Monte Carlo Simulation

with applications to mathematical finance

Johan van Leeuwaarden
Jorn van der Pol
September 2011

Monte Carlo Simulation

with applications to mathematical finance

September 2011

Johan van Leeuwaarden
Jorn van der Pol

Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands

Introduction

Mathematical modelling traditionally focussed on realistic yet tractable models. It was often the case that a simple model was favoured over a complex but more realistic one, since the latter, more complex model was harder to analyse mathematically. Nowadays, fast computers are available and more complex models can be analysed numerically. One of the approaches in this area is simulation.

In these lecture notes, we provide the basic techniques behind a simple simulation tool known as Monte Carlo simulation. Monte Carlo simulation relies on repeated random sampling to compute their results. They are widely used in different areas in mathematics and physics such as fluids, cellular structures, queueing theory and risk theory.

These lecture notes come with many examples written in the statistical programming language R. The theoretical background of Monte Carlo simulation may not be very hard; however, simulation building requires some practice. The vast majority of the theory and problems in these lecture notes stem from the areas of mathematical finance and risk theory. Practical problems are indicated with (P), while theoretical problems are indicated with (T).

Although R is generally used as a statistical package, it is suitable for writing simulations and analysing their results as well. A short introduction to R, along with many examples, can be found on the website http://www.student.tue.nl/V/j.g.v.d.pol/Teaching/R.

We would like to thank Marko Boon and Harry van Zanten for their valuable comments and suggestions.

Eindhoven,
September 2011

Johan van Leeuwaarden
Jorn van der Pol

Contents

1 Monte Carlo simulation

If you throw a fair coin many times, you expect that the fraction of heads will be about 50%. In other words, if we see the tossing of coins as Bernoulli experiments and we define random variables Z_i to be equal to 1 if heads occurs, and 0 if tails occurs, we expect the sample mean, $\overline{Z}_n := (Z_1 + Z_2 + \ldots + Z_n)/n$, to be close to the theoretical mean, $\mathbb{E}[Z_i] = 1/2$.

The law of large numbers states that this holds in a general setting: if $Z_1, Z_2, \ldots Z_n$ are i.i.d. random variables with mean $z := \mathbb{E}[Z_1]$ and finite variance, the probability of the *sample mean* being close to z is large. In fact, for every $\varepsilon > 0$,

$$\lim_{n\to\infty} \mathbb{P}\left(\left|\frac{Z_1 + Z_2 + \ldots + Z_n}{n} - z\right| < \varepsilon\right) = 1. \tag{1.1}$$

Now suppose that we have a method to obtain i.i.d. outcomes $Z_1, Z_2, Z_3, \ldots$, while we do not know the value of z. Think for example of a coin with an unknown probabilty z of throwing heads. The discussion above suggests using the sample mean

$$\overline{Z} \equiv \overline{Z}_n := \frac{Z_1 + Z_2 + \ldots + Z_n}{n} \tag{1.2}$$

as an estimator for z. This method is known as *Monte Carlo simulation* (after the famous city with many casinos). As we will see, many quantities of interest can be expressed as an expectation and can therefore be estimated using Monte Carlo simulation.

First we will define some terminology that will often be used in these lecture notes. Each of the random variables above is called a *replication*, or the result of a *run*. The number n that appears in Equation (1.2) is called the *number of independent runs*, or the *number of independent replications*.

Example 1.0.1 (Estimation of mean and tail probability)**.** Consider $X \sim \text{Exp}(10)$ and suppose that we are interested in the expected value of X. We know, of course, that $\mathbb{E}[X] = 0.1$, but this is a nice example to illustrate the Monte Carlo technique. If we are able to simulate

$X_1, X_2, \ldots, X_n \sim \text{Exp}(10)$, it is intuitively clear that we can use the sample mean as an estimator for $\mathbb{E}[X]$ and that this estimate will be better if n gets larger.

In Figure 1.1, fifteen estimates are plotted for each number of runs in $20, 40, 60, \ldots, 2000$. As you can see, the spread among the estimates decreases as the number of runs increases. Hence, estimates become better as the number of runs they are based on increases. See Listing 1.1 for the code we used to estimate $\mathbb{E}[X]$ based on a fixed number of runs.

Listing 1.1: Estimating $\mathbb{E}[X]$.

```
1  # The number of independent replications.
2  runs <- 1000;
3
4  # Generate runs EXP(10) distributed random variables.
5  z <- rexp(runs, 10);
6
7  # Calculate estimator of expectation.
8  est <- mean(z);
```

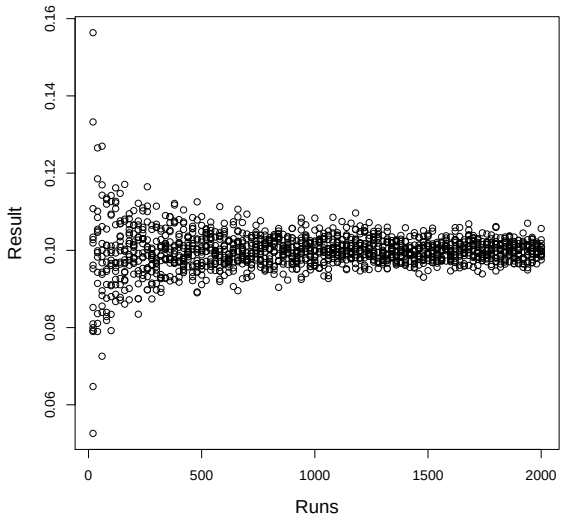


Figure 1.1: Plot of the spread in estimators against the number of independent runs they are based on.

It is also possible to estimate the probability that $X > 0.15$. If we denote by $\mathbf{1}_{\{\cdot\}}$ the indicator function, that is, $\mathbf{1}_{\{A\}} = 1$ if A holds, and $\mathbf{1}_{\{A\}} = 0$ otherwise, then $\mathbf{1}_{\{A\}}$ is a Bernoulli random variable and $\mathbb{E}\left[\mathbf{1}_{\{X>0.15\}}\right] = \mathbb{P}(X > 0.15)$. Hence, if we let $Y_i = \mathbf{1}_{\{X_i>0.15\}}$, the sample mean $\overline{Y} = (Y_1 + Y_2 + \ldots + Y_n)/n$ is an estimator for $\mathbb{P}(X > 0.15)$. The R source code for this situation can be found in Listing 1.2.

Remark: In R, it is not necessary to replace `TRUE` by 1 and `FALSE` by 0: this is done automatically, when you want to calculate with these values. E.g., `mean(c(TRUE,FALSE))` evaluates to $\frac{1}{2}$ and `sum(c(TRUE, FALSE, TRUE))` evaluates to 2. From

now on, we will not explicitly translate logical values to zeroes and ones.

Listing 1.2: Estimating $\mathbb{P}(X > 0.15)$.

```
# The number of independent replications.
runs <- 1000;

# Generate runs EXP(10) distributed random variables.
z <- rexp(runs, 10);

# Replace all entries > 0.15 by TRUE
# entries <= 0.15 by FALSE.
y <- (z>0.15);
# Replace TRUE by 1 and FALSE by 0
y[y==TRUE] <- 1;
y[y==FALSE] <- 0;

# Calculate estimator of tail probability.
est <- mean(y);
```

Exercise 1.0.1 (P)**.** Suppose that $X_1, X_2, \ldots, X_{10}$ are independently distributed according to the normal distribution with mean $\mu = 1$ and variance $\sigma^2 = 1$. Use simulation to estimate the mean of $X_1 + X_2 + \ldots + X_{10}$.

Solution:

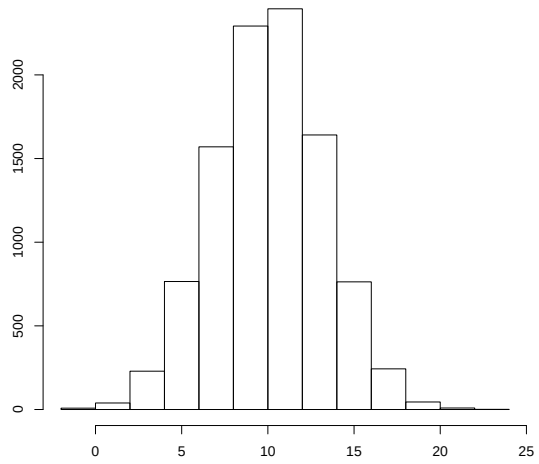


Figure 1.2: Histogram based on 10,000 independent simulation runs.

Sample source code can be found in Listing 1.3.

Since $X_1+X_2+\ldots+X_{10}$ is the sum of ten random variables with mean 1, its expected value is 10. Figure 1.2 depicts a histogram based on 10,000 independent simulation runs. As you can see, the values are centered around the value 10, but some runs yield a result that is much smaller or larger than 10.

Listing 1.3: Estimating $\mathbb{E}[X_1 + X_2 + \ldots + X_{10}]$ in Exercise 1.0.1.

```
runs <- 1000; # Number of runs
mu <- 1; # Mean of X_i
sigma <- 1; # Standard deviation (!) of X_i

res <- c(); # Results

for(run in 1:runs)
{
    # Draw ten X_i
    rnd <- rnorm(10, mu, sigma)
    # Add their sum to the results vector
    res <- c(res, sum(rnd));
}

# Calculate sample mean
mean(res)
```

Exercise 1.0.2 (P)**.** Let Y be uniformly distributed on the random interval $[0, X]$, where X follows an exponential distribution with parameter $\lambda = 1$. Use simulation to calculate $\mathbb{E}[Y]$ and $\mathbb{P}(Y > 1)$.

Solution: In each run of the simulation, two random variables have to be drawn. We first draw X, which is distributed according to the exponential distribution with intensity λ. If x is the realization of X, in the next step the random variable Y has to be drawn from the uniform distribution on $[0, x]$. Repeating this algorithm a suitable number of times and taking the average of all the outcomes, yields an estimate for $\mathbb{E}[Y]$.

The quantity $\mathbb{P}(Y > 1)$ can be calculated in much the same way, but here we use as estimator Z, where $Z = 1$ if $Y > 1$ and $Z = 0$ otherwise. See Listing 1.4.

The true values of $\mathbb{E}[Y]$ and $\mathbb{P}(Y > 1)$ can be calculated by conditioning on X. We find

$$\mathbb{E}[Y] = \mathbb{E}[\mathbb{E}[Y \mid X]] = \mathbb{E}\left[\frac{1}{2}X\right] = \frac{1}{2}; \tag{1.3}$$

and

$$\mathbb{P}(Y > 1) = \int_{x=0}^{\infty} \mathbb{P}(Y > 1 \mid X = x) f_X(x) \mathrm{d}x = \int_{x=1}^{\infty} \frac{x-1}{x} \mathrm{e}^{-x} \mathrm{d}x, \tag{1.4}$$

since $\mathbb{P}(Y > 1 \mid X = x) = 0$ if $x < 1$ and $\mathbb{P}(Y > 1 \mid X = x) = (x-1)/x$ otherwise. The integral on the right cannot be calculated explicitly (which may be a reason to use simulation!), but it is approximately equal to 0.1484.

Various estimates, based on 10,000 independent runs, can be found in Table 1.1.

Exercise 1.0.3 (P)**.** The birthday problem is the seemingly paradoxical result that in a group of only 23 people, the probability that two people share the same birthday is approximately 50%. Use simulation to show that this is indeed the case. Also, write a simulation to estimate the probability that in a group of n people, at least two people share the same birthday.

Listing 1.4: Estimating $\mathbb{E}[Y]$ and $\mathbb{P}(Y > 1)$ using Monte Carlo simulation.

```
runs <- 10000; # number of runs

lambda <- 1; # The parameter lambda.

# The following two vectors will contain the results
# used for estimating E(Y) and P(Y > 1) respectively.
result_expectation <- c();
result_probability <- c();

for(run in 1:runs)
{
    # Construct a realisation of y
    x <- rexp(1,lambda); # Draw the exponential random variable X.
    y <- runif(1,0,x); # Draw the uniform(0,x) random variable Y.

    # Add y to result array.
    result_expectation <- c(result_expectation, y);
    result_probability <- c(result_probability, y > 1);
}

# Calculate the mean of the results.
mean(result_expectation);
mean(result_probability);
```

	True value	**Result 1**	**Result 2**	**Result 3**	**Result 4**	**Result 5**
$\mathbb{E}[Y]$	0.5	0.4977	0.5060	0.4982	0.4944	0.5011
$\mathbb{P}(Y > 1)$	≈ 0.1484	0.1458	0.1516	0.1488	0.1493	0.1501

Table 1.1: True values and simulation results for $\mathbb{E}[Y]$ and $\mathbb{P}(Y > 1)$, based on 10,000 independent runs. We see that the true values are closely approximated.

Solution:

The birthday of the i-th person is a discrete uniform random variable on $1, 2, 3, \ldots, 365$. In each run of the simulation, n such random variables have to be drawn. Let Z be a random variable that takes on the value 1 if at least two of the birthdays are the same and 0 otherwise. Then the probability of at least two people sharing their birthday is $\mathbb{E}[Z]$, and we find that Z is our estimator.

Sample source code for this exercise can be found in Listing 1.5. From Figure 1.3, it can be seen that the probability that in a group of 23 people, at least two of them share the same birthday, is about 50%. The function `drawunif(n,x)` draws `n` independent realizations of a uniform random variable that takes values in the array `x`.

Example 1.0.2. Two players play the following game. A coin is tossed $N = 10$ times. If a head occurs, player A receives one point, and otherwise player B receives one point. What is the probability that player A is leading at least 60% of the time?

The main ingredient of our simulation will be the simulation of a coin toss. Recall that tossing

Listing 1.5: Sample code for the birthday problem simulation.

```
# Estimate the probability that in a group of n people
# at least 2 people share the same birthday.
birthdaysim <- function(n, runs)
{
    result <- c(); # Empty result set, will contain zeroes and ones.

    for(run in 1:runs)
    {
        # Draw n random birthdays.
        birthdays <- sample(1:365, n, replace=TRUE);

        # Check if at least two birthdays are the same.
        res <- 0;
        for(i in 1:(n-1))
        {
            for(j in (i+1):n)
            {
                if((birthdays[i] == birthdays[j]))
                {
                    res <- 1;
                    break;
                }
            }

            # If we "breaked" in the inner for loop,
            # we are done.
            if(res == 1)
            {
                break;
            }
        }

        # Add the result to the result vector.
        result <- c(result, res);
    }

    # Estimate the probability by taking the mean of
    # the zeroes and ones in the result vector.
    return(mean(result));
}

runs <- 10000; # Number of runs.
n <- 2:50; # All n for which we want to estimate the probability

# Estimate probabilities for each n
prob <- c();
for(i in n)
{
    prob <- c(prob, birthdaysim(i, runs));
}

# Plot results
plot(n, prob, pch=16);
lines(c(23, 23), c(-1, prob[22]), lty=3);
lines(c(0, 23), c(prob[22], prob[22]), lty=3);
```

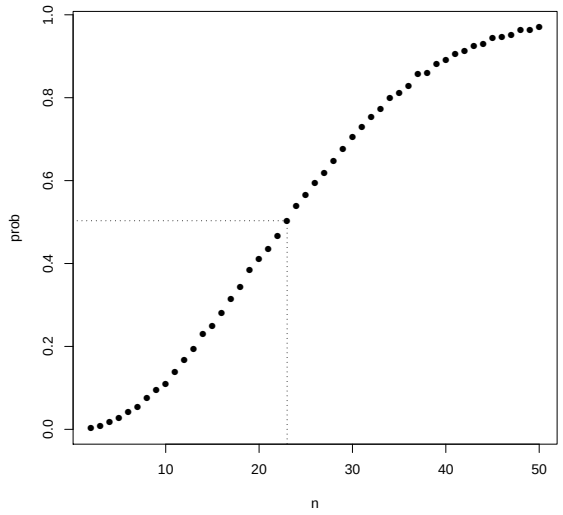


Figure 1.3: Sample output for the birthday problem. Here, the size of the group, n, varies between 2 and 50. The number of independent replications is 10,000.

a coin is essentially drawing a Bernoulli random variable with parameter $p = 0.5$.

Let A_n, resp. B_n, be the number of times player A, resp. player B, received a point before or at the n-th toss and define $D_n = A_n - B_n$. Note that $D_n > 0$ if and only if player A is leading at time n.
Now if D_n is known, and we toss the coin for the $n+1$-st time, D_n either increases by one (player A receives a point) or decreases by one (player B receives a point). This can be used to write a simulation.

As an estimator, we will use Z, where $Z = 1$ if $D_n > 0$ for at least six different n and $Z = 0$ otherwise. See Listing 1.6.

We find that the probability that A is leading at least 60% of the time is approximately 0.38.

Exercise 1.0.4 (P)**.** Recall that in the Poisson process interarrival times are independent and exponentially distributed. Generate a single trajectory of a Poisson process with rate 1. Use simulation to estimate the probability that the tenth arrival occurs before time 8.

Solution: Sample source code for generating single trajectories can be found in Listing 1.7.

For the second part: note that the arrival time of the tenth arrival is the sum of ten independent exponential random variables with rate 1, so we want to estimate

$$\mathbb{P}(X_1 + X_2 + \ldots + X_{10} < 8),$$

where $X_i \sim \text{Exp}(1)$, independent of all other X_i.

Listing 1.6: Source code for simulation of the coin tossing game in Example 1.0.2.

```
runs <- 10000; # Number of independent runs.

results <- c(); # Array of results.

for(run in 1:runs)
{
    D <- 0; # Initialise
    numAlead <- 0; # Number of times that A leads

    # simulate 10 tosses
    for(toss in 1:10)
    {
        C <- rbinom(1, 1, 0.5); # Toss a coin.
        if(C == 1)
        {
            D <- D + 1; # A wins.
        }
        else{
            D <- D - 1; # B wins.
        }

        # Check if A leads.
        if(D > 0)
        {
            numAlead <- numAlead + 1;
        }
    }

    # if A lead at least 60% of the time, add
    # a 1 to the result vector, otherwise, add a 0.
    if(numAlead >= 6)
    {
        results <- c(results, 1);
    } else
    {
        results <- c(results, 0);
    }
}

mean(results)
```

Listing 1.7: Source code used to generate a sample path from the Poisson process with rate 1. The sample path is cut off at time 10.

```
# Exercise
# Simulate a path in the Poisson process.

lambda <- 1; # Rate of the Poisson process

maxtime <- 10; # Time limit

# Draw exponential random variables, until maxtime is reached
arrivaltimes <- c(0); # Array of arrival times
cumtime <- rexp(1, lambda); # Time of next arrival
while(cumtime < maxtime)
{
    arrivaltimes <- c(arrivaltimes, cumtime);
    # Draw a new interarrival time and add it
    # to the cumulative arrival time
    cumtime <- cumtime + rexp(1, lambda);
}

# Draw a nice graph.
plot(c(), c(), xlim=c(0,maxtime), ylim=c(0,length(arrivaltimes) - 1),
    xlab="Time", ylab="Arrivals", cex.lab=1.3);
for(i in 1:(length(arrivaltimes) - 1))
{
    # Horizontal lines
    lines(c(arrivaltimes[i], arrivaltimes[i+1]), c(i-1,i-1), type="S");
    # Vertical dashed lines
    lines(c(arrivaltimes[i+1], arrivaltimes[i+1]), c(i-1,i), lty=2);
    # Add dots
    lines(arrivaltimes[i], i-1, type="p", pch=16);
    lines(arrivaltimes[i+1], i-1, type="p", pch=1);
}

# "End line"
lines(c(arrivaltimes[length(arrivaltimes)], maxtime),
    c(length(arrivaltimes) - 1, length(arrivaltimes) - 1));
lines(arrivaltimes[length(arrivaltimes)],
    length(arrivaltimes)-1, type="p", pch=16);
```

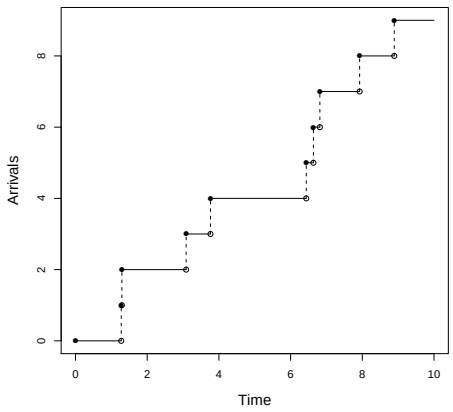


Figure 1.4: A sample path from the Poisson process with rate 1.

Before we start using simulation to estimate this probability, note that $X_1 + X_2 + \ldots + X_{10}$ follows an Erlang distribution with parameters 10 and 1, as it is the sum of ten exponentially distributed random variables. Thus, the probability can be calculated in an exact way and we find

$$\mathbb{P}(X_1 + X_2 + \ldots + X_{10} < 8) = 1 - e^{-8} \sum_{i=0}^{9} \frac{8^i}{i!} \approx 0.2834$$

In each run, we set $Z_i = 1$ if $X_1 + X_2 + \ldots X_{10} < 8$ and $Z_i = 0$ otherwise. We will repeat the same experiment a certain number n of times, and then use as an estimator $(Z_1 + Z_2 + \ldots + Z_n)/n$. For an implementation, see Listing 1.8.

Listing 1.8: Source code used for estimating the probability that the tenth arrival in a Poisson process with rate 1 takes place before time 8.

```
runs <- 10000; # Number of independent runs

results <- c(); # Will contain the results (zeroes and ones).

for(i in 1:runs)
{
    # Draw ten exponential random variables with rate 1.
    interarrivaltimes <- rexp(10, 1);
    tentharrivaltime <- sum(interarrivaltimes);

    results <- c(results, tentharrivaltime < 8);
}

# Output estimate
mean(results);
```

True value	Result 1	Result 2	Result 3	Result 4	Result 5
≈ 0.2834	0.2859	0.2927	0.2879	0.2846	0.2803

Table 1.2: True values and simulation results for $\mathbb{P}(X_1 + X_2 + \ldots + X_{10} < 8)$.

Various estimates, based on 10,000 independent runs, can be found in Table 1.2.

Monte Carlo simulation can be used for more than just the analysis of stochastic systems. One particular nice example is the estimation of π.

Example 1.0.3 (Estimation of π)**.** Consider the square with $(\pm 1, \pm 1)$ as its corner points and the inscribed circle with centre $(0,0)$ and radius 1 (see Figure 1.5). The area of the square is 4 and the area of the circle is π. If we choose a point at random in the square, it falls within the circle with probability $\pi/4$. Hence, if we let $Z_i = (X_i, Y_i)$, with $X_i, Y_i \sim \text{Uniform}(-1,1)$ independently and if we define the sequence $C_i = \mathbf{1}_{\{X_i^2 + Y_i^2 \leq 1\}}$, then the sample mean of the C_i converges (in probability) to $\pi/4$:

$$\widehat{p} := \frac{C_1 + C_2 + \ldots + C_n}{n} \xrightarrow{p} \frac{\pi}{4}. \tag{1.5}$$

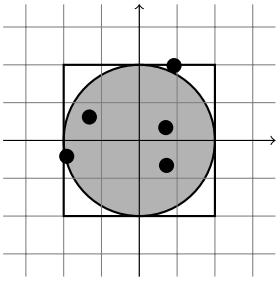


Figure 1.5: The setting used for the hit-and-miss estimator for π, with five random points.

Listing 1.9: Estimating the value of π using Monte Carlo integration.

```
runs <- 1000; # Number of independent runs.

# Generate the X and Y random variables.
x <- runif(runs,-1,1);
y <- runif(runs,-1,1);

# Generate the corresponding sequence C_i of zeros
# and ones and multiply them by 4.
results <- (x^2 + y^2 <= 1);
results <- 4*results;
mean(results);
```

We find that $\widehat{\pi} := 4\widehat{p}$ is an estimator for π.

R source code for this example can be found in Listing 1.9.

For obvious reasons the sample average of the C_i is often called the hit-or-miss estimator.

Another nice application of Monte Carlo simulation is numerical integration. Recall that integrating some function g over an interval $[a,b]$ yields a number that can be interpreted as $(b-a)$ multiplied by the "mean value" of g on $[a,b]$. This can be written in terms of the expected value operator:

$$\int_{x=a}^{b} g(x)\frac{1}{b-a}\mathrm{d}x = \mathbb{E}[g(U)], \tag{1.6}$$

where U is uniformly distributed on (a,b).

Exercise 1.0.5 (T)**.** If $U_1, U_2, \ldots, U_n$ are independent and uniformly distributed on $(0,1)$, show that

$$Z := \frac{g(U_1)+g(U_2)+\ldots+g(U_n)}{n}$$

is an unbiased estimator for

$$z := \int_{x=0}^{1} g(x)\mathrm{d}x.$$

Solution: Let $U \sim \text{Uniform}(0,1)$. The density of U is then $f(x) = 1$ for $0 < x < 1$ and $f(x) = 0$ otherwise. We find that

$$\mathbb{E}[g(U)] = \int_{x=0}^{1} g(x)f(x)\mathrm{d}x = z. \tag{1.7}$$

From this, it follows immediately that Z is an unbiased estimator for z.

Exercise 1.0.6 (P)**.** We know that

$$\int_{x=0}^{1} 4\sqrt{1-x^2}\mathrm{d}x = \pi.$$

Use numerical integration to estimate the value of π and compare the results to the results obtained via the hit-or-miss estimator.

Solution: It follows from the previous exercise that an unbiased estimator for $\int_{x=0}^{1} 4\sqrt{1-x^2}\mathrm{d}x$ is given by

$$\frac{1}{n}\sum_{i=1}^{n} 4\sqrt{1-U_i^2}, \tag{1.8}$$

with $U_1, U_2, \ldots, U_n$ independent and uniformly distributed on $(0,1)$. Sample code, based on this observation, can be found in Listing 1.10.

Listing 1.10: Numerical integration, sample source code for exercise 1.0.6.

```
1  runs <- 100000; # Number of independent runs
2
3  # Draw "runs" uniform random variables.
4  rnd <- runif(runs, 0, 1);
5
6  # Estimates:
7  results <- 4*sqrt(1-rnd*rnd)
8
9  mean(results)
```

For a comparison of the two estimators, see Figure 1.6, which depicts the results of 200 repetitions of both algorithms, each repetition using 1,000 independent realizations. From this plot, it is clear that the numerical integration algorithm performs better than the hit-or-miss estimator, since the estimates produced by numerical integration algorithm are much more concentrated around the true value π. However, this does by no means imply that numerical integration is always favorable over the hit-or-miss estimator. For example, try to integegrate the following integral numerically:

$$\int_{x=0}^{1} \frac{2}{\sqrt{1-x^2}}\mathrm{d}x = \pi. \tag{1.9}$$

The quality of the numerical estimator based on this integral is much less than the quality of the hit-or-miss estimator.

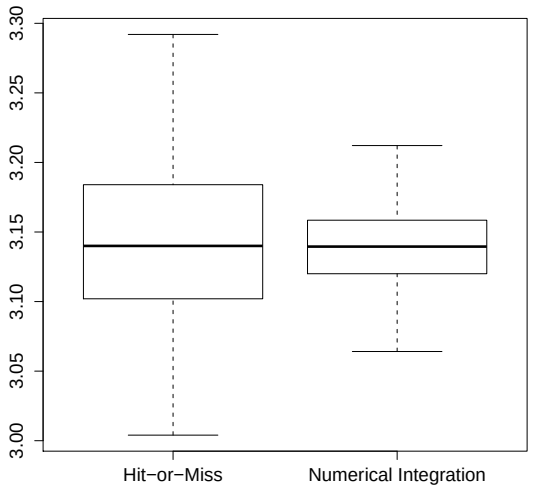


Figure 1.6: Comparison of results of two simulation algorithms, hit-or-miss vs. numerical integration.

Exercise 1.0.7 (P)**.** Use Monte Carlo integration to evaluate the integral

$$\int_{x=0}^{2} \mathrm{e}^{-x}\mathrm{d}x. \tag{1.10}$$

Solution: There are various ways in which a Monte Carlo algorithm for calculating this integral can be implemented:

(i) The most straightforward way, perhaps, is to write the integrand as $2\mathrm{e}^{-x} \times \frac{1}{2}$, and to recognise the density of the Uniform$(0,2)$-distribution. Generating Uniform$(0,2)$-random numbers $U_1, U_2, \ldots, U_n$, calculating $Z_i = 2\mathrm{e}^{-U_i}$ and then setting $(Z_1 + Z_2 + \ldots + Z_n)/n$ as an estimator for the integral.

(ii) You could also indentify the integral as the probability that $X \sim \mathrm{Exp}(1)$ it at most 1. In fact,

$$\int_{x=0}^{1} \mathrm{e}^{-x}\mathrm{d}x = \int_{x=0}^{\infty} \mathbf{1}_{\{x\leq 1\}}\mathrm{e}^{-x}\mathrm{d}x = \mathbb{P}(X \leq 1). \tag{1.11}$$

Hence, we could draw Exp(1)-random numbers $X_1, X_2, \ldots, X_n$, put $Z_i = \mathbf{1}_{\{X_i \leq 1\}}$ and use $(Z_1 + Z_2 + \ldots + Z_n)/n$ as an estimator for the integral.

Both methods will produce an estimate that lies around 0.865. Using the methods that we will discuss in Chapter 2, you will be able to show that the second method produces an estimator of higher quality.

The following exercise shows that Monte Carlo integration also works for higher dimensional integrals.

Exercise 1.0.8 (P)**.** Use Monte Carlo integration to calculate the double integral

$$\iint_A x^2 y \mathrm{d}y\mathrm{d}x, \tag{1.12}$$

where A is the triangle with corner points $(0,0)$, $(1,0)$ and $(1,2)$.

Solution: In order to integrate over the triangular area, we need to generate random points in this area. We will describe a nice method to do this. Note first that (x,y) is a point in A if and only if $0 \leq x \leq 1$ and $0 \leq y \leq 3x$.

(i) Generate a point (X,Y) in the rectangle R, which has corner points $(0,0)$, $(1,0)$, $(1,3)$, and $(0,3)$. This can be done by choosing X uniformly between 0 and 1 and choosing Y uniformly between 0 and 2.

(ii) If the point (X,Y) lies in A we are done, else, go to step (i).

An implementation of this algorithm can be found in Listing 1.11. If we use this method to calculate the integral, we must first note that the density of the uniform distribution over A is indentically $\frac{2}{3}$. Our estimator therefore will be $\frac{3}{2}X^2Y$, with (X,Y) distributed as above. A correct implementation will yield an estimate close to the true value of 0.9.

Listing 1.11: A function for generating points in the triangle with corner points $(0,0)$, $(1,0)$, and $(1,3)$.

```
sample_triangle <- function()
{
    repeat
    {
        # Draw point in rectangle.
        x <- runif(1, 0, 1);
        y <- runif(1, 0, 2);

        if(y <= 2*x) break;
    }

    return(c(x, y));
}
```

2 Output analysis

2.1 Confidence intervals

From now on, the setting will be as follows: we are interested in estimating a certain quantity z, and we have a random variable Z, such that $z = \mathbb{E}[Z]$.

Given independent realisations $Z_1, Z_2, \ldots, Z_n$ of the random variable Z, we already know that the sample mean $\overline{Z} := (Z_1 + Z_2 + \ldots + Z_n)/n$ is an unbiased estimator of z. Although this is interesting in itself, we are often also interested in the quality of the estimator: if there is a large spread in the outcomes Z_i, it is quite possible that the estimator $\overline{Z}$ is far off from the true value z.

We will often use *confidence intervals* centered around the estimator $\overline{Z}$ to give an estimate that expresses in some sense the spread of the estimate. A confidence interval is an interval in which we can assert that z lies with a certain degree of confidence.

Mathematically speaking, a $100(1-2\alpha)\%$ confidence interval is a random interval (usually depending on the outcomes Z_i), such that the probability that the interval covers the true value of z equals $1-2\alpha$. In these lecture notes, we will focus on approximate confidence intervals, inspired by the central limit theorem.[1]

2.2 The Central Limit Theorem

Consider a sequence of i.i.d. random variables $Z_1, Z_2, \ldots, Z_n$, with common mean z and variance σ^2. We have already introduced the sample mean $\overline{Z}$. We will now define the sample variance S^2:

[1] Better confidence intervals can be constructed using Student's t-distribution, see e.g. Wikipedia [WikiA].

Definition 2.2.1 (Sample variance)**.** The quantity S^2, defined by

$$S^2 := \frac{1}{n-1}\sum_{i=1}^{n}(Z_i - \overline{Z})^2, \tag{2.1}$$

is called the *sample variance* of the random sample $Z_1, Z_2, \ldots, Z_n$.

The following exercise justifies the name of the sample variance:

Exercise 2.2.1 (T)**.** Show that the sample variance is an unbiased estimator of σ^2. That is, show that

$$\mathbb{E}\left[S^2\right] = \sigma^2. \tag{2.2}$$

Solution: Note that

$$\sum_{i=1}^{n}(Z_i - \overline{Z})^2 = \sum_{i=1}^{n}(Z_i^2 - 2Z_i\overline{Z} + \overline{Z}^2) = \sum_{i=1}^{n} Z_i^2 - 2n\overline{Z}^2 + n\overline{Z}^2 = \sum_{i=1}^{n} Z_i^2 - n\overline{Z}^2. \tag{2.3}$$

From this, it follows that

$$\begin{aligned}
\mathbb{E}\left[S^2\right] &= \frac{1}{n-1}\left(\sum_{i=1}^{n}\mathbb{E}\left[Z_i^2\right] - n\mathbb{E}\left[\overline{Z}^2\right]\right) = \\
&= \frac{n}{n-1}\left(\mathbb{E}\left[Z_1^2\right] - \mathbb{E}\left[\overline{Z}^2\right]\right) \\
&= \frac{n}{n-1}\left(\mathrm{Var}[Z_1] + \mathbb{E}[Z_1]^2 - \mathrm{Var}\left[\overline{Z}\right] - \mathbb{E}\left[\overline{Z}\right]^2\right) \\
&= \frac{n}{n-1}\left(\sigma^2 + z^2 - \frac{\sigma^2}{n} - z^2\right) \\
&= \sigma^2,
\end{aligned} \tag{2.4}$$

which is what had to be shown.

The *Central Limit Theorem*, often abbreviated as CLT, relates the sample mean of a sufficiently large number of random variables to the normal distribution.

Theorem 2.2.1 (Central Limit Theorem)**.** *Let the sequence of random variables Y_n be defined by*

$$Y_n := \sqrt{n}\frac{\overline{Z} - z}{\sqrt{\sigma^2}}. \tag{2.5}$$

Then Y_n converges in distribution to a standard normal random variable, or

$$\lim_{n\to\infty}\mathbb{P}(Y_n \le y) = \Phi(y), \tag{2.6}$$

where $\Phi(\cdot)$ is the standard normal cdf, given by

$$\Phi(y) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{y} e^{-\frac{1}{2}t^2}dt. \tag{2.7}$$

This theorem is often interpreted as

$$\mathbb{P}(Y_n \leq y) \approx \Phi(y) \tag{2.8}$$

and we will use it to construct approximate confidence intervals.

Note that Y_n depends on an unknown parameter, σ^2. Fortunately, by a theorem known as Slutsky's theorem, the central limit theorem still holds if we replace σ^2 by the sample variance, S^2, so that the quantity

$$Y_n := \sqrt{n}\frac{\overline{Z} - z}{\sqrt{S^2}}, \tag{2.9}$$

still converges in distribution to a standard normal random variable.

If we let z_α be the α quantile of the standard normal distribution, that is, $\Phi(z_\alpha) = \alpha$, it follows that

$$\mathbb{P}(z_\alpha < Y_n < z_{1-\alpha}) \approx 1 - 2\alpha. \tag{2.10}$$

By rewriting the event $\{z_\alpha < Y_n < z_{1-\alpha}\}$, we find that

$$z_\alpha < \sqrt{n}\frac{\overline{Z} - z}{\sqrt{S^2}} < z_{1-\alpha} \tag{2.11}$$

is equivalent to

$$\overline{Z} - z_{1-\alpha}\sqrt{\frac{S^2}{n}} < z < \overline{Z} - z_\alpha\sqrt{\frac{S^2}{n}}, \tag{2.12}$$

so that

$$\mathbb{P}\left(\overline{Z} - z_{1-\alpha}\sqrt{\frac{S^2}{n}} < z < \overline{Z} - z_\alpha\sqrt{\frac{S^2}{n}}\right) \approx 1 - 2\alpha. \tag{2.13}$$

From this, it follows that

$$\left(\overline{Z} - z_{1-\alpha}\sqrt{\frac{S^2}{n}}, \overline{Z} - z_\alpha\sqrt{\frac{S^2}{n}}\right) \tag{2.14}$$

is an approximate $100(1-2\alpha)\%$ confidence interval for z.

We will often use $\alpha = 0.025$. In this case, $z_{1-\alpha} = -z_\alpha \approx 1.96$ and we thus use the 95% confidence interval

$$\left(\overline{Z} - 1.96\sqrt{\frac{S^2}{n}}, \overline{Z} + 1.96\sqrt{\frac{S^2}{n}}\right). \tag{2.15}$$

The quantity $z_{1-\alpha}\sqrt{S^2/n}$ is called the *half-width* of the confidence interval. From the factor $\sqrt{n}$ in the denominator, it can be seen that in order to obtain one extra digit of the value z, the number of runs must be increased by a factor 100.

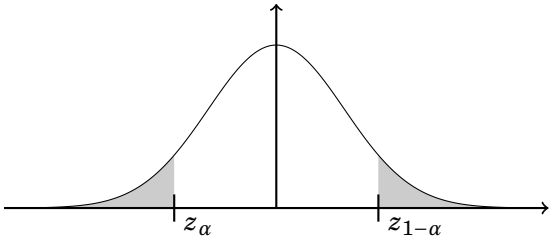


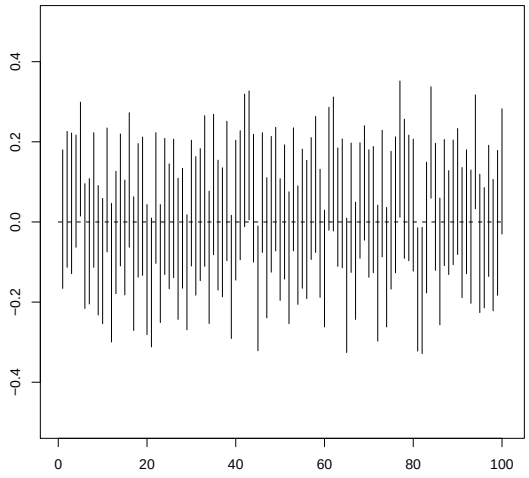
Figure 2.1: α and $1-\alpha$ quantiles of the standard normal distribution. The shaded areas both have area α.

Exercise 2.2.2 (T)**.** Show that $z_{1-\alpha} = -z_\alpha$.

Solution:

We know that the standard normal distribution is symmetric around zero. It is now clear from Figure 2.1 that $z_{1-\alpha} = -z_\alpha$.

Example 2.2.1. In order to get some feeling for confidence intervals, consider the simple example where we want to estimate the mean of a uniform random variable on $(-1,1)$. We will construct 100 approximate 95% confidence intervals, each of which is based on 50 observations. The result can be found in Figure 2.2 and the code used to generate this plot can be found in Listing 2.1. From the plot, it can be seen that most of the constructed interval contain the true expected value (0), while some of the intervals (approximately 5%) do not contain the true value.



.

Figure 2.2: 100 confidence intervals for the estimation of the mean of a uniform random variable on $(-1,1)$. Each interval is based on 50 observations. The dashed line indicates the theoretical mean.

Listing 2.1: The source code used to construct Figure 2.2.

```
repetitions <- 100; # The number of confidence intervals.
observations <- 50; # The number of observations to use for each
                    # interval.

# upper and lower limits of confidence intervals.
upper <- c();
lower <- c();

# Normal quantile.
z <- 1.96; # For 95% confidence interval.

for(run in 1:repetitions)
{
    # Draw a number of uniform random variables.
    r <- runif(observations, min=-1, max=1)

    # Calculate sample mean and sample variance.
    v <- var(r);
    m <- mean(r);

    # Calculate upper and lower limit of
    # the confidence intervals.
    halfwidth <- z * sqrt(v/observations);
    lower <- c(lower, m - halfwidth);
    upper <- c(upper, m + halfwidth);
}

# Make a nice plot.
plot(c(), c(), xlim=c(0,101), ylim=c(-0.5,0.5), type="n", xlab="", ylab="");
lines(x=c(0,101), y=c(0,0), lty=2);
for(run in 1:repetitions)
{
    lines(x=c(run, run), y=c(lower[run],upper[run]));
}
```

3 Generating random variables

R has many built-in functions to generate random variables according to a certain distribution. A few examples are listed in Table 3.1. However extensive this list may be, it does only cover the distributions that are most common. In general, you may want to draw random variables from another distribution. It may also be the case that you have to implement simulations in a different programming language, for which such a large library of functions does not exist.

There are many algorithms to draw random variables from different distributions. The aim of this chapter is to provide a few general methods, that work in many cases. It should be noted, however, that these methods are not always efficient.

We will assume that we have a method to draw Uniform$(0,1)$ random variables available. Most programming languages provide a function to generate such random variables. Given a method to draw Uniform$(0,1)$ random variables, we can easily draw from a Uniform(a,b) distribution: if U follows a uniform distribution on $(0,1)$, it follows that $a+(b-a)U$ follows a uniform distribution on (a,b).

3.1 Discrete random variables

Suppose that U is uniform on $(0,1)$ and that $p \in (0,1)$. The probability that $U \leq p$ is then just p. So if we want to draw a Bernoulli random variable, X say, that attains the value 1 with probability p, we can just take U and set $X = 1$ if $U \leq p$ and set $X = 0$ otherwise.
This observation can be extended to more general discrete random variables. Assume that we want to create a random variable that takes on values $x_1, x_2, \ldots x_n$ with probability $p_1, p_2, \ldots, p_n$, respectively. This can be done as follows: take a uniform random variable on $(0,1)$ and set $X = x_1$ if $U \leq p_1$, $X = x_2$ if $p_1 < U \leq p_1 + p_2$ and so on. It follows easily that $\mathbb{P}(X = x_i) = p_i$, for $i = 1, 2, \ldots, n$.

Exercise 3.1.1 (P)**.** Write a function that generates a realisation of a Bin(n,p) random variable (without using built-in function to perform this action, such as `rbinom` and `sample`).

Distribution	Function
Binomial	`rbinom(runs, size, prob)`
Poisson	`rpois(runs, lambda)`
Geometric (on $0,1,\ldots$)	`rgeom(runs, prob)`
Hypergeometric	`rhyper(N, m, n, k)`
Negative binomial	`rnbinom(runs, size, prob)`
Exponential	`rexp(runs, rate)`
Normal	`rnorm(runs, mean, stddev)`
Gamma	`rgamma(runs, shape, rate=...)`
Uniform	`runif(runs, min, max)`

Table 3.1: List of random number generators in R. The first parameter, `runs`, is the number of repetitions. For example, `rexp(10, 2)` returns a vector of ten exponential random numbers with mean 1/2.

Solution: Sample code can be found in Listing 3.1.

Listing 3.1: A function for drawing binomial random variables.

```
1  # This function generates a single binomial random variable
2  # with parameters n and p.
3  randombinomial <- function(n, p)
4  {
5      values <- 0:(n-1);
6      probabilities <- choose(n, values) * p^values * (1-p)^(n-values);
7
8      # Draw uniform random variable
9      u <- runif(1, 0, 1);
10     for(k in values)
11     {
12         if(sum(probabilities[1:(k+1)]) >= u)
13         {
14             return(k);
15         }
16     }
17
18     return(n);
19 }
```

3.2 Inverse transformation method

The *inverse transformation method* is widely applied for simulating continuous random variables that is often used. Its name stems from the fact that the inverse of the cdf is used in the construction of a random variable with the desired distribution. The method can be explained best by an example.

Exercise 3.2.1 (T)**.** If U follows a uniform distribution on $(0,1)$, show that

$$X := -1/\lambda \log(1-U) \tag{3.1}$$

follows an exponential distribution with parameter λ. Also show that $Y := -1/\lambda \log(U)$ follows an exponential distribution with parameter λ.

Solution: We need to show that the cdf of X has the form $1-\exp(-\lambda x)$. This can be shown as follows:

$$\begin{aligned}\mathbb{P}(X \le x) &= \mathbb{P}(-1/\lambda \log(1-U) \le x) = \mathbb{P}(\log(1-U) \ge -\lambda x) \\ &= \mathbb{P}(1-U \ge \exp(-\lambda x)) = \mathbb{P}(U \le 1-\exp(-\lambda x)) = 1-\exp(-\lambda x). \quad (3.2)\end{aligned}$$

The second assertion follows immediately from the facts that U and $1-U$ are both Uniform$(0,1)$ distributed random variables.

The approach used in the above exercise is applicable more generally. First, we define the quantile function F^{-1} of a distribution function F. If $u \in [0,1]$, we set

$$F^{-1}(u) = \inf\{x \in \mathbb{R} : F(x) \ge u\}. \tag{3.3}$$

We are now able to show that

Theorem 3.2.1. *If F is a cdf and U is a uniform random variable on $(0,1)$, then $F^{-1}(U)$ is distributed according to F.*

Proof. Let $U \sim$ Uniform$(0,1)$ and let F^{-1} be the inverse of a cdf, as defined in Equation (3.3). Note that F is non-decreasing, from which it follows that $F^{-1}(u) \le x$ if and only if $\inf\{y : F(y) \ge u\} \le x$ (by definition of F^{-1}) if and only if $u \le F(x)$. From this, it follows that

$$\mathbb{P}\left(F^{-1}(U) \le x\right) = \mathbb{P}(U \le F(x)) = F(x), \tag{3.4}$$

which shows that F is the cdf of the random variable $F^{-1}(U)$. □

Using this theorem, we are now able to generate random variables with cdf F, *provided that we know the quantile function*.

Exercise 3.2.2 (T)**.** The Pareto distribution is often used for extreme claim sizes. Its CDF is given by

$$F(x) = \begin{cases} 0 & \text{if } x < x_m \\ 1-\left(\frac{x_m}{x}\right)^{\alpha} & \text{if } x \ge x_m \end{cases} \tag{3.5}$$

where x_m is called the location parameter and $\alpha > 0$ is called the shape parameter. Given a random variable U, that is uniformly distributed on $(0,1)$, construct a random variable that is distributed according to a Pareto distribution with parameters x_m and α.

Solution: Let $u > 0$. Note that $F(x) = u$ if and only if $1-(x_m/x)^{\alpha} = u$, from which it follows that $x_m/x = (1-u)^{1/\alpha}$, and hence $x = x_m(1-u)^{-1/\alpha}$. From this, it follows that the random variable X, defined by

$$X = \frac{x_m}{(1-U)^{\frac{1}{\alpha}}}, \tag{3.6}$$

follows a Pareto distribution with parameters x_m and α. The same holds for the random variable $Y = x_m(U)^{-1/\alpha}$, since U and $1-U$ follow the same distribution.

Exercise 3.2.3 (P)**.** Often, the inverse F^{-1} cannot be written down explicitly. In this case, numerical inversion can be used. Write an R function that is able to compute the numerical inverse of a cdf F. Hint: take a look at the function `uniroot`, or write a numerical routine yourself.

Solution: For simplicity, we will assume that $F : [0,\infty) \to [0,1]$ is an invertible function. Suppose that we want to calculate the value $x = F^{-1}(y)$ for some $y \in [0,1]$. This is the same as requiring that $F(x) = y$, or equivalently, $F(x) - y = 0$. So if y is some fixed value, we can find x by solving $F(x) - y = 0$ for x. This is exactly what the function `uniroot` does.

The function `uniroot`, in its basic form, takes two parameters: a function and an interval in which it will look for a root of the specified function. Suppose that we have a function called `f`, and we want to solve $f(x) = 0$, and we know that this x must lie somewhere between 2 and 3. We then use `uniroot(f, c(2,3))` to find the value of x.

In our case, the function f has another parameter, namely the value of y. Suppose again that we have a function called `f`, which is now a function of two variables, `x` and `y`, say. If we want to find the value of x such that $f(x,12) = 0$, we can use `uniroot(f, c(2,3), y=12)`. The value of `y`, is now substituted in the function `f`. For an example, see Listing 3.2. Note that `uniroot` returns a lot of information, not only the value of x. To retrieve just the value of x, you can use `$root`.

Listing 3.2: Sample source code for using the function `uniroot`. We use this function to calculate the square root of 2.

```
# Some function of two variables.
f <- function(x, y)
{
    return(x^2 - y);
}

# Find the square root of 2. We know that it
# lies somewhere between 0 and 2.
# The function uniroot find the value of x,
# between 0 and 2, such that f(x, 2) = 0.
# In this case, this means that x^2 = 2.
info <- uniroot(f, c(0,2), y=2);

# Take only value from info.
info$root
```

We are now ready to use the function `uniroot` for the numerical inversion of the function F. For a sample implementation, see the source code in Listing 3.3.

Listing 3.3: Source code for computing the numerical inverse of a cdf F.

```
# The function that we would like to invert.
F <- function(x, y)
{
    return(1 - (1+2*x+(3/2)*x^2)*exp(-3*x) - y);
}

# Returns x (between 0 and 1) such that f(x) = y
inverse <- function(f, y)
{
    root <- uniroot(f, c(0, 1), y=y);
    return(root$root);
}

F(0.7, 0)
inverse(F, 0.6160991)
```

4
Fitting distributions

Consider for a moment the compound Poisson risk model, in which insurance claims arrive according to a Poisson process, and claim sizes have some known distribution. The time between the arrival of subsequent claims follows an exponential distribution. However, in general it may not be known what the parameter of this exponential distribution is. Also, it may be postulated (on the basis of statistical considerations, or experts' opinions) that the claim sizes follow a particular distribution, say a gamma distribution, with unknown parameters.

In order to be able to simulate such processes, we must find a way to estimate the parameters of these distributions. If we are lucky enough, we have some historical data at hand, for example a list containing the times and sizes of subsequent claims.

This is one of the basic questions in statistics: given a set of samples (data points) $X_1, X_2, \ldots, X_n$, determine the parameters of the underlying distribution.

In this chapter, we will briefly describe two methods that can be used to estimate the parameters of a given distribution. It is not meant as a complete exposition of what is called *parametric statistics*. Consult any statistical text book for many more techniques of distribution fitting.

4.1 Method of moment estimators

Suppose that we have a data set $X_1, X_2, \ldots, X_n$. Suppose that we know that the X_i are drawn from an exponential distribution with unknown parameter λ. The problem of specifying the complete distribution of the X_i is now reduced to specifying the value of λ. From the law of large numbers, we know that the sample mean,

$$\overline{X}_n = \frac{X_1 + X_2 + \ldots + X_n}{n}, \tag{4.1}$$

converges to the expected value (first moment (!)) of the underlying distribution, in this case

$1/\lambda$. It therefore seems natural to choose for the parameter λ the value such that

$$\frac{1}{\lambda} = \overline{X}_n, \tag{4.2}$$

so our estimate of λ in this case would be $\widehat{\lambda} = 1/\overline{X}_n$.

In general, the k-th sample moment of $X_1, X_2, \ldots, X_n$ is defined as

$$M_k = \frac{X_1^k + X_2^k + \ldots + X_n^k}{n}, \qquad (k \in \mathbb{N}), \tag{4.3}$$

hence, M_k is the average of the k-th powers of the sample points. Note that the first sample moment is just the sample mean.

Suppose next that our model implies that the X_i are distributed according to the gamma distribution, with unknown parameters $\alpha > 0$ and $\beta > 0$. The gamma distribution has pdf

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} \mathrm{e}^{-\beta x}, \qquad x > 0. \tag{4.4}$$

Its mean is given by α/β. Now fitting the gamma distribution to the first sample moment gives the equation

$$\frac{\alpha}{\beta} = M_1. \tag{4.5}$$

A single equation is not sufficient to solve for both α and β, so we must add another equation to the system. The second moment of the gamma distribution is $\alpha(\alpha+1)/\beta^2$. A second equation sets

$$\frac{\alpha(\alpha+1)}{\beta^2} = M_2. \tag{4.6}$$

Now the system of equations (4.5)-(4.6) has a unique solution, which can be found in the following way. From Equation (4.5), it follows that $\beta = \alpha/M_1$. Substituting this into Equation (4.6) yields

$$\frac{\alpha+1}{\alpha} = 1 + \frac{1}{\alpha} = \frac{M_2}{M_1^2}, \tag{4.7}$$

from which it follows that

$$\alpha = \frac{M_1^2}{M_2 - M_1^2}, \tag{4.8}$$

and then, using Equation (4.5), we find

$$\beta = \frac{\alpha}{M_1} = \frac{M_1}{M_2 - M_1^2}. \tag{4.9}$$

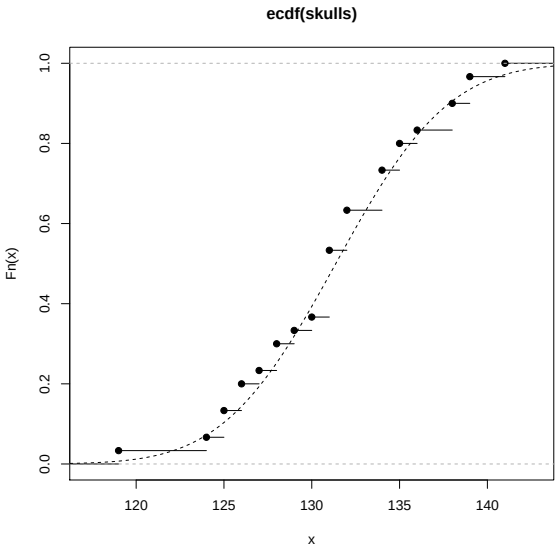
Hence, assuming that the data $X_1, X_2, \ldots, X_n$ is distributed according to the gamma distribution, the method of moment estimators results in the following parameters of the distribution:

$$\widehat{\alpha} = \frac{M_1^2}{M_2 - M_1^2}, \qquad \widehat{\beta} = \frac{M_1}{M_2 - M_1^2}. \tag{4.10}$$

Exercise 4.1.1 (P)**.** In a 1905 research[1] into the measurement of Egyptian skulls through the ages, 150 skulls were measured. Part of the data set is displayed in Figure 4.1(a). In the table the maximal breadth of thirty skulls stemming from the year 4000BC are shown. Use the method of moment estimators to fit a normal distribution to this data.

131	125	131	119
136	138	139	125
131	134	129	134
126	132	141	131
135	132	139	132
126	135	134	128
130	138	128	127
131	124		

(a) Raw data.



(b) Empirical distribution function, compared to the normal distribution plot with parameters estimated by the method of moments.

Figure 4.1: Maximal breadth of 30 Egyptian skulls.

Solution: The first and second moment of the normal distribution are μ and $\sigma^2+\mu^2$. The method of moment estimators gives rise to the following system of equations:

$$\begin{cases} \mu = M_1, \\ \sigma^2+\mu^2 = M_2 \end{cases} \tag{4.11}$$

from which it follows that $\widehat{\mu} = M_1$ and $\widehat{\sigma^2} = M_2 - M_1^2$. Substituting the data from Figure 4.1(a), we find that $\widehat{\mu} \approx 131.4$ and $\widehat{\sigma^2} \approx 25.4$. See Listing 4.1.

Listing 4.1: Fitting a normal distribution to the first two moments of the skull data in Figure 4.1(a).

```
1  # Data set
2  skulls <- c(131, 125, 131, 119, 136, 138, 139,
3       125, 131, 134, 129, 134, 126, 132, 141,
4       131, 135, 132, 139, 132, 126, 135, 134,
```

[1]Thomson A. and Randall-Maciver R., *Ancient Races of the Thebaid*, Oxford University Press (1905). Data retrieved from The Data and Story Library.

```
5       128, 130, 138, 128, 127, 131, 124);
6
7   # First and second moment
8   M1 = mean(skulls);
9   M2 = mean(skulls^2);
10
11  # Estimates for mu and sigma^2
12  mu <- M1
13  sigma2 <- M2 - M1*M1
14
15  # Plot of the empirical distribution function and
16  # the theoretical distribution function.
17  plot(ecdf(skulls));
18  x <- seq(mu-3*sqrt(sigma2), mu+3*sqrt(sigma2), length=100);
19  lines(x, pnorm(x, mu, sqrt(sigma2)), lty="dashed");
```

Exercise 4.1.2 (P)**.** Using the command `rgamma(1000, shape=1, rate=2)`, generate a list of 1000 gamma-distributed random variables with parameters 1 and 2. Use the method of moment estimators to estimate the parameters of the gamma distribution, based on the random sample.

Solution: See the source code in Listing 4.2.

Listing 4.2: Implementation of the method of moment estimators for the gamma distribution.

```
1   data <- rgamma(1000, shape=1, rate=2);
2
3   # Calculate first and second moment
4   M1 <- mean(data);
5   M2 <- mean(data^2);
6
7   alpha <- M1^2 / (M2 - M1^2);
8   beta <- M1 / (M2 - M1^2);
```

4.2 Maximum likelihood estimation

Another method of parameter estimation is known as maximum likelihood estimation (MLE). Let us assume again that the sample $X_1, X_2, \ldots, X_n$ is drawn from the exponential distribution with unknown parameter λ. Their joint distribution is

$$f(X_1, X_2, \ldots, X_n|\lambda) = \prod_{i=1}^{n} \left(\lambda \mathrm{e}^{-\lambda X_i}\right) = \lambda^n \mathrm{e}^{-\lambda \sum_{i=1}^n X_i}. \tag{4.12}$$

The function $f(X_1, X_2, \ldots, X_n|\lambda)$ is also called the likelihood of the data $X_1, X_2, \ldots, X_n$ under the parameter λ. The maximum likelihood estimator for λ is the estimator $\widehat{\lambda}$ that maximises the likelihood of the given data set, hence, we are looking for the value $\lambda > 0$ for which

$$\lambda^n \mathrm{e}^{-\lambda \sum_{i=1}^n X_i} \tag{4.13}$$

is maximal. Hence, we have to take the derivative (with respect to λ):

$$\frac{\mathrm{d}}{\mathrm{d}\lambda} \lambda^n \mathrm{e}^{-\lambda \sum_{i=1}^n X_i} = \left(n - \lambda \sum_{i=1}^{n} X_i\right) \lambda^{n-1} \mathrm{e}^{-\lambda \sum_{i=1}^n X_i}. \tag{4.14}$$

From setting the right hand side equal to 0 we find that

$$n - \lambda \sum_{i=1}^{n} X_i = 0, \tag{4.15}$$

(since λ^{n-1} and $\exp\left(-\lambda \sum_{i=1}^{n} X_i\right)$ are always positive) which yields the estimator $\widehat{\lambda} = n/\sum_{i=1}^{n} X_i = 1/\overline{X}_n$.

If a distribution depends on more than one parameter, we have set the partial derivatives with respect to all of these parameters equal to 0. This yields a system of equations, to be solved simultaneously. For example, suppose that we want to fit a normal distribution to the set of data $X_1, X_2, \ldots, X_n$. The density of the normal distribution is

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{4.16}$$

so the likelihood of $X_1, X_2, \ldots, X_n$ under the parameters μ and σ^2 is given by

$$f(X_1, X_2, \ldots, X_n|\mu,\sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left(-\frac{\sum_{i=1}^{n}(X_i-\mu)^2}{2\sigma^2}\right). \tag{4.17}$$

We would like to take the partial derivatives of this function with respect to both μ and σ^2, but this will be rather difficult. Since the natural logarithm is an increasing function, the likelihood and the so-called *log-likelihood* (the natural logarithm of the joint density function) will attain their maximum values in the exact same point (see Figure 4.2). The log-likelihood is given by

$$\begin{aligned} \Lambda(X_1, X_2, \ldots, X_n|\mu,\sigma^2)) &= \log(f(X_1, X_2, \ldots, X_n|\mu,\sigma^2)) \\ &= \log\left(\left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left(-\frac{\sum_{i=1}^{n}(X_i-\mu)^2}{2\sigma^2}\right)\right) \\ &= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{\sum_{i=1}^{n}(X_i-\mu)^2}{2\sigma^2}. \end{aligned} \tag{4.18}$$

Taking the partial derivatives of the log-likelihood with respect to μ and σ^2 of the log-likelihood gives the following system of equations

$$\begin{cases} \frac{\partial}{\partial\mu}\Lambda(X_1, X_2, \ldots, X_n|\mu,\sigma^2) = \frac{1}{\sigma^2}\left(\sum_{i=1}^{n}(X_i-\mu)\right) = 0, \\ \frac{\partial}{\partial\sigma^2}\Lambda(X_1, X_2, \ldots, X_n|\mu,\sigma^2) = -\frac{n}{2\sigma^2} + \frac{\sum_{i=1}^{n}(X_i-\mu)^2}{2\sigma^4} = 0, \end{cases} \tag{4.19}$$

from which the maximum likelihood estimators follow:

$$\widehat{\mu} = \overline{X}_n, \qquad \widehat{\sigma^2} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X}_n)^2. \tag{4.20}$$

Note that in this particular case, the maximum likelihood estimators are exactly the same as the estimators we found in Exercise 4.1.1. In general, this is not the case.

Exercise 4.2.1 (T)**.** Verify Equation (4.20).

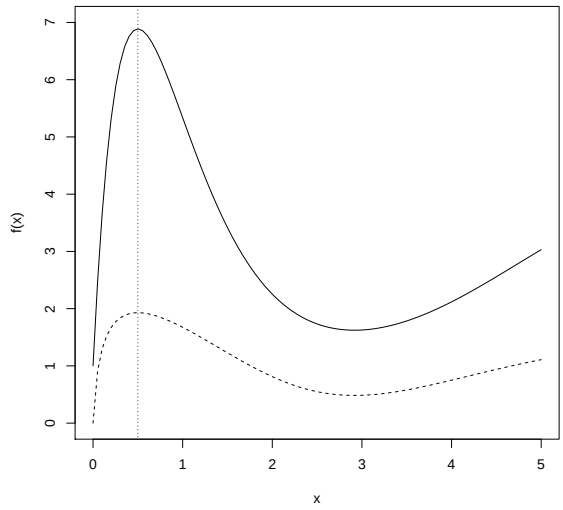


Figure 4.2: A function and its logarithm (dashed) — the dotted line indicates the maximum of the function, showing that the function and its logarithm assume their maximum on the same value.

Exercise 4.2.2 (P)**.** Consider again Exercise 4.1.1. Fit a normal distribution to this data using a maximum likelihood estimator.

Solution: We can use the formulas in Equation (4.20) to determine the estimates for μ and σ^2. In particular, we find $\widehat{\mu} \approx 131.4$ and $\widehat{\sigma^2} \approx 25.4$. Also see the source code in Listing 4.3.

Listing 4.3: Fitting a normal distribution using the maximum likelihood estimators for the mean and variance, based on the data in Figure 4.1(a).

```
# Data set
skulls <- c(131, 125, 131, 119, 136, 138, 139,
    125, 131, 134, 129, 134, 126, 132, 141,
    131, 135, 132, 139, 132, 126, 135, 134,
    128, 130, 138, 128, 127, 131, 124);

# Maximum likelihood estimates
mu <- mean(skulls);
sigma2 <- mean((skulls-mu)^2);

# Plot of the empirical distribution function and
# the theoretical distribution function.
plot(ecdf(skulls));
x <- seq(mu-3*sqrt(sigma2), mu+3*sqrt(sigma2), length=100);
lines(x, pnorm(x, mu, sqrt(sigma2)), lty="dashed");
```

4.3 Empirical distribution functions

Given independent realizations $X_1, X_2, \ldots, X_N$ of some distribution F, we can define a function $\hat{F}$ as follows

$$\hat{F}(x) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{X_i \leq x\}}. \tag{4.21}$$

For each x, the function $\hat{F}$ gives the fraction of the X_i that does not exceed x. It is therefore a natural approximation to the distribution function F. Indeed, using the strong law of large numbers, it can be shown that $\hat{F}(x) \to F(x)$ almost surely as $N \to \infty$, for all x. For obvious reasons, the function $\hat{F}$ is called the *empirical distribution function*.
R has built-in functionality for obtaining the empirical distribution function, using the command `ecdf`. An example of how to use this function can be found in Listing 4.4. The script in the listing gives as output a plot such as in the left panel in Figure 4.3, in which the empirical distribution function is compared to the actual distribution function, based on a sample of size $N = 100$ from a standard normal distribution.

Another way of assessing a distribution based on simulations, is the use of histograms or empirical densities. An example can be found in the right panel in Figure 4.4 (also see the last three lines in Listing 4.4).

Listing 4.4: Source code for comparison of the empirical distribution function to the actual distribution function of the standard normal distribution.

```
1  # Draw 100 standard normal random variables.
2  r <- rnorm(100,0,1);
3
4  # Plot ecdf and theoretical cdf
5  plot(ecdf(r), do.points=FALSE);
6
7  x <- seq(min(r), max(r), 0.1);
8  lines(x, pnorm(x, 0, 1));
9
10 # Show the histogram and the (theoretical) PDF.
11 hist(r, freq=FALSE, xlim=c(-3,3), cex.lab=1.3);
12 lines(x, dnorm(x,0,1));
```

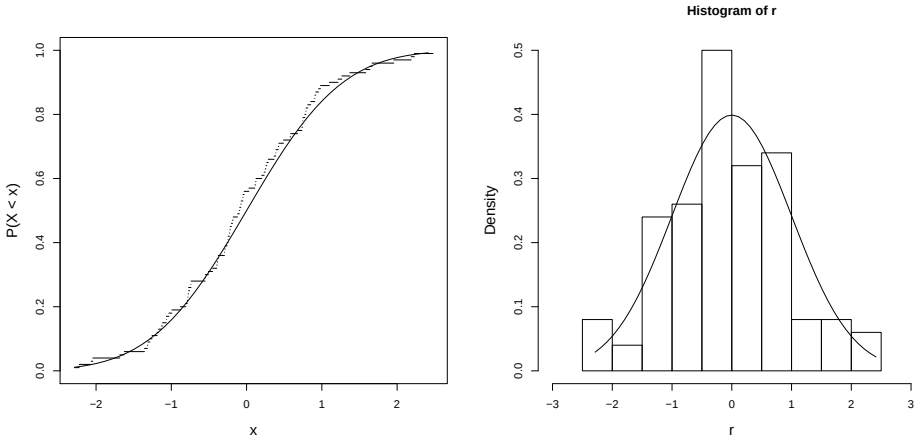


Figure 4.3: Comparison of the empirical distribution function to the CDF and of a histogram to the PDF in a standard normal setting; the results are based on a sample of length 100.

5

Ruin probabilities in the compound Poisson risk model

In this chapter, we will use simulation to analyze a model that is used extensively in the field of insurance mathematics. We will start with the simulation of the Poisson process and then extend this simulation to the compound Poisson process. The compound Poisson process is the main ingredient for the compound Poisson risk process, so once we are able to simulate the compound Poisson process, we will use this for simulating the compound Poisson risk process.

Much background information on this topic can be found in Chapter 4 of [Kaas01].

5.1 Simulating a Poisson process

Recall that a *Poisson process* is a counting process in which the interarrival times are independent and identically distributed according to an exponential distribution. This basically means that our counter is set to zero at time 0 and after an exponential amount of time the counter is increased by one; then, we wait another exponential amount of time and again increase the counter by one, and so on. A sample path of a Poisson process with rate 0.5 is visualized in Figure 5.1.

Exercise 5.1.1 (P)**.** Simulate a sample path from a Poisson process and represent it graphically.

Solution:

See the sample source code in Listing 5.1. This source code produces figures such as Figure 5.1.

A *compound Poisson process* with jump size distribution G is another stochastic process, which, as the name suggests, is related to the Poisson process. If the *underlying Poisson process* is $N(t)$,

Listing 5.1: Source code for generating sample paths from the Poisson process.

```
lambda <- 0.5; # Rate of the Poisson process

maxtime <- 10; # Time limit

# Draw exponential random variables, until maxtime is reached
arrivaltimes <- c(0); # Array of arrival times
cumtime <- rexp(1, lambda); # Time of next arrival
while(cumtime < maxtime)
{
    arrivaltimes <- c(arrivaltimes, cumtime);

    # Draw a new interarrival time and add it
    # to the cumulative arrival time
    cumtime <- cumtime + rexp(1, lambda);
}

# Draw a nice graph.
plot(c(), c(), xlim=c(0,maxtime), ylim=c(0,length(arrivaltimes) - 1),
    xlab="Time", ylab="Arrivals", cex.lab=1.3);
for(i in 1:(length(arrivaltimes) - 1))
{
    # Horizontal lines
    lines(c(arrivaltimes[i], arrivaltimes[i+1]),
        c(i-1,i-1), type="S");

    # Vertical dashed lines
    lines(c(arrivaltimes[i+1], arrivaltimes[i+1]),
        c(i-1,i), lty=2);

    # Add dots
    lines(arrivaltimes[i], i-1, type="p", pch=16);
    lines(arrivaltimes[i+1], i-1, type="p", pch=1);
}

# Last piece of line in the graph.
lines(c(arrivaltimes[length(arrivaltimes)], maxtime),
    c(length(arrivaltimes) - 1, length(arrivaltimes) - 1));
lines(arrivaltimes[length(arrivaltimes)],
    length(arrivaltimes)-1, type="p", pch=16);
```

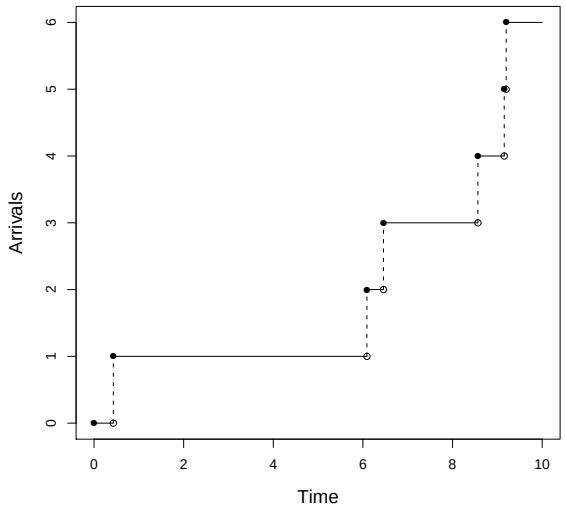


Figure 5.1: A realisation of a Poisson process (rate: 0.5, cut off at time 10).

the compound Poisson process can be written as

$$S(t) = \sum_{i=1}^{N(t)} X_i, \tag{5.1}$$

where the X_i are i.i.d. according to some distribution function G (which is called the jump size distribution) and independent of $N(t)$. The process can be explained as follows: consider the Poisson process. Whenever an arrival occurs, a stochastic amount or *jump* (the size of a claim, say) arrives and this is added to the compound Poisson process.

Exercise 5.1.2 (P)**.** Simulate a sample path from a compound Poisson process with exponential jump sizes with mean 1.

Solution:

Sample source code can be found in Listing 5.2. This code produces figures such as Figure 5.2.

Exercise 5.1.3 (P)**.** This exercise is Example 5.15 in [Ross07]. Suppose independent Exp(0.1) offers to buy a certain item arrive according to a Poisson process with intensity 1. When an offer arrives, you must either accept it or reject it and wait for the next offer to arrive. You incur costs at rate 2 per unit time. Your objective is to maximise the expected total return. One can think of various policies to choose which offer to accept.

(i) Accept the first offer that arrives;

(ii) Accept the first offer that is at least 9;

(iii) Accept the first offer that is at least 16;

Listing 5.2: Source code for generating sample paths from the compound Process process with exponentially distributed jumps.

```
lambda <- 0.5; # Rate of the Poisson process
maxtime <- 10; # Time limit

# Draw exponential random variables, until maxtime is reached
arrivaltimes <- c(0); # Array of arrival times
cumtime <- rexp(1, lambda); # Time of next arrival
level <- c(0); # Level of the compound Poisson process
while(cumtime < maxtime)
{
    arrivaltimes <- c(arrivaltimes, cumtime);

    # Draw a new jump from the exponential distribution
    # and add it to the level.
    oldLevel <- level[length(level)];
    newLevel <- oldLevel + rexp(1,0.5);
    level <- c(level, newLevel);

    # Draw a new interarrival time and add it
    # to the cumulative arrival time
    cumtime <- cumtime + rexp(1, lambda);
}

# Draw a nice graph.
plot(c(), c(), xlim=c(0, maxtime), ylim=c(min(level), max(level)),
    xlab="Time", ylab="Level");
for(i in 1:(length(arrivaltimes) - 1))
{
    lines(c(arrivaltimes[i], arrivaltimes[i+1]),
        c(level[i], level[i]), type="S");
    lines(c(arrivaltimes[i+1], arrivaltimes[i+1]),
        c(level[i], level[i+1]), lty=2);
    # Add dots
    lines(arrivaltimes[i], level[i], type="p", pch=16);
    lines(arrivaltimes[i+1], level[i], type="p", pch=1);
}

# Last piece of line in the graph.
lines(c(arrivaltimes[length(arrivaltimes)], maxtime),
    c(level[length(level)], level[length(level)]));
lines(arrivaltimes[length(arrivaltimes)],
    level[length(level)], type="p", pch=16);
```

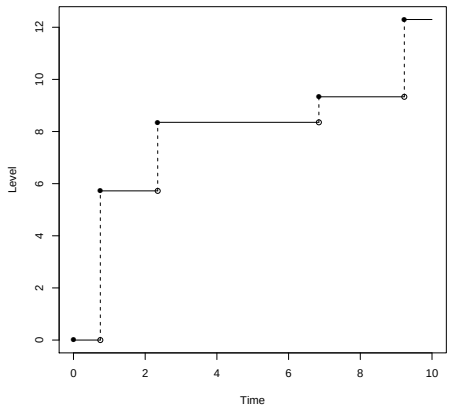


Figure 5.2: Sample path of the compound Poisson process with exponentially distributed jumps.

(iv) Accept the first offer that is at least 25.

Use simulation to decide which policy is optimal.

Solution: The first policy is somewhat easier to simulate than the other three policies, since it merely involves drawing the arrival time and the size of the first offer. The other policies involve drawing a sequence of interarrival times and offers, until the first offer larger than the threshold arrives. All policies are implemented in the source code in Listing 5.3. The results, based on 10,000 independent simulation runs for each policy, can be found in Table 5.1.

Policy 1	**Policy 2**	**Policy 3**	**Policy 4**
7.94 ± 0.2	14.1 ± 0.2	16.3 ± 0.3	10.66 ± 0.5

Table 5.1: Approximate 95% confidence intervals based on 10,000 independent simulation runs for the three different policies.

The results show that the third policy is optimal, which is in accordance with Example 5.15 in [Ross07]. The example shows that the optimal policy is to accept the first offer that is larger than $10 \log 5 \approx 16.1$.

5.2 The compound Poisson risk model

One of the main performance characteristics in insurance mathematics is the probability of ruin. In order to study this probability from a mathematical point of view, various models have been developed, of which the most well-known is probably the *compound Poisson risk model*. In this model, i.i.d. claims arrive according to a Poisson process, while if no claim arrives the capital of the insurance company grows at a constant rate: the premium per time unit.

Listing 5.3: Implementation of all three policies in the search for the policy that maximises the expected total return.

```
# Determine the expected return value when the first offer is chosen
choosefirst <- function(runs, arrivalrate, offerrate, costrate)
{
    # Arrival and offer rate
    arrival <- rexp(runs, arrivalrate);
    offer <- rexp(runs, offerrate);
    # Result vector
    result <- offer - costrate*arrival;
    return(result);
}

# Determine the expected return value when the first offer larger
# than a certain threshold is chosen.
choosethreshold <- function(runs, arrivalrate,
    offerrate, costrate, threshold)
{
    # Result vector
    result <- c();

    for(i in 1:runs)
    {
        arrival <- rexp(1, arrivalrate);
        offer <- rexp(1, offerrate);
        # Choose the first offer that is larger than
        # the threshold
        while(offer < threshold)
        {
            arrival <- arrival + rexp(1, arrivalrate);
            offer <- rexp(1, offerrate);
        }

        result <- c(result, offer - costrate*arrival);
    }

    return(result);
}

runs <- 10000; # Number of independent runs
arrivalrate <- 1;
offerrate <- 0.1;
costrate <- 2;

data1 <- choosefirst(runs, arrivalrate, offerrate, costrate);
data2 <- choosethreshold(runs, arrivalrate, offerrate, costrate, 9);
data3 <- choosethreshold(runs, arrivalrate, offerrate, costrate, 16);
data4 <- choosethreshold(runs, arrivalrate, offerrate, costrate, 25);
data <- c(data1, data2, data3, data4);

# Display the data
data <- matrix(data, byrow=FALSE, nrow=runs);
boxplot(data, range=0);
```

The model can be described as

$$U(t) = u + ct - \sum_{i=1}^{N(t)} X_i. \tag{5.2}$$

Here, $U(t)$ is the capital of the insurance company at time t, c is the premium income rate, $N(t)$ is a Poisson process with rate λ. The X_i are i.i.d. claims, with common distribution function F and mean $\mu_1 := \mathbb{E}[X_1]$. Equation (5.2) can be written as

$$U(t) = u - S(t), \tag{5.3}$$

where

$$S(t) = \sum_{i=1}^{N(t)} X_i - ct. \tag{5.4}$$

The premium income rate c is often chosen such that

$$c = (1+\theta)\lambda\mu_1, \tag{5.5}$$

where θ is called the *safety loading*, with $\theta = \frac{c}{\lambda\mu_1} - 1$. This name is justified by the fact that

$$\mathbb{E}\left[\sum_{i=1}^{N(t)} X_i\right] = \lambda\mu_1 t, \tag{5.6}$$

so that $\lambda\mu_1$ is the basis for the premium income rate, covering merely the expected claim per unit time. To this an extra amount $\theta\lambda\mu_1$ is added, to provide the insurance company with some extra financial room, in case a large claim arrives.

We say that the insurance company goes bankrupt (ruins) at the moment its capital $U(t)$ drops below zero. The time of ruin, denoted by $T(u)$, is then defined as

$$T(u) := \inf\{t \geq 0 : U(t) < 0\} = \inf\{t \geq 0 : S(t) > u\}. \tag{5.7}$$

If we let $\inf\emptyset = \infty$, then $T(u) = \infty$ if ruin does not occur.

Often, the parameter u is omitted and we write $T = T(u)$.

5.2.1 Quantities of interest

The probabilities of ruin in finite or infinite time, defined as, respectively,

$$\begin{aligned} \psi(u,\tau) &:= \mathbb{P}(T \leq \tau), \\ \psi(u) &:= \mathbb{P}(T < \infty), \end{aligned}$$

are probably the most important performance characteristics in the compound Poisson risk model. Many other quantities may be of interest as well, for example the distribution of the deficit at ruin, which is the level of the process at the moment ruin occurs, given that ruin occurs, i.e.

$$\mathbb{P}(|U(T)| \leq s \mid T < \infty).$$

In order to study the probability that a company recovers from ruin, it is useful to consider quantities such as the length of ruin and the maximal severity of ruin. The first quantity is the time between the moment of ruin (T, the first time the capital of the insurance company drops below 0) and the first time after T that the capital is larger than 0 again. The second quantity denotes the minimum of the capital $U(t)$ in this time interval.

In some cases it is possible to calculate some of these quantities, while in other cases it is very hard, if possible at all. For example, when the claims follow an exponential distribution, it is well possible to derive expressions for the probability of ruin $\psi(u)$ and the distribution of the deficit at ruin. See e.g. [Kaas01].

The following result provides an upper bound for the probability of ruin. It is known as *Lundberg's upper bound*.

Proposition 5.2.1. *Let $M_X(t)$ be the moment generating function of the claim size distribution and let R be the positive solution of the Lundberg equation,*

$$1+(1+\theta)\mu_1 R = M_X(R). \tag{5.8}$$

Then, for all initial capitals u, we have

$$\psi(u) \leq e^{-Ru}. \tag{5.9}$$

The R environment can be used to solve equations such as Equation (5.8), using the function `uniroot`. This function finds a root of a given function, in a given interval. For example, suppose that we want to find the positive value of x such that $x^2 = 3$, this corresponds to the positive root of $x^2 - 3$. We know that this root lies in the interval $(1,2)$. The source code in Listing 5.4 shows how this can be done using R.

Listing 5.4: An example of the use of the function `uniroot`.

```
1  f <- function(x){ return(x^2-3); }
2
3  # Find the root, and a lot more information.
4  rootinfo <- uniroot(f, c(1,2));
5
6  # The root itself.
7  rootinfo$root
```

Some risk measures admit nice formulas in the case where claim sizes follow an exponential distribution. The following result provides some of these formulas. These will later be used to compare to simulation results.

Proposition 5.2.2. *If the claim sizes follow an exponential distribution with mean μ_1, the following formulas hold:*

$$\psi(u) = \frac{1}{1+\theta}\exp\left(-\frac{\theta}{1+\theta}\frac{u}{\mu_1}\right),$$
$$\mathbb{P}(|U(T)| \leq s \mid T < \infty) = 1 - \exp(-s/\mu_1).$$

Note that the latter formula states that the undershoot follows an exponential distribution with mean μ_1.

5.3 Simulation

In this section, we will first focus on estimating $\psi(u,\tau)$ using simulation. Estimation of this quantity is rather straightforward. In each run, the process is simulated up to time τ and the result of the i-th simulation run Z_i is set to 1 if ruin occurs at or before time τ and to 0 if this is not the case. After n runs, our Monte Carlo estimator is then given by $\hat{Z} = \frac{1}{n}\sum_{i=1}^{n} Z_i$.

The following algorithm can be used to obtain the result of a single simulation run.

1. Initialize the simulation: set $U = u$ and $t = 0$.
2. Draw an exponential interarrival time I with parameter λ. Draw a claim size $X \sim F$.
3. If $t + I > \tau$ return $Z_i = 0$. If not, set $U = U + cI - X$ and if $U < 0$ return $Z_i = 1$. Set $t = t + I$. Otherwise, return to step 2.

Exercise 5.3.1 (P)**.** Simulate the compound Poisson risk model and estimate the probability of ruin before time $T = 100$, with $\lambda = 0.5$, $\theta = 0.1$, $u_0 = 3$, and claim sizes are exponentially distributed with mean 0.5.

Solution:

See the source code in Listing 5.5. An approximate 95% confidence interval, based on 100,000 independent, runs is given by 0.06717 ± 0.001551.

The estimation of $\psi(u,\tau)$ may be straightforward, but the estimation of $\psi(u)$ is less so. Since ruin need not occur within finite time, some of the runs may take an infinite time to finish, which makes a direct implementation impossible. However, such a simulation may be implemented approximately by choosing an appropriate stopping criterion, such as

1. Simulate up to some finite time T_∞ instead of $T = \infty$, and set $\psi(u) \approx \psi(u, T_\infty)$;
2. Simulate until either ruin occurs or the level u_∞ is crossed for the first time;

The idea of the first criterion, is that for large values of T_∞, the value of $\psi(u,T_\infty)$ is close to the value of $\psi(u)$. The rationale behind the second criterion is that for a high level u_∞, the probability of ruin after reaching this level becomes very small. This follows from Lundberg's exponential upper bound, $\psi(u_\infty) \leq \mathrm{e}^{-Ru_\infty}$. So whenever the level u_∞ is reached, we may as well assume that ruin will never happen.

Exercise 5.3.2 (P)**.** Implement and compare both options to estimate the probability of ultimate ruin, $\psi(u)$, in the following cases:

(a) $\lambda = 1$, $\theta = 0.1$ and $X_i \sim \mathrm{Exp}(1)$;

(b) $\lambda = 1$, $\theta = 0.1$ and $X_i \sim \mathrm{Gamma}(3,3)$;

In the first case, also compare the simulation result to the theoretical result.

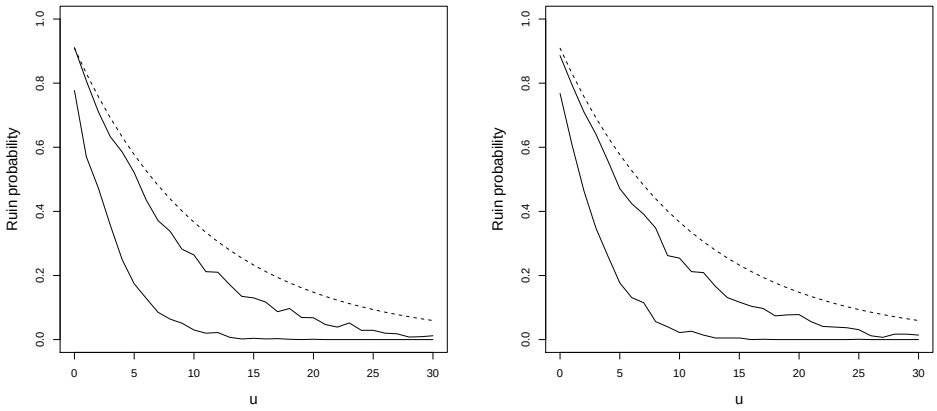
Solution: An implementation of simulation up to time T_∞ can be found in Listing 5.5. An implementation of simulation until either 0 or u_∞ is reached can be found in Listing 5.6. A comparison of the results to the theoretical value is given in Figure 5.3(a)-5.3(b). From these plots, it is easy to see that the quality of the estimate increases with increasing T_∞ resp. u_∞.

Listing 5.5: Using Monte Carlo simulation to estimate $\psi(u,T)$ in the compound Poisson risk model.

```
lambda <- 0.5; # Rate of the underlying Poisson process
claimrate <- 2; # Rate of claim size distribution
u0 <- 3; # Initial level
T <- 10; # Time horizon
theta <- 0.1; # Loading factor
c <- (1 + theta) * lambda / claimrate;

runs <- 100000; # Number of independent runs
result <- c(); # Result

for(r in 1:runs)
{
    time <- 0; # Keep track of the time
    level <- u0; # Keep track of the level
    interarrivaltime <- rexp(1, lambda);

    while(time + interarrivaltime < T)
    {
        # A new claim arrives
        time <- time + interarrivaltime;
        claim <- rexp(1, claimrate);

        # Update level
        level <- level + c*interarrivaltime - claim

        if(level < 0)
        {
            break;
        }

        interarrivaltime <- rexp(1, lambda);
    }

    result <- c(result, level < 0);
}

# Output analysis
m <- mean(result);
v <- var(result);
print(c(m - 1.96 * sqrt(v/runs), m + 1.96 * sqrt(v/runs)));
```

Listing 5.6: Sample code for simulating the compound Poisson risk process until either the level 0 or u_∞ is reached.

```
runs <- 1000; # Number of runs
u0 <- 0; # Initial level
U <- 1000; # Maximal level
c <- 1.1; # Income rate
lambda <- 1; # Arrival rate
claimrate <- 1; # Claim rate

result <- c();

for(r in 1:runs)
{
    time <- 0; # Keep track of the time
    level <- u0; # Keep track of the level
    interarrivaltime <- rexp(1, lambda);

    while(level < U && level >= 0)
    {
        # A new claim arrives
        time <- time + interarrivaltime;
        claim <- rexp(1, claimrate);
        #claim <- rgamma(1, shape=3, rate=3); # Gamma distributed claims

        # Update level
        level <- level + c*interarrivaltime - claim;

        interarrivaltime <- rexp(1, lambda);
    }

    result <- c(result, level < 0);
}

# Output analysis
m <- mean(result);
v <- var(result);
print(c(m - 1.96 * sqrt(v/runs), m + 1.96 * sqrt(v/runs)));
```



(a) Simulation up to time T_∞. The lower line indicates $T_\infty = 10$, the central line indicates the case $T_\infty = 100$.

(b) Simulation until either 0 or u_∞ is reached. The central line indicates the case $u_\infty = 10$ and the central line indicates the case $u_\infty = 100$.

Figure 5.3: Illustration of the two approaches to estimate $\psi(u)$. The dashed line indicates the theoretical value. Claim size distribution is Exp(1). Every value is based on 1,000 independent simulation runs.

6 Models in credit risk

This chapter is concerned with the analysis of credit risk using Monte Carlo simulation.

6.1 Credit risk measurement and the one-factor model

This section intends to be a short introduction to some of the terminology used in credit risk analysis. For more background information, we refer to [BOW03].

Banks and other financial institutions set out loans to *obligors*. These loans are subject to *credit risk*, i.e. risk induced by the fact that some of the obligors may not (fully) meet their financial obligations, for example by not repaying the loan. When this happens, we say that the obligor *defaults*, or is in default.

In order to prevent the event that defaulting obligors will let the bank go bankrupt, the bank needs some sort of insurance. Therefore, it charges a risk premium for every loan set out, and collects these in an internal bank account called an expected loss reserve, or capital cushion.

The question remains, however, how to assess credit risk in order to choose a reasonable risk premium.

The main idea is as follows: a bank assigns to each loan a number of parameters:

- The *Exposure At Default* (EAD), the amount of money subject to be lost;
- A loss fraction, called *Loss Given Default* (LGD), which describes the fraction of the EAD that will be lost when default occurs.
- A *Probability of Default* (PD).

The *loss variable* L is then defined as

$$L := \mathrm{EAD} \times \mathrm{LGD} \times \mathbf{1}_{\{D\}}, \qquad \mathbf{1}_{\{D\}} \sim \mathrm{Bin}(1, \mathrm{PD}) \tag{6.1}$$

Next, consider a portfolio of n obligors, each bearing its own risk. We attach a subscript i to the variables, indicating that they belong to obligor i in the portfolio. We can then define the *portfolio loss* as

$$\widetilde{L}_n := \sum_{i=1}^{n} \mathrm{EAD}_i \times \mathrm{LGD}_i \times \mathbf{1}_{\{D_i\}}, \qquad \mathbf{1}_{\{D_i\}} \sim \mathrm{Bin}(1, \mathrm{PD}_i), \tag{6.2}$$

which is just the sum over all individual loss variables.

The distribution of $\widetilde{L}_n$ is of course of great interest, since it contains all the information about the credit risk. Several characteristics of this distribution have special names. Some of them are

- *Expected Loss* (EL), defined as $\mathrm{EL} := \mathbb{E}\left[\widetilde{L}_n\right]$;
- *Unexpected Loss* (UL), defined as the standard deviation of the portfolio loss, $\mathrm{UL} := \sqrt{\mathrm{Var}\left[\widetilde{L}_n\right]}$; capturing deviations away from EL;
- *Value-at-Risk* (VaR), defined as the α quantile of the loss distribution, $\mathrm{VaR}_\alpha := \inf\{x \geq 0 : \mathbb{P}\left(\widetilde{L}_n \leq x\right) \geq \alpha\}$. Here, α is some number in $(0,1)$.
- *Economic Capital*, defined as $\mathrm{EC}_\alpha = \mathrm{VaR}_\alpha - \mathrm{EL}$. This is the capital cushion.

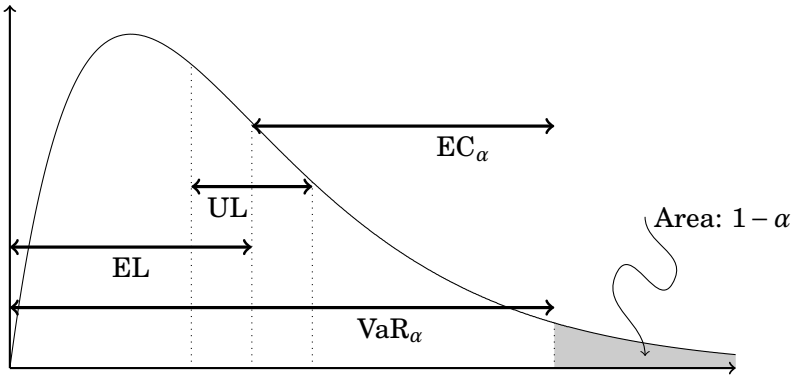


Figure 6.1: Distribution and characteristics of the portfolio loss variable $\widetilde{L}_n$.

Value-at-Risk can be used as a way to choose a suitable *capital cushion*. For example, if the capital cushion is chosen to be equal to the one-year Value-at-Risk (at a level α), the capital cushion is expected to be sufficient in $100 \times (1-\alpha)\%$ of the years. So if $\alpha = 0.998$, the portfolio loss is expected to exceed the capital cushion only twice in 1000 years.

What makes the analysis of the distribution of $\widetilde{L}_n$ interesting, is that variables $\mathbf{1}_{\{D_i\}}$ are usually correlated. This means, for example, when they are positively correlated, knowing that one of the obligors defaults, there is a higher probability of other obligors defaulting as well.

One possible model of correlated obligors in the portfolio is known as the one-factor model. In this model, the probability of default depends on the asset value at the planning horizon; in particular, we assume that there are standard normal random variables $\tilde{r}_i$, modeling the state

of the obligors at the planning horizon.[1] We will denote the critical threshold for obligor i by C_i, and we say that obligor i is in default at the planning horizon, if $\tilde{r}_i < C_i$. It follows immediately that

$$\mathrm{PD}_i = \mathbb{P}(\tilde{r}_i < C_i). \tag{6.3}$$

We will now break up the variable $\tilde{r}_i$ into two parts. One is called the *composite factor*, which, loosely speaking, captures the aspects of the real world that are of influence on the company's economic state (for example: oil prices, the political situation, ...). The other part is the *idiosyncratic part* of the company's economic state, or, as it is often called, the *firm-specific effect*.

We will denote the composite factor by $Z \sim \mathcal{N}(0,1)$ and the idiosyncratic factor of obligor i by $Y_i \sim \mathcal{N}(0,1)$. We will assume that Z and the Y_i are independent. We choose a correlation factor $0 \leq \rho \leq 1$ and set

$$\tilde{r}_i = \sqrt{\rho} Z + \sqrt{1-\rho} Y_i. \tag{6.4}$$

This introduces correlation into the model.

6.1.1 Simulation

If we assume that the values of EAD_i and LGD_i are given constants, a single run of a Monte Carlo simulation can be done as follows:

1. Initialize the portfolio loss, set $\widetilde{L}_n = 0$.
2. Draw a random number $Z \sim \mathcal{N}(0,1)$. This is the composite factor.
3. For each obligor $i = 1, 2, \ldots, n$:
   (a) Draw a random number $Y_i \sim \mathcal{N}(0,1)$. This is the idiosyncratic factor.
   (b) Set $\tilde{r}_i \leftarrow \sqrt{\rho} Z + \sqrt{1-\rho} Y_i$.
   (c) If $\tilde{r}_i < C_i$, set $\widetilde{L}_n \leftarrow \widetilde{L}_n + \mathrm{EAD}_i \times \mathrm{LGD}_i$.

Implementation

After the discussion at the start of this section, building the simulation is straightforward. An implementation of the simulation can be found in Listing 6.1. As you can see, the main part of the source code is a function that performs a single simulation run and this function is executed a number of times. The loss after each run is then stored in an array and after N runs are completed, the estimated density is shown as a histogram. In this example, we choose the following values for the parameters of the model: $\rho = 0.2$, $n = 100$, $\mathrm{EAD}_i = 1$, $\mathrm{LGD}_i = 1$ and $\mathrm{PD}_i = 0.25$ for all $i = 1, 2, \ldots, n$. The number of runs is set to $N = 5000$. The result is shown in Figure 6.2

[1] Usually, the random variable $\tilde{r}_i$ is defined as follows. Suppose that the company's current asset value is denoted by $A_0^{(i)}$, while the value at the planning horizon T is given by $A_T^{(i)}$. We will assume that $A_T^{(i)}/A_0^{(i)}$ follows the so-called log-normal distribution. It follows that $r_i = \log(A_T^{(i)}/A_0^{(i)})$ follows a normal distribution. The values $\tilde{r}_i$ are the standardized r_i, which means that $\tilde{r}_i = (r_i - \mathbb{E}[r_i])/\sqrt{\mathrm{Var}[r_i]} \sim \mathcal{N}(0,1)$.

Listing 6.1: Implementation of a simulation of the one factor model.

```
# n: number of obligors in the portfolio.
# PD: vector of default probabilities.
# EAD: vector of EADs.
# LGD: vector of LGDs.
# rho: correlation factor.
lossrealization <- function(n, PD, EAD, LGD, rho)
{
    # Keep track of the loss in this portfolio.
    totalloss <- 0;

    # Draw a normal random variable for the
    # systematic factor.
    sf <- rnorm(1,0,1);

    # Loop through all obligors to see if they
    # go into default.
    for(obligor in 1:n)
    {
        # Draw idiosyncratic factor.
        of <- rnorm(1,0,1);

        # Asset value for this obligor.
        rtilde <- sqrt(rho)*sf + sqrt(1-rho)*of;

        # Critical threshold for this obligor.
        c <- qnorm(PD[obligor], 0, 1);

        # check for default.
        if(rtilde < c)
        {
            totalloss <- totalloss + EAD[obligor] * LGD[obligor];
        }
    }

    return(totalloss);
}

n <- 100; # The number of obligors in the portfolio.
runs <- 5000; # Number of realizations.

# Run a number of realizations.
EAD <- rep(1, n);
LGD <- rep(1, n);
PD <- rep(0.25, n);
rho <- 0.2;
losses <- c();
for(run in 1:runs)
{
    # Add a new realization of the loss variable.
    losses <- c(losses, lossrealization(n, PD, EAD, LGD, rho));
}

# Output: normalised histogram.
hist(losses, freq=FALSE, main="Histogram of Loss",
    xlab="Loss", ylab="Density");

alpha <- 0.95; # Alpha level

# Sort losses and select right value
losses <- sort(losses);
j <- floor(alpha*runs);
var <- losses[j];
```

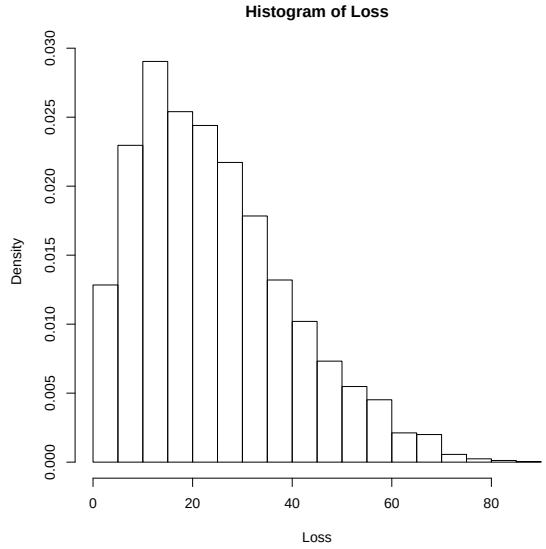


Figure 6.2: A histogram of the loss distribution in the one factor model, based on 5000 realisations.

6.1.2 Estimating the value-at-risk

The Value-at-Risk is just the α-quantile of the loss distribution. From basic statistics, it is known that empirical quantiles can be based on an ordered random sample.

Assume that L_i, $i = 1,2,\ldots,N$ is an independent realization from the loss distribution $\widetilde{L}_n$ (the simulation output, N is the number of runs), and denote by $L_{(i)}$ the order statistics, that is, the same sequence, but ordered in an increasing fashion, so $L_{(1)} \leq L_{(2)} \leq \ldots \leq L_{(N)}$. Now choose j such that

$$\frac{j}{N} \leq \alpha < \frac{j+1}{N}. \tag{6.5}$$

Then $L_{(j)}$ is an approximation of the $(1-\alpha)$-quantile of the loss distribution.

Exercise 6.1.1 (P)**.** Using simulation, estimate the 75% quantile of the standard normal distribution.

Solution:

See Listing 6.2. The sample source code can be used to generate 1,000 estimates, each based on 10,000 independent random numbers. Hence, an approximate 95% confidence interval can be found, in this case we find 0.675 ± 0.003. Using R's `qnorm` function, we find that the true value of the 75% quantile of the standard normal distribution is approximately 0.6745.

Exercise 6.1.2 (P)**.** Extend the source code in Listing 6.1, such that it can be used to estimate VaR_α.

Listing 6.2: Sample code for estimating the 75% quantile of the standard normal distribution.

```
runs <- 100000; # Number of independent runs
q <- 0.75; # Quantile

rnd <- rnorm(runs, 0, 1); # Get random sample
rnd <- sort(rnd); # Sort random sample

# Select the right quantile
j <- floor(runs * q);
rnd[j]
```

Listing 6.3: Source code used for estimating VaR_α.

```
alpha <- 0.95; # Alpha level

# Sort losses and select right value
losses <- sort(losses);
j <- floor(alpha*runs);

# Output VaR
cat('value at risk (alpha=', alpha, '): ', losses[j], sep='');
```

Solution:

Append the source code in Listing 6.3 to the source code in Listing 6.1 to obtain an estimate of the value-at-risk at level α.

Exercise 6.1.3 (P)**.** The expected shortfall, or tail conditional expectation, is the expected value of the portfolio loss, given that it exceeds the value-at-risk (at level α), that is,

$$\mathrm{TCE}_\alpha := \mathbb{E}\left[\widetilde{L}_n \,|\, \widetilde{L}_n \geq \mathrm{VaR}_\alpha\right].$$

Extend the source code in Listing 6.1, such that it can be used to estimate the expected shortfall.

Solution:

Listing 6.4: Source code used for estimating TCE.

```
alpha <- 0.95; # Alpha level

# Sort losses and select right value
losses <- sort(losses);
j <- floor(alpha*runs);
VaR <- losses[j];

# Select the losses that are larger than VaR
largelosses <- losses[losses>=VaR];

# Output TCE
TCE <- mean(largelosses);
cat("tail conditional expectation (alpha=",
     alpha, "): ", TCE, sep="");
```

Append the source code in Listing 6.4 to the source code in Listing 6.1 to obtain an estimate of the value-at-risk at level α.

6.2 The Bernoulli mixture model

The setting of the *Bernoulli mixture model* is the same as that of the one-factor model. Again, we have n obligors, which we will call obligor $1, 2, \ldots, n$. For each obligor i, there is a random variables $\mathbf{1}_{\{D_i\}}$ indicating whether it defaults or not: $\mathbf{1}_{\{D_i\}} = 1$ means that obligor i defaults, while $\mathbf{1}_{\{D_i\}} = 0$ indicates that obligor i does not default.

We will assume that there exists some underlying random variable P, which takes values in $[0,1]$, and that, given $P = p$, the $\mathbf{1}_{\{D_i\}}$ are independent and follow a Bernoulli distribution with parameter p:

$$\mathbf{1}_{\{D_i\}} | P \sim \text{Bin}(1, P). \tag{6.6}$$

The cdf of P is often called the *mixture distribution function*. It can be shown that for $i \neq j$ the correlation between $\mathbf{1}_{\{D_i\}}$ and $\mathbf{1}_{\{D_j\}}$ is given by

$$\text{corr}\left(\mathbf{1}_{\{D_i\}}, \mathbf{1}_{\{D_j\}}\right) = \frac{\text{Var}[P]}{\mathbb{E}[P](1 - \mathbb{E}[P])}. \tag{6.7}$$

Typical choices for the mixture distribution are the Beta distribution, or the probit-normal distribution, $\Phi^{-1}(P) \sim \mathcal{N}\left(\mu, \sigma^2\right)$, with Φ the standard normal cdf.

6.2.1 Simulation

The scheme for simulating the portfolio loss $\widetilde{L}_n$ is now straightforward:

(i) Initialize the portfolio loss, set $\widetilde{L}_n \leftarrow 0$;

(ii) Draw a single realization of P from the mixture distribution function;

(iii) For each obligor $i = 1, 2, \ldots, n$:

  (a) Draw $\mathbf{1}_{\{D_i\}} \sim \text{Bin}(1, P)$;

  (b) If $\mathbf{1}_{\{D_i\}} = 1$, set $\widetilde{L}_n \leftarrow \widetilde{L}_n + \text{EAD}_i \times \text{LGD}_i$.

Exercise 6.2.1. Write a simulation to estimate $\text{VaR}_{0.95}$ in the Bernoulli mixture model, where P is drawn from a beta distribution with parameters 1 and 3. Also, plot a histogram of the losses.

Solution: The scheme in the section "Simulation" shows how the simulation should be implemented; a sample implementation can be found in Listing 6.5. We find that $\text{VaR}_{0.95} \approx 64$. A histogram of the losses can be found in Figure 6.3.

Listing 6.5: Sample implementation of a simulation of the Bernoulli mixture model.

```
# n: number of obligors in the portfolio
# PD: vector of default probabilities
# EAD: vector of EADs
# LGD: vector of LGDs
# b1, b2: parameters of the beta distribution
lossrealization <- function(n, PD, EAD, LGD, b1, b2)
{
    # Keep track of the loss in this portfolio.
    totalloss <- 0;

    # Draw Q
    Q <- rbeta(1, b1, b2);

    # Loop through all obligors to see if they
    # go into default.
    for(obligor in 1:n)
    {
        # Does this obligor go into default or not?
        default <- rbinom(1, 1, Q);

        # check for default.
        if(default == 1)
        {
            totalloss <- totalloss + EAD[obligor] * LGD[obligor];
        }
    }

    return(totalloss);
}

runs <- 10000; # Number of runs
n <- 100; # Number of obligors in the portfolio

# Run a number of realizations.
EAD <- rep(1, n);
LGD <- rep(1, n);
PD <- rep(0.25, n);
b1 <- 1;
b2 <- 3;
losses <- c();
for(run in 1:runs)
{
    # Add a new realization of the loss variable.
    losses <- c(losses, lossrealization(n, PD, EAD, LGD, b1, b2));
}

# Output: normalised histogram.
hist(losses, freq=FALSE, main="Histogram of Loss",
    xlab="Loss", ylab="Density");

alpha <- 0.95; # Alpha level

# Sort losses and select right value
losses <- sort(losses);
j <- floor(alpha*runs);
var <- losses[j];
```

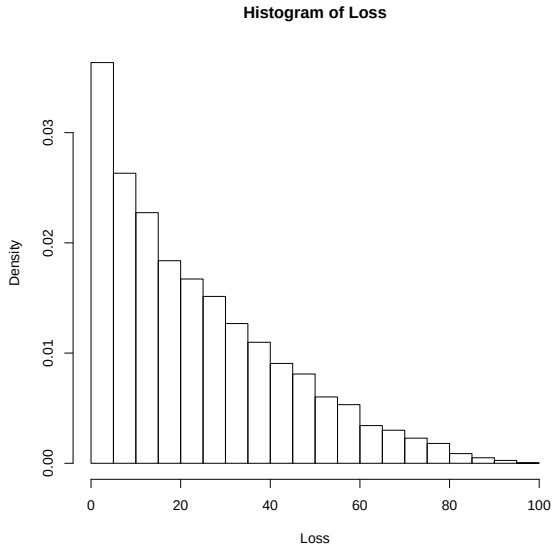


Figure 6.3: Histogram of the losses in the Bernoulli mixture model.

7 Option pricing

This chapter is devoted to option pricing using Monte Carlo simulation.

In the first section, we will give a brief overview of the underlying model (stock prices driven by a geometric Brownian motion) that we will be using and a short list of various types of options.

In the second and third section, we consider Monte Carlo simulation of the underlying process and how these techniques can be used to value different styles of options.

The fourth section describes a more advanced way of simulating Brownian motion, using wavelets.

7.1 The underlying process and options

7.1.1 Options

There exist two types of financial market instruments:

- *Stocks* – e.g. shares, bonds and commodities;
- *Derivatives* – promises of some payment or delivery in the future, that are called derivatives because their value is derived from the underlying stock (or stocks).

Stocks prices are often modelled as stochastic processes (S_t) where t is a (discrete or continuous) index indicating time. The current stock price is denoted S_0.

Options are a specific type of derivative, that give the holder the right (but not the obligation) to buy or sell a certain amount of the underlying stock on (or sometimes before) a specified date (the *expiration date* or *maturity*) for a specified price (the strike price. We will denote the maturity by T and the strike price by K.

Options that give the holder the right to buy are called *call options*, while options that give the holder the right to sell are called *put options*. When an option is used to buy or sell stock, we say that it is exercised.

Options come in many flavours, of which we will mention the two that are most well-known.

(i) *European options.* These options give the holder the right to buy (call option) or sell (put option) a number of units of the underlying stock on time T for strike price K. The pay-off of a European call option is $(S_T - K)^+$ and that of a put option is $(K - S_T)^+$, where $(x)^+ = \max(x, 0)$.

(ii) *American options.* These are like European options, but they can be exercised at or before time T.

7.1.2 Modelling the stock price

A model that is often used for financial markets is the Black-Scholes model that is named after Fisher Black and Myron Scholes, who described the model in their 1973 paper "The pricing of options and corporate liabilities" [Bla73]. Black and Scholes model the stock price as a geometric Brownian motion. That is, given the current stock price S_0, the stock price process $(S_t)_{t\geq 0}$ is such that

$$\log\left(\frac{S_t}{S_0}\right) \tag{7.1}$$

is a Brownian motion.

7.1.3 Valuation of options

An important problem in mathematical finance is the valuation – or pricing – of derivatives. At the moment that a derivative is taken out, it is not known what its value will be in the future, as it depends on the future value of the underlying stock.

7.2 Simulating geometric Brownian motion

7.2.1 Simulating Brownian motion

We will start off with the definition of the standard Brownian motion, which forms the basis of the Black-Scholes model.[1].

Definition 7.2.1. A *standard Brownian motion (SBM)* $(B_t)_{t\geq 0}$ is a stochastic process having

[1]Brownian motion is one of the most well-known and fundamental stochastic processes. It was first studied in 1827 by botanist Robert Brown, who observed a certain jiggling sort of movement pollen exhibited when brought into water. Later, it was studied by Albert Einstein, who described it as the cumulative effect many small water particles had on the trajectory of the pollen. Even later, the Brownian motion was given a rigorous mathematical treatment by Norbert Wiener, which is why it is often called the Wiener process.
Einsteins explanation of the Brownian motion explains why the Brownian motion is a popular model for stock prices. Namely, stock prices are the result of many individual investors. In this section, we will price options within this model.
For much more information on Brownian motion, see e.g. [Ross96] or the lecture notes by Joe Chang [Chang99]

(i) continuous sample paths;

(ii) independent increments (for each $t > s > 0$, $B_t - B_s$ is independent of the values $(B_u)_{0 \le u \le s}$);

(iii) stationary increments (for each $t > s$, the distribution of $B_t - B_s$ depends only on $t - s$);

(iv) $B_0 = 0$;

(v) increments are stationary and normally distributed (for each $t \ge s \ge 0$, $B_t - B_s \sim \mathcal{N}(0, t-s)$).

Properties *(ii)* and *(iii)* show that if $s < t$, we have $B_t = Z_1 + Z_2$, where $Z_1 \sim \mathcal{N}(0,s)$ and $Z_2 \sim \mathcal{N}(0,t-s)$. This inspires the following discrete approximation of the Brownian motion.
Let T be our time horizon; we will approximate sample paths from the Brownian motion on the time interval $[0,T]$. Let M be large and let $\tau = T/M$. By definition, for each $i = 1,2,\ldots,M$, we have

$$L_i := B_{i\tau} - B_{(i-1)\tau} \sim \mathcal{N}(0,\tau), \tag{7.2}$$

and all these L_i are mutually independent. Note that

$$L_1 + L_2 + \ldots + L_k = B_{k\tau} - B_0 = B_{k\tau} \tag{7.3}$$

Thus, we can use $\sum_{i=1}^{k} L_i$ as a discretized approximation to the process up to time $k\tau$. Increasing M makes the grid finer. It should be noted that discrete sampling of the Brownian motion yields just a random walk with $\mathcal{N}(0,\tau)$-distributed increments.

The above discussion is formalised in the following theorem:

Theorem 7.2.1. *Let $0 = t_0 < t_1 < t_2 < \ldots < t_M = T$ be a subdivision of the interval $[0,T]$, and $Z_1, Z_2, \ldots, Z_M \sim \mathcal{N}(0,1)$ be independent. Define*

$$X_0 = 0, \qquad X_i = X_{i-1} + \sqrt{t_i - t_{i-1}} Z_i \quad (i = 1,2,\ldots,M), \tag{7.4}$$

then $(X_0, X_1, X_2, \ldots, X_M)$ is equal in distribution to $(B_{t_0}, B_{t_1}, B_{t_2}, \ldots, B_{t_M})$, where $(B_t)_{t \ge 0}$ is a standard Brownian motion.

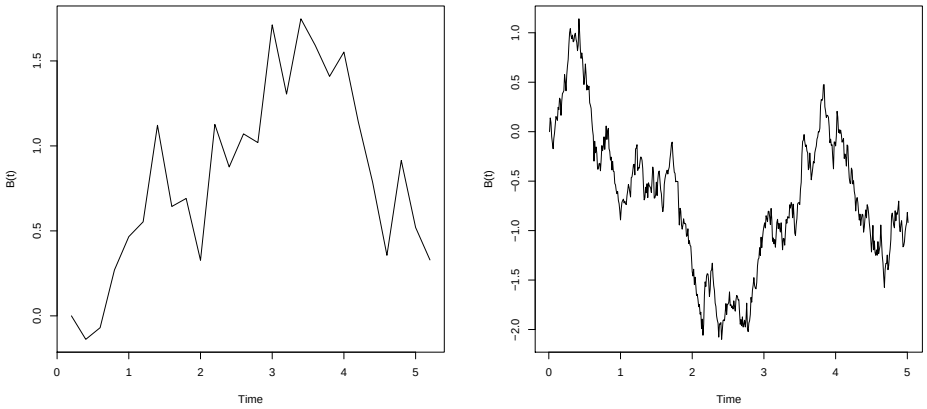
A sample path of the standard Brownian motion can be found in Figure 7.1.

Exercise 7.2.1 (P)**.** Simulate a sample path from the standard Brownian motion on the interval $[0,5]$. Experiment with the parameter M.

Solution:

Listing 7.1: Simulation of the standard Brownian motion.

```
1  T <- 5; # Time horizon
2  M <- 500; # Split time interval up in M pieces
3
4  grid <- T/M; # Grid width
5
6  B <- c(0); # Contains samples from BM
7
8  for(i in 1:M)
9  {
10     # B contains the partial sums of L.
11     increment <- rnorm(1, 0, sqrt(grid));
```



(a) $M = 25$. (b) $M = 500$.

Figure 7.1: Sample path of the standard Brownian motion.

```
12      oldB <- B[length(B)];
13      B <- c(B, oldB + increment);
14  }
15
16  # Make a nice plot
17  plot(1:length(B), B, type="l", xlab="Time", ylab="B(t)", xaxt="n");
18  axis(1, at=(0:T)/T*M, labels=0:T);
```

See the source code in Listing 7.1. Sample paths of the standard Brownian motion for different values of M can be found in Figure 7.1.

7.2.2 General Brownian motion

Let $(B_t)_{t\geq 0}$ be a standard Brownian motion. A stochastic process $(S_t)_{t\geq 0}$ is called a *Brownian motion* (also (μ,σ^2)-Brownian motion) if it is equal in distribution to

$$S_t = \mu t + \sigma B_t. \tag{7.5}$$

The parameters μ and $\sigma > 0$ are called the drift and the volatility of the Brownian motion. For example, if $\mu > 0$, the process has the tendency to increase over time, while a large value of σ increased the variability (volatility) of the process.

The simple formula (7.5) shows that if we are able to simulate a standard Brownian motion, then we are also able to simulate a general Brownian motion.

Exercise 7.2.2 (P)**.** Simulate sample paths from the (μ,σ^2)-Brownian motion in the following cases:

(i) $\mu = 1$, $\sigma = 1$;

(ii) $\mu = 1$, $\sigma = 5$;

(iii) $\mu = -0.1$, $\sigma = 1$.

7.2.3 Simulating geometric Brownian motion

Brownian motion does not seem to be adequate as a model for stock markets, because, for example, it would allow for negative stock prices. The Black-Scholes model uses a *geometric Brownian motion* instead. If we denote, as before, by $(B_t)_{t\geq 0}$ a standard Brownian motion, then the process

$$S_t = S_0 \exp(\mu t + \sigma B_t) \tag{7.6}$$

is a geometric Brownian motion with drift μ and volatility σ. $(S_t)_{t\geq 0}$ will be the model of our stock prices.

As before, the simple formula (7.6) allows us to simulate a geometric Brownian motion, as soon as we are able to simulate the standard Brownian motion.

7.2.4 The martingale measure

Pricing options in the GBM model can be done using the risk-neutral measure. Under this measure, the stock price follows a geometric Brownian motion with drift $r - \sigma^2/2$ and volatility σ. Here, r is the continuously compounded interest rate.

A general option has a pay-off function that depends on the sample path of the underlying Brownian motion, say

$$\text{pay-off} = F((S_t)_{0\leq t\leq T}) \tag{7.7}$$

for some function G. The price of this option then will be

$$C = \mathrm{e}^{-rT}\mathbb{E}\left[F((S_0 \exp((r-\sigma^2/2)t + \sigma B_t)_{0\leq t\leq T})\right], \tag{7.8}$$

with $(B_t)_{t\geq 0}$ a Brownian motion and r is the continuously compounded interest rate. For details, see e.g. [Etheridge02] or the lecture notes by Harry van Zanten [Zanten10].

7.3 Option pricing in the GBM model

7.3.1 European options

Pricing European options using Monte Carlo simulation is relatively easy, as the price of the option only depends on the stock price path $(S_t)_{0\leq t\leq T}$ through the value S_T: the stock price at maturity. For now, we will focus on the European call options, but simulating put options works just the same.

In the case of European call options, the function F from the previous sections will be

$$F((S_t)_{0\leq t\leq T}) = (S_T - K)^+, \tag{7.9}$$

with K the strike price of the option. In other words: the exact dynamics of the underlying process are not important, we only care about the value

$$S_T = S_0 \exp((r-\sigma^2/2)T + \sigma B_T), \tag{7.10}$$

which is easily generated, once we are able to generate the value of the standard Brownian motion at time T, which is just an $\mathcal{N}(0,T)$-distributed random variable! Sample source code for a Monte Carlo algorithm for pricing European call options can be found in Listing 7.2.

Listing 7.2: A Monte Carlo algorithm for pricing European call options in the GBM model.

```
T <- 5; # Time horizon
r <- 0.03; # Interest rate
sigma <- 0.05; # Volatility
runs <- 10000; # Number of runs
S0 <- 1; # Initial stock level
K <- 1.01; # Strike price

results <- c(); # Result vector

# Simulate stock price under risk neutral measure
stockprice <- function(r, sigma, T, S0)
{
    Z <- rnorm(1, 0, sqrt(T));
    return( S0*exp((r-sigma*sigma/2)*T + sigma*Z) );
}

# Calculate payoff given stock price and strike price
payoff <- function(s, K)
{
    return(max(s-K,0));
}

# Calculation
for(run in 1:runs)
{
    s <- stockprice(r, sigma, T, S0);
    p <- payoff(s, K);
    results <- c(results, exp(-r*T)*p);
}

# Mean and variance
mean(results)
var(results)
```

Exercise 7.3.1 (P)**.** Adapt the source code in Listing 7.2 so that the algorithm prices European put options instead of European call options.

Solution: To price put options instead, change `return(max(s-K,0));` into `return(max(K-s,0));`.

Exercise 7.3.2 (P)**.** A binary put option is an option that pays out one unit of cash if the stock price at maturity is below the strike price. Adapt the source code in Listing 7.2 so that it prices such a binary put option.

Solution: To price the binary put option, it suffices to replace the `payoff(s,K)`-function, see Listing 7.3.

Listing 7.3: Pay-off function for a binary put option.

```
# Calculate payoff given stock price and strike price
payoff <- function(s, K)
{
    p <- 1;
    if(s > K) p <- 0;
    return(p);
}
```

7.3.2 General options

The general version of Equation (7.7) shows that the payoff of an option may depend on the whole sample path, rather than only on the value of the stock at maturity, as is the case with European options. Pricing a general option therefore needs simulation of the whole sample path of a geometric Brownian motion rather than only its value at maturity.

We can adapt the "discretisation" strategy for simulating sample paths of the standard Brownian motion to simulate sample paths of the geometric Brownian motion. As an example, we will consider Asian call options. The value of an Asian call option is based on the average price of the underlying over the time until maturity. Its pay-off may be given as the maximum of the average value over the time until maturity minus the strike, and zero. A sample implementation can be found in Listing 7.4.

Listing 7.4: A Monte Carlo algorithm for pricing Asian call options in the GBM model.

```
T <- 5; # Time horizon
M <- 1000; # Number of steps
r <- 0.03; # Interest rate
sigma <- 0.05; # Volatility
runs <- 1000; # Number of runs
S0 <- 1; # Initial stock level
K <- 1.01; # Strike price

results <- c(); # Result vector

# Simulate stock path under risk neutral measure
stockprice <- function(r, sigma, T, M, S0)
{
    path <- c(S0); # Path to be generated
    dt <- T/M; # Time step

    # Simulate a sample path
    prev <- S0; # Previous value
    for(step in 1:M)
    {
        Z <- rnorm(1, 0, sqrt(dt));
        s <- prev*exp((r-sigma*sigma/2)*dt + sigma*Z);
        path <- c(path, s);
        prev <- s;
    }

    return(path);
}

# Calculate payoff given stock price and strike price
payoff <- function(s, K)
{
    return(max(mean(s)-K,0));
}

# Calculation
for(run in 1:runs)
{
    s <- stockprice(r, sigma, T, M, S0);
    p <- payoff(s, K);
    results <- c(results, exp(-r*T)*p);
}

# Mean and variance
mean(results)
```

```
46  var(results)
```

Exercise 7.3.3 (P)**.** Adapt the source code in Listing 7.4 so that it can be used to price lookback put options that expire in 5 months and look back two months.

A lookback option gives the holder the right to buy/sell the underlying stock at its lowest/highest price over some preceding period.

Solution: To price the binary put option, we need to extend the `payoff(s,K)`-function. It needs to find the minimum value in the second half of the list of sample points `s`, starting from the index corresponding to the moment that the lookback period starts (in this case, after three months), for which we will use

$$\frac{3}{5} \times \#\text{sample points} = \frac{t}{T} \times \#\text{sample points}, \tag{7.11}$$

rounded up. See Listing 7.5.

Listing 7.5: Pay-off function for a lookback put option.

```
1  # Calculate payoff given stock price and strike price
2  payoff <- function(s, K, T, t)
3  {
4      # Index at which the lookback period starts
5      index <- ceiling((t/T)*length(s));
6      # Select lowest value
7      lowest <- min(s[index:length(s)]);
8
9      return(s[length(s)]-lowest);
10 }
```

7.4 Advanced topic: Simulation of Brownian motion using wavelets

Under construction.

Bibliography

[Bla73] Black F. and Scholes M., *The pricing of options and corporate liabilities*, Journal of Political Economy, 81(3):637-654.

[BOW03] Bluhm C., Overbeck L., Wagner C., *An Introduction to Credit Risk Modeling*, CRC Press, 2003.

[Chang99] Chang J., *Brownian Motion*, UC Santa Cruz, 1999. Available via http://users.soe.ucsc.edu/~chang/203/bm.pdf.

[Etheridge02] Etheridge A., *A Course in Financial Calculus*, Cambridge University Press, 2002.

[Kaas01] Kaas R., Goovaerts M., Dhaene J., Denuit M., *Modern Actuarial Risk Theory*, Kluwer Academic Publishers, 2001.

[Ross96] Ross S.M., *Stochastic Processes*, Wiley, 1996.

[Ross99] Ross S.M., *An Introduction to Mathematical Finance: Options and Other Topics*, Cambridge University Press, 1999.

[Ross07] Ross S.M., *Introduction to Probability Models*, Academic Press, 2007.

[WikiA] Wikipedia, *Student's t-distribution*, retrieved September 8, 2011, via http://en.wikipedia.org/wiki/Student's_t-distribution.

[Zanten10] Van Zanten H., *Introductory lectures on derivative pricing*, Eindhoven University of Technology, 2010. Available via http://www.win.tue.nl/~jzanten.