

# Simulation techniques in Statistics with R

Marcello Chiodi

`marcello.chiodi@unipa.it` <http://dssm.unipa.it/chiodi>

Dipartimento di Scienze Economiche Aziendali e Statistiche (SEAS)  
Università di Palermo

Stuttgart, February 2019



## 1 Introduction

- Why Monte Carlo?
- Monte Carlo integration
- Monte Carlo overcomes the curse of dimensionality
- Monte Carlo Integration and simulated sampling distributions
- Variance reduction techniques
  - Classical Monte Carlo integration
  - Importance sampling
  - Control variables
- Monte Carlo: strength and weakness

## 2 Generating pseudo random numbers

- Generating random numbers from uniform distributions
- Linear congruential generators
  - Repeating a sequence
- Notes on nonlinear congruential generators and other generators
- Testing for randomness

- 3 Pseudorandom number generation from univariate distributions
  - Inversion of the distribution function
  - Techniques based on transformations of random variables
  - Acceptance-rejection method
  - The ratio-of-uniforms method
  - Examples of generating pseudorandom observations for some known random variables
  
- 4 Generation of pseudorandom numbers vectors
  - Introduction
  - Generation of pseudorandom vectors from a multivariate normal distribution
  - Mixtures of multivariate normal distributions
  - Generation of contingency tables
  - The system of R for the management of probability distributions
  
- 5 Simulation Techniques in Statistics



- Overview
- Simulated sampling distributions
  - A simulation with  $m = 100000$
- Comment on the example
- Outline of algorithm of simulation of sampling distributions
- Simulation of regression models

## 6 Theory behind simulation studies

- Standard errors of simulation studies
- Simulated significance levels
- Power of a test
- Advantages and disadvantages of simulation techniques
- Disadvantages of simulation techniques
- Mixtures of distributions
  - Multivariate Distributions: examples

## 7 Other topics

## 8 References



- The simulation techniques are now widely used in statistics, also in methodological papers.
- **Monte Carlo methods**, although not elegant from a formal point of view, represent an effective and convenient way to obtain approximate results by simply using the brute force of a computer for one of its basic tasks:

## Monte Carlo Method and computer tasks

Repeating calculations many times!

- In many situations they are, at least currently, the only way, or at least the more convenient, to study the small-sample behavior of estimators or test whose exact distributions is unknown (or that we do not know how to derive)



- They are also used to evaluate empirically the goodness of an analytical asymptotic approximation *with respect to*  $n$ ;
- On the other hand, the impact of methods like Monte Carlo or other simulation techniques in statistics well consolidated: for example the development of methods for Monte Carlo Markov chains, such as the Gibbs sampler, etc..
- numerical integration techniques;
- optimization techniques (simulated annealing);
- simulation techniques provide a significant educational support for students to **show sampling distributions** or, rather, a sample extracted from a simulated sampling distribution, and to provide examples of complex analytical results.

example

```
exec Simul2000
```

## Some example

- Approximating a sampling distribution
- Approximating a significance level
- Approximating an integral
- Overcome the **Curse of dimensionality**
- We can reduce the variance of the results
- The method is *slow*
- It is an empirical approach: it is difficult to generalize results



# The birth of simulation techniques

*The idea was to experiment thousands of these possibilities and, at each stage, select at random, in other words, using a random number with a proper chance, you might investigate the outcome of certain types of events, so as to follow, so to speak, a linear succession, instead of considering all the ramifications. After examining the possible trends in only a few thousand cases, it would provide a good sample and an approximate answer to the problem. All you need to know is the average of the trends of new samples. Such a procedure was particularly suitable to be performed automatically and the birth of modern computers it was a consequence of that fact.*

(Ulam, 1976)





- Simulation techniques, as other fields in statistics, are intimately related to the automatic calculation and computers: in fact, the first implementations of the Monte Carlo method had to await the birth of an automatic machine, basically by Von Neumann:
- indeed, for some scientists, including Ulam and Von Neumann, just the ability to implement quickly the repetitive calculations necessary to conduct the simulations was an incentive for the actual construction of an automatic machine for calculation.
- The occasion was the multiplication molecular calculations necessary in Los Alamos laboratory during the Manhattan Project to build the first atomic bomb!



# Why “Monte Carlo”?

- Simulate (in the sense of mimicking more than in that of pretend) a process is to experience the same process under similar conditions a large number of times, each time by detecting the final state of the system. If the system state is expressed by numerical variables, an initial summary of the simulation is given, for example, the arithmetic average of the states and their variance.
- This technique (*not by chance!*) has been given the name of **Monte Carlo method** (Metropolis, Ulam, 1949), which originally was the code name given for reasons of secrecy
- The name was not so much associated with gambling, but rather to the fact that in a gambling house, for example at the roulette table, the same random experiment is repeated under similar conditions a large number of times.



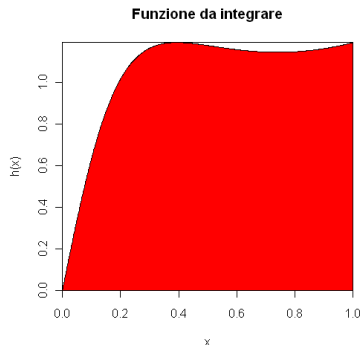
# Historical references: Student's t

- Gosset (alias Student) used an empirical method similar to Monte Carlo method to study the distribution of standardized averages by means of 750 samples of width 4, obtained from an empirical distribution of 3000 real measurements approximately normal.
- See Stigler (1991) and Piccolo(1998) for historical quotes on simulations in statistics.



# Simulations of deterministic models

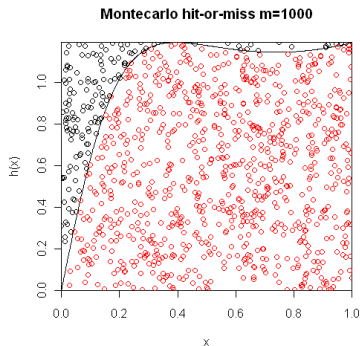
- It may seem a contradiction the application of simulation techniques to deterministic problems, but ...
- we want to calculate the value of a definite integral on a closed interval  $[a, b]$  of a function  $h(x)$  restricted to be always positive, which takes in  $[a, b]$  maximum  $H$ ;



suppose you obviously can not get the exact result, or you do not want to use approximations based on suitable quadrature formulas.

# Points at random: *hit-or-miss*

In the figure we apply an elementary simulation technique with  $m = 1000$ . We extracted  $m$  pairs of independent random numbers  $(X_i, Y_i)$  with  $X_i$  uniformly distributed in  $[a, b]$  and  $Y_i$  uniformly distributed in  $[0, H]$ ; ie we throw 1000 points in the rectangle that contains  $h(x)$ .



## Monte Carlo technique Hit or Miss

The estimate of the integral is given by the ratio between points in the curve and the total points thrown multiplied by the area of the rectangular area **basic technique**

# Monte Carlo hit-or-miss

- Let  $v$  the number of points that falls under  $h(x)$ , i.e. where one has:  $Y_i < h(X_i)$ ; the estimate of the integral is given by:
- $\hat{I} = \frac{H(b-a)v}{m}$
- Let  $\hat{p} = v/m$ , a confidence interval to 95 % for  $I$  is given by:

$$[H(b-a)] \left( \hat{p} \mp 1.96 \sqrt{\frac{\hat{p}(1-\hat{p})}{m}} \right)$$

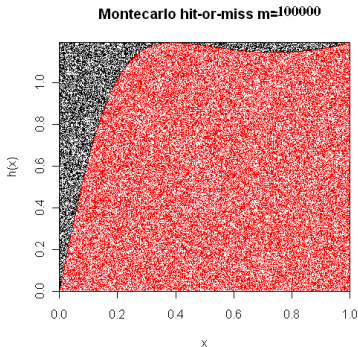
example

codiceIntegr1.R



# Increase the precision

- Now apply again the same elementary simulation technique but throwing 100,000 points in the rectangle that contains  $h(x)$ .
- We obtain a confidence interval on average 10 times shorter  $\left(\sqrt{\frac{m_1}{m_2}}\right)$

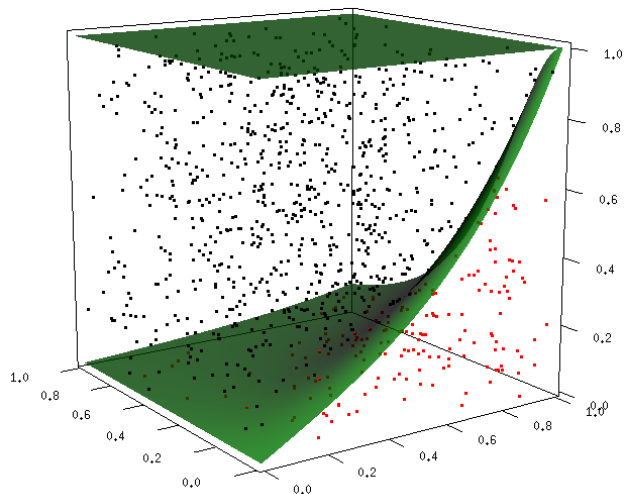


# R code used for integration hit-or-miss

```
1 # numerical integration
2 xvec      =      seq(0,1,by=0.001)
3 h         =      function(x) log(1+exp(x^2))*x/(.1+x^2)
4 polx      =c(0,xvec,1)
5 poly      =c(0,h(xvec),0)
6 plot(polx,poly,col="black",type="l",xlim=c(0,1),main="Function to
   integrate",xaxs="i",yaxs="i",xlab="x",ylab="h(x)")
7 polygon(polx,poly,col="red",xlim=c(0,1),xaxs="i")
8 H =max(poly)
9 m =1000
10 x =runif(m)
11 y =runif(m, min=0,max=H)
12 plot(polx,poly,col="black",type="l",xlim=c(0,1),xaxs="i",yaxs="i",
   main="Montecarlo hit-or-miss m=1000",xlab="x",ylab="h(x)")
13 points(x,y,col=(y<h(x))*1+1)
14 x11()
15 m=100000
16 x=runif(m)
17 y=runif(m, min=0,max=H)
18 plot(polx,poly,col="black",type="l",xlim=c(0,1),xaxs="i",yaxs="i",
   main="Montecarlo hit-or-miss m=100,000",xlab="x",ylab="h(x)")
19 points(x,y,col=(y<h(x))*1+1,pch=".")
```



# Integration of a 2d function



i

ntegrazione3d.mpeg

- Think again to the example described above: if we adopt a Monte Carlo type technique for multiple integrals, the standard error of the estimate is always the same, and **depends only on the number of points used!**

## Monte Carlo error estimate

With techniques like Monte Carlo, based on  $m$  points randomly chosen, the standard error of the estimate is generally of order  $\frac{1}{\sqrt{m}}$ , regardless of the number of dimensions of the function to integrate. This fact overcome the *curse of dimensionality*.



# Integration and Monte Carlo sampling distributions

- If we reflect carefully, we can see many similarities with the approximation to a sampling distribution of an estimator.
- In fact, the mathematical expectation of an estimator  $T = T(x_1, \dots, x_n)$  (where  $X$  has density distribution  $f(\cdot)$ ) is given by:

$$E(T) = \int \dots \int_{\mathfrak{R}_n} t(x_1, \dots, x_n) f(x_1, \dots, x_n) dx_1, \dots, dx_n$$

- so that the problem reduces to that of the calculation, by simulation, of an integral.



- Although the standard error of estimate is generally of the order of  $\frac{1}{\sqrt{m}}$ , the hit-or-miss technique is not very efficient, but we will not explore this aspect now.
- Remind only that if we want to estimate an integral

$$I = \int_a^b h(x) dx$$

a better estimate is given by:

$$\hat{I} = (b - a) \frac{\sum_{i=1}^m h(X_i)}{m}$$

where the  $X_i$  are uniformly distributed on  $[a, b]$



# Importance sampling

- Rewrite the integral in this way, putting  $g(x) = \frac{h(x)}{f(x)}$

$$I = \int_a^b h(x)dx = \int_a^b g(x)f(x)dx = E_f(g(X)) = E_f \left[ \frac{h(X)}{f(X)} \right]$$

- Now  $I$  has been expressed as the expected value of  $\frac{h(X)}{f(X)}$  with  $X$  a r.v. with density  $f(x)$ !!!
- We will estimate the integral drawing a sample of size  $m$  of independent random numbers  $X_i$  (from the distribution of density  $f(x)$ ), with sample mean:

$$\hat{I}_m = \frac{\sum_{i=1}^m \frac{h(X_i)}{f(X_i)}}{m} = \frac{\sum_{i=1}^m g(X_i)}{m} \quad \text{since} \quad \hat{I}_m \rightarrow I$$

- Since the variance of the estimate depends on the variance of  $g(X)$  we will have good results if  $f(x)$  has a shape similar to that of  $h(x)$

- A very simple and standard technique of variance reduction is the control variables approach.
- Suppose we want to estimate an integral which is the expected value of some variable  $Y$ , whose distribution is unknown, but for which we can obtain pseudo random values  $Y_i$ .
- For example if  $Y$  is a function of an  $n$ -dimensional random vector  $X$  ( $Y = g(\mathbf{x})$ ), and possibly  $Y$  is an estimator of some parameter  $\theta$ .

$$E[Y] \text{ is usually estimated by: } M(Y) = \frac{1}{m} \sum_{i=1}^m Y_i$$



- If for each  $Y_i$  we can compute  $Z_i$ ; determination of a variable  $Z$ , of known expected value  $E[Z]$ , and correlated with  $Y$ , we can estimate  $E[Y]$  with a regression type estimator, with a reduction of variance:

$$\hat{E}(Y) = M(Y) + b[M(Z) - E[Z]] \quad b = \frac{\text{Cov}(Y, Z)}{V[Z]}$$

The reduction of variance is given by  $r_{YZ}^2$

- For example  $Y$  and  $Z$  could be two estimator of the same parameter: we know the first moments of  $Z$  but not of  $Y$ , and reasonably the two estimators are correlated.



# Monte Carlo: strength and weakness

From the previous simple examples we can see the main strength and the main weakness of Monte Carlo method:

## Monte Carlo strength

- The (expected) error of the estimate depends on the number of generated samples,  $m$  (generally it is of order  $\frac{1}{\sqrt{m}}$ )
- The (expected) error does not depend on the number of dimensions of the function to integrate (or on the sample size on which we compute  $T_n$ ).

## Monte Carlo weakness

- The (expected) error of the estimate decreases, as the the number of generated samples  $m$  increases, **very slowly** generally at a rate of  $\frac{1}{\sqrt{m}}$
- For example, if we want to gain one more decimal digit of precision, we should multiply  $m$  by 100!



# Generate uniform pseudo-random numbers

## Pseudo random numbers

From above examples, it is obvious that we must have a quick method to produce a large amount of random numbers, possibly through a computer and not through a real tossing of coins....

## Pseudo random numbers

As we will see later, the basic problem is to get sequences of random numbers from a uniform distribution, because the generation of random numbers from any other distribution depends on it



## Pseudo random numbers

*Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.*

(Von Neumann, 1951, cit. by Knuth, 1981, p.1)

- The random numbers generated by special algorithms are called pseudo-random numbers: it is not conceivable for us to generate numbers **really random** by an automatic algorithm.
- Indeed, the idea of being able to use the algorithms (i.e. an automatic processes with fixed rules) to simulate random mechanisms seems at first sight a contradiction.



# Generate pseudo-random numbers

- This is possible because the aim of the generation of pseudo-random numbers is to obtain, by means of algorithms, sequences of numbers that are certainly not physically random, because obtained by analytical procedures, but that **resemble in many ways to really random sequences of numbers**;
- for example, their empirical distribution has characteristics that do not differ significantly from those of the corresponding theoretical distribution.
- Technically, we use algorithms that produce *chaotic sequences*, i.e. appropriate sequences that behave similarly to real random sequences.



# Linear congruential generators

The techniques for the generation of pseudo-random numbers from a uniform distribution are based on specific algorithms, such as the linear congruential algorithm, rather than a mechanical or physical device, hard to implement on a computer.

$$a_i = (\lambda a_{i-1} + b) \bmod c$$

$$U_i = a_i / c$$

with  $\lambda$ ,  $b$ ,  $c$ ,  $a_0$  integer numbers.

We have:

$\lambda$	(multiplier)	$0 < \lambda < c$
$b$	(increment)	$0 \leq b < c$
$c$	(modulus)	$c > 0$
$a_0$	(seed or start value)	$0 \leq a_0 < c$



# Examples of linear congruential generators

$i$	$a_i$	$U_i$	$i$	$a_i$	$U_i$
0	1	0,0625	9	4	0,2500
1	12	0,7500	10	11	0,6875
2	3	0,1875	11	14	0,8750
3	6	0,3750	12	13	0,8125
4	5	0,3125	13	8	0,5000
5	0	0,0000	14	15	0,9375
6	7	0,4375	15	2	0,1250
7	10	0,6250	16	1	0,0625
8	9	0,5625	17	12	0,7500

Sequence of numbers obtained with  $\lambda = 5$ ,  $b = 7$ ,  $c = 16$ ,  $a_0 = 1$   
(sequence with maximum length cycle)



# Examples of linear congruential generators

$i$	$a_i$	$U_i$	$i$	$a_i$	$U_i$
0	1	0,0625	5	11	0,6875
1	11	0,6875	6	9	0,5625
2	9	0,5625	7	3	0,1875
3	3	0,1875	8	1	0,0625
4	1	0,0625	9	11	0,6875

Sequence of numbers obtained with  $\lambda = 11$ ,  $b = 0$ ,  $c = 16$ ,  $a_0 = 1$   
(sequence with length cycle 4)



# Examples of linear congruential generators

$i$	$a_i$	$U_i$	$i$	$a_i$	$U_i$
0	1	0,0010	4	835	0,8154
1	27	0,0264	5	323	0,3154
2	547	0,5342	6	323	0,3154
3	707	0,6904	7	323	0,3154

Sequence of numbers obtained with  $\lambda = 20$ ,  $b = 7$ ,  $c = 1024$ ,  $a_0 = 1$   
after  $a_5$  all the numbers are equal to 323



# Examples of linear congruential generators

$i$	$a_i$	$U_i$	$i$	$a_i$	$U_i$
0	1	0,0000	8	25673	0,7835
1	942	0,0287	9	4918	0,1501
2	6755	0,2061	10	5931	0,1810
3	15712	0,4795	11	14696	0,4485
4	22501	0,6867	12	13613	0,4154
5	5938	0,1812	13	1978	0,0604
6	14983	0,4572	14	16463	0,5024
7	25380	0,7745	15	20524	0,6263

Sequence of numbers obtained with  $\lambda = 41$ ,  $b = 901$ ,  $c = 32768$ ,  $a_0 = 1$   
(sequence with maximum length cycle)





# Properties of linear congruential generators

If  $b = 0$  and  $c$  is a prime number we have a sequence with maximum length cycle  $c - 1$  if and only if  $\lambda$  is a **generator modulus  $c$** , i.e. if:  $c - 1$  is the smallest value of  $n$  satisfying  $\lambda^n \bmod c = 1$

The following theorem is computationally more useful:

if  $c$  is a prime number, then  $\lambda$  is a generator modulus  $c$  if and only if

- 1  $\lambda$  and  $c$  are relatively prime numbers;
- 2  $\lambda^{(c-1)/p} \bmod c \neq 1$  for each prime factor  $p$  of  $c - 1$ .



# Properties of linear congruential generators

Sufficient conditions for 2 is a generator modulus  $c$  (prime number) are:

- 1  $\frac{c-1}{2}$  is prime;
- 2  $c \bmod 8 = 3$ .

Starting from  $g$ , a generator modulus  $c$ , we can find other generators by relationship

$$\lambda \bmod c = g^a$$

with  $a$  and  $c - 1$  primes among them.



# Generation of random numbers from uniform distribution

- In general, algorithms provide integer values  $a_i$ , with  $a_i < n$ , on the basis of  $k$  previous values of  $a_i$ , according to the generic recursive scheme:

$$a_i = g(a_{i-1}, a_{i-2} \dots, a_{i-k}).$$

- The type of sequence is determined by the function  $g(\cdot)$ , which can be linear or nonlinear.
- The elements of the sequence are then standardized:  $U_i = \frac{a_i}{n}$



# Choice of $\lambda$ , $b$ , $c$ and $a_0$

We search for a sequence that:

- has a long cycle;
- is founded on a high modulus  $c$ ;
- is not too sensitive to the choice of  $a_0$

A sequence has maximum length  $c$  if and only if,  $\lambda$ ,  $b > 0$  and  $c$  meet the following conditions:

- 1  $b$  and  $c$  have no common divisors besides the unit;
- 2  $\lambda$  is a multiple of each prime factor of  $c$ ;
- 3  $\lambda$  is a multiple of 4 if  $c$  is a multiple of 4.



# Some of the values of $\lambda$ and $c$ more used in literature ( $b = 0$ )

$\lambda$	$c$
$7^5 = 16.807$	$2^{31} - 1 = 2.147.483.647$
742.938.285	$2^{31}-1$
950.706.376	$2^{31}-1$
1.343.714.438	$2^{31}-1$
1.226.874.159	$2^{31}-1$
62.089.911	$2^{31}-1$
397.204.094	$2^{31}-1$
$13^{13} = 302.875.106.592.253$	$2^{59}-1$
8192	67.101.323
8192	67.099.547
32768	16.775.723



# Other properties of linear congruential generators

- If  $b = 0$  then

$$a_{i+k} = \lambda^k a_i \bmod c$$

- If  $b = 0$ , then the autocorrelation  $\rho_1(U)$  of the entire sequence of length  $n$  is included between the range

$$\rho_1(U) \in \left[ \frac{1}{\lambda} \pm \frac{\lambda}{c} \right]$$

with the autocorrelation of lag 1 defined by:

$$\rho_1(U) = \frac{1}{n-1} \sum_{i=1}^{n-1} [U_i - M(U)][U_{i+1} - M(U)] / \text{Var}(U)$$



# Other properties of linear congruential generators

- If  $U_1$  e  $U_2$  are two independent standard uniform r.v., then  $\text{frac}(U_1 + U_2)$  is a standard uniform r.v.;
- If  $U_1$  and  $U_2$  are pseudo-random numbers derived from congruential generators with full period and with modulus  $c_1$  e  $c_2$ , respectively, then  $U = \text{frac}(U_1 + U_2)$  has period  $c_1 \cdot c_2$  ( $c_1$  and  $c_2$  should be prime among them). Usually a combination of multiple generators is used in practice.
- Linear congruential generators have the following behavior (code `./routine/fig_cas_alg_congr.r` and `./routine/fig_cas_alg_congr2.r`).



# Key features required in a linear congruential generator

According to Ripley (1990), the ideal properties of a good pseudo-random number generator are:

- 1 a very good approximation to a uniform distribution;
- 2 very close to independent output in a moderate number of dimensions;
- 3 a very long period;
- 4 repeatability from a simply specified starting point, but if the starting point is not specified the sequence should not be predictable.





# Main linear congruential generators recommended

Noting that in general no algorithm for generating pseudorandom numbers is the best one for all applications, according to Wichmann and Hill (2006) linear congruential generators to be used are:

Name	Period	Lines of code	Dimension in bytes	Relative times
ISAAC (Jenkins)	$\geq 2^{40}$	97	1024	1, 0
AES (NIST, 2001)	unknown	85	16	2, 1
Mersenne twister (Matsumoto and Nishimura, 1998)	$2^{19937} - 1$	48	2500	2, 3
MRG32k3a (L'Ecuyer, 1999)	$\approx 2^{191}$	31	48	2, 7
Knuth, TAOCP (Knuth, 1981)	$\approx 2^{129}$	90	404	4, 9
CLCG4 (L'Ecuyer and Andres, 1997)	$\approx 2^{121}$	34	16	9, 2
Wichmann e Hill - 4 cycles, 2006	$\approx 2^{120}$	26	16	10, 0
MRG63k3a (L'Ecuyer, 1999)	$\approx 2^{377}$	40	48	14, 3



# Linear congruential generators used in R

Pseudorandom number generators available in R are the following (the default generator is “Mersenne-Twister”):

- **Wichmann-Hill**: has a period of length  $6.9536e + 12$ .
- **Marsaglia-Multicarry**: has a period of length greater than  $2^{60}$  and has passed all tests, according to Marsaglia.
- **Super-Duper**: has a period of about  $4.6 * 10^{18}$ .
- **Mersenne-Twister**: has a period of length  $2^{19937} - 1$ .
- **Knuth-TAOCP-2002**: has a period of about  $2^{129}$ .
- **Knuth-TAOCP**: a previous version.
- **user-supplied**: uses a user-supplied generator.



# The system of random number generators of R

```
1 .Random.seed <- c(rng.kind, n1, n2, \dots)
2 RNGkind(kind = NULL, normal.kind = NULL)
3 RNGversion(vstr)
4 set.seed(seed, kind = NULL, normal.kind = NULL)
5 # or:
6 save.seed <- .Random.seed
7 set.seed(save.seed, kind = NULL, normal.kind = NULL)
```

## codiceSeed1.R

- The default kind is "Mersenne-Twister": (Matsumoto and Nishimura, 1998).
- It is a random number generator, currently one of the best, if not the best (also used in online games).
- It has a period  $2^{19937} - 1$  (Mersenne prime number)
- The initial seed is a set of 624 integers of 32 bits,



# Generating the same sequence

Since it uses a recursive formula, starting from the same sequence of  $k$  values, we obtain again the same sequence of **uniform numbers**

```
1 a<- .Random.seed
2 x1=runif(1000)
3 .Random.seed<-a
4 x2=runif(1000)
5 x1-x2
6 # or
7 a<- .Random.seed
8 set.seed(a)
9 x1=runif(1000)
10 set.seed(a)
11 x2=runif(1000)
12 x1-x2
13 #output
14 # [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      0 0 0 0 0 0
15 # ...
16 # [1000] 0
```

codiceSeed2.R

# Pseudo-random numbers

- E.g. an algorithm for generating random numbers integers between 1 and 6 is satisfactory if, given a sequence of  $m$  integers between 1 and 6, it is difficult or even impossible to determine if it has been generated by an algorithm or by  $m$  true independent tosses of a balanced dice

① 23433661213245126156216324422541513424532144346651

② 54443162343665452263452416444336546456133221556333

③ 61525352442212136156341544324365235163633555155221

④ 65522622244141624521435433263412614656553352425261

⑤ 15263542534251425366635362433222132326326464365432

⑥ 45354256422136366662335112233324453464613112324342

(1) and (3) from a computer;(2) and (6) throwing real dices; (4) transforming the (3): 7-(3) from right to left (5) pounding on the keys



## A good algorithm

What matters is whether we, operationally, would know whether a certain result comes from an analytical procedure or by a true random experiment. If we can understand it, then that is not a good algorithm!

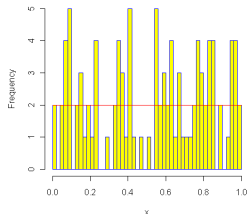
- Can we understand how this sequence was derived? was it derived from a true random experiment?
- 0,275 0,301 0,548 0,592 0,208 0,161 0,262 0,793 0,754 0,179 0,083  
0,591 0,619 0,574 0,287 0,000 0,083 0,051 0,167 0,190 0,853 0,128  
0,264 0,583 0,542 0,413 0,498 0,764 0,377 0,057 0,165 0,908 0,203  
0,018 0,642 0,042 0,627 0,484 0,931 0,619 0,304 0,411 0,586 0,908  
0,447 0,402 0,262 0,797 0,476 0,264



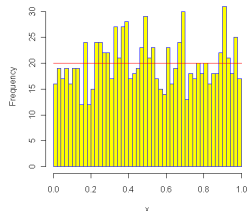
# Sequences of different lengths

Of course, longer simulated sequences will have a distribution closer to the theoretical one, with error of order  $\frac{1}{\sqrt{m}}$

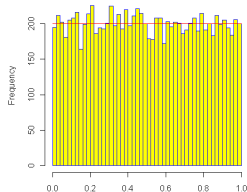
random numbers from a uniform n= 100



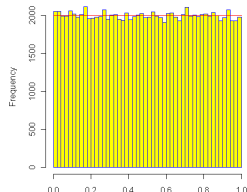
random numbers from a uniform n= 1000



random numbers from a uniform n= 10000



random numbers from a uniform n= 100000



```
1 univsimul<-function(nsamples=1000, nparz=nsamples, n=1,  
2   modello="normal",parm=c(0,1),  
3   nclass = 50,  
4   nscreen =c(1,1),  
5   newscreen =TRUE,  
6   iscreen =1,  
7   singlesample= FALSE,  
8   teoplot = TRUE,  
9   tpause = 0.01,  
10  colorhist = "yellow",  
11  colorborder = "blue",  
12  colorteo = "red",  
13  colorbg = "white",  
14  dimpoint = 0.8)
```

codiceSimul1.R





# Nonlinear congruential generators

Linear congruential generators have some undesirable features, as seen above, even if remain the most popular tool for generating pseudorandom numbers.

Some *nonlinear congruential generators* have been developed but with a greater computational cost.

It is to be observed that in relative terms diminishes the influence of the time taken for the calculation of  $U_i$  in relation to overall time required by a statistical simulation (Ripley, 1983).



# Inversive congruential generators

An important class of nonlinear generators introduced by Eichenauer-Hermann (1992) are based on the concept of *multiplicative inverse modulo c*.

Let  $Z_n$  be the set of the integers less than  $n$ , for any integer  $n$ . Let  $p \geq 5$  be a prime number and  $z$  an integer; we define a *multiplicative inverse of  $z$  modulo  $p$*  the unique element  $\bar{z}$  of  $Z_p$  such that:

$$z \cdot \bar{z} \bmod p \equiv 1$$

An inverse congruential sequence of elements of  $Z_p$  is defined as

$$a_i \equiv (\lambda \bar{a}_{i-1} + b) \bmod p, \quad U_i = a_i/p$$

with  $\lambda \neq 0$ ,  $b$ ,  $a_0$  integer numbers and  $p$  prime number.



# Feedback shift register (Tausworthe, 1965)

The foundation is given by the recursive formula:

$$a_i = \sum_{j=1}^k \lambda_j a_{i-j} \pmod{c}$$

with  $\lambda_j$  not all zero coefficients,  $a_i$  not all zero starting values,  $c$  usually a prime number (the period of the sequence could not exceed  $c^k - 1$ ).

In particular, if  $c = 2$ , we get a sequence of binary digits (0.1); combining then groups of  $s$  binary digits  $a_i$ , we get fractional parts of numbers  $U_i$  between 0 and 1 and represented in binary notation.



# Feedback shift register (Tausworthe, 1965)

The advantages of this technique are basically two:

- We do not need to standardize to have the interval  $[0,1)$ , because we get the binary digits to the right of the decimal that compose the pseudorandom number  $U_i$ ;
- We can use the same generator to get numbers  $U_i$  with the desired accuracy: simply by increasing the number  $s$  of binary digits.



# Other techniques for generating random digits

- Use of the digits of  $\pi$  (Dodge, 1996);
- Tables of random numbers;
- ...

## Drawbacks:

- Regarding the use of  $\pi$  is very costly in terms of computing time;
- We can think to store a large number of digits (a few billion) on a mass storage device, but is not a practical solution; same problem for tables of random numbers;
- About the tables of random numbers, it has the disadvantage that we could always use the same.



# Testing for randomness

To verify the goodness of pseudorandom numbers that are generated, we can check the following properties:

- Matching between empirical and theoretical moments: for example, for a standard uniform distribution we have

$$\mu = 0.5, \sigma^2 = 1/12 = 0.08\bar{3}, \beta_1 = 0, \beta_2 = 1.8$$

or use the following tests:

- Test  $\chi^2$  of goodness of fit;
- Kolmogorov-Smirnov test based on the test statistics

$$D_{max} = \max |F_i - i/m|$$



# Testing for randomness

- Test  $X^2$  on pairs  $(U_i, U_{i+1})$  to verify that they are uniformly distributed in a square with unitary surface;
- Generalization of the previous test  $X^2$  for  $n$ -tuples of pseudorandom numbers;
- Test  $X^2$  on pairs  $(U_i, U_{i+r})$  to verify that there are no undesirable autocorrelations for lag above the first (in this case the test statistic  $X^2$  should be amended accordingly).



# Testing for randomness

For uniform pseudorandom number generators, battery of statistical tests exists that are publicly available. The most important are

- **DIEHARD** (Marsaglia, 1985), that can be found at the URL <http://www.stat.fsu.edu/pub/diehard/>;
- **TestU01** (L'Ecuyer and Simard, 2005), that can be found at the URL <http://www.iro.umontreal.ca/~simardr/>.

The **TestU01** battery is more “stringent” of the **DIEHARD** battery.





# Inversion of the distribution function

Let  $X$  be a r.v. with distribution function  $F(\cdot)$ .

Let  $G(\cdot)$  be the inverse of the distribution function (this function always exists due to the monotonicity of  $F(\cdot)$ ), i.e.

$$G(F(x)) = x$$

Considering a generic event  $X \leq y$  with probability  $F(y)$ , this is equivalent to the event  $F(X) \leq F(y)$ , and then we have:

$$Prob\{X \leq y\} = Prob\{F(X) \leq F(y)\} = F(y)$$

so, putting  $U = F(X)$ ,  $U$  is uniformly distributed in  $[0, 1]$ .



# Inversion of the distribution function

Conversely, if we consider a standard uniform r.v.  $U$ , we have:

$$\text{Prob}\{U \leq F(x)\} = F(x)$$

so by applying the function  $G(\cdot)$  we have:

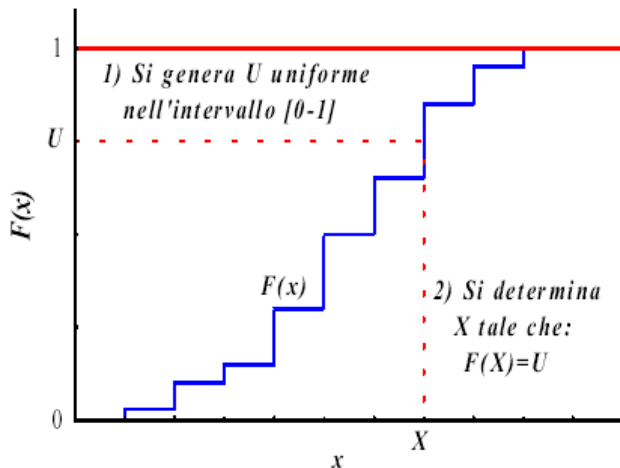
$$\text{Prob}\{G(U) \leq G(F(x))\} = \text{Prob}\{G(U) \leq x\} = F(x)$$

that is,  $G(U)$  is distributed as  $X$ .

Therefore, we can generate  $U$  from a standard uniform distribution and then calculate  $X = G(U)$  which will be a random number from a distribution with distribution function  $F(\cdot)$ .



# Use of inversion of the distribution function (discrete case)



# Generation from any distributions

- The method is applicable to any random variable, continuous or discrete.
- Obviously the answer is given by an equation usually not in explicit form (and which contains also an integral)

Given  $U$ , solve with respect to  $X$ :

$$\int_{-\infty}^X f(x) dx = U$$

- However, this method **is not adaptable to simulate the extraction of pseudo-random vectors from multivariate distributions**
- There are however several methods for generating random numbers from univariate and multivariate distributions



# Discrete distribution with finite number of values

$x_i$	$P(X = x_i) = p_i$	$F(x_i)$
0	0,1	0,1
1	0,7	0,8
2	0,2	1,0

To generate a pseudorandom determination of  $X$  we proceed as follows:

- 1 we generate a random number  $U$  from a standard uniform distribution.
- 2 if  $U \leq 0,1$  we put  $X = 0$  else
- 3 if  $U \leq 0,8$  we put  $X = 1$  else we put  $X = 2$



# Discrete distribution with finite number of values

$x_i$	$P(x_i)$	$H_i = \sum_{j=1}^i P(x_j)$
1	0,7	0,7
2	0,2	0,9
0	0,1	1,0

The above algorithm is modified as follows:

- 1 we generate a random number  $U$  from a standard uniform distribution.
- 2 if  $U \leq 0,7$  we put  $X = 1$  else
- 3 if  $U \leq 0,9$  we put  $X = 2$  else we put  $X = 0$



# Discrete distribution with finite number of values

Which of the two algorithms should be used?

Let  $C$  be the r.v. “number of comparisons to be made”. The expected value of this r.v. is given by:

$$E(C) = \sum c_i P(C = c_i)$$

Number of comparisons $c_i$	$P(C = c_i)$ non ordered values of $X$	$P(C = c_i)$ ordered values of $X$
1	0,1	0,7
2	0,9	0,3
<b>E(C)</b>	<b>1.9</b>	<b>1.3</b>



- Sorting the values is convenient only if we need to generate many pseudorandom numbers from the same distribution;
- We take time advantage for distributions with strong heterogeneity between probabilities.





# Generation from any discrete distributions

## Initialization

- steps from a) to c) are required only for discrete r.v. with an infinite number of values;
  - for discrete r.v. with a finite number  $k$  of values we can skip ahead to step d).
- a) set  $\varepsilon$ ;
  - b) determine the smallest value of  $k$  for which  $1 - F_k \leq \varepsilon$ ;
  - c) put  $F_{max} = F_k$ ;
  - d) compute  $p_i$  ( $i = 1, 2, \dots, k$ );
  - e) sort the  $k$  probability  $p_i$  in descending order and rearrange consequently the  $x_i$  by getting the two vectors  $x_{(i)}$  and  $p_{(i)}$ ;
  - f) set  $H_0 = 0$ ;
  - g) compute  $H_i = H_{i-1} + p_{(i)}$ , ( $i = 1, 2, \dots, k$ ) ( $H_k$  is equal to  $F_k$ ).

# Generation from any discrete distributions

*Body of the algorithm for generating a single number*

- h) generate  $U$  from a standard uniform distribution;
- i) if  $U < 1 - \varepsilon$  go on, otherwise go to step m);
- j) set  $i = 1$ ;
- k) if  $U < H_i$  set  $X = x_i$  and go to h) to compute another random number; otherwise go on;
- l) set  $i = i + 1$  and go to k);

*Tail of the distribution*

- m) set  $i = k + 1$  and  $C = F_k$ ;
- n) compute  $p_i$  and set  $C = i + p_i$ ;
- o) if  $U < C$  set  $X = x_i$  and go back to h) to get another random number; otherwise go on;
- p) set  $i = i + 1$  and go back to n).



# Generation from an exponential distribution

The exponential r.v. has

$$f(x) = \lambda e^{-\lambda x}; \quad F(x) = 1 - e^{-\lambda x}$$

with  $\lambda > 0$  and  $x \geq 0$ .

Then, we have

$$U = 1 - e^{-\lambda X} \Rightarrow e^{-\lambda X} = 1 - U \Rightarrow X = -\log(1 - U)/\lambda$$

Since  $1 - U$  is distributed as  $U$ , we can consider the final relation:

$$X = -\log(U)/\lambda$$

When  $\lambda = 1/2$  the exponential distribution is a  $\chi^2(2)$  distribution.



- Generate from a Bernoulli distribution
- Generate from a Geometric distribution ( $F(X) = 1 - (1 - p)^x$ )



# Techniques based on transformations of random variables

- Random number generation methods based on transformations of random variables are based on known theorems of probability to get the distribution function of random variables.
- If we know that  $X = g(Y)$  and we know how to generate random numbers from the distribution of  $Y$  to obtain a random number from the distribution of  $X$  just generate a number  $y$  and then compute  $x = g(y)$ .



# Box-Muller (1958) transformation: normal distribution

Suppose we want to generate a random point  $P(X_P, Y_P)$  from a standardized bivariate normal distribution.

We represent  $P$  polar coordinate system:  $P(\rho_P, \theta_P)$ ;  $\rho_P$  (radial coordinate) and  $\theta_P$  (angular coordinate) are independent.

By definition we have

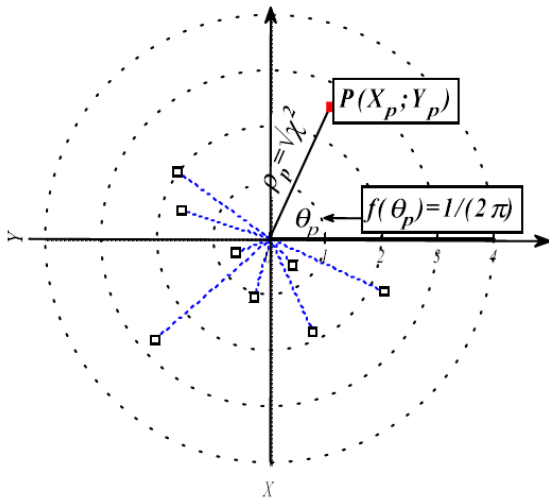
$$\rho_P^2 = X_P^2 + Y_P^2$$

and then  $\rho_P^2 \sim \chi^2(2)$ .

In addition, we have  $\theta_P \sim U(0, 2\pi)$ , since, for independent standardized normal variables, the density of points along a circle with center at the origin is constant.



# Box-Muller transformation: normal distribution



# Box-Muller transformation: normal distribution

We can then generate  $\rho_P$  and  $\theta_P$  through the following relations:

$$\rho_P = \sqrt{\chi^2(2)} = \sqrt{-2 \log(U)}$$

$$\theta_P = 2\pi V$$

To get two random numbers  $X_P$  e  $Y_P$  just use the following relations:

$$X_P = \rho_P \cos(\theta_P) = \sqrt{-2 \log(U)} \cos(2\pi V)$$

$$Y_P = \rho_P \sin(\theta_P) = \sqrt{-2 \log(U)} \sin(2\pi V)$$





# Poisson distribution

We can take advantage of the connection between Poisson distribution and the arrivals process regulated by an exponential distribution.

A Poisson distribution with parameter  $\lambda$ , with:

$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!}, \quad x \geq 0; \lambda > 0$$

gives the probability that per unit of time occur exactly  $x$  arrivals, in a Poissonian arrivals process with independent increments, with

$$f(t) = \lambda e^{-\lambda t}, \quad t \geq 0$$

the density function of the probability distribution of the waiting time  $t$  between successive arrivals.



# Poisson distribution

Therefore, instead of simulating the numbers of arrivals  $X$  per unit of time, we simulate independent arrival times  $T_i$  from an exponential distribution, until we pass, as total time, one.

The event  $\{X = x\}$  is equivalent to the event

$$\left\{ \sum_{i=1}^x T_i < 1 \leq \sum_{i=1}^{x+1} T_i \right\}$$

Then we generate the  $T_i$  from an exponential distribution with parameter  $\lambda$  until we have:

$$1 \leq \sum_{i=1}^{x+1} T_i$$

and then we take  $x$  as a random number generated by a Poisson distribution with parameter  $\lambda$ .



# Poisson distribution

For the implementation of the algorithm, we have

$$T_i = -\log(U_i)/\lambda$$

and then

$$1 \leq \sum_{i=1}^{x+1} T_i = -\sum_{i=1}^{x+1} \log(U_i)/\lambda$$

which yields

$$-\lambda \geq \sum_{i=1}^{x+1} \log(U_i) \Rightarrow -\lambda \geq \log\left(\prod_{i=1}^{x+1} U_i\right) \Rightarrow e^{-\lambda} \geq \prod_{i=1}^{x+1} U_i$$



# Algorithm for generating a Poisson number

## *Initialization*

- a) Compute  $L = e^{-\lambda}$ ;

## *Body of the algorithm*

- b) Set  $X = 0$  and  $W = 1$ ;
- c) Generate  $U$  from a standard uniform distribution;
- d) Set  $W = W \cdot U$ ;
- e) If  $L < W$ , set  $X = X + 1$  go back to step c), else  $X$  is the number sought and go back to step b) for a new generation.



# Acceptance-rejection method (Von Neumann, 1951)

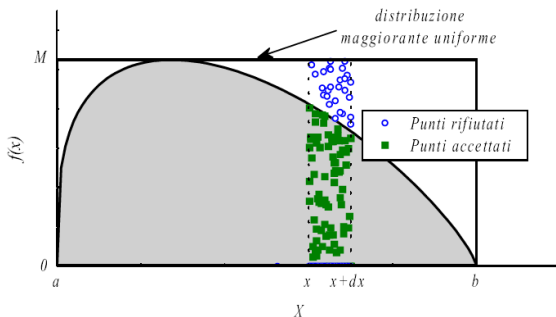
Suppose we want to generate pseudorandom numbers from the density distribution  $f(x)$  of a continuous random variable  $X$  defined in a limited range  $[a, b]$ , which takes on a maximum value of  $M$ .

If we get points uniformly distributed over the surface subtended by  $f(x)$ , we could generate abscissa  $X$  from the distribution with density function  $X$  and ordinate  $Y$  uniformly distributed in  $[0, f(X)]$ .

Conversely, given a point  $P(X, Y)$  uniformly distributed on the surface subtended by  $f(x)$ , its abscissa  $X$  follows the distribution with density function  $f(x)$ .



# Acceptance-rejection method (Von Neumann, 1951)



We may be able to get those points  $P(X, Y)$  by generating them uniformly in the rectangle bounded by two horizontal lines with ordinate 0 and  $M$ , and two vertical lines with abscissa  $a$  and  $b$ , conditionally to the fact that  $Y \leq f(x)$ , i.e. that  $P$  is below  $f(x)$ .

Simply generate  $X$  uniform in  $[a, b]$  and  $Y$  uniform in  $[0, M]$ . The  $x$ -coordinates  $X$  of the points that have  $y$ -coordinates  $Y \leq f(X)$  are random numbers that we want.



Factorization of  $f(x)$ :

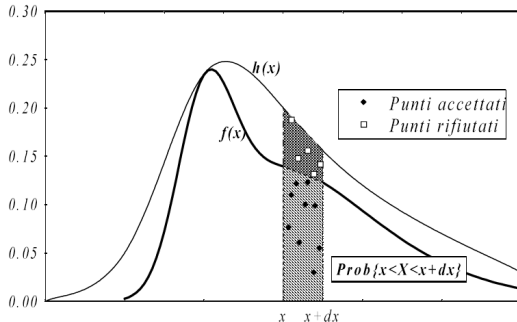
$$f(x) = C r(x) g(x)$$

with

- $r(x)$  density function from which we know how to generate pseudorandom numbers;
- $g(x)$  a function such that  $0 < g(x) \leq 1$ ;
- $C \geq 1$  normalizing constant.



# General case



Setting  $h(x) = C r(x)$ , we have  $g(x) = f(x)/h(x)$  and then  $h(x)$  is always greater than  $f(x)$ :  $h(x)$  is a dominating function of  $f(x)$ .

We have also:

$$C = \int_{-\infty}^{+\infty} f(x)/g(x) dx = \int_{-\infty}^{+\infty} h(x) dx$$

with  $1/C$  representing the method efficiency.



## *Initialization*

- a) determine a factorization:  $f(x) = Cr(x)g(x) = h(x)g(x)$ ;

## *Body of the algorithm*

- b) generate  $X$  from a distribution with density function  $r(x)$ ;
- c) generate  $V$ , independent from  $X$ , from a standard uniform distribution;
- d) if we have  $V \leq g(X)$  we accept  $X$ , else we reject  $X$  and we go back to step b).



We have to compute the probability that  $X$ , random variable with distribution with density function  $r(x)$ , takes on a value between  $x$  and  $x + dx$ , conditionally to  $V \leq g(x)$ :

$$\begin{aligned} & P \{x \leq X \leq x + dx | V \leq g(x)\} = \\ &= P \{[x \leq X \leq x + dx] \cap [V \leq g(x)]\} / P \{V \leq g(x)\} = \\ &\approx \frac{r(x)g(x)dx}{\int_a^b r(x)g(x)dx} = \frac{r(x)g(x)dx}{1/C} = f(x)dx \end{aligned}$$



# The squeeze principle

We have to find two functions  $b(x)$  and  $h(x)$  such that for every  $x$  we have:

$$b(x) \leq f(x) \leq h(x) \quad \text{con} \quad h(x) = C r(x)$$

Thus, we have to divide step d) of the previous algorithm in two steps, making a so called *pretest*

$$V \leq b(x)/h(x)$$

that if is verified means automatically the acceptance of  $x$ , since:

$$V \leq b(x)/h(x) \leq f(x)/h(x)$$



# Greedy squeezing algorithm

## *Initialization*

- a) determine a factorization:  $f(x) = Cr(x)g(x) = h(x)g(x)$  and find a function  $b(x) \leq f(x)$ ;

## *Body of the algorithm*

- b) generate  $X$  from a distribution with density function  $r(x)$ ;
- c) generate  $V$ , independent from  $X$ , from a standard uniform distribution;
- d) if we have  $V \leq b(X)/h(X)$  (pre-test), we accept directly  $X$ , else we go on with the step e);
- e) if we have  $V \leq g(X)$  we accept  $X$ , else we reject  $X$  and we go back to step b).



# The squeeze principle

Ultimately the squeeze principle is convenient if:

- $b(x)$  and  $h(x)$  are simple to compute, or at least it is easy to compute  $b(x)/h(x)$  (for example  $b(x)$  and  $h(x)$  may be the union of suitable line segments);
- if it is easy to generate pseudorandom numbers from  $r(x)$ ;
- if the two functions  $b(x)$  and  $h(x)$  squeeze well  $f(x)$ , meaning that the area between the curves described by  $b(x)$  and  $h(x)$  is small.



# Generation from a discrete r.v.

To generate pseudorandom numbers from the distribution of a discrete r.v. with a finite number of values and with probability  $p_j$  ( $j = 1, 2, \dots, k$ ), we can use the following procedure:

- determine initially

$$p_{max} = \max_j p_j \quad (j = 1, 2, \dots, k)$$

- generate  $I$  from a discrete uniform distribution in the range  $[1; k]$ , i.e. such that  $Prob(I = i) = 1/k$  ( $i = 1, 2, \dots, k$ ): we generate a standard uniform number  $U$  and then:  $I = \text{int}(k \cdot U + 1)$ ;
- generate a random number  $Z$  uniformly distributed in  $[0; p_{max}]$  with  $f(z) = 1/p_{max}$  (we generate a standard uniform number  $V$  and then:  $Z = V p_{max}$ );
- if  $Z < p_i$  we accept  $X = x_i$ , otherwise go back to b) (or if  $V$  is a standard uniform number, we accept  $X$  if  $V < p_i/p_{max}$ );



# Generation from a discrete r.v.

To show that the exposed method actually generates pseudorandom numbers from the distribution of interest note that:

$$\begin{aligned} \text{Prob}(I = i | Z < p_i) &= \frac{\text{Prob}(I = i \cap Z < p_i)}{\text{Prob}(Z < p_i)} = \\ &= \frac{\text{Prob}(I = i \cap Z < p_i)}{\sum_{i=1}^k \text{Prob}(I = i \cap Z < p_i)} = \\ &= \frac{(1/k) (p_i/p_{\max})}{\sum_{i=1}^k (1/k) (p_i/p_{\max})} = \frac{p_i}{\sum_{i=1}^k p_i} = p_i \end{aligned}$$

Theoretical efficiency of this method is given by:

$$\mathcal{E} = \sum_{i=1}^k \text{Prob}(I = i \cap Z < p_i) = \sum_{i=1}^k (1/k) (p_i/p_{\max}) = \frac{1}{k} \frac{1}{p_{\max}}$$



# Generation from a discrete r.v.

The average number of generations required to accept  $l$  is given by:

$$N = \frac{1}{\mathcal{E}} = k p_{max},$$

and then grows with  $p_{max}$ .

For distributions with the same number of values of the r.v. the method's efficiency will be greater for distributions with smaller modal probability.

Obviously the efficiency is greatest if trivially  $p_{max} = 1/k$ , i.e. in the case of a discrete uniform distribution.





# Example

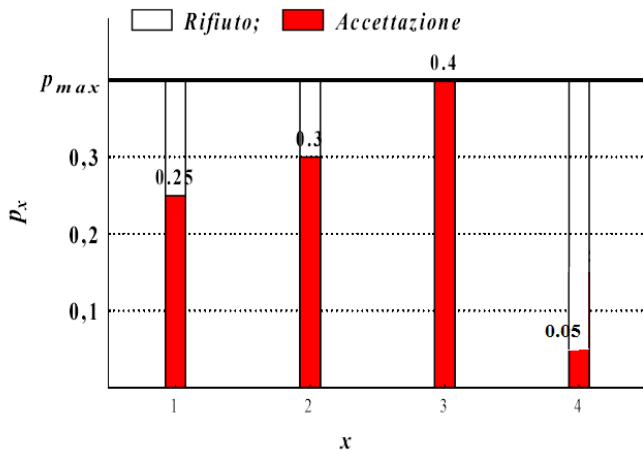
$l$	$p_i$	$p_i/p_{max}$
1	0,25	0,625
2	0,30	0,750
3	0,40	1,000
4	0,05	0,125

## Algorithm

- we generate  $U$  from a standard uniform distribution and then we put  $l = \text{int}(4U + 1)$ , so  $l$  is an integer between 1 and 4, with  $\text{Prob}(l = i) = 1/4$ ;
- we generate another standard uniform number  $V$  (acceptance test) and if  $V = p_i/p_{max}$  we accept  $l$  as originating from the distribution in question.



# Example



# Generation from a continuous r.v. (normal distribution)

Consider the positive part of a standardized normal distribution and factor the density function in this way:

$$f(x) = \sqrt{2/\pi} e^{-x^2/2} = \sqrt{2e/\pi} e^{-x} e^{-(x-1)^2/2}$$

and then we have

$$C = \sqrt{2e/\pi} = 1,3155; \quad r(x) = e^{-x}; \quad g(x) = e^{-(x-1)^2/2}.$$

Acceptance test  $V \leq g(x)$  is given by:

$$e^{-(X-1)^2/2} \geq V \Rightarrow -(X-1)^2/2 \geq \log V \Rightarrow X-1 \leq \sqrt{-2 \log V}$$

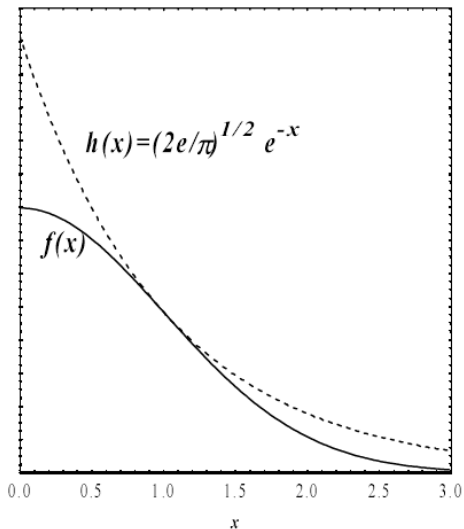
and being  $X = -\log U$ , since  $r(x)$  is the density of an exponential, ultimately we have:

$$-\log U \leq 1 + \sqrt{-2 \log V}$$

being  $U$  and  $V$  two independent standardized uniform numbers.



# Generation from a continuous r.v. (normal distribution)



# Generation from a continuous r.v. (normal distribution)

To  $X$  can be attributed a sign  $S$  using a single uniform pseudorandom number both for  $U$  and for the sign  $S$ :

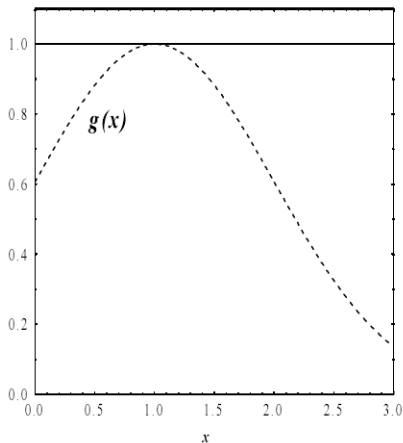
- generate a standard uniform number  $U_0$ ;
- If  $U_0 < 0.5 \Rightarrow U = 2U_0$  and  $S = -1$ ;
- If  $U_0 \geq 0.5 \Rightarrow U = 2U_0 - 1$  and  $S = +1$ ;



# Generation from a continuous r.v. (normal distribution)

The efficiency of the method is given by:

$$\mathcal{E} = P(V \leq g(x)) = 1/C = 0,7602.$$

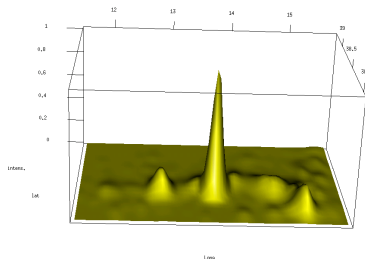


# Simple acceptance rejection for multivariate distributions

The technique of *acceptance-rejection* can also be implemented to generate random points from processes of non-homogeneous Poisson spatial point

In the figure there is the theoretical density function (or the intensity function, in the case of point processes),  $f(x,y) \leq M$ .

We can generate a point from a uniform distribution, on the region  $x, y$  and then accept it if it is  $U < \frac{f(x,y)}{M}$  being  $U$  a pseudo-random number from a uniform distribution in  $[0, 1]$



Obviously the method will be very inefficient in this case, given the great non-uniformity of the function.



# The ratio-of-uniforms method

Suppose we want to generate random numbers from a distribution with density function  $f(x)$ , and we want to use the transformation  $X = V/U$ , with  $U$  and  $V$  the coordinates of a point chosen at random, with uniform density, in a closed region  $C$ . Then  $f(U, V) = 1/\text{Area}[C]$ , conditionally to the fact that  $(U, V)$  belong to  $C$ .

Suppose that the boundaries of the region  $C$  are expressed by a limited function  $g(v/u)$  such that  $u^2 + v^2 = [g(v/u)]^2$ .

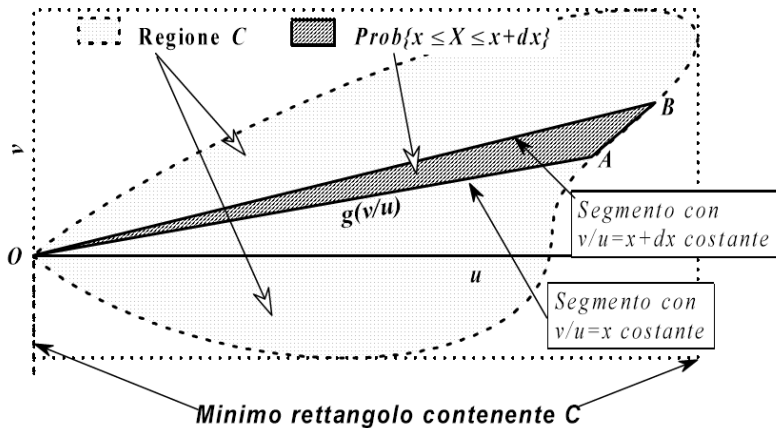
The function  $g(v/u)$  expresses the distance from the origin of points located on the border of the region  $C$ . Thus, the region  $C$  consists of all points  $P(u, v)$  such that:

$$\{u \geq 0; u^2 + v^2 \leq [g(v/u)]^2\}.$$





# The ratio-of-uniforms method



# The ratio-of-uniforms method

The event  $\{X = x + dx\}$ , i.e.  $\{X = v/u + d(v/u)\}$ , is represented on the plane  $(u, v)$  by points belonging to line segment  $OA$ , for which  $v/u$  is constant, of course.

The event  $\{X = x + dx\}$ , i.e.  $\{X = v/u + d(v/u)\}$  is similarly represented on the plane  $(u, v)$  by points belonging to line segment  $OB$ .

These two segments form an angle in  $O$  indicated with  $\theta$ .

Given the uniformity of the distribution of points in  $C$ , the probability of the event  $\{x = X = x + dx\}$  is proportional to the area of the region  $AOB$ , enclosed by the two segments  $OA$  and  $OB$  and by the line of the boundary of the region  $C$  connecting  $A$  and  $B$ .

Assuming  $dx$  very small, we can approximate the area with the triangle's area  $AOB$ , unless a higher-order infinitesimal with respect to  $(dx)^2$ , or with the area of the circular sector of Center  $O$ , radius  $OA$  and angle  $\theta$ .

# The ratio-of-uniforms method

Then, we have:

$$\text{Area}(AOB) \cong (OA)^2 \theta = [g(x)]^2 [\arctg(x+dx) - \arctg(x)] \cong [g(x)]^2 \frac{dx}{1+x^2}$$

The replacement of  $\theta$  with  $dx/(1+x^2)$  gives an approximation of the increase of  $\arctg(x)$  of a higher-order infinitesimal with respect to  $dx$ , being:

$$\frac{d\arctg(x)}{dx} = \frac{1}{1+x^2}$$

Ultimately we have:

$$\text{Prob}\{x \leq X \leq x + dx\} \propto [g(x)]^2 \frac{dx}{1+x^2},$$

$$\text{Prob}\{x \leq X \leq x + dx\} = f(x)dx$$



# The ratio-of-uniforms method

and then

$$[g(x)]^2 \frac{dx}{1+x^2} \propto f(x) dx \quad \Rightarrow \quad [g(x)]^2 = k f(x) (1+x^2)$$

with  $k$  a constant that does not depend on  $x$ .

Then, the region  $C$  is formed by the points

$$u \geq 0, \quad u^2 + v^2 \leq [g(v/u)]^2 = k f(v/u) (1 + v^2/u^2)$$

and then

$$u^2 + v^2 \leq k f(v/u) (1 + v^2/u^2) \quad \Rightarrow \quad u^2 + v^2 \leq k f(v/u) (u^2 + v^2)/u^2$$

i.e.

$$u^2 \leq k f(v/u)$$



# The ratio-of-uniforms method

Since in the region  $C$  by construction we have always  $u \geq 0$ , we ultimately have:

$$C : \left\{ (u, v) \mid 0 \leq u \leq \sqrt{k f(v/u)} \right\}$$

Therefore, the region  $C$  is bounded by a function which must be proportional to the square root of the density of the r.v.  $X$ .

The presence of the constant of proportionality is obvious if we consider that what matters is the form of the region  $C$ , and not its size: in fact the density of points in  $C$  is equal to  $1/\text{Area}[C]$ , so it's irrelevant the choice of the proportionality factor.

Think about that if  $X = V/U$  then:  $X = (aV)/(aU)$ , with  $a$  a constant.



# The ratio-of-uniforms method

To generate a random point with uniform distribution in  $C$ , the simplest method is:

- consider a rectangle  $R$  including  $C$ ;  $R$  has sides parallel to the coordinate axes (this can be done only if the region  $C$  is bounded: this happens if  $f(x)$  and  $x^2f(x)$  are limited);
- generate the two coordinates  $U$  and  $V$  uniformly distributed along the sides of the rectangle, so that the point is generated inside the rectangle;
- then accept the point if it is inside the region and compute  $X = V/U$ .



# The ratio-of-uniforms method

The efficiency of this generation in  $C$  is given by:

$$\mathcal{E} = \frac{\text{Area}(C)}{\text{Area}(R)}$$

The smallest rectangular region  $R$  that contains  $C$  consists of the points  $(u, v)$  such that:  $(u, v)$  tali che:

$$0 \leq u \leq \max(u), \quad \min(v) \leq v \leq \max(v)$$

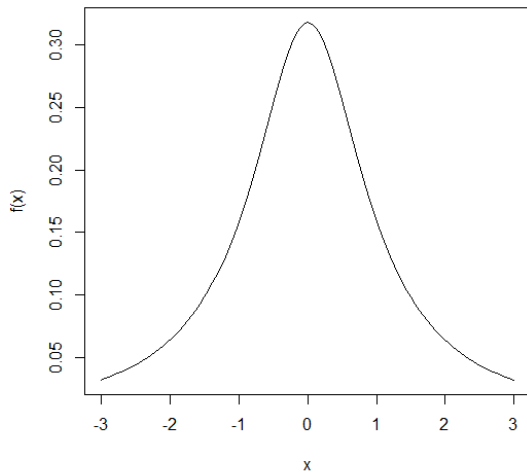
with

$$\max(u) = \sup_x \sqrt{h(x)} \quad \min(v) = \inf_{x < 0} x \sqrt{h(x)} \quad \max(v) = \sup_{x \geq 0} x \sqrt{h(x)}$$

if  $\max(u)$ ,  $\min(v)$  and  $\max(v)$  are limited (we put  $h(x) = k f(x)$ ).



# Cauchy distribution



art  
Real Sciences



dSEAS  
Department of  
Economic Statistics  
and Applied  
Mathematics



# Cauchy distribution

For the Cauchy distribution, with density function:

$$f(x) = \frac{1}{\pi (1 + x^2)}, \text{ proporzionale ad } h(x) = \frac{1}{1 + x^2},$$

the region  $C$  is bounded by:

$$k\sqrt{f(x)} = \sqrt{h(x)} = \frac{1}{\sqrt{1 + x^2}} = \frac{1}{\sqrt{1 + (v/u)^2}} = \frac{u}{\sqrt{u^2 + v^2}}$$

from which the condition:

$$u \leq \sqrt{u^2 + v^2} \text{ and then } u^2 + v^2 \leq 1.$$

The efficiency of the method is given by:  $\mathcal{E} = \text{Area}(C)/\text{Area}(R) = \pi/4$ .

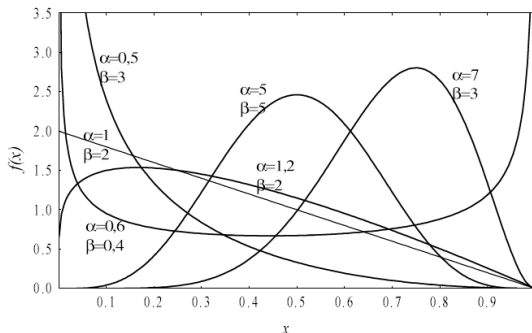


# Beta distribution (Johnk's method, 1964)

$$f(x) = \frac{x^{\alpha-1} (1-x)^{\beta-1}}{B(\alpha, \beta)}, \quad 0 \leq x \leq 1 \quad \alpha, \beta > 0$$

with

$$B(\alpha, \beta) = \int_0^1 x^{\alpha-1} (1-x)^{\beta-1} dx = \frac{\Gamma(\alpha) \Gamma(\beta)}{\Gamma(\alpha + \beta)}$$



# Beta distribution (Johnk's method, 1964)

According to the Johnk's method we can generate pseudorandom observations from a beta distribution with the following simple steps:

- $U$  and  $V$  are two independent standard uniform pseudorandom numbers;
- we put:  $Y = U^{1/\alpha}$ ,  $Z = V^{1/\beta}$  and  $X = Y/(Y + Z)$ ;
- conditionally to  $Y + Z \leq 1$ ,  $X$  is a pseudorandom number from a beta distribution with parameters  $\alpha$  and  $\beta$ .



# Beta distribution (Johnk's method, 1964)

We demonstrate that this is true.

The density functions of  $Y$  and  $Z$  are given by:

$$f_Y(y) = \alpha y^{\alpha-1}, \quad f_Z(z) = \beta z^{\beta-1}$$

Considering

$$X = \frac{Y}{Y+Z}, \quad W = Y+Z$$

we have the inverse transformations

$$y = w x, \quad z = w(1-x)$$

so it is easy to see that the Jacobian of the transformation from  $(Y, Z)$  is given by:  $J = w$ .



# Beta distribution (Johnk's method, 1964)

The non conditional density function of  $X$  and  $W$  is given by:

$$f_{X,W}(x, w) = \alpha\beta x^{\alpha-1}(1-x)^{\beta-1}w^{\alpha+\beta-1}, \quad 0 \leq x \leq 1, \quad 0 \leq w \leq 2.$$

To find the density function of  $X$ , conditional to  $W \leq 1$ , we apply the theorem of conditional probabilities:

$$f_X(x|0 \leq W \leq 1) = \frac{f_{X,W}(x, 0 \leq W \leq 1)}{P(0 \leq W \leq 1)} = \frac{\int_0^1 f_{X,W}(x, w)dw}{\int_0^1 \int_0^1 f_{X,W}(x, w)dw dx}$$

For the numerator it is easy to see that:

$$\int_0^1 f_{X,W}(x, w)dw = \alpha\beta x^{\alpha-1}(1-x)^{\beta-1}/(\alpha + \beta).$$



# Beta distribution (Johnk's method, 1964)

The denominator is just a normalization constant given by:

$$\begin{aligned} P(0 \leq W \leq 1) &= \int_0^1 \int_0^1 f_{X,W}(x, w) dw dx = \\ &= \frac{\alpha\beta\Gamma(\alpha)\Gamma(\beta)}{(\alpha + \beta)\Gamma(\alpha + \beta)} = B(\alpha, \beta) \frac{\alpha\beta}{\alpha + \beta}, \end{aligned}$$

so we have finally:

$$f_X(x|0 \leq W \leq 1) = \frac{x^{\alpha-1} (1-x)^{\beta-1}}{B(\alpha, \beta)}.$$



# Beta distribution (Johnk's method, 1964)

The efficiency of this method is given by the probability of acceptance, which can also be expressed as:

$$\mathcal{E} = P(0 \leq W \leq 1) = B(\alpha, \beta) \frac{\alpha\beta}{\alpha + \beta} = \frac{\Gamma(\alpha + 1)\Gamma(\beta + 1)}{\Gamma(\alpha + \beta + 1)}$$

$\alpha$	$\beta$				
	<b>0,25</b>	<b>0,50</b>	<b>1,00</b>	<b>2,00</b>	<b>5,00</b>
<b>0,25</b>	0,927	0,874	0,800	0,711	0,588
<b>0,50</b>	0,874	0,785	0,667	0,533	0,369
<b>1,00</b>	0,800	0,667	0,500	0,333	0,167
<b>2,00</b>	0,711	0,533	0,333	0,167	0,048
<b>5,00</b>	0,588	0,369	0,167	0,048	0,004

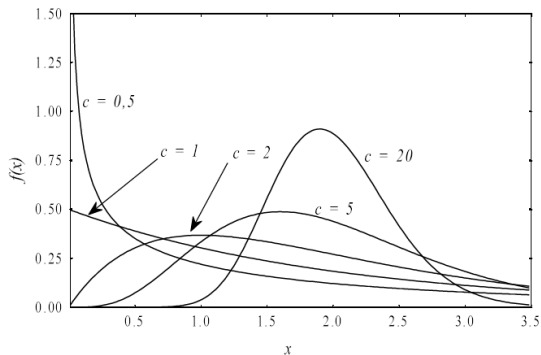


# Gamma distribution

$$f(x) = \frac{\lambda^\alpha x^{\alpha-1} e^{-\lambda x}}{\Gamma(\alpha)}, \quad x \geq 0 \quad \alpha, \lambda > 0$$

with

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x} dx$$



art



dSEAS  
Department of  
Economic Statistics  
and Applied  
Mathematics



# Gamma distribution ( $\alpha < 1$ and $\lambda = 1$ )

Consider the case  $\lambda = 1$ ; to get a random number  $Y$  from a gamma distribution with any value of the parameter  $\lambda$ , simply generate  $X$  from a gamma distribution with  $\lambda = 1$  and then put  $Y = X/\lambda$ .

We use the rejection method.

To search for a dominating function when  $\alpha < 1$  we can use the inequalities:

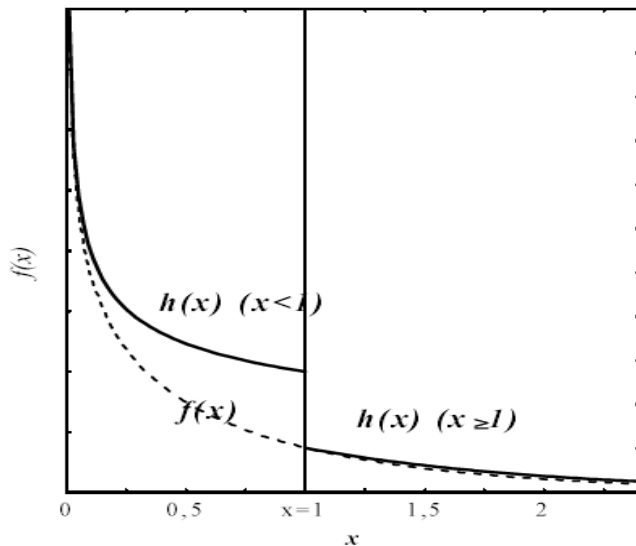
$$\begin{cases} x^{\alpha-1} e^{-x} \leq x^{\alpha-1} & 0 \leq x \leq 1 \\ x^{\alpha-1} e^{-x} \leq e^{-x} & x > 1 \end{cases}$$

so dividing both the members of the two inequalities for  $\Gamma(\alpha)$  we have:

$$f(x) \leq h(x) = \begin{cases} x^{\alpha-1}/\Gamma(\alpha) & 0 \leq x \leq 1 \\ e^{-x}/\Gamma(\alpha) & x > 1 \end{cases}$$



# Gamma distribution ( $\alpha < 1$ and $\lambda = 1$ )



# Gamma distribution ( $\alpha < 1$ and $\lambda = 1$ )

To get  $r(x)$  should be calculated  $C$  by normalizing  $h(x)$ :

$$C = \int_0^{\infty} h(x) dx = \left( \int_0^1 x^{\alpha-1} dx + \int_1^{\infty} e^{-x} dx \right) / \Gamma(\alpha) = \left( \frac{1}{\alpha} + \frac{1}{e} \right) / \Gamma(\alpha)$$

so

$$r(x) = \frac{h(x)}{C} = \begin{cases} x^{\alpha-1} / \left( \frac{1}{\alpha} + \frac{1}{e} \right) & 0 \leq x \leq 1 \\ e^{-x} / \left( \frac{1}{\alpha} + \frac{1}{e} \right) & x > 1 \end{cases}$$

To generate by  $r(x)$  we can use the method of inversion of the distribution function:

$$R(x) = \int_0^x r(t) dt = \begin{cases} x^{\alpha-1} / \left( 1 + \frac{\alpha}{e} \right) & 0 \leq x \leq 1 \\ \frac{1}{1 + \frac{\alpha}{e}} - \frac{e^{-x} - \frac{1}{e}}{\frac{1}{\alpha} + \frac{1}{e}} = 1 - \frac{e^{-x}}{\frac{1}{\alpha} + \frac{1}{e}} & x > 1 \end{cases}$$



# Gamma distribution ( $\alpha < 1$ and $\lambda = 1$ ): Algorithm

## *Initialization*

a) Set  $k = 1 + \alpha/e$ .

## *Body of the algorithm*

b) Generate  $U$  and  $V$  from a standard uniform distribution;

c) if  $U \leq 1/k$  put  $X = (U k)^{1/\alpha}$ , else go on to e)  
(First part of the distribution);

d) if  $V = e^{-X}$  accept  $X$  else reject  $X$  and go back to b);

e)  $X = -\log[k(1 - U)/\alpha]$  (Second part of the distribution);

f) if  $V = X^{\alpha-1}$  accept  $X$  else reject  $X$  and go back to b).



# Gamma distribution ( $\alpha < 1$ and $\lambda = 1$ )

As usual, theoretical efficiency is given by  $1/C$  (probability of acceptance):

$$\mathcal{E} = \frac{1}{C} = \frac{\Gamma(\alpha)}{1/\alpha + 1/e} = e \frac{\Gamma(\alpha + 1)}{\alpha + e}$$

We have also:

$$\lim_{\alpha \rightarrow 0^+} \mathcal{E} = 1, \quad \lim_{\alpha \rightarrow 1} \mathcal{E} = \frac{e}{1 + e} \cong 0,731$$

$\alpha$	0,010	0,100	0,250	0,500	0,750	0,900
$\mathcal{E}$	0,991	0,918	0,830	0,749	0,720	0,723



# Gamma distribution ( $\alpha > 1$ and $\lambda = 1$ )

In general, are  $f(x)$  and  $r(x)$  two density functions and  $F(x)$  and  $R(x)$  the corresponding distribution functions.

We want to generate random numbers from  $F(x)$ , but we know how to generate from  $R(x)$  by inversion of the distribution function.

We set:

$$M = \max_x \left( \frac{f(x)}{r(x)} \right)$$

## Algorithm

- generate  $U$  and  $V$  from a standard uniform distribution;
- generate  $X$  from  $R(\cdot)$ , i.e.  $X = R^{-1}(U)$ ;
- if  $\frac{f(x)}{M r(x)} \geq V$  then  $X$  is accepted.



# Gamma distribution ( $\alpha > 1$ and $\lambda = 1$ )

To obtain random numbers from a gamma distribution with  $\alpha > 1$  and  $\lambda = 1$ , we can use:

$$R(x) = \frac{x^h}{\alpha^h + x^h}$$

$$r(x) = \frac{h \alpha^h x^{h-1}}{(\alpha^h + x^h)^2}$$

with

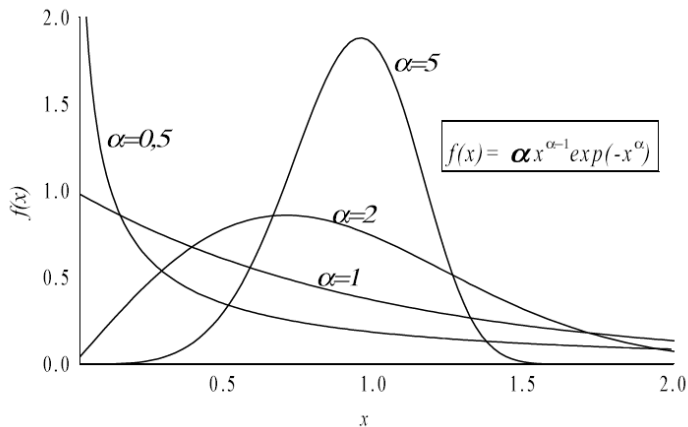
$$h = \sqrt{2 \alpha - 1}$$

The algorithm has an efficiency that, for large values of  $\alpha$ , tends to  $\sqrt{\pi/2}$ .



# Weibull distribution

$$f(x) = \lambda \alpha x^{\alpha-1} e^{-\lambda x^\alpha}; \quad F(x) = 1 - e^{-\lambda x^\alpha}; \quad x \geq 0$$





# Weibull distribution

To generate  $X$  from such distribution, it is sufficient to observe that

$$Y = X^\alpha$$

has an exponential distribution with  $\lambda$  parameter, so we need to generate only  $Y$  from this distribution and then get  $X$  using the inverse transformation

$$X = Y^{1/\alpha};$$

so, ultimately:

$$X = [(-\log U)/\lambda]^{1/\alpha}$$

is a random number from a Weibull distribution, with  $U$  a random number generated from a standard uniform distribution, as usual.



Compared to the techniques used for univariate distributions, be aware that for generating pseudorandom numbers vectors:

- the technique of inversion of the distribution function is not directly applicable because  $F(X_1, X_2, \dots, X_n) = U$ , in general, has not a unique solution  $\{X_1, X_2, \dots, X_n\}$ .
- we can generate pseudorandom numbers vectors through transformations of multiple random variables.
- rejection method is broadly applicable, although not always leads to efficient algorithms.
- the ratio-of-uniforms method is generalizable to the multivariate case.



# Use of the conditional distributions

A rather general technique makes use of conditional distributions (Rosenblatt, 1952).

$$f(x_1, x_2, \dots, x_n) = f(x_n | x_1, x_2, \dots, x_{n-1}) \cdot f(x_{n-1} | x_1, x_2, \dots, x_{n-2}) \cdots \\ f(x_3 | x_1, x_2) \cdot f(x_2 | x_1) \cdot f(x_1)$$

Then, we may proceed as follows:

- generate a random number  $X_1^*$  from distribution with density function  $f(x_1)$ ;
- generate  $X_2^*$  from distribution with density function  $f(x_2 | X_1^*)$ ;
- go on in this way until the generation of the last vector component,  $X_n^*$  from distribution with density function  $f(x_n | X_1^*, X_2^*, \dots, X_{n-1}^*)$ .
- $(X_1^*, X_2^*, \dots, X_{n-1}^*, X_n^*)$  is a random vector from the n variate distribution with joint density function  $f(x_1, x_2, \dots, x_n)$ .



# Generation of pseudorandom vectors from a multivariate normal distribution

To generate pseudorandom vectors  $\mathbf{y}$  from a  $p$  variate normal distribution with density function:

$$f(\mathbf{y}; \boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}_y|}} \cdot \exp \left\{ -\frac{1}{2} [(\mathbf{y} - \boldsymbol{\mu}_y)^\top \boldsymbol{\Sigma}_y^{-1} (\mathbf{y} - \boldsymbol{\mu}_y)] \right\}$$

we have to determine first a matrix  $\mathbf{A}$  such that:

$$\mathbf{A} \mathbf{A}^\top = \boldsymbol{\Sigma}_y$$

(the matrix  $\mathbf{A}$  can be determined using the Cholesky decomposition).



# Generation of pseudorandom vectors from a multivariate normal distribution

At this point, just generate a vector  $\mathbf{x}$  of  $p$  independent normal standardized pseudo-random numbers and compute:

$$\mathbf{y} = \mathbf{A} \mathbf{x} + \boldsymbol{\mu}_y$$

In fact, remembering that

$$E(\mathbf{x}) = \mathbf{0}, \quad \boldsymbol{\Sigma}(\mathbf{x}) = E \left[ (\mathbf{x} - \boldsymbol{\mu}_x) (\mathbf{x} - \boldsymbol{\mu}_x)^T \right] = E(\mathbf{x} \mathbf{x}^T) = \mathbf{I}$$

we have

$$E(\mathbf{y}) = E[\mathbf{A} \mathbf{x} + \boldsymbol{\mu}_y] = \mathbf{A} E(\mathbf{x}) + \boldsymbol{\mu}_y = \boldsymbol{\mu}_y$$

$$\boldsymbol{\Sigma}(\mathbf{y}) = E \left[ (\mathbf{y} - \boldsymbol{\mu}_y) (\mathbf{y} - \boldsymbol{\mu}_y)^T \right] = E \left[ (\mathbf{A} \mathbf{x}) (\mathbf{A} \mathbf{x})^T \right] = \mathbf{A} E(\mathbf{x} \mathbf{x}^T) \mathbf{A}^T = \boldsymbol{\Sigma}_y$$



# Mixtures of multivariate normal distributions

The density function of a mixture of  $k$  multivariate normal distributions with  $m$  components is given by:

$$f(\mathbf{x}) = \sum_{i=1}^k p_i f_i(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \quad \text{con} \quad \sum_{i=1}^k p_i = 1$$

To generate a random number from this mixture of distributions you must:

- generate an index  $j$  from a discrete distribution with probability  $p_j$ ;
- generate then the pseudo-random vector  $\mathbf{x}$  from  $f_j(\cdot)$  in correspondence of  $j$ .



# Contingency tables with margins not assigned

Let  $A$  and  $B$  be two characters with values  $a_i$  ( $i = 1, 2, \dots, r$ ) and  $b_j$  ( $j = 1, 2, \dots, c$ ); to generate a table we need to generate  $N$  pseudorandom value pairs  $(A_k, B_k)$  ( $k = 1, 2, \dots, N$ ), so that  $Prob(A_k = a_i, B_k = b_j) = p_{ij}$ .

## Algorithm

- first generate  $A_k$  from the marginal distribution of  $A$ , with probability  $p_{i.}$ , choosing the row in the table.
- conditionally to this choice, generate  $B_k$  from the  $i$ -th conditional distribution of  $B$ , with probability:  $p_{j|i} = p_{ij}/p_{i.}$



# Contingency tables with margins assigned

## Algorithm

### *Initialization*

- a) set  $N$  the total of the considered table frequencies.
- b) set  $T_i = n_{i.}$  ( $i = 1, 2, \dots, r$ ) and  $S_j = n_{.j}$  ( $j = 1, 2, \dots, c$ );
- c) set  $M = 0$ ;
- d) set  $n_{ij} = 0$  ( $i = 1, 2, \dots, r; j = 1, 2, \dots, c$ );





# Contingency tables with margins not assigned

## Body of the algorithm

- e) Generate  $I$  from a discrete distribution with probabilities  $p_i$  given by:  
 $p_i = T_i / (N - M) \quad (i = 1, 2, \dots, r);$
  
- f) Generate  $j$  from a discrete distribution with probabilities  $p_j$  given by:  
 $p_j = S_j / (N - M) \quad (j = 1, 2, \dots, c);$
  
- g) Update  $n_{IJ} = n_{IJ} + 1;$
  
- h) Update:  $M = M + 1; T_I = T_I - 1; S_J = S_J - 1;$
  
- i) If  $M < N$  go back to step e) else stop the algorithm.



# The system of R for the management of probability distributions

The pseudo random number generation from a particular distribution is within the general system of R for statistical distributions

## R: The Gamma Distribution

Density, distribution function, quantile function and random generation for the Gamma distribution with parameters shape and scale.

Usage

```
1 dgamma (x, shape, rate = 1, scale = 1/rate, log = FALSE)
2 pgamma (q, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p
   = FALSE)
3 qgamma (p, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p
   = FALSE)
4 rgamma (n, shape, rate = 1, scale = 1/rate)
```

codiceGamma1.R



# Example: the gamma distribution in R

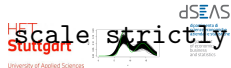
`x, q`  
vector of quantiles.

`p`  
vector of probabilities.

`n`  
number of observations.

`rate`  
an alternative way to specify the scale.

`shape, scale`  
shape and scale parameters. Must be positive, **scale strictly.**



# Generating random numbers with various distributions of R

Routines to generate random numbers in R are fairly simple to use and belong to one scheme of function:

```
1 unigenera<-function(n=1,distribution="uniform",par=c(0,1)){
2   x=switch(distribution,
3     "uniform" = runif(n,min=par[1],max=par[2]),
4     "normal"   = rnorm(n,mean=par[1],sd=par[2]),
5     "gamma"    = rgamma(n,shape=par[1],rate=par[2]),
6     "beta"     = rbeta(n,par[1],par[2]),
7     "binomial" = rbinom(n,par[1],par[2]),
8     "poisson"  = rpois(n,par[1]),
9     "logistic" = rlogis(n,location=par[1],scale=par[2]),
10    "student"   = rt(n,par[1]),
11    "weibull"   = rweibull(n,par[1],scale=par[2]),
12    "lognormal" = rlnorm(n,meanlog=par[1],sdlog=par[2])
13  )
14  return(x)
15 }
```

codiceSimul2.R



- For each distribution there are now many algorithms for generating random numbers and it is likely that some topic may soon become **obsolete (or be already!)** but it is always a risk for everything that is somehow linked not only to theoretical, but also to technological developments;
- the continuous increase of processing speed and the evolution of processors (at least for the PC's), often frustrate the need for subtle technical tricks measures that were used to save some operations in the execution of an algorithm for generating random numbers!

## example

There is a lot of examples of obsolete algorithms



## Simulated distributions

*The idea behind Monte Carlo methods is that the next best thing to knowing the statistics of a distribution-or equivalently, its density-is to have a very large sample from that distribution. Since the empirical cumulative distribution function (ecdf) converge to the true cdf, any function of the ecdf will converge to the corresponding function of the cdf itself*

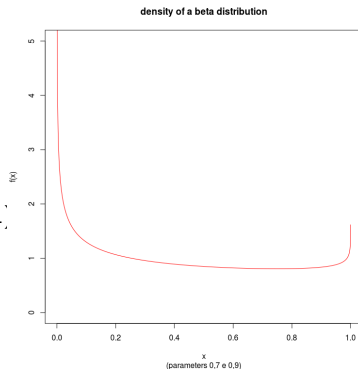
(Thisted, 1988, pag. 303)

- If you do not know the exact distribution of a function of sample data, the best thing to do is to **extract a sample as large as possible!**
- This is exactly what you do when you do not know the value of a specific statistics in a real population: **you draw a sample!**
- Clearly, the greater  $m$  (the amplitude of the extracted sample), the better the approximation to the sampling distribution

# Simulation of sampling distributions: Beta distribution

Consider a beta distribution with parameters  $\alpha = 0.7$  and  $\beta = 0.9$ .

$$f(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} \quad (0 \leq x \leq 1)$$



- You might wonder what shape has the sampling distribution of averages of sample of size  $n$  drawn from this distribution
- We know, however, at least the first and second theoretical moments of the distribution, but what shape will it take?
- In particular, we are interested in small values of  $n$  (because for large  $n$ , large sample theory helps us)

# Simulated sampling distribution of arithmetic mean

- To visualize the concept of simulated sampling distribution, observe the example.
- I put  $n = 5$
- I drew 20 samples of size 5 of pseudo-random numbers from a beta distribution with parameters  $\alpha = 0.7$  and  $\beta = 0.9$ .
- the R code is as follows:

```
1 x =rbeta(100, shape1=0.7, shape2=0.9)
2 x =matrix(x,20,5)
3 means =apply(x,1,mean)
```

codiceMBeta1.R





## An Example

The 20 samples of size 5:

```
[, 1] [, 2] [, 3] [, 4] [, 5]
[1,] 0,611 0,083 0,079 0,833 0,340
[2,] 0,087 0,475 0,010 0,008 0,866
[3,] 0,107 0,589 0,399 0,831 0,907
[4,] 0,194 0,630 0,546 0,479 0,873
... ..
... ..
[19,] 0,937 0,937 0,962 0,173 0,533
[20,] 0,106 0,055 0,260 0,596 0,509
```

The arithmetic means  
calculated on 20 samples:

```
[1] 0,389 0,289 0,567 0,544
0.609 0.622 0.577
0.537 0.412 0.613
0.386 0.423 0.322
0.698 0.539 0.288
0.407 0.388 0.708
0,305
```

**These averages are a sample of size 20 drawn from the theoretical distribution averages of samples of size 5 from a beta distribution with parameters  $\alpha = 0.7$  and  $\beta = 0.9$ .**



# Simulation with $m = 100000$

- Clearly, 20 samples were too few to get information of any use
- I rerun the same code with some modifications in R

```
1 m =100000
2 n =5
3 x =rbeta(n*m,shape1=0.7, shape2=0.9)
4 x =matrix(x,m,n)
5 means =apply(x,1,mean)
6 moments =first4(means)
7 moments$moments
8 [1] 0.4375046723 0.0189213123 0.0002306912 0.0009818761
```

codiceMBeta2.R

## R execution

- Now we have a sample of size 100000! We can expect that the information in this new sample are much more precise and that we can get a very good approximation of the real sampling distribution of  $M_5$

# Descriptive statistics of the simulated distribution

- It is convenient to summarize the simulated distribution by some descriptive statistics

- > Summary (average)

Min 1st Qu. Mean Median 3rd Qu. Max

0.02043 0.34060 0.43520 0.43750 0.53220 0.94330

- = first4 moments (averages)

[1] 0.4375046723 0.0189213123 0.0002306912 0.0009818761

The theoretical values of the first two moments of the distribution of  $M_5$  are

$$E[M_5] = E[X] = \frac{\alpha}{\alpha + \beta} = 0.4375$$

$$V[M_5] = \frac{V[X]}{n} = \frac{1}{n} \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)} = 0.01893029$$

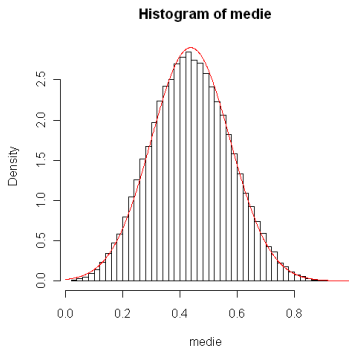


The sample of 100,000 elements will approximate well the **true sampling distribution**.

In the histogram the density of a normal distribution has been superimposed (for simplicity, with data parameters from the mean and the variance of the simulated distribution)

**Obviously the size of individual samples ( $n = 5$ ) is too small to let us think to approximate the sampling distribution through the normal one!**

For the graphics we used the R code:



```
1 xvec=seq(0,1,by=0.001)
2 hist(medie,nclass=50,freq=FALSE)
3 lines(xvec,dnorm(xvec,mean=moments$moments[1],
4             sd=sqrt(moments$moments[2])),col="red")
```

codiceMBeta3.R



# Comments on the example

Some general considerations on this simple example:

Keep in mind the deep difference between  $m$  number of simulations and  $n$ , the individual samples size

## Differences between $m$ and $n$

- $m$  relates the amplitude of the simulated distribution: the results obtained generally have an average error proportional to  $\frac{1}{\sqrt{m}}$  (beyond a certain limit we do not gain too much when increasing  $m$ )
- $n$  relates to the sampling distribution being simulated: changing  $n$  means to study the distribution of another r.v.  $T_n$

Discussion on the example:

- 1 Increasing  $m$  beyond 100,000 give use more precision (less simulation variance), not necessary useful
- 2 Changing  $n$  means investigating the sampling distribution of another r.v. ( $M_{10}$  instead of  $M_5$ )

# Comments on the example

- In R is particularly easy to generalize the scheme used.
- For example, it is enough to know how to generate numbers from other distributions.
- In general we might be interested in the simultaneous distribution of  $k$  estimators.
- Since the algorithm of generation of random number generates a sequence starting from some initial values, **it is always possible to repeat the same sequence**, storing the value of the starting seed (`.Random.seed` in R)



# A general simulation scheme

- to simulate the sampling distribution of  $k$  estimators based on  $m$  samples of  $n$  pseudo-random numbers (or vectors) generated from a particular population, a general structure of the program that generalizes the example shown on the beta distribution can be used.

```
1 m =100000
2 n =5
3 x =rbeta(n*m,shape1=0.7, shape2=0.9)
4 x =matrix(x,m,n)
5 means =apply(x,1,mean)
6 moments =first4(means)
7 moments$moments
8 [1] 0.4375046723 0.0189213123 0.0002306912 0.0009818761
```

codiceMBeta2.R

- quite simply in the previous rows we can change  $m$ ,  $n$ , the type of distribution (beta with certain parameters, or other distributions) or we can compute other estimates of which we want to approximate the sampling distribution

# A general algorithm

**procedure** SIMULATED.SAMPLING.DISTR( $m, n, k, \text{type.pop}$ )

- ▷  $m$  is the number of simulated samples
- ▷  $n$  is the size of each sample
- ▷  $k$  is the number of estimators to study
- ▷  $\text{type.pop}$  is the type of population

**for**  $j = 1 \rightarrow m$  **do**

Generate the  $j$ -th sample  $\mathbf{x}$  of size  $n$  from  $\text{type.pop}$

**for**  $h = 1 \rightarrow k$  **do**

$t(h) = \text{compute.estimate}[h](\mathbf{x})$

- ▷ e.g.  $t(1) = \text{median}(\mathbf{x})$ ,  $t(2) = \text{trimmed}(\mathbf{x})$

$\text{estimate}(j, h) = t(h)$

**end for**

- ▷ End operations  $j$ -th sample

**end for**

- ▷ End of samples generation

Summarize( $\text{estimate}$ )

- ▷ Estimation of the  $k$  empirical distributions

**end procedure**



- at the end of the program we may conduct any descriptive analysis on the  $k$  columns of the matrix *estimate* as for example the calculation of empirical significance levels of tests or computation of percentiles of  $k$  columns, useful when we want to approximate the distributions of test and we do not know the exact distributions.
- It was assumed, in the skeleton of the algorithm, that there is the ability to store the  $m \times k$  simulated estimates in a matrix. This hypothesis may be unrealistic if  $m$  is the order of million and if  $k$  is not small
- In this case, we should modify the procedure so that, for the vector estimates  $t()$  obtained for the  $j$ -th sample, we should update sums and sums of squares if we want the sample means, variances and covariance of the simulated distributions of the estimates  $t$
- we will save memory, but you will be required to know in advance which synthetic summary of the  $k$  simulated sampling distributions  $t$  are needed.
- Alternatively the values of  $t()$  can be stored on file (for each sample): this can lead to considerable inefficiencies, since it obliges for each sample to writing data to files that could be slower than the step of computation for each of the samples.



- In fact, if one is interested not only in the first two moments of simulated distribution of the estimators, but also in estimating its shape, we should compute the frequency distribution in  $h$  classes of values of  $k$  estimators, for construction of histograms.
- this will involve the disadvantage that the system of intervals must be already prepared, so we would need to a rough idea of the magnitude of range of each estimator and fix a value of  $h$  sufficiently large.



# Repeat a simulation

- Remember that the use of a generator based on recursive relations of any kind, allows the possibility of *repeat the simulation with the same samples*, if we store the value of the vector of initial seeds  
`.Random.seed`
- It is always a good practice to store the initial value, so to be able to repeat the same simulation, adding the computation of other quantities, or we can make a first assessment of the ranges of variation of the amount covered by the simulation, and then computing histograms and frequency distribution in a second execution.



# R code for a basic scheme of simulation

## code `simulgen1` to rewrite with english comments

```
1
2
3 #
4 #
5 # routines di simulazione a UNA variabile
6 #
7 # per ogni distribuzione "distr" è necessario definire una serie di
  defaults:
8 #
9 # paramdefaults[[distr]]
10 # limitdefaults[[distr]]
11 # una definizione di estrazione di vettori pseudo-casuali in
    unigenera (o bivgenera per le bivariate)
12 #
13 #
14 #
15 #
```

# R code for a basic scheme of simulation

```
1 univariate.samples<- function(nsamples=10000,n=5,modello="normal",
2   parm=c(0,1)){
3   if(missing(parm)) parm <- paramdefaults[[modello]]
4   nsamples = min(sup.nsamples, trunc(abs(nsamples)))
5   n = min(sup.n, trunc(abs(n)))
6   x = unigenera(nsamples*n,distribution=modello,par=parm)
7   x = matrix(x,nsamples,n)
8   return(list(x=x,m=apply(x,1,mean), median=apply(x,1,median),
9     central=apply(x,1,central), var.n=(n-1)*apply(x,1,var)/n
10 )) }
```

codiceSimulgen2.R



# Simulation for regression models

Simulation of extraction of samples from general regression models.  
initialization:

- 1 Fix the sample size  $n$  and the number of samples to generate,  $m$ .
- 2 Fix  $n$  vectors of  $p$ -component  $\mathbf{x}_i$ ,  $i = 1, 2, \dots, n$ , (with a random choice, or according to a fixed design).
- 3 Fix the values of the  $k$  components vector of parameters  $\theta$ .
- 4 Fix the values of  $\psi$ .
- 5 Compute the  $n$  theoretical values  $\mu_i = g(\mathbf{x}_i, \theta)$ ,  $i = 1, 2, \dots, n$ ;  
generation of the  $j$ -th sample:
- 6 then for the  $j$ -th sample of simulated  $j = 1, 2, \dots, m$  generates a sample of size  $n$  (i.e.  $\epsilon_{j1}, \dots, \epsilon_{jn}$ ) from the distribution  $f(\epsilon, \psi)$  established for accidental component
- 7 The single sample is obtained putting  $y_{ji} = h(g(\mu_i), \epsilon_{ji})$ ,  
 $i = 1, 2, \dots, n$
- 8 On each sample various quantities are computed;



# Standard errors of simulation

- The standard error of the estimated quantities in a study based on  $m$  simulations is proportional to  $\frac{1}{\sqrt{m}}$
- If  $m$  is large, the distributions of means and variances of the simulated distributions are approximated by the normal distribution.
- Let  $M_m(Z)$  and  $S_m^2(Z)$  the mean and variance of the  $m$  simulated values of  $Z$  (an estimator, a test, a 0-1 variable result of a test and so on)
- If the true values of the mathematical expectation and variance (generally unknown) of the estimator  $Z$ , are  $E[Z]$  and  $V[Z]$  then one can construct  $(1 - \alpha)$  level asymptotic confidence intervals (certainly good approximations if  $m$  is very large)



# Standard errors of simulation studies

- These results are very important and provide good approximations because  $m$  is generally high and normal approximations to distributions of  $M_m(Z)$  and  $S_m^2(Z)$  are undoubtedly very accurate.
- these results come from the fact that

$$\frac{M_m(Z) - \mathbb{E}[Z]}{\sqrt{S_m^2(Z)/m}} \rightarrow N(0,1)$$

$$\frac{S_m^2(Z) - \mathbb{V}[Z]}{\sqrt{(\mu_4(Z) - S_m^4(Z))/m}} \rightarrow N(0,1)$$

( $\mu_4(Z)$  is the fourth central moment of  $Z$ )





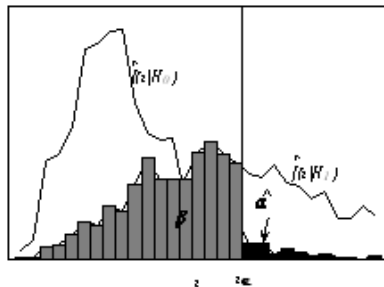
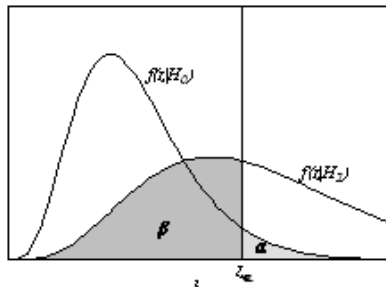
Asymptotic  $1 - \alpha$  confidence intervals for simulated mean and variance of an estimator  $Z$

$$M_m(Z) - k_\alpha \frac{S_M(Z)}{\sqrt{m}} \leq E[Z] \leq M_m(Z) + k_\alpha \frac{S_M(Z)}{\sqrt{m}}$$

$$S_m^2(Z) - k_\alpha \sqrt{\frac{\mu_4(Z) - S_m^4(Z)}{m}} \leq V[Z] \leq S_m^2(Z) + k_\alpha \sqrt{\frac{\mu_4(Z) - S_m^4(Z)}{m}}$$



- Simulated significance level of a test based on the binomial theory



- Theoretical distribution (left) of a generic z-test under the null hypothesis and under an alternative hypothesis: theoretical probability of errors of first type (and of the second type). On the right the simulated distribution of the same z-test under the null hypothesis and under an alternative hypothesis: the estimated probabilities of errors of first and second kind.

# Simulated significance levels

- Even a significance level can be estimated through a simulation, if the sampling distribution of  $Z$  under  $H_0$  is not known. Since  $\alpha$  is a probability, a basic idea is to **estimate  $\alpha$  through a relative frequency of success**.
- Suppose there is a fixed area of rejection  $R_\alpha$
- generated  $m$  independent samples of size  $n$  from the distribution of interest or from a distribution for which  $H_0$  is true;
- on each of them calculate the value of the test  $z_j$ ,  $j = 1, 2, \dots, m$
- $z_j$  occurs ( $z_j = 1$ ) if  $z_j \in R_\alpha$  that is if for the  $j$ -th sample simulated we reject the null hypothesis, **which we know to be true by construction**

At the end we count  $m_R$ , the number of values  $z_j$  falling in the rejection region and estimate:

$$\hat{\alpha} = \frac{m_R}{m}, \quad \text{with} \quad \hat{\sigma}(\hat{\alpha}) = \sqrt{\frac{\hat{\alpha}(1 - \hat{\alpha})}{m}}$$



- The above approach of course can be extended to compute a **simulated power of a test**
- It is still based on the binomial theory (we will count samples in the rejection region)
- Of course we will sample from  $H_1$  possibly for a sequence of parameter values for the alternative hypothesis, so to have a **simulated power function**
- The simulation approach is adaptable to the estimation of the level of coverage of confidence intervals **discussion on possible examples** We will count the number of times that the interval contains the *true value of  $\theta$*



The advantages are:

- Great flexibility of use, and applicability to a large number of situations, at least all those for which can be set to a parametric model;
- Great simplicity: in fact for elementary applications it is sufficient to know how to generate samples of the particular random numbers and be able to calculate the estimator or the test of interest on each sample, so this technique is usable at different levels of the study; rapidity of application;
- Independency from the number of dimensions (at least there is a computational complexity linearly dependent on  $d$ , but not explosive!)
- ease of communication: it is now very common the use of simulation techniques in scientific work, it is rarely necessary to explain in detail the particular pattern used.



- The value of a simulation study can also be of a mainly comparative: to show that an estimator is better than another, that a test is more powerful than another, that a certain approximation is satisfactory analytical (or it is it better than others) for certain values of  $n$ , etc..
- The simulation techniques provide a very flexible tool to empirically evaluate the effectiveness of some inferential techniques that you do not know the exact characteristics, as we will see with some examples



# Disadvantages of simulation techniques

The disadvantages, in my view, are basically:

- it is an empirical approach: one thing is, for example, to show through simulation that increasing  $n$  the simulated level of significance of a test approaches the nominal value, (or on the contrary, that it differs in a significant way), an other thing is instead to show analytically that the significance level tends to some value  $\alpha$  when  $n$  diverges!
- The results, albeit with the cautions of the previous point, relate only to situations actually tested, so if a certain result is confirmed with a large simulated samples drawn from a population that depends on a parameter  $\theta$  , and we fixed for example 10 values of  $\theta$  , nothing in general ensure use that the results can be extended to any value of  $\theta$  !
- You could also make another objection: it is not analitically very elegant to use a simulation study if you can use some asymptotic analytical result even if only approximated.



Exact sampling distributions are very few, compared to the infinite range of possible models that could be built, or otherwise used in practice. Most of the time are known only asymptotic approximations including the most classical used in statistics





# Mixtures of multivariate normal distributions

- Generate random numbers from mixtures of  $k$ ,  $p$ -variate normal distributions with proportions of the mixture given by:

$$\{p_1, p_2, \dots, p_j, \dots, p_k\}:$$

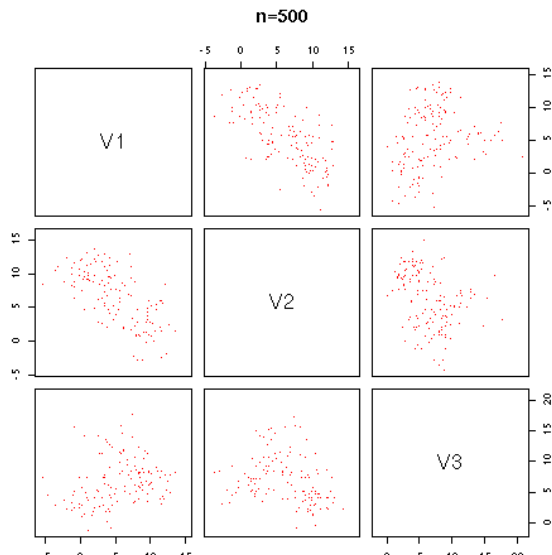
theoretical aspects (technically very simple)

- 1 You can p.e. first generate a random integer  $J$  between 1 and  $k$  from a distribution with probability given by the ratio  $p_j$
  - 2 we extract a random vector from the  $J$ -th component
- R package:  
library(ks)
  - function: `rmvnorm.mixt (n = 100, mu = c (0.0), Sigmas = diag (2), props = 1)`



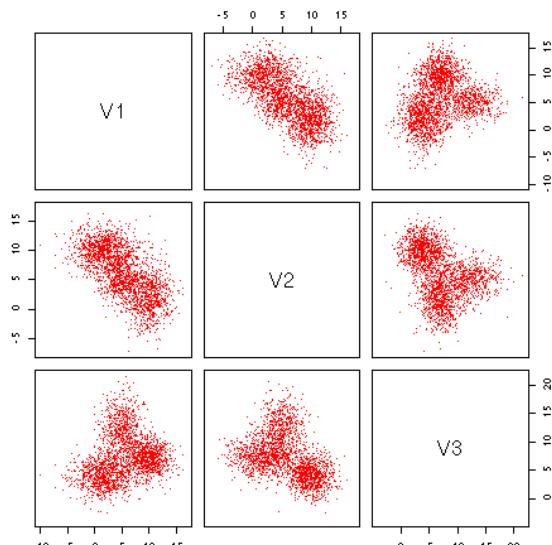
# Examples of multivariate distributions: $n = 500$

Generation of 500 numbers from a trivariate distribution



# Examples of multivariate distributions: $n = 10000$

Generation of 10000 numbers from a distribution trivariate





# Some other development


- Generation from an  $ARMA(p, q)$  model;
- Generation from a space-time Point Process
- Generation of non parametric estimators sampling distribution
- variance reduction techniques





# References I

 Box G.E.P., Muller M.E., (1958)  
A note on the generation of random normal deviates.  
*Annals of Mathematical Statistics*, 29:610–611.

 Chiodi M., (1998)  
*Tecniche di Simulazione in Statistica*.  
Rocco Curto Editore.

 Dodge Y., (1996)  
A natural random number generator.  
*International Statistical Revue*, 64(3):329-344.

 Eichenauer-Herrmann J., (1992)  
Inversive congruential pseudorandom numbers: a tutorial.  
*International Statistical Revue*, 60(2):167–176.

 Johnk M.D. (1964)  
Erzeugung von Betarerteilten und Gammaverteilten Zufallszahlen.  
*Metrika*, 8:5–15.





Knuth D., (1981)

*The Art of Computer Programming: Seminumerical Algorithms (Vol.2).*  
Addison–Wesley Edition.

▶ L'Ecuyer P., Simard R., (2005)

TestU01: A software library in ANSI C for empirical testing of random number generators.

*Laboratoire de simulation et d'optimisation. Université de Montréal IRO.*

▶ Marsaglia G., (1985)

The Marsaglia random number CDROM, with the Diehard battery of tests of randomness.

*Florida State University under a grant from The National Science Foundation.*








Metropolis N., Ulam M., (1949)

The Monte Carlo Method.






*Journal of the American Statistical Association, 44:335–341.*



-  Piccolo D., (1998)  
*Statistica*.  
Il Mulino.
-  Ripley B.D., (1983)  
Computer generation of random variables: a tutorial.  
*International Statistical Review*, 51:301–319.
-  Ripley B.D., (1990)  
Thoughts on pseudorandom number generators.  
*J. Comput. Appl. Math.*, 31:153–163.
-  Rosenblatt M., (1952)  
Remarks on a Multivariate Transformation.  
*Annals of Mathematical Statistics*, 23:470–472.
-  Stigler S.M., (1991)  
Stochastic Simulation in the Nineteenth Century.  
*Statistical Science*, 6:89–97.



# References IV

-  Tausworthe R.C., (1965)  
Random numbers generated by linear recurrence modulo two.  
*Math. Comp.*, 19:201–209.
-  Thisted R., (1988)  
*Elements of statistical computing: numerical computation.*  
Chapman and Hall.
-  Ulam S., (1976)  
*Adventures of a mathematician.*  
Charles Scribner's sons.
-  Von Neumann J., (1951)  
Various techniques used in connection with random digits.  
*Monte Carlo Method, U.S: Nat. Bur. Stand. Appl. Math. Ser.*, 12:36–38.
-  Wichmann B.A., Hill I.D., (2006)  
Generating good pseudorandom numbers.  
*Computational Statistics and Data Analysis*, 51:1614–1622

