Network Analysis 2
Statistical Approaches and Modeling

Random

Vladimir Batagelj

IMFM Ljubljana, IAM UP Koper, and NRU HSE Moscow

XIII Summer School of the ANR-Lab

Network Analysis and Contemporary Decision Sciences

International Laboratory for Applied Network Research

NRU HSE, Moscow, August 2022

Outline

Vladimir Batagelj: vladimir.batagelj@fmf.uni-lj.si

Current version of slides (August 17, 2022 at 03 : 24): slides PDF

Sequences of random numbers

A computer, at least as seen by the user, is a deterministic machine: the same program for the same data is supposed to give the same results. On the other side, we would need in our programs random events such us:

- tossing a coin or a die;
- shuffling a deck of cards or other objects (data);
- (unbiased) sampling from a larger set;

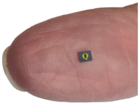
or to analyze some processes

- collection of trading cards (baseball , animals, . . .);
- traffic in the city;
- queues in the market.

How to get (at least the impresion of) random behavior on a computer?
In some applicatios we can simply use some quantities (such as system clock) on which the user has no influence.

Randomness

TrueRNG-V3; Amazon



ComScire

Real *randomness* can be achieved by providing a special hardware device (random card or stick). In computing, a hardware random number generator (HRNG) or true random number generator (TRNG) is a device that generates random numbers from a physical process, rather than by means of an algorithm (WP, HRNG).

It turned out that is much less expensive and reliable if we simply compute a sequence of numbers that behaves as random (unpredictable). Because such a sequence is essentially deterministic it is called a *(pseudo) random numbers* sequence. In the following we will usually omit the term pseudo.

Basic sequences of random numbers

Sequences of random numbers used in computers are usually uniformly distributed on the interval $[0, 1)$. We call them *basic* and are denoted with (r_i). From them we get, using different transformatins, random sequences having other distributions.

Having a selected distribution is not enough for a sequence to be considered random. It has also to behave randomly. For example, the sequence $(i \bmod 3 : i \in \mathbb{N})$.

$$0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, \ldots$$

is, as a source of numbers $0, 1, 2$, uniformly distributed; but it is far away from to be considered random. A sequence to be considered (pseudo) random has to satisfy a set of conditions that are expected from a random sequence.

Generating basic sequences

In programming languages a sequence of random numbers is provided by a function that at each call returns next element of the sequence. These functions are called *random number generator*s. Often they are based on the *linear congruential method* (LCG), proposed in 1948 by D.H. Lehmer.

It is based on the integer sequence

$$s_{n+1} = (a \cdot s_n + b) \bmod m$$

with values on the interval $[0, m-1]$. It is determined by four 'magic numbers': s_0 – *seed* (start value), a – *multiplier*, b – *increment*, m – *modulus*. All four numbers are integers. $s_0, a, b < m$, $a > 1$.

Linear congruential generator

Ussually the sequence (s_n) is not used directly, but the associated sequence (r_n)

$$r_n = \frac{s_n}{m}$$

that is uniformly distributed on the interval $[0, 1)$ (on possible values). There are at most m such values. For very large values of m the sequence (r_n) is nevertheless an acceptable approximation for continuous uniformly distributed sequence.

Because $0 \leq s_n < m$, at most after m steps, a value is repeated. After the first repetition all elements of the sequence are repeating. The number of values that are repeating is called a *period*. A good random number generator has a large period.

Hull-Dobell theorem

First we have to select the modulus m as large as possible – close to the largest integer representable on the computer. Because of this and also because of computational efficiency the moduli of the form 2^k or $2^k \pm 1$ are of special interest. In the best case the period is equal to the modulus.

Hull-Dobell theorem: A period of the sequence (s_n) is equal to m (for all seed values) iff

1. b and m are relatively prime, $b > 0$;
2. $a - 1$ is divisible by all prime factors of m ;
3. $a - 1$ is divisible by 4 if m is divisible by 4.

. . . Hull-Dobell theorem

From condition 2 we see that if m is a product of different prime numbers then $a = 1$.

Suppose that d divides m. Then the sequence (p_n) defined as $p_n = s_n \bmod d$ satisfies the relation

$$p_{n+1} = (a \cdot p_n + b) \bmod d$$

implying that the last places of numbers in the sequence (s_n) are not sufficiently random. For example, in the case $m = 2^k$, the elements of the sequence (s_n) are alternately odd and even.

For a good generator the multiplicator a should be selected in the range

$$\sqrt{m} < a < m - \sqrt{m}$$

and $a > m/100$.

For the increment b Knuth suggests $b \approx \frac{m}{2}(1 - \frac{\sqrt{3}}{3})$.

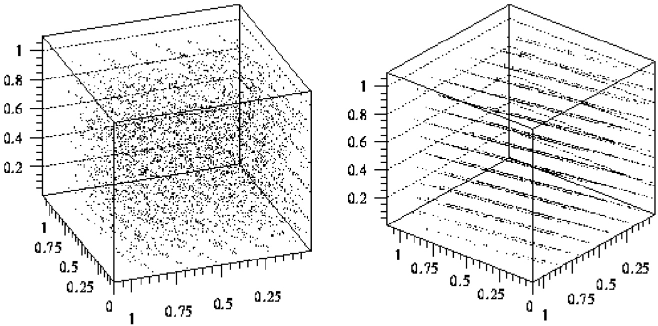
. . . Hull-Dobellov theorem

An important class of LCGs are generators with $b = 0$. For them the value 0 is a fixed point. We have to consider only values on the interval $[1..m - 1]$. If m is a prime number then the generator has a full period $m - 1$.

These conditions do not garantee that we will get a good generator. It has to pass also a list of statistical test that it behaves enough randomly. DieHard tests: 1, 2.

Most programming languages contain a standard function for generating a basic random sequence. In the following we shall use the name *random* for it.

Marsaglia



Marsaglia (1968) showed that the consecutive values of LCG are not a good source of coordinates in multidimensional space – they lie in hyperplanes.

. . . Generators

In rigorous analyses we have to use better basic generators Mersenne Twister, Shift register (R250), Marsaglia-Zaman, GNU.

Because in a LCG the next value depends on the current value s_n it has to 'survive' the call of the function. At the beginning it has to be set to the seed value s_0. For generators with a full period s_0 can be any integer in the interval $[0, m-1]$. Sometimes a special function "randomly" selects s_0 on the basis of some computer state (system clock, etc.). Setting the same seed enables *repeatability* of "random" computations.

Wichmann and Hill's generator

Wichmann and Hill (1982, 1987) showed that we can get a very good basic random generator by combining some basic generators. It holds: if (r_a) in (r_b) are independent basic sequences than also the sequence $(r_a + r_b \bmod 1)$ is basic.

```
WichHill <- function(n){
   output <- numeric(n)
   seed <- get(".WHseed",env=.WH)
   x <- seed[1]; y <- seed[2]; z <- seed[3]
   for(i in 1:n){
      x <- (171*x) %% 30269
      y <- (172*y) %% 30307
      z <- (170*z) %% 30323
      output[i] <- (x/30269 + y/30307 + z/30323) %% 1.0
   }
   assign(".WHseed",c(x,y,z),env=.WH)
   output
}

.WH <- new.env(); assign(".WHseed",1:3,env=.WH)
WichHill(10)
```

Basic generators in R

RNGkind: `"Mersenne-Twister"`, `"Marsaglia-Multicarry"`, `"Super-Duper"`, `"Wichmann-Hill"`, `"Knuth-TAOCP"`, `"Knuth-TAOCP-2002"`, `"user-supplied"`.

Some generators require a large number of seed values (some hundreds). R has a function `set.seed(n)`, $n \in \mathbb{N}$. It makes repeatability much easier:

```
> RNGkind()
[1] "Mersenne-Twister" "Inversion"
> set.seed(2018)
> .Random.seed[1:6]
[1]        403        624  -79855522 -803040953  -27922212 -118944723
> runif(6)
[1] 0.33615347 0.46372327 0.06058539 0.19743361 0.47431419 0.30104860
> .Random.seed[1:6]
[1]        403          6 -881521081  953668654 1212651912  902275211
> set.seed(2018)
> runif(6)
[1] 0.33615347 0.46372327 0.06058539 0.19743361 0.47431419 0.30104860
> RNGkind("Super")
> RNGkind()
[1] "Super-Duper" "Inversion"
> .Random.seed[1:6]
[1]         402  1572731790 -1300846921          NA          NA          NA
```

Uniform and Bernoulli distribution

In our examples in R we shall use as a generator of a basic sequence (r_i) the function

```
random <- function(){return(runif(1,0,1))}
```

Uniformly distributed discrete random sequence (s_i) with values in the set $\{1, 2, 3, \ldots, n\}$ can be obtained from the basic sequence (r_i) using the transformation

$$s = \lfloor r \cdot n \rfloor + 1$$

It maps values from the interval $[\frac{k-1}{n}, \frac{k}{n})$ into number k. Because all the intervals are of the same width and the sequence (r_i) is uniformly distributed so is the sequence (s_i). Note that the intervals are right-open.

```
dice <-
function(n=6){return(1+trunc(n*random()))}
Bernoulli <- function(p){
  if (random()<=p) return(1) else return(0)}
```

Geometric distribution

The probability distribution of the number X of Bernoulli trials needed to get one success.

$$P(X = k) = (1 - p)^{k-1} p, \quad \text{for } k = 1, 2, 3, \ldots$$

The sequence (s_i) determined by relation

$$s = \lfloor \frac{\log(1 - r)}{\log(1 - p)} \rfloor + 1$$

is distributed geometrically with the parameter p. math

```
geometric <- function(p){
  if (p>=1) return(1)
  if (p<=0) return(Inf)
  return(trunc(log(1-random())/log(1-p))+1)
}
```

Tabelaric distribution

A distribution is given in a table $p[i]$, $i = 1, \ldots n$. The corresponding sequence (s_i) is generated with

```
> tabelaR <- function(p){
+   r <- random(); k <- 0;
+   while (r >= 0) {k <- k+1; r <- r - p[k]}
+   return(names(p)[k])
+ }
>
> f <- c(1,2,3,2,1); names(f) <- c("mon","tue","wed","thu
> p <- f/sum(f)
> s <- vector("character",10)
> for (i in 1:10) s[i] <- tabelaR(p)
> s
 [1] "fri" "tue" "wed" "wed" "tue" "wed" "tue" "wed" "tue
```

Improvements:

precompute $q[i] = q[i-1] + p[i], \quad q[0] = 0$;
large tables reorder in decreasing order of probabilities or use binary search;
for distributions such as binomial or Poisson the table is not needed – probabilities can be computed (recursion).

Poisson distribution

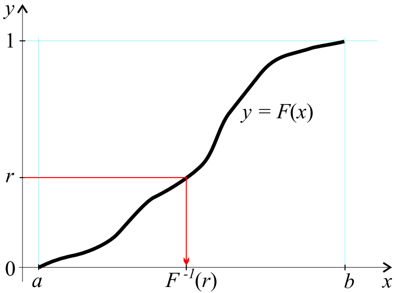
Let λ denote the event rate (number of events in time or space unit). Then $p_x = P(\#\text{events} = x) = e^{-\lambda}\frac{\lambda^x}{x!}$, $x = 0, 1, 2, \ldots$. It holds

$$p_{x+1} = \frac{\lambda}{x+1} p_x, \qquad p_0 = e^{-\lambda}$$

```
PoissonRnd <- function(lambda){
  k <- 0; p <- exp(-lambda); r <- random()-p
  while(r >= 0) {k <- k+1; p <- p*lambda/k; r <- r-p}
  return(k)
}

n <- 10000; s <- numeric(n)
for(i in 1:n) s[i] <- PoissonRnd(0.3)
table(s)
z <- rpois(n,0.3)
table(z)
```

Continuous distributions



Let X be distributed continuously on (a, b) with a density $g(x) > 0$. The corresponding random sequence (s_i) can be obtained as solutions of the equation

$$F(s) = \int_a^s g(x)dx = r$$

for each element of the basic sequence (r_i).

If we are not able to compute the integral analytically we can evaluate it numerically.
The proof is simple:

$$P(F^{-1}(r) \leq x) = P(r \leq F(x)) = F(x)$$

Continuous uniform sequence

Continuous variable uniformly distributed on the interval $[a, b]$ has on this interval the density

$$g(x) = \frac{1}{b-a}$$

Therefore

$$\int_a^s \frac{dt}{b-a} = r \qquad \text{or} \qquad \frac{s-a}{b-a} = r$$

A continuous uniform sequence (s_i) on the interval $[a, b)$ can be obtained using the transformation

$$s = a + r \cdot (b - a)$$

```
uniformRnd <- function(a,b){a+random()*(b-a)}
```

(Negative) exponential distribution

(Negative) exponential distribution has the density $g(x) = \lambda e^{-\lambda x}$, $x \geq 0$ and the cumulative distribution

$$F(x) = \int_0^x \lambda e^{-\lambda t} dt = 1 - e^{-\lambda x}$$

It describes the time between events in a Poisson point process. We get the equation

$$r = F(s) = 1 - e^{-\lambda s}$$

with a solution

$$s = -\frac{1}{\lambda} \ln(1 - r)$$

```
exponRnd <- function(lambda){
  return(-log(1-random())/lambda) }
```

Cauchy distribution

Cauchy distribution has the density

$$g(x) = \frac{a}{\pi} \frac{1}{1 + a^2(x-b)^2}, \qquad -\infty < x < \infty,\ a > 0$$

and the cumulative distribution

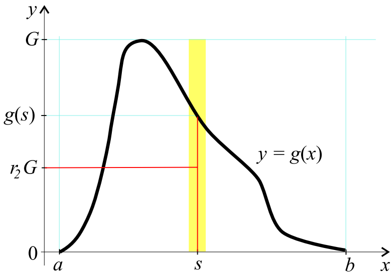
$$F(x) = \frac{a}{\pi} \int_{-\infty}^{x} \frac{1}{1 + a^2(t-b)^2} \mathrm{d}t = \frac{1}{\pi} \operatorname{arc\,tg}(a(x-b)) + \frac{1}{2}$$

It is the canonical example of a "pathological" distribution.

Therefore $s = F^{-1}(r) = \frac{1}{a} \operatorname{tg}(\pi(r - \frac{1}{2})) + b$

```
cauchyRnd <- function(a=1,b=0){
  tan(pi*(random()-0.5))/a + b )}
```

von Neumann rejection method



Let Y be distributed with a density $g(x)$ on a closed interval $[a, b]$ and for all $x \in [a, b]$ it holds $g(x) \leq G$.

Then the corresponding random sequence (s_i) can be generated as follows:

repeat $s := a + random * (b - a)$
until $random * G <= g(s)$;

. . . von Neumann rejection method in R

```
vonNeumann <- function(a,b,G,g,...){
  repeat { s <- a + random()*(b-a)
    if (G*random()<=g(s,...)) return(s)
  }
}

trik <- function(x){if (x>2) return(0); if (x<0) return(0);
  if (x<1) return(x) else return(2-x)
}

vonNeumann(-0.5,2.5,1,trik)
```

. . . von Neumann rejection method / proof

We have to show

$$P(s \in [a', b']/t < g(s)) = \int_{a'}^{b'} g(x)dx, \quad s = a+r_1\cdot(b-a), \quad t = r_2\cdot G$$

Because the points (r_1, r_2) are uniformly distributed on $[0, 1) \times [0, 1)$ also the points (s, t) are uniformly distributed on $[a, b) \times [0, G)$. Since $P(A/B) = P(A \wedge B)/P(B)$ and $\int_q^b g(x)dx = 1$ we have finally

$$P(s \in [a', b']/t < g(s)) = P(s \in [a', b'] \wedge t < g(s))/P(t < g(s)) =$$

$$\frac{\int_{a'}^{b'} g(x)dx}{G(b-a)} : \frac{\int_a^b g(x)dx}{G(b-a)} = \int_{a'}^{b'} g(x)dx$$

Generalized von Neumann rejection method

Assume that we know a distribution $h(x)$ for which the sequence can be generated using the function *hRandom* and that $h(x)$ is close to the distribution $g(x)$ in the sense

$$\forall x \in [a, b] : \frac{g(x)}{h(x)} \leq c$$

Then we can generate the sequence corresponding to $g(x)$ as follows

repeat $s :=$ *hRandom* **until** $c * random * h(s) <= g(s)$;

NA2-1, random

V. Batagelj

Sequences of random numbers

Bertrand paradox

Monte Carlo method

Variance reduction

Monte Carlo in statistics

Resources

Normal or Gaussian distribution

Normaly, with parameters μ and σ, distributed sequence (s_i) can be obtained from a basic sequence (r_i) in different ways.

We can use the property that the sequence (s_i) with elements obtained from k consecutive basic elements as

$$s = \mu + \sigma(\frac{1}{k}\sum_{i=1}^{k} r_i - \frac{1}{2})\sqrt{12k}$$

is with increasing k approaching normal distribution with parameters μ and σ. In many cases we can take $k = 12$ that gives a computationally nice formula

$$s = \mu + \sigma(\sum_{i=1}^{12} r_i - 6)$$

... Normal or Gaussian distribution

Often the approach based on a pair of Box-Muller's transformations is used

$$z_1 = \sqrt{-2 \ln r_1} \sin 2\pi r_2$$

$$z_2 = \sqrt{-2 \ln r_1} \cos 2\pi r_2$$

The sequence (z_i) is normaly distributed with $\mu = 0$ and $\sigma = 1$. To get the sequence (s_i) we apply the transformation $s = \mu + \sigma \cdot z$ on it.

```
gaussRnd <- function(st=1,m=0,s=1){
  if (st==0){new <- TRUE; .Gauss <<- list(m=m,s=s,n=new,r=0)}
  else {m <- .Gauss$m; s <- .Gauss$s; new <- .Gauss$n }
  if (new) {p <- sqrt(-2*log(random())); q <- 2*pi*random()
    x <- p*sin(q); .Gauss$r <<- p*cos(q)
  } else  x <- .Gauss$r
  .Gauss$n <<- !new
  return(m+s*x)
}
```

Multidimensional normal distribution

Let **R** be a *m*-dimensional correlation matrix (symmetric and positive definite). It can be written in the form (Cholesky) $\mathbf{R} = \mathbf{T}\mathbf{T}^T$, where **T** is a lower triangular matrix. Let

$$x = [s_1, s_2, \ldots, s_m]$$

be a random vector in which each s_i is from a sequence with the standard normal distribution. Then the sequence of vectors (y)

$$y = \mathbf{T}x$$

has the *m*-dimensional normal distribution

$$\varphi(x) = \frac{1}{\sqrt{(2\pi)^m |\mathbf{R}|}} e^{-\frac{1}{2}x^T\mathbf{R}^{-1}x}$$

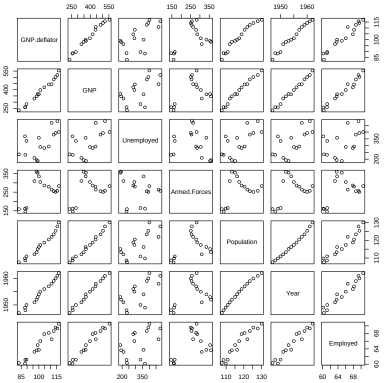
...Multidimensional normal distribution

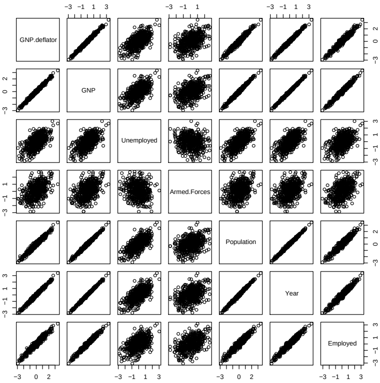
```
multinormal <- function(T){return(t(t(T)%*%rnorm(dim(T)[1])))}

data(longley); R <- cor(longley); T <- chol(R); pairs(longley)
n <- 2000; s <- NULL;
for (i in 1:n) s <- rbind(s,multinormal(T))
pairs(s)
```

```
> (C <- cor(s))
             GNP.deflator       GNP Unemployed Armed.Forces Population      Year  Employed
GNP.deflator    1.0000000 0.9918324  0.6301733    0.4616242  0.9799199 0.9914436 0.9710698
GNP             0.9918324 1.0000000  0.6144270    0.4413537  0.9914626 0.9953361 0.9836551
Unemployed      0.6301733 0.6144270  1.0000000   -0.1718522  0.6945332 0.6770740 0.5123590
Armed.Forces    0.4616242 0.4413537 -0.1718522    1.0000000  0.3601524 0.4129422 0.4556520
Population      0.9799199 0.9914626  0.6945332    0.3601524  1.0000000 0.9941552 0.9610493
Year            0.9914436 0.9953361  0.6770740    0.4129422  0.9941552 1.0000000 0.9715907
Employed        0.9710698 0.9836551  0.5123590    0.4556520  0.9610493 0.9715907 1.0000000
> R
             GNP.deflator       GNP Unemployed Armed.Forces Population      Year  Employed
GNP.deflator    1.0000000 0.9915892  0.6206334    0.4647442  0.9791634 0.9911492 0.9708985
GNP             0.9915892 1.0000000  0.6042609    0.4464368  0.9910901 0.9952735 0.9835516
Unemployed      0.6206334 0.6042609  1.0000000   -0.1774206  0.6865515 0.6682566 0.5024981
Armed.Forces    0.4647442 0.4464368 -0.1774206    1.0000000  0.3644163 0.4172451 0.4573074
Population      0.9791634 0.9910901  0.6865515    0.3644163  1.0000000 0.9939528 0.9603906
Year            0.9911492 0.9952735  0.6682566    0.4172451  0.9939528 1.0000000 0.9713295
Employed        0.9708985 0.9835516  0.5024981    0.4573074  0.9603906 0.9713295 1.0000000
>
```

. . . Multidimensional normal distribution





Random direction

In the plane we select a random angle (in the polar coordinate system)

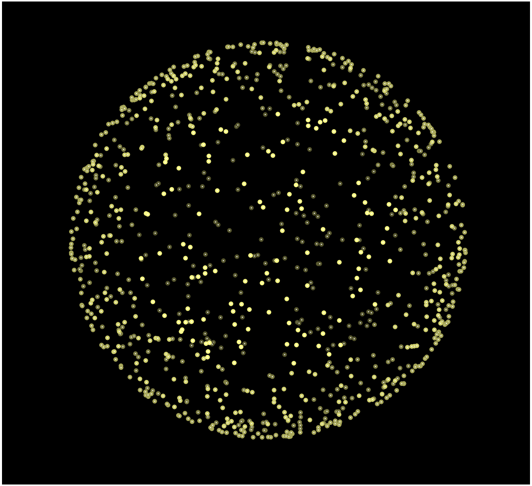
$$\varphi = 2\pi r$$

In the 3D space we express the direction in the spherical coordinate system (φ, ψ) W

$$\varphi = 2\pi r_1 \qquad \psi = \arccos(1 - 2r_2)$$

```
dir3D<-function(){return(c(2*pi*random(),acos(1-2*random())))}
vector3D <- function(dir){
  return( c( sin(dir[2])*cos(dir[1]),
    sin(dir[2])*sin(dir[1]), cos(dir[2]) ))
}
points3D <- function(n){
  cat(file="obla.net",c("*vertices ",n,"\n"))
  for (i in 1:n) {
    cat(file="obla.net",i,i,vector3D(dir3D()),"\n",append=TRUE)
  }
}
```

Random direction / picture



Random multidimensional direction

A m-dimensional random unit vector

$$d = [s_1, s_2, \ldots, s_m]/s$$

where s_i have standard normal distribution and $s = \sqrt{s_1^2 + s_2^2 + \ldots + s_m^2}$ is uniformly distributed on the surface of the m-dimensional sphere.

```
direction <- function(m=3){
  x <- rnorm(m); return(x/sqrt(sum(x^2)))}
```

Levy flights

A vector x uniformly distributed on the surface of the m-dimensional sphere can be, using $y := \sqrt[m]{random} * x$, transformed in a vector y uniformly distributed inside the m-dimensional sphere.

Uniform distribution in multidimensional ellipsoid

Let

$$E = \{x : x^T \mathbf{R}^{-1} x \leq 1\}$$

be an ellipsoid in central position determined by a square positive definite matrix $\mathbf{R}$. The matrix $\mathbf{R}$ can be written in the form (Cholesky) $\mathbf{R} = \mathbf{T}\mathbf{T}^T$ where $\mathbf{T}$ is a lower triangular matrix.
The ellipsoid E can be obtained from the sphere $S = \{x : x^T x \leq 1\}$ with $y = \mathbf{T}x$ – as it holds

$$1 \geq x^T x = (\mathbf{T}^{-1} y)^T (\mathbf{T}^{-1} y) = y^T \mathbf{R}^{-1} y$$

Because the linear transformation $\mathbf{T}$ preserves uniformity of distribution we can get uniformly distributed points inside an ellipsoid as follows: we first generate a random vector x uniformly distributed inside the sphere S and afterward transform it into an ellipsoid point using $y = \mathbf{T}x$.

. . . Uniform distribution in multidimensional ellipsoid in R


```
elipsoid <- function(T){
  m <- dim(T)[1]; r <- random()^(1/m)
  x <- rnorm(m); y <- x*r/sqrt(sum(x^2))
  return(t(t(T) %*% y))
}

R <-
  c( 4.0000000, 1.1395235, 1.775876, 0.7753723,
     1.1395235, 4.0000000, 1.065415, 0.7881692,
     1.7758761, 1.0654147, 4.000000, 1.5841046,
     0.7753723, 0.7881692, 1.584105, 4.0000000 )
dim(R) <- c(4,4); T <- chol(R)
s <- NULL; for (i in 1:2000) s <- rbind(s,elipsoid(T))
pairs(s)

R <- matrix(0,4,4); diag(R) <- c(1,2,3,4); T <- chol(R)
```

Random permutation (shuffling)

Assume that the table $x[i]$, $i = 1, \ldots, n$ contains a permutation of integers from 1 to n. A fair (uniformly distributed) shuffling can be done with the following procedure

```
shuffle <- function(x){
  n <- length(x)
  for ( i in n:2 ){j <- dice(i)
    t <- x[i]; x[i] <- x[j]; x[j] <- t }
  return(x)
}
```

Random sample

We would like to select randomly (fairly) m units from a set of n units, $m \leq n$. Such a selection is called an (unbiased) *sample*.

An option is to repat throwing a fair n-dice until we get m different units. This requires an expensive “bookkeeping”.

Another option is to select each unit with the probability $\frac{m}{n}$. The expected number of selected units is m, but with a standard variation $\sqrt{m(1 - \frac{m}{n})}$. This idea can be improved.

To get an unbiased random sample we proceed as follows:

. . . Random sample

We are selecting units sequentially $i = 1, 2, \ldots, n$. Suppose that in i steps we selected k units. Then the unit $i + 1$ has to be selected with the probabilty

$$\binom{n-i-1}{m-k-1} / \binom{n-i}{m-k} = \frac{m-k}{n-i}$$

```
sampleN <- function(n,m){
  k <- 0; s <- integer(m)
  for(i in 0:n) if ((n-i)*random() < m-k) {
    k <- k+1; s[k] <- i
    if (k==m) return(s)
  }
}
```

What if the length of the sequence is not known? In this case we assign to each unit as a weight a basic random number. We save the unit in a storage (heap) with capacity of m units in which we keep units with the largest weights – *sieving*.

Distributions in R

Most of the standard distributions is already available in R. For a distribution *dist* we have four functions:

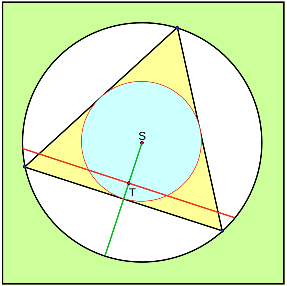
- `d`*dist* – density/mass function,
- `p`*dist* – cumulative distribution,
- `q`*dist* – inverse – quantile function,
- `r`*dist* – random sequence generation.

where *dist* can be: `unif`, `beta`, `binom`, `cauchy`, `exp`, `chisq`, `f`, `gamma`, `geom`, `hyper`, `lnorm`, `logis`, `nbinom`, `norm`, `pois`, `signrank`, `t`, `weibull`, `wilcox`.

See also the function `sample`.

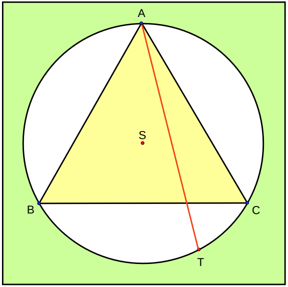
Bertrand paradox

Consider an equilateral triangle inscribed in a circle with radius r. Suppose a chord of the circle is chosen at random. What is the probability that the chord is longer than a side of the triangle?



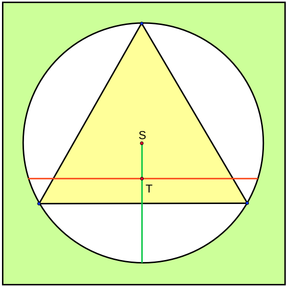
a. The **random midpoint** method: Choose a point T anywhere within the circle and construct a chord with the chosen point T as its midpoint. The chord is longer than a side of the inscribed triangle if the chosen point falls within a concentric circle of radius $\frac{r}{2}$. The area of the smaller circle is one fourth the area of the larger circle, therefore $p_a = \frac{1}{4} = 0.25$.

Bertrand paradox / b



b. The **random endpoints** method: Choose two random points on the circumference of the circle and draw the chord joining them. Imagine the triangle rotated so that its vertex A coincides with one of the chord endpoints. Observe that if the other chord endpoint T lies on the arc BC opposite to the point A, the chord is longer than a side of the triangle. The length of the arc is one third of the circumference of the circle, therefore $p_b = \frac{1}{3} = 0.3333$.

Bertrand paradox / c



c. The **random radius** method: Choose a radius of the circle, choose a point T on the radius and construct the chord through this point and perpendicular to the radius. Imagine the triangle rotated so that a side is perpendicular to the radius. The chord is longer than a side of the triangle if the chosen point is nearer the center of the circle than the point where the side of the triangle intersects the radius. The side of the triangle bisects the radius, therefore $p_c = \frac{1}{2} = 0.5$.

Monte Carlo method

Monte Carlo (MC) methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.

The idea to engage randomness in solving problems is an old one. In 18th century a french scientist Buffon proposed a method to estimate the value of π by tossing a needle on a paper with lines. First practical application of MC was during the WW II. In 1943 J. von Neumann and S. Ulam with collaborators from Los Alamos Scientific Laboratories applied it for solving some problems in construction of atomic bomb (project Manhattan) that were not able to solve analytically. This way it left the “lumber room” of recreational mathematics.

The expansion of the use of MC highly depends on development of computers that are providing necessary computing resources – MC methods are computationaly highly intensive methods.

. . . Monte Carlo method

Monte Carlo simulation is a technique used to study how a model responds to randomly generated inputs. It typically involves a three-step process:

1. Randomly generate *n* inputs (sometimes called scenarios).
2. Run a simulation for each of the *n* inputs. Simulations are run on a computerized model of the system being analyzed.
3. Aggregate and assess the outputs from the simulations. Common measures include the mean value of an output, the distribution of output values, and the minimum or maximum output value.

... Monte Carlo method

A theoretical basis of MC methods comes from probability theory.

1. Kolmogorov's **strong law of large numbers** (SLLN): Suppose $X_1, X_2, X_3, \ldots$ are independent and identically distributed and $\mathrm{E}X$ exists. Then, $\sum_{k=1}^{n} x_k/n \to \mathrm{E}X$, almost surely.

   Conversely, if $\sum_{k=1}^{n} x_k/n \to \mu$ which is finite, then $\mu = \mathrm{E}X$.

2. **Central limit theorem** (CLT)

$$P\left(\frac{1}{n}\sum_{k=1}^{n}(x_k - \mathrm{E}X) < \frac{t\mathrm{D}X}{\sqrt{n}}\right) \xrightarrow{n\to\infty} \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{t} e^{-\frac{z^2}{2}}\,dz$$

Square root law

CLT enables us to estimate the error: with a given probability p

$$|\mathrm{E}X - \frac{1}{n}\sum_{k=1}^{n} x_k| \leq \frac{\gamma \mathrm{D}X}{\sqrt{n}}$$

where γ depends on p.

Therefore

$$|\mathrm{E}X - \overline{X}| \leq \frac{c}{\sqrt{n}}$$

where c is a constant that depends on the problem and p. This relation is also called the (inverse) *square root law*.

Implications of the Square root law

From the square root law we see that with increasing number of repetitions n the estimate $\mu(n) = \overline{X}$ is approaching to μ. Therefore, **MC works**! But, how fast?

How many repetitions are needed to improve the result for one decimal place? Let us denote with ε the error and with n the number of repetitions for the current result. Assume that for the improved result N repetitions are needed. By the square root law we get:

$$\varepsilon \approx \frac{c}{\sqrt{n}} \qquad \text{and} \qquad \frac{\varepsilon}{10} \approx \frac{c}{\sqrt{N}} \qquad \text{implying} \qquad N \approx 100n$$

For each new decimal place of precision we have to invest 100 times more work.

The square root law is a foundation of MC but also its main constraint that limits the precision of results.

MC methods are very powerful tool. Often they provide satisfactory estimates for problems for which no other (analytical) methods are known.

Number of repetitions

Let $\theta = EX$ be the parameter of our interest and $\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} x_i$ its estimator. Then

$$P(|\hat{\theta} - \theta| \leq \lambda \mathrm{D}\hat{\theta}) = \frac{1}{\sqrt{2\pi}\mathrm{D}\hat{\theta}} \int_{\theta-\lambda\mathrm{D}\hat{\theta}}^{\theta+\lambda\mathrm{D}\hat{\theta}} e^{-\frac{(x-\theta)^2}{2\mathrm{D}^2\hat{\theta}}}\,\mathrm{d}x =$$

$$\frac{1}{\sqrt{2\pi}} \int_{-\lambda}^{\lambda} e^{-\frac{t^2}{2}}\,\mathrm{d}t = 2\Phi(\lambda) - 1$$

where $2\Phi(2) - 1 \approx 0.95$ and $2\Phi(3) - 1 \approx 0.997$.

Let $\sigma = \mathrm{D}X$. Then $\mathrm{Var}\hat{\theta} = \frac{1}{n^2}\mathrm{Var}\sum_{i=1}^{n} x_i = \frac{\sigma^2}{n}$ or $\mathrm{D}\hat{\theta} = \frac{\sigma}{\sqrt{n}}$.

Assume that we allow the error δ. Then

$$\delta = \lambda\mathrm{D}\hat{\theta} = \frac{\lambda\sigma}{\sqrt{n}} \quad \Longrightarrow n = \left(\frac{\lambda\sigma}{\delta}\right)^2$$

The value of σ can be estimated from a small sample.

MC estimation of π

As an example let us describe an application of MC method for estimating the area of a given shape P in the plane.

We include the shape P in a simpler shape Q for which we know how to generate uniformly distributed sequence of random points $(x, y) \in Q$. The probability $p(P/Q)$ that a point from the sequence belongs also to the shape P is equal to the quotient of the area of P, $a(P)$, and the area of Q, $a(Q)$. By SLLN the sequence $n(P)/n(Q)$ ($n(P)$ is the number of points belonging to shape P among the first $n(Q)$ points) tends to $p(P/Q)$. Therefore

$$\frac{a(P)}{a(Q)} \approx \frac{n(P)}{n(Q)} \qquad \text{or} \qquad a(P) \approx \frac{n(P) \cdot a(Q)}{n(Q)}$$

To estimate the area of the shape P using MC we need also to be able to efficiently test the membership of a point (x, y) to shape P. For the shape Q we usually take a rectangle around the shape P.

... Number π

NA2-1, random

V. Batagelj

Sequences of random numbers

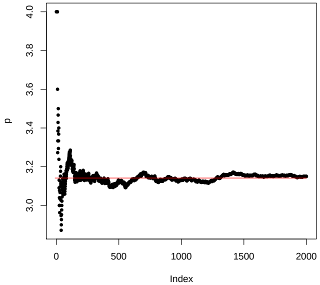
Bertrand paradox

Monte Carlo method

Variance reduction

Monte Carlo in statistics

Resources



Let us estimate the value of $\pi = 3.1416\ldots$ that is equal to 4 times the area of unit circle in the first quadrant – shape P. For a shape Q we select the square $[0,1] \times [0,1]$ with $a(Q) = 1$. A point $(x,y) \in Q$ belongs to P iff $x^2 + y^2 \leq 1$.

```
k <- 0; n <- 2000; u <- integer(n)
for(i in 1:n){
  if (random()^2+random()^2 < 1) k <- k+1;
  u[i] <- k }
p <- u/seq(n)*4
plot(p,pch=20)
lines(c(-10,n),c(pi,pi),col="red")
```

Estimating values of integrals

Let $p(x)$ be a density of distribution on the set A ($\int_A p(x)dA = 1$; $p(x) \geq 0, \forall x \in A$). To estimate the value of the integral

$$\theta = \int_A f(x)p(x)dA$$

we generate a random sequence $(s_i)_{i=1}^m$ from A distributed as $p(x)$ and compute the sample average

$$\hat{\theta} = \frac{1}{m}\sum_{i=1}^{m} f(s_i)$$

By SLLN the value of $\hat{\theta}$ with $m \to \infty$ tends to $\mathrm{E}\hat{\theta} = \theta$ and by CLT the distribution

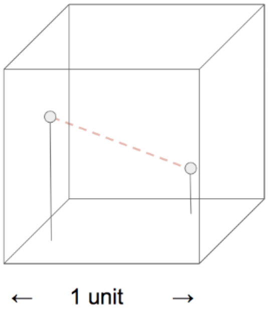
$$\frac{\hat{\theta} - \mathrm{E}\hat{\theta}}{\sqrt{\mathrm{Var}\hat{\theta}}}$$

tends to the standard normal distribution $N(0, 1)$.

This approach is used primarly for estimating values of multidimensional integrals for which the traditional numerical methods are too slow.

Estimating values of integrals

Example



Randomly pick two points inside a unit cube. What is the expected distance between them?

Requires to compute

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2} dx_1 dx_2 dx_3 dy_1 dy_2 dy_3$$

freeCodeCamp; R (answer 0.6617)

Sandokan

Let us apply the MC in answering the following question:

Every chocolate contains one trade card. To fill the album we need to collect n different trade cards. How many chocolates we have to buy to fill the album?

The idea of the MC procedure is simple: in each step we are randomly bying trade cards until the album is full and output the number of trade cards bought. The average of outputs is the answer.

Sandokan MP3, Sandokan, pictures

Sandokan – theoretical solution

To check the result obtained with MC we will compare it to the theoretical solution. We get it as follows:

expected number of cards bought to get a new card = 1 / probability to get a new card

Suppose that m cards are still missing in the album.Then

$$\textit{probability to get a new card} = \frac{m}{n}$$

Let $N(n, m)$ denote the expected number of cards to be bought to fill the album if m cards are missing. We get the following recursive relation

$$N(n, m) = \frac{n}{m} + N(n, m - 1) \qquad \text{and} \qquad N(n, 0) = 0$$

with a solution

$$N(n, m) = n.(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{m})$$

. . . Sandokan – theoretical solution

The sum of reciprocals of the first m integers is called m-th *harmonic number* and is denoted with H_m. Therefore

$$N(n, m) = nH_m$$

For a small value of m we compute H_m by definition. For larger values of m we get very precise value using the formula

$$H_m \approx \ln(m + \frac{1}{2}) + \gamma$$

where $\gamma = 0.57721\,56649$ is the Euler's constant.

The expected number of cards bought to collect m different cards is:

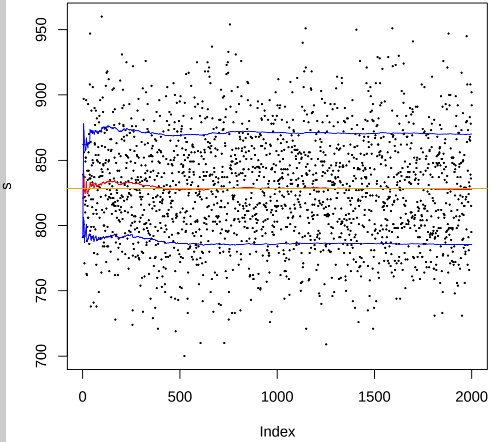
$$N(n, n) - N(n, n - m) = n(H_n - H_{n-m})$$

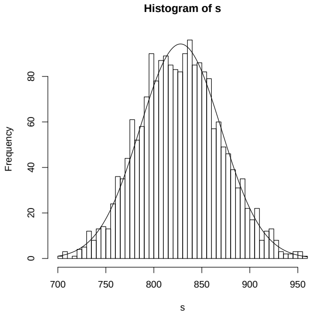
. . . Sandokan in R

```
sandokan <- function(n,m=-1,r=100){
  if (m<0) m <- n
  s <- integer(r)
  for ( i in 1:r ){
    album <- rep(TRUE,n)
    different <- 0; cards <-0
    while (different < m) {
      k <- dice(n); cards <- cards + 1
      if (album[k]) {
        album[k] <- FALSE; different <- different + 1
      }
    }
    s[i] <- cards
  }
  return(s)
}

sandokanT <- function(n,m=-1){
  if (m<0) m <- n
  return(n*sum(1/((n-m+1):n)))
}
```

...Sandokan / picture





Importance sampling

Variance reduction is a procedure used to increase the precision of the estimates.

Let us reconsider the estimation of the integral

$$I = \int_0^1 f(x) \mathrm{d}x, \qquad f(x) \geq 0$$

In *importance sampling* all the values are not selected uniformly, but larger values have priority. This should be accounted in weights. Let $g(x)$ be a density on $[0, 1]$, such that $f(x) > 0 \Rightarrow g(x) > 0$, and the random sequence (s_i) is distributed as $g(x)$. Then

$$I = \int_0^1 \frac{f(x)}{g(x)} g(x) \mathrm{d}x = \int_0^1 \frac{f(x)}{g(x)} \mathrm{d}G(x)$$

where $G(x) = \int_0^x g(t) \mathrm{d}t$ is a cumulative distribution of $g(x)$.

... Importance sampling

For the estimator

$$\hat{I}_P = \frac{1}{n}\sum_{i=1}^{n}\frac{f(s_i)}{g(s_i)}$$

we have $\quad \mathrm{E}\hat{I}_P = I$ and

$$\mathrm{Var}\hat{I}_P = \frac{1}{n}\left(\int_0^1 \frac{f^2(x)}{g^2(x)}\mathrm{d}x - I^2\right)$$

The variance is minimal ($= 0$) for $f(x) = cg(x)$ where $c = \frac{1}{I}$ – knowing this konstant we know also the value of the integral $I = \frac{1}{c}$.

Control variate

We would like to estimate the parameter $\theta = \mathrm{E}X$. Suppose that we have another *control* variate $\mathrm{E}Y = \mu$. Then for every value of the constant c also the expression $X + c(Y - \mu)$ is an unbiased estimator of θ. Consider its variance

$$\mathrm{Var}(X + c(Y - \mu)) = \mathrm{Var}(X + cY) = \mathrm{Var}X + c^2 \mathrm{Var}Y + 2c\mathrm{Cov}(X, Y)$$

Because $\mathrm{Var}Y > 0$ it has a minimum for $c^* = -\frac{\mathrm{Cov}(X,Y)}{\mathrm{Var}Y}$. The variance of the corresponding estimator is

$$\mathrm{Var}(X + c^*(Y - \mu)) = \mathrm{Var}X - \frac{\mathrm{Cov}(X, Y)^2}{\mathrm{Var}Y} = \mathrm{Var}X(1 - \mathrm{Corr}(X, Y)^2)$$

We get large reduction of variance when X and Y are well correlated.

...Control variate

In some cases $\operatorname{Var} Y$ is known; otherwise we estimate it from a sample

$$\widehat{\operatorname{Var}} Y = \frac{1}{m-1} \sum_{i=1}^{m} (y_i - \overline{Y})^2$$

and the covariance $\operatorname{Cov}(X, Y)$ with

$$\widehat{\operatorname{Cov}}(X, Y) = \frac{1}{m} \sum_{i=1}^{m} (x_i - \overline{X})(y_i - \overline{Y})$$

This gives us a good approximation $\hat{c} = -\frac{\widehat{\operatorname{Cov}}(X, Y)}{\widehat{\operatorname{Var}} Y}$ for c^*.

Conditioning

Conditioning is based on the following property of variance

$$\mathrm{Var} X = \mathrm{E}\,\mathrm{Var}(X|\,Y) + \mathrm{Var}\,\mathrm{E}(X|\,Y)$$

Since both terms on the right are nonnegative we have $\mathrm{Var} X \geq \mathrm{Var}\,\mathrm{E}(X|\,Y)$.

With MC we are estimating the parameter $\theta = \mathrm{E}X$. Suppose that we create another variate Y for which we know $\mathrm{E}(X|\,Y)$. Because $\mathrm{E}\,\mathrm{E}(X|\,Y) = \mathrm{E}X = \theta$ also $\mathrm{E}(X|\,Y)$ is an unbiased estimator for θ with smaller variance.

Monte Carlo in statistics

Suppose that we have a sample with n values $\{x_i\}$ distributed as F. We would like to estimate the value of parameter θ. We take as its estimate the value $\theta_{\hat{F}}$ obtained from the sample distribution $\hat{F}$. $\hat{\theta}$ is the estimator of parameter θ.

$$\theta = \mathrm{E}X, \qquad \hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\theta = \mathrm{Var}X, \qquad \hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} (x_i - \overline{x})^2$$

$$\theta = F(c) = P(X \le c), \qquad \hat{\theta} = \frac{1}{n}\mathrm{card}\{i : x_i \le c\}$$

How well $\hat{\theta}$ estimates the parameter θ ? We use measures

$$\mathrm{Bias}_\theta\hat{\theta} = \mathrm{E}_F\hat{\theta} - \theta, \quad \mathrm{se}_\theta\hat{\theta} = \sqrt{\mathrm{Var}\hat{\theta}}, \quad \mathrm{MSE}_\theta\hat{\theta} = \mathrm{E}_F(\hat{\theta} - \theta)^2$$

linked with eqality $\mathrm{MSE}_\theta\hat{\theta} = \mathrm{Var}\hat{\theta} + (\mathrm{Bias}_\theta\hat{\theta})^2$.

. . . Monte Carlo in statistics

For some distributions F these measures can be expressed analitically. Otherwise we can use the computer:

- generate a sequence of samples $\{x_i\}$ distributed as F and for each of them compute the estimate $\hat{\theta}$
- average and variance of the obtained estimates $\hat{\theta}$ are good estimates for $\mathrm{E}_F\hat{\theta}$ and $\mathrm{Var}_F\hat{\theta}$.

. . . Monte Carlo in statistics / Example

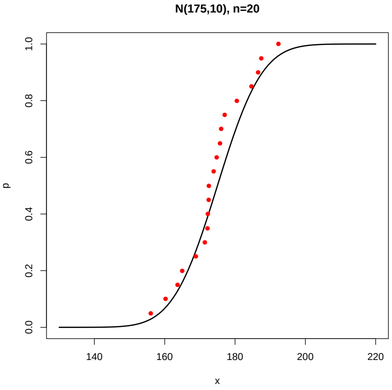
Variates X_1 and X_2 have standard normal distribution. Estimate the expected value of their absolute difference $\theta = \mathrm{E}|X_1 - X_2|$.

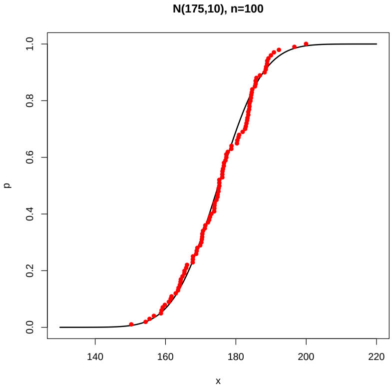
```
> N <- 10000
> x1 <- rnorm(N); x2 <- rnorm(N); y <- abs(x1-x2)
> print(theta.hat <- mean(y))
[1] 1.127599
> print(se.theta <- sd(y)/sqrt(N))
[1] 0.008472097
> print(theta <- 2/sqrt(pi))
[1] 1.128379
> print(se <- sqrt((2-4/pi)/N))
[1] 0.008525025
```

Let $Y = |X_1 - X_2|$. The theoretical answers are $\theta = \mathrm{E}\,Y = 2/\sqrt{\pi}$ and $\mathrm{Var}\,Y = \sqrt{2 - 4/\pi}$. Therefore $se = \mathrm{Var}\,Y/\sqrt{N} = \sqrt{(2-4/\pi)/N}$, where N is size of the sample.

Resampling

Approximating a distribution F by its sample distribution $\hat{F}$





Resampling

What if the distribution F is not known? Assuming that the given sample $\{x_i\}$ represents well the distribution F we can replace it with the sample distribution $\hat{F}$ – to each value x_i we assign the probability $\frac{1}{n}$, n is the sample size.

Let θ^* be the estimate for θ obtained from $\hat{F}$. Because of 'closeness' of distributions F and $\hat{F}$ we expect that $\mathrm{Bias}_\theta\hat{\theta} \approx \mathrm{Bias}_{\hat{\theta}}\theta^*$ and $\mathrm{Var}_F\hat{\theta} \approx \mathrm{Var}_{\hat{F}}\theta^*$.

This is the basis of *resampling* (bootstrap) approach.

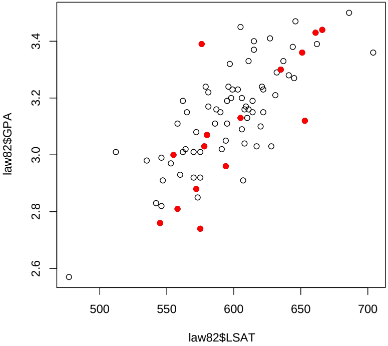
Resampling / procedure

1. using sampling with replacement create from the given sample $\{x_i\}$ N random samples; and for each of them compute the estimate θ^*_k, $k = 1, \ldots, N$.
2. compute the estimate $\hat{\theta}$ from the given sample $\{x_i\}$.
3. $\overline{\theta^*} = \frac{1}{N}\sum_{k=1}^{N}\theta^*_k, \quad \widehat{\text{Bias}}_{\hat{\theta}}\theta^* = \overline{\theta^*} - \hat{\theta}$
   $\widehat{\text{Var}}\theta^* = \frac{1}{N-1}\sum_{k=1}^{N}(\theta^*_k - \overline{\theta^*})^2, \quad \widehat{\text{se}}_{\hat{F}}\theta^* = \sqrt{\widehat{\text{Var}}\theta^*}$

In R two resampling libraries are available: `bootstrap` (Efron in Tibshirani) and `boot` (Davison in Hinkley).

Resampling / Example

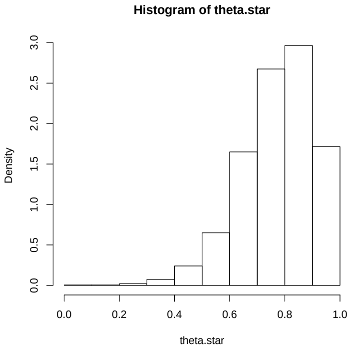
Library `bootstrap` contains also a data set `law82` (Efron, Tibshirani, 1993) about 82 law schools in US: two variables `LSAT` – average score on a national law test; `GPA` – average undergraduate grade-point average. From it a random sample of 15 schools was created and stored in data set `law` – red dots on the picture. The correlation between variables on the sample is 0.776, and for the complete data set 0.760.



Using resampling we estimate the precision of this result. For standard error we get 0.131 and for bias -0.005.

Resampling / Example in R


```
> library(bootstrap)
> plot(law82$LSAT,law82$GPA)
> points(law$LSAT,law$GPA,pch=16,col='Red')
> print(cor(law82$LSAT,law82$GPA))
[1] 0.7599979
> print(theta.hat <- cor(law$LSAT,law$GPA))
[1] 0.7763745
> N <- 2000
> n <- nrow(law)
> theta.star <- numeric(N)
> for(k in 1:N){
+   i <- sample(1:n,size=n,replace=TRUE)
+   L <- law$LSAT[i];G <- law$GPA[i]
+   theta.star[k] <- cor(L,G)
+ }
> hist(theta.star,prob=TRUE)
> print(theta.star.mean <- mean(theta.star))
[1] 0.7714522
> print(bias <- theta.star.mean - theta.hat)
[1] -0.004922274
> print(se.theta.star <- sd(theta.star))
[1] 0.1307119
```



How many samples?

Similarly as for Monte Carlo

$$P(|\widehat{\text{Bias}}_{\hat{\theta}}\theta^* - \text{Bias}_{\hat{\theta}}\theta^*| < \frac{\lambda \widehat{\text{se}}_{\hat{F}}\theta^*}{\sqrt{N}}) = 2\Phi(\lambda) - 1$$

If we want a precision δ (for example 0.001) we need N samples

$$N = (\frac{\lambda \widehat{\text{se}}_{\hat{F}}\theta^*}{\delta})^2$$

Resources I

NA2-1, random

V. Batagelj

- Bruce, Peter C. (2015): Introductory Statistics and Analytics: A Resampling Perspective. Wiley.
- Deák, István (1990): Random number generator and simulation, Akadémiai Kiadó, Budapest.
- Ermakov, S.M. (1971): Metod Monte-Karlo i smežnye voprosy, Nauka, Moskva.
- Good, Phillip I. (2005): Permutation, Parametric and Bootstrap Tests of Hypotheses: A Practical Guide to Resampling Methods for Testing Hypotheses. Springer Series in Statistics. Springer.
- Good, Phillip I. (2013): Introduction to Statistics Through Resampling Methods and R. Wiley.
- Knuth, Donald Ervin (1997): Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison-Wesley, 3rd edition.

Resources II

- Marsaglia, George (1968). Random Numbers Fall Mainly in the Planes. Proceedings of the National Academy of Science (USA) 61 (September): 25-28.
- Ross, Sheldon M. (1990): A Course in Simulation, Prentice Hall.
- Székely, Gábor J. (1986): Paradoxes in probability theory and mathematical statistics, Akadémiai Kiadó, Budapest.
- Pierre L'Ecuyer's publications
- Eudaemons