Estatística Computacional

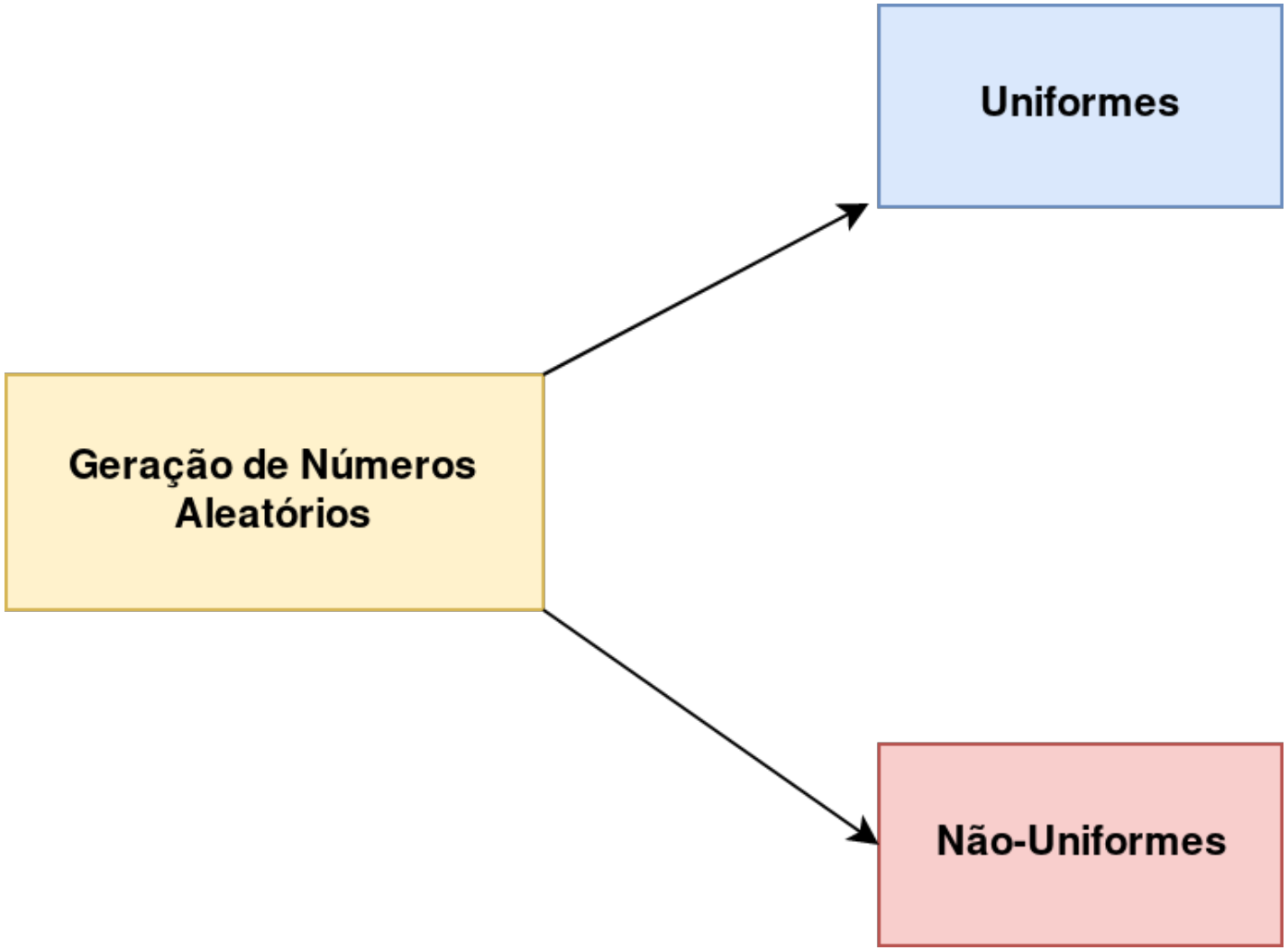
Pedro Rafael Diniz Marinho

Universidade Federal da Paraíba
Departamento de Estatística

2019.1

Geração de Números Pseudo-Aleatórios

Geração de Números Pseudo-Aleatórios



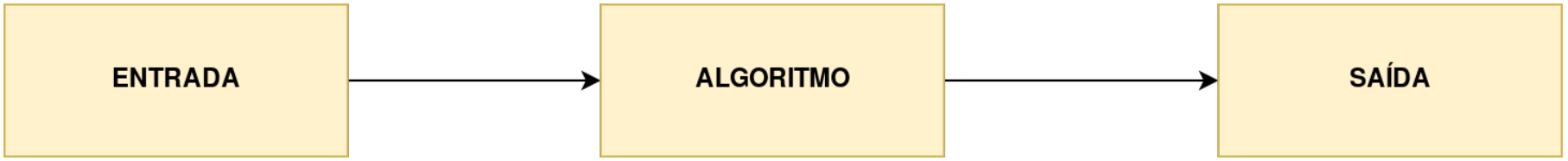
Geração de Números Pseudo-Aleatórios

Nota: Diversos materiais usam o título **Geração de Números Aleatórios**. Trata-se de um "péssimo" título uma vez que resultados puramente aleatórios **não são** reproduzíveis.

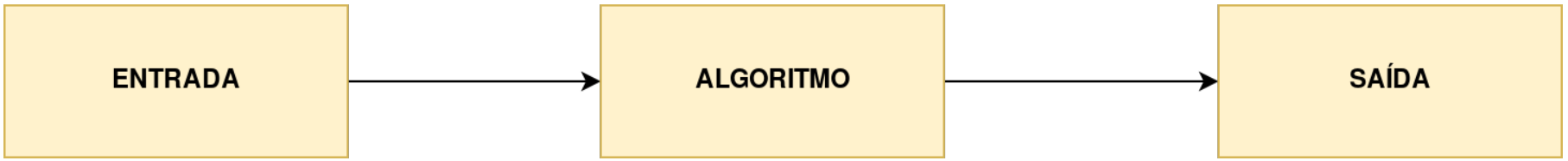
Um título mais adequado seria **Sequência Pseudo-Aleatória de Números**. De toda forma, sempre que for mencionado o título **Geração de Números Aleatórios** entenda que se estar a falar sobre **Sequência Pseudo-Aleatória de Números**.

Importante: A sequência de números pseudo-aleatórios são reproduzíveis uma vez que tal sequência é obtida por meio de um algoritmo.

Geração de Números Pseudo-Aleatórios

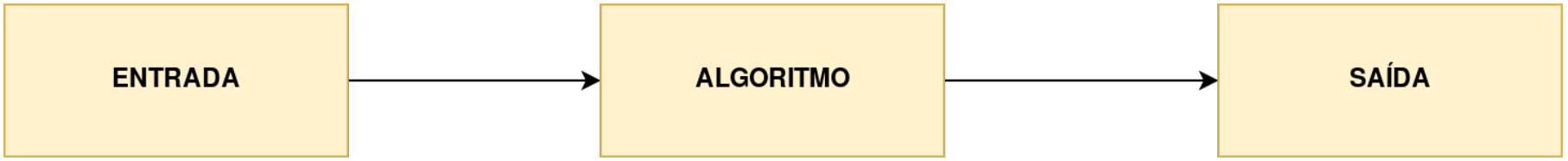


Geração de Números Pseudo-Aleatórios



Nota: A entrada normalmente é chamada de **semente** do gerador de números pseudo-aleatórios.

Geração de Números Pseudo-Aleatórios



Nota: A entrada normalmente é chamada de **semente** do gerador de números pseudo-aleatórios.

Fato: A partir de sequências uniformes, podemos gerar sequências não-uniformes.

Geração de Números Pseudo-Aleatórios

Definição (Transformação Integral de Probabilidade): Seja X uma variável aleatória com função de distribuição F_X. A transformação de X tal que $U = F_X(X)$ é denominada **transformação integral de probabilidade**.

Geração de Números Pseudo-Aleatórios

Definição (Transformação Integral de Probabilidade): Seja X uma variável aleatória com função de distribuição F_X. A transformação de X tal que $U = F_X(X)$ é denominada **transformação integral de probabilidade**.

Observe que o uso da transformação acima depende da possibilidade de invertermos a função F. A função inversa tem domínio em $[0, 1]$. Porém, se F tiver saltos ou for escada, F não admitirá inversa. Dessa forma, utilizaremos a **função inversa generalizada** e que por abuso de notação será representada por F^{-1}.

Geração de Números Pseudo-Aleatórios

Definição (**Inversa Generalizada de** F): Seja F uma função de distribuição qualquer. A inversa generalizada denotada por F^{-1} é definida como:

Geração de Números Pseudo-Aleatórios

Definição (**Inversa Generalizada de** F): Seja F uma função de distribuição qualquer. A inversa generalizada denotada por F^{-1} é definida como:

$$F^{-1}(u) = \inf\{x \in \mathbb{R} : F(x) \geq u\}.$$

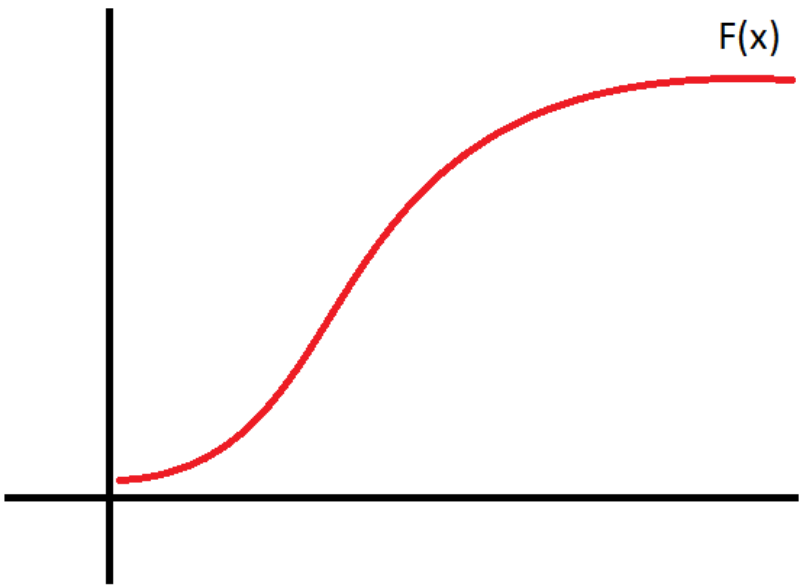
Geração de Números Pseudo-Aleatórios

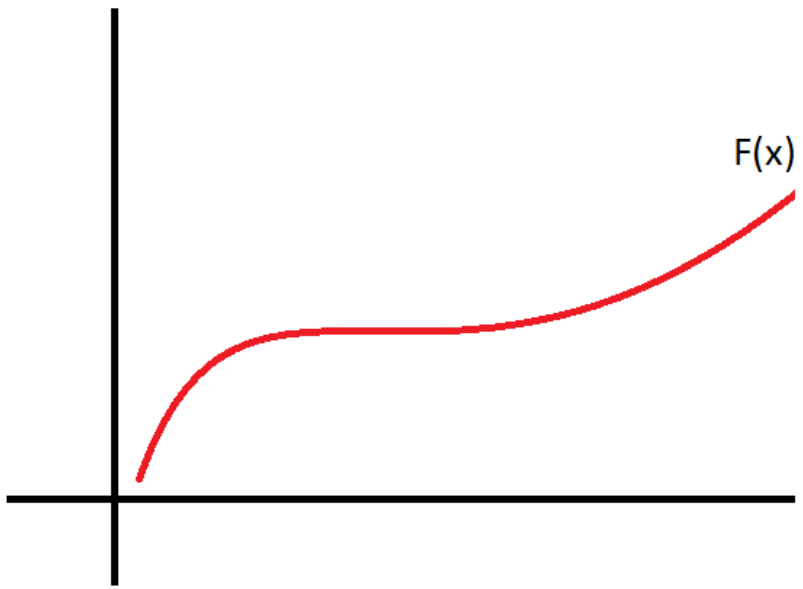
Definição (**Inversa Generalizada de** F): Seja F uma função de distribuição qualquer. A inversa generalizada denotada por F^{-1} é definida como:

$$F^{-1}(u) = \inf\{x \in \mathbb{R} : F(x) \geq u\}.$$

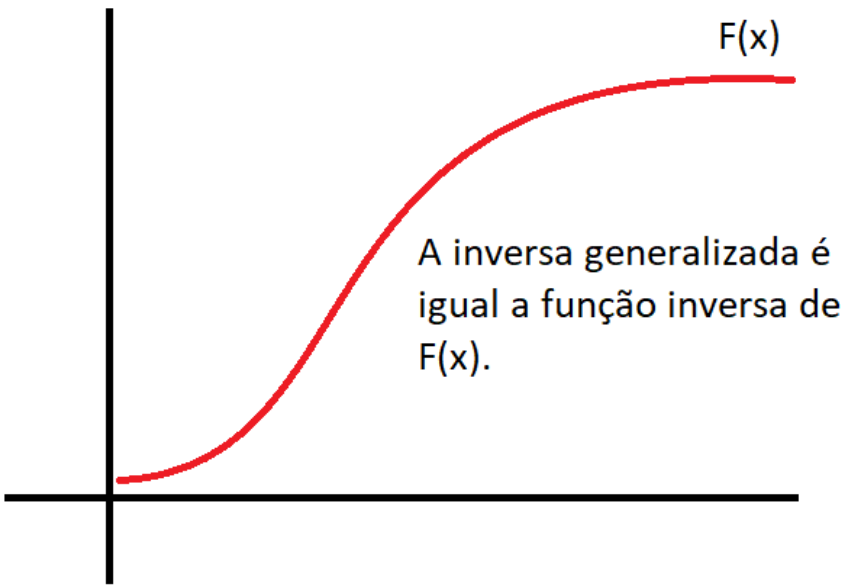
Nota: Caso a função inversa de F exista no sentido usual, esta coincidirá com a função inversa generalizada de F.

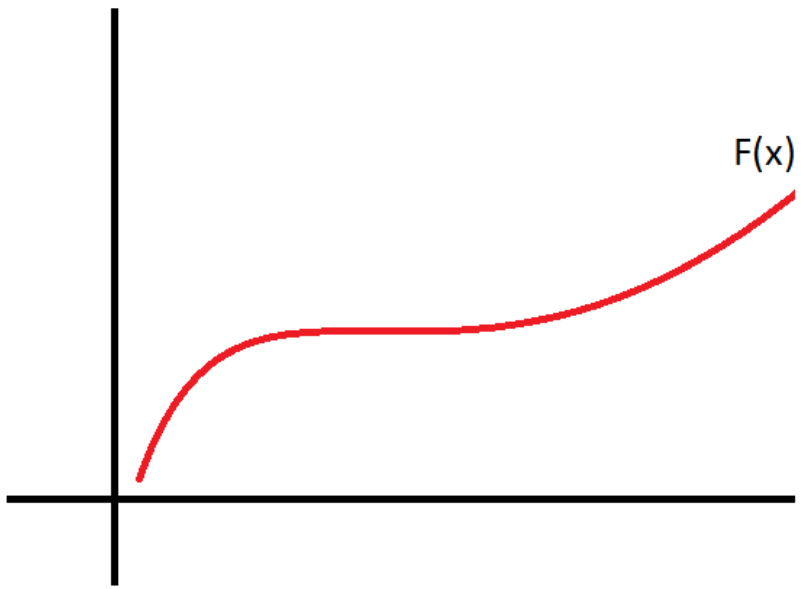
Geração de Números Pseudo-Aleatórios



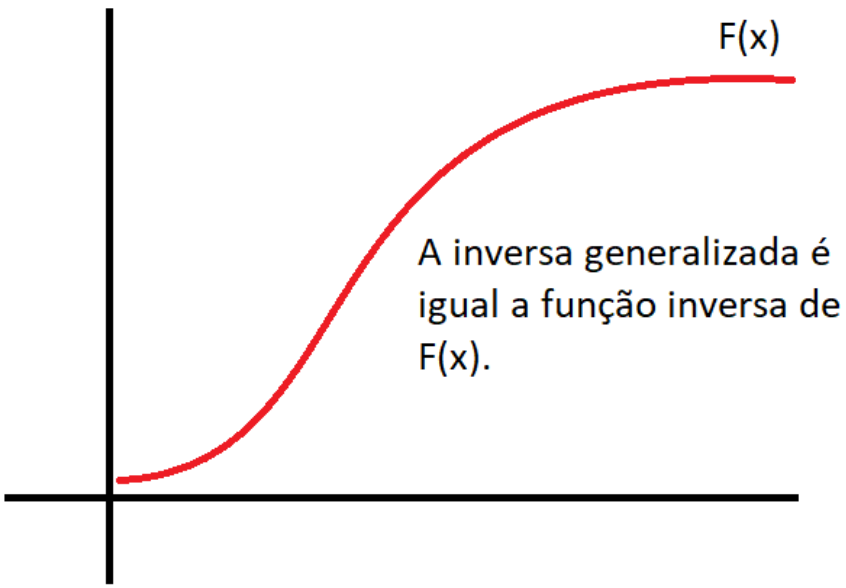


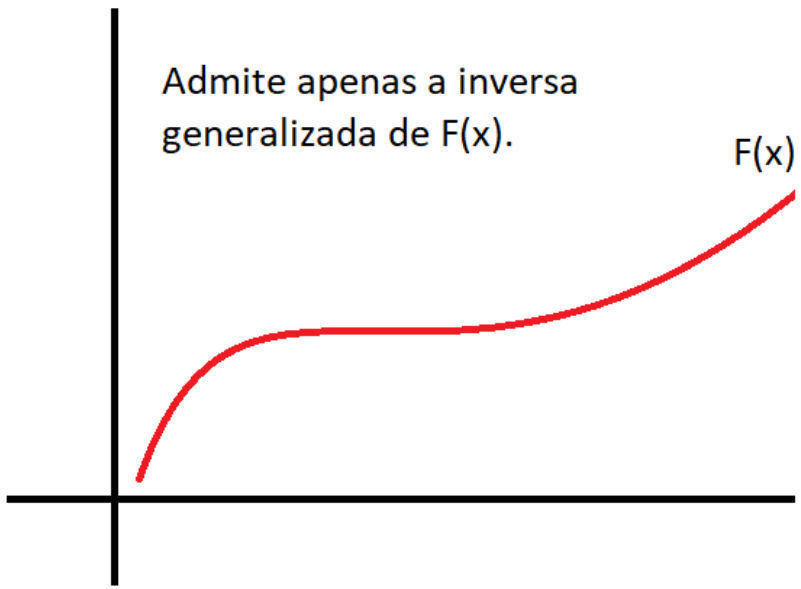
Geração de Números Pseudo-Aleatórios



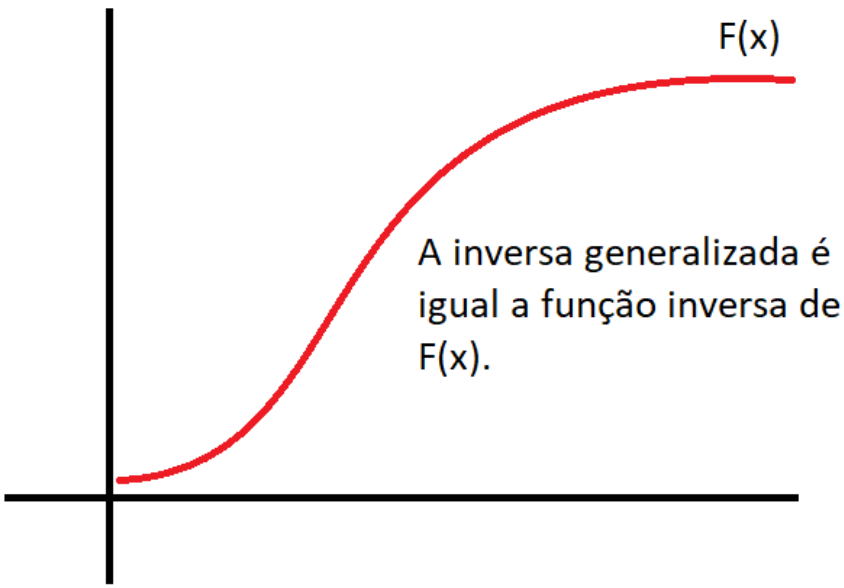


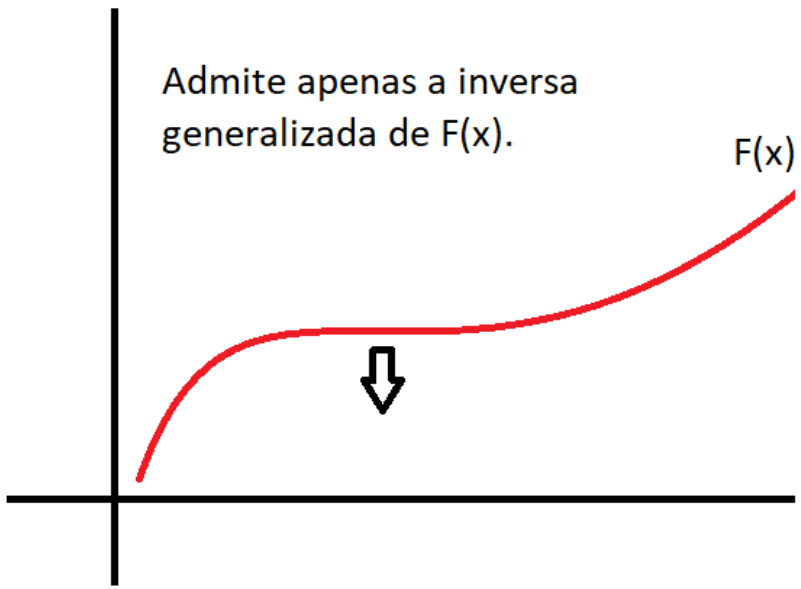
Geração de Números Pseudo-Aleatórios



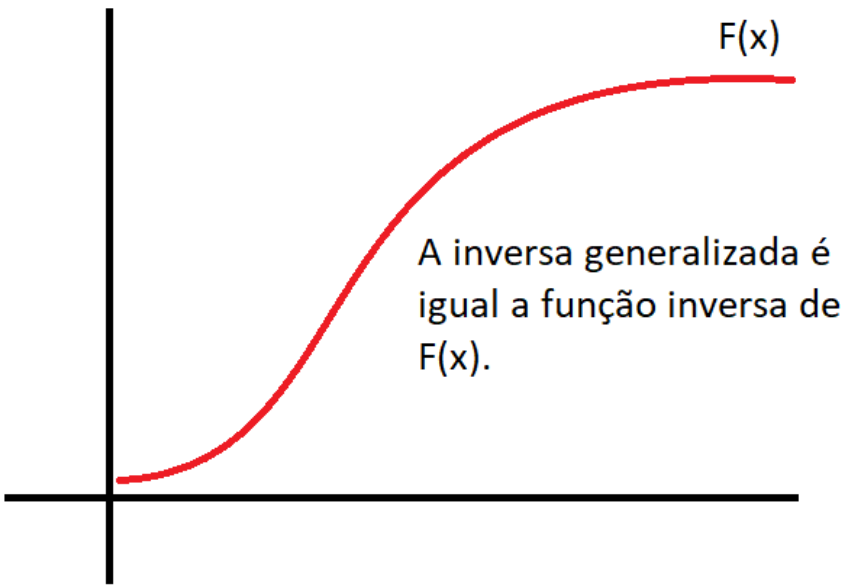


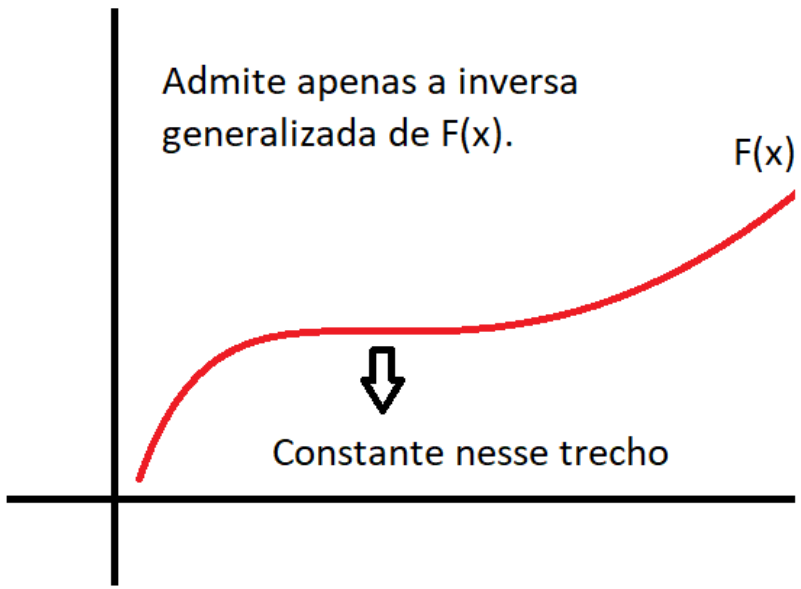
Geração de Números Pseudo-Aleatórios





Geração de Números Pseudo-Aleatórios





Geração de Números Pseudo-Aleatórios

Observação Importante: A inversa generalizada cuida de situações em que a função F não é invertível.

Proposição: Seja X uma variável aleatória contínua com função de distribuição F. Sendo $U \sim U_c[0,1]$, então $X = F_X^{-1}(U)$ terá função de distribuição F.

Geração de Números Pseudo-Aleatórios

Observação Importante: A inversa generalizada cuida de situações em que a função F não é invertível.

Proposição: Seja X uma variável aleatória contínua com função de distribuição F. Sendo $U \sim U_c[0,1]$, então $X = F_X^{-1}(U)$ terá função de distribuição F.

Prova:

$$P(X \leq x) = P(F_X^{-1}(U) \leq x) = P(U \leq F_X(x)) = F(x).$$

Geração de Números Pseudo-Aleatórios

O método poderá ser aplicado para geração de observações de variáveis aleatórias contínuas ou discreta de uma determinada distribuição de probabilidade.

Algoritmo (Método da Inversão)

Geração de Números Pseudo-Aleatórios

O método poderá ser aplicado para geração de observações de variáveis aleatórias contínuas ou discreta de uma determinada distribuição de probabilidade.

Algoritmo (Método da Inversão)

1. Obtenha a inversa $F_X^{-1}(u)$.

Geração de Números Pseudo-Aleatórios

O método poderá ser aplicado para geração de observações de variáveis aleatórias contínuas ou discreta de uma determinada distribuição de probabilidade.

Algoritmo (Método da Inversão)

1. Obtenha a inversa $F_X^{-1}(u)$.
2. Escreva um comando ou função que calcule $F_X^{-1}(u)$.

Geração de Números Pseudo-Aleatórios

O método poderá ser aplicado para geração de observações de variáveis aleatórias contínuas ou discreta de uma determinada distribuição de probabilidade.

Algoritmo (Método da Inversão)

1. Obtenha a inversa $F_X^{-1}(u)$.
2. Escreva um comando ou função que calcule $F_X^{-1}(u)$.
3. Para cada observação de uma variável aleatória, faça:

Geração de Números Pseudo-Aleatórios

O método poderá ser aplicado para geração de observações de variáveis aleatórias contínuas ou discreta de uma determinada distribuição de probabilidade.

Algoritmo (Método da Inversão)

1. Obtenha a inversa $F_X^{-1}(u)$.
2. Escreva um comando ou função que calcule $F_X^{-1}(u)$.
3. Para cada observação de uma variável aleatória, faça:
   a) Gere uma observação u de uma variável aleatória $U \sim U_c[0, 1]$.

Geração de Números Pseudo-Aleatórios

O método poderá ser aplicado para geração de observações de variáveis aleatórias contínuas ou discreta de uma determinada distribuição de probabilidade.

Algoritmo (Método da Inversão)

1. Obtenha a inversa $F_X^{-1}(u)$.
2. Escreva um comando ou função que calcule $F_X^{-1}(u)$.
3. Para cada observação de uma variável aleatória, faça:
   a) Gere uma observação u de uma variável aleatória $U \sim U_c[0, 1]$.
   b) Calcule $x = F_X^{-1}(u)$.

Geração de Números Pseudo-Aleatórios

Exercício: Considere a função densidade de probabilidade $f_X(x) = 3x^2, 0 \leq x \leq 1$. Obtenha utilizando o algoritmo acima uma sequência de 1000 números pseudo-aleatórios com distribuição $F_X(x)$.

Geração de Números Pseudo-Aleatórios

Solução:

Temos que $F_X(x) = \int_0^x f_X(x)dx = x^3$, com $0 \leq x \leq 1$. Fazendo $u = x^3$, temos que

$$F_X^{-1}(u) = u^{1/3},\ 0 \leq u \leq 1.$$

Utilizaremos o algoritmo apresentado e $F_X^{-1}(u)$ obtido logo acima para obter uma sequência de observações $x_1, \ldots, x_n$.

Geração de Números Pseudo-Aleatórios

Código R:

```
1:  # Função densidade de X.
2:  pdf_f <- function(x){
3:    3 * x^2
4:  }
5:
6:  # Gerando mil números pseudo-aleatórios
7:  # com distribuição U[0,1].
8:  set.seed(0); u <- runif(n = 1000, min = 0, max = 1)
9:
10: # Aplicando u à inversa de F_X(x).
11: x <- u^(1/3)
```

Geração de Números Pseudo-Aleatórios

```
12: # Histograma da densidade da amostra.
13: hist(x, prob = TRUE, xlab = "Dominio",
14:      ylab = "Densidade",
15:      main = expression(f(x)==3*x^2))
16: dominio <- seq(from=0, to = 1, by = .01)
17: lines(dominio, pdf_f(dominio), lwd = 2, col = "red")
```

Será que deu certo?

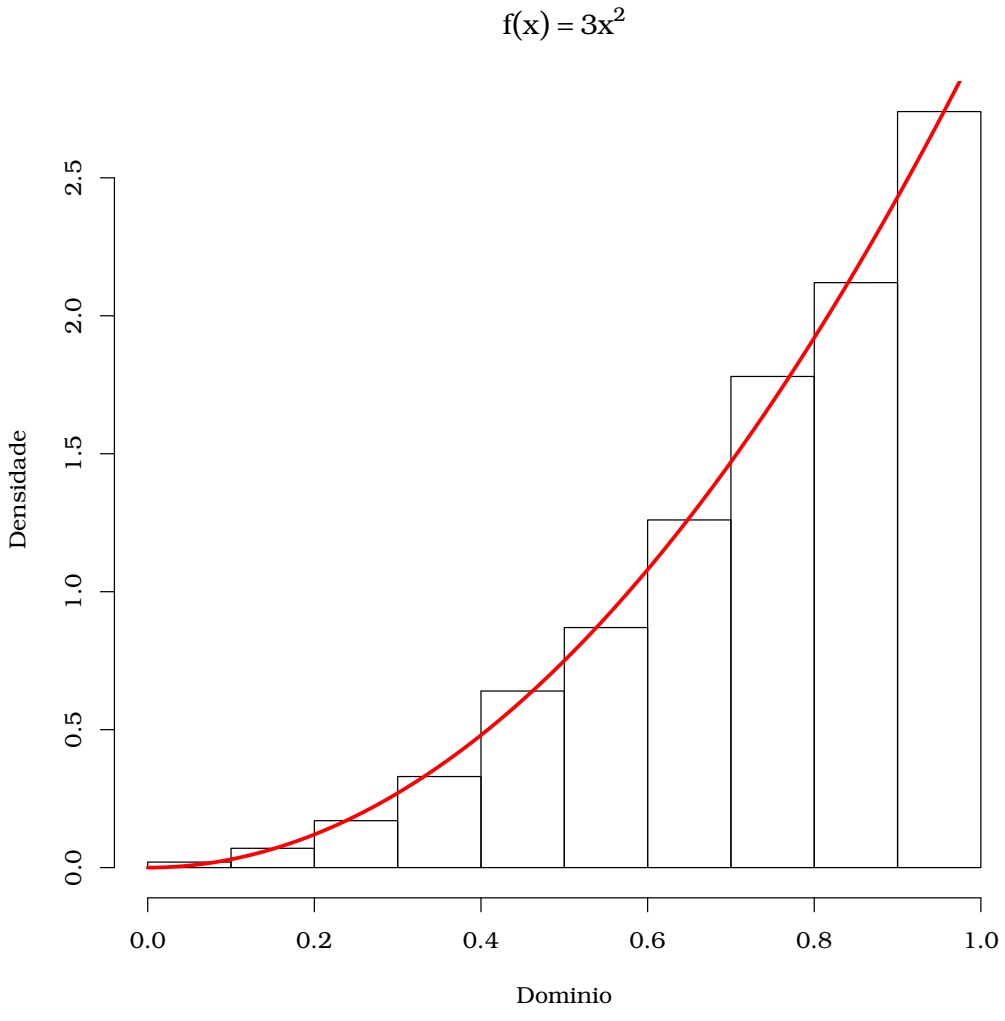
Geração de Números Pseudo-Aleatórios

```
12: # Histograma da densidade da amostra.
13: hist(x, prob = TRUE, xlab = "Dominio",
14:      ylab = "Densidade",
15:      main = expression(f(x)==3*x^2))
16: dominio <- seq(from=0, to = 1, by = .01)
17: lines(dominio, pdf_f(dominio), lwd = 2, col = "red")
```

Será que deu certo?

Observemos o ajustamento da distribuição $f_X(x) = 3x^2, 0 \leq x \leq 1$ **aos dados gerados**.

Geração de Números Pseudo-Aleatórios



Geração de Números Pseudo-Aleatórios

Nota: Na Figura acima, o título possui uma expressão matemática. Este título é obtido especificando no argumento `main` a função `expression()` com a expressão matemática desejada. Consulte a documentação da linguagem para maiores detalhes.

Observação: O método da inversão é o algoritmo mais geral e utilizado para geração de números pseudo-aleatórios de uma distribuição qualquer.

Exercício: Seja $X \sim \text{Exp}(\lambda)$, tal que $f_X(x) = \lambda e^{-\lambda x}$, com $x > 0$ e $\lambda > 0$. Sem utilizar a função `rexp()`, construa uma função para geração de números pseudo-aleatórios de uma distribuição $\text{Exp}(\lambda)$.

Geração de Números Pseudo-Aleatórios

Exercício: É possível fazer uso do método da inversão para implementarmos uma função que faz uso do método da inversão para geração de números pseudo-aleatórios de uma variável aleatória $X \sim \mathcal{N}(\mu, \sigma^2)$? Justifique sua resposta.

Geração de Números Pseudo-Aleatórios

Para gerarmos números pseudo-aleatórios de uma distribuição qualquer, foi preciso gerar números pseudo-aleatórios proveniente de uma distribuição uniforme.

Definição: Um gerador de números pseudo-aleatórios é um **algoritmo** que iniciando em um valor (**semente**) e usando uma transformação determinística D, produz uma sequência $u_i = D^i(u_0)$ de valores em $(0, 1)$.

Observação: O conhecimento de $u_1, \ldots, u_n$ não deverá conduzir ao conhecimento de $u_{n+1}, u_{n+2}, \ldots$. Dessa forma, desejamos que a sequência não poderá ser previsível.

Geração de Números Pseudo-Aleatórios

Por exemplo, se gerarmos uma sequência $\{0, 1, \ldots, n\}$ (com n muito grande) e ao final dividirmos cada número por n, teremos uma sequência em $[0, 1]$, mas nesse caso há previsibilidade.

Muito Importante

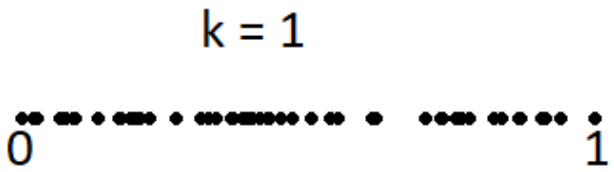
Um bom gerador de números pseudo-aleatórios com distribuição uniforme não permite a detecção de qualquer padrão em $[0, 1]^k$, com $k = 1, \ldots, 6$, isto é, não é possível detectar padrões até a sexta dimensão. Em outras palavras, é preciso que a uniformidade esteja presente até a sexta dimensão.

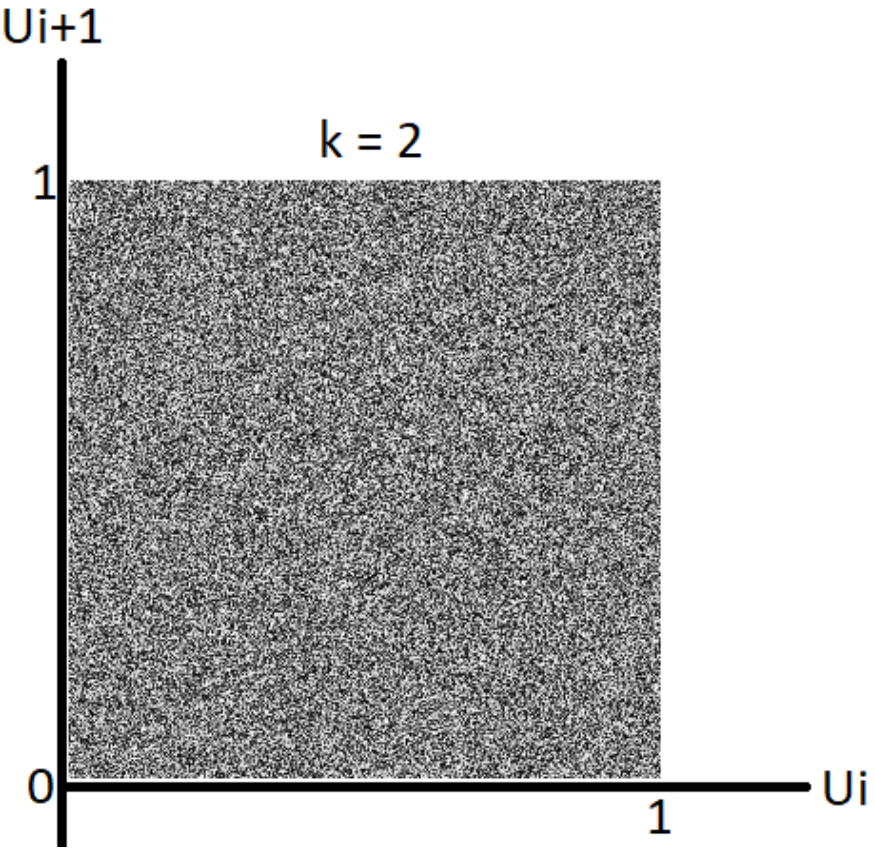
Geração de Números Pseudo-Aleatórios

Por exemplo, para $k = 1$ e $k = 2$, desejamos:

Geração de Números Pseudo-Aleatórios

Por exemplo, para $k = 1$ e $k = 2$, desejamos:





Geração de Números Pseudo-Aleatórios

Porém, lembre-se, precisamos de uniformidade até a sexta dimensão.

Geração de Números Pseudo-Aleatórios

Figura: John von Neumann.

Um dos primeiros algoritmos de número pseudo-aleatório (**Midsquare**) foi idealizado pelo matemático húngaro (naturalizado americano) **John von Neumann** (1903 - 1957).

von Neumann deixou contribuições em diversas áreas como ciência da computação, economia, teoria dos jogos, análise funcional, teoria dos conjuntos, entre outras. von Neumann foi um dos construtores do primeiro computador digital eletrônico (ENIAC).

Geração de Números Pseudo-Aleatórios

Algoritmo Midsquare de von Neumann ($\sim$ 1949):

Geração de Números Pseudo-Aleatórios

Algoritmo Midsquare de von Neumann ($\sim$ 1949):

1. Tome uma semente: um número inteiro de 4 dígitos.

Geração de Números Pseudo-Aleatórios

Algoritmo Midsquare de von Neumann ($\sim$ 1949):

1. Tome uma semente: um número inteiro de 4 dígitos.
2. Calcule o quadrado desse número e preencha, se necessário, com zero à esquerda, para que o número gerado tenha 8 dígitos.

Geração de Números Pseudo-Aleatórios

Algoritmo Midsquare de von Neumann ($\sim$ 1949):

1. Tome uma semente: um número inteiro de 4 dígitos.
2. Calcule o quadrado desse número e preencha, se necessário, com zero à esquerda, para que o número gerado tenha 8 dígitos.
3. Repita os passos 1 e 2 a quantidade de vezes desejadas e ao final divida cada número obtido, incluindo a semente por 10 mil.

Geração de Números Pseudo-Aleatórios

Exercício: Construa, em `R`, uma função que implemente o gerador Midsquare. **Dica**: Talvez seja necessário manipular strings para efetuar o passo 2 do algoritmo. As funções `strsplit()` e `paste()` podem ser úteis. Procurem maiores detalhes na documentação da linguagem.

Geração de Números Pseudo-Aleatórios

Solução:

```
1: midsquare <- function(n = 1, semente = 1987){
2:   valores <- NULL; i = 1
3:   repeat{
4:      semente <- semente^2
5:      quadrado <- unlist(strsplit(as.character(semente),
6:                                  split=""))
7:      quadrado <- c(rep(0,8 - length(quadrado)),
8:                    quadrado)
9:      valores[i] <- as.numeric(paste(quadrado[3:6],
10:                                    collapse = ""))
```

Geração de Números Pseudo-Aleatórios

```
12:       semente <- valores[i]
13:       i <- i + 1
14:       if (i > n) break
15:   }
16:   valores
17:}
18: midsquare(n=4, semente = 1348)
#> [1] 8171 7652 5531 5919
```

Geração de Números Pseudo-Aleatórios

Exercício: O que acontece se um dado valor gerado $x_i = 0$? Discutam.

Geração de Números Pseudo-Aleatórios

Exercício: O que acontece se um dado valor gerado $x_i = 0$? Discutam.

Resposta: A sequência fica "presa" no zero, isto é:

$$x_{i+1} = x_{i+2} = \ldots = 0.$$

Geração de Números Pseudo-Aleatórios

Exercício: O que acontece se um dado valor gerado $x_i = 0$? Discutam.

Resposta: A sequência fica "presa" no zero, isto é:

$$x_{i+1} = x_{i+2} = \ldots = 0.$$

Exercício: Qual o problema do algoritmo ao considerar a semente ($x_0 = 3792$)? Discutam.

Geração de Números Pseudo-Aleatórios

Exercício: O que acontece se um dado valor gerado $x_i = 0$? Discutam.

Resposta: A sequência fica "presa" no zero, isto é:

$$x_{i+1} = x_{i+2} = \ldots = 0.$$

Exercício: Qual o problema do algoritmo ao considerar a semente ($x_0 = 3792$)? Discutam.

Resposta: A sequência não é aparentemente aleatória.

Geração de Números Pseudo-Aleatórios

Exercício: Qual o problema do algoritmo ao considerar $x_0 = 2100$?

Geração de Números Pseudo-Aleatórios

Exercício: Qual o problema do algoritmo ao considerar $x_0 = 2100$?
Resposta: O ciclo do gerador é muito curto.

Geração de Números Pseudo-Aleatórios

Exercício: Qual o problema do algoritmo ao considerar $x_0 = 2100$?
Resposta: O ciclo do gerador é muito curto.

Queremos geradores que possuam ciclos longos.

Geração de Números Pseudo-Aleatórios

Exercício: Qual o problema do algoritmo ao considerar $x_0 = 2100$?
Resposta: O ciclo do gerador é muito curto.

Queremos geradores que possuam ciclos longos.

Sendo assim, precisaremos de um algoritmo melhor...

Geração de Números Pseudo-Aleatórios

Gerador Congruencial:

$$x_i = (a * x_{i-1} + c) \bmod M, i = 1, 2, \ldots,$$

em que x_0 é a semente do gerador, a é o multiplicador, c é o deslocamento e M é o módulo.

Observação: Usualmente, tem-se que:

$$a, c, x_i \in \{0, 1, 2, \ldots, M-1\}.$$

Geração de Números Pseudo-Aleatórios

Gerador Congruencial:

$$x_i = (a * x_{i-1} + c) \bmod M, i = 1, 2, \ldots,$$

em que x_0 é a semente do gerador, a é o multiplicador, c é o deslocamento e M é o módulo.

Observação: Usualmente, tem-se que:

$$a, c, x_i \in \{0, 1, 2, \ldots, M-1\}.$$

Definição: Chamaremos de **período** de um gerador ao número de termos gerador a partir de uma semente x_0 sem a sequência até então gerada se repetir.

Geração de Números Pseudo-Aleatórios

Nota: $a \bmod b$ representa o **resto da divisão** entre a e b.

Terminologias:

Geração de Números Pseudo-Aleatórios

Nota: $a \bmod b$ representa o **resto da divisão** entre a e b.

Terminologias:

- Se $c \neq 0$, diremos que o gerador é **misto** (período máximo igual à M).

Geração de Números Pseudo-Aleatórios

Nota: $a \bmod b$ representa o **resto da divisão** entre a e b.

Terminologias:

- Se $c \neq 0$, diremos que o gerador é **misto** (período máximo igual à M).
- Se $c = 0$, diremos que o gerador é **multiplicativo** (período máximo igual à $M - 1$). Nesse caso, excluímos o valor 0 da sequência.

Geração de Números Pseudo-Aleatórios

Exercício: Utilizando $M = 64$, $x_0 = 1$, $a = 19$ e $c = 15$, gere uma sequência de 10 números pseudo-aleatórios obtido pelo gerador congruencial.

Geração de Números Pseudo-Aleatórios

Exercício: Utilizando $M = 64$, $x_0 = 1$, $a = 19$ e $c = 15$, gere uma sequência de 10 números pseudo-aleatórios obtido pelo gerador congruencial.

Exercício: Utilizando a linguagem `R`, implemente uma função para o algoritmo do gerador congruencial.

Geração de Números Pseudo-Aleatórios

```
1: rcongruencial <- function(n = 10, semente = 1987,
2:                           parametros, unif = TRUE){
3:
4:    a <- parametros[1]; c <- parametros[2]
5:    M <- parametros[3]
6:
7:    i <- 2; vetor <- NULL;
8:    vetor[1] <- semente
9:
10:   repeat{
11:      vetor[i] <- (a * vetor[i-1] + c) %% M
12:      i <- i + 1
13:
14:      if(i > n + 1) break
15:   }
```

Geração de Números Pseudo-Aleatórios

```
16:   ifelse(unif == TRUE, vetor <- vetor[-1]/M,
17:   vetor <- vetor[-1])
18:   vetor
19:}
20:
21:rcongruencial(n = 10, semente = 1,
22:                  parametros = c(19,15,64), unif = F)
#> [1] 34 21 30 9 58 29 54 17 18 37
```

Geração de Números Pseudo-Aleatórios

Na linguagem `C`, a função `rand()` da biblioteca padrão **stdlib.h** utiliza-se do gerador. Mais especificamente, alguns compiladores de `C`, consideram:

Geração de Números Pseudo-Aleatórios

Na linguagem `C`, a função `rand()` da biblioteca padrão **stdlib.h** utiliza-se do gerador. Mais especificamente, alguns compiladores de `C`, consideram:

1. **gcc**: $M^{32}, a = 69069, c = 5$ (misto);

Geração de Números Pseudo-Aleatórios

Na linguagem `C`, a função `rand()` da biblioteca padrão **stdlib.h** utiliza-se do gerador. Mais especificamente, alguns compiladores de `C`, consideram:

1. **gcc**: $M^{32}, a = 69069, c = 5$ (misto);
2. **Microsoft**: $M^{32}, a = 214013, c = 2531011$ (misto);

Geração de Números Pseudo-Aleatórios

Na linguagem `C`, a função `rand()` da biblioteca padrão **stdlib.h** utiliza-se do gerador. Mais especificamente, alguns compiladores de `C`, consideram:

1. **gcc**: $M^{32}, a = 69069, c = 5$ (misto);
2. **Microsoft**: $M^{32}, a = 214013, c = 2531011$ (misto);
3. **Borland**: $M^{32}, a = 22695477, c = 1$ (misto).

Observação: Em `C`, a função `rand()` gera números entre 0 e `RAND_MAX` (constante simbólica que representa o maior inteiro representável pelo computador).

Geração de Números Pseudo-Aleatórios

Na linguagem de programação `Ox`, desenvolvida por Jurgen Doornik há o gerador de **Park-Miller** que é um gerador de números pseudo-aleatório congruencial multiplicativo. Nesse gerador, considera-se:

$$M = 2^{32} - 1, a = 16807, c = 0.$$

Geração de Números Pseudo-Aleatórios

Na linguagem de programação `Ox`, desenvolvida por Jurgen Doornik há o gerador de **Park-Miller** que é um gerador de números pseudo-aleatório congruencial multiplicativo. Nesse gerador, considera-se:

$$M = 2^{32} - 1, a = 16807, c = 0.$$

Comentário particular do professor: Não aconselharia o uso da linguagem `Ox` por se tratar de uma linguagem fechada mantida especialmente por uma única pessoa. Maiores detalhes em `http://www.doornik.com/`. Muito embora `Ox` é vendida como uma linguagem eficiente, temos a disposição diversas outras linguagem ainda mais eficientes e livres como é o caso de `C` e `C++`. Além disso, linguagem como, por exemplo, `C` e `C++` conversam facilmente com outras linguagem de programação.

Geração de Números Pseudo-Aleatórios

Gerador Randu:

$$x_{i+1} = 65539 * x_i \mod 31.$$

Observação: Há diversos materiais falando sobre este gerador em que podemos citar Donald E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd edition (Addison-Wesley, Boston, 1998).

Nota: O gerador randu é um gerador congruencial que foi utilizado por muito tempo em manframes entre as décadas de 60 e 70. Porém, este gerador possui erros consideráveis e com o tempo ele foi deixado de lado. **Ele é considerado um dos piores algoritmos geradores de números pseudo-aleatórios já criado**.

Geração de Números Pseudo-Aleatórios

Exercício: Gere uma sequência de números pseudo-aleatórios utilizando o gerador randu.

Exercício: Implemente, utilizando a linguagem `R`, uma função para geração de números pseudo-aleatórios utilizando o gerador randu.

Geração de Números Pseudo-Aleatórios

Solução:

```
1: rrandu <- function(n, semente){
2:     vetor <- NULL
3:    vetor[1] <- semente
4:    i <- 2
5:    repeat{
6:        vetor[i] <- (65539 * vetor[i-1]) %% 2^(31)
7:        i <- i + 1
8:        if (i > n+1) break
9:     }
10:    vetor[-1]
11:}
```

Geração de Números Pseudo-Aleatórios

Exemplo: Corra o código abaixo e descreva o motivo pelo qual o gerador não é razoável. Explique!

```
1: a <- NULL
2: for(i in 1:10)
3:    a[i] <- rrandu(n=1,semente=i)
4: plot(a, xlab = "x_i", ylab = "x_i+1",
5:      main = "Gerador Randu",
6:      pch = 16)
```

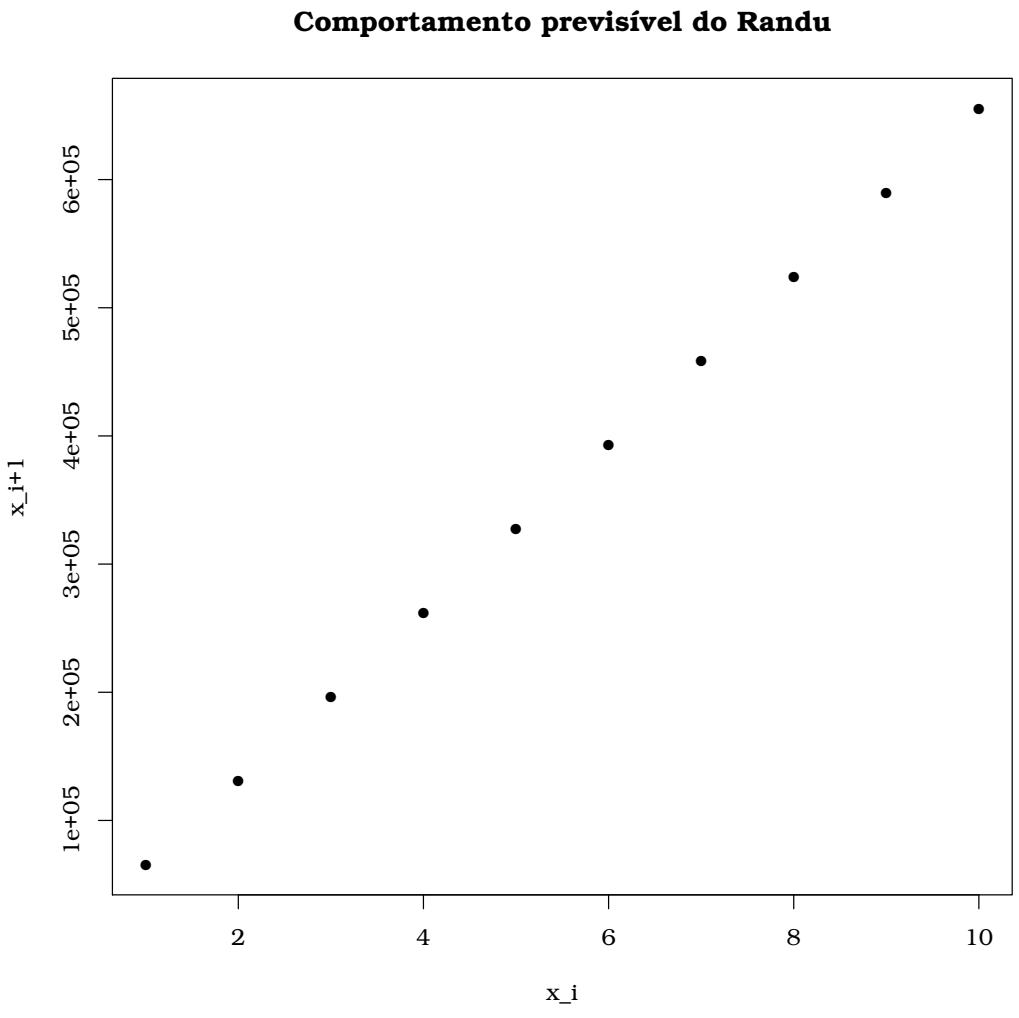
Geração de Números Pseudo-Aleatórios

Exemplo: Corra o código abaixo e descreva o motivo pelo qual o gerador não é razoável. Explique!

```
1: a <- NULL
2: for(i in 1:10)
3:     a[i] <- rrandu(n=1,semente=i)
4: plot(a, xlab = "x_i", ylab = "x_i+1",
5:       main = "Gerador Randu",
6:       pch = 16)
```

Observando o gráfico de x_i por x_{i+1}, é fácil perceber o comportamento previsível do gerador randu. Trata-se de uma característica muito indesejável em um gerador.

Geração de Números Pseudo-Aleatórios



Propriedades de Geradores Congruenciais

Notação: $a \equiv b \mod M$.

Lê-se: "a é congruente para b módulo M".

Tal notação apresentada acima significa: $a - b$ é divisível por M.

Exemplo:

Propriedades de Geradores Congruenciais

Notação: $a \equiv b \mod M$.

Lê-se: "a é congruente para b módulo M".

Tal notação apresentada acima significa: $a - b$ é divisível por M.

Exemplo:

a) $10 \equiv 4 \mod 2$;

Propriedades de Geradores Congruenciais

Notação: $a \equiv b \mod M$.

Lê-se: "a é congruente para b módulo M".

Tal notação apresentada acima significa: $a - b$ é divisível por M.

Exemplo:

a) $10 \equiv 4 \mod 2$;

b) $7 \equiv 1 \mod 3$;

Propriedades de Geradores Congruenciais

Notação: $a \equiv b \mod M$.

Lê-se: "a é congruente para b módulo M".

Tal notação apresentada acima significa: $a - b$ é divisível por M.

Exemplo:

a) $10 \equiv 4 \mod 2$;

b) $7 \equiv 1 \mod 3$;

c) $-10 \equiv 2 \mod 2$;

Propriedades de Geradores Congruenciais

Notação: $a \equiv b \mod M$.

Lê-se: "a é congruente para b módulo M".

Tal notação apresentada acima significa: $a - b$ é divisível por M.

Exemplo:

a) $10 \equiv 4 \mod 2$;

b) $7 \equiv 1 \mod 3$;

c) $-10 \equiv 2 \mod 2$;

d) $10 \equiv 0 \mod 5$.

Propriedades de Geradores Congruenciais

Definição: a é raiz primitiva de M, se e só se:

Propriedades de Geradores Congruenciais

Definição: a é raiz primitiva de M, se e só se:

c1) $a^{M-1} - 1 \equiv 0 \mod M$, isto é, $a^{M-1} - 1$ é divisível por M;

Propriedades de Geradores Congruenciais

Definição: a é raiz primitiva de M, se e só se:

c1) $a^{M-1} - 1 \equiv 0 \mod M$, isto é, $a^{M-1} - 1$ é divisível por M;

c2) Para todo inteiro positivo I tal que $I < M - 1$, temos que $(a^I - 1)/M$ **não** é inteiro.

Propriedades de Geradores Congruenciais

Teorema (Gerador linear congruencial multiplicativo): Seja M um número primo. O gerador congruencial multiplicativo tem **período completo**, isto é, $M - 1$, se e somente se, a (multiplicador) é raiz primitiva de M.

Propriedades de Geradores Congruenciais

Teorema (Gerador linear congruencial multiplicativo): Seja M um número primo. O gerador congruencial multiplicativo tem **período completo**, isto é, $M - 1$, se e somente se, a (multiplicador) é raiz primitiva de M.

Exercício: Postule um gerador congruencial linear considerando $M = 13$ e aplique o teorema acima para determinar se seu gerador possui período completo igual à 12.

Propriedades de Geradores Congruenciais

Exercício: Considere o gerador:

$$x_i = a * x_{i-1} \mod 11.$$

Considere também $a = 2, 3, \ldots, 10$. Para quais valores de a somos conduzidos a período completo igual à 10?

Propriedades de Geradores Congruenciais

Exercício: Considere o gerador:

$$x_i = a * x_{i-1} \mod 11.$$

Considere também $a = 2, 3, \ldots, 10$. Para quais valores de a somos conduzidos a período completo igual à 10?

Resposta: $a \in \{2, 6, 7, 8\}$ conduzem o gerador $x_i = a * x_{i-1} \mod 11$ a ter período completo.

Propriedades de Geradores Congruenciais

Conhecendo uma raiz primitiva de M, é possível facilmente obter outros valores de a que conduzem a período completo considerando a regra abaixo:

Regra: Se a é raiz primitiva de M, então $b = a^k \bmod M$ também é raiz primitiva de M se, e somente se, k e $M - 1$ forem primos relativos (**coprimos**), isto é, se o único divisor comum entre eles é o número 1.

Propriedades de Geradores Congruenciais

Conhecendo uma raiz primitiva de M, é possível facilmente obter outros valores de a que conduzem a período completo considerando a regra abaixo:

Regra: Se a é raiz primitiva de M, então $b = a^k \mod M$ também é raiz primitiva de M se, e somente se, k e $M - 1$ forem primos relativos (**coprimos**), isto é, se o único divisor comum entre eles é o número 1.

Exercício: Voltando ao exercício anterior, supomos que conhecemos que $a = 2$ é raiz primitiva de $M = 11$. Utilize a regra a cima para obter os valores de a que conduzem à período completo.

Propriedades de Geradores Congruenciais

Resposta:

Propriedades de Geradores Congruenciais

Resposta:

- $2^1 \bmod 11 = \mathbf{2}$;

Propriedades de Geradores Congruenciais

Resposta:

- $2^1 \mod 11 = \mathbf{2}$;
- $2^3 \mod 11 = \mathbf{8}$;

Propriedades de Geradores Congruenciais

Resposta:

- $2^1 \mod 11 = \mathbf{2}$;
- $2^3 \mod 11 = \mathbf{8}$;
- $2^7 \mod 11 = \mathbf{7}$;

Propriedades de Geradores Congruenciais

Resposta:

- $2^1 \bmod 11 = \mathbf{2}$;
- $2^3 \bmod 11 = \mathbf{8}$;
- $2^7 \bmod 11 = \mathbf{7}$;
- $2^9 \bmod 11 = \mathbf{6}$.

Propriedades de Geradores Congruenciais

Resposta:

- $2^1 \bmod 11 = \mathbf{2}$;
- $2^3 \bmod 11 = \mathbf{8}$;
- $2^7 \bmod 11 = \mathbf{7}$;
- $2^9 \bmod 11 = \mathbf{6}$.

Assim, $a \in \{2, 6, 7, 8\}$ conduz a período completo.

Propriedades de Geradores Congruenciais

Teorema (Gerador congruencial linear misto, $c \neq 0$): Considere o gerador congruencial linear misto $x_i = (a * x_{i-1} + c) \bmod M$, com $c > 0$ (inteiro). Para $M = 2^\beta$ (β inteiro positivo), o gerador possui período completo (isto é, 2^β), se e somente se:

Propriedades de Geradores Congruenciais

Teorema (Gerador congruencial linear misto, $c \neq 0$): Considere o gerador congruencial linear misto $x_i = (a * x_{i-1} + c) \bmod M$, com $c > 0$ (inteiro). Para $M = 2^\beta$ (β inteiro positivo), o gerador possui período completo (isto é, 2^β), se e somente se:

c1) $a \equiv \bmod 4$ ($a - 1$ é divisível por 4);

Propriedades de Geradores Congruenciais

Teorema (Gerador congruencial linear misto, $c \neq 0$): Considere o gerador congruencial linear misto $x_i = (a * x_{i-1} + c) \bmod M$, com $c > 0$ (inteiro). Para $M = 2^\beta$ (β inteiro positivo), o gerador possui período completo (isto é, 2^β), se e somente se:

c1) $a \equiv \bmod 4$ ($a - 1$ é divisível por 4);

c2) $\bmod(c, M) = 1$ (c e M são primos relativos, isto é, coprimos).

Propriedades de Geradores Congruenciais

Teorema (Gerador congruencial linear misto, $c \neq 0$): Considere o gerador congruencial linear misto $x_i = (a * x_{i-1} + c) \bmod M$, com $c > 0$ (inteiro). Para $M = 2^\beta$ (β inteiro positivo), o gerador possui período completo (isto é, 2^β), se e somente se:

c1) $a \equiv \bmod 4$ ($a - 1$ é divisível por 4);

c2) $\bmod(c, M) = 1$ (c e M são primos relativos, isto é, coprimos).

Nota: Qualquer valor ímpar de c satisfaz a propriedade c2. Além disso, não é necessário que M seja um número primo.

Propriedades de Geradores Congruenciais

Exemplo: O gerador utilizado pelo compilador **gcc** da linguagem C é um gerador congruencial linear, tal que:

- $a = 69069$;
- $c = 5$ (é um gerador misto e satisfaz a propriedade c2);
- $M = 2^{32}$ (M não é primo).

Gerador Mersenne Twister

O gerador **Mersenne Twister** foi proposto por Makoto Matsumoto e Takuji Nishimura, 1998. O gerador é de longe o mais utilizado na computação e em estudos de simulações. Sendo assim, o Mersenne Twister é, normalmente, o gerador de números pseudo-aleatórios padrão de diversas linguagens, softwares e bibliotecas como é o caso das linguagens **R**,**Julia**, **Python**, **Ruby**, das bibliotecas **GNU Scientific Library (GSL)**, **GNU Multiple Precision Arithmetic Library**, **Cuda** e de softwares como **Mathematica** e **Maple**.

Gerador Mersenne Twister

Algumas características do gerador Mersenne Twister são:

Gerador Mersenne Twister

Algumas características do gerador Mersenne Twister são:

- Não é um gerador congruencial linear;

Gerador Mersenne Twister

Algumas características do gerador Mersenne Twister são:

- Não é um gerador congruencial linear;
- Produz observações geradas de forma pseudo-aleatória com alta qualidade;

Gerador Mersenne Twister

Algumas características do gerador Mersenne Twister são:

- Não é um gerador congruencial linear;
- Produz observações geradas de forma pseudo-aleatória com alta qualidade;
- É um gerador rápido;

Gerador Mersenne Twister

Algumas características do gerador Mersenne Twister são:

- Não é um gerador congruencial linear;
- Produz observações geradas de forma pseudo-aleatória com alta qualidade;
- É um gerador rápido;
- Passou por diversos testes estatísticos para mensurar sua qualidade;

Gerador Mersenne Twister

Algumas características do gerador Mersenne Twister são:

- Não é um gerador congruencial linear;
- Produz observações geradas de forma pseudo-aleatória com alta qualidade;
- É um gerador rápido;
- Passou por diversos testes estatísticos para mensurar sua qualidade;
- Tem período de ocorrência $2^{19937} - 1$;

Gerador Mersenne Twister

Algumas características do gerador Mersenne Twister são:

- Não é um gerador congruencial linear;
- Produz observações geradas de forma pseudo-aleatória com alta qualidade;
- É um gerador rápido;
- Passou por diversos testes estatísticos para mensurar sua qualidade;
- Tem período de ocorrência $2^{19937} - 1$;
- É baseado em números primos de Mersenne, isto é, se a é um número primo e $M_a = 2^a - 1$ também é, diremos que M_a é número primo de Mersenne.

Gerando Normal

Pelo método da inversão, necessitamos conhecer F_X^{-1} para geramos observações de X. Computacionalmente não há problemas em utilizarmos o método da inversão para gerarmos observações gaussianas, uma vez que que há implementado em R a a inversa da acumulada da normal que é chamada de **função erro**, em que

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \text{e}^{-t^2} dt.$$

Gerando Normal

Algoritmo do método de Box-Muller

Gerando Normal

Algoritmo do método de Box-Muller

1. Gere $U_1, U_2 \sim \mathcal{U}(0,1)$;

Gerando Normal

Algoritmo do método de Box-Muller

1. Gere $U_1, U_2 \sim \mathcal{U}(0,1)$;
2. Faça $R^2 = -2 \log U_1$ (distribuição exponencial) e $S^2 = 2\pi U_2$ (distribuição uniforme);

Gerando Normal

Algoritmo do método de Box-Muller

1. Gere $U_1, U_2 \sim \mathcal{U}(0,1)$;
2. Faça $R^2 = -2 \log U_1$ (distribuição exponencial) e $S^2 = 2\pi U_2$ (distribuição uniforme);
3. Retorne $X = R\cos(S^2)$ e $Y = R\sin(S^2)$.

As ocorrências X e Y são ocorrências da distribuição normal padrão.

Gerando Normal

Algoritmo do método de Box-Muller

1. Gere $U_1, U_2 \sim \mathcal{U}(0,1)$;
2. Faça $R^2 = -2\log U_1$ (distribuição exponencial) e $S^2 = 2\pi U_2$ (distribuição uniforme);
3. Retorne $X = R\cos(S^2)$ e $Y = R\sin(S^2)$.

As ocorrências X e Y são ocorrências da distribuição normal padrão.

Nota: Perceba que em cada execução completa do algoritmo geramos duas novas observações de $X \sim \mathcal{N}(0,1)$.

Gerando Normal

Algoritmo pelo método polar:

Gerando Normal

Algoritmo pelo método polar:

1. Gere $U_1, U_2 \sim \mathcal{U}(0, 1)$;

Gerando Normal

Algoritmo pelo método polar:

1. Gere $U_1, U_2 \sim \mathcal{U}(0,1)$;
2. Transforme $U_1 = 2U_1 - 1$, $U_2 = 2U_2 - 1$ e faça $W = U_1^2 + U_2^2$;

Gerando Normal

Algoritmo pelo método polar:

1. Gere $U_1, U_2 \sim \mathcal{U}(0,1)$;
2. Transforme $U_1 = 2U_1 - 1$, $U_2 = 2U_2 - 1$ e faça $W = U_1^2 + U_2^2$;
3. Se $W > 1$, volte ao primeiro passo;

Gerando Normal

Algoritmo pelo método polar:

1. Gere $U_1, U_2 \sim \mathcal{U}(0,1)$;
2. Transforme $U_1 = 2U_1 - 1$, $U_2 = 2U_2 - 1$ e faça $W = U_1^2 + U_2^2$;
3. Se $W > 1$, volte ao primeiro passo;
4. Retorne

$$X = \sqrt{\frac{-\log W}{W}} \times U_1 \text{ e } Y = \sqrt{\frac{-\log W}{W}} \times U_2.$$

Gerando Normal

Algoritmo pelo método polar:

1. Gere $U_1, U_2 \sim \mathcal{U}(0,1)$;
2. Transforme $U_1 = 2U_1 - 1$, $U_2 = 2U_2 - 1$ e faça $W = U_1^2 + U_2^2$;
3. Se $W > 1$, volte ao primeiro passo;
4. Retorne

$$X = \sqrt{\frac{-\log W}{W}} \times U_1 \text{ e } Y = \sqrt{\frac{-\log W}{W}} \times U_2.$$

A vantagem do algoritmo é que não requer avaliação de funções trigonométricas. Porém, a desvantagem é que normalmente precisamos gerar mais de duas uniformes para obter duas ocorrências gaussianas.

Gerando Normal

Exercício: Implemente uma função para geração de números pseudo-aleatórios com distribuição normal padrão. A função deverá implementar o método de Box-Muller e o método polar. Ao final obtenha um histograma com os números gerados (mil valores) e realize um teste de normalidade.