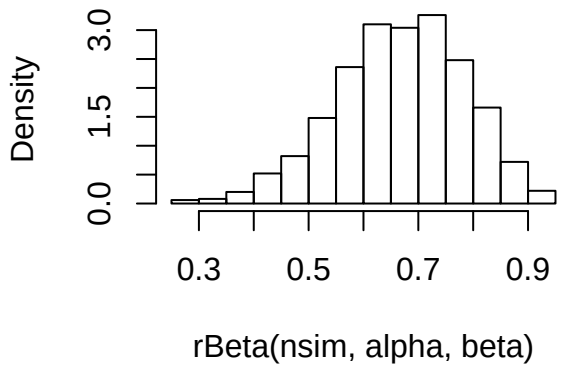

Figure 2.5: Acceptance-rejection for a Beta distribution: histogram of simulated observations.

```
rBeta.aux(alpha, beta, verbose=TRUE)

##         Y      U f.over.Mg Accepted
## 1 0.0119 0.5266    0.0000    FALSE
## 2 0.0990 0.6736    0.0000    FALSE
## 3 0.5218 0.5151    0.4578    FALSE
## 4 0.9931 0.0698    0.0000    FALSE
## 5 0.2994 0.6724    0.0142    FALSE
## 6 0.8710 0.5923    0.2441    FALSE
## 7 0.1621 0.4379    0.0001    FALSE
## 8 0.5788 0.1415    0.7007     TRUE
```

This of course, was the simplest case when g can be taken constant. However, other options could had been used. Observe we cannot a constant g if our objective density f is supported in a set of infinite length. ■

Exercise 2.1 Use the acceptance rejection method to simulate observations of $Z = |X|$ where X is a standard normal random variable. *Hint:* Deduce that the density function of Z is twice the density function of X due to symmetry. Set $g(x) = e^{-x}$ and find an appropriate constant M.

Observe the for the acceptance-rejection method to work efficiently we better had M as small as possible so that the majorising conditions still holds. Otherwise, the acceptance probability would be too low.

Exercise 2.2 (Von Neumann bias correction) Suppose we are throwing a biased coin with probability of observing heads $p > 1/2$. Try to devise a method based on the ideas similar to the ones in this section to obtain an unbiased sample from the biased sample. *Note:* sometimes true random numbers have a natural biased and a method like this, called von Neumann bias correction, is used to obtain unbiased samples.

2.4 The Box-Müller method

The Box-Muller method is a great trick for easily generating sample of a random normal distribution. It is *only* useful for the normal distribution and no other distribution. It is widely used for its simplicity of application. The whole idea is simply an observation coming from multivariate calculus.

The change of variable formula. One of the first results studied in multivariate calculus is the change of variable formula. It is a generalisation of integration by substitution. Suppose we have a invertible transform $T = (T_1, \dots T_n) \colon \mathbb{R}^n \to \mathbb{R}^n$. Suppose that $x_1, \dots, x_n$ are coordinates of some integrable function f. Then the change of variable formula states how we can integrate f in terms of the new coordinates $T_1(x_1, \dots, x_n), \dots, T_n((x_1, \dots, x_n)$. This is given as follows:

$$\begin{aligned}&\int_{T(E)} f(x_1, \dots, x_n)\, \mathrm{d}x_1 \dots \mathrm{d}x_n \\ &\qquad = \int_E f(x_1(T_1, \dots, \dots, T_n), \dots, x_n(T_1, \dots, \dots, T_n))\, |\mathrm{D}T|\, \mathrm{d}T_1 \dots \mathrm{d}T_n.\end{aligned}$$

Here, $|\mathrm{D}T|$ is the determinant of the jacobian of the transform T.

Example 2.15 (Integration using polar coordinates) One of the most usual applications of the change of variable formula is the use of polar coordinates. in $\mathbb{R}^2$. Define a transformation $T \colon \mathbb{R}^2 \to \mathbb{R}^2$ from polar coordinates $\{(r, \theta) \mid r \in (0, \infty), \theta \in [0, 2\pi)\}$ to rectangular coordinates $\{(x, y) \mid x \in \mathbb{R}, r \in \mathbb{R}\}$ as

$$\begin{cases} T(r, \theta) = (T_1(r, \theta), T_2(r, \theta)) = (r \cos(\theta), r \sin(\theta)). \end{cases}$$

The Jacobian of T is the linear transform represented by the matrix

$$\begin{bmatrix} \frac{\partial T_1}{\partial r} & \frac{\partial T_1}{\partial \theta} \\ \frac{\partial T_2}{\partial r} & \frac{\partial T_2}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -r \sin(\theta) \\ \sin(\theta) & r \cos(\theta) \end{bmatrix}.$$

The determinant of this matrix is easily seen to be r. Thus the change of variable formula becomes

$$\int_{T(E)} f(x,y)\,\mathrm{d}x\,\mathrm{d}y = \int_E f(r\cos(\theta), r\sin(\theta))r\,\mathrm{d}r\,\mathrm{d}\theta.$$

In what follows we will see how to use polar coordinates to find an easy way to generate random variables ■

The Box-Müller trick consists of realising that if we take the joint density function of two independent standard normal random variables X and Y, then the transformation to polar coordinates yields a nice expression easy to work with. The joint density function $f_{X,Y}$ of two independent standard normal random variables X and Y is

$$f_{X,Y}(x,y) = f_X(x)f_Y(y) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}y^2} = \frac{1}{2\pi}e^{-\frac{1}{2}(x^2+y^2)}.$$

Then by the change of variable formula, the joint density of (R,Θ), where R and Θ are defined by the transformation $X = R\cos(\Theta)$ and $Y = R\sin(\Theta)$ is given by

$$f_{R,\Theta}(r,\theta) = \frac{1}{2\pi}e^{-\frac{r}{2}}r\mathbb{1}_{[0,2\pi)}(\theta)\mathbb{1}_{[0,\infty)}(r) = \left(\frac{1}{2\pi}\mathbb{1}_{[0,2\pi)}(\theta)\right)\left(\frac{1}{2\pi}e^{-\frac{r}{2}}\mathbb{1}_{[0,\infty)}(r)\right).$$

From the the above expression we can see that R and Θ are independent random variables (since the joint density is the multiplication of a density depending only on R and a density depending only on Θ, and this is the definition of independence). The marginal density of Θ is

$$f_\Theta(\theta) = \int_0^\infty f_{R,\Theta}(r,\theta)\,\mathrm{d}r = \frac{1}{2\pi}\mathbb{1}_{[0,2\pi)}(\theta),$$

hence Θ is uniformly distributed in $[0,2\pi)$. This implies that we can easily simulate an observation of Θ by simulating $U_1 \sim \mathrm{Unif}[0,1)$ and multiplying U_1 by 2π. Analogously, the marginal density of R is

$$f_R(r) = \int_0^{2\pi} f_{R,\Theta}(r,\theta)\,\mathrm{d}\theta = re^{-\frac{r^2}{2}}\mathbb{1}_{[0,\infty)}(r).$$

Simulating an observation of R with this density can be used using the inverse-function method with a little extra trick. By the integration by substitution formula (which is the one-dimensional case of the change of variable we just discussed), the density function of $S := R^2$ is $f_S(s) = e^{-s/2}$. So R^2 is distributed as an exponential variable with parameter $1/2$. He have already seen how to simulate an observation of an exponential using a uniform random variable. To be precise, we can simulate R^2 by simulating $U_2 \sim \mathrm{Unif}[0,1)$ and setting $R^2 = -2\log(1-U_2)$, or even simpler, $R^2 = -2\ln(U_2)$, as U_2 and $1-U_2$ have exactly the same distribution. As a conclusion of this discussion, we can simulate a sample of $(R,\Theta$ by

simulating $U_1, U_2 \sim \text{Unif}[0,1)$ and setting

$$\begin{cases} \Theta = 2\pi U_1 \\ R = \sqrt{-2\ln(U_2)}. \end{cases}$$

Finally, since $X = R\cos(\Theta)$ and $Y = R\sin(\Theta)$, we can recover a simulation of (X, Y) by setting

$$\begin{cases} X = \sqrt{-2\ln(U_2)}\cos(2\pi U_1) \\ Y = \sqrt{-2\ln(U_2)}\sin(2\pi U_1). \end{cases}$$

Observe that this method will always produce two instead of only one observation of a standard normal random variable, furthermore, this two observations are assumed to be independent.

Exercise 2.3 Make a programme in R to simulate a sample of a normal distributions with mean μ and variance σ^2 using the Box-Müller method.

Chapter 3

Monte-Carlo integration

Imagine you have a sample $x_1, x_2, \ldots, x_N$ of independent of observations of a random variable X. If someone was to ask what is the expected value of X, a very sensible answer would be assume it is somewhere close to

$$\sum_{i=1}^{n} x_i.$$

Suppose now that $\phi : \mathbb{R} \to \mathbb{R}$ is a (measurable[1]) function. Then $\phi(X)$ is a random variable. If we were now asked about the expected value of the random variable $\phi(X)$, an equally sensible answer would to suppose it is somewhere near

$$\sum_{i=1}^{n} \phi(x_i).$$

In other words, $\sum_{i=1}^{n} \phi(x_i)$ is a *statistical estimator* of $\mathbb{E}[\phi(X)]$. In essence, this is all what Monte-Carlo integration is about. In what follows, we will want to provide a mathematical framework to justify this procedure and to develop a theory that allows to an estimate of the error of estimation and modify the strategy in the name of efficiency. Before doing so, let us start with some history.

History of Monte-Carlo. The method of Monte-Carlo is attributed to the mathematician Stanislaw Ulam who implemented it with the help of no other than John von Neumann. So as simple as it may seem, Monte-Carlo integration was devised by two of the greatest minds of the 20th century. Furthermore, it was first implemented on the ENIAC, the first modern computer. The following extract is taken from [Eck87] and it is a recount that Ulam made of the method:

> 'The first thought and attempts I made to practice [the Monte Carlo method] were suggested to me in 1946 as I was convalescing from an illness and playing

[1] Simply ignore the word 'measurable' if you are not familiar with measure theory.

> solitaires. The questions was what are the chances that a Canfield solitaire laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than "abstract thinking" might not be able to lay it out say one hundred times and simply observe and count the number of successful plays. This was already possible to envisage with the beginning of the new era of fast computers, and I immediately thought of problems of neutron diffusion and other question of mathematical physics, and more generally how to change processes described by certain differential equations into an equivalent form interpretable as a succession of random operations. Later... I described the idea to John von Neumann and we began to plan actual calculations.'

By than time, von Neumann and Ulam were working at the Manhattan Project at Los Alamos National Laboratory, US. It was customary in the kind of work they did to have code names for most projects. It is said that a colleague of them, Nicholas Metropolis, suggested the name Monte-Carlo in reference to the use of chance and Ulam's uncle, who used borrow money for gambling at the famous Monte Carlo casino in Monaco. The ENIAC had just been developed back then, Neumann and Ulam used configured the ENIAC to run Monte Carlo simulations and used it for their study related to the Manhattan Project.

3.1 Classical integration theory and the curse of dimensionality

Monte Carlo is, first of all, a theory of integration. An expected value is an integral. Suppose f is the density function of a random variable X, then

$$\mathbb{E}[\phi(X)] = \int_{\mathbb{R}} \phi(x) f(x)\,\mathrm{d}x.$$

Even more generally, if $X = (X_1, \ldots, X_n)$ is a random vector in $\mathbb{R}^n$ with density f (its support may be contained in a smaller set, but we can always define f to be zero outside this set) and $\phi : \mathbb{R}^n \to \mathbb{R}$ a (measurable) function, then

$$\mathbb{E}[\phi(X)] = \int_{\mathbb{R}} \cdots \int_{\mathbb{R}} \phi(x_1, \ldots, x_n) f(x_1, \ldots, x_n)\,\mathrm{d}x_1 \ldots, \mathrm{d}x_n.$$

Classical quadrature rules for 1-dimensional integrals. In numerical analysis, a quadrature rule for a integral $I = \int_a^b \phi(x)\,\mathrm{d}x$ is a rule $\hat{I}$ to approximate the integral. We now briefly described the most popular rules that are studied in any introductory course of numerical integration. Suppose we choose an *even* grid $a = x_0, x_1, \ldots, x_N = b$ such that $x_{i+1} - x_i = 1/N$. Then the rules are

1. The (left) *rectangular* or *Riemann sums* approach

$$\hat{I}_R := \sum_{i=1}^{N}(x_{i+1}-x_i)\phi(x_i) = \frac{1}{N}\sum_{i=1}^{N}\phi(x_i).$$

   The error rate is[2] $O(N^{-1})$ provided that[3] $\phi \in C^1$.

2. The *Trapezoidal rule* is given by

$$\hat{I}_T := \frac{1}{2}\sum_{i=0}^{N}(x_{i+1}-x_i)(\phi(x_{i+1})+\phi(x_i)) = \frac{1}{2N}\sum_{i=1}^{N}\big(\phi(x_{i+1})+\phi(x_i)\big).$$

   The error rate is $O(N^{-2})$ provided $\phi \in C^2$.

3. Suppose N is even. The *Simpson's rule* is given by

$$\hat{I}_S := \frac{1}{3N}\sum_{i=0}^{N}\left(\phi(x_0)+2\sum_{j=1}^{N/2-1}\phi(x_{2j})+4\sum_{j=1}^{N/2}\phi(x_{2j-1})+\phi(x_N)\right)$$

   The error rate is $O(N^{-4})$ provided $f \in C^4$. The intuition behind the previous rules is very clear. This one is a little bit more tricky. it is derived from first dividing in $N/2$ subintervals. And then interpolating a quadratic polynomial that passes through the value of the function at the midpoint of each subinterval.

The higher-dimensional rules. When we have to compute an integral in higher dimensions, we can use approach the problem as integrating many times. For example, suppose we want to compute an integral

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2}\phi(x_1,x_2)\,\mathrm{d}x_1\,\mathrm{d}x_2.$$

Define a function $I^{(1)}(x_2) = \int_{a_2}^{b_2}\phi(x_1,x_2)\,\mathrm{d}x_1$. This is a function of x_2 only. So computing our integral reduces to the one-dimensional problem

$$\int_{a_2}^{b_2} I^{(1)}(x_2)\,\mathrm{d}x_2.$$

We can implement a recursive algorithm to compute this integral. At each point where we need to evaluate $I^{(1)}$ we will need to use the quadrature rule again.

[2] Recall that the notation $f = O(g)$ means that there is a constant $M > 0$ such that $|f(x)| \le M|g(x)|$ for all x.

[3] The notation $f \in C^k$ is the standard notation for indicating that f is differentiable k-times with its k-derivative continuous.

Exercise 3.1 Write a program that receives a 2-dimensional function ϕ, points (a_1, b_1) and (a_2, b_2) and a string indicating a quadrature rule and that recursively computes the integral of ϕ over the rectangle $(a_1, b_1) \times (a_2, b_2)$.

As a consequence of using the composite rule. The error rates grow exponentially. Since if we have M points on each dimension. We will need a total of $N = M^n$ function evaluations. Hence a method that per dimension is $O(m^{-r})$ becomes of order $O(m^{-r/N})$ (intuitively, at each step, the best approximation is of order M but you are paying the price M^n). Thus, the n-dimensional problem error rates are:

1. *Rectangular rule*: $O(N^{-1/n})$ if $f \in C^1$.
2. *Trapezoidal rule*: $O(N^{-2/n})$ if $f \in C^2$.
3. *Simpson's rule*: $O(N^{-4/n})$ if $f \in C^4$.

As $n \to \infty$ the error rates become seriously bad. This is known as the *curse of dimensionality*. One of the strongest motivations for Monte Carlo simulation is that, as we shall see, it independent of the number of dimensions! It will always be $O(N^{-1/2})$ as a consequence of the central limit theorem.

3.2 The plug-in principle and non-parametric inference

The main purpose of Monte-Carlo integration is to develop techniques to compute quantities of the form

$$\mathbb{E}[\phi(X)] = \int \phi(x) f(x)\, \mathrm{d}x.$$

where f is the density function of a integrable random variable X ($E[|X|]$ exists) and $\phi : \mathbb{R} \to \mathbb{R}$ is a bounded (measurable) function. Observe that if we were able to compute these integrals that come as expectations, then we would be able to compute the integral of any function (convince yourself why). Hence, Monte-Carlo is regarded as in integration method. But contrary to the methods usually studied in numerical integration, it comes naturally within a probabilistic framework and we can do inference and apply statistical techniques.

It will be very important to start with a convenient set up. Recall that the cumulative distribution function $F(x) := \mathbb{P}(X \leq x)$ of a random variable X is enough to determine its law (e.g., if X is discrete, then $\mathbb{P}(X = x) = F(x) - F(x-1)$). In fact, any right-continuous function such that $\lim_{x \to -\infty} F(x) = 0$ and $\lim_{x \to \infty} F(x) = 1$ can be the distribution function of a random variable X. Furthermore, using arguments from calculus, we can prove that any such function F is differentiable almost everywhere, which is enough to guarantee the existence of a density function f. In other words, a distribution function F immediately

gives a corresponding random variable X with associated distribution function F and density function function f.

A *functional* is a fancy word for a real-valued function that takes functions as arguments. For example, define a functional θ_ϕ that receives distribution functions and returns $\mathbb{E}[\phi(X)] = \int \phi(x)f(x)\,\mathrm{d}x$ where X is a random variable having associated distribution function F and density function f. The idea behind Monte-Carlo integration is to simulate an observation coming from a sequence of independent identically distributed random variables having common distribution function F and use their values to make an estimate of $\mathbb{E}[\phi(X)]$.

The classical theory of parametric statistics–which is the theory studied in most introductory courses in mathematical statistics–is oriented to the discovery of an unknown parameter η governing the common law of a collection of N independent and identically distributed random variables $Z_1, \ldots, Z_N$. For example, we may assume the Z_i follow a normal distribution with unknown mean μ and variance σ^2.

An *estimator* $\hat{\eta}$ of a parameter η is a *any* function of the Z_i designed to estimate the value of η (an a priori, an estimator does not to be good). In classical statistics, if one is interested in knowing a quantity $\theta(\eta)$ depending on a parameter η, we can substitute the value of η for its estimator $\hat{\eta}$ and use $\theta(\hat{\eta})$ as en estimator of $\theta(\eta)$. For example, a possible estimator for the parameter λ of a sequence Z_i of exponential random variables with parameter λ would be $1/\frac{1}{N}\sum_{i=1}^{N} Z_i$, since $\mathbb{E}[Z_i] = 1/\lambda$ in this case.

Such an approach does not pay off much sense in our setting. It does not make any sense to assume we have a particular distribution in our data. Our problem is naturally non-parametric, wince we do not assume anything about the law of the X_i. However, we may choose to do something analogous. Given a collection (called a sample) $X_1, \ldots, X_N$ of independent identically distributed random variables, the *non-parametric estimator* of their common distribution F is

$$\hat{F}_N(x) := \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\{X_i \leq x\}}(x).$$

The estimator $\hat{F}_N$ is called the *empirical distribution function* of the sample $X_1, \ldots, X_N$.

The *plug-in principle* is a technique for non-parametric estimation that consists of replacing F by $\hat{F}_N$ in order to estimate the value of a functional $\theta(F)$. We can prove that if $\theta_\phi(F) = \mathbb{E}[\phi(X)]$, then

$$\theta_\phi(\hat{F}_N) = \frac{1}{N}\sum_{i=1}^{N} \phi(X_i), \tag{3.2.1}$$

which is exactly what our initial intuition suggested it should be at the beginning of the chapter. In the next section we recall some basic of the Riemann-Stieltjes integral (or the Lebesgue-Stieltjes integral) in order to justify that the resultant estimator after the plug-in is indeed (3.2.1). The reader may safely skip this section and take (3.2.1) simply as the definition of the Monte-Carlo estimator. However, the Stieltjes integral is a way to show the explicit dependence of $\theta_\phi(X)$ on F and compute the result of the plug-in. A more detailed

survey of the Stieltjes integral can be found in [Bar76].

Integrating with respect to $\mathrm{d}F(x)$[4]. For simplicity, assume we want to integrate on $[0,1]$. The Riemann integral is constructed as the limit of a sum the number of terms going to infinity. Suppose we have a partition $x_0 = a < x_1 < \dots x_n < x_{n+1} = b$ of $[a,b]$ and $c_i \in [x_i, x_{i+1}]$. Then the $\int_a^b \phi \,\mathrm{d}x$ is defined as the limit

$$\lim_{n\to\infty} \sum_{i=0}^{n} (x_{i+1} - x_i) f(c_i).$$

This limit may not be defined for some pathological functions[5] The idea of the Riemann-Stieltjes integral is to do a similar thing integrating with respect to an appropriate function G by attempting to compute the limit

$$\lim_{n\to\infty} \sum_{i=0}^{n} (G(x_{i+1}) - G(x_i))\phi(c_i).$$

If this limit exists, we denote it as $\int_a^b \phi(x)\,\mathrm{d}G(x)$. It can be proved that if ϕ is continuous and G is cumulative distribution function of a random variable which has density function g, then $\int_a^b \phi(x)\,\mathrm{d}G(x) = \int_a^b \phi(x)g(x)\,\mathrm{d}x$. For this reason, probabilists usually write the expectation of a random varaible $\phi(X)$ as

$$\mathbb{E}[\phi(X)] = \int_a^b \phi(x)\,\mathrm{d}F(x)$$

where $F(x)$ is distribution function of X. It is easy to verify that it is linear with respect to the integrators in the sense that

$$\int_a^b \phi(x)\,\mathrm{d}\big(\alpha G_1(x) + \beta G_2(x)\big) = \alpha \int_a^b \phi(x)\,\mathrm{d}G_1(x) + \beta \int_a^b \phi(x)\,\mathrm{d}G_2(x).$$

So, in order to know what happens when we substitute F by $\hat{F}_N$, we need to know that is the Riemann-Stieltjes with respect to a step function of the form $\mathbb{1}_{\{x_i \le t\}}$. It is clear that if t_1 and t_2 are both smaller that x_i or they are both larger than x_i, then $\mathbb{1}_{\{x_i \le t_2\}}(t_2) - \mathbb{1}_{\{x_i \le t_1\}}(t_1) = 0$. So we need to see what happens when we approach the discontinuity as $t_1 \to x_i^-$ and $t_2 \to x_i^+$. In this case, we will always have $\mathbb{1}_{\{x_i \le t_2\}}(t_2) - \mathbb{1}_{\{x_i \le t_1\}}(t_1) = 1$ and this will be preserved in the limit. It as easy exercise in calculus (I recommend to do it) to check that if

[4] This discussion is not really necessary, you may take (3.2.1) as the definition and skip it.

[5] A precise condition for this limit to exist is that the set of points of discontinuity has measure zero. But you may ignore that as it is not important for the discussion.

ϕ is continuous, we deduce from this discussion that

$$\int_a^b \phi(t)\,\mathrm{d}\big(\hat{F}_n(t)\big) = \frac{1}{N}\sum_{i=1}^{N}\int_a^b \phi(t)\,\mathrm{d}\mathbb{1}_{\{x_i\le t\}}(t) = \frac{1}{N}\sum_{i=1}^{N}\phi(x_i).$$

And hence we deduce the plug-in formula (3.2.1).

Observe that if we take ϕ as the indicator function of a (measurable) set. For example, $\phi(X) = \mathbb{1}_{X\in E}$, then $\theta_\phi = \mathbb{E}[\phi(X)] = \mathbb{P}(X \in E)$. So using indicator functions, Monte-Carlo can be used to compute the probability of events.

There is one additional advantage of the approach we are taking. Suppose that we want to consider instead functionals θ that are of the form $\theta_p(F) = F^{-1}(p)$, that is, θ is the p-th quantile. For example, if $p = 1/2$, the θ_p is the median. We can use the same framework to estimate quantiles and not only integrals. We will see more of this later.

We want to prove that plug-in estimator is unbiased and we want to compute its variance. Then, we want to deduce its asymptotic distribution using the central limit theorem, which will be helpful estimate the variance of our estimator.

Definition 3.1 En estimator $\hat{\eta}$ of η is called *unbiased* if $\mathbb{E}[\hat{\eta}] = \eta$.

Proposition 3.2 *The plug-in estimator $\theta(\hat{F}_N)$ is an unbiased estimator of $\theta(F)$.*

Proof. The proof is a direct computation

$$\mathbb{E}[\theta(\hat{F}_N)] = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\phi(X_i)\right] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[\phi(X_i)] = \mathbb{E}[\phi(X_i)] = \theta(F),$$

which proves the result. □

In principle, ϕ can be any function. A priori, there is no reason why to suppose the variance of $\phi(X)$ is well defined. From now on, let us supposed that the condition $\int |\phi(x)|^2 f(x)\,\mathrm{d}x < \infty$ holds (this will be the case for most applications we can think of.

We now give two results about the asymptotic behaviour of $\theta(\hat{F}_N)$.

Proposition 3.3 *The estimator $\theta(\hat{F}_N)$ converges (almost surely) to $\theta(F)$.*

Proof. The statement is simply a direct substitution in the statement of the strong law of large numbers. □

Lemma 3.4 *The plug-in estimator $\theta(\hat{F}_N)$ has variance*

$$\mathbb{V}[\theta(\hat{F}_N)] = \int |\phi(x)|^2 f(x)\,\mathrm{d}x - (\theta(F))^2,$$

in particular, our assumption $\int |\phi(x)|^2 f(x)\,\mathrm{d}x < \infty$ implies that $\mathbb{V}[\theta(\hat{F}_N)] < \infty$.

Proof. This is simply and application of the previous proposition and the standard formula $\mathbb{V}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$ used to simplify the computation of the variance. □

Proposition 3.5 *The quantity* $\sqrt{N}\left(\theta(\hat{F}_N) - \theta(F)\right)$ *converges (in distribution) to a normal random variable with mean* $\theta(F)$ *and variance* $\int |\phi(x)|^2 f(x)\,\mathrm{d}x - (\theta(F))^2$.

Proof. The condition $\mathbb{V}[\theta(\hat{F}_N)] < \infty$ guarantees that we can apply the central limit theorem. Again the statement of the proposition is simply a direct substitution in the content of the central limit theorem. □

3.3 Confidence intervals for the Monte-Carlo estimate

From Proposition Proposition 3.5, we conclude that

$$\theta(\hat{F}_N) \approx \mathcal{N}(\theta(F), \sigma^2/n) \tag{3.3.1}$$

where as before we have denoted $\theta(F) = \int \phi(x) f(x)\,\mathrm{d}x$ and $\sigma^2 = \int \phi^2(x) f(x)\,\mathrm{d}x - \theta^2(F)$. These suggest naturally a way to build asymptotic confidence intervals for the mean. The advantage of doing this, is that not only we can give an estimate of the quantity we want to know, but we can also give a region at which the real value must lie with probability α. Denote the $(1-\alpha)$-quantile of a standard normal random variable Z as $Z_\alpha = \inf\left\{x \mid \mathbb{P}(Z \geq x) \leq \alpha\right\}$ as illustrated in Figure 3.1. Observe that symetry of the density function of a normal distribution implies that $Z_{1-\alpha} = -Z_\alpha$.

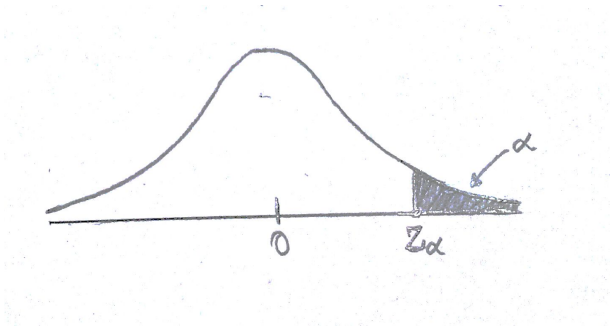

Figure 3.1: We denote Z_α as the $(1-\alpha)$-quantile of a standard normal distribution. In other words, the area accumulated behind the density function of a standard normal distribution to the right of Z_α is precisely α.

If $\theta(\hat{F}_N)$ followed exactly the distribution prescribed in (3.3.1), we would observe

$$\mathbb{P}\left(-Z_{\alpha/2} \leq \frac{\theta(\hat{F}_N) - \theta(F)}{\sigma/\sqrt{n}} \leq Z_{\alpha/2}\right) = 1 - \alpha$$

And by putting the previous inequalities in terms of $\theta(F)$ exclusively, we would have

$$\mathbb{P}\left(\theta(\hat{F}_N)-\sqrt{\frac{\sigma^2}{N}}Z_{\alpha/2}\leq\theta(F)\leq\theta(\hat{F}_N)+\sqrt{\frac{\sigma^2}{N}}Z_{\alpha/2}\right)=1-\alpha.$$

There is still one more problem we need to solve. We do not know the value of σ^2. The usual strategy to solve this is to estimate σ^2 along the way. If you have never done it before, be sure to solve the following (slightly tedious) exercise below.

Exercise 3.2 Suppose that $X_1,\ldots,X_N$ is a sequence of independent identically distributed random variables and let $\sigma^2=\mathbb{V}(X_i)$ and $\bar{X}=\frac{1}{N}\sum_{i=1}^N X_i$. Show that $\hat{S}_N^2\frac{1}{N-1}\sum_{i=1}(X_i-\bar{X})^2$ is a unbiased estimator of σ^2, that is, $\mathbb{E}[\hat{S}_N^2]=\sigma^2$. *Note:* Observe that we divide by $N-1$ and not N in the definition of $\hat{S}_N^2$. This result can be surprising as one would expect that diving by N is the logic option. However, this choice would yield a biased estimator.

By virtue of the previous exercise, one commonly seen strategy to define asymptotic estimators for the Monte-Carlo estimate $\theta(\hat{F}_N)$ is simply to replace σ^2 for its estimator $\hat{S}_N^2$ in our last attempt. With this choice, our final intervals for the real value $\theta(F)=\int\phi(x)f(x)\,\mathrm{d}x$ at the level $1-\alpha$ would be

$$\left(\theta(\hat{F}_N)-\sqrt{\frac{\hat{S}_N^2}{N}}Z_{\alpha/2},\theta(\hat{F}_N)-\sqrt{\frac{\hat{S}_N^2}{N}}Z_{\alpha/2}\right). \tag{3.3.2}$$

We summarise the above in the following proposition.

Proposition 3.6 (Monte-Carlo estimation with confidence intervals) *Let $X_1,\ldots,X_N$ be independent identically distributed with density f and $\phi:\mathbb{R}\to\mathbb{R}$ a (measurable) function. Let $\theta(F):=\mathbb{E}[\phi(X)]$ be the quantity we want to estimate. Then*

$$\lim_{N\to\infty}\mathbb{P}\left(\theta(\hat{F}_N)-\sqrt{\frac{\hat{S}_N^2}{N}}Z_{\alpha/2}\leq\theta(f)\leq\theta(\hat{F}_N)-\sqrt{\frac{\hat{S}_N^2}{N}}Z_{\alpha/2}\right)=1-\alpha. \tag{3.3.3}$$

Thus, whenever we give a Monte-Carlo estimate, we can given also give confidence intervals at any desired confidence level $(1-\alpha)$. In the next example, we give an example of a Monte-Carlo method with confidence intervals.

Example 3.7 (Estimation with confidence intervals) Suppose we want to compute the value of the integral

$$\int_0^2\sqrt{4-x^2}\,\mathrm{d}x. \tag{3.3.4}$$

The real value of this integral can be obtained immediately by noting the area below the curve $\sqrt{4-x^2}$ on the interval $[0,2]$ represents a quarter-circle of radius 2, so the exact value of the integral is π.

The first step we have to do is to rewrite (3.3.4) as an expectation of a random variable $X \sim \text{Unif}[0,2]$, which has density $f(x) = \frac{1}{2}\mathbb{1}_{[0,2]}(x)$:

$$\theta(F) = \int_0^2 \sqrt{4-x^2}\,\mathrm{d}x = \int_0^2 2\sqrt{4-x^2}\frac{1}{2}\,\mathrm{d}x = \mathbb{E}\left[2\sqrt{4-X^2}\right].$$

Now we can perform our Monte-Carlo estimates in R . We will modify our Montecarlo function to additionally receive an argument α and returns confidence intervals at the level $1-\alpha$.

```
require(plyr) # To handle data efficiently.

## Loading required package:  plyr

mc.intervals <- function(Phi, N, X.dens=runif, alpha=0.05){
  # FUN must be a function for which compute E(phi(X))
  # X.dens must be a function from which draw X. Must be a function of the
  # desired sample size.
  # N is a vector which contains different sample sizes for our estimate.
  # alpha determines the confidence intervals of level 1-alpha

  # Loop part, run for each element of N
  results.list <- lapply(N, function(nsim){
    # MonteCarlo step
    X <- sapply(FUN=X.dens, nsim) # N samples of the density of X
    PhiX <- sapply(X, Phi) # Evaluate phi at each X_i
    estim <- mean(PhiX) # Estimate of int_a^b \phi(x)f(x)df=E[phi(X_i)]
    S2 <- var(PhiX) # Estimate of the variance of phi(X_i)
    quant <- qnorm(alpha/2, lower.tail=FALSE) # Right quantile for alpha/2
    int.upper <- estim + sqrt(S2/nsim)*quant # Upper confidence interval
    int.lower <- estim - sqrt(S2/nsim)*quant # Lower confidence interval
    return(data.frame(N=nsim, Estimate=estim, LI=int.lower, UI=int.upper))
    # -------
  })
  #
  results.table <- ldply(results.list) # Assembles list in data.frame
  return(results.table)
}
```

We now apply our function two our particular problem.

```
set.seed(110104)
Phi <- function(x) 2*sqrt(4-x^2)
X.dens <- function(nsim) runif(nsim, 0, 2)
N <- seq(from=1000, to=10000, by=1000)
data <- mc.intervals(Phi=Phi, N=N, X.dens=X.dens)
data

##         N Estimate       LI       UI
## 1    1000 3.104285 3.048454 3.160116
## 2    2000 3.175281 3.136471 3.214092
## 3    3000 3.155253 3.123016 3.187491
## 4    4000 3.146294 3.118335 3.174252
## 5    5000 3.134197 3.109278 3.159116
## 6    6000 3.122012 3.099035 3.144989
## 7    7000 3.146445 3.125484 3.167405
## 8    8000 3.118658 3.098615 3.138700
## 9    9000 3.137457 3.119093 3.155821
## 10 10000 3.138538 3.121055 3.156021
```

It is nice to be able to visualise the results.

```
require(ggplot2)
ggplot(data, aes(x=N)) +
  geom_ribbon(aes(ymin=LI, ymax=UI), fill="grey", alpha=.4) +
  geom_line(aes(y=Estimate),  colour="blue") +
  geom_hline(aes(yintercept=pi) , colour="red", linetype="dotted", size=1)
```

We see that the reduction of the error is in fact considerably slow. That is why several improvements over this 'crude' version of Montecarlo simulations is need. In ■

We close this section with an alternative derivation of confidence intervals for the mean. We include this for completeness, although there is no improvement when using these intervals instead of the asymptotic intervals we gave before. Observe that they can only be used when we want to build intervals for the mean of normal random variables. Many people use them whenever the distribution is symmetric, even if it is not normal. For large samples, it makes almost no difference. In the next section we compare Montecarlo with other numerical integration methods.

Intervals for the mean of a normal distribution with unknown variance* [Optional]. In classical statistics there is a well known to build confidence intervals when sampling from

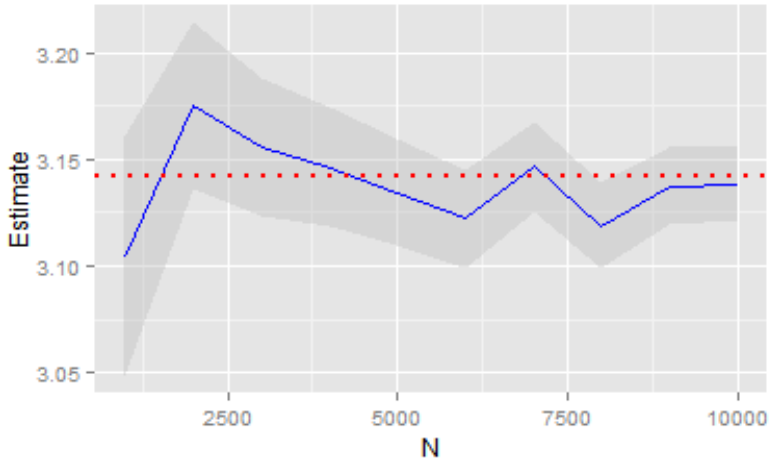

Figure 3.2: The shaded are represents the confidence intervals at 95%. The narrow down (slowly) as the number of simulations increases. The red line is the computer's value of π.

a normal distribution and the variance is unknown. In contrast with the intervals giving in the previous subsection, these intervals are not asymptotic. They rely on the technique of *pivotal quantities*. The resulting intervals look similar to the ones given in (**??**), but the use a t-student distributions instead.

Using these exact intervals instead (**??**) can be very important when sampling from a normal distribution and the sample size is small (usually $N \leq 30$). They are not used very often for Monte-Carlo simulation because our sample rarely follow a normal distribution or a similar symmetric distribution and, most importantly, sample sizes tend to be huge, so there is no necessity. However, if the reader opts for using these type of intervals instead. The result will be almost the same.

In order to derive confidence intervals, we will need to use the t-student distribution. The easiest way to define is the following.

Definition 3.8 A random variable T is said to be distributed *t-student* with k degrees of freedom if it can be written in the form

$$T = \frac{Z}{\sqrt{V/k}}$$

where Z is a standard normal random variable and V is a χ^2-random variable with k degrees of freedom. *Observation*: It might also be pertinent to recall that one way to define the χ^2-squared distribution with k degrees of freedom is as the distribution of the sum of the squares of k-independent standard normal random variables.

We will now use the t-student distribution to construct confidence intervals for $\theta(F)$

using the Monte-Carlo estimation. Observe that for all $i = 1, \dots, N$, we have that

$$\phi(X_i) - \theta(\hat{F}_N)$$

has mean zero and variance $\mathbb{V}[X]$.

Definition 3.9 Let $X_1, \dots, X_N$ be independent identically distributed random variable with mean μ and variance σ^2. The *unbiased estimator of the mean* is the *sample mean* $\overline{X}_N := \frac{1}{N}\sum_{i=1}^N X_i$. The *unbiased estimator of the variance* is $\hat{S}_N^2 = \frac{1}{N-1}\sum_{i=1}^N (X_i - \overline{X}_N)^2$. As their name suggests, we can prove that $\mathbb{E}[\overline{X}_N] = \mu$ and $\mathbb{E}[\hat{S}_N^2] = \sigma^2$.

Exercise 3.3 The term $N-1$ in the definition of S^2 can be very surprising. Our intuition could suggest that an unbiased estimator would be the *sample second moment* $\frac{1}{N}\sum_{i=1}^N (X_i - \overline{X}_N)^2$. Prove that this estimator is biased by showing that $\mathbb{E}[\sum_{i=1}^N (X_i - \overline{X}_N)^2] = (N-1)\mathbb{V}[X_i]$.

The following result is very useful when trying to describe asymptotically the behaviour of the estimator S_N^2.

Proposition 3.10 *Let $X_1, \dots, X_N$ be independent identically distributed standard normal random variable with mean μ and variance σ^2. Then*

$$\frac{N-1}{\sigma^2}\hat{S}_N^2$$

has distribution χ^2 with $(N-1)$-degrees of freedom.

Proof. This proof is given in every introductory course in mathematical statistics. Unfortunately, it requires several computations. I will only sketch the idea of the proof. Please consult [HC04].

Inspired by the previous proposition. Define now

$$T = \frac{\theta(\hat{F}_N) - \theta(F)}{\sqrt{\frac{1}{N-1}\sum_{i=1}^N (\phi(X_i) - \theta(F))^2}} = \frac{\theta(\hat{F}_N) - \theta(F)}{\sqrt{\hat{S}_N^2/N}}. \tag{3.3.5}$$

follows a t-student distribution with $(N-1)$ degrees of freedom. Moreover, it is a *pivotal quantity* for $\theta(F)$ (meaning that we know its distribution and we can use it to approximate our parameter of interest).

The t-student distribution is symmetrical. The following example shows different plots of the density function of the t-student distribution.

Example 3.11 We show the plot of the density function of the t-student distribution for different degrees k of freedom. When $k = 1$, it is known as the Cauchy distribution. As $k \to \infty$ it is known to approximate a normal distribution. The function in `R` to obtain the density function of a t-student is `dt`.

```
library(ggplot2)
domain <- data.frame(x=c(-4,4))
ggplot(domain, aes(x)) +
  stat_function(fun = dt, args = list(df = 1), aes(colour = " 1"))  +
  stat_function(fun = dt, args = list(df = 2), aes(colour = " 2"))  +
  stat_function(fun = dt, args = list(df = 5), aes(colour = " 5"))  +
  stat_function(fun = dt, args = list(df = 10), aes(colour = "10"))  +
  stat_function(fun = dnorm, aes(colour = "Standard normal N(0,1)"))  +
  scale_colour_manual("Degrees of freedom", values = c("red", "blue",
    "green", "orange", "gray")) + xlab("") + ylab("")
```

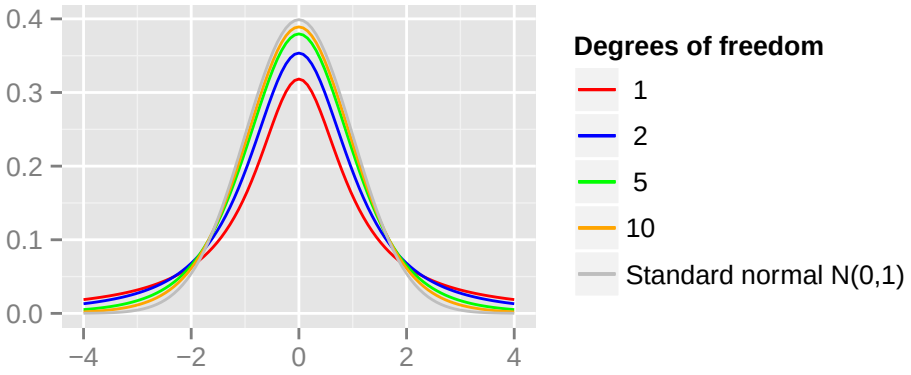

Figure 3.3: Density function of a t-student with k degrees of freedom.

Alternatively, we could have simulated a standard normal, k independent χ^2 random variables (by simulating k normal standard and squaring) and compute the formula for the t-student. We see in Figure 3.3 how the t-student distributions approximates a standard normal distribution. ■

Let $\alpha \in (0,1)$ an denote $t_{k,\alpha/2}$ the unique value such that $\mathbb{P}(T > t_{k,\alpha/2}) = \alpha/2$ for T distributed t-student with k degrees of freedom. By the symmetry of the density function of a t-student, we have $\mathbb{P}(T < -t_{k,\alpha/2}) = \alpha/2$. Therefore, from (3.3.5) we conclude

$$\mathbb{P}\left(-t_{N-1,\alpha/2} \leq \frac{\theta(\hat{F}_N) - \theta(F)}{\sqrt{\hat{S}_N^2/N}} \leq t_{N-1,\alpha/2}\right) = 1-\alpha$$

or equivalently

$$\mathbb{P}\left(\theta(\hat{F}_N) - t_{N-1,\alpha/2}\sqrt{\hat{S}_N^2/N} \leq \theta(F) \leq \theta(\hat{F}_N) + t_{N-1,\alpha/2}\sqrt{\hat{S}_N^2/N}\right) = 1-\alpha \tag{3.3.6}$$

which gives us asymptotic confidence intervals for $\theta(F)$ of level $1-\alpha$.

3.4 Montecarlo vs Numerical integration

As promised, we will show how Montecarlo is robust and does not suffer of the curse of dimensionality like other numerical integration methods. We will compare with the trapezoidal rule only. In the previous section we showed that the error of the 'crude' Montecarlo method goes to zero at a rate $O(\sqrt{N})$. According to what we saw in previous sections, the error rate of the Trapezoidal rule in one dimension is $O(N^{-2})$, so it decreases a lot faster. However, in higher dimensions d it is $O(N^{-2/d})$, so the Montecarlo method is considerably better.

To exemplify this, we will try to compute a very highdimensional probability. Suppose we want to know the value of the integral

$$\int_{-2}^{2}\cdots\int_{-2}^{2}\frac{1}{(2\pi)^{-m/2}}\exp\left\{-\frac{1}{2}(x_1^2+\ldots x_m^2)\right\}\mathrm{d}x_1\ldots\mathrm{d}x_m. \tag{3.4.1}$$

It is a very complicated m-dimensional integral. However we chose it smartly because we know a good approximation of its real value. The function we are trying to integrate is the joint density function of m independent standard normal random variables $Z_1,\ldots,Z_m$. In particular, it has the value $\mathbb{P}(-2\leq Z_1\leq 2)^m$.

We will break our comparison into different steps: *First*: we will program a recursive program to use the trapezoidal rule for higher dimensional integrals.

```
trapezoid <- function(N, a, b, FUN){
  # FUN is the function we one to integrate FUN: R^dim-->R
  # a,b are dim-dimensional vectors specifying the limits of integration.
  # Each (ai,bi), aiEb, biEb, will be partitioned in N segments of same size.
  dim <- length(a)
  x <- seq(a[1], b[1], (b[1]-a[1])/N)
  if (dim==1){
    fi <- sapply(x, FUN)
  } else{
    fi <- sapply(x, function (x){
      trapezoid(N, a[-1], b[-1], function(y) FUN(c(x,y)))
    })
```

```
  }
  return(((b[1]-a[1])/(2*N))*sum(fi[-1]+fi[-(N+1)]))
}
# Example: the volume of a sphere with n=20:
trapezoid(20, c(-1,-1,-1), c(1, 1, 1), function(x) {sum(x^2)<=1})

## [1] 4.154

# real volume
(4/3)*pi

## [1] 4.18879
```

In order to evaluate the integral 3.4.1, we must write as an expectation. A general way to do this is to use a uniform distribution on the integrating region and divide the objective function by the volume of the integrating region (since the uniform distribution has density the reciprocal of the volume times the indicatorX of the region). So (3.4.1) becomes

$$\int_{-2}^{2}\cdots\int_{-2}^{2}\frac{4}{(2\pi)^{-m/2}}\exp\left\{-\frac{1}{2}(x_1^2+\ldots x_m^2)\right\}\frac{1}{4^m}\,\mathrm{d}x_1\ldots\mathrm{d}x_m=\mathbb{E}\left[\frac{4^m}{(2\pi)^{m/2}}e^{-\frac{1}{2}X^\top X}\right],$$

where $X\sim\text{Unif}([-2,2]^m)$.

We will compare the performance of MonteCarlo and the Trapezoid rule in dimensions 1 and 5. For higher dimensions the problems becomes very hard to compute. We will maintain constant the number of simulations. For example, in dimension 5, the trapezoidal rule that divides each interval in N subintervals, requires N^5 evaluations, so we compare with MonteCarlo with N^5 simulations. We see that performance of the Trapezoidal rule is better when the dimension is one but a worst when the dimension is 5.

```
options(digits=4)
set.seed(1011)
res <- data.frame() # we will store here all the results
# different dimension sizes
for (m in c(1, 5)){
  real.value <- (pnorm(2)-pnorm(-2))^m
  # different partition sizes (trapezoid)/ sims (montecarlo)
  for (N in 2:7){ # Recall trapezoid needs nsim=N^m points.
    FUN <- function(x) (2*pi)^(-m/2)*exp((-1/2)*t(x)%*%x)
    a <- rep(-2, m)
    b <- rep(2, m)
```

```
    estim.num <- trapezoid(N, a, b, FUN)
    # X.dens must be a list of m-dimensional vector of Unif(-2,2).
    X.dens <- function(nsim) replicate(nsim, runif(m, -2, 2), simplify=FALSE)
    Phi <- function(x) (4^m)*FUN(x)
    # For M^c we evaluate in N^m points intead, so that it is comparable.
    estim.mc <- mc.intervals(Phi, N^m, X.dens)
    res.temp <- data.frame(Dim=m, Nsim=N^m, Real=real.value,
                           Num=estim.num,
                           Err.Num=abs(estim.num-real.value),
                           MC=estim.mc$Estimate,
                           MC.LI=estim.mc$LI, MC.UI=estim.mc$UI,
                           Err.MC=abs(estim.mc$Estimate-real.value)
    )
    res <- rbind(res, res.temp)
  }
}
res
```

```
##    Dim  Nsim   Real    Num  Err.Num     MC  MC.LI  MC.UI    Err.MC
## 1    1     2 0.9545 0.9059 0.048633 0.5584 0.1786 0.9382 0.3960915
## 2    1     3 0.9545 0.9238 0.030650 1.0007 0.9156 1.0858 0.0462010
## 3    1     4 0.9545 0.9369 0.017625 0.6353 0.4294 0.8413 0.3191558
## 4    1     5 0.9545 0.9431 0.011377 1.2246 0.8565 1.5928 0.2701400
## 5    1     6 0.9545 0.9466 0.007933 1.3456 1.2683 1.4229 0.3910864
## 6    1     7 0.9545 0.9487 0.005842 1.3299 1.0900 1.5697 0.3753764
## 7    5    32 0.7923 0.6100 0.182293 0.9342 0.3810 1.4873 0.1419072
## 8    5   243 0.7923 0.6730 0.119295 0.7314 0.6057 0.8571 0.0608893
## 9    5  1024 0.7923 0.7218 0.070496 0.7932 0.7298 0.8567 0.0009624
## 10   5  3125 0.7923 0.7462 0.046105 0.8335 0.7944 0.8727 0.0412543
## 11   5  7776 0.7923 0.7599 0.032383 0.7927 0.7687 0.8167 0.0004024
## 12   5 16807 0.7923 0.7683 0.023952 0.7869 0.7707 0.8032 0.0053345
```

In Figures 3.4 and 3.5 we show how the errors compare.

```
ggplot(subset(res, Dim==1), aes(x=Nsim)) +
  geom_line(aes(y=Err.Num, colour="Trapezoidal")) +
  geom_line(aes(y=Err.MC, colour="MonteCarlo")) +
  scale_colour_manual("Error", values=c("red", "blue"))
```

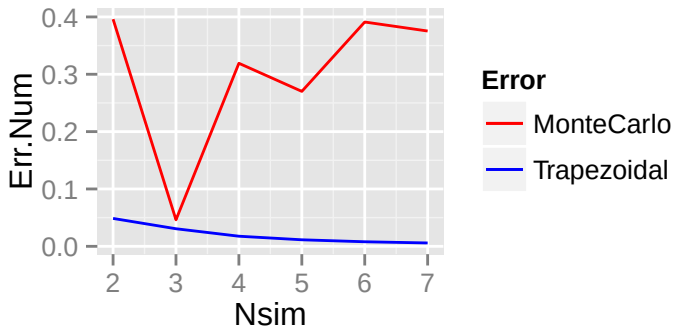

Figure 3.4: Error comparison in dimension 1

```
ggplot(subset(res, Dim==5), aes(x=log(Nsim,5))) +
  geom_line(aes(y=Err.Num, colour="Trapezoidal")) +
  geom_line(aes(y=Err.MC, colour="MonteCarlo")) +
  scale_colour_manual("Error", values=c("red", "blue"))
```

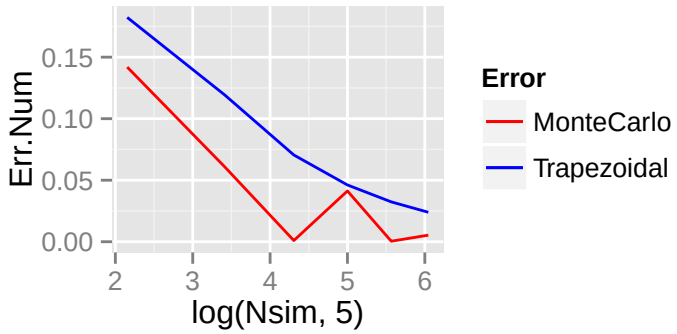

Figure 3.5: Error comparison in dimension 5

There is, however, a much more worst problem than just the accuracy. If we want to evaluate a 200 dimensional integral using this method, we would require at least 2^200 evaluations in order to use a numerical integration method! This is beyond ridiculous! With Monte Carlo integration we can much more easily handles this huge integrals.

3.5 Importance sampling

The variance of an unbiased estimator is a measure of its speed of convergence towards the desired value as a function of the sample size. In this section and the next we review

methods to reduce the variance of the Monte Carlo method we have studied so far, and which is often called *crude Monte Carlo*, as it does not incorporate any variance reduction addition.

The idea of importance sampling is that sometimes computing an integral is done more or less efficiently according to what distribution we are sampling from. It is easy to see how the Monte Carlo methodology can be applied in different ways to the same problem: Suppose X and Y are random variables supported in $[a,b]$ with density functions f and g respectively. Assume further that U is uniformly distributed in $[a,b]$ and let ϕ be any function and suppose we want to compute $\int_a^b \phi(x)\,\mathrm{d}x$, then this integral can be written in three different ways

$$\int_a^b \phi(x)\,\mathrm{d}x = \mathbb{E}_U[(b-a)\phi(U)] = \mathbb{E}_X[\phi(X)/f(X)] = \mathbb{E}_Y[\phi(Y)/g(Y)].$$

Thus, we could simulate a sample from either U, X or Y and average the corresponding function inside the expectation in order to approximate the integral's real value. The question is, what strategy is the most efficient one? Importance sampling solves this question. Observe that the same idea applies if instead a bounded interval $[a,b]$ we consider infinite intervals. Before proving the same result, we will give an example of the application. The idea will be to sample more where the integrand is higher in absolute value. Intuitively, this works as we are sampling more from the points that contribute more to the integral.

Example 3.12 (Beta distribution vs uniform distribution) Suppose we want to compute the integral

$$I = \int_0^1 100(1-x)^{99}\,\mathrm{d}x$$

which has real value its $I = 1$. A feature of this integral is that the integrand approaches 0 very rapidly in $(0,1)$. One can do Monte Carlo by sampling from a uniform distribution in $[0,1]$ and averaging the function $100(1-x)^{99}$ at those sample values. Another approach would be to simulate from any other density g supported in $[0,1]$ and average the function $100(1-x)^{99}/g(x)$ instead.

For this example, we will use a Beta distribution, which is a type of distribution depending on two parameters α and β and which is supported in $[0,1]$. The density function $\mathrm{Beta}(\alpha,\beta)$ is given by

$$g(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}\mathbb{1}_{[0,1]}(x).$$

The choice of α and β determines the shape of g. For instance, the graph of g with $\alpha = 1$ and $\beta = 20$ is shown in Figure 3.6. The Beta density function is invoked in `R` with the command `dbeta`. Similarly, observations of a Beta distribution can be simulated with the command `rbeta` (or you can use the acceptance-rejection method to generate them yourself from a

uniform distribution).

```
require(ggplot2)
ggplot(data.frame(x=c(0,1)), aes(x)) +
  stat_function(fun=function(x) dbeta(x,shape1=1, shape2=10)) + ylab("Density")
```

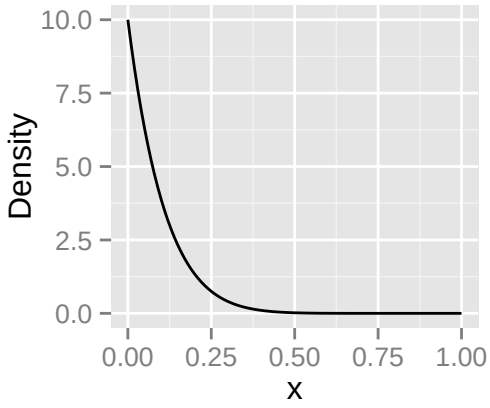

Figure 3.6: Density function of a Beta distribution with parameters $\alpha = 1$ and $\beta = 20$.

If we interchange the values of α and β then we get a graph that is reflected about the line $y = 0.5$.

We will compute three different approximations of the integral using a uniform random variable in $(0, 1)$, a beta random variable with $\alpha = 1$ and $\beta = 20$ and another beta random variable with $\alpha = 20$ and $\beta = 1$. So far, we have not seen which method should be the most efficient, but as we shall see in what follows, the best one should be the second one as it is sampling more from the points were the integrand is higher. For this reason, the third one should be the worst.

```
options(digits=6)
```

```
set.seed(110104)
nsim <- c(100, 500, 1000)
EstimUnif <- EstimBeta1 <- EstimBeta2 <- numeric(3)
for (i in 1:3){
    U <- runif(nsim[i], 0, 1)
    X <- rbeta(nsim[i], shape1=1, shape2=20)
    Y <- rbeta(nsim[i], shape1=20, shape2=1)
```

```
    EstimUnif[i] <- mean(100*(1-U)^(99))
    EstimBeta1[i] <- mean(100*(1-X)^(99)/dbeta(X, shape1=1, shape2=20))
    EstimBeta2[i] <- mean(100*(1-Y)^(99)/dbeta(X, shape1=20, shape2=1))
}
data.frame(Simulations=nsim, Uniform=EstimUnif,
     Beta1_20=EstimBeta1, Beta20_1=EstimBeta2)

##   Simulations  Uniform Beta1_20    Beta20_1
## 1         100 2.317355 0.853535 2.03751e-25
## 2         500 0.721314 0.875119 2.09491e-06
## 3        1000 0.818633 0.992148 6.64327e+05
```

In the rest of the section we will formalise this intuition. ■

Let us recover the notation of the previous section and suppose we want to estimate a quantity of the form

$$\theta(F) = \int_{\mathbb{R}} \phi(x) f(x)\,\mathrm{d}x = \mathbb{E}_X[\phi(X)].$$

Let g be any other density function and $X_1, \ldots, X_N$ a sample of independent identically distributed random variables. The only assumption we will do is that $g(x) \neq 0$ whenever $\phi(x)f(x) \neq 0$; this is true, in particular, whenever f and g have the same support.

Define the *importance weights* according to g as

$$\omega_i := f(X_i)/g(X_i)$$

and define the importance sampling estimator as

$$\hat{\theta}_g = \frac{1}{N}\sum_{i=1}^{N} \omega_i \phi(X_i). \tag{3.5.1}$$

What we are doing is giving different weights to each observation $\phi(X_i)$ according to the ratio f/g. The word weight could be misleading, we do NOT have that $\sum_i \omega_i = 1$ as it happens often with weights. Where do these mysterious weights come from? Suppose X has density f, then

$$\mathbb{E}_X[\phi(X)] = \int \phi(x) f(x)\,\mathrm{d}x = \int \frac{f(x)\phi(x)}{g(x)} g(x)\,\mathrm{d}x = \mathbb{E}_Y\left[\frac{f(Y)\phi(Y)}{g(Y)}\right]$$

where Y is some random variable having density g.

Proposition 3.13 *The quantity $\hat{\theta}_g$ is an unbiased estimator of $\theta(F)$.*

Note: here the expected value must be considered with respect to the density g as our simulated sample has common density g.

Proof. We just compute

$$\begin{aligned}\mathbb{E}_g[\hat{\theta}_g] &= \mathbb{E}_g\left[\frac{1}{N}\sum_{i=1}^{N}\omega_i\phi(X_i)\right]\\ &= \mathbb{E}_g[\omega_1\phi(X_1)]\\ &= \mathbb{E}_g\left[\frac{f(X_1)}{g(X_1)}\phi(X_1)\right]\\ &= \int \phi(x)\frac{f(x)}{g(x)}g(x)\,\mathrm{d}x\\ &= \int \phi(x)f(x)\,\mathrm{d}x\\ &= \theta(F).\end{aligned}$$

So the estimator is unbiased. □

Corollary 3.14 *The variance of θ_g is given by*

$$\mathbb{V}[\theta_g] = \frac{1}{N}\left(\int \frac{f^2(x)\phi^2(x)}{g(x)}\,\mathrm{d}x - \left[\theta(F)\right]^2\right). \tag{3.5.2}$$

Proof. This is just the usual formula $\mathbb{V}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$ simplified. It is left as exercise. Remember that the variance of the sum on independent random variables is the sum of the variances. Using also that the variance takes out a constant squared, we have $\mathbb{V}[\theta_g] = \frac{1}{N^2}\mathbb{V}[\sum_{i=1}^{N}\omega_i\phi(X_i)] = \frac{1}{N}\mathbb{V}[\omega_1\phi(X_1)]$. □

From the above corollary, we now deduce that the intuition 'sample more from higher values of the integrand' is correct.

Proposition 3.15 *The density function g^* that minimises the density of the estimator θ_g is given by*

$$g^*(x) = \frac{f(x)|\phi(x)|}{\int f(s)\phi(s)\,\mathrm{d}s}$$

Proof. This is really a consequence of the previous corollary. Observe that the quantity (3.5.2) is maximised whenever $\int \frac{f^2(x)\phi^2(x)}{g(x)}\,\mathrm{d}x$ is maximised. This is a variational problem, which in general are not easy to solve. But in this case, Jensen's inequality from probability

gives

$$\begin{aligned}\int \frac{f^2(x)\phi^2(x)}{g(x)}\,\mathrm{d}x &= \mathbb{E}_g\left[\frac{f^2(X_1)}{g^2(X_1)}\phi^2(X_1)\right] \\ &\geq \left(\mathbb{E}_g\left[\frac{f(X_1)}{g(X_1)}|\phi(X_1)|\right]\right)^2 \\ &= \left(\int |\phi(x)|f(x)\,\mathrm{d}x\right)^2.\end{aligned}$$

And thus, substituting in (3.5.2) we get

$$\mathbb{V}_g[\theta_g] \geq \frac{1}{N}\left(\left(\int |\phi(x)|f(x)\,\mathrm{d}x\right)^2 - \left[\theta(F)\right]^2\right)$$

and we see that this lower bound is realised precisely when g is the g^* proposed. □

In practice, sampling from g^* is virtually impossible. Nonetheless, importance is used very often in practice in the form: sample more from the values where the integrand is higher (in absolute value). At this point, you should be convinced that this is what we did we chose the Beta function with paremeters $\alpha = 1$ and $\beta = 20$ in Example 3.12.

Index

Bibliography

[Bar76] R. Bartle. *The Elements of Real Analysis*. 2nd ed. Wiley, 1976.

[Eck87] R. Eckhardt. “Stan Ulam, John Von Neumann and the Monte Carlo Method”. In: *Los Alamos Science Special Issue* (1987), pp. 131–136.

[HC04] R. Hogg and A. Craig. *Introduction to Mathematical Statistics*. Pearson, 2004. Chap. 4.

[HD62] T. E. Hull and A. R. Dobell. “Random Number Generators”. In: *SIAM Rev* 4.3 (1962), pp. 230–254.

[Leh43] D. Lehmer. “Mathematical methods in large-scale computing units”. In: *Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery* (1943), pp. 141–146.