CONTS

Random_distributions.mth: Random samples from distributions with DERIVE

Galán García, José Luis jl_galan@uma.es
Aguilera Venegas, Gabriel gabri@ctima.uma.es
Padilla Domínguez, Yolanda ypadilla@ctima.uma.es
Rodríguez Cielos, Pedro prodriguez@uma.es

Department of Applied Mathematic
University of Málaga (Spain)

Abstract

This paper establishes the theoretical aspects which have been considered in order to elaborate the `Random_distiributions` package for DERIVE 6 as well as the description of the different algorithm developed in the package. In section 1 the theory on random number generation is presented (from [Rubinstein, 1981]). After explaining DERIVE's random function (section 1.1) the more efficient algorithms `ran2` and `mzran13` are developed (section 1.2 and 1.3 respectively). Section 2 presents three different general methods for generating continuous distributions together with one for generating discrete distributions. Section 3 is dedicated to describe different algorithms for generating random values from continuous distributions (Uniform, Exponential, Normal, Lognormal, Weibul, Gamma, Beta, Chi-square, Student's t, F, Z, Pareto, Logistic and Cauchy distributions). Section 4 presents different algorithms for generating discrete distributions (Uniform discrete, Bernouille, Binomial, Poisson, Geometric, Negative Binomial and Hypergeometric). Finally, in section 5 some algorithms for generating different distributions using other distributions as approximations are developed.

1 Random Number Generation

The most commonly used methods for generating pseudorandom numbers are *congruential generators.* A congruential method is one that produces a nonrandom sequence of numbers according to some recursive formula based on calculating the residues modulo of some integer m of a linear transformation. It is readily seen from this definition that each term of the sequence is available in advance, before the sequence is actually generated. Although these processes are completely deterministic, it can be shown that the numbers generated by the sequence appear to be uniformly distributed and statistically independent.

Congruential methods are based on a fundamental congruence relationship, which may be expressed as:

$$X_{i+1} = a\,X_i + c \pmod{m} \qquad i = 0, 1, 2, \ldots, n \tag{1}$$

where the *multiplier* a, the *increment* c and the *modulus* m are nonnegative integers.

Given an initial starting value X_0 (called the *seed*), (1) yields a congruence relationship (modulo m) for any value i of the sequence $\{X_i\}$. Generators that produce random numbers according

to (1) are called *mixed congruential generators.* The random numbers on the unit interval $(0,1)$ can be obtained by:

$$U_i = \frac{X_i}{m} \tag{2}$$

CONTS

Clearly, such a sequence will repeat itself in at most m steps, and will therefore be periodic.

As $X_i < m$ for all i, the period of the generator cannot exceed m, that is, the sequence X_i contains at most m different numbers. Because of the deterministic character of the sequence, the entire sequence recurs as soon as any number is repeated. The sequence is said to *get into a loop,* that is, there is a cycle of numbers that is repeated endlessly. Modulus m should be chosen as large as possible and appropriated values of a and c in order to make the *period* p maximum (that is, $p = m$) must be found. When this happens the random number generator has a *full period.* It can be shown that the generator defined in (1) has a full period m, if and only if:

1. c is *relative prime* to m, that is, c and m have no common divisor.
2. $a \equiv 1 \pmod{g}$ for every prime factor g of m.
3. $a \equiv 1 \pmod{4}$ if m is a multiple of 4.

Since most computers utilize either a binary or a decimal digit system, the best selection for m is $m = 2^\beta$ or $m = 10^\beta$, respectively where β is the word–length of the particular binary or decimal computer.

For binary computers, in order to develop a full period generator when $m = 2^\beta$, the parameter c must be odd and $a = 4k + 1$ for some $k \in \mathbb{N}$.

The second widely used generator is the *multiplicative generator:*

$$X_{i+1} = a\, X_i \pmod{m} \qquad i = 0, 1, 2, \ldots, n \tag{3}$$

which is a particular case of the mixed generator (1) with $c = 0$.

Another common type of generator in which X_{i+1} depends on more than one of the preceding values[1]. For example:

$$\begin{aligned} X_{i+1} &= a_1\, X_{i-j_1} + a_2\, X_{i-j_2} + \cdots + a_k\, X_{i-j_k} + c \pmod{m} \qquad \text{or} \\ X_{i+1} &= a\, X_{i-j_1} \cdot X_{i-j_2} \cdots X_{i-j_k} + c \pmod{m} \end{aligned}$$

Nowadays "the best" generators use combinations of the generators described along this section in order to increase the randomness and the period of the generated sequences.

1.1 DERIVE's random function

DERIVE's random function uses a mixed generator given by:

$$X_{i+1} = 2.654.435.721\, X_i + 1 \pmod{2^{32}}$$

which satisfies the conditions to be a full period generator, that is, the period of DERIVE's random function is $2^{32} = 4.294.967.296$.

DERIVE's random function `RANDOM(n)` can be used with any $n \in \mathbb{Z}$ with the following meanings:

[1] These generators are often called *Fibonacci* generators because one example is given by the Fibonacci serie:

$$X_{i+1} = X_i + X_{i-1} \pmod{m}$$

- If $n > 1$, `RANDOM(n)` simplifies to a random integer in the interval $[0, n)$.
- `RANDOM(1)` simplifies to a random number in the interval $[0, 1)$.
- If $n < 0$, `RANDOM(n)` simplifies to $-n$ and initializes the random number state variable to $-n$.
- `RANDOM(0)` simplifies to the time in centiseconds since the current calendar year began and initializes the random number state variable to that time.

Although this is a "good" generator, the following two subsections describe two different algorithms, implemented in the package `Random_distributions`, which periods are quite much longer and also improve the randomness.

1.2 ran2 algorithm

The `ran2` algorithm was proposed by L'Ecuyer and is described in [Press and Teukolsky, 1992] and [Press et al., 1999].

This algorithm merges the following two multiplicative generators:

$$\begin{aligned} X_{i+1} &= 40014\, X_i \pmod{2^{31} - 85} \\ Y_{i+1} &= 40692\, Y_i \pmod{2^{31} - 249} \end{aligned}$$

This algorithm has been used for a long time as one of the best generators and its period is about $2.3 \cdot 10^{18} = 2.300.000.000.000.000.000$ which is more than 535.510.480 times longer than DERIVE's random generator period.

The implementation on DERIVE has been carried out "translating" the "C" code developed in [Press et al., 1999] and it uses the following two functions:

- `ran2(n)` which is the main algorithm. This function returns a vector of size n of random numbers in the interval $[0, 1)$.
- `ran2_initialize()` This auxiliar function is used to set the variables and constants needed for the algorithm.

1.3 mzran13 algorithm

The `mzran13` algorithm was proposed by G. Marsaglia and A. Zaman as an alternative to `ran2`. This algorithm is described in [Marsaglia and Zaman, 1994].

This algorithm merges the two generators: a mixed one with a Fibonacci's like one.

$$\begin{aligned} X_{i+1} &= 69069\, X_i + 1.013.904.243 \pmod{2^{32}} \\ Y_{i+1} &= Y_{i-1} - Y_{i-2} - \text{"c"} \pmod{2^{32} - 18} \end{aligned}$$

where the second one is a subtract-with-borrow generator (because of the term "c").

This algorithm has been found to be at least as good as `ran2` but simpler, much faster and with periods "millions and millions" of times longer. Specifically, its period is over

CONTS

$$2^{94} = 19.807.040.628.566.084.398.385.987.584,$$

that is, 8.611.756.795 times longer than `ran2`'s period and 4.611.686.018.427.387.904 times longer than DERIVE's period.

The implementation on DERIVE has been carried out "translating" the "C" code developed in [Marsaglia and Zaman, 1994] and it uses the following function:

- `mzran13(n)`. This function returns a vector of size n of random numbers in the interval $[0, 1)$. Previously, when the package `Random_distribution` is loaded, the needed constants and variables are initialized.

This algorithm is the base for all the random distribution generations developed in this package.

2 Different methods for random variate generation

This section presents some general methods for generating random variables from different continuous and discrete distributions. In the following subsections three general methods for continuous distributions and one for discrete distributions are described.

2.1 Inverse transform method

Let $\mathcal{X}$ be a random variable with cumulative probability distribution function $F_{\mathcal{X}}(x)$. Since $F_{\mathcal{X}}(x)$ is a nondecreasing function, the inverse function $F_{\mathcal{X}}^{-1}(y)$ may be defined for any value of y between 0 and 1 as: $F_{\mathcal{X}}^{-1}(y)$ is the smallest x satisfying $F_{\mathcal{X}}(x) \geq y$, that is,

$$F_{\mathcal{X}}^{-1}(y) = \inf \{F_{\mathcal{X}}(x) \geq y\}, \qquad 0 \leq y \leq 1$$

If $\mathcal{U}$ is uniformly distributed over the interval $(0, 1)$, then $\mathcal{X} = F_{\mathcal{X}}^{-1}(\mathcal{U})$. So, to get a value x of the random variable $\mathcal{X}$, a value u from a random uniform variable $\mathcal{U}(0, 1)$ can be obtained and compute $x = F_{\mathcal{X}}^{-1}(y)$. Thus, the general algorithm for the inverse transform method is:

1. Generate a value u from $\mathcal{U}(0, 1)$.
2. Obtain $x = F_{\mathcal{X}}^{-1}(u)$ as the random number from the variable $\mathcal{X}$.

The only condition needed for this method is that $F_{\mathcal{X}}^{-1}$ exists in an analytical form.

The following DERIVE's program has been developed in the package `Random_distributions` to obtain a formula to generate $\mathcal{X}$ using the inverse transform method:

```
Inverse_transform_method(f, ini := 0, u) :=
Prog
(
  u :∈ Real [0,1],
  Solve(u = INT(f, x, ini, x), x, Real)
)
```

2.2 Composition method

CONTS

This method is employed by Butler and consists of expressing the probability density function $f_{\mathcal{X}}(x)$ of the distribution to be simulated as a probability mixture of properly selected density functions.

Let $g(x|y)$ be a family of one-parameter density functions, where y is the parameter identifying a unique $g(x)$. If a value of y is drawn from a continuous cumulative function $F_{\mathcal{Y}}(y)$ and then if $\mathcal{X}$ is sampled from the $g(x)$ for that chosen y, the density function for $\mathcal{X}$ will be

$$f_{\mathcal{X}}(x) = \int g(x|y)\, \mathrm{d}F_{\mathcal{Y}}(y)$$

If y is an integer parameter, then

$$f_{\mathcal{X}}(x) = \sum_i P_i\, g(x|y=i)$$

where

$$\sum_i P_i = 1 \qquad ; \qquad P_i > 0 \qquad ; \qquad P_i = P\left[\mathcal{Y} = i\right] \qquad i = 1, 2, \ldots$$

This method may be applied for generating complex distributions from simpler distributions that are themselves easily generated by the inverse transform method or by the acceptance-rejection method described below.

2.3 Acceptance–rejection method

This method is due to von Neumann and consists on sampling a random variate from an appropriate distribution and subjecting it to a test to determine whether or not it will be acceptable for use.

To carry out this method, the probability density function $f_{\mathcal{X}}(x)$ of the variable $\mathcal{X}$ must be expressed as:

$$f_{\mathcal{X}}(x) = C \cdot h(x) \cdot g(x)$$

where $C \geq 1$, $h(x)$ is also a probability density function, and $0 < g(x) \leq 1$. After generating two random values u and y from $\mathcal{U}(0,1)$ and $h(y)$, respectively, the test to see wether or not the inequality $u \leq g(y)$ holds must be done, and:

1. If the inequality holds, then accept y as a variate generated from $f_{\mathcal{X}}(x)$.
2. If the inequality is violated, reject the pair u , y and try again.

So, the general algorithm for the acceptance–rejection method is:

1. Generate a value u from $\mathcal{U}(0,1)$
2. Generate y from the probability density function $h(y)$.
3. If $u \leq g(y)$, return y as the variate generated from $f_{\mathcal{X}}(x)$
4. Go to step 1.

2.4 Inverse transform method for discrete distributions

CONTS

The inverse transform method is the easier method to use not only for continuous distributions but also for discrete distribution.

Let $\mathcal{X}$ be a random discrete variate which finite or infinite possible values are

$$x_1, x_2, \ldots, x_i, \ldots.$$

Let $F_{\mathcal{X}}(x)$ be its probability mass function given by

$$F(x_i) = P[\mathcal{X} = x_i] = p_i \quad i = 1, 2, \ldots$$

The inverse transform method can be described as follow:

1. Generate u from $\mathcal{U}(0, 1)$.
2. $i := 1$.
3. $p := p_1$.
4. If $u \leq p$ deliver x_i as the generated value.
5. $i := i + 1$.
6. $p := p + p_i$
7. Go to step 4.

On the other hand, the values x_i can be assumed to be all integers and $x_{i+1} = x_i + 1$, because if this is not the case, the correspondence $\phi(x_i) = i$ can be established and consider a new random discrete variate $\mathcal{Y}$ which values are $1, 2, \ldots$ and $F_{\mathcal{Y}}(i) = P[\mathcal{Y} = i] = P[\mathcal{X} = x_i] = p_i$, which is equivalent to $\mathcal{X}$ and verify the above condition.

Let $\mathcal{X} = \{\texttt{ini}, \texttt{ini} + 1, \texttt{ini} + 2, \ldots\}$ for some $\texttt{ini} \in \mathbb{Z}$ with probability mass function $F_{\mathcal{X}}(x) := P[\mathcal{X} = x] = p_x \quad ; \quad x = \texttt{ini}, \texttt{ini} + 1, \texttt{ini} + 2, \ldots$

The following DERIVE's program has been developed in the package `Random_distributions` to generate an element of $\mathcal{X}$ using the inverse transform method:

```
random_discrete_aux(F, ini := 0, aleat, i_, p) :=
Prog
(
  i_ := ini,
  p := SUBST(F, x, i_),
  Loop
  (
    If aleat ≥ p,
      RETURN i_),
    i_ := i_ + 1,
    p := p + SUBST(F, x, i_)
  )
)
```

while the following DERIVE's program generate a sample of size n of $\mathcal{X}$.

```
random_discrete(n := 1, F, ini := 0, vecaleat) :=
Prog
(
  vecaleat := random_uniform(n),
  VECTOR(random_discrete_aux(F, ini, vecaleat sub j), j, n)
)
```

CONTS

3 Continuous distributions random generation

This section describes generating procedures for different continuous distributions.

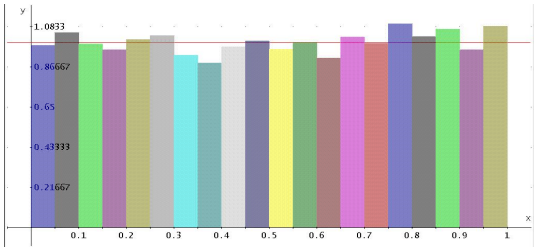
All these continuous distributions are presented with their probability density functions and with a drawing of a bars diagram, obtained by the DERIVE's algorithm developed in the package `Random_distributions`, together with the plot of the corresponding probability density function in order to see graphically if it is a "good" sample of generated values.

See [Rubinstein, 1981] and [Galán, 1991] for further information on the algorithm described below.

3.1 Uniform distribution

A random variable $\mathcal{X}$ has an uniform distribution in the interval (a, b) ($\mathcal{X} \rightsquigarrow \mathcal{U}(a, b)$) if its probability density function is:

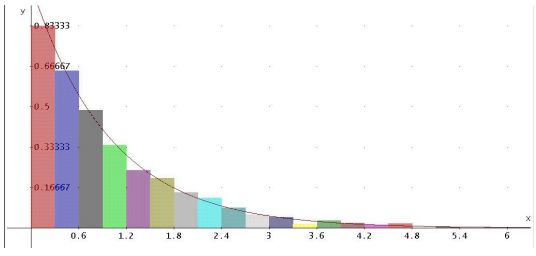
$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{1}{b-a} & x \in (a, b) \\ 0 & \text{otherwise} \end{cases}$$



$\mathcal{U}(0, 1)$

To generate $\mathcal{X}$, first u must be generated from $\mathcal{U}(0, 1)$ and then return $\boxed{a + (b - a)\, u}$ as the generated value.

3.2 Exponential distribution



$\mathcal{E}(1)$

A random variable $\mathcal{X}$ has an exponential distribution with parameter $\lambda > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{E}(\lambda)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{1}{\lambda}\, \mathrm{e}^{-x/\lambda} & x \in [0, \infty) \\ 0 & \text{otherwise} \end{cases}$$

To generate $\mathcal{X}$ the inverse transform method is used:

$$\mathcal{U} = F_{\mathcal{X}}(x) = \int_0^x \frac{1}{\lambda}\, \mathrm{e}^{-t/\lambda}\, \mathrm{d}t = 1 - \mathrm{e}^{-x/\lambda} \quad \Longrightarrow \quad \mathcal{X} = -\lambda\, \ln(1 - \mathcal{U}) \equiv -\lambda\, \ln(\mathcal{U})$$

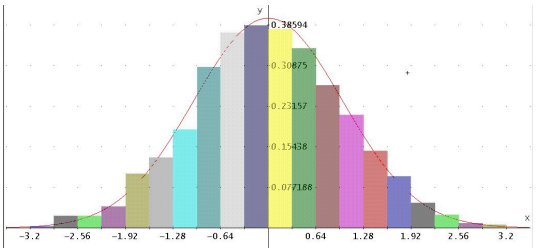
Thus, to generate $\mathcal{X}$, u must be generated from $\mathcal{U}(0, 1)$ and then return $\boxed{x = -\lambda \ln u}$ as the generated value.

3.3 Normal distribution

CONTS

A random variable $\mathcal{X}$ has a normal distribution with parameters μ and σ ($\mathcal{X} \rightsquigarrow \mathcal{N}(\mu, \sigma)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \frac{1}{\sigma\sqrt{2\pi}}\, \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad x \in \mathbb{R}$$



$\mathcal{N}(0,1)$

To generate $\mathcal{X}$, Box and Muller theorem establishes that:

If $\mathcal{U}_1 \rightsquigarrow \mathcal{U}(0,1)$ and $\mathcal{U}_2 \rightsquigarrow \mathcal{U}(0,1)$ then

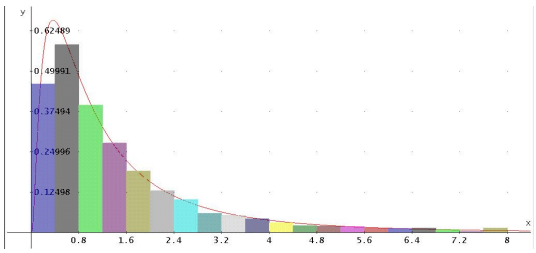
$$\mathcal{Z}_1 = \sqrt{-2\ln(\mathcal{U}_1)}\,\cos(2\pi\mathcal{U}_2) \qquad \text{and} \qquad \mathcal{Z}_2 = \sqrt{-2\ln(\mathcal{U}_1)}\,\sin(2\pi\mathcal{U}_2)$$

are independent standard normal deviates $\mathcal{N}(0,1)$

On the other hand, if $\mathcal{Z} \rightsquigarrow \mathcal{N}(0,1)$ then $(\mu + \sigma\,\mathcal{Z}) \rightsquigarrow \mathcal{N}(\mu, \sigma)$

Thus, to generate $\mathcal{X}$, u_1 and u_2 must be generated from $\mathcal{U}(0,1)$ and then return any of the values $\boxed{\mu + \sigma\sqrt{-2\ln(u_1)}\,\cos(2\pi\, u_2)}$ or $\boxed{\mu + \sigma\sqrt{-2\ln(u_1)}\,\sin(2\pi\, u_2)}$ as the generated value.

3.4 Lognormal distribution



$\mathcal{LN}(0,1)$

If the random variable $\mathcal{Z} \rightsquigarrow \mathcal{N}(\mu, \sigma)$ then $\mathcal{X} = \mathrm{e}^{\mathcal{Z}}$ has the lognormal distribution with parameters μ and σ ($\mathcal{X} \rightsquigarrow \mathcal{LN}(\mu, \sigma)$). Its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{1}{x\sigma\sqrt{2\pi}}\, \mathrm{e}^{-\frac{(\ln x-\mu)^2}{2\sigma^2}} & x \in [0, \infty) \\ 0 & \text{Otherwise} \end{cases}$$

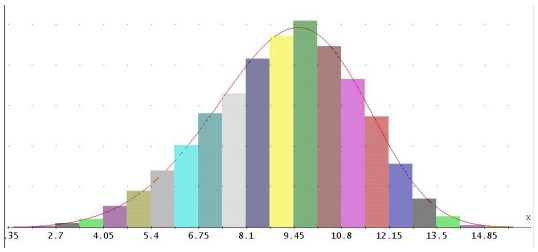
To generate $\mathcal{X}$, z must be generated from $\mathcal{N}(\mu, \sigma)$ and then return $\boxed{x = \mathrm{e}^z}$ as the generated value.

3.5 Weibul distribution

CONTS

A random variable $\mathcal{X}$ has a Weibul distribution with parameters $\alpha > 0$ and $\beta > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{W}(\alpha, \beta)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{\alpha}{\beta^{\alpha}}\, x^{\alpha-1}\, \mathrm{e}^{-(\frac{x}{\beta})^{\alpha}} & x \in [0, \infty) \\ \\ 0 & \text{Otherwise} \end{cases}$$



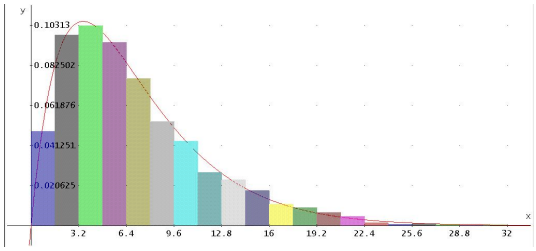
$\mathcal{W}(5, 10)$

To generate $\mathcal{X}$ the inverse transform method can be used:

$$\begin{aligned} \mathcal{U} &= F_{\mathcal{X}}(x) = \int_0^x \frac{\alpha}{\beta^{\alpha}}\, t^{\alpha-1}\, \mathrm{e}^{-(\frac{t}{\beta})^{\alpha}}\, \mathrm{d}t = 1 - \mathrm{e}^{-(\frac{x}{\beta})^{\alpha}} \qquad \Longrightarrow \\ \left(\frac{\mathcal{X}}{\beta}\right)^{\alpha} &= -\ln(1-\mathcal{U}) \equiv -\ln \mathcal{U} \rightsquigarrow \mathcal{E}(1) \qquad \Longrightarrow \\ \mathcal{X} &\equiv \beta \left(\mathcal{E}(1)\right)^{1/\alpha} \end{aligned}$$

To generate $\mathcal{X}$, v must be generated from $\mathcal{E}(1)$ and then return $\boxed{x = \beta v^{1/\alpha}}$ as the generated value.

3.6 Gamma distribution



$\mathcal{G}(2, 3.5)$

A random variable $\mathcal{X}$ has a Gamma distribution with parameters $\alpha > 0$ and $\beta > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{G}(\alpha, \beta)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{x^{\alpha-1}\, \mathrm{e}^{-x/\beta}}{\beta^{\alpha}\, \Gamma(\alpha)} & x \in [0, \infty) \\ \\ 0 & \text{Otherwise} \end{cases}$$

The inverse transform method cannot be applied since $F_{\mathcal{X}}^{-1}(x)$ does not exist in an explicit form.

In this case, different algorithms have been developed in order to generate samples of the Gamma distribution. Finally, the main algorithm chooses which is the appropriated one depending on the values of parameters α and β.

3.6.1 random_gamma1

This algorithm is valid for values of $\alpha > 1$. The following two properties of gamma distribution are the base to develop this algorithm:

1. $\mathcal{G}(1, \beta) = \mathcal{E}(\beta)$.
2. If $\mathcal{X}_1 \rightsquigarrow \mathcal{G}(\alpha_1, \beta)$ and $\mathcal{X}_2 \rightsquigarrow \mathcal{G}(\alpha_2, \beta)$ then $\mathcal{X} = \mathcal{X}_1 + \mathcal{X}_2 \rightsquigarrow \mathcal{G}(\alpha_1 + \alpha_2, \beta)$, that is, gamma distribution is reproductive with respect its first parameter.

Let $\alpha > 1$, $m = \text{floor}(\alpha)$ and $\delta = \alpha - m$ where $\text{floor}(\alpha)$ is the integer part of α. In order to generate $\mathcal{X} \rightsquigarrow \mathcal{G}(\alpha, \beta)$ a mixture of $\mathcal{G}(m, \beta)$ and $\mathcal{G}(m+1, \beta)$ with probabilities $1 - \delta$ and δ respectively can be used. On the other hand, in order to generate them, m or $m+1$ variables from $\mathcal{G}(1, \beta) = \mathcal{E}(\beta)$ must be generated as shown in 3.2. Thus, given $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_{m+1} \rightsquigarrow \mathcal{U}(0,1)$:

CONTS

$$\mathcal{X} = -\beta \ln(\mathcal{U}_1) - \beta \ln(\mathcal{U}_2) - \cdots - \beta \ln(\mathcal{U}_m) = -\beta \ln\left(\prod_{i=1}^{m} \mathcal{U}_i\right) \rightsquigarrow \mathcal{G}(m, \beta) \qquad \text{and}$$

$$\mathcal{Y} = -\beta \ln(\mathcal{U}_1) - \beta \ln(\mathcal{U}_2) - \cdots - \beta \ln(\mathcal{U}_{m+1}) = -\beta \ln\left(\prod_{i=1}^{m+1} \mathcal{U}_i\right) \rightsquigarrow \mathcal{G}(m+1, \beta)$$

The algorithm is:

1. Get $u, u_1, u_2, \ldots, u_m, u_{m+1}$ from $\mathcal{U}(0,1)$.
2. Let $x = \prod_{i=1}^{m} u_i$.
3. If $\delta \leq u$ then let $x = x \cdot u_{m+1}$.
4. Return $-\beta \ln(x)$ as the generate value.

3.6.2 random_gamma2

If $0 < \alpha < 1$ then $\mathcal{X} = \mathcal{Y} \cdot \mathcal{V}$ where $\mathcal{X} \rightsquigarrow \mathcal{G}(\alpha, \beta)$; $\mathcal{Y} \rightsquigarrow \mathcal{B}e(\alpha, 1-\alpha)$ and $\mathcal{V} \rightsquigarrow \mathcal{E}(\beta)$. (Beta distribution $\mathcal{B}e$ is described in section 3.7). Thus, `random_gamma2` algorithm to generate $\mathcal{X} \rightsquigarrow \mathcal{G}(\alpha, \beta)$ $(0 < \alpha < 1)$ can be described by:

1. Generate y from $\mathcal{B}e(\alpha, 1-\alpha)$ (using algorithm `random_beta4` described in 3.7.2).
2. Generate v from $\mathcal{E}(\beta)$.
3. Return $y \cdot v$ as generated value.

3.6.3 random_gamma5

This is an acceptance–rejection method due to Cheng and describes gamma generation $\mathcal{G}(\alpha, 1)$ for $\alpha > 1$. Let remember that the acceptance–rejection method is based in the following decomposition:

$$f_{\mathcal{X}}(x) = C \cdot h(x) \cdot g(x)$$

Cheng's procedure uses:

$$h(x) = \begin{cases} \dfrac{\lambda\ \mu\ x^{\lambda-1}}{(\mu + x^{\lambda})^2} & x \geq 0 \\ 0 & \text{Otherwise} \end{cases}$$

$$C = \frac{4\alpha^{\alpha}}{\Gamma(\alpha)\ \mathrm{e}^{\alpha}\ \lambda}$$

$$g(x) = x^{\alpha-\lambda}\ \left(\mu + x^{\lambda}\right)^2 \frac{\mathrm{e}^{\alpha-x}}{4\ \alpha^{\alpha+\lambda}} \qquad \text{where}$$

$$\lambda = \sqrt{2\alpha - 1} \qquad ; \qquad \mu = \alpha^{\lambda}$$

Setting $a = \dfrac{1}{\lambda}$, $b = \alpha - \ln 4$ and $c = \alpha + a$ Cheng's algorithm can be written as:

1. Get u_1 and u_2 from $\mathcal{U}(0,1)$.
2. Let $v = a \ln\left(\frac{u_1}{1-u_1}\right)$.
3. Let $x = \alpha\, e^v$.
4. If $b + c\, v - x \geq \ln(u_1^2 u_2)$ return x as the generated value for $\mathcal{G}(\alpha,1)$.
5. Go to step 1.

CONTS

3.6.4 random_gamma9

This algorithm uses the approximation to Gamma distribution by Normal distribution when α and β are not "small".

Specifically, if $\mathcal{Z} \rightsquigarrow \mathcal{N}\left(\ln\left(\frac{\alpha}{\beta}\right) - \frac{1}{2\alpha}, \sqrt{\frac{1}{\alpha}}\right)$ then $\mathcal{G}(\alpha,\beta) \approx e^{\mathcal{Z}}$. Thus, the algorithm is:

1. Generate z from $\mathcal{N}\left(\ln\left(\frac{\alpha}{\beta}\right) - \frac{1}{2\alpha}, \sqrt{\frac{1}{\alpha}}\right)$.
2. Deliver $x = e^z$ as the generated value for $\mathcal{G}(\alpha,\beta)$.

3.6.5 random_gamma

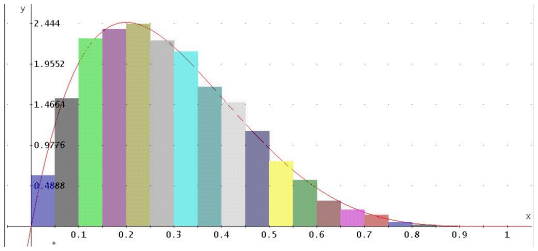
Finally, the following algorithm runs the above algorithms depending on the parameters α and β in order to generate a sample of size n from $\mathcal{G}(\alpha,\beta)$:

```
random_gamma(n := 1, α := 1, β := 1) :=
Prog
(
  If (α = 1, return random_exponential(n, β)),
  If (α < 1, return random_gamma2(n, α, β)),
  If (β = 1, return random_gamma5(n, α, β)),
  If (α > 20 and β > 20, return random_gamma9(n, α, β)),
  return random_gamma1(n, α, β)
)
```

3.7 Beta distribution

$\mathcal{X}$ has a Beta distribution with parameters $\alpha > 0$ and $\beta > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{B}e(\alpha,\beta)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\cdot\Gamma(\beta)}\, x^{\alpha-1}(1-x)^{\beta-1} & x \in [0,1] \\ 0 & \text{Otherwise} \end{cases}$$



$\mathcal{B}e(2,5)$

The inverse transform method cannot be applied since $F_{\mathcal{X}}^{-1}(x)$ does not exist in an explicit form.

In this case, as in the case of Gamma distribution, different algorithms have been developed in order to generate samples of the Beta distribution. Finally, the main algorithm chooses which is the appropriated one depending on the values of parameters α and β.

CONTS

3.7.1 random_beta1

This algorithm is based in the following result:

> If $\mathcal{Y}_1 \rightsquigarrow \mathcal{G}(\alpha, 1)$ and $\mathcal{Y}_2 \rightsquigarrow \mathcal{G}(\beta, 1)$ then $\mathcal{X} = \dfrac{\mathcal{Y}_1}{\mathcal{Y}_1 + \mathcal{Y}_2} \rightsquigarrow \mathcal{B}\mathrm{e}(\alpha, \beta)$.

Its implementation is therefore trivial:

1. Generate y_1 and y_2 from $\mathcal{G}(\alpha, 1)$ and $\mathcal{G}(\beta, 1)$ respectively.
2. Deliver $x = \dfrac{y_1}{y_1 + y_2}$ as the generated value for $\mathcal{B}\mathrm{e}(\alpha, \beta)$.

3.7.2 random_beta4

This algorithm has been developed for using in algorithm `random_gamma2`. It is due to Jöhnk and is based on the following result:

Let $\mathcal{U}_1 \rightsquigarrow \mathcal{U}(0, 1)$ and $\mathcal{U}_2 \rightsquigarrow \mathcal{U}(0, 1)$ and let $\mathcal{Y}_1 = \mathcal{U}_1^{1/\alpha}$ and $\mathcal{Y}_2 = \mathcal{U}_2^{1/\beta}$. If $\mathcal{Y}_1 + \mathcal{Y}_2 < 1$ then $\mathcal{X} = \dfrac{\mathcal{Y}_1}{\mathcal{Y}_1 + \mathcal{Y}_2} \rightsquigarrow \mathcal{B}\mathrm{e}(\alpha, \beta)$.

The algorithm is:

1. Generate u_1 and u_2 from $\mathcal{U}(0, 1)$.
2. Set $y_1 = u_1^{1/\alpha}$ and $y_2 = u_2^{1/\beta}$.
3. If $y_1 + y_2 < 1$ deliver $x = \dfrac{y_1}{y_1 + y_2}$ as the generated value for $\mathcal{B}\mathrm{e}(\alpha, \beta)$.
4. Go to step 1.

3.7.3 random_beta7

This algorithm uses the approximation to Beta distribution by Normal distribution when α and β are not "large enough".

Specifically, if $\mathcal{X} \rightsquigarrow \mathcal{B}\mathrm{e}(\alpha, \beta)$ then $\ln\left(\dfrac{\mathcal{X}}{1 - \mathcal{X}}\right) \approx \mathcal{N}(\mu, \sigma)$ where $\mu = \ln\left(\dfrac{\alpha}{\beta}\right) + \dfrac{\alpha - \beta}{2\alpha\beta}$ and $\sigma = \sqrt{\dfrac{\alpha + \beta}{\alpha\beta}}$.

Thus,

$$\ln\left(\frac{x}{1 - x}\right) = z \qquad \Longrightarrow \qquad x = \frac{\mathrm{e}^z}{(1 + \mathrm{e}^z)}$$

and hence, the algorithm is:

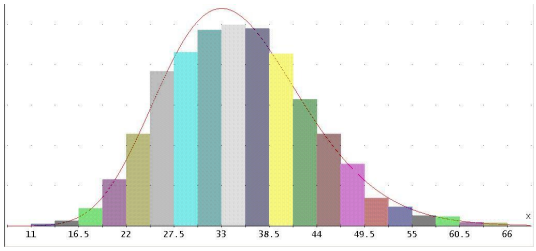
1. Generate z from $\mathcal{N}\left(\ln\left(\dfrac{\alpha}{\beta}\right) + \dfrac{\alpha - \beta}{2\alpha\beta}, \sqrt{\dfrac{\alpha + \beta}{\alpha\beta}}\right)$.
2. Deliver $x = \dfrac{\mathrm{e}^z}{(1 + \mathrm{e}^z)}$ as the generated value for $\mathcal{B}\mathrm{e}(\alpha, \beta)$.

3.7.4 random_beta

Finally, the following algorithm runs the above algorithms depending on the parameters α and β in order to generate $\mathcal{B}e(\alpha, \beta)$:

```
random_beta(n := 1, α := 1, β := 1) :=
If ( α < 4 or β < 4,
    random_beta1(n, α, β),
    random_beta7(n, α, β)
    )
```

3.8 Chi–Square distribution



$\chi^2(35)$

Let $\mathcal{Z}_i \rightsquigarrow \mathcal{N}(0,1) \quad i = 1, \ldots, k \quad k$ standard normal independent distributions. In this case, $\mathcal{X} = \sum_{i=1}^{k} \mathcal{Z}_i^2$ has the chi–square distribution with k degrees of freedom ($\mathcal{X} \rightsquigarrow \chi^2(k)$).

Its probability density function is given by:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{x^{k/2-1}\ \mathrm{e}^{-x/2}}{2^{k/2}\ \Gamma\left(\frac{k}{2}\right)} & x \in [0, \infty) \\ \\ 0 & \text{Otherwise} \end{cases}$$

Although the algorithm to generate $\mathcal{X} \rightsquigarrow \chi^2(k)$ would be trivial by definition, it would need k values from $\mathcal{N}(0,1)$ which require many operations if k is "large". Thus, in the next two sections, two different algorithms which improve (in number of operations) the "definition algorithm" are described.

3.8.1 random_chi_square2

This algorithm is valid for "large" values of k (say $k > 30$) and it uses the following approximation from the standard normal distribution:

If $\mathcal{X} \rightsquigarrow \chi^2(k)$ then $\mathcal{Z} = \sqrt{2\mathcal{X}} - \sqrt{2k-1}$ is such that $\mathcal{Z} \rightsquigarrow \mathcal{N}(0,1)$

Solving for $\mathcal{X}$, $\mathcal{X} = \dfrac{\left(\mathcal{Z} + \sqrt{2k-1}\right)^2}{2}$. Thus, to generate $\mathcal{X} \rightsquigarrow \chi^2(k)$ z must be generated from $\mathcal{N}(0,1)$ and then return $\boxed{x = \dfrac{\left(z + \sqrt{2k-1}\right)^2}{2}}$ as the generated value.

3.8.2 random_chi_square3

This algorithm is based in the fact that $\chi^2(k)$ is a particular case of a gamma density. Specifically, $\chi^2(k) \equiv \mathcal{G}\left(\frac{k}{2}, 2\right)$. Thus, to generate $\mathcal{X} \rightsquigarrow \chi^2(k)$, g must be generated from $\mathcal{G}\left(\frac{k}{2}, 2\right)$ and then return $\boxed{x = g}$ as the generated value.

3.8.3 random_chi_square

CONTS

Finally, the following algorithm runs the above algorithms depending on the parameter k in order to generate $\chi^2(k)$:

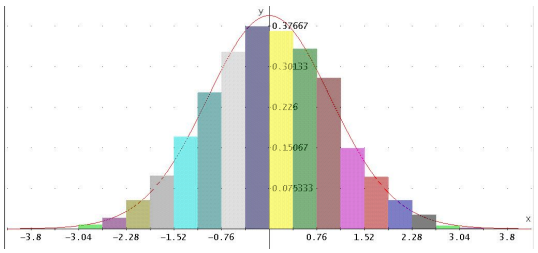
```
random_chi_square(n := 1, k := 1) :=
If ( k > 30,
    random_chi_square2(n, k),
    random_chi_square3(n, k)
    )
```

3.9 Student's t distribution

Let $\mathcal{Z} \rightsquigarrow \mathcal{N}(0,1)$ and $\mathcal{Y} \rightsquigarrow \chi^2(k)$ independents. Then $\mathcal{X} = \dfrac{\mathcal{Z}}{\sqrt{\mathcal{Y}/k}}$ has a Student's t distribution with k degrees of freedom ($\mathcal{X} \rightsquigarrow t(k)$).

Its probability density function is:

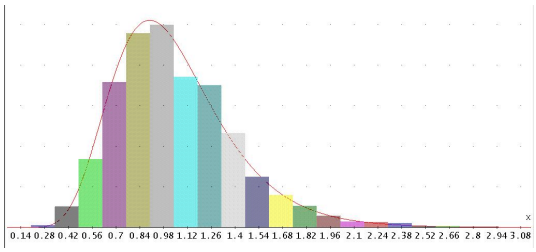
$$f_{\mathcal{X}}(x) = \frac{\Gamma\left(\frac{k+1}{2}\right)}{\sqrt{k\pi}\,\Gamma\left(\frac{k}{2}\right)} \left(1 + \frac{x^2}{k}\right)^{-(k+1)/2} \qquad x \in \mathbb{R}$$



$t(35)$

To generate $\mathcal{X} \rightsquigarrow t(k)$, z must be generated from $\mathcal{N}(0,1)$ and y from $\chi^2(k)$ and then return $\boxed{x = \dfrac{z}{\sqrt{y/k}}}$ as the generated value.

3.10 F distribution



$\mathcal{F}(32, 45)$

Let $\mathcal{Y}_1 \rightsquigarrow \chi^2(k_1)$ and $\mathcal{Y}_2 \rightsquigarrow \chi^2(k_2)$ independents. Then $\mathcal{X} = \dfrac{\mathcal{Y}_1/k_1}{\mathcal{Y}_2/k_2}$ has a $\mathcal{F}$ distribution with k_1 and k_2 degrees of freedom ($\mathcal{X} \rightsquigarrow \mathcal{F}(k_1, k_2)$).

Its probability density function is given by:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{\Gamma\left(\frac{k_1+k_2}{2}\right)\left(\frac{k_1}{k_2}\right)^{k_1/2}\, x^{k_1/2-1}}{\Gamma\left(\frac{k_1}{2}\right)\Gamma\left(\frac{k_2}{2}\right)\left(1+\frac{k_1}{k_2}x\right)^{(k_1+k_2)/2}} & x \in (0,\infty) \\ 0 & \text{Otherwise} \end{cases}$$

To generate $\mathcal{X} \rightsquigarrow F(k_1, k_2)$, y_1 and y_2 must be generated from $\chi^2(k_1)$ and $\chi^2(k_2)$ respectively and then return $\boxed{x = \dfrac{y_1/k_1}{y_2/k_2}}$ as the generated value.

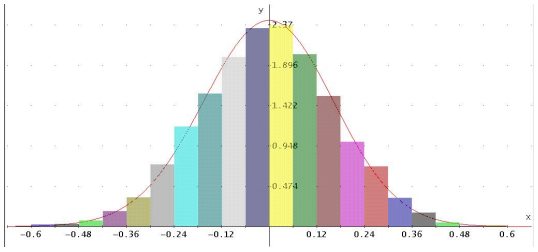
3.11 Z distribution

CONTS

Let $\mathcal{Y} \rightsquigarrow \mathcal{F}(k_1, k_2)$. Then, the variable $\mathcal{X} = \dfrac{\ln(\mathcal{Y})}{2}$ has a $\mathcal{Z}$ distribution with k_1 and k_2 degrees of freedom ($\mathcal{X} \rightsquigarrow \mathcal{Z}(k_1, k_2)$).

Its probability density function is given by:

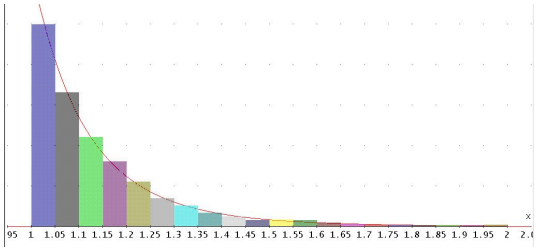
$$f_{\mathcal{X}}(x) = \frac{2\,\Gamma\left(\frac{k_1+k_2}{2}\right)\left(\frac{k_1}{k_2}\right)^{k_1/2}\,\mathrm{e}^{k_1 x}}{\Gamma\left(\frac{k_1}{2}\right)\Gamma\left(\frac{k_2}{2}\right)\left(1+\frac{k_1\,\mathrm{e}^{2x}}{k_2}\right)^{(k_1+k_2)/2}} \qquad x \in \mathbb{R}$$



$\mathcal{Z}(32, 45)$

To generate $\mathcal{X} \rightsquigarrow \mathcal{Z}(k_1, k_2)$, y must be generated from $\mathcal{F}(k_1, k_2)$ and then return $\boxed{x = \dfrac{\ln(y)}{2}}$ as the generated value.

3.12 Pareto distribution



$\mathcal{P}\mathrm{a}(8, 1)$

A random variable $\mathcal{X}$ has a Pareto distribution with parameters $\alpha > 0$ and $x_0 > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{P}\mathrm{a}(\alpha, x_0)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \begin{cases} \dfrac{\alpha}{x_0}\left(\dfrac{x_0}{x}\right)^{\alpha+1} & x \in [x_0, \infty) \\ 0 & \text{otherwise} \end{cases}$$

To generate $\mathcal{X}$, the inverse transform method can be used since:

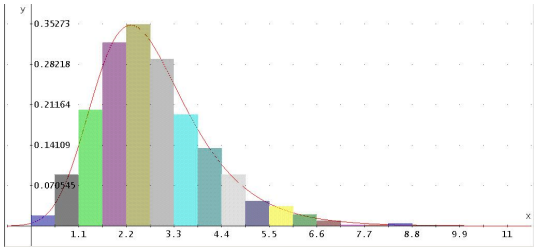
$$\mathcal{U} = F_{\mathcal{X}}(x) = \int_{x_0}^{x} \frac{\alpha}{x_0}\left(\frac{x_0}{t}\right)^{\alpha+1} \mathrm{d}t = 1 - \left(\frac{x_0}{x}\right)^{\alpha} \quad \Longrightarrow \quad \mathcal{X} = \frac{x_0}{(1-\mathcal{U})^{1-\alpha}} \equiv \frac{x_0}{\mathcal{U}^{1-\alpha}}$$

Thus, to generate $\mathcal{X} \rightsquigarrow \mathcal{P}\mathrm{a}(\alpha, x_0)$, u must be generated from $\mathcal{U}(0, 1)$ and then deliver $\boxed{\dfrac{\mathtt{x_0}}{\mathtt{u}^{1/\alpha}}}$ as the generated value.

3.13 Logistic distribution

A random variable $\mathcal{X}$ has a Logistic distribution with parameter $\alpha > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{L}(\alpha)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \frac{\alpha\,\mathrm{e}^{-x}}{(1+\mathrm{e}^{-x})^{\alpha+1}} \qquad x \in \mathbb{R}$$



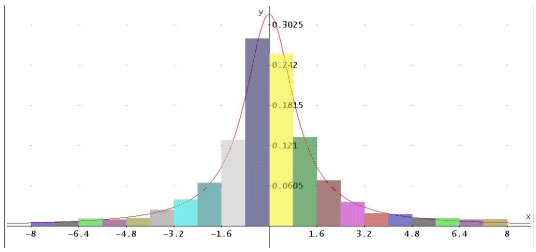
$\mathcal{L}(10)$

To generate $\mathcal{X}$, the inverse transform method can be used since:

$$\mathcal{U} = F_{\mathcal{X}}(x) = \int_{-\inf}^{x} \frac{\alpha\ \mathrm{e}^{-t}}{(1+\mathrm{e}^{-t})^{\alpha+1}}\,\mathrm{d}t = \frac{1}{(1+\mathrm{e}^{-x})^{\alpha}} \quad \Longrightarrow \quad \mathcal{X} = -\ln\left(\frac{1-\mathcal{U}^{1/\alpha}}{\mathcal{U}^{1/\alpha}}\right)$$

Thus, in order to generate $\mathcal{X} \rightsquigarrow \mathcal{L}(\alpha)$, u must be generated from $\mathcal{U}(0,1)$ and then deliver $\boxed{-\ln\left(\frac{1-u^{1/\alpha}}{u^{1/\alpha}}\right)}$ as the generated value.

3.14 Cauchy distribution



$\mathcal{C}(0,1)$

A random variable $\mathcal{X}$ has a Cauchy distribution with parameters $\alpha \geq 0$ and $\beta > 0$ ($\mathcal{X} \rightsquigarrow \mathcal{C}(\alpha,\beta)$) if its probability density function is:

$$f_{\mathcal{X}}(x) = \frac{\beta}{\pi\left[\beta^2 + (x-\alpha)^2\right]} \qquad x \in \mathbb{R}$$

To generate $\mathcal{X}$, the inverse transform method can be used since:

$$\mathcal{U} = F_{\mathcal{X}}(x) = \int_{-\infty}^{x} \frac{\beta}{\pi\left[\beta^2 + (t-\alpha)^2\right]}\,\mathrm{d}t = \frac{1}{2} + \frac{\operatorname{atan}\left(\frac{x-\alpha}{\beta}\right)}{\pi} \quad \Longrightarrow \quad \mathcal{X} = \alpha + \beta\tan\left[\pi\left(\mathcal{U} - \frac{1}{2}\right)\right]$$

Thus, in order to generate $\mathcal{X} \rightsquigarrow \mathcal{C}(\alpha,\beta)$, u must be generated from $\mathcal{U}(0,1)$ and then deliver $\boxed{\alpha + \beta\tan\left[\pi\left(u - \frac{1}{2}\right)\right]}$ as the generated value.

4 Discrete distributions random generation

In this section different procedures are presented in order to generate discrete distributions. The inverse transform method for discrete distributions described in 2.4 is used in order to generate a sample of the distribution. The only thing to do is to use the `random_discrete` function developed in the same section with the corresponding parameters.

4.1 Uniform discrete distribution

$\mathcal{X} = \{a, a+1, \ldots, b\}$ has an uniform discrete distribution with parameters a and b ($\mathcal{X} \rightsquigarrow \mathcal{U}_{\mathcal{D}}(a,b)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \frac{1}{b-a+1} \qquad ; \qquad x = a, a+1, \ldots, b$$

Thus, in order to generate a sample of size n from an $\mathcal{U}_{\mathcal{D}}(a,b)$, the following DERIVE code can be used:

```
random_uniform_discrete(n := 1, a := 0, b := 1) :=
    random_discrete(n,1/(b-a+1),a)
```

4.2 Bernouille distribution

CONTS

$\mathcal{X} = \{0, 1\}$ has a Bernouille distribution with parameter p ($\mathcal{X} \rightsquigarrow \mathcal{B}\text{er}(p)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = p^x (1-p)^{1-x} \qquad ; \qquad x = 0, 1$$

or, equivalently, $F(0) = 1 - p$ and $F(1) = p$.

Thus, in order to generate a sample of size n from a $\mathcal{B}\text{er}(p)$, the following DERIVE code can be used:

```
random_bernouille(n := 1, p := 1/2) := random_discrete(n,p^x(1 - p)^(1-x),0)
```

4.3 Binomial distribution

$\mathcal{X} = \{0, 1, \ldots, n\}$ has a binomial distribution with parameters n and p ($\mathcal{X} \rightsquigarrow \mathcal{B}\text{i}(n, p)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \binom{n}{x} p^x (1-p)^{n-x} \qquad ; \qquad x = 0, 1, \ldots, n$$

Thus, in order to generate a sample of size m from a $\mathcal{B}\text{i}(n, p)$, the following DERIVE code can be used:

```
random_binomial(m := 1, n := 100, p := 1/2) :=
    random_discrete(m, ncom(n,x) p^x(1 - p)^(n-x),0)
```

4.4 Poisson distribution

$\mathcal{X} = \{0, 1, 2, \ldots\}$ has a poisson distribution of parameter λ ($\mathcal{X} \rightsquigarrow \mathcal{P}(\lambda)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \frac{\mathrm{e}^{-\lambda}\ \lambda^x}{x!} \qquad ; \qquad x = 0, 1, 2, \ldots$$

Thus, in order to generate a sample of size n from a $\mathcal{P}(\lambda)$, the following DERIVE code can be used:

```
random_poisson(n := 1, λ := 1) := random_discrete(n, e^(-λ) λ^x / x!,0)
```

4.5 Geometric distribution

$\mathcal{X} = \{0, 1, 2, \ldots\}$ has a Geometric distribution with parameter p ($\mathcal{X} \rightsquigarrow \mathcal{G}\text{e}(p)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = p\ (1-p)^x \qquad ; \qquad x = 0, 1, 2, \ldots$$

Thus, in order to generate a sample of size n from a $\mathcal{G}\text{e}(p)$, the following DERIVE code can be used:

```
random_geometric(n := 1, p := 1/2) := random_discrete(n,p (1 - p)^x,0)
```

4.6 Negative Binomial distribution

CONTS

$\mathcal{X} = \{0, 1, 2, \ldots\}$ has a negative binomial distribution with parameters r and p ($\mathcal{X} \rightsquigarrow \mathcal{NB}(r, p)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \binom{r + x - 1}{x} p^r (1 - p)^x \quad ; \quad x = 0, 1, 2, \ldots$$

Thus, in order to generate a sample of size n from a $\mathcal{NB}(r, p)$, the following DERIVE code can be used:

```
random_negative_binomial(n := 1, r := 10, p := 1/2) :=
    random_discrete(n, ncom(r+x-1,x) p^r(1 - p)^x,0)
```

4.7 Hypergeometric distribution

$\mathcal{X} = \{\max(0, n_1 + m - n), \ldots, \min(n_1, m)\}$ has an hypergeometric distribution with parameters n , m and n_1 ($\mathcal{X} \rightsquigarrow \mathcal{H}(n, m, n_1)$) if its distribution mass function F is:

$$F(x) = P[\mathcal{X} = x] = \frac{\binom{n_1}{x}\binom{n - n_1}{m - x}}{\binom{n}{m}} \quad ; \quad x = \max(0, n_1 + m - n), \ldots, \min(n_1, m)$$

Thus, in order to generate a sample of size l from a $\mathcal{NB}(r, p)$, the following DERIVE code can be used:

```
random_hypergeometric(l:=1, n := 100, m := 50, n1 := 50) :=
    random_discrete(l, ncom(n1,x)ncom(n - n1,m - x) / ncom(n,m) ,max(0, n1 + m - n))
```

5 Approximative algorithms

In this section several algorithms used to generate different distributions using some kind of approximations are presented.

The main reason in order to use such algorithm is that they produce "goods" samples and they are quite more faster than other "exact" algorithm. In fact, some of the algorithms described above are approximative algorithm (`random_gamma9`, `random_beta7` and `random_chi_square2` are approximative algorithms for $\mathcal{G}(\alpha, \beta)$, $\mathcal{B}e(\alpha, \beta)$ and $\chi^2(k)$ distributions which use $\mathcal{N}(\mu, \sigma)$ distribution as approximation).

In the followings subsections some other approximative algorithms which have been implemented in `Random_distribution` package are developed.

5.1 Approximation to Binomial distribution by Poisson distribution

Let $\mathcal{X} \rightsquigarrow \mathcal{B}i(n, p)$. If p is "small" then

$$\mathcal{B}i(n, p) \approx \mathcal{P}(n\,p)$$

Thus, the algorithm to generate a sample of size m from the binomial distribution approximated by a poisson distribution is:

```
random_binomial_approx_poisson(m:=1, n := 100, p:= 0.01) :=
    random_poisson(m,np)
```

5.2 Approximation to Binomial distribution by Normal distribution

CONTS

Let $\mathcal{X} \rightsquigarrow \mathcal{B}\text{i}(n,p)$. If n is "large" and p is not close to 0 or 1, then

$$\mathcal{B}\text{i}(n,p) \approx \mathcal{N}\left(n\,p, \sqrt{n\,p\,(1-p)}\right)$$

It can be shown that "good" approximations are reached when $n\,p > 10$ with $0 << p < 0.5$ or $n\,(1-p) > 10$ with $0.5 < p << 1$ ($<<$ indicates less than but not close to).

On the other hand, as $\mathcal{B}\text{i}(n,p)$ has only nonnegative integer values, the nearest nonnegative integer to the value returned by $\mathcal{N}\left(n\,p, \sqrt{n\,p\,(1-p)}\right)$ must be chosen as the generated value for $\mathcal{X}$. It is easy to understand that the nearest nonnegative integer to a value z can be found by the operation:

$$\max\left(0, \text{floor}\left(z + \frac{1}{2}\right)\right)$$

where floor(x) is the integer part of x.

Thus, the algorithm to generate a sample of size m from the binomial distribution approximated by a normal distribution is:

```
random_binomial_approx_normal(m:=1, n := 100, p:= 1/4) :=
    vector(max(0,floor(k+1/2)),k,random_normal(m,np,sqrt(np(1-p))))
```

5.3 Approximation to Poisson distribution by Normal distribution

Let $\mathcal{X} \rightsquigarrow \mathcal{P}(\lambda)$. If $\lambda > 10$ then $\mathcal{P}(\lambda) \approx \mathcal{N}\left(\lambda, \sqrt{\lambda}\right)$

On the other hand, as $\mathcal{P}(\lambda)$ has only nonnegative integer values, the nearest nonnegative integer to the value returned by $\mathcal{N}\left(\lambda, \sqrt{\lambda}\right)$ must be chosen as the generated value for $\mathcal{X}$. It is easy to understand that the nearest nonnegative integer to a value z can be found by the operation:

$$\max\left(0, \text{floor}\left(z + \frac{1}{2}\right)\right)$$

where floor(x) is the integer part of x.

Thus, the algorithm to generate a sample of size n from the poisson distribution approximated by a normal distribution is:

```
random_poisson_approx_normal(n := 1, λ:= 15) :=
    vector(max(0,floor(k+1/2)),k,random_normal(n,λ,sqrt(λ)))
```

5.4 Approximation to Geometric distribution by Exponential distribution

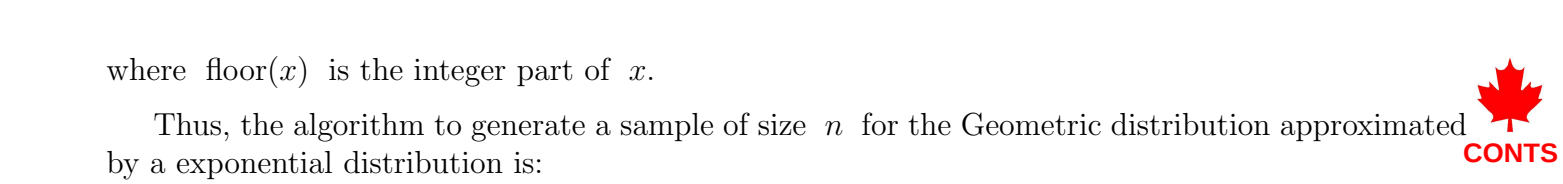
If $\mathcal{X} \rightsquigarrow \mathcal{G}\text{e}(p)$ then $\mathcal{X} \approx \mathcal{E}\left(\ln\left(\frac{1}{1-p}\right)\right)$

On the other hand, as $\mathcal{G}\text{e}(p)$ has only nonnegative integer values, the nearest nonnegative integer to the value returned by $\mathcal{E}\left(\ln\left(\frac{1}{1-p}\right)\right)$ must be chosen as the generated value for $\mathcal{X}$. The nearest nonnegative integer to a value $z > 0$ (exponential distribution has only positive real numbers) can be found by the operation:

$$\text{floor}\left(z + \frac{1}{2}\right)$$

where floor(x) is the integer part of x.

Thus, the algorithm to generate a sample of size n for the Geometric distribution approximated by a exponential distribution is:




```
random_geometric_approx_exponential(n := 1, p:= 1/2) :=
    vector(floor(k+1/2),k,random_exponential(n,ln(1/(1-p))))
```

References

[Galán, 1991] Galán, J. L. (1991). *Simulación de Variables Aleatorias*. Proyecto Fin de Carrera, Universidad de Málaga.

[Marsaglia and Zaman, 1994] Marsaglia, G. and Zaman, A. (1994). Some portable very-long-period random number generators. *Computers in Physics*, 8(1):117–121.

[Press and Teukolsky, 1992] Press, W. H. and Teukolsky, S. A. (1992). Portable Random Number Generators. *Computers in Physics*, 6(5):522–524.

[Press et al., 1999] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1999). *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, United States.

[Rubinstein, 1981] Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo Method*. John Wiley & Sons, New York.