

Aula 7

Roteiro

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

Aula passada

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



Motivação

O que é gerar amostra de uma variável aleatória?

- Realizar, observar um fenômeno aleatório
 - jogar uma moeda, jogar um dado, retirar bola de uma urna, construir um grafo, etc

Como fazer isso usando o computador?

- Tema da aula de hoje

Por que gerar amostras de v.a.?

- Implementar métodos de Monte Carlo, simular sistemas aleatórios, construir jogos, projetar algoritmos, etc

Gerando Amostras Uniformes

- Seja U v.a. contínua com distribuição uniforme em $(0, 1]$
- **Premissa:** computador consegue gerar amostras de U

Mas a premissa é falsa!

- 1) número gerado não é contínuo: representação discreta de números no computador (ex. 32 bits)
- 2) gerador não é aleatório: computador é determinístico, e um algoritmo vai gerar 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 e independência
 - ex. Mersenne-Twister (1997) um dos mais usados algoritmos para geração de números pseudo-aleatórios

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

Gerando Outras Distribuições

- Temos então gerador de $U \sim \text{unif}(0,1)$

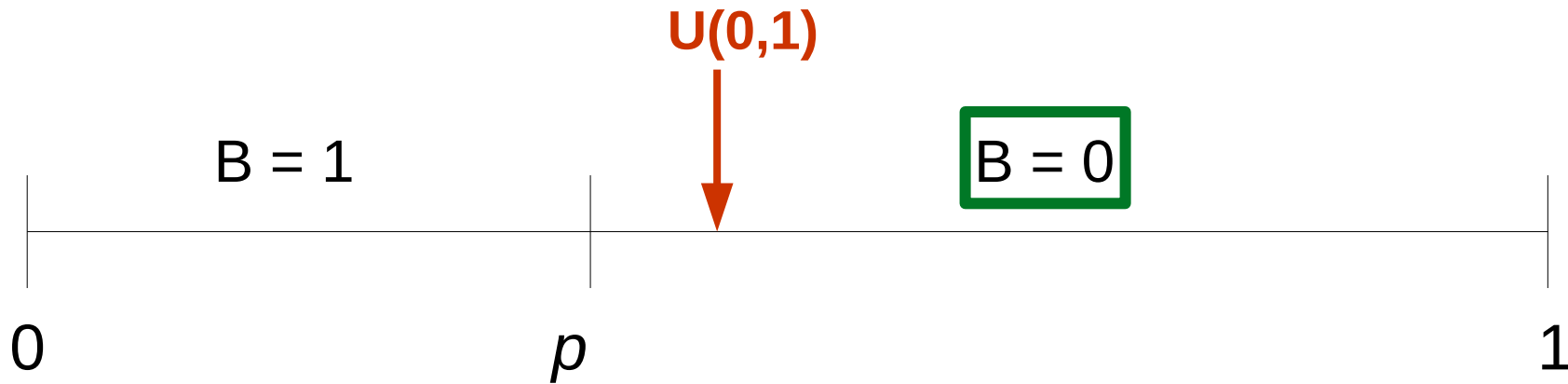
Como gerar v.a. com outras distribuições?



- Muitas técnicas e algoritmos
 - diferente complexidade e aplicabilidade
- Melhor abordagem em geral depende da distribuição a ser gerada

Lançando uma Moeda

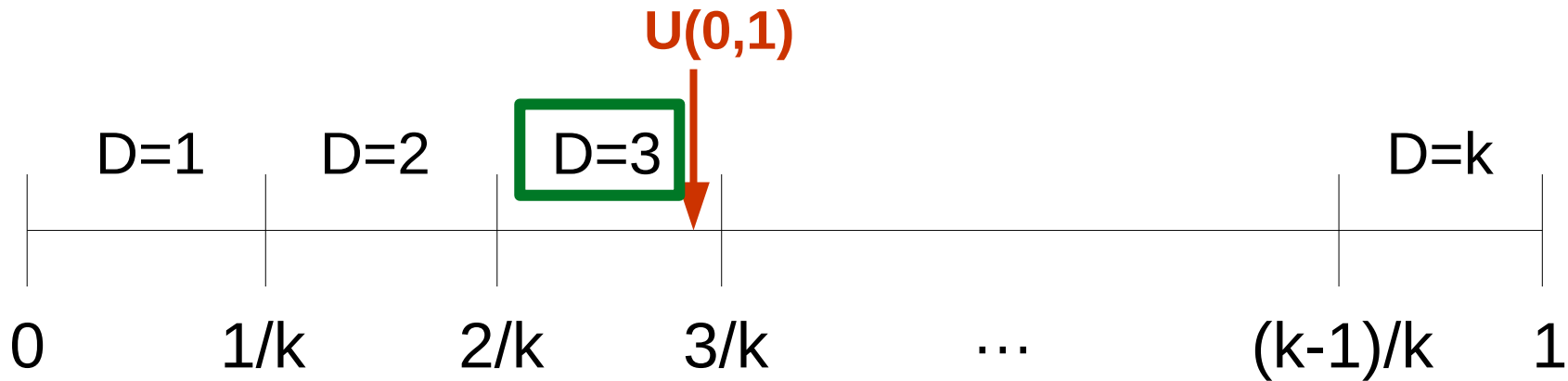
- Considere uma moeda enviesada
- Seja $B \sim \text{Bernoulli}(p)$ v.a. que representa a moeda
 - $P[B = 1] = p$
- Como gerar a face da moeda (0 ou 1)?



- Dividir intervalo $[0, 1]$ em $p, 1-p$
- Gerar U uniforme
- Se $U \leq p$ retorna 1, caso contrário retorna 0

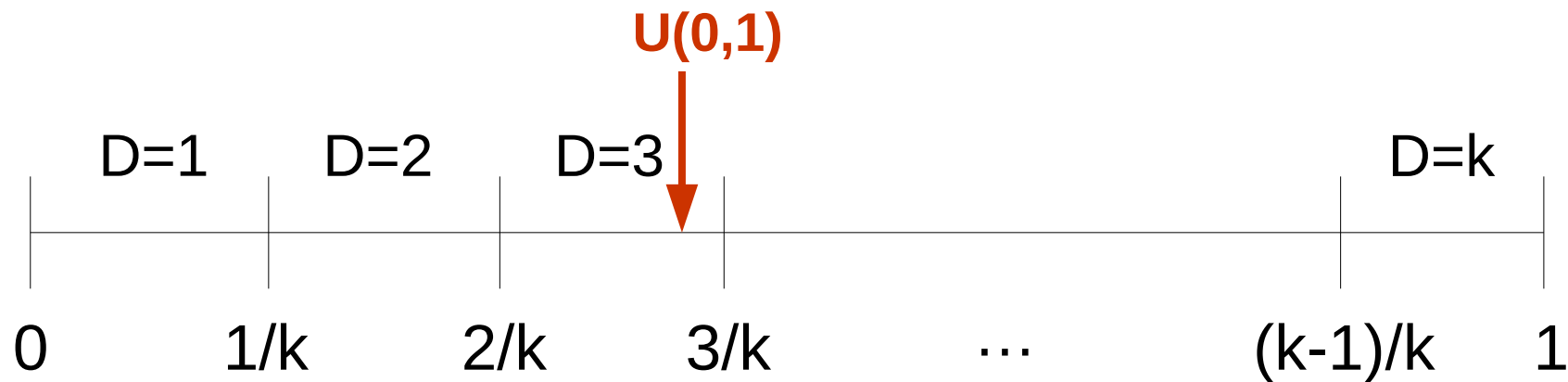
Lançando um Dado

- Considere um dado honesto com k faces
- Seja D v.a. que denota o valor da face
 - $P[D = i] = 1/k$
- Como gerar valor da face?



- Dividir intervalo $[0, 1]$ em k faixas
- Gerar U uniforme
- Encontra a faixa que possui o valor gerado de U
- Retorna índice da faixa

Algoritmo 1 para Gerar Dado

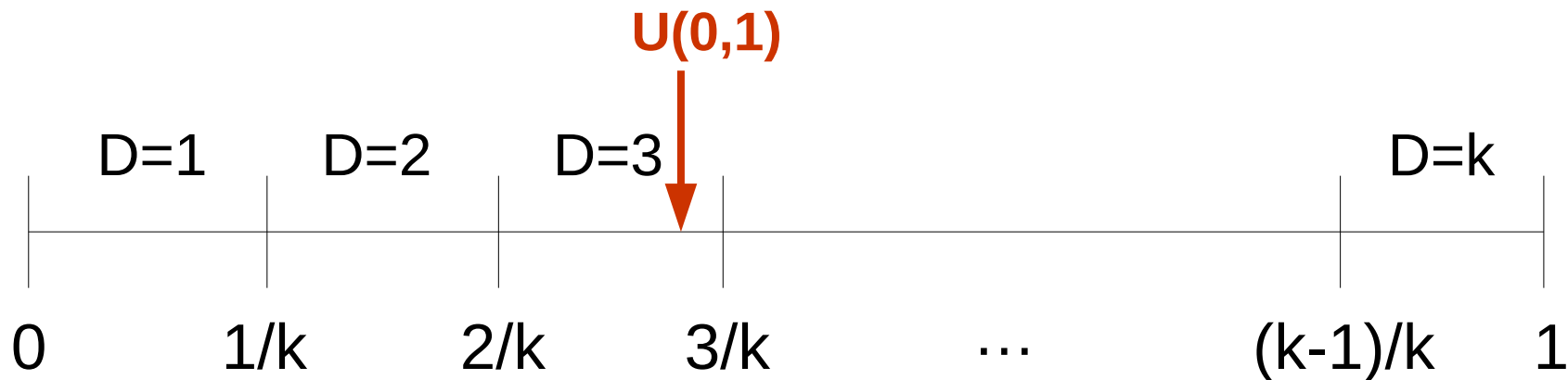


- Algoritmo
 $U = \text{unif}(0,1)$
 $i = 1$
 while(! (($(i-1)/k < U \leq i/k$)) { $i = i + 1$ }
 return i
- Complexidade? Quantas vezes passa pelo *loop*?
- Melhor caso: 1 – Pior caso: k – Caso médio: $k/2$

Poderia ser mais eficiente?

Algoritmo 2 para Gerar Dado

- Podemos tirar proveito do dado ser honesto



- Algoritmo mais eficiente, determina i diretamente, multiplicando U por k

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

$i = \text{int}(k \cdot U) + 1$ ← $\text{int}(r)$: retorna parte inteira de r
return i

- Complexidade?
- Tempo constante, independente de k e do valor de U

Muito eficiente!

Lançando outro Dado

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

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

- Generalização do primeiro algoritmo gera amostras
- Gerar U . Determinar i , tal que

$$\sum_{j=1}^{i-1} w_j < WU \leq \sum_{j=1}^i w_j$$

```
U = unif(0,1); s = 0; i=1;
while (s <= WU) {
    s += w[i]; i++;
} return i;
```

Lançando Dado Eficiente

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

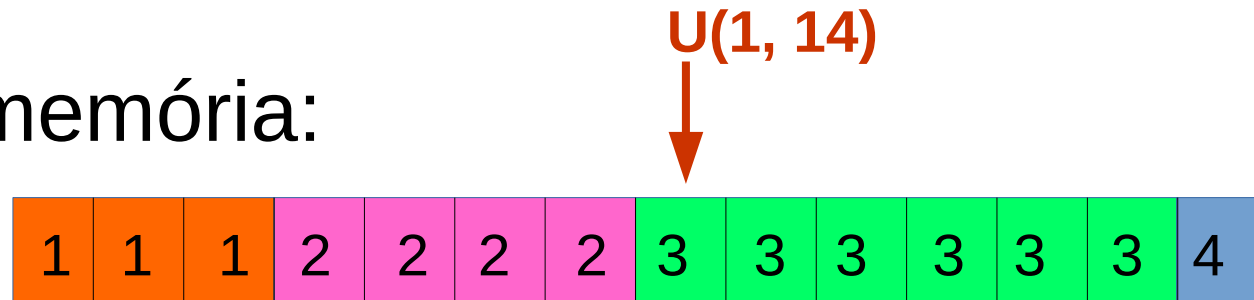
- Considere w_i inteiro. Como gerar v.a. de forma eficiente? Em tempo constante?
- **Ideia:** pré-processamento
 - 1) alocar um vetor de tamanho W
 - 2) preencher w_i posições com valor i
- Para gerar amostra do dado
 - 1) escolher inteiro uniforme em $[1, W]$
 - 2) acessar vetor para retornar face

Exemplo

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

- $k = 4, w_1 = 3, w_2 = 4, w_3 = 6, w_4 = 1$
- $W = 3 + 4 + 6 + 1 = 14$

- Vetor na memória:



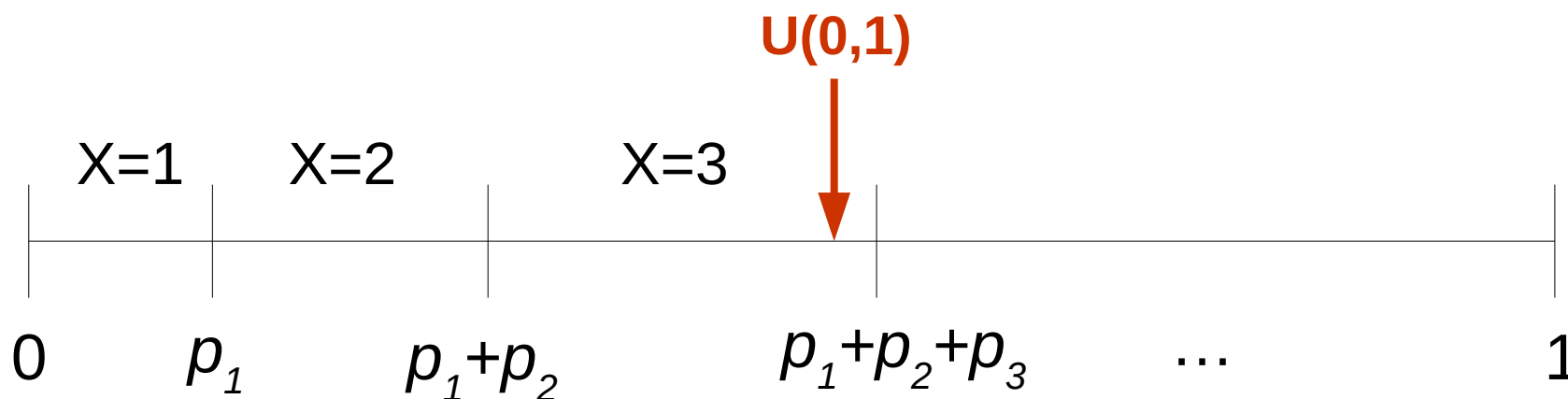
- Retorna conteúdo do valor acessado
- Complexidade de W , para criar vetor (uma vez)
- Complexidade constante, para gerar amostra

Alias Method

- Método eficiente para gerar D
 - 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
- Utilizado para acelerar geração de sequência iid longas quando k é grande

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 $O(1/p)$

Gerando Geométrica

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

$e = 0; c = 1;$

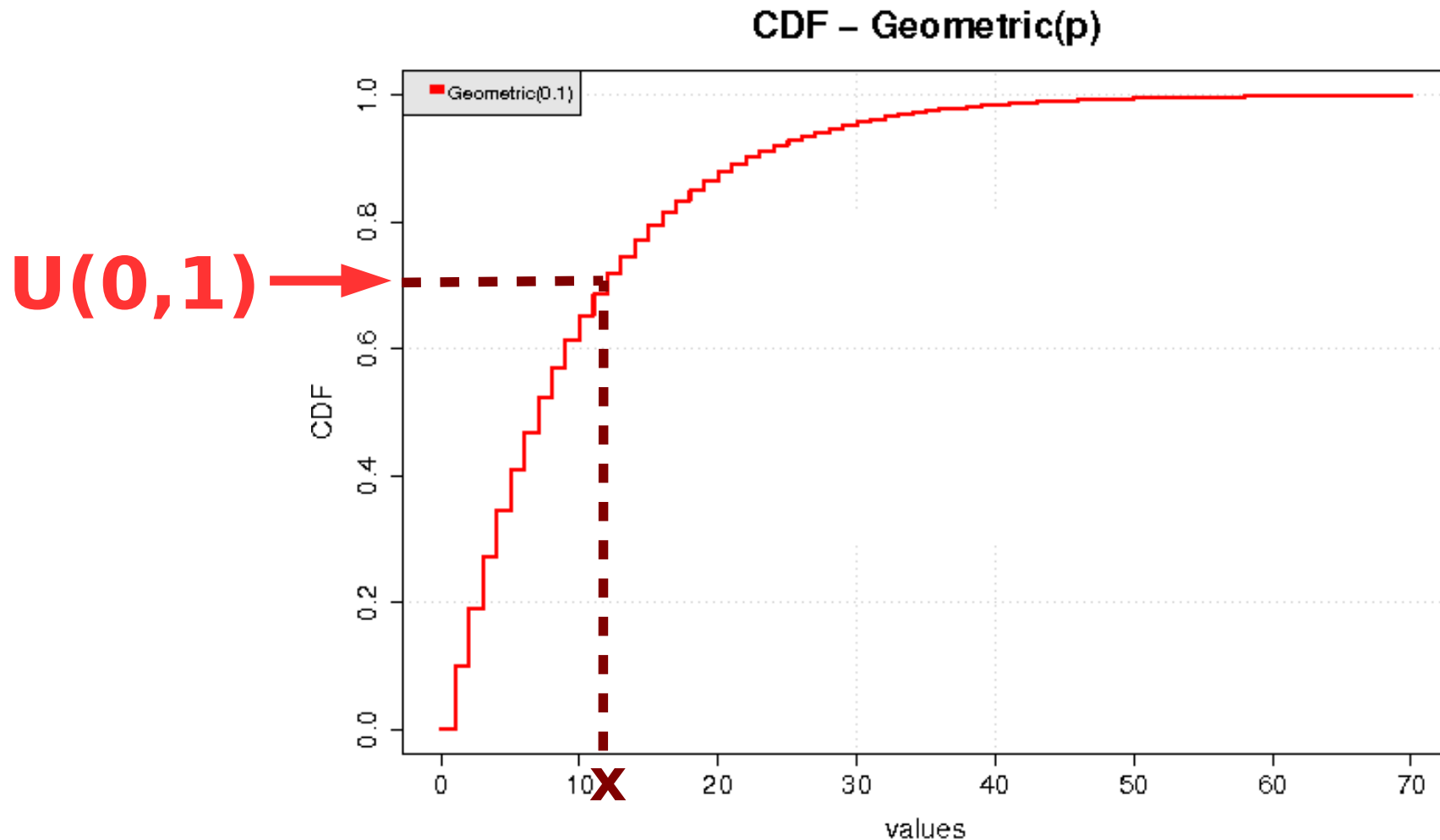
while (! e) if (unif(0,1) $\leq p$) $e=1$ else $c+=1;$

return c;

- Vantagem: não calcula valor p_i da Geométrica
- Desvantagem: gera muitas amostras uniformes
- Complexidade?

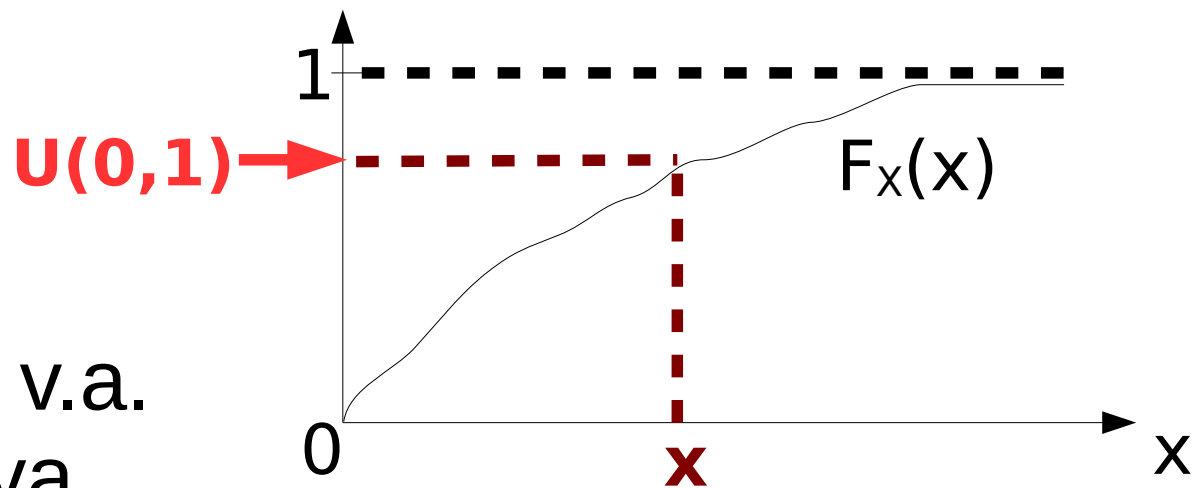
Gerando Geométrica

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



- Retornar $F_X^{-1}(\text{unif}(0,1))$. Complexidade?

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

$$X = F_X^{-1}(U)$$

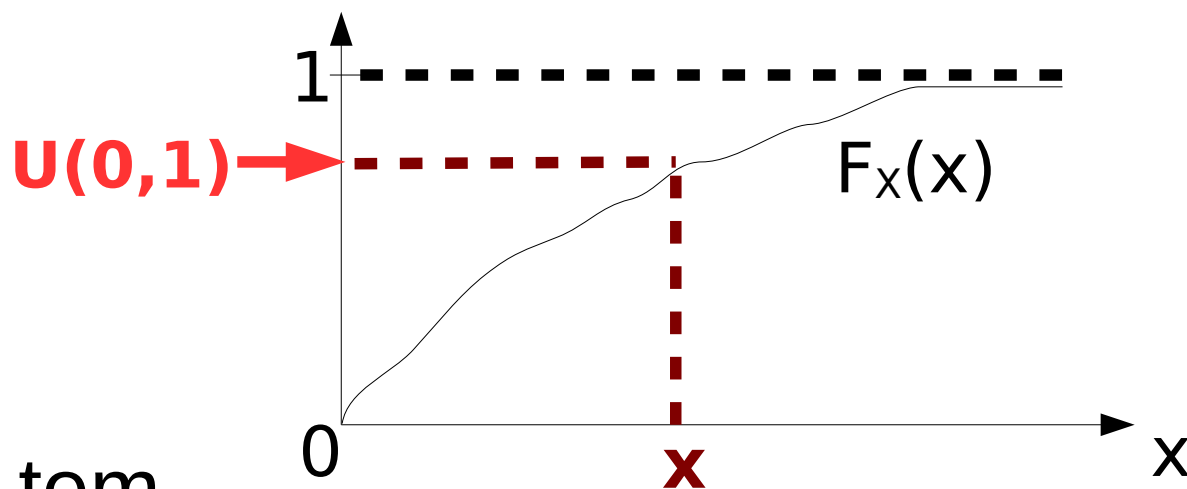
- onde F_X^{-1} é a inversa de F_X (valor de x para qual $F_X(x) = u$)
- Podemos usar a inversa para gerar amostras!

Método da Transformada Inversa

- Prova da equivalência

$$X = F_X^{-1}(U)$$

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



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

- Mas temos que obter a função inversa!

Inversa da Geométrica

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

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

$$F_X^{-1}(u) = ?$$

$$1 - (1 - p)^i = u$$

$$\rightarrow (1 - p)^i = 1 - u$$

$$\rightarrow i = \frac{\log(1 - u)}{\log(1 - p)} \quad \leftarrow \text{Se } u \text{ é unif}(0,1) \text{ então } 1-u \text{ também é unif}(0,1)$$

$$F_X^{-1}(u) = \text{int} \left(\frac{\log u}{\log(1 - p)} \right) + 1 \quad \leftarrow \text{inteiro maior que zero}$$

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

Gerando Binomial

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

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

- Não temos como inverter analiticamente F_X
- Alternativas?
- Gerar sequência iid com 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

Gerando Binomial

- Sequência de N v.a. Bernoullis(p) iid

000000100000001000000010000010001000000000100

- Distribuição da posição do primeiro valor 1? ← Geométrica(p)
- Distribuição da posição do segundo valor 1? ← 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

$$X = k - 1 \mid \min_k \sum_{i=1}^k G_i > N \quad \leftarrow \text{onde } G_i \sim \text{Geom}(p)$$

- Algoritmo: gerar sequência iid $G_i \sim \text{Geom}(p)$ até soma ser $> N$
- Complexidade? Quantas geométricas serão geradas, na média?
- $N/(1/p) = Np$ ← valor esperado da Binomial

Gerando Permutações

- Gerar uma permutação das cartas do baralho, com probabilidade uniforme. Baralho tem N cartas.
- **Ideia 1**
 - Gerar i uniforme entre 1 e $N!$ (todas permutações)
 - Retornar a i -ésima permutação
- Problema? ← Como saber qual é a i -ésima permutação?
- **Ideia 2**
 - Escolher um elemento da permutação de forma iterativa e uniforme (sem repetição)
 - Fazer n escolhas sucessivas
- Problema? ← Como fazer escolhas de forma eficiente?

Knuth Shuffle

- **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 $i=1,\dots,N$; // inicialização

para $i=0$ até $N-1$

`j = unif(1, N-i);` $\longleftarrow O(1)$

`tmp = permuta[j];`

`permuta[j] = permuta[N-i];`

`permuta[N-i] = tmp;`

- Cada permutação gerada tem a mesma probabilidade
- Complexidade (de tempo e memória): $O(N)$

Exemplo

- *Knuth Shuffle* com $N = 14$
- Construir vetor ordenado na memória

1	2	3	4	5	6	7	8	9	10	11	12	13	14
---	---	---	---	---	---	---	---	---	----	----	----	----	----

- Escolher uniforme $[1, 14]$
- Permutar com último elemento

1	2	3	4	5	6	7	8	14	10	11	12	13	9
---	---	---	---	---	---	---	---	----	----	----	----	----	---

- Escolher uniforme $[1, 13]$
- Permutar com penúltimo elemento

1	2	3	4	13	6	7	8	14	10	11	12	5	9
---	---	---	---	----	---	---	---	----	----	----	----	---	---

- Escolher uniforme $[1, 12]$
- ...
- Ao final, vetor contém uma permutação uniforme!