

Aula 7

Aula passada

- Método de Monte Carlo
- Estimando somatórios
- Calculando erro
- Estimando π
- Erro de π
- Integração de Monte Carlo
- Monte Carlo Ray Tracing

Aula de hoje

- Gerando amostras de v.a. discretas
- Gerando Geométrica
- Método da transformada inversa
- Gerando Binomial
- Gerando permutações

Gerando Amostras

- Aula passada usamos apenas gerador uniforme
- Como gerar amostras de v.a. não uniforme

Usar a uniforme!

- **Premissa:** gerador de v.a. uniforme contínua [0,1] disponível

Premissa é (ligeiramente) falsa!

- 1) número não é contínuo: representação discreta de números no computador (ex. 32 bits) *Não podemos gerar, por exemplo, o número $\sqrt{2}$*
- 2) gerador não é aleatório: computador é determinístico (um algoritmo gera o número aleatório)

Gerador Pseudo-aleatório!

Gerador Pseudo-Aleatório

- Problema 1 é contornado usando maior precisão
 - ex. 64 bits divide intervalo $[0,1]$ em 2^{64} pedacinhos
- Problema 2 é contornado usando algoritmos que misturam os bits com manipulações algébricas
 - geram sequência de números no qual o próximo depende do anterior (ou anteriores)
 - passam em muitos testes estatísticos para aferir aleatoriedade
 - ex. Mersenne-Twister (1997) um dos mais usados algoritmos para geração de números pseudo-aleatório

**Premissa é falsa na teoria
mas contornável na prática!**

O algoritmo que gera o número pseudo-aleatório deve ser rápido, pois chamaremos o mesmo diversas vezes.

Figueiredo 2018

Gerando Outras Distribuições

- Assumindo gerador de amostras de uma v.a. uniforme contínua $[0,1]$
 - $U \sim \text{unif}(0,1) \rightarrow U$ é uma amostra da uniforme

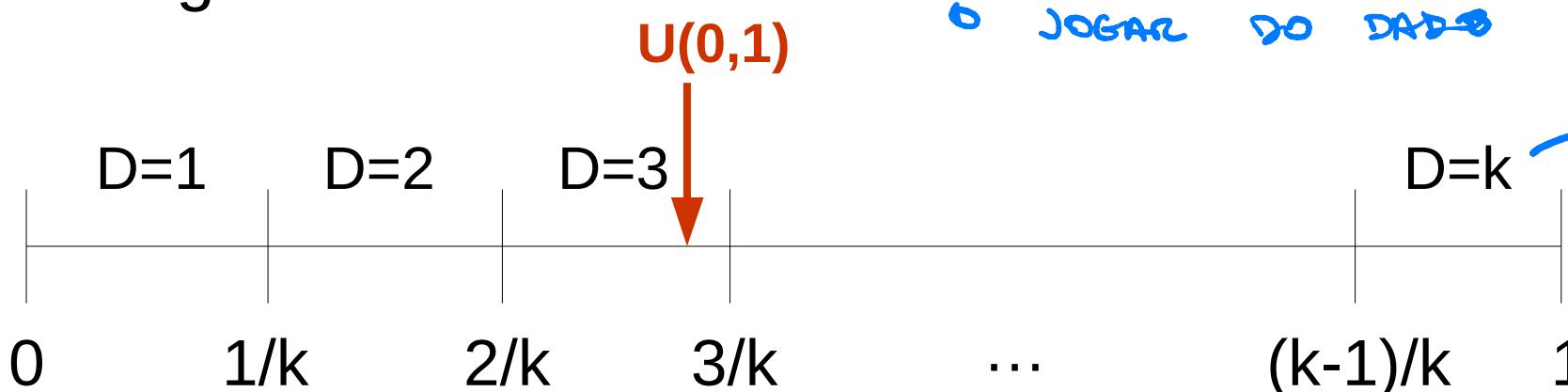


Como gerar v.a. com outras distribuições?

- Ex. como gerar uma amostra de uma v.a. geométrica com parâmetro p ?
- Muitas técnicas e algoritmos
 - diferentes complexidade e aplicabilidade
- Melhor abordagem em geral depende da distribuição a ser gerada

Lançando um Dado

- Considere um dado honesto com k faces
- Seja D o valor da face ao lançar o dado
 - $P[D = i] = 1/k$ (DADO HONESTO)
- Como gerar valor da face? → Algoritmo simula o JOGAR DO DADO



- Gerar U . Determinar i , tal que $(i-1)/k < U \leq i/k$

$U = \text{unif}(0,1); i=1;$

`while(! ((i-1)/k < U <= i/k)) i++;`

`return i;`

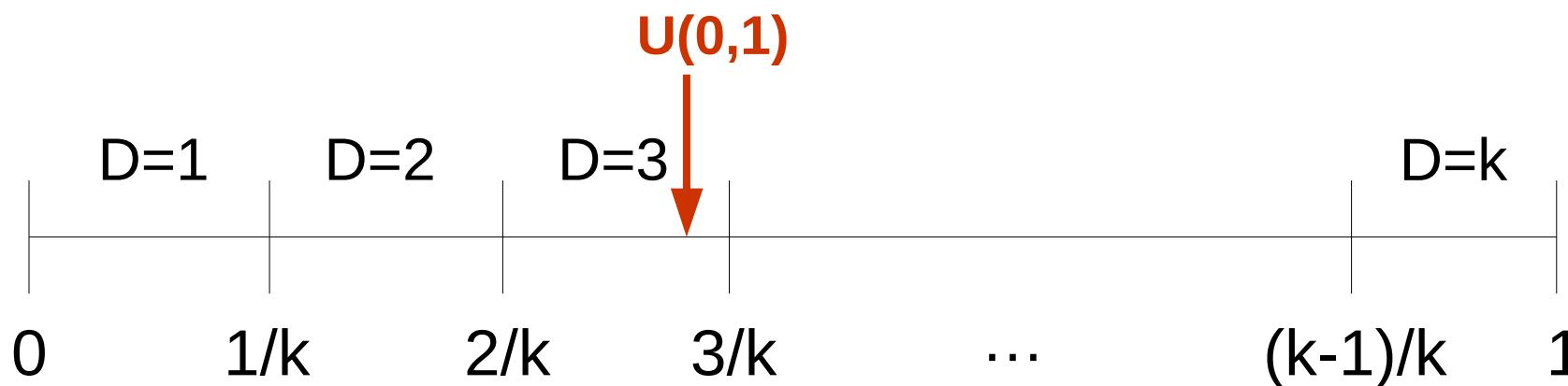
- Complexidade? Em geral, a complexidade é calculada para o pior caso. Logo, a complexidade deste algoritmo é $O(k)$



Dividir o intervalo $[0,1]$ em k pedaços, atribuir um índice a cada intervalo e gerar uma amostra de VA.

Lançando um Dado

- Podemos tirar proveito que o dado é honesto



- Algoritmo mais eficiente, determina i diretamente

$U = \text{unif}(0,1);$

$i = \text{int}(kU) + 1;$ $\leftarrow \text{int}(r)$: retorna parte inteira de r

$\text{return } i;$ *Gera um número pseudo aleatório entre 0 e k*

- Complexidade?

$\mathcal{O}(1)$

Lançando outro Dado

- Considere que dado com k faces não é honesto
- Seja D o valor da face ao lançar o dado
 - prob. da face i é proporcional a w_i ,

$$P[D=i] = \frac{w_i}{W} \quad W = \sum_{i=1}^k w_i$$

- Como gerar valor da face? Generalização do algoritmo inicial resolve!
 - Gerar U . Determinar i , tal que $\sum_{j=1}^{i-1} w_j < WU \leq \sum_{j=1}^i w_j$
- Generalizações do i^{th} algoritmo
separando os intervalos em w_i/W
- ```
U = unif(0,1); s = 0; i=1;
while (s <= WU) s += w[i]; i++;
return i;
```
- Complexidade?  $O(k)$



$O$  algoritmo independe de  $W$

Figueiredo 2018

# Lançando Dado Eficiente

$$P[D=i] = \frac{w_i}{W} \quad W = \sum_{i=1}^k w_i$$

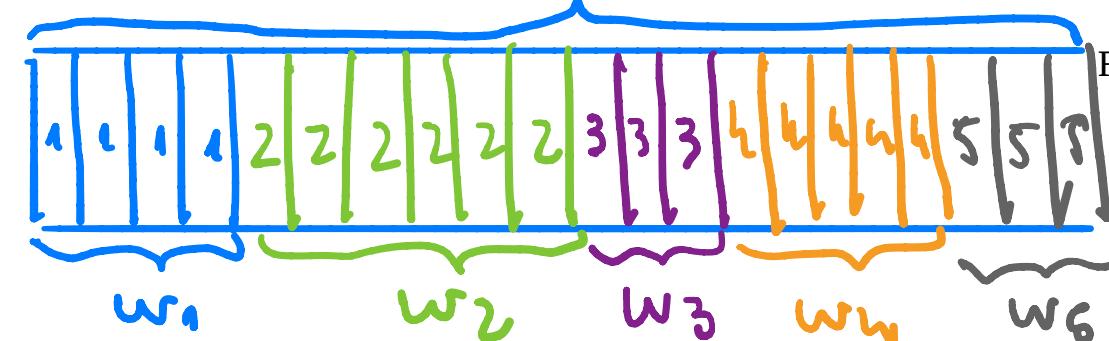
- Considere  $w_i$  inteiro. Como gerar v.a. de forma mais eficiente? Em  $O(1)$ ?
  - Ideia: pré-processamento
    - Gasto um tempo w para alocar o vetor θ
      - 1) alocar um vetor de tamanho  $W$
      - 2) preencher  $w_i$  posições com valor  $i$
  - Para escolher face do dado
    - 1) escolher uniforme inteiro em  $[1, W] \rightarrow \text{INT}(w)$
    - 2) acessar vetor para retornar face
  - Complexidade? Memória?
  - Truque muito usado!

A chance de sair 1 depende apenas da qtd de 1 ( $w_1$ ), e não da ordem com que eles estão alocados

Fazemos trade-off entre tempo e memória

Como as amostras são independentes, podemos embaralhar o vetor de resultados em random

Figueiredo 2018



# *Alias Method*

- Método eficiente para gerar D (DADO)
  - não assume  $w_i$  inteiros, mas precisa de  $W$
  - usa memória  $\Theta(k)$ , dois vetores
- Pré-processamento (criação dos vetores)
  - $O(k \log k)$  ou  $O(k)$ , dependendo do método
- Geração de número aleatório usando vetores
  - $O(1)$  – duas escolhas uniformes
- Muito usado para acelerar geração de sequência iid com muitas opções em cada escolha
  - ex. escolher aluno da UFRJ com probabilidade proporcional ao CRA,  $w_i$

#faces = #alunos

Algoritmo NRD - APRESENTADO

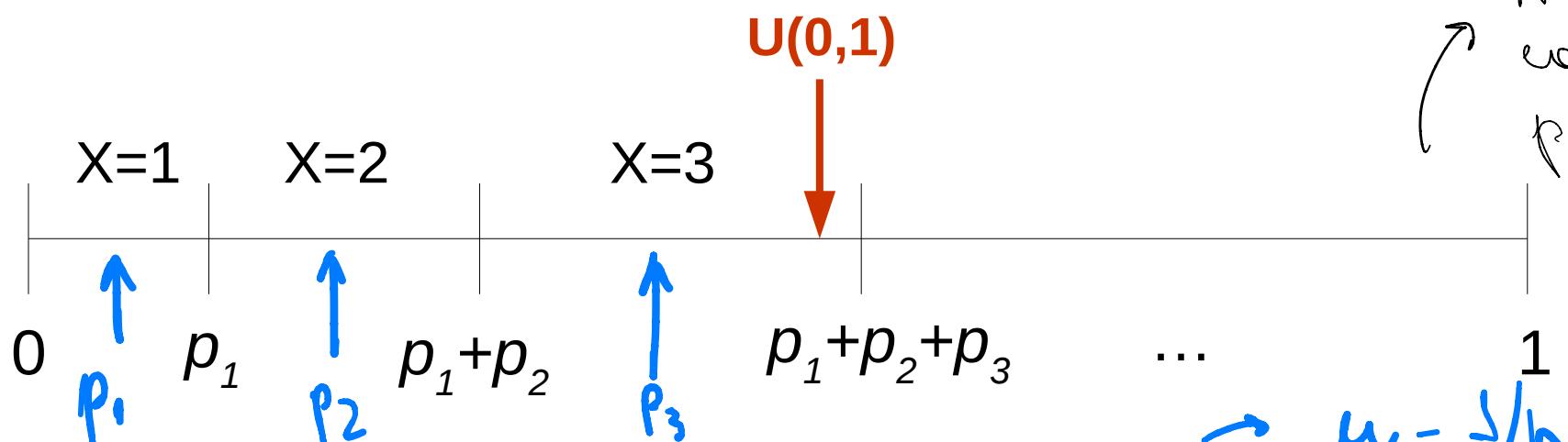
Figueiredo 2018

# Gerando Geométrica

- Seja  $X$  uma v.a. com distribuição geométrica,  $p$

$$p_i = P[X=i] = (1-p)^{i-1} p, i=1,2,\dots$$

- Como gerar valores para  $X$ ?
- Abordagem 1: adaptação do anterior (sem limite superior para valor que v.a. pode assumir)



- Complexidade? Caso médio é  $O(1/p)$

Como a distribuição não tem limite superior, em teoria o prior caso teria infinitas probabilidades. Na prática, venho a complexidade no caso médio

Figueiredo 2018

# Gerando Geométrica

- Abordagem 2: Geométrica é sequência de Bernoulli até ocorrência de positivo
- Gerar uma sequência de Bernoulli( $p$ ) até observar evento positivo

DADO DE 2 FACES

$e = 0; c = 1;$

while (! e) if (unif(0,1) <= p)  $e=1$  else c+=1;

return c;

aumenta o contador

- Vantagem: não calcula valor  $p_i$  da Geométrica

$$p_i = (1-p)^{i-1} \cdot p$$

- Desvantagem: gera muitas uniformes

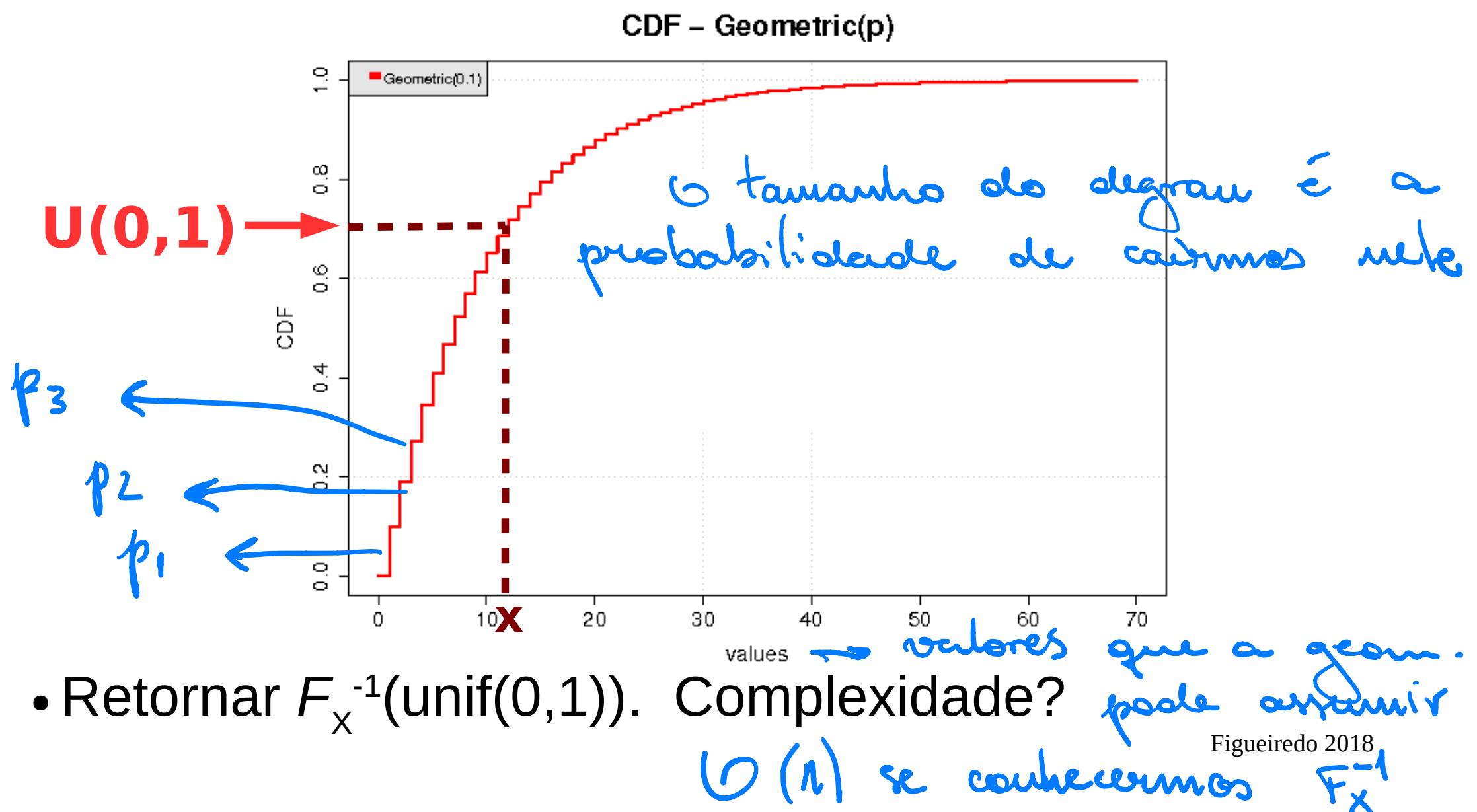
Se  $i \rightarrow \infty$ , o cálculo demora

- Complexidade?

Gera uma unif.  
ou caíde rodada

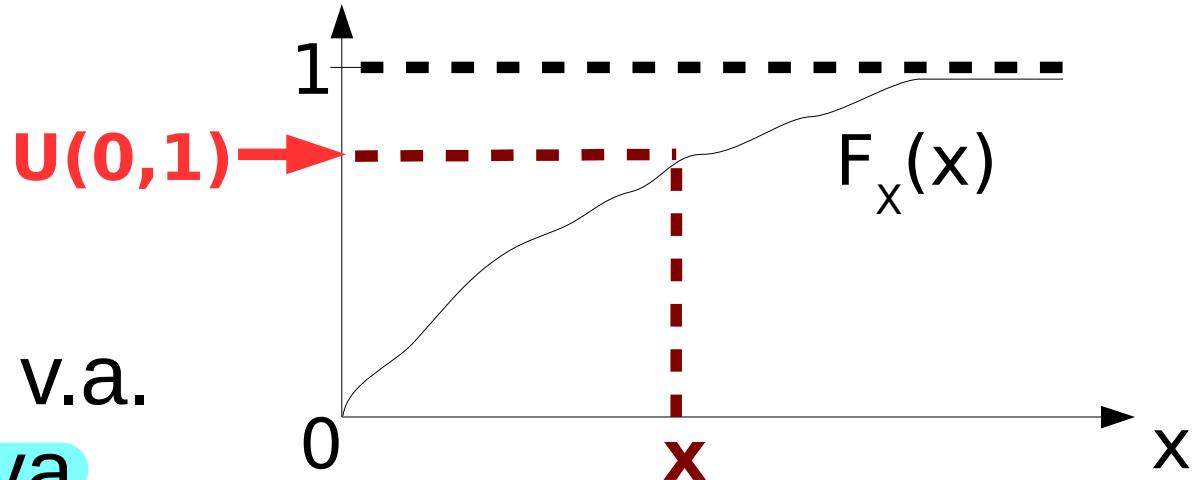
# Gerando Geométrica

- Abordagem 3: Usar a inversa da função de distribuição cumulativa,  $F_x(x)$



Funciona para qualquer distribuição

## Método da Transformada Inversa



- Seja  $U \sim \text{unif}(0,1)$ , e  $X$  v.a. com função cumulativa  $F_x(x) = P[X \leq x]$
- Temos que  $X = F_x^{-1}(U)$  → Basta sabermos  $F_x^{-1}$  (que é determinística)
- onde  $F_x^{-1}$  é a inversa de  $F_x$  (valor de  $x$  para qual  $F_x(x) = u$ )
- Podemos usar a inversa para gerar amostras!

# Método da Transformada Inversa

- Prova da equivalência

$$X = F_X^{-1}(U) \quad U \sim \text{Unif}[0,1]$$

- Mostrar que  $X$  acima de fato tem distribuição  $F_X(x)$

$$P[X \leq x] = P[F_X^{-1}(U) \leq x] \quad \leftarrow \text{por definição de } X$$

*Aplico  $F_X(x)$  em ambos os lados. Pela monotonicidade, não altera a desigualdade*

$$\begin{aligned} &= P[F_X(F_X^{-1}(U)) \leq F_X(x)] \quad \leftarrow \text{Equivalência de eventos e monotonicidade de } F_X \\ &= P[U \leq F_X(x)] \\ &= F_X(x) \quad \leftarrow \text{Prob. da uniforme ser menor que } \beta \text{ é igual a } \beta \text{ (} \beta \text{ em } [0,1]\text{)} \end{aligned}$$

- Mas temos que obter a função inversa!

\*  $P[U \leq \beta] = \beta \rightarrow$

Figueiredo 2018

CDF DA GEOMÉTRICA:  $F_X(i) = 1 - (1-p)^i$

## Inversa da Geométrica

- Seja  $X$  uma v.a. com distribuição geométrica,  $p$

$$F_X(i) = P[X \leq i] = 1 - (1-p)^i, i = 1, 2, \dots$$

$$\begin{aligned} F_X^{-1}(u) &=? & 1 - (1-p)^i &= u \\ f(x) &= y & \rightarrow (1-p)^i &= 1 - u \\ \downarrow f^{-1} & & \rightarrow i &= \frac{\log(1-u)}{\log(1-p)} \end{aligned}$$

← Se  $u$  é unif(0,1) então  $1-u$  também é unif(0,1)

$$f(y) = x$$

$$F_X^{-1}(u) = \underset{\text{PISO}}{\text{integer}}\left(\frac{\log u}{\log(1-p)}\right) + 1$$

←  $i$  é inteiro, maior que zero

- Gerar  $u \sim \text{unif}(0,1)$  e aplicar fórmula acima
- Complexidade:  $O(1)$

# Gerando Binomial

- Seja  $X \sim \text{Binom}(N, p)$

CDF DA  
binomial

$$F_X(i) = P[X \leq i] = \sum_{k=0}^i \binom{N}{k} p^k (1-p)^{N-k}$$

$= P[X = 0] + P[X = 1] + \dots + P[X = i]$   
 $= \sum_{k=0}^i P[X = k]$

- Não temos como inverter analiticamente  $F_X$
- Alternativas?
- Gerar sequência iid de  $N$  Bernoulli( $p$ ), contar quantos eventos foram positivos
- **Problema:**  $p$  muito baixo,  $N$  grande, teremos muitos zeros
  - ex.  $p=10^{-3}$ ,  $N=10^5$
  - Ideia: Usar geométrica para acelerar geração de 1

Uma binomial é uma soma de  $N$  Bernoullis. O número de bernoullis que der 1 é o valor da binomial

Figueiredo 2018

A posição desse 1  
é o valor da  
geométrica

Geom(1) + Geom(2)

## Gerando Binomial

- Sequência de N v.a. Bernoullis( $p$ ) iid  
00000010001010100000001000001000100000000100
- Distribuição da posição do primeiro valor 1?  $\leftarrow \text{Geométrica}(p)$
- Distribuição da posição do segundo valor 1?  $\leftarrow \text{Geométrica}(p)$ , a partir do primeiro valor 1!
- Número de 1 na sequência de tamanho  $N$  é igual ao número de geométricas que somadas ocorreram antes de  $N$

$$X = k | \min_k \sum_{i=1}^{\infty} G_i \geq N \quad \leftarrow \text{onde } G_i \sim \text{Geom}(p)$$

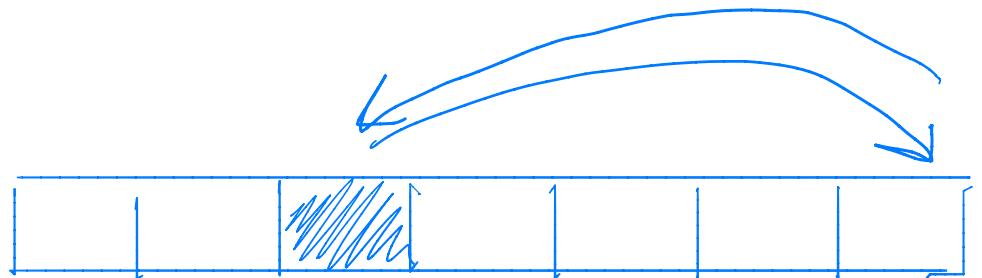
- mínimo de parcelas que precisamos para produzir  $N$  quando somar as geométricas
- Algoritmo: gerar sequência iid Geom( $p$ ) até soma ser  $\geq N$
  - Complexidade? Quantas geométricas serão geradas, na média?
  - $N/(1/p) = Np$

No pior caso,  $G_i = 1$ , então terímos que fazer  $N$  somas. Cada soma tem complexidade  $1/p$ .

Valor esperado da Binomial

# Gerando Permutações

- Aula passada: gerar uma permutação da cartas do baralho, com probabilidade uniforme
- **Ideia 1**
  - Gerar  $i$  uniforme entre 1 e  $52!$  (todas permutações)
  - Retornar a  $i$ -ésima permutação
- Problema? ← Como saber qual é a  $i$ -ésima permutação?
- **Ideia 2**
  - Escolher um elemento por vez de forma uniforme (sem repetição)
  - Fazer  $n-1$  escolhas sucessivas
- Problema? ← Como fazer escolhas de forma eficiente?



Escolhe um índice aleatório, escolhe para a permutação o número desse índice é coloca o número no final da lista.

## Gerando Permutações

- **Ideia:** usar um vetor para alocar elementos e as escolhas realizadas (conhecido por *Knuth Shuffle*)
- Algoritmo eficiente para gerar uma permutação de N elementos de forma uniforme

```

vetor permuta[1,...,N];
permuta[i] = i para i=1,...,N;
para i=0 até N-1
 j = unif(1, N-i); ← O(1)
 tmp = permuta[j];
 permuta[j] = permuta[N-i];
 permuta[N-i] = tmp;
}

```

coloca o número  
escolhido no fim do  
vetor

- Funciona?
- Complexidade?  $O(n)$

**Knuth Shuffle:** Algoritmo eficiente para gerar permutações de  $n$  elementos de forma que cada permutação tenha a mesma prob. de ser gerada.

Figueiredo 2018