



## SIMULATION

### 1. Introduction

- **Basic idea:** To use methods based on random number generation to simulate “complex” processes to get an insight about the random behavior of some important variables;
- The first “real” use of simulation was related to the “Manhattan project” at Los Alamos at the end of World War 2 (nuclear bomb) and was linked to the names of von Neumann, Stanislaw Ulam and Nicholas Metropolis. Since then, simulation techniques are used in many scientific domains and cover a great variety of applications.
- The use of simulation to obtain an approximation to the sampling distribution of a statistic illustrates the process. The process can be summarized in three stages:
  - a. Simulate the production of random samples from the population;
  - b. For each sample generated in the previous stage, get the value of the statistic for which we want to approximate the sampling distribution;
  - c. Now we can use the simulated values of the statistic to get an approximation to its sampling distribution (each replica provides an “observed value” of the statistic).
- Two problems remain: how to generate the random samples and how many replicas should we use.



## 2. How to generate random samples from a population with a known distribution?

- We follow a two-step procedure:
  - a. First, we generate pseudorandom numbers, i.e. values that can be considered as independent observations of a random variable with a uniform  $(0;1)$  distribution;
  - b. Second, using probability theory we transform the pseudorandom numbers to get pseudo random observations from the wanted distribution.

### 2.1 – Pseudo random numbers generation

- **Random numbers versus pseudorandom numbers.** A random number is a realization of a continuous random variable following a uniform distribution between 0 and 1. All the generated variables should be independent from each other. As it is not possible to generate these numbers (the best we can get is to generate discrete variables) we will use pseudorandom numbers.
- A **pseudorandom number generator** is an algorithm for generating a sequence of numbers that look like independent observations of a  $U(0;1)$  distribution. Obviously, the sequence is not random as it is completely determined by a relatively small set of initial values.



- One common method to generate pseudorandom numbers is based on the multiplicative linear congruential generator introduced by Lehmer in 1949. The idea is the following

$$n_t = a n_{t-1} \bmod c \quad t = 1, 2, \dots$$

$$u_t = n_t / c$$

where  $u_t$  is  $t$ -th generated random number ( $t = 1, 2, \dots$ ) and  $a$ ,  $c$  and  $n_0$  are three positive integers with adequate properties.  $n_0$  is called the seed.

- Example

Let  $a = 123$ ,  $c = 1000$  and  $n_0 = 135$ . Generate 25 pseudo random numbers. Verify that  $n_{20} = n_0$  and comment (Use Excel or another software).

- As illustrated by the example, the maximum length of the sequence before it begins to repeat can be a major concern. Another important points are to verify if we can assume that the observations **(i)** could have been generated by an independent process and **(ii)** mimic a  $U(0;1)$  behavior.

Parameters  $a$ ,  $c$  and  $n_0$  are chosen to answer these points.

- Pseudorandom generation is still an area of active research (one main application of random numbers is cryptography!). The Mersenne-Twister algorithm, introduced in 1997, is replacing linear congruential generator as the “standard”.



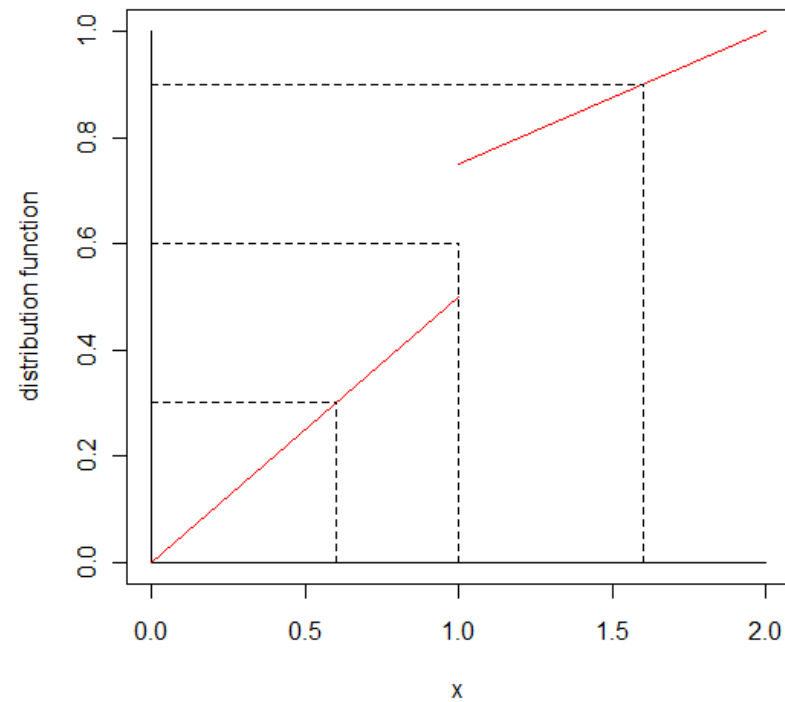
## 2.2. From a uniform distribution to other distributions

- Assuming that we got a sequence of independently distributed uniform variables between 0 and 1, the problem is now how to define functions of these variables that follow a given distribution. We can give a general answer to this problem, but in many situations we can design more efficient methods.
- The general answer is called the **inverse transform** and is based on the following theorem:
- **Theorem:** Let  $Y \sim U(0;1)$  and  $F(x)$  be the distribution function of a continuous random variable (we are assuming that  $F$  is strictly increasing on the open interval  $(a,b)$  where  $X$  has positive density – the interval can be unlimited – with  $F(a) = 0$  and  $F(b) = 1$ ). Then the random variable  $X = F^{-1}(Y)$  is continuous and  $F(x)$  is its distribution function.
- Example: Illustrate the procedure using (i) the exponential distribution and (ii) the normal distribution
- Extensions:
  - Suppose that the distribution function has a jump at  $x = c$ , i.e.  $F(c^-) = a$  and  $F(c) = b$  with  $b > a$ . If the random number  $u$  is such that  $a \leq u < b$ , choose  $c$  as the simulated value;
  - Suppose that the distribution function is constant in a given interval, i.e.  $F(x) = p$  for  $a \leq x \leq b$ . If the random number  $u$  is equal to  $p$  (abnormal case) choose  $x = b$ ;



- **Example 20.2** – Suppose  $F_X(x) = \begin{cases} 0.5x & 0 \leq x < 1 \\ 0.5 + 0.25x & 1 \leq x \leq 2 \end{cases}$

Determine the simulated values of  $x$  resulting from the uniform numbers 0.3, 0.6 and 0.9.





- **Specific answers** - For many situations one can design a computationally more efficient (and reliable) method to use with specific distributions. The idea is to take advantage of the properties of these distributions.
  - Bernoulli
  - Binomial as the sum of independent Bernoulli variables
  - Normal (Box-Muller transform, Central Limit Theorem)
  - Poisson (Time between events follows an exponential distribution)
  - Discrete mixture:  $F_Y(y) = \alpha_1 F_{X_1}(y) + \alpha_2 F_{X_2}(y) + \dots + \alpha_k F_{X_k}(y)$ 

A 2-steps procedure usually works better than a direct approach:

    - Simulate a discrete random variable where  $\Pr(J = j) = a_j, j = 1, 2, \dots, k$ .
    - Simulate  $X_j$  using an appropriate method
- Box-Muller formula:  $(U_1, U_2)$  independent uniform variables, then  $Z_1 = \sqrt{-2 \ln U_1} \cos(2\pi U_2)$  and  $Z_2 = \sqrt{-2 \ln U_1} \sin(2\pi U_2)$  are independent  $n(0,1)$  random variables.
- **Example:** generate 10 uniforms and then generate (i) 10 Bernoulli variables with parameter 0.2 (ii) 10 normally distributed variables (mean=3, standard deviation=2) using the inverse method and Box-Muller formula.



- **Example 20.1** – generate 10000 pseudo Pareto (with  $\alpha = 3$  and  $\theta = 1000$ ) variables and verify that they are indistinguishable from real Pareto observations.

Procedure:

- Generate 10000 (pseudo) uniforms,  $u_i$
- Obtain 10000 (pseudo) Pareto,  $x_i = \theta \left( (1 - u_i)^{-1/\alpha} - 1 \right)$
- Compare the generated values with the theoretical model:
  - Kolmogorov-Smirnov test
  - Anderson-Darling test
  - $\chi^2$  goodness of fit test (*Loss Models* solution )
  - Other approaches (graphical techniques, ...)

A possible solution using R

```
> # generate n Pareto (alpha,theta) variables - inverse method
> n=10000; alpha=3; theta=1000
> u=runif(n); x=theta*((1-u)^(-1/alpha)-1)
>
> Pareto_dist_func=function(x,alpha,theta) 1-(theta/(x+theta))^alpha
```



```
> # KS test
> ks.test(x,Pareto_dist_func,alpha=3,theta=1000)
```

One-sample Kolmogorov-Smirnov test

```
data: x
D = 0.008, p-value = 0.5499
alternative hypothesis: two-sided
>
```

```
> # AD test
> library(ADGofTest)
> ad.test(x,Pareto_dist_func,alpha=3,theta=1000)
```

Anderson-Darling GoF Test

```
data: x and Pareto_dist_func
AD = 1.256, p-value = 0.2471
alternative hypothesis: NA
> # Qui2 test for large samples - m classes
> m=100; aux1=(1:(m-1))/m; aux2=theta*((1-(aux1))^(1/alpha)-1);
> lb=c(0,aux2); ub=c(aux2,Inf); counts=rep(NA,m)
> for (j in 1:m) counts[j]=sum((x>=lb[j]) & (x<ub[j]))
```





```
> expected=rep(n/m,m)
> chi2=((counts-expected)^2)/expected
> chi2.test=sum(chi2)
> p.value=pchisq(chi2.test,m-1,lower.tail=FALSE)
> cat("chi-square statistic = ",chi2.test,"\n","p-value =
",p.value,"\n")
chi-square statistic = 104.32
p-value = 0.3377174
>
> result=cbind(lb,ub,counts,expected,chi2)
> result
```

	lb	ub	counts	expected	chi2
[1,]	0.000000	3.355730	97	100	0.09
[2,]	3.355730	6.756962	94	100	0.36
...					
[99,]	2684.031499	3641.588834	114	100	1.96
[100,]	3641.588834	Inf	87	100	1.69



### 3. How many replicas should be used?

- The answer depends on the problem we want to solve.
- Nowadays we can define a **huge number** of replicas for many situations. This is the usual solution.
- Sometimes, using the Central Limit Theorem we can define an approximate value to the number of replicas to ensure a given precision. Example 21.5 illustrates 3 situations.
- **Example 20.12** – Use simulation to estimate the mean,  $F_X(1000)$  and  $\pi_{0.9}$ , the 90<sup>th</sup> percentile of the Pareto distribution with  $\alpha = 3$  and  $\theta = 1000$ . In each case, stop the simulation when you are 95% confident that the answer is within  $\pm 1\%$  of the true value.
  - As we know the true values, the simulation is useless but we will behave as these values are unknown ( $\mu = 500$ ,  $F_X(1000) = 0.875$ ,  $\pi_{0.9} = 1000 \times (0.1^{-1/3} - 1) \approx 1154.435$ )
  - We will discuss only the first 2 situations.



- **Mean ( $\mu$ ):** The usual estimator is the statistic  $T = \bar{X}$  and  $\frac{T - \mu}{\sigma / \sqrt{n}} \sim n(0;1)$

$$\Pr(|T - \mu| \leq 0.01\mu) = \Pr(|T - \mu| \leq 0.01\mu) = \Pr\left(-\frac{0.01\mu}{\sigma / \sqrt{n}} \leq \frac{T - \mu}{\sigma / \sqrt{n}} \leq \frac{0.01\mu}{\sigma / \sqrt{n}}\right) = 2\Phi\left(\frac{0.01\mu}{\sigma / \sqrt{n}}\right) - 1$$

$$\text{Then } \Pr(|T - \mu| \leq 0.01\mu) \geq 0.95 \Leftrightarrow \Phi\left(\frac{0.01\mu}{\sigma / \sqrt{n}}\right) \geq 0.975 \Leftrightarrow \frac{0.01\mu}{\sigma / \sqrt{n}} \geq 1.96 \Leftrightarrow n \geq \left(\frac{1.96}{0.01}\right)^2 \times \left(\frac{\sigma^2}{\mu^2}\right)$$

As  $\mu$  and  $\sigma^2$  are unknown we run a first simulation to estimate them:  $\tilde{\mu} = \bar{x}$  and  $\tilde{\sigma}^2 = s^2$

**Procedure:**

- Define  $m$  (number of replicas of the first simulation)
- Generate  $m$  (pseudo) Pareto observations and compute  $\tilde{\mu} = \bar{x}$  and  $\tilde{\sigma}^2 = s^2$
- Let  $n$  be the smallest integer greater than or equal to  $\left(\frac{1.96}{0.01}\right)^2 \times \left(\frac{\tilde{\sigma}^2}{\tilde{\mu}^2}\right)$
- Generate an additional sample with  $n - m$  observations and recomputed  $\tilde{\mu}$  using all observations (from both samples).



### Using R and choosing $m = 10000$ :

```
> alpha=3; theta=1000; m=10000;
> u=runif(m); x1=theta*(((1-u)^(-1/alpha))-1)
> tau2_h=var(x1); miu_h=mean(x1)
> n_min=((1.96/0.01)^2)*(tau2_h/(miu_h^2))
> miu_h; tau2_h; n_min
[1] 502.2063
[1] 878943.7
[1] 133877.9
> m2=123878
> u=runif(m2); x2=theta*(((1-u)^(-1/alpha))-1)
> x=c(x1,x2)
> miu_h=mean(x)
> tau2_h=var(x1); n_min=((1.96/0.01)^2)*(tau2_h/(miu_h^2))
> miu_h; tau2_h; n_min
[1] 498.7444
[1] 737651.6
[1] 113921.9
```



- **Distribution Function** ( $F_X(1000)$ ): Let  $p = F_X(1000)$ . The usual estimator is the ecdf

$$T = F_n(1000) = \frac{\#\{X_i \leq 1000\}}{n} \text{ and we get } \frac{T - p}{\sqrt{p \times (1-p)} / \sqrt{n}} \overset{\circ}{\sim} n(0;1)$$

$$\Pr(|T - p| \leq 0.01 p) = 2\Phi\left(\frac{0.01 p}{\sqrt{p \times (1-p)} / \sqrt{n}}\right) - 1, \text{ then}$$

$$\Pr(|T - p| \leq 0.01 p) \geq 0.95 \Leftrightarrow n \geq \left(\frac{1.96}{0.01}\right)^2 \times \left(\frac{1-p}{p}\right)$$

As  $p$  is unknown we run a first simulation ( $m$  replicas) to estimate it:  $\tilde{p} = \#\{x_i \leq 1000\} / m$

**Procedure:**

- Define  $m$  (number of replicas of the first simulation)
- Generate  $m$  (pseudo) Pareto observations and compute  $\tilde{p} = t = \#\{x_i \leq 1000\} / m$
- Let  $n$  be the smallest integer greater than or equal to  $\left(\frac{1.96}{0.01}\right)^2 \times \left(\frac{1-\tilde{\theta}}{\tilde{\theta}}\right)$
- Generate an additional sample with  $n - m$  observations and recomputed  $\tilde{p}$  using all observations (from both samples).



**Using R and choosing  $m = 10000$ :**

```
> alpha=3; theta=1000; m=10000;
> u=runif(m); x1=theta*(((1-u)^(-1/alpha))-1)
> t=mean(x1<=1000)
> n_min=((1.96/0.01)^2)*(1-t)/t
> t; n_min
[1] 0.881
[1] 5188.994
```

As  $10000 > 5189$  we stop.



#### 4. How to use simulation?

- Build a model for the random variable for which we want to approximate the distribution,  $S$ . This variable can be a function of other random variables.
- Define the number of replicas to be used,  $NR$
- For each replica generate as many pseudo-random variables as we need and compute a value for  $S$  using the model from step 1. Let us call this value  $s_j$ .
- The cdf of  $S$  is then approximated by the ecdf based on  $s_1, s_2, \dots, s_{NR}$ . Compute quantities of interest such as the mean, variance, percentiles, probabilities, ... from the ecdf.



### 3.1 – Approximating the sampling distribution of a statistic

**Example A** – Consider a normal population with mean 10 and standard deviation 3 from which we observe a sample of size  $n=5$ . Using simulation, obtain the sampling distribution of  $\bar{X}$  and compare with the theoretical result.

Procedure:

- Choose the number of replicas,  $NR$
- For each replica  $i, i = 1, 2, \dots, NR$ 
  - Generate 5 random (pseudo) numbers and obtain 5 r.v. with a  $n(10; \sigma = 3)$  distribution
  - Compute the sample average and keep this value as element  $i$  of the vector  $res$
- Perform a test to check if the values in vector  $res$  can be considered as observations of a normal with mean 10 and standard deviation  $3/\sqrt{5}$ .





Solution using R

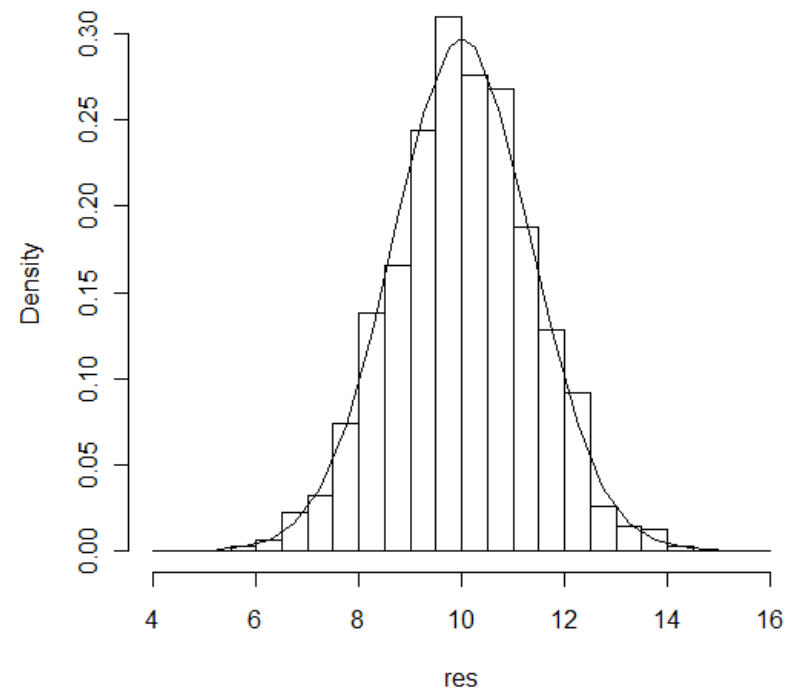
```
> NR=1000; n=5; miu=10; sig=3
> res=rep(NA,NR)
> for(i in 1:NR){
+   x=rnorm(n,miu,sig); res[i]=mean(x)
+ }
> mean(res); sd(res) # can also compute skewness and kurtosis
[1] 10.01766
[1] 1.351131
> ks.test(x,"pnorm",10,3/sqrt(5))
```

One-sample Kolmogorov-Smirnov test

```
data: x
D = 0.4427, p-value = 0.2087
alternative hypothesis: two-sided
> breaks=seq(4,16,0.5)
> points=c(seq(5+0.5/2,16-0.5/2,0.5),10); points=sort(points)
> dens=dnorm(points,miu,sig/sqrt(n))
> hist(res,breaks,prob=TRUE)
> lines(points,dens,type="l")
```



Histogram of res





**Example B** – Consider a Pareto population ( $\alpha = 1.5$ ,  $\theta = 100$ ) from which we observe a sample of size 10.

(i) Explain how to use simulation to get an approximation to the sampling distribution of  $\bar{X}$ . (ii) Perform the simulation with 1000 replicas and compare its results with the normal distribution. (iii) **Do you think that increasing the sample size will help?**

(i)

Determine  $NR$ , the number of replicas to be used.

For each of the  $NR$  replicas,  $i = 1, 2, \dots, NR$

- Generate 10 pseudo Pareto distributed variables – we generate 10 uniforms(0,1),  $u_j$ ,  $j = 1, 2, \dots, 10$  and using the inverse method we get 10 Paretos,  $x_j = \theta \left( (1 - u_j)^{-1/\alpha} - 1 \right)$
- Calculate the sample mean,  $\bar{x}_i = \sum_{j=1}^{10} x_j$

Now we compare the simulated distribution of  $\bar{X}$  using our  $NR$  pseudo observations  $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_{NR})$  with a normal distribution and conclude (descriptive statistics, qqplot, ecdf, ...)



(ii)

```
> NR=1000; n=10; alpha=1.5; theta=100
> res=rep(NA, NR)
> for(i in 1:NR){
+   u=runif(n); x=theta*((1-u)^(-1/alpha)-1); res[i]=mean(x)
+ }
> library(moments)
> cbind(mean(res), median(res), sd(res), skewness(res), kurtosis(res))
      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 195.7378 126.9872 339.5334 13.12755 248.533
> qqnorm(res) # result on next slide
```

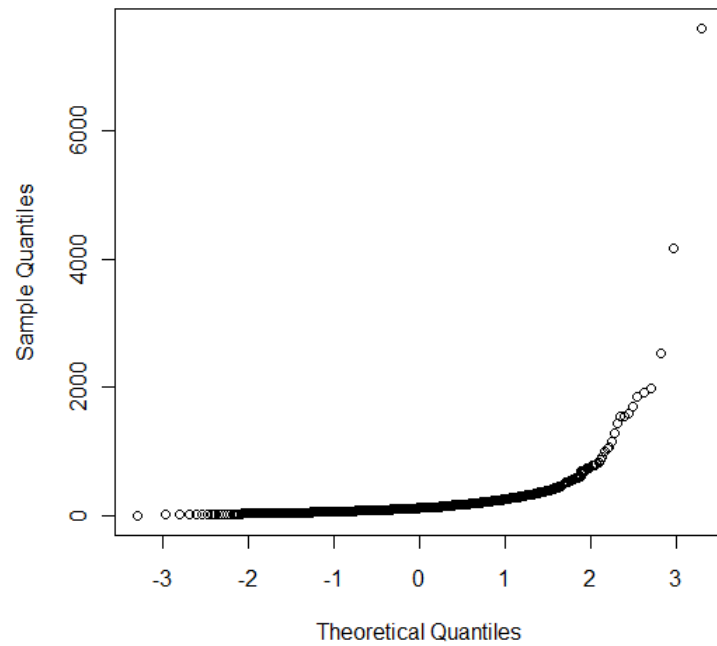
(iii) As  $\alpha = 1.5 < 2$  the variance of the Pareto distribution does not exist and consequently the CLT does not apply. The sampling distribution of  $\bar{X}$  does not converge to a normal distribution. Using the same R program with  $n = 1000$  we get

```
> cbind(mean(res), median(res), sd(res), skewness(res), kurtosis(res))
      [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 197.5064 186.5814 115.4582 26.57593 789.149
> > qqnorm(res) # result on next slide
```



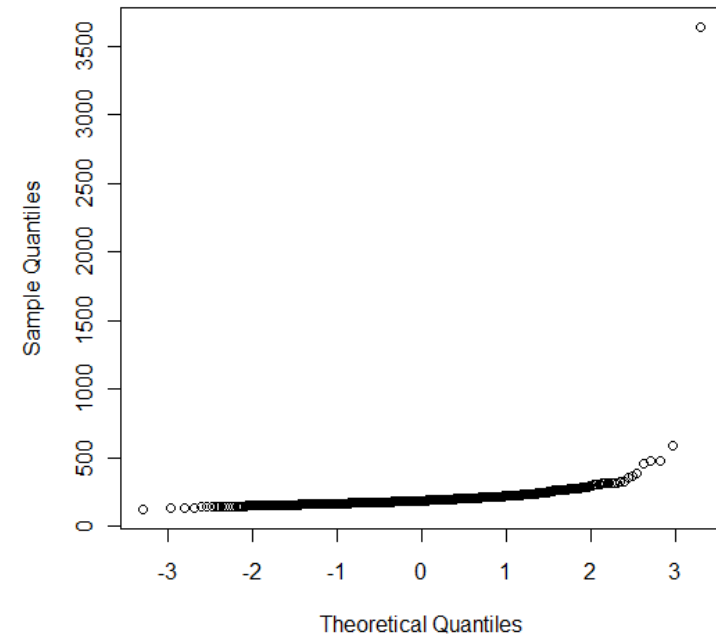
Sample size=10

Normal Q-Q Plot



Sample size=1000

Normal Q-Q Plot





**Example C** – Using simulation determine the p-value of the Kolmogorov-Smirnov test to test if the sample (11.79,11.25,6.83,10.47,13.60,10.60,17.40,10.99,16.45,12.47,8.19,13.46,13.82,11.93,7.47,10.09,14.26, 12.04,12.13,9.66) came from a normal distribution with mean 10 and standard deviation 3.

Procedure

- Compute the value of the test statistic using the observed sample,  $D$
- Determine  $NR$ , the number of replicas to be used
- For each of the  $NR$  replicas,  $i = 1, 2, \dots, NR$ 
  - Generate 20 pseudo normal (mean 10 and standard deviation 3) variables, i.e. generate 20 uniforms(0,1),  $u_j$ ,  $j = 1, 2, \dots, 20$ , then, using the Box-Muller method, get 20  $n(0,1)$ ,  $z_j$ , and finally get  $x_j = 10 + 3 \times z_j$ ,  $j = 1, 2, \dots, 20$ .
  - Using the generated sample, compute the value of the K-S test statistic,  $KS_i$
- The estimated p-value is given by the proportion of values of  $KS$  greater than  $D$



Using R and 10000 replicas we get

```
> x=c(11.79,11.25,6.83,10.47,13.60,10.60,17.40,10.99,16.45,12.47,8.19,
+     13.46,13.82,11.93,7.47,10.09,14.26,12.04,12.13,9.66)
>
> # test if x follows a normal distribution with mean 10 and stdev=3
> a=ks.test(x,"pnorm",10,3)
> D=a$statistic
> a
      One-sample Kolmogorov-Smirnov test
data:  x
D = 0.3122, p-value = 0.03132
alternative hypothesis: two-sided
>
> NR=10000; n=length(x)
> res=rep(NA,NR)
> for(i in 1:NR){
+   y=rnorm(20,10,3); a=ks.test(y,"pnorm",10,3)
+   res[i]=a$statistic
+ }
> p.value=mean(res>=D); p.value
[1] 0.0326
```



### 3.1 – More complex analysis

**Example 20.15 (simplified)** – Suppose that we want to approximate the distribution (namely its expected value and variance) of  $S$ , the present value of all payments made in respect of a policy issued today and covering loss events that occur in the next year,  $S = \sum_{j=1}^N X_j V_{T_j}$  where

- $N$  number of loss events during the next year. Losses are ordered according to time of occurrence
- $X_j$  amount of the  $j$ th loss. Assume that  $X \sim \text{Pareto}(\alpha = 3, \theta = 1000)$
- $V_t$  discount rate to be applied to time  $t$ . Assume that  $V_1 = 0.06$  (continuous time) – **simplified**
- $T_j$  time of payment of the  $j$ th loss. Let  $T_j = C_j + L_j$  where  $C_j$  is the time of the event and  $L_j$  is the time from occurrence to payment. Assume that they are independent.
- Assume that  $C_j - C_{j-1}$  (the times between events) are i.i.d. random variables following an exponential distribution with mean 0.2 (year). Let  $C_0 = 0$ . Also assume that  $C_j$  is independent of  $X_j$ .
- Assume that  $L_j$  are independent of each other and that  $L_j | X_j \sim \text{Weibull}(\tau = 1.5, \theta = (1/6) \times \ln X_j)$  if  $X_j > 1$  and  $L_j | X_j = 0$  if  $X_j \leq 1$ . **Corrected**





### Simulation procedure:

- Define  $NR$ , number of replicas
- For  $k = 1, 2, \dots, NR$  - For each replica we will generate a value for  $S$ ,  $s_k$ 
  1. Generate a value for  $C_j$  and, consequently, the number of claims,  $n$ 
    - a. Generate  $u$  (pseudo-random number) and get  $v = -0.2 \times \ln(1 - u)$
    - b. If  $v \geq 1$  then  $s_k = 0$  and begin a new replica. If not, define  $j = 1$ ,  $c_j = c_1 = v$
    - c. Generate a new value  $u$  and obtain  $v = -0.2 \times \ln(1 - u)$
    - d. If  $(v + c_j) < 1$ , compute  $c_{j+1} = c_j + v$ ,  $j \leftarrow j + 1$ , and go back to step c. If not go to step e.
    - e. Let  $n = j$  and keep array  $c$  (length  $n$ )
  2. Generate values for  $X_j$  and  $L_j$  and compute the value of  $T_j$  ( $j = 1, 2, \dots, n$ ).
    - a. Generate  $x_j$ , i.e. generate  $u$  and compute  $x_j = 1000 \times \left( (1 - u)^{-1/3} - 1 \right)$
    - b. Generate  $l_j$ , i.e. generate  $u$  and compute  $l_j = \max \left( \frac{1}{6} \ln(x_j) \times (-\ln(1 - u))^{2/3}; 0 \right)$
    - c. Compute  $t_j = c_j + l_j$
  3. Compute  $s_k = \sum_{j=1}^n x_j \times e^{-0.06 \times t_j}$  and begin a new iteration.
- Using the  $NR$  generated values for  $S$ , conclude.



## R program

```
NR=10000; res=rep(NA, NR); n.claims.iter=rep(NA, NR)
for(i in 1:NR){
  u=runif(50); v=-0.2*log(1-u) # over-sized to avoid a while loop
  c=cumsum(v); n=sum(c<1); c=c[1:n] # hair-cut of the over-sized array
  n.claims.iter[i]=n # to check that n<50
  if (n==0) {res[i]=0} else {
    u=runif(n); x=1000*((1-u)^(-1/3)-1) # claims amount
    u=runif(n); l=pmax((1/6)*log(x)*((-log(1-u))^(2/3)), 0) # Lj
    t=c+l; # Tj
    pay =x*exp(-0.06*t) # discounted values for each claim
    res[i]=sum(pay)
  }
}

mean(res); sd(res); var(res); hist(res)
table(n.claims.iter)
```



**Challenging Question:** Replace the constant rate of discount by the following hypothesis:

Assume that, for  $t > s$ ,  $\frac{\ln(V_s / V_t)}{(t - s)}$  has a normal distribution with mean 0.06 and variance  $0.0004(t - s)$ .



Solution:

Between steps 2 and 3 introduce

- As we must generate the values of the discount rate in order of increasing  $t_j$ , sort vector  $\mathbf{t}$  (ascending order) and vector  $\mathbf{x}$  (according to the values of  $\mathbf{t}$ ). Let  $\mathbf{t}^*$  and  $\mathbf{x}^*$  be the sorted vectors.
- Compute  $\Delta t_j = t_j - t_{j-1}$  where  $t_0 = 0$ .  $\Delta t_j$  represents the time between payment  $j-1$  and payment  $j$
- For  $j = 1, 2, \dots, n$

- Generate a value for  $\frac{\ln(V_{t_{j-1}} / V_{t_j})}{(t_j - t_{j-1})}$ , i.e. a normal random variable with mean 0.06 and variance

$0.0004\Delta t_j$ . We can generate a pseudo random value  $u$  and use the inverse transform or we can use the Box-Muller transforms. Let  $y$  be the generated value.

- Using  $\frac{\ln(V_{t_{j-1}} / V_{t_j})}{(t_j - t_{j-1})} = y$ , obtain  $V_{t_j} = V_{t_{j-1}} \times e^{-y_j(t_j - t_{j-1})}$ . Note that  $t_0 = 0$  and  $V_0 = 1$ .

- Now we can return to step 3.



```
NR=30000; res=rep(NA, NR); n.claims.iter=rep(NA, NR)
for(i in 1:NR){
  u=runif(50); v=-0.2*log(1-u) # over-sized to avoid a while loop
  c=cumsum(v); n=sum(c<1); c=c[1:n] # hair-cut of the over-sized array
  n.claims.iter[i]=n # to check that n<50
  if (n==0) {res[i]=0} else {
    u=runif(n); x=1000*((1-u)^(-1/3)-1) # claims amount
    u=runif(n); l=pmax((1/6)*log(x)*((-log(1-u))^(2/3)), 0) # Lj
    t=c+l; # Tj
    t.ord=t[order(t)]; x.ord=x[order(t)]
    delta.t.ord=c(t.ord[1], diff(t.ord)) # computing the differences
    u.norm=rnorm(n, 0.06, sqrt(0.0004*delta.t.ord)) # generating normals
    aux1=delta.t.ord*u.norm; aux2=cumsum(aux1)
    V=exp(-aux2)
    pay =x.ord*V # discounted values for each claim
    res[i]=sum(pay)
  }
}
mean(res); sd(res); var(res); hist(res)
table(n.claims.iter);
```