

MPEI 2018-2019

Aula 10

Geração de números aleatórios
(e simulação)

Motivação (exemplos)

- Gerar strings “aleatórias” em que:
 - comprimento assume valores entre 1 e 10 e tendo cada comprimento a mesma probabilidade
 - o caracter em cada posição é uma das letras minúsculas ou maiúsculas do alfabeto português e tendo todas a mesma probabilidade
- Gerar strings em que quer as letras quer o comprimento assumem distribuições mais próximas da realidade
 - Comprimento seguindo uma distribuição Normal com média e variância estimada de um conjunto de textos
 - As letras seguem a distribuição para o Português
 - Que vimos numa aula anterior

Problema

- Para situações como as do exemplo, necessitamos de resolver o problema de gerar, ou simular, vectores aleatórios tendo uma determinada distribuição
- Nos primeiros tempos da simulação utilizavam-se métodos mecânicos para obter valores aleatórios
 - Moedas, dados, roletas, cartas
- Mais tarde utilizaram-se propriedades de dispositivos, como o ruído de um díodo
- Estes métodos foram abandonados na área da Computação/Informática e substituídos por algoritmos que se podem implementar facilmente em computador
 - São algoritmos determinísticos, pelo que é usual designar os números gerados por “pseudo-aleatórios”

Abordagens principais

- Gerar directamente
- Gerar número “aleatório” de uma distribuição uniforme (contínua) e transformar ...
 - Neste caso, torna-se necessário ser capaz de gerar variáveis aleatórias com a distribuição uniforme
 - Em geral distribuída entre 0 e 1
 - É a abordagem comum

Geração de variáveis aleatórias com distribuição uniforme entre 0 e 1

Método da Congruência (LCG)

- Os métodos mais comuns para gerar sequência pseudo-aleatórias usam os chamados *linear congruential generators*
- Estes geradores geram uma sequência de números através da *fórmula recursiva*

$$X_{i+1} = aX_i + c \pmod{m}$$

- Com X_0 sendo a “semente” (seed) e a , c , m (todos inteiros positivos) designados de multiplicador, incremento e módulo, respetivamente
- Como X_i pode apenas assumir os valores $\{0, 1, \dots, m-1\}$, os números

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

são designados por número pseudo-aleatórios e constituem uma aproximação a uma sequência de variáveis aleatórias uniformemente distribuídas

Processo de cálculo em detalhe

1. Escolher os valores de a , c e m
2. Escolher a semente X_0 (tal que $1 \leq X_0 \leq m$)
3. Calcular o próximo número aleatório usando a expressão $X_1 = (aX_0 + c) \bmod m$
4. Substituir X_0 por X_1 e voltar ao ponto anterior

Exemplo

- Fazendo $a=9$, $c=1$, $m=17$ e $X_0 = 7$

n	x_n	$y=9x_n+1$	$y \bmod 17$	$x_{n+1}/17$
0	$X_0=7$	$9*7+1=64$	13	$13/17 = 0.7647$
1	$X_1=13$	118	16	$16/17 = 0.9412$
2	$X_2=16$	145	9	0.5294
3	$X_3=9$	82	14	0.8235
4	$X_4=14$	127	8	0.4706

números pseudo-aleatórios inteiros entre 0 e 16 (=17-1)

números pseudo-aleatórios inteiros entre 0 e 1

Como escolher os parâmetros ?

- A sequência repete-se no máximo após m números
- Será, portanto, periódica com um período que não excede m
- Mas pode ser muito pior
 - Exemplo: $a=c=X_0=3$ e $m=5$ gera a sequência $\{3,2,4,0,3 \dots\}$ com período 4
- Apenas algumas combinações de parâmetros produzem resultados satisfatórios
 - Exemplo: Usar $m = 2^{31} - 1$ e $a = 7^5$ em computadores de 32 bits

Demo Matlab

- Exemplo: $a=c=X_0=3$ e $m=5$ gera a sequência $\{3,2,4,0,3 \dots\}$

```
function U=lcg(X0,a,c,m, N)
```

```
U=zeros(1,N);
```

```
U(1)=X0;
```

```
for i=2:N
```

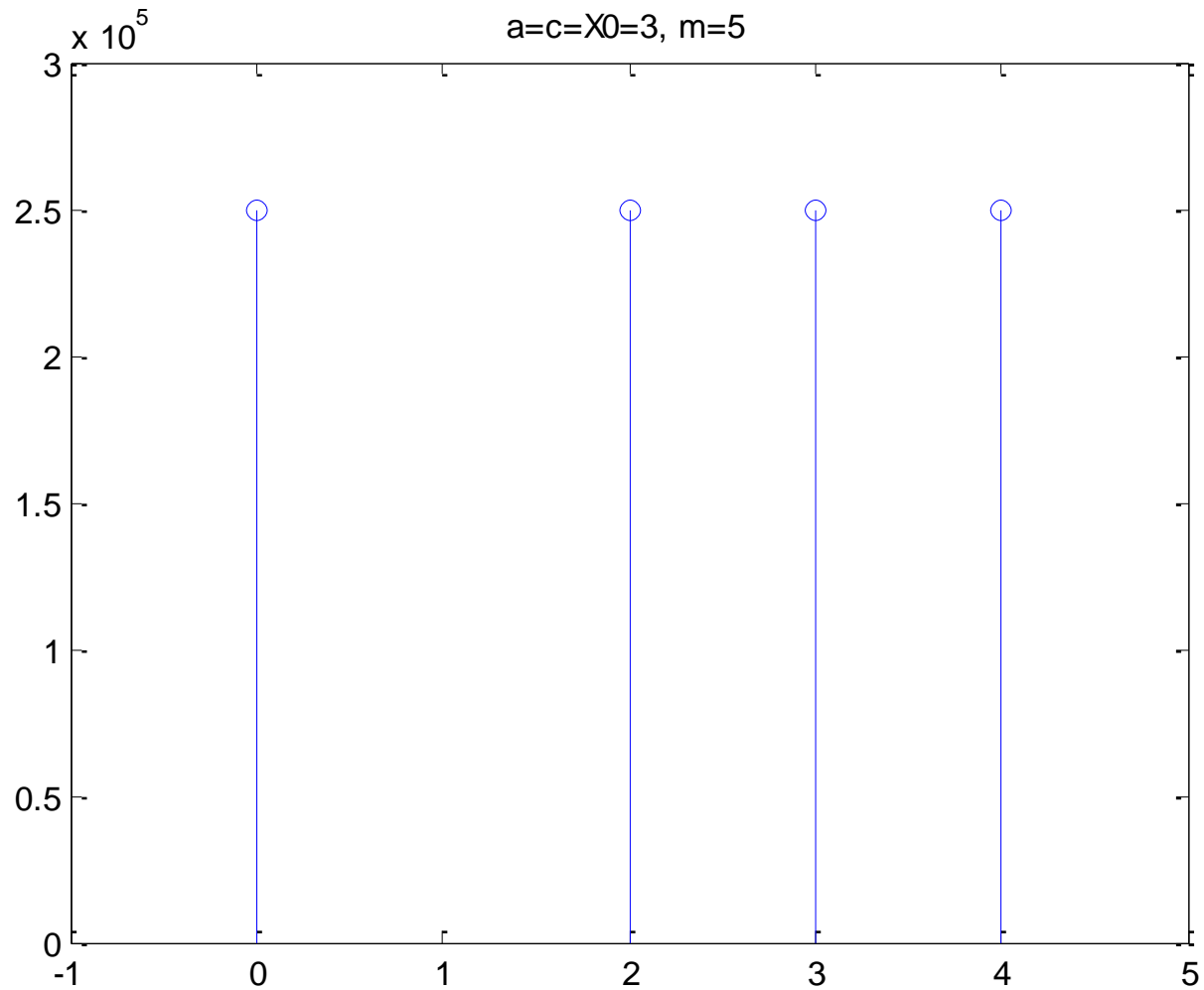
```
    U(i) = rem(a*U(i-1)+c, m);
```

```
end
```



demo1.m

Resultados - histograma



O gerador base da biblioteca NAG

- O LCG é o gerador base em bibliotecas como a NAG

2.1.1 NAG Basic Generator

The NAG basic generator is a linear congruential generator (LCG) and, like all linear congruential generators, has the form:

$$\begin{aligned}x_i &= a_1 x_{i-1} \bmod m_1, \\u_i &= \frac{x_i}{m_1},\end{aligned}$$

where the u_i , for $i = 1, 2, \dots$, form the required sequence.

The NAG basic generator uses $a_1 = 13^{13}$ and $m_1 = 2^{59}$, which gives a period of approximately 2^{57} .

This generator has been part of the NAG Library since Mark 6 and as such has been widely used. It suffers from no known problems, other than those due to the lattice structure inherent in all linear congruential generators, and, even though the period is relatively short compared to many of the newer generators, it is sufficiently large for many practical problems.

Método do Meio Quadrado

- Proposto por von Neumann (1946)
- Calcular o quadrado e reter a parte do meio dos números
 - Quando resultar zero deve utilizar-se nova semente
- Exemplo (simples):

$$X_0 = 76 \Rightarrow 76^2 = 5776$$

$$X_1 = 77 \Rightarrow 77^2 = 5929$$

$$X_2 = 92 \Rightarrow 92^2 = \dots$$

Gera {76,77,92,46,11,12,14,...}

Outros geradores

- Para além destes geradores, outras classes foram propostas por forma a obter períodos mais longos e melhor aproximação à distribuição uniforme
- A biblioteca NAG, por exemplo, inclui vários:

Pseudorandom Numbers

2.1.1 NAG Basic Generator

2.1.2 Wichmann–Hill I Generator

2.1.3 Wichmann–Hill II Generator

2.1.4 Mersenne Twister Generator

2.1.5 ACORN Generator

2.1.6 L’Ecuyer MRG32k3a Combined Recursive Generator . .

Outros geradores - Exemplo

- Wichman-Hill I
- Usa uma combinação de 4 LCGs

This series of Wichmann–Hill base generators (see Maclaren (1989)) use a combination of four linear congruential generators and has the form:

$$\begin{aligned}w_i &= a_1 w_{i-1} \bmod m_1 \\x_i &= a_2 x_{i-1} \bmod m_2 \\y_i &= a_3 y_{i-1} \bmod m_3 \\z_i &= a_4 z_{i-1} \bmod m_4 \\u_i &= \left(\frac{w_i}{m_1} + \frac{x_i}{m_2} + \frac{y_i}{m_3} + \frac{z_i}{m_4} \right) \bmod 1,\end{aligned}\tag{1}$$

where the u_i , for $i = 1, 2, \dots$, form the required sequence. The NAG Library implementation includes 273 sets of parameters, a_j, m_j , for $j = 1, 2, 3, 4$, to choose from.

Na prática...

- A maioria das linguagens de computador disponibilizam geradores de números pseudo-aleatórios
 - Em geral o utilizador apenas fornece o valor da semente
- Java
 - Classe Random
 - `Random rnd = new Random();`
 - `rnd.nextDouble();`
- Matlab
 - Tem a função `rand()`

Em Matlab

```
>> rand                                % generate a uniform random number
    0.0196
>> rand                                % generate another uniform random number
    0.823
>> rand(1,4)                           % generate a uniform random vector
    0.5252    0.2026    0.6721    0.8381
rand('state',1234)                      % set the seed to 1234
>> rand                                % generate a uniform random number
    0.6104
rand('state',1234)                      % reset the seed to 1234
>> rand                                % the previous outcome is repeated
    0.6104
```



demo2.m

Demonstração do uso de rand()

N=1000

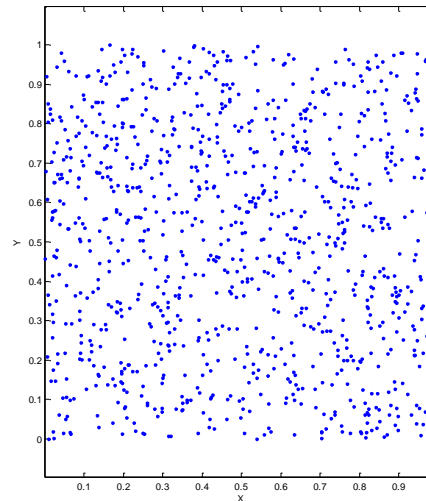
```
X = rand(1, N);  Y= rand(1,N);
```

```
subplot(121), plot(X,Y, '.')
```

```
axis equal
```

```
xlabel('X')
```

```
ylabel('Y')
```

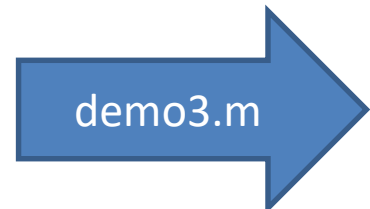
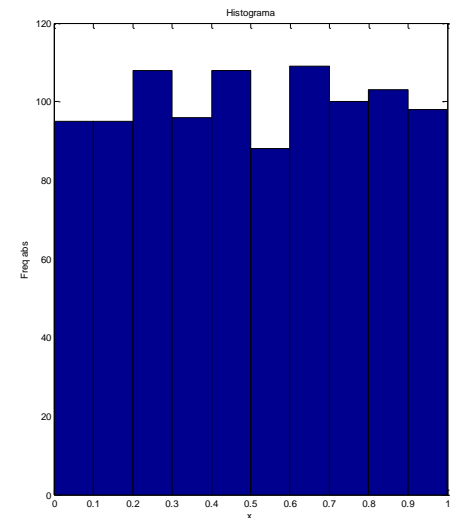


```
subplot(122), hist(x)
```

```
title('Histograma');
```

```
xlabel('x')
```

```
ylabel('Freq abs');
```



Transformações

Transformações simples

- Aplicando a de **transformação linear** $Y = a U + b$ é simples obter variáveis com distribuição uniforme num intervalo
ex: $Y = 2 U + 1$ permite intervalo $[1, 3]$
- A aplicação da transformação linear seguida da conversão para inteiros permite obter, por exemplo, uma simulação de lançamentos de um dado (**uma gama de números inteiros**)
 - Em versões mais recentes do Matlab existe mesmo a função `randi()`

Exemplos em Matlab

% geração de n resultados do lançamento de uma moeda

```
function Y=moeda(n)
```

```
if nargin ==0
```

```
    n=1;
```

```
end
```

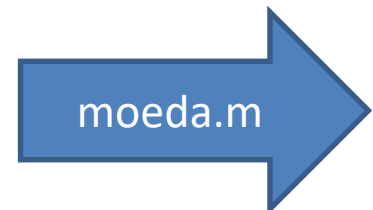
```
z=round(rand(1,n));
```

```
Y(1:n)='C';          % CARA
```

```
Y(find(z==0))='R';    % COROA
```

% usando

```
moeda(10)
```



Exemplos em Matlab

% n resultados do lançamento de um dado

```
function Y=dado(n)
```

```
if nargin==0
```

```
    n=1;
```

```
end
```

```
Y=floor(rand(1,n)*6)+1;
```

dado → 5

dado(10) → 3 1 4 5 6 3 4 3 2 4



dado.m

Métodos Genéricos para gerar variáveis aleatórias (unidimensionais)

Variáveis aleatórias com distribuições não uniformes

- Números aleatórios com outras distribuições podem ser obtidos das sequências com distribuição uniforme através de:
 - Métodos de transformação
 - Métodos de rejeição
 - Procura em tabelas

Método da Transformação (Inversa)

- Para uma v.a. contínua, se a função de distribuição acumulada é $F(x)$ então para uma variável U com distribuição uniforme em $(0,1)$

$$X = F^{-1}(U) \text{ tem por função distrib. acum. } F(x)$$

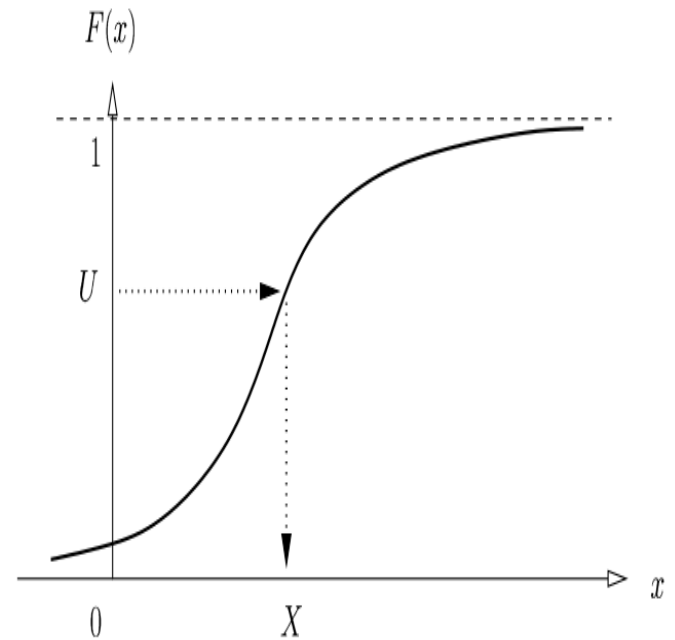
- Este método é apenas eficiente num conjunto pequeno de casos (ex: distribuição exponencial)
- Também não é possível ou é difícil determinar a inversa de muitas distribuições

Demonstração

- $X = F^{-1}(U)$ tem por função de distribuição acumulada $F(x)$??
- Por definição $F(x) = P(X \leq x)$
- $P(X \leq x) = P(F^{-1}(U) \leq x)$
- $= P(U \leq F(x))$
- $= F(x)$ porque $P(U \leq a) = a$

Algoritmo

1. Gerar U com distribuição $U(0,1)$
2. Devolver $X = F^{-1}(U)$



Exemplo de aplicação – Simulação de uma variável aleatória **exponencial**

- Sendo $F(x) = 1 - e^{-x}$ (exponencial de média 1)
- $F^{-1}(u)$ será o valor de x que verifique

$$1 - e^{-x} = u$$

- ou seja $x = -\log(1 - u)$

- Portanto:

$$F^{-1}(u) = -\log(1 - u)$$

É exponencialmente distribuída com média 1

- $1-U$ é também uniforme em $(0,1)$
- Como cX é exponencial com média c para obter uma exponencial de média c basta usar $-c \log(U)$

Exemplo em Matlab

```
function X=exponencial(m,N)  
U=rand(1,N);  
X=-m*log(U)
```

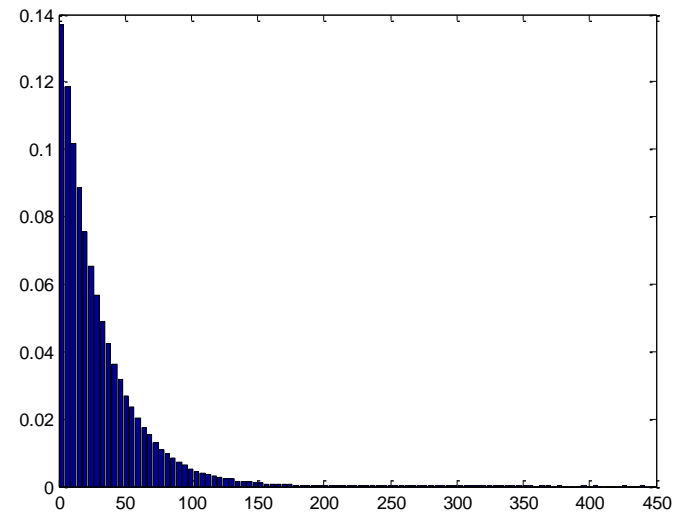
```
%
```

```
N=1e6
```

```
X=exponencial(10,N);
```

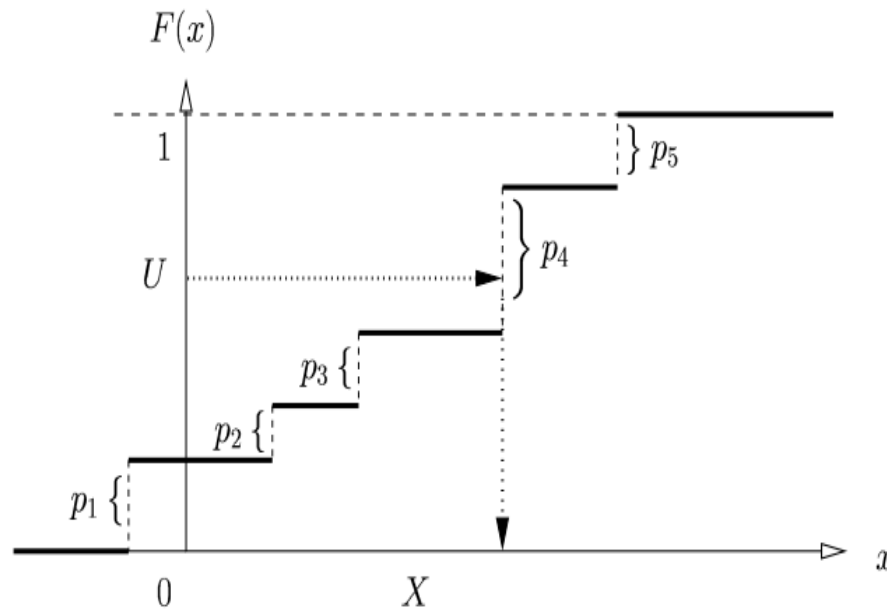
```
[n,xout] = hist(X,100);
```

```
bar(xout,n/N)
```



demo4.m

Algoritmo para caso discreto



1. Gerar U com distribuição $U(0,1)$ exemplo $U=0,7$
2. Ir aumentando x e determinar o primeiro para o qual $F(x) \geq U$
3. Devolver esse valor de x

A procura pode ser tornada mais rápida usando técnicas de procura eficientes

Método de procura numa tabela

- Se a **função cumulativa for guardada numa tabela**, então este algoritmo pode ser visto como uma simples procura numa tabela de

$$i \text{ tal que } F_{i-1} < u \leq F_i$$

- Ou seja:

$$X = \begin{cases} x_1, & \text{if } U < P_1 \\ x_2, & \text{if } P_1 < U < P_1 + P_2 \\ \vdots & \\ x_j, & \text{if } \sum_{i=1}^{j-1} P_i < U < \sum_{i=1}^j P_i \\ \vdots & \end{cases}$$

Exemplo de aplicação

- Gerar pseudo-palavras com as letras assumindo a probabilidade das letras em Português
 - Que já vimos anteriormente

Em Matlab

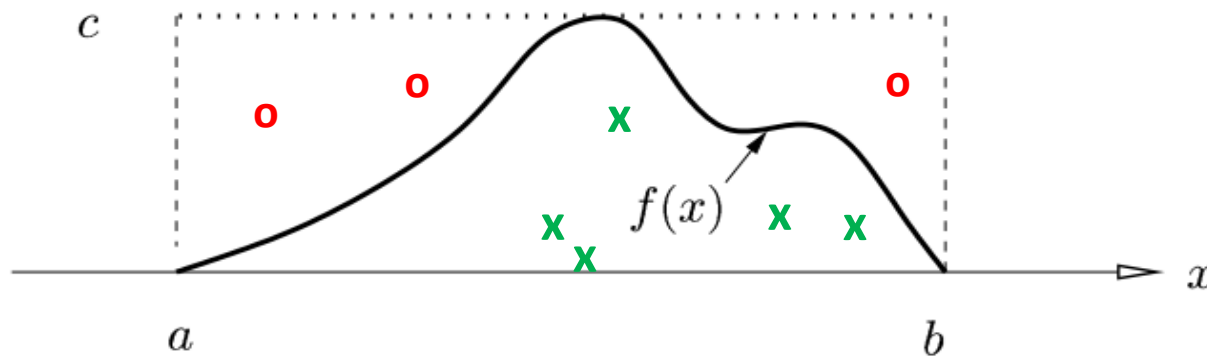
```
letters='abcde';  
% p=[0.0828  0.0084  0.0201  0.0342  0.0792]; % PT real  
p=[0.800  0.01  0.01  0.01  0.17]; % fake  
p=p/sum(p); % só existem para nós 5 letras  
  
X= zeros(1,60);  
for j=1:60  
    U=rand();  
    i = 1 + sum( U > cumsum(p) );  
    % out sera valor entre 1 e 5  
    % de acordo com as probabilidades p  
    X(j)= letters(i);  
end  
char(X)
```



demo5.m

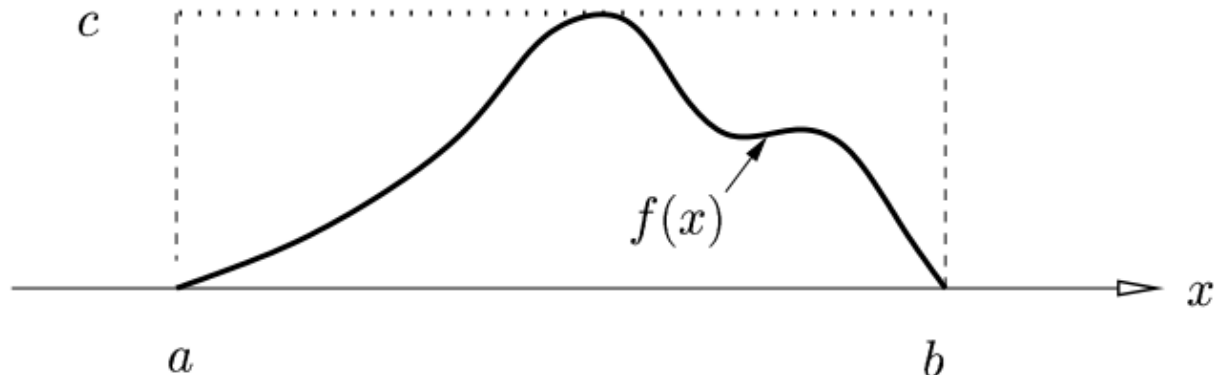
Métodos baseados em Rejeição

- Na sua forma mais simples:
 - define-se uma zona que contém todos os valores da função densidade de probabilidade no intervalo em que está definida
 - Geram-se números com distribuição uniforme nessa zona e rejeitam-se os que ficam acima de $f(x)$



Algoritmo

1. Gerar X com distribuição $U(a, b)$
2. Gerar Y com distribuição $U(0, c)$ independente de X
3. Se $Y \leq f(X)$ devolver $Z = X$; Caso contrário ir para o passo 1

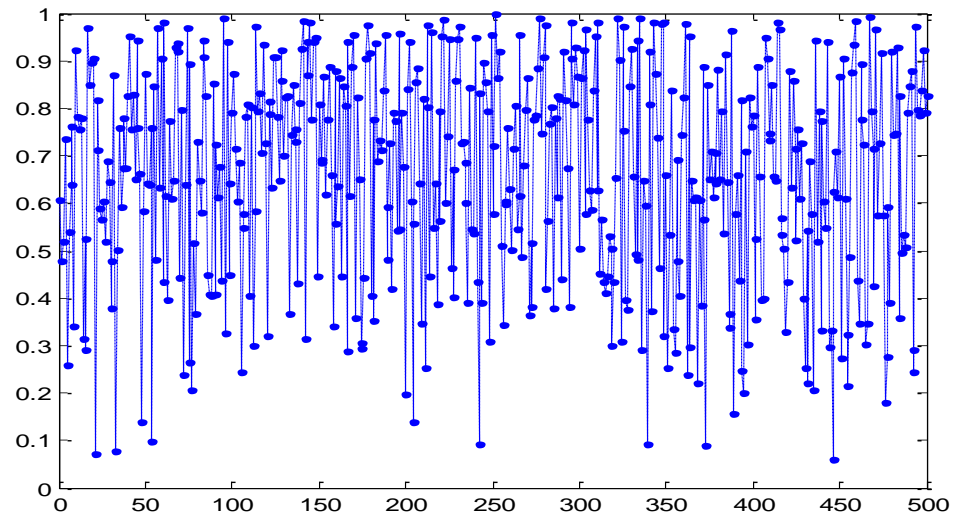
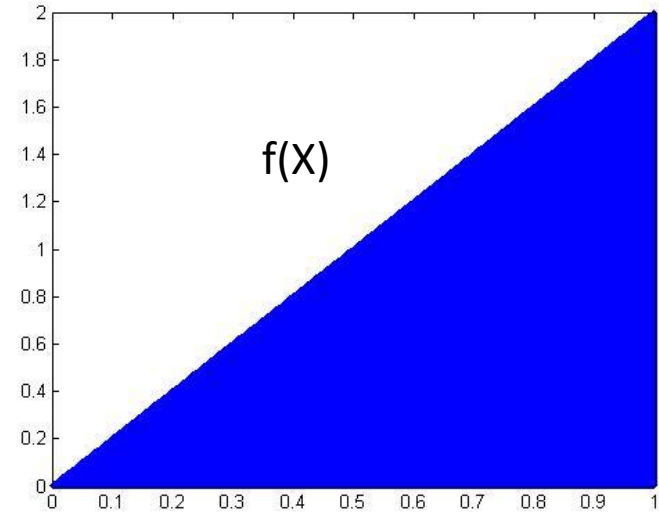


Exemplo

- $f(x) = \begin{cases} 2x & 0 \leq x \leq 1 \\ 0 & \text{outros valores} \end{cases}$

- Temos de usar $c=2$, $a=0$ e $b=1$

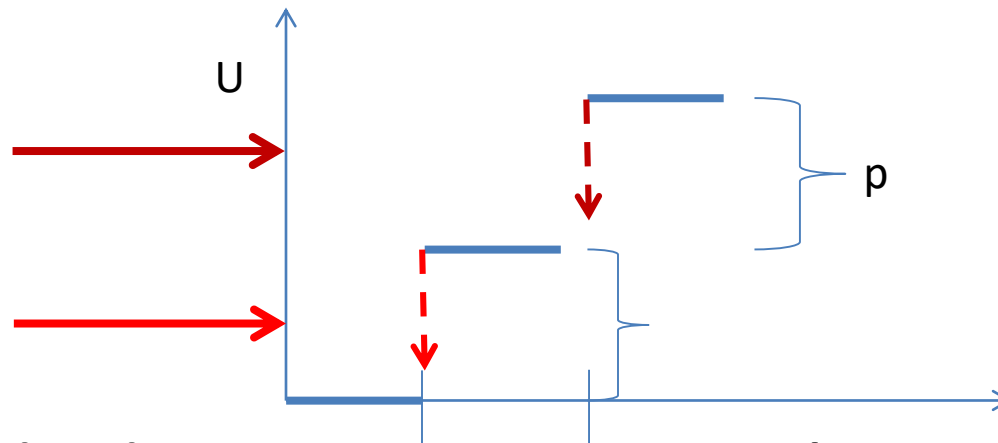
```
%  
N=1e6;  
X=rand(1,N);  
Y=rand(1,N)*2;  
  
Z=X(Y<=2*X);  
  
% grafico  
Y2= Y(Y<=2*X);  
plot(Z,Y2,'.')
```



Algoritmos específicos para distribuição mais comuns (discretas)

Bernoulli

- Aplicando o método da transformação inversa para o caso discreto tem-se



- De onde decorre o seguinte algoritmo:
 - 1 – Gerar U com distribuição $U(0,1)$
 - 2 – Se $U \leq p$ $X=1$; caso contrário $X=0$

Exemplo Matlab

```
function X=Bernoulli (p,N)
```

```
X=rand(1,N)<=p
```

```
% usando
```

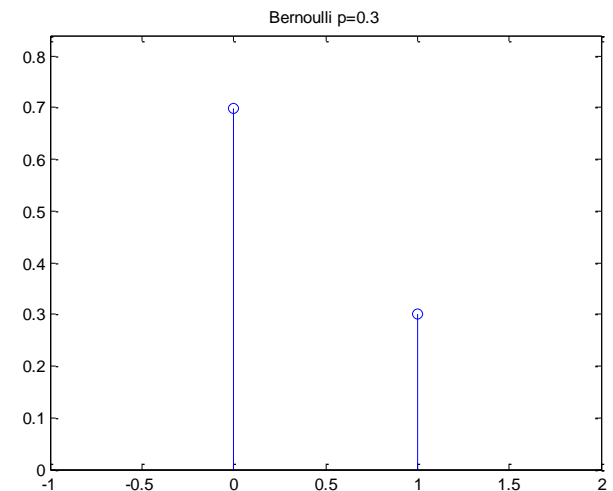
```
N=1e6
```

```
X=Bernoulli(0.3, N);
```

```
myhist(X,'Bernoulli p=0.3')
```

```
p=sum(X==1) /N
```

```
→ 0.2999
```



demo7.m

Técnicas especiais - Obter Binomial

- Pode obter-se uma variável aleatória Binomial usando o facto de que esta pode ser expressa como a soma de n variáveis de Bernoulli independentes
- $X = \sum_{i=1}^n X_i$ é uma v.a. Binomial com parâmetros n e p quando X_i é de Bernoulli com parâmetro p

Obter Binomial - Algoritmo

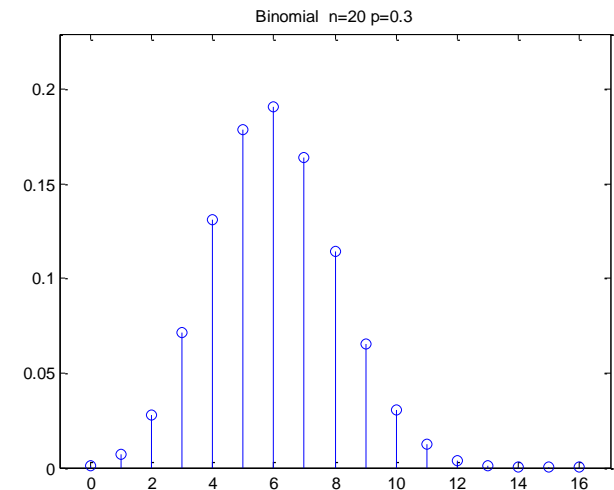
- Gerar variáveis independentes e identicamente distribuídas (iid) X_1, \dots, X_n usando distribuição de Bernoulli com parâmetro p
- Devolver $X = \sum_{i=1}^n X_i$

Demo obtenção binomial

```
function X=binomial(n,p, N)
Bern=rand(n,N)<=p;    % n Bernoulli(p)
X=sum(Bern);
```

% usando

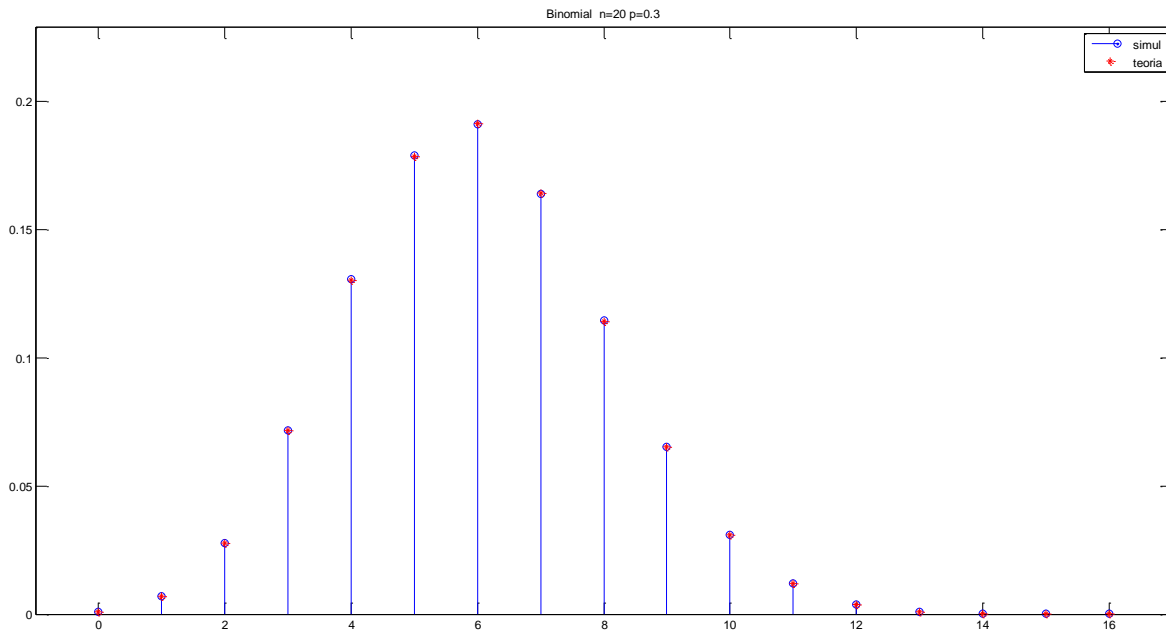
```
N=1e6; n=20; p=0.3;
X=binomial(n,p, N);
myhist(X,'Binomial n=20 p=0.3')
```



demo8.m

Simulação versus teoria

- $N=1e6$



Algoritmos específicos para distribuição mais comuns (contínuas)

Técnicas especiais – Distrib. Normal

- Algoritmo de **Box e Müller**:

1 – Gerar 2 variáveis independentes U_1 e U_2 uniformes em (0,1)

2 – Obter 2 variáveis com distrib. Normal, X e Y , através de:

$$X = (-2 \ln U_1)^{1/2} \cos(2\pi U_2) ,$$

$$Y = (-2 \ln U_1)^{1/2} \sin(2\pi U_2) .$$

Box Müller em Matlab

```
function[X,Y]=BoxMuller(N)
```

```
U1=rand(1,N); % gerar uma v.a. uniforme
```

```
U2=rand(1,N); % gerar outra v.a. uniforme
```

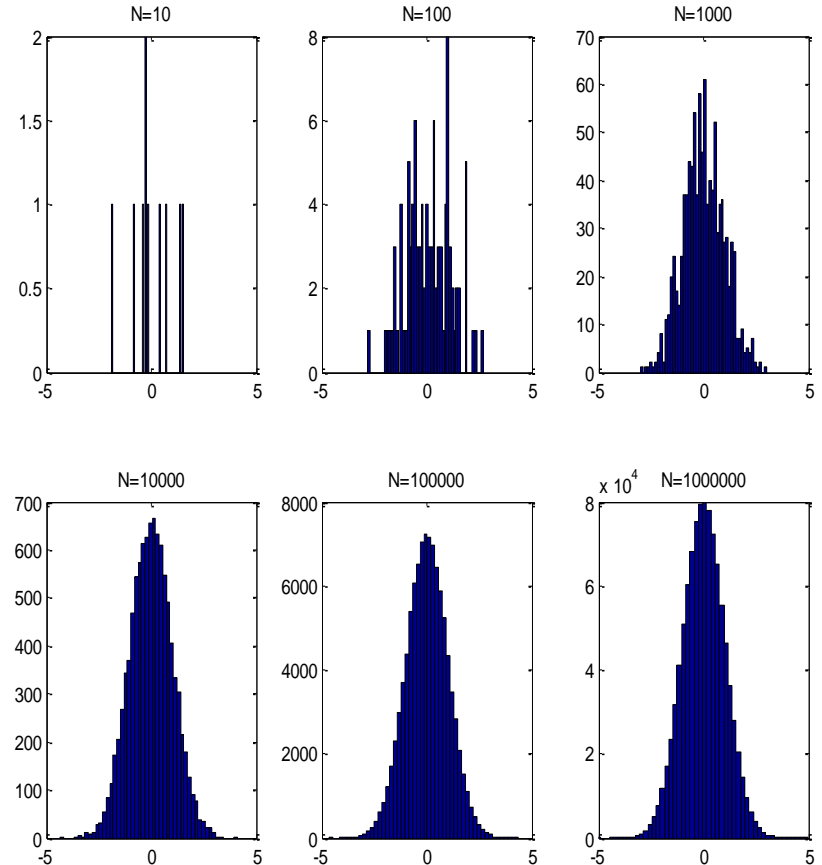
```
X=(-2*log(U1)).^(1/2).* cos(2*pi*U2);
```

```
Y=(-2*log(U1)).^(1/2).* sin(2*pi*U2);
```

- Atenção ao uso de `.^` e `.*`

Demonstração em Matlab

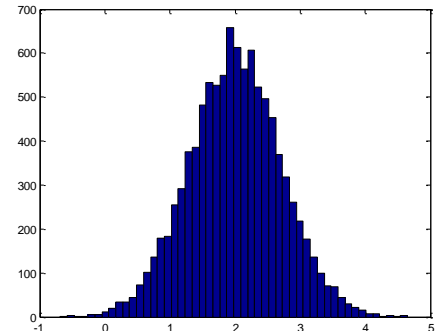
```
for i=1:6
    subplot(2,3,i)
    N=10^i;
    [X,Y]=BoxMuller(N);
    hist(X,50)
    title(['N=' num2str(N)]);
    ax=axis;
    ax(1)=-5; ax(2)=5;
    axis(ax)
end
```



demo9.m

randn() do Matlab

- Em Matlab, e outras linguagens, não necessitamos de implementar esta transformação pois está disponível a função **randn()**
 - **Que gera números aleatórios com uma distribuição Normal de média 0 e variância 1**
- Utilizando as já referidas propriedades $E(X + c) = E(X) + c$ e $Var(cX) = c^2 Var(X)$ podem gerar-se valores de distribuições com média e variância arbitrárias
- Exemplo: média 2 e variância $\frac{1}{2}$
 $Y = \sqrt{1/2} * \text{randn}(1, 1e4) + 2;$
 $\text{hist}(Y, 50)$



Em Java

- É similar a gerar números de uma distribuição uniforme
- O exemplo seguinte mostra como gerar um número aleatório de uma distribuição Gaussiana com média 0 e variância 1

```
import java.util.*;  
Random r = new Random();  
g = r.nextGaussian();
```

- De cada vez que se invoca `r.nextGaussian()` obtém-se um novo número

Para aprender mais

- Online
 - Capítulo “RANDOM NUMBERS, RANDOM VARIABLES AND STOCHASTIC PROCESS GENERATION”
http://moodle.technion.ac.il/pluginfile.php/220739/mod_resource/content/0/slava_fall_2010/Random_number_2_.pdf
- Cap. 1 e Apêndice B do livro “Probabilidades e Processos Estocásticos”, F. Vaz, Universidade de Aveiro