
# Estatística Computacional

## Universidade Federal da Bahia

Gilberto Pereira Sassi

Tópico 5

# Pacotes que iremos usar na semana 6

```
library(readxl)
library(readODS)
library(writexl)
library(ggthemes)
library(lvplot)
library(glue)
library(tidyverse)

library(plotly) # gráfico 3d
library(randtests)
library(goft)
```

# Gerando número aleatórios

- Geralmente, *números aleatórios* são gerados por algoritmos matemáticos
- Algoritmos matemáticos são determinísticos, e usamos o termo *pseudoaleatório*
- Amostra construída com números *pseudoaleatórios*: pseudo-amostra
- Geração de números aleatórios de distribuições de probabilidade é baseada na geração de números aleatórios da distribuição uniform
- É possível coletar valores aleatórios da distribuição uniforme que *verdadeiramente* aleatório usando fenômenos naturais (como ruídos atmosféricos e o magnetismo da terra).
- Em geral, evitamos o uso de números *verdadeiramente* aleatórios em favor dos número *pseudoaleatórios*: números *pseudoaleatórios* são reprodutíveis
- Números *pseudoaleatórios* são baseados em um valor chamado *chave* ou *semente*
- Uma sequência de números *pseudoaleatórios* geralmente é periódica

# Gerador de números *pseudoaleatórios*

**Definição 1 (Gerador de números pseudoaleatórios)** Um número gerador pseudoaleatório é uma estrutura $\Xi = (S, s_0, T, U, G)$, $S$ é um conjunto de números (chamados *estados*), $s_0$ é a *chave* ou *semente*, $T : S \to S$ é uma função (chamada de *função de transformação*), $U$ é um conjunto de números (chamados *símbolos de saída*), e $G : S \to U$ é uma função (chamada de *função de saída*).

- $s_n = T(s_{n-1}), n = 1, 2, 3, \ldots$
- Como $|S| < \infty$, então existe $0 \leq i < j$ tal que $s_i = s_j$ e $s_{i+n} = s_{j+n}, \forall n \geq 0$
- $s_{i+n} = s_{j+n} \Rightarrow s_{i+n} = s_{i+(j-i)+n}, \forall n \geq 0 \Rightarrow s_m \geq s_{(j-i)+m}, \forall m \geq i$
- **Período:** menor inteiro $p$ tal que $s_{n+p} = s_n$ para $n \geq r$ para algum $r \geq 0$
- Note que $p \leq |S|$
- **Desejável:** $p \approx |S|$ (poucas repetições na sequência de números pseudoaleatórios)

# Gerador congruente linear

**Definição 2 (Gerador Congruente Linear)** Seja $m > 0$, $0 < a < m$, $0 \leq c \leq m$ e $0 \leq s_0 < m$. Considere o gerador pseudoaleatório $\Xi = (S, s_0, T, U, G)$ onde

$$T(s_i) = (a \cdot s_{i-1} + c) \mod m, \quad 0 \leq s_i < m, \quad i = 0, 1, 2, 3, \ldots.$$

Chamamos $a$ de multiplicador, $m$ de módulo, $c$ de incremento, $s_0$ de *chave* ou *semente* e $\Xi$ é denominado de **Gerador Congruente Linear**.

---

O *gerador congruente linear* mais famoso é o `randu` proposto em 1951 e implementado pela IBM logo em seguida.

# RANDU

**Definição 3 (RANDU)** Considere o *Gerador Congruente Linear* $\Xi$, em que $S = \mathbb{Z} \cap [0, 2^{31}]$, $s_0 \in S$, $T(s) = (2^{16} + 3)s \mod 2^{31}$, $U = \frac{S}{2^{31}}$ e $G(s) = \frac{s}{2^{31}}$.

---

**Restrições**

Suponha que $s_{k+1} < 2^{31}$, então

$$\begin{aligned} s_{k+2} &= (2^{16} + 3)s_{k+1} = (2^{16} + 3)(2^{16} + 3)s_k, \\ &= (2^{32} + 6 \cdot 2^{16} + 9)s_k = 6(2^{16} + 3)s_k - 9s_k, \\ &= 6s_{k+1} - 9s_k. \end{aligned}$$

Ou seja, existe uma relaçãos simples entre $x_{k+2}$, $x_{k+1}$ e $x_k$, e podemos descobrir o *gerador pseudoaleatório* a partir da sequência de número pseudoaleatórios.

# RANDU

```
randu <- function(n, seed = 1) {
  x <- vector(mode = "double", n)
  a <- 2^16 + 3
  m  <- 2^31
  x[1] <- (a * seed) %% m
  for (i in 2:n) x[i]  <- (a * x[i - 1]) %% m
  x / m
}
```

Marsaglia (1968) provou os valores desta matriz estão dentro de 15 hiperplanos.

Vamos usar o teste de aleatoriedade (Wald e Wolfowitz 1940): `runs.test()` do pacote `randtests`.

# Criando uma amostra com RANDU

```
tamanho_amostra <- 9999
amostra <- randu(tamanho_amostra)
matriz <- tibble(
  xk = amostra[seq_len(tamanho_amostra) %% 3 == 0],
  xk1 = amostra[seq_len(tamanho_amostra) %% 3 == 1],
  xk2 = amostra[seq_len(tamanho_amostra) %% 3 == 2]
)
```

# Checando a amostra

```
ks.test(amostra, "punif")
```

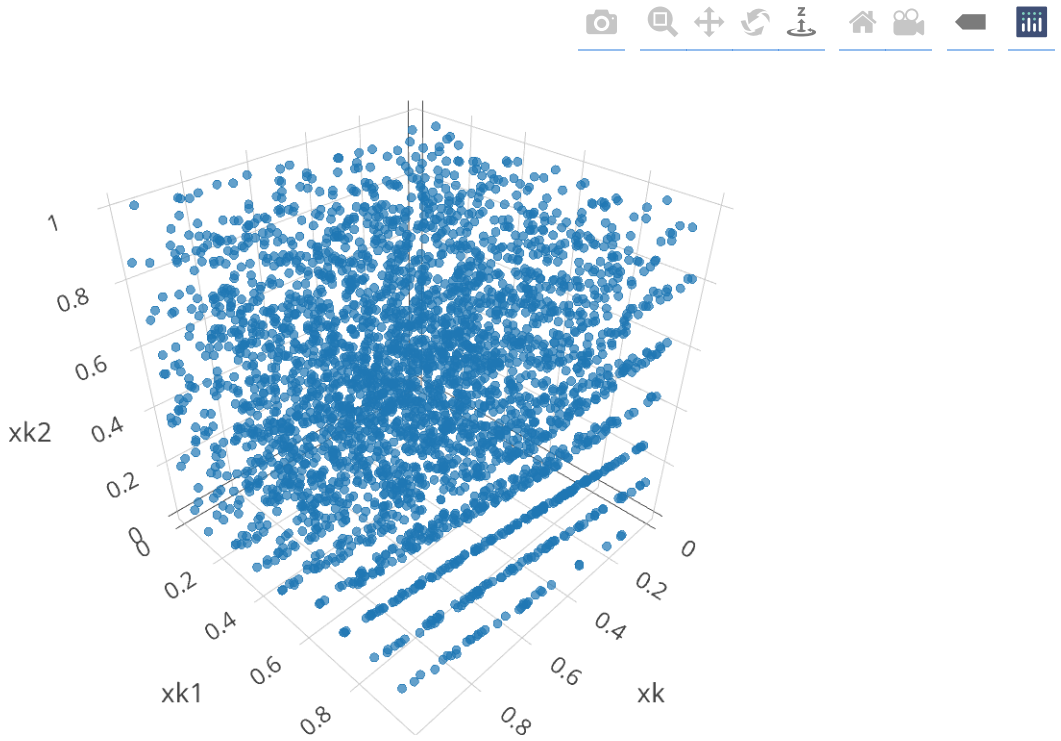
```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  amostra
## D = 0.0063368, p-value = 0.8168
## alternative hypothesis: two-sided
```

```
runs.test(amostra)
```

```
##
##  Runs Test
##
## data:  amostra
## statistic = -2.0203, runs = 4899, n1 = 4999, n2 = 4999, n = 9998, p-value = 0.04335
## alternative hypothesis: nonrandomness
```

# Gráfico com RANDU

```
plot_ly(matriz, x = ~xk, y = ~xk1, z = ~xk2, size = 0.2)
```



# Gerador Defasado de Fibonacci

**Definição 4 (Gerador Defasado de Fibonacci)** Seja $i, j, k$ números inteiros tal que $0 < k < j < i$ , e considere o *Gerador Congruente Linear* $\Xi$ com em que $S = \mathbb{Z} \cap [0, 2^{31}], T(s_i) = (s_{i-j} + s_{j-k}) \mod 2^{31}, U = \frac{S}{2^{31}}, G(s) = \frac{s}{2^{31}}$ e $s_0$ é um conjunto de valores iniciais tal que $|s_0| > j$. Chamamos $\Xi$ de *Gerador Desafado de Fibonacci*.

Este algoritmo é uma melhoria sobre o `randu`, e pode ser usado para simulações. (Mas não é recomendado para criptografia).

O `R` implementou o algoritmo *Mersenne Twister* (vide Härdle, Okhrin, e Okhrin 2017 para mais detalhes) com a função `runif(n)`.

# GDF

```r
k <- 5
j <- 17
m <- 2^31
semente <- runif(j, 0, m) |> round()
gdf <- function(n, seed = semente) {
  x <- seed
  for (i in seq_len(n)) {
    x[j + i] <- (x[i] + x[j + i - k]) %% m
  }
  x[(j + 1):length(x)] / m
}
```

# Amostra

```
tamanho_amostra <- 9999
amostra <- gdf(tamanho_amostra)
matriz <- tibble(
  xk = amostra[seq_len(tamanho_amostra) %% 3 == 0],
  xk1 = amostra[seq_len(tamanho_amostra) %% 3 == 1],
  xk2 = amostra[seq_len(tamanho_amostra) %% 3 == 2]
)
```

# Checando a amostra

```
ks.test(amostra, "punif")
```

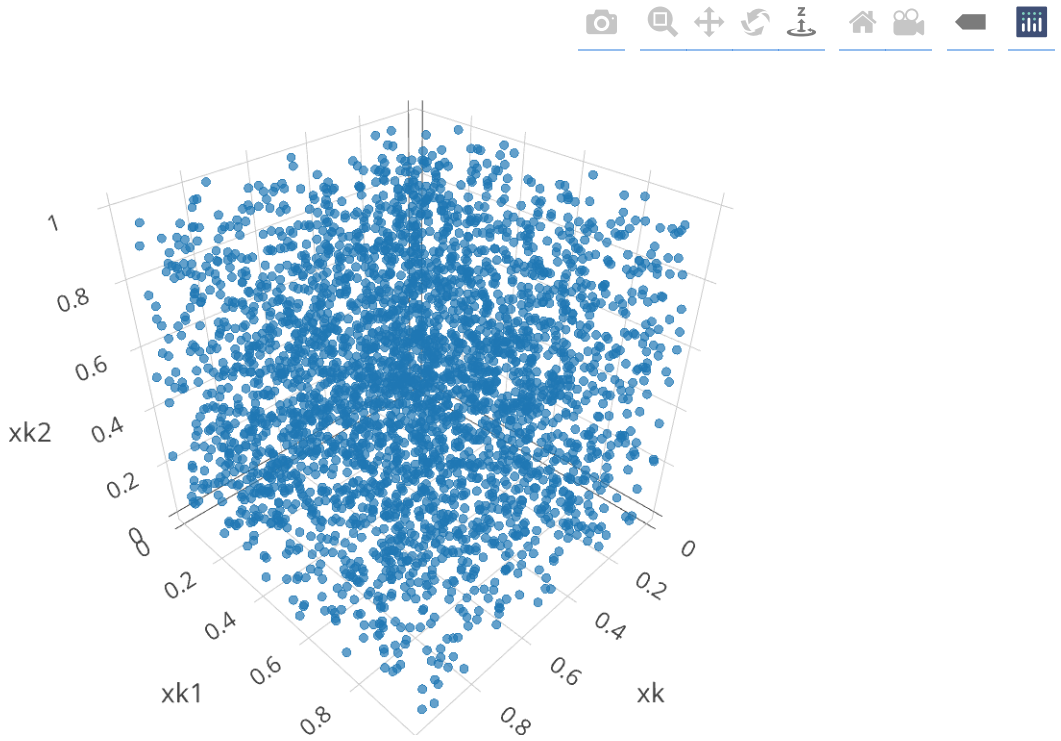
```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  amostra
## D = 0.0082316, p-value = 0.507
## alternative hypothesis: two-sided
```

```
runs.test(amostra)
```

```
##
##  Runs Test
##
## data:  amostra
## statistic = 1.4002, runs = 5070, n1 = 4999, n2 = 4999, n = 9998, p-value = 0.1615
## alternative hypothesis: nonrandomness
```

# Gráfico com GDF

```
plot_ly(matriz, x = ~xk, y = ~xk1, z = ~xk2, size = 0.2)
```



# Métodos para geração de outras distribuições

## Método da Transformação Inversa

**Definição 5 (Inversa Generalizada)** Seja $F$ uma função de distribuição acumulada, chamamos $F^{-}(u) = \inf\{x \mid F(x) \geq u\}, \forall u \in [0,1]$ de *função inversa generalizada*.

**Proposição 1 (Método da transformação inversa.)** Se $U \sim U(0,1)$, então $F^{-}(U)$ é uma variável aleatória com FDA $F$.

# Método da Transformação Inversa

*Prova.* Note que

- $F(z) \geq u \Rightarrow z \in \{x \mid F(x) \geq u\} \Rightarrow z \geq F^{-}(u) \Rightarrow [F(z) \geq U] \subset [z \geq F^{-}(U)]$
- $F^{-}(u) \leq z \Rightarrow u \leq F(z) \Rightarrow [F^{-}(U) \leq z] \subset [U \leq F(z)]$

Então, temos que $[F^{-}(U) \leq z] = [U \leq F(z)]$ e consequentemente temos que

$$P(F^{-}(U) \leq z) = P(U \leq F(z)) = F(z),$$

$F^{-}(U)$ é uma variável aleatória com função de distribuição acumulada dada por $F$.

# Método da Transformação Inversa

## Algoritmo

1. Encontre $F^-(u)$
2. Gere um valor $u$ de $U \sim U(0,1)$
3. Calcule $F^-(u)$

## Como encontrar $F^-$

- Se $X$ é uma variável aleatória contínua: $F^{-1} = F^-$
- Se $X$ é uma variável alatória discreta com suporte $\chi = \{x_1, \ldots, x_k, \ldots\}$:

$$F^-(u) = \begin{cases} x_i, & F(x_{i-1}) < u \leq F(x_i), \\ \min(\chi), & u \leq F(\min(\chi)) \end{cases}$$

# Método da Transformação Inversa

Exemplo (Caso Discreto):

- Seja $X$ uma variável aleatória discreta com suporte $\chi = \{0, 1, 2\}$ com função de probabilidade $f(0) = 0,3$, $f(1) = 0,2$ e $f(2) = 0,5$.
- $$F(x) = \begin{cases} 0, & x < 0, \\ 0,3, & 0 \leq x < 1, \\ 0,5, & 1 \leq x < 2, \\ 1, & x \geq 2 \end{cases}$$
- $$F^{-}(u) = \begin{cases} 0, & u \leq 0,3, \\ 1, & 0,3 < u \leq 0,5 \\ 2, & 0,5 < u \leq 1 \end{cases}$$

# Método da Transformação Inversa

## Exemplo (Caso Discreto):

```r
ex_discreto <- function(n) {
  if (length(n) > 1) stop("n precisa ser um valor escalar inteiro.")
  seq_len(n) |>
    map_dbl(\(x) {
      u <- runif(1)

      if(u <= 0.3) {
        return(0)
      } else if (0.3 < u && u <= 0.5) {
         return(1)
      } else if (u > 0.5) {
         return(2)
      }
  })
}
```

# Método da Transformação Inversa

## Exemplo (Caso Discreto):

```
set.seed(121343)
tibble(amostra = ex_discreto(1000)) |>
  group_by(amostra) |>
  summarise(freq = n()) |>
  mutate(fr = freq / sum(freq))
```

```
## # A tibble: 3 × 3
##   amostra  freq    fr
##     <dbl> <int> <dbl>
## 1       0   304 0.304
## 2       1   195 0.195
## 3       2   501 0.501
```

# Método da Transformação Inversa

## Exemplo (Caso Contínuo):

- Seja $X \sim Exp(\lambda)$ com função densidade $f(x) = \lambda \exp(-\lambda x), x > 0$ e $F(x) = 1 - \exp(-\lambda x), x > 0$
- $$F^{-}(u) = \begin{cases} \frac{-\ln(1-u)}{\lambda}, & u > 0, \\ 0, & u \leq 0, \end{cases}$$

```r
lambda <- 2
exp_2 <- function(n) {
  if (length(n) > 1) stop("n precisa ser um valor escalar inteiro.")
  seq_len(n) |>
    map_dbl(\(x) {
      u <- runif(1)

      if (u <= 0) {
         return(0)
      } else {
         return(-log(1 - u) / lambda)
      }
    })
}
```

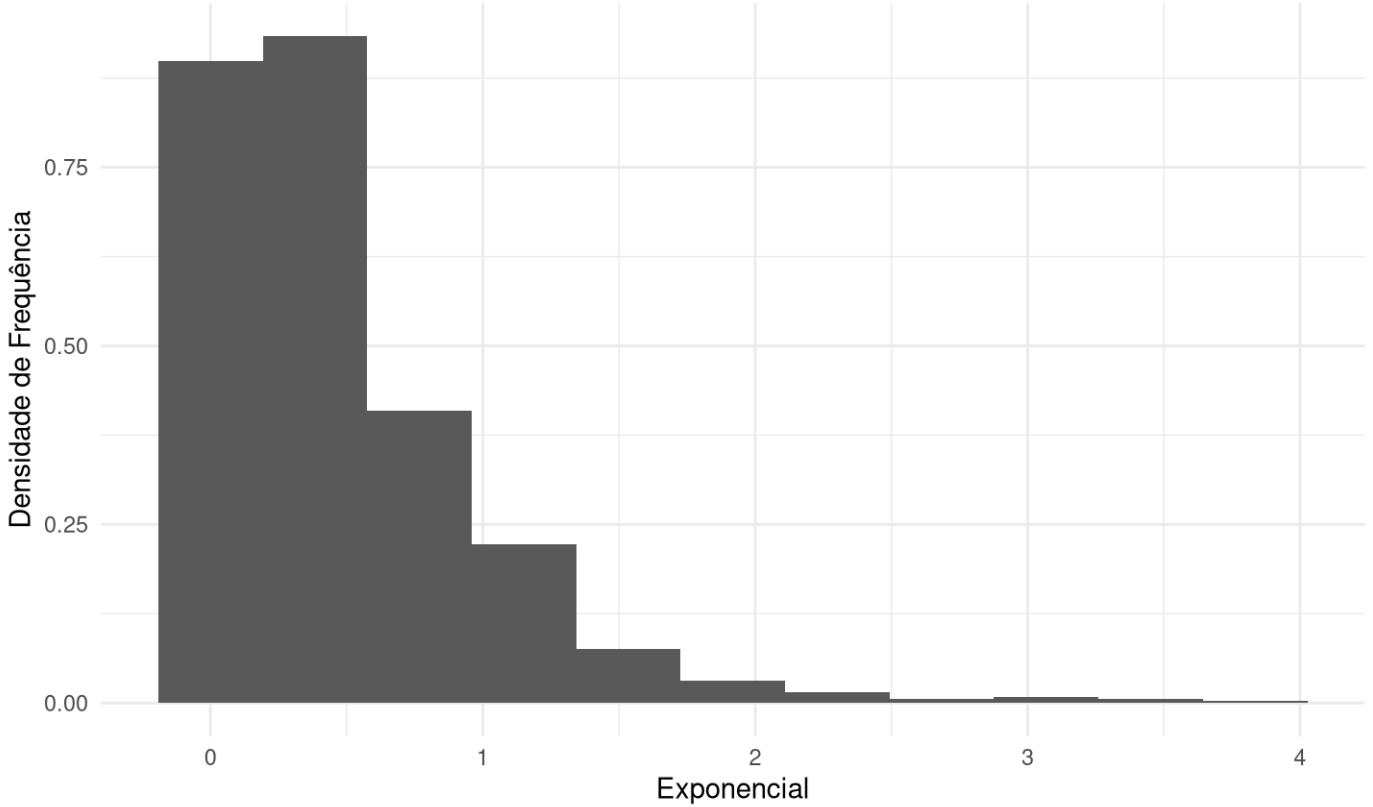
# Método da Transformação Inversa

## Exemplo (Caso Contínuo):

```
n <- 1000
k <- (1 + log2(n)) |> round()
dados <- tibble(amostra = exp_2(n))
ggplot(data = dados) +
  geom_histogram(mapping = aes(x = amostra, y = ..density..), bins = k) +
  theme_minimal() +
  labs(x = "Amostra", y = "Densidade de Frequência")
```

# Método da Transformação Inversa

Exemplo (Caso Contínuo):



```r
ks.test(dados$amostra, pexp, rate = 2)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  dados$amostra
## D = 0.03783, p-value = 0.1143
## alternative hypothesis: two-sided
```

# Método de Aceitação-Rejeição

## Ideia

- Difícil gerar valores de $X \sim f$
- Simples gerar valores de $Y \sim g$
- $X$ e $Y$ têm o mesmo suporte
- $g(x)$ *cobre* $f(x)$: $\exists c > 0$ tal que $f(x) \leq c \cdot g(x)$
- Gerar valores de $X$ a partir de $Y$

# Método de Aceitação-Rejeição

## Algoritmo

1. Encontre $Y$ e $c > 0$ tal que $f(x) \leq c \cdot g(x)$
2. Gere um valor $y$ de $Y$
3. Gere um valor $u$ de $U \sim U(0, 1)$
4. Se $u \leq \frac{f(y)}{c \cdot g(y)}$, então aceitamos $y$ como valor gerado de $X$. Se $u > \frac{f(y)}{c \cdot g(y)}$, repetimos 2. e 3.

Geralmente, usamos $c$ como sendo o valor máximo de $\frac{f(x)}{g(x)}$.

# Método de Aceitação-Rejeição

## Exemplo (Caso Contínuo)

- $X \sim B(2,1), \chi = (0,1), f(x) = 2 \cdot x, x \in (0,1)$ e $F(x) = x^2, x \in (0,1)$
- Vamos considerar $Y \sim U(0,1)$ com $g(y) = x, x \in (0,1)$
- $\frac{f(x)}{g(x)} = 2 = c$

```
beta_21 <- function(n) {
  if (length(n) > 1) stop("'n' precisa ser um escalar inteiro.")
  seq_len(n) |>
    map_dbl(\(i) {
      y <- runif(1); u <- runif(1)

      while(u > dbeta(y, shape1 = 2, shape2 = 1) / (2 * dunif(y))) {
        y <- runif(1); u <- runif(1)
      }

      return(y)
    })
}
```

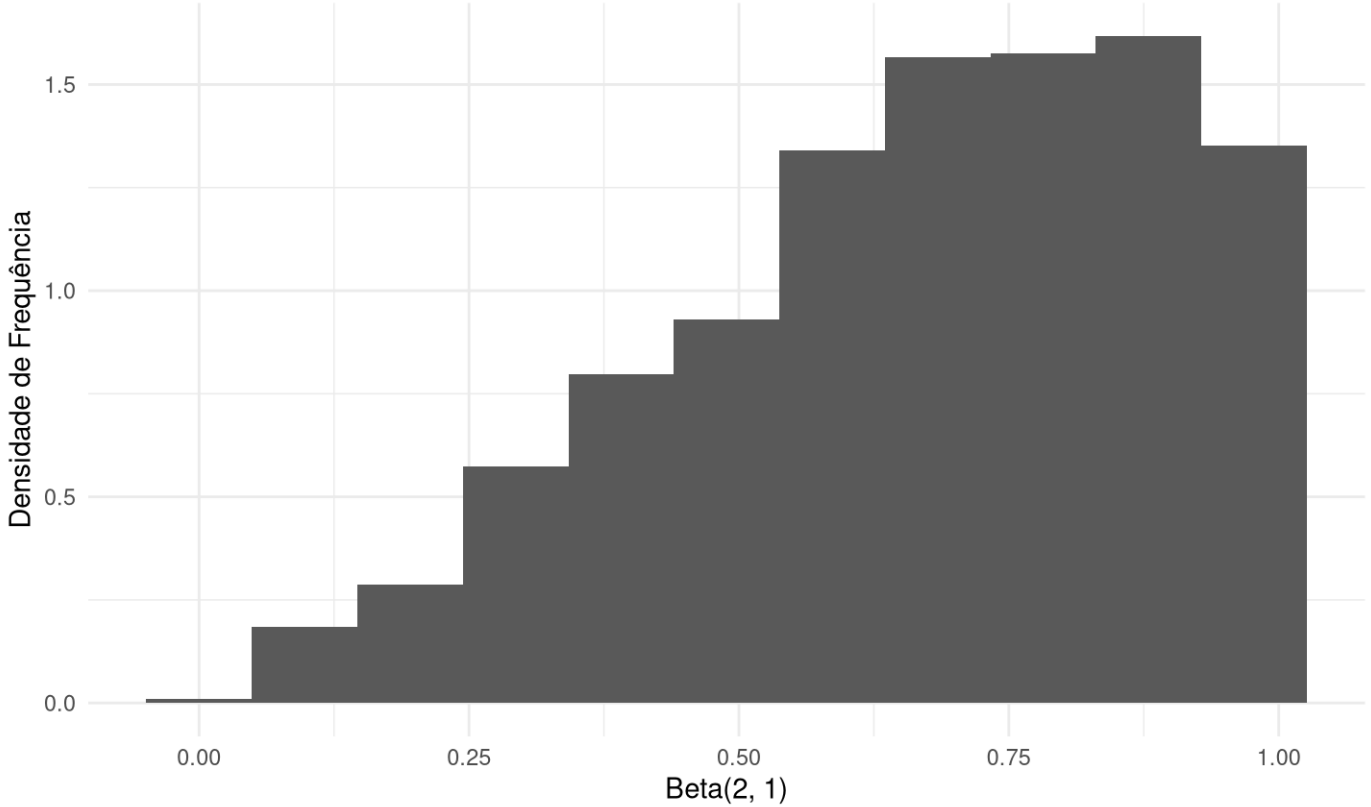
# Método de Aceitação-Rejeição

## Exemplo (Caso Contínuo)

```
n <- 1000
k <- (1 + log2(n)) |> round()
dados <- tibble(amostra = beta_21(n))
ggplot(data = dados) +
  geom_histogram(mapping = aes(x = amostra, y = ..density..), bins = k) +
  theme_minimal() +
  labs(x = "Beta(2, 1)", y = "Densidade de Frequência")
```

# Método de Aceitação-Rejeição

## Exemplo (Caso Contínuo)



```r
ks.test(dados$amostra, pbeta, shape1 = 2, shape2 = 1)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  dados$amostra
## D = 0.030908, p-value = 0.295
## alternative hypothesis: two-sided
```

# Método de Aceitação-Rejeição

## Exemplo (Caso Discreto)

- Seja $X$ com $\chi = \{1, 2, 3, 4, 5\}$ com

| $x$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $f(x)$ | $0,15$ | $0,22$ | $0,33$ | $0,10$ | $0,20$ |

- Seja $Y$ com $\chi = \{1, 2, 3, 4, 5\}$ com $g(x) = 0,20, \quad x \in \chi$
- O valor máximo do conjunto $\{\frac{f(x)}{g(x)} \mid x \in \chi\}$ é $c = 1,65$

# Método de Aceitação-Rejeição

## Algoritmo – caso discreto

```
fp <- c(0.15, 0.22, 0.33, 0.10, 0.20)
const <- 1.65
mar_discreto <- function(n) {
  if (length(n) > 1) {
     stop("'n' precisa um escalar inteiro.")
  }
  seq_len(n) |>
    map_dbl(\(i) {
      y <- sample.int(5, 1); u <- runif(1)

      while(u > fp[y] / (0.2 * const)) {
        y <- sample.int(5, 1); u <- runif(1)
      }

      return(y)
    })
}
```

# Método de Aceitação-Rejeição

## Algoritmo – caso discreto

```
n <- 1000
dados <- tibble(amostra = mar_discreto(n))
print(fp)
```

```
## [1] 0.15 0.22 0.33 0.10 0.20
```

```
dados |>
  group_by(amostra) |>
  summarise(freq = n()) |>
  mutate(fr = freq / sum(freq))
```

```
## # A tibble: 5 × 3
##   amostra  freq    fr
##     <dbl> <int> <dbl>
## 1       1   146 0.146
## 2       2   211 0.211
## 3       3   340 0.34
## 4       4    98 0.098
## 5       5   205 0.205
```

# Amostrando das principais distribuições de probabilidade

# Distribuição normal

## Algoritmo Box-Müller

Sejam $U_1 \sim U(0,1)$ e $U_2 \sim U(0,1)$ duas variáveis aleatórias contínuas independentes, e considere

$$X_1 = \sqrt{-2\ln(U_1)}\cos(2\pi U_2),$$
$$X_2 = \sqrt{-2\ln(U_1)}\sin(2\pi U_2),$$

então $X_1$ e $X_2$ são independentes, e $X_1 \sim N(0,1)$ e $X_2 \sim N(0,1)$.

- Neave (1973) provou que o algoritmo de Box-Müller quando usado com *randu* não tem distribuição normal padrão.

# Distribuição normal

## Algoritmo Box-Müller

```
box_muller <- function(n) {
  if (length(n) > 1) stop("'n' precisa ser um escalar inteiro.")
  seq_len(n) |>
    map_dbl(\(i) {
      u1 <- runif(1); u2 <- runif(1)
      sqrt(-2 * log(u1)) * cos(2 * pi * u2)
    })
}
```

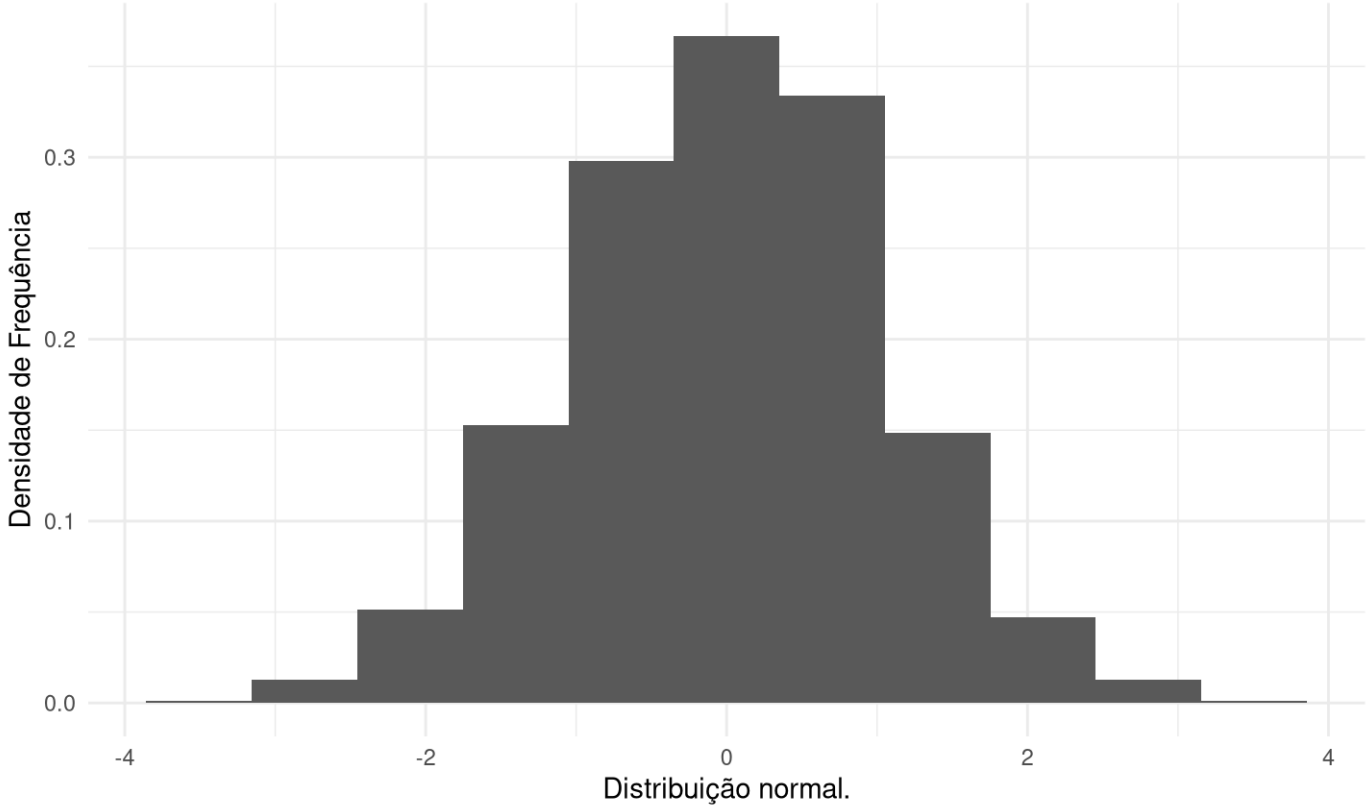
# Distribuição normal

## Exemplo – Box-Müller

```
n <- 1000
k <- (1 + log2(n)) |> round()
dados <- tibble(amostra = box_muller(n))
ggplot(data = dados) +
  geom_histogram(mapping = aes(x = amostra, y = ..density..), bins = k) +
  theme_minimal() +
  labs(x = "Distribuição normal.", y = "Densidade de Frequência")
```

# Distribuição normal

## Exemplo – Box-Müller



```
ks.test(dados$amostra, pnorm)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  dados$amostra
## D = 0.022602, p-value = 0.6866
## alternative hypothesis: two-sided
```

# Distribuição normal

## Método polar

Sejam $U_1 \sim U(-1,1)$ e $U_2 \sim U(-1,1)$ duas variáveis aleatórias contínuas independentes. Suponha que $\omega = U_1^2 + U_2^2 \leq 1$ , e considere

$$X_1 = \sqrt{\frac{-2\ln(\omega)}{\omega}} U_1,$$
$$X_2 = \sqrt{\frac{-2\ln(\omega)}{\omega}} U_2.$$

Então é possível provar que $X_1$ e $X_2$ são duas variáveis aleatórias independentes, e $X_1 \sim N(0,1)$ e $X_2 \sim N(0,1)$.

Para gerar valores de $U(-1,1)$ usamos o método da transformação inversa. Note que $2 \cdot U - 1 \sim U(-1,1)$ com $U \sim U(0,1)$.

# Distribuição normal

## Exemplo – método polar

```
metodo_polar <- function(n) {
   if (length(n) > 1) stop("'n' precisa ser um escalar inteiro.")
   seq_len(n) |>
    map_dbl(\(i) {
      u1 <- 2 * runif(1) - 1; u2 <- 2 * runif(1) - 1; w <- u1^2 + u2^2

      while(w > 1) {
        u1 <- 2 * runif(1) - 1; u2 <- 2 * runif(1) - 1; w <- u1^2 + u2^2
      }

      return(sqrt(-2 * log(w) / w) * u1)
    })
}
```

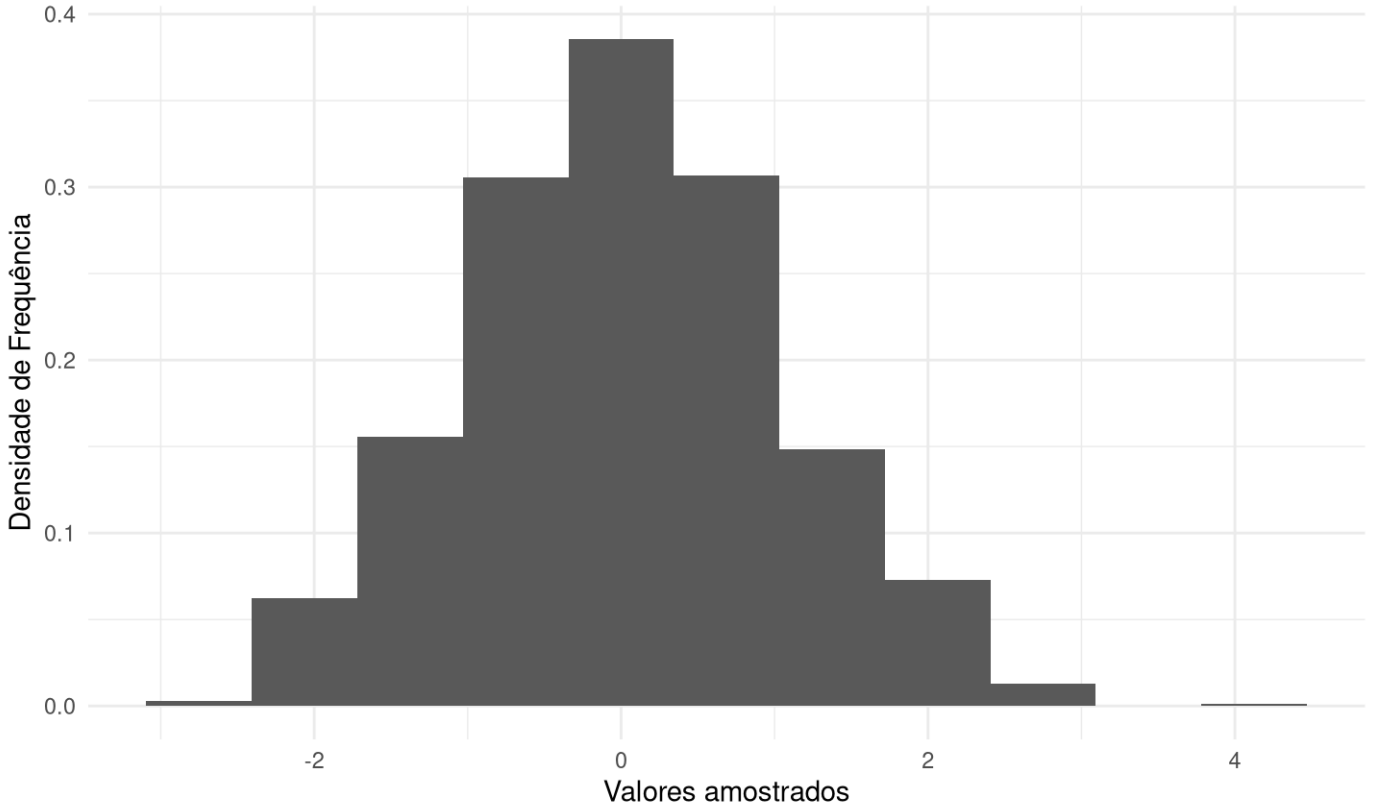
# Distribuição normal

## Exemplo – método polar

```
n <- 1000
k <- (1 + log2(n)) |> round()
dados <- tibble(amostra = metodo_polar(n))
ggplot(data = dados) +
  geom_histogram(mapping = aes(x = amostra, y = ..density..), bins = k) +
  theme_minimal() +
  labs(x = "Valores amostrados", y = "Densidade de Frequência")
```

# Distribuição normal

## Exemplo – método polar



```
ks.test(dados$amostra, pnorm)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  dados$amostra
## D = 0.021913, p-value = 0.7229
## alternative hypothesis: two-sided
```

# Distribuição gamma

- $X \sim \Gamma(\alpha, \beta)$ com função densidade $f(x) = \frac{1}{\beta \cdot \Gamma(\alpha)} x^{\alpha-1} \exp\left(-\frac{x}{\beta}\right)$
- Se $Y \sim \Gamma(\alpha, 1)$, então $\beta \cdot Y \sim \Gamma(\alpha, \beta)$

## Ideia

- Algoritmos para $0 < \alpha \leq 1$
- Algoritmos para $\alpha > 1$

# Distribuição gamma

$\alpha > 1$: proposto por Cheng (1977)

- **Passo 0)** Calcule $a = \sqrt{\frac{1}{2\alpha - 1}}$, $b = \alpha - \ln(4)$ e $c = \alpha + \frac{1}{a}$
- **Passo 1)** Gere valores $u_1 \sim U(0,1)$ e $u_2 \sim U(0,1)$
- **Passo 2)** Calcule $v = a \cdot \log\left(\frac{u_1}{1-u_1}\right)$ e $x = \alpha \exp(v)$
- **Passo 3)** Se $b + cv - x \geq \ln(u_1^2 u_2)$, então aceite que $x$ é um valor gerado de $\Gamma(\alpha, 1)$. Caso contrário, repita passos 1) e 2)
- **Passo 4)** Calcule $\beta \cdot x$

# Distribuição gamma

## $0 < \alpha \leq 1$: proposto por Ahrens e Dieter (1974)

- **Passo 0)** Calcule $b = \frac{\alpha + e}{e}$, em que $e = \exp(1)$
- **Passo 1)** Gere um valor $u_1 \sim U(0, 1)$ e calcule $p = b \cdot u_1$
- **Passo 2)** Gere um valor $u_2 \sim U(0, 1)$ e calcule $E = -\ln(u_2)$
- **Passo 3)**
  - Se $p \leq 1$, então calcule $x = p^{\frac{1}{\alpha}}$. Se $x \leq E$, então aceite que $x$ é um valor gerado de $\Gamma(\alpha, 1)$. Caso contrário, repita os passo 1) e 2).
  - Se $p > 1$, então calcule $x = -\ln\left(\frac{b-p}{\alpha}\right)$. Se $(1 - \alpha)\ln(x) \leq x$, então aceite que $x$ é um valor gerado de $\Gamma(\alpha, 1)$. Caso contrário, repita os passos 1) e 2).

# Distribuição gamma

## Exemplo: $\alpha > 1$

```
gamma_alpha_grande <- function(n, shape, scale) {
   if (length(n) > 1) stop("'n' precisa ser um escalar inteiro.")
   if (shape <= 1) stop("'shape' precisa ser um escalar maior que 1.")
   if (scale < 0) stop("'scale' precisa ser um escalar positivo.")
   seq_len(n) |>
    map_dbl(\(i) {
      a  <- 1 / sqrt(2 * shape - 1); b <- shape - log(4)
      ce <- shape + 1 / a

      u1 <- runif(1); u2 <- runif(1)
      v <- a * log(u1 / (1 - u1)); x <- shape * exp(v)

      while(b + ce * v - x < log(u1^2 * u2)) {
        u1 <- runif(1); u2 <- runif(1)
        v <- a * log(u1 / (1 - u1)); x <- shape * exp(v)
      }

      return(scale * x)
    })
}
```

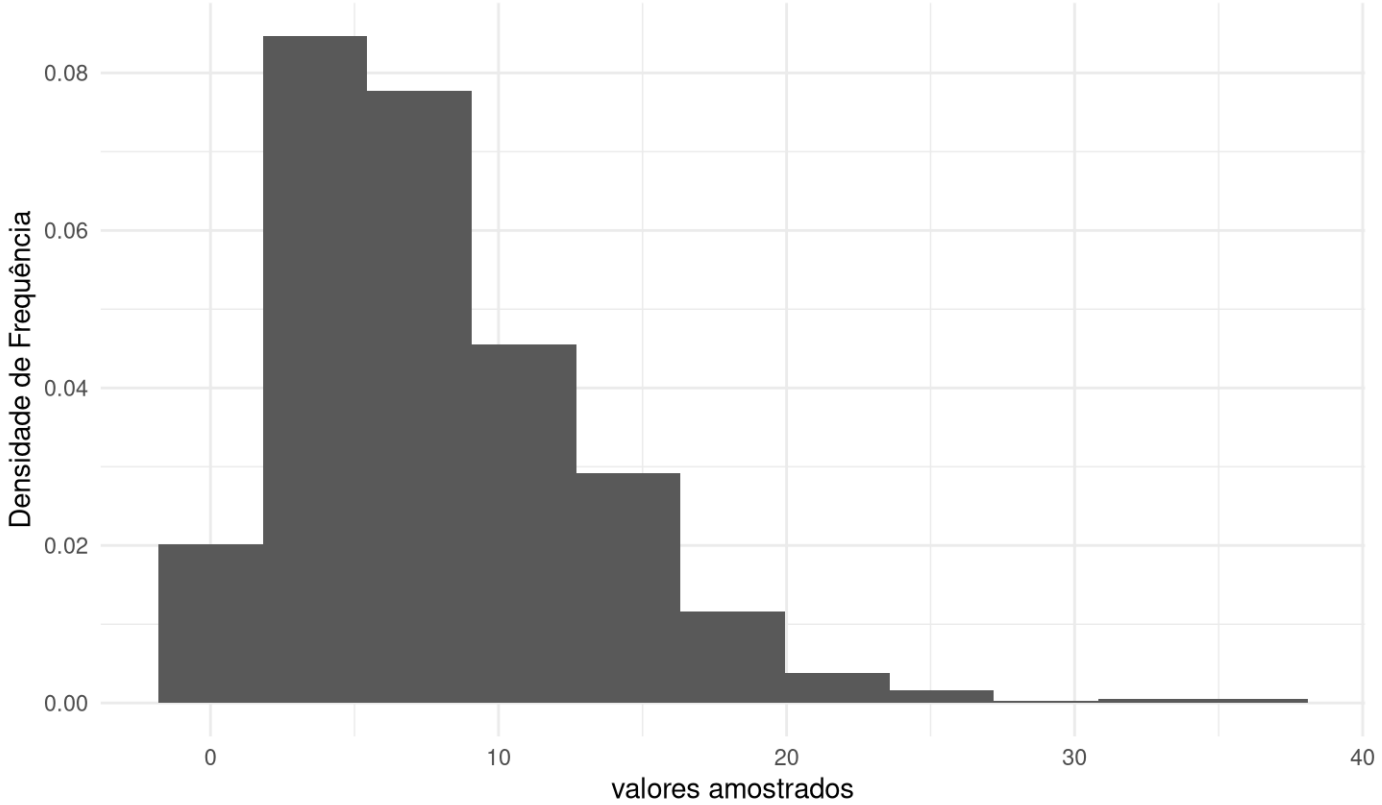
# Distribuição gamma

## Exemplo: $\alpha > 1$

```
n <- 1000
k <- (1 + log2(n)) |> round()
dados <- tibble(amostra = gamma_alpha_grande(n, shape = 2, scale = 4))
ggplot(data = dados) +
  geom_histogram(mapping = aes(x = amostra, y = ..density..), bins = k) +
  theme_minimal() +
  labs(x = "valores amostrados", y = "Densidade de Frequência")
```

# Distribuição gamma

Exemplo: $\alpha > 1$



```r
ks.test(dados$amostra, pgamma, shape = 2, scale = 4)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  dados$amostra
## D = 0.020675, p-value = 0.7861
## alternative hypothesis: two-sided
```

# Distribuição gamma

## Exemplo: $0 < \alpha \leq 1$

```
gamma_alpha_pequeno <- function(n, shape, scale) {
  seq_len(n) |>
    map_dbl(\(i) {
      b <- (shape + exp(1)) / exp(1); u1 <- runif(1); u2 <- runif(1); p <- b * u1; E <- -log(u2)
      laco <- TRUE
      while(laco) {
        if (p <= 1) {
          x <- p^(1 / shape)
          if (x > E) {
              b <- (shape + exp(1)) / exp(1); u1 <- runif(1); u2 <- runif(1); p <- b * u1; E <- -log(u2)
          } else {
            laco <- FALSE
          }
        } else {
          x <- -log((b - p) / shape)
          if ((1 - shape) * log(x) > E) {
              b <- (shape + exp(1)) / exp(1); u1 <- runif(1); u2 <- runif(1); p <- b * u1; E <- -log(u2)
          } else {
            laco <- FALSE
          }
        }
      }
      return(scale * x)
    })}
```

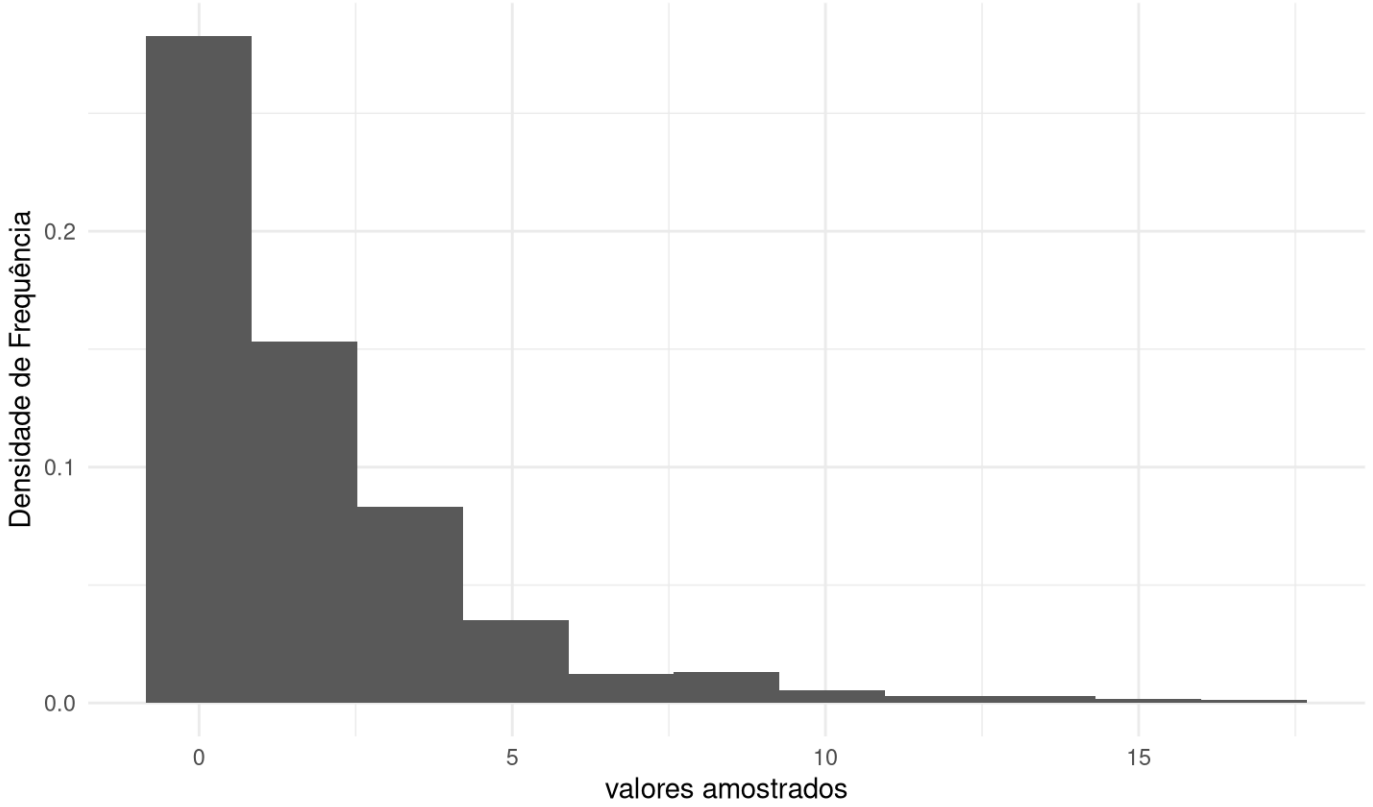
# Distribuição gamma

## Exemplo: $0 < \alpha \leq 1$

```
n <- 1000
k <- (1 + log2(n)) |> round()
dados <- tibble(amostra = gamma_alpha_pequeno(1000, shape = 0.5, scale = 4))
ggplot(data = dados) +
  geom_histogram(mapping = aes(x = amostra, y = ..density..), bins = k) +
  theme_minimal() +
  labs(x = "valores amostrados", y = "Densidade de Frequência")
```

# Distribuição gamma

Exemplo: $0 < \alpha \leq 1$



```r
ks.test(dados$amostra, pgamma, shape = 0.5, scale = 4)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  dados$amostra
## D = 0.024642, p-value = 0.5782
## alternative hypothesis: two-sided
```

# Distribuição beta (Atkinson e Pearce 1976)

Seja $X \sim B(\alpha, \beta)$ com função densidade dada por $f(x) = \frac{\Gamma(\alpha)\Gamma(b)}{\Gamma(\alpha+\beta)} x^{\alpha-1}(1-x)^{\beta-1}$.

- **Passo 1)** Gere valores $u_1 \sim U(0,1)$ e $u_2 \sim U(0,1)$
- **Passo 2)** Calcule $v_1 = u_1^{\frac{1}{\alpha}}$, $v_2 = u_2^{\frac{1}{\beta}}$ e $w = v_1 + v_2$
- **Passo 3)** Se $w \leq 1$, então aceite que $x$ é um valor gerado de $B(\alpha, \beta)$. Caso contrário, repita os passos 1) e 2).

# Distribuição beta (Atkinson e Pearce 1976)

## Exemplo

```
r_beta <- function(n, shape1, shape2) {
  seq_len(n) |>
    map_dbl(\(i) {
      u1 <- runif(1); u2 <- runif(1); v1 <- u1^(1 / shape1)
      v2 <- u2^(1 / shape2); w <- v1 + v2

      while(w > 1) {
        u1 <- runif(1); u2 <- runif(1); v1 <- u1^(1 / shape1)
        v2 <- u2^(1 / shape2); w <- v1 + v2
      }

      return(v1 / w)
    })
}
```

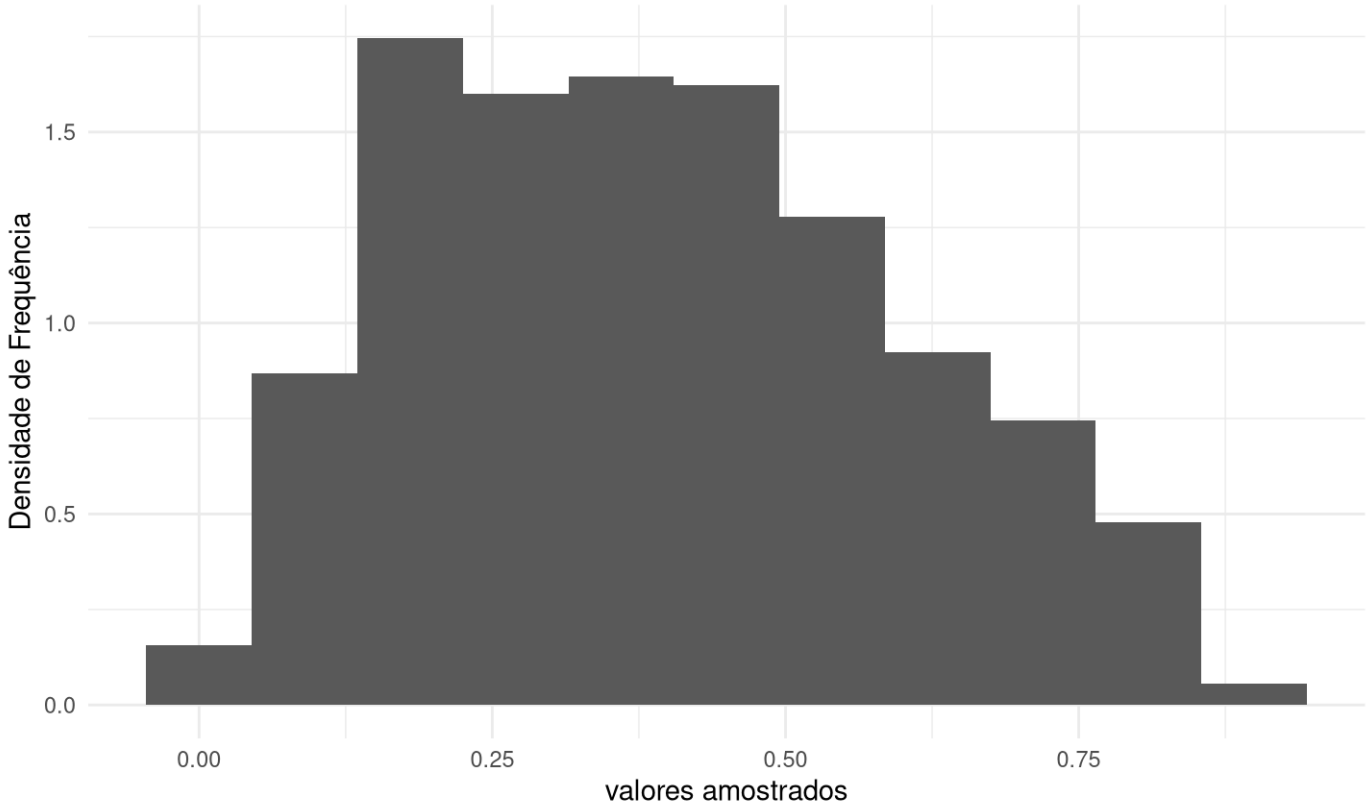
# Distribuição beta (Atkinson e Pearce 1976)

## Exemplo

```
n <- 1000
k <- (1 + log2(n)) |> round()
dados <- tibble(amostra = r_beta(n, shape1 = 2, shape2 = 3))
ggplot(data = dados) +
  geom_histogram(mapping = aes(x = amostra, y = ..density..), bins = k) +
  theme_minimal() +
  labs(x = "valores amostrados", y = "Densidade de Frequência")
```

# Distribuição beta (Atkinson e Pearce 1976)

## Exemplo



```r
ks.test(dados$amostra, pbeta, shape1 = 2, shape2 = 3)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  dados$amostra
## D = 0.035974, p-value = 0.1502
## alternative hypothesis: two-sided
```

# Distribuição binomial

Seja $X \sim b(n, p)$.

- **Passo 0)** Defina inicialmente $x = 0$
- **Passo 1)** Gere um valor $u \sim U(0, 1)$
- **Passo 2)** Se $u \leq p$, então acrescente $1$ a x
- **Passo 3)** Repita os passos 1) e 2) $n$ vezes

# Distribuição binomial

## Exemplo

```
r_binom <- function(n, size, p) {
  seq_len(n) |>
    map_dbl(\(i) {
      seq_len(size) |> map_dbl(~ runif(1) <= p) |> sum()
    })
}
```

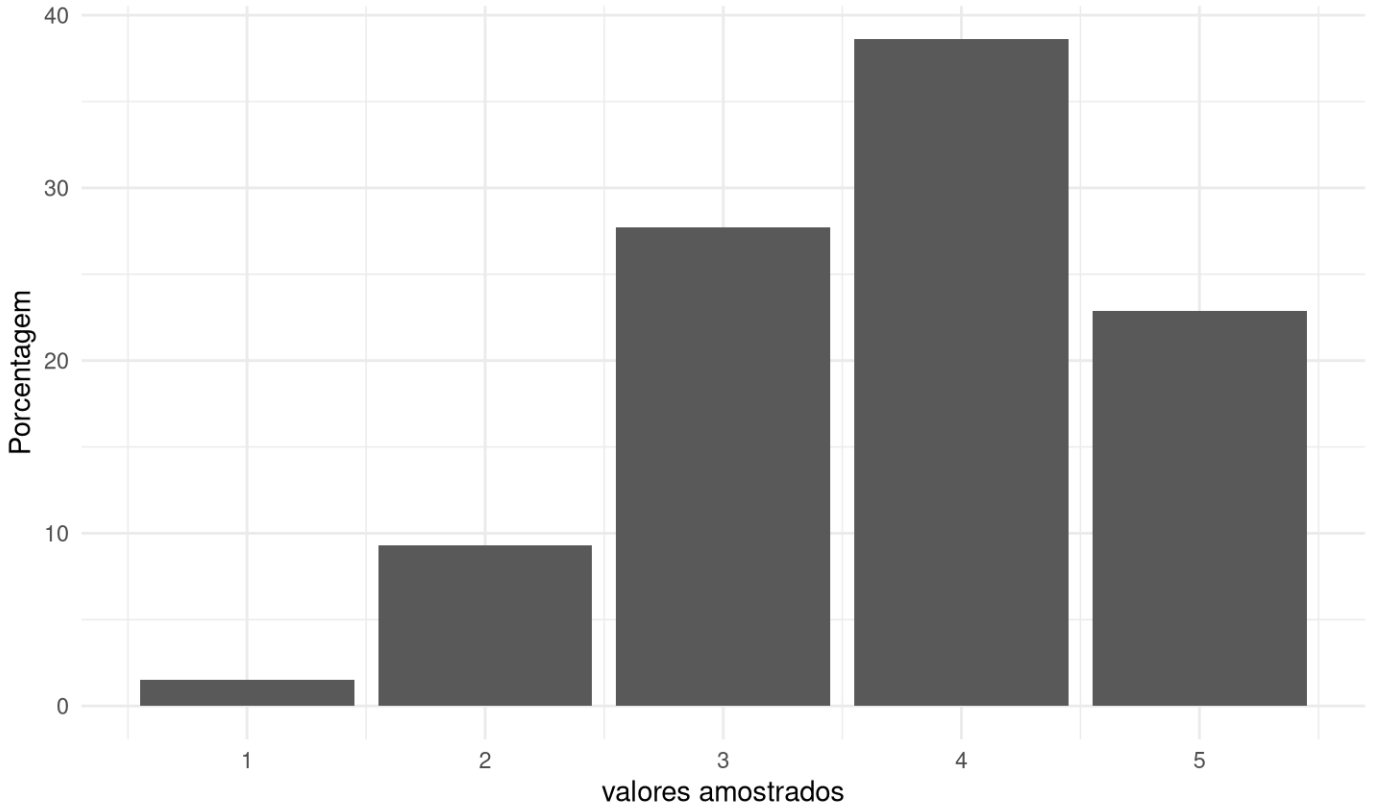
# Distribuição binomial

## Exemplo

```
n <- 1000
k <- (1 + log2(n)) |> round()
dados <- tibble(amostra = r_binom(1000, 5, 0.75))
ggplot(data = dados) +
  geom_bar(aes(x = amostra, y = 100 * ..prop..)) +
  theme_minimal() +
  labs(x = "valores amostrados", y = "Porcentagem")
```

# Distribuição binomial

## Exemplo



```r
n <- 1000
size <- 5; p <- 0.55
suporte <- 0:size
fda <- stepfun(x = suporte, y = c(0, pbinom(suporte, size = size, prob = p)))
dgof::ks.test(rbinom(1000, size, p), fda)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rbinom(1000, size, p)
## D = 0.024218, p-value = 0.6006
## alternative hypothesis: two-sided
```

# Distribuição poison

Seja $X \sim Poison(\lambda)$.

- **Passo 0)** Defina inicialmente $k = 0$ e $p = 1$
- **Passo 1)** Acrescente $1$ a $k$, gere um valor de $u \sim U(0, 1)$ e atualize $p$ para $p \cdot u$
- **Passo 2)** Se $\exp(-\lambda) >= p$, aceite $k - 1$ como um valor gerado da distribuição $Poison(\lambda)$. Caso contrário, repita 1).

# Distribuição poison

## Exemplo

```
r_pois <- function(n, lambda) {
   seq_len(n) |>
    map_dbl(\(i) {
      k <- 0; p <- 1

      while(exp(-lambda) < p) {
        k <- k + 1; p <- p * runif(1)
      }

      return(k - 1)
    })
}
```

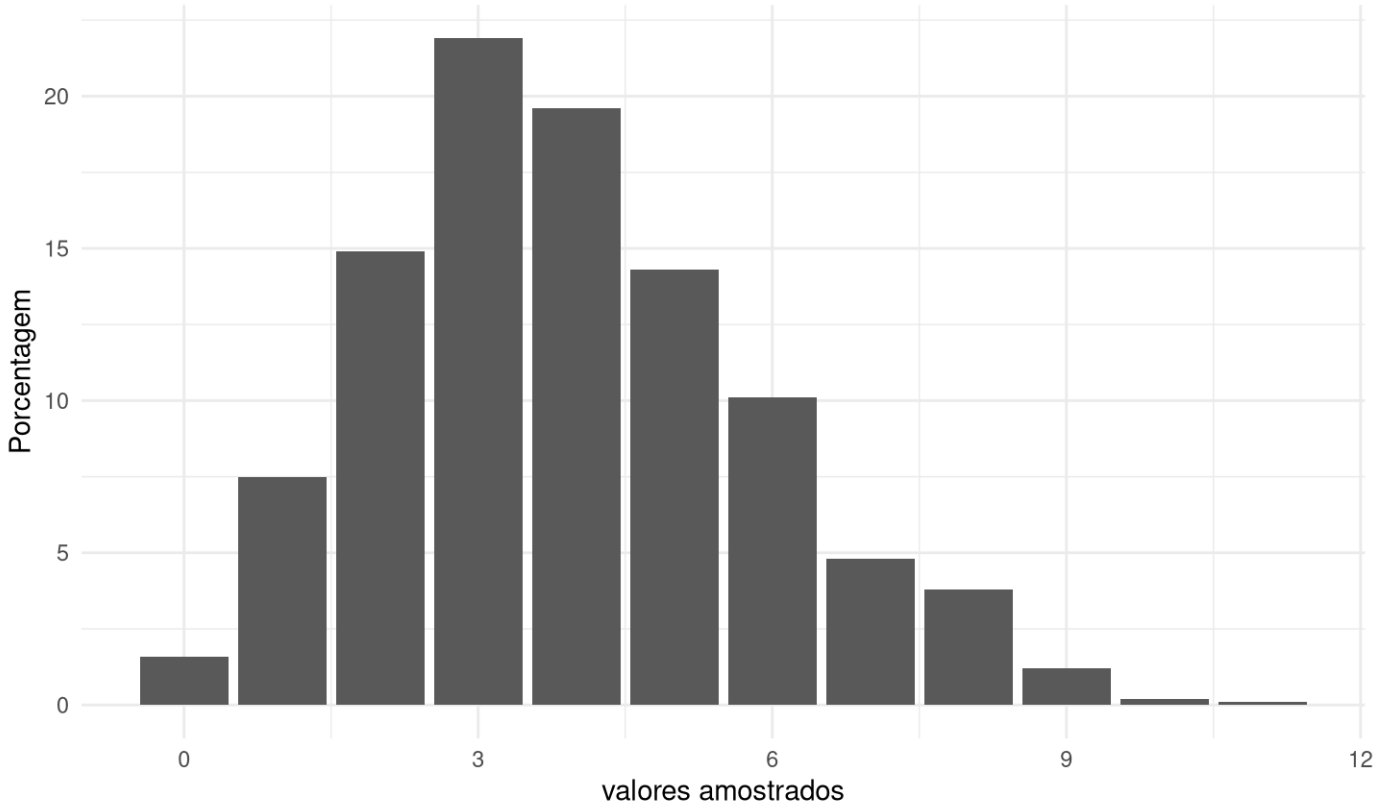
# Distribuição poison

## Exemplo

```
n <- 1000
k <- (1 + log2(n)) |> round()
dados <- tibble(amostra = r_pois(1000, lambda = 4))
ggplot(data = dados) +
  geom_bar(aes(x = amostra, y = 100 * ..prop..)) +
  theme_minimal() +
  labs(x = "valores amostrados", y = "Porcentagem")
```

# Distribuição poison

## Exemplo



```r
suporte <- 0:10000
fda <- stepfun(suporte, c(0, ppois(suporte, lambda = 4)))
dgof::ks.test(dados$amostra, fda)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  dados$amostra
## D = 0.026163, p-value = 0.5004
## alternative hypothesis: two-sided
```

# Distribuição multivariada

# Distribuição normal multivariada

- Seja $\mathbf{X}$ um vetor aleatório. Lembre que $E(A \cdot \mathbf{X} + \mathbf{b}) = A \cdot E(\mathbf{X}) + b$ , e $Var(A \cdot \mathbf{X} + \mathbf{b}) = A \cdot Var(\mathbf{X}) \cdot A^\top$
- $Se \mathbf{X} \sim N(\mu, \Sigma)$, então $R^{-1}(\mathbf{X} - \mu) \sim N(\mathbf{0}, I)$ em que $R$ é decomposição de Cholesky ( $R \cdot R^\top = \Sigma$ )
- $Se \mathbf{X} \sim N(\mathbf{0}, \Sigma)$, então $R \cdot \mathbf{X} + \mu \sim N(\mu, \Sigma)$

## Exemplo

```
r_mnorm <- function(n, sigma, media) {
  media <- matrix(media, ncol = 1)
  R <- chol(m_var)
  seq_len(n) |>
    sapply(\(i){
      return(R %*% matrix(nrow(sigma) |> rnorm(), ncol = 1) + media)
    }) |>
    t()
}
```

# Distribuição normal multivariada

## Exemplo

```
p <- 0.90
dim <- 10
m_var <- seq_len(dim) |>
  sapply(\(i) {
    seq_len(dim) |> map_dbl(\(j) p^abs(i - j))
  })
v_media <- rep(2, dim)
dados <- r_mnorm(1000, sigma = m_var, media = v_media)

goft::mvshapiro_test(dados)
```

```
##
##  Generalized Shapiro-Wilk test for Multivariate Normality
##
## data:  dados
## MVW = 0.99832, p-value = 0.5471
```

# Referências


Ahrens, Joachim H, e Ulrich Dieter. 1974. «Computer methods for sampling from gamma, beta, poisson and bionomial distributions». *Computing* 12 (3): 223–46.

Atkinson, AC, e MC Pearce. 1976. «The computer generation of beta, gamma and normal random variables». *Journal of the Royal Statistical Society: Series A (General)* 139 (4): 431–48.

Cheng, RCH. 1977. «The generation of gamma variables with non-integral shape parameter». *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 26 (1): 71–75.

Härdle, Wolfgang Karl, Ostap Okhrin, e Yarema Okhrin. 2017. *Basic elements of computational statistics*. Springer.

Marsaglia, George. 1968. «Random numbers fall mainly in the planes». *Proceedings of the National Academy of Sciences of the United States of America* 61 (1): 25.

Neave, Henry R. 1973. «On using the Box-M üller transformation with multiplicative congruential pseudo-random number generators». *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 22 (1): 92–97.

Wald, Abraham, e Jacob Wolfowitz. 1940. «On a test whether two samples are from the same population». *The Annals of Mathematical Statistics* 11 (2): 147–62.