

MPEI 2023-2024

Geração de números aleatórios

Motivação – Exemplo de necessidade

- 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

Geradores

- Para situações como as do exemplo, necessitamos de gerar, ou simular, **vectores de números aleatórios tendo uma determinada distribuição**
- Nos primeiros tempos da simulação utilizavam-se métodos mecânicos
 - Ex: moeda, dado
- Mais tarde utilizaram-se propriedades de dispositivos e elementos
 - Exemplos (atuais): decaimento do Césio-137
- Na área da Informática, foram substituídos por algoritmos que se podem implementar em computador, os **Geradores de números pseudo-aleatórios**
 - 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

Algoritmos congruenciais

- Os métodos mais comuns para gerar sequência pseudo-aleatórias são os chamados *linear congruential generators* - *LCG* (algoritmo congruencial linear)
- Estes geradores geram uma sequência de números através da fórmula

$$X_{i+1} = (aX_i + c) \bmod m$$

- Com X_0 sendo a “semente” (seed) e a , c , m (todos inteiros positivos) designados de multiplicador, incremento e módulo, respetivamente
- Quando $c = 0$ o algoritmo designa-se por *congruencial multiplicativo*

Algoritmos congruenciais

- Como X_i pode apenas assumir os valores $\{0, 1, \dots, m-1\}$, os números

$m \nmid \Rightarrow U_i \neq 1$

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

são designados por **números 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 ciclo de repetição da sequência é no máximo m
- 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
Maior número com 4 bytes = 32 bits

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
```

Outros algoritmos congruenciais

- Uma **generalização** que se pode fazer do algoritmo congruencial multiplicativo **é basear o cálculo do novo valor numa combinação linear das k amostras anteriores**

- Um exemplo deste tipo baseia-se na sequência de **Fibonacci**

$$x_i = x_{i-1} + x_{i-2}, \quad x_1 = 1, x_0 = 0$$

→ combinar mais do que 1 amostra...

- Como a utilização directa não dá bons resultados, usa-se

$$x_i = (x_{i-j} + x_{i-k}) \bmod m$$

→ Os melhores!

- Para $j=31, k=63, m=2^{64}$ temos período de 2^{124}



Outros algoritmos congruenciais

- Outra estratégia: combinar os resultados obtidos com dois geradores congruenciais
 - que, com a escolha conveniente dos parâmetros, vai permitir maiores períodos
 - Conhecida por Combined Multiple Recursive Generator
 - Na implementação em Matlab consiste em:
- $$x_{1,n} = (14033580x_{1,n-2} - 810728x_{1,n-3}) \bmod m_1$$
$$x_{2,n} = (527612x_{2,n-1} - 1370589x_{2,n-3}) \bmod m_2$$
- Sendo a saída

$$z_n \equiv (x_{1,n} - x_{2,n}) \bmod m_1$$

Outros geradores

- **FSR** – Feedback Shift Register // + Rápido
 - Relacionados com os geradores recursivos anteriores
 - A **formula é aplicada a bits**
 - Conjuntos de k bits representam inteiros
 - A formula é realizada **recorrendo a um Shift Register**
 - Vetor de bits que pode ser deslocado para a esquerda um bit de cada vez
 - Feita num computador recorrendo aos registos internos e programação em linguagem máquina
- Mersenne Twister
 - Desenvolvido para resolver problemas de uniformidade do FSR
 - Apresenta um **período extraordinário de $2^{19937} - 1$** //
 - Informação em:
 - <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>

Outros geradores

- Outras classes de algoritmos foram propostas para obter períodos mais longos e melhor aproximação à distribuição uniforme
- A biblioteca NAG, por exemplo, inclui vários:

	Pseudorandom Numbers
Biblioteca	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 . .

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

- Comando `rand()`
 - Gera números uniformemente distribuídos no intervalo (0, 1)
 - usando um algoritmo similar aos anteriormente descritos
- `rand()` utiliza o algoritmo Mersenne twister por defeito
 - ↳ Podemos mudar...
 - Mas permite que se altere, usando `rng()`

rng

- `s=rng`
- `s =`

struct with fields:

Type: 'twister' %% algoritmo por defeito

Seed: 0

State: [625×1 uint32]

rng

- `rng(seed, type)`

Type define o tipo de algoritmo usado e pode ser:

nome	descrição	state
'twister'	Mersenne Twister	625x1 uint32
'combRecursive'	Alg. multiplo recursivo	12x1 uint32
'multFibonacci'	Alg. Fibonacci multiplica- tivo com atraso	130x1 uint64
'v5uniform'	Gerador uniforme do MATLAB® 5.0	35x1 double
'v5normal'	Gerador normal do MAT- LAB 5.0	2x1 double
'v4'	Gerador do MATLAB 4.0	1 uint=seed



Demonstração do uso de rand()

N=1000

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

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

```
axis equal
```

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

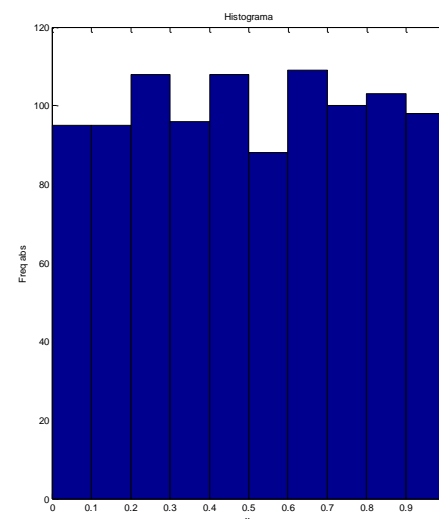
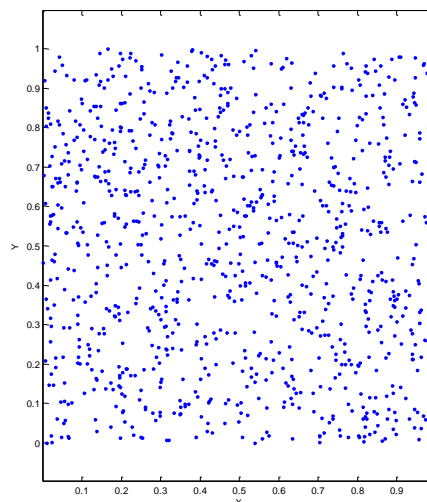
```
ylabel('Y')
```

```
subplot(122), hist(X)
```

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

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

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



Transformações simples

*O randi() também
pode ser usado*

- Aplicando a **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**)
 - Nas versões a partir da 2008b do Matlab existe a **função randi()**

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;    %% ou randi(6,1,n)
```

dado → 5

dado(10) → 3 1 4 5 6 3 4 3 2 4



Métodos para gerar 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 $(0,1)$

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

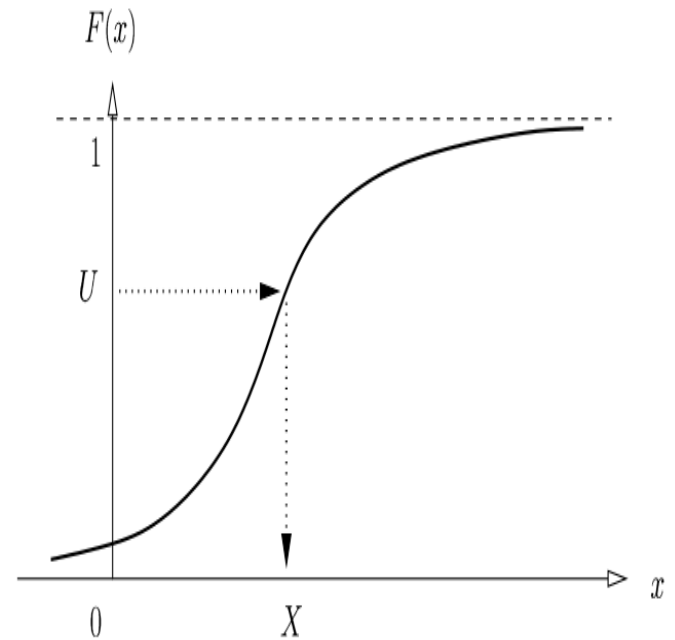
- Este método é apenas eficiente num conjunto pequeno de casos (ex: distribuição exponencial)
- Também não é possível ou é difícil determinar a inversa de muitas distribuições

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

- Como c X 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)
```

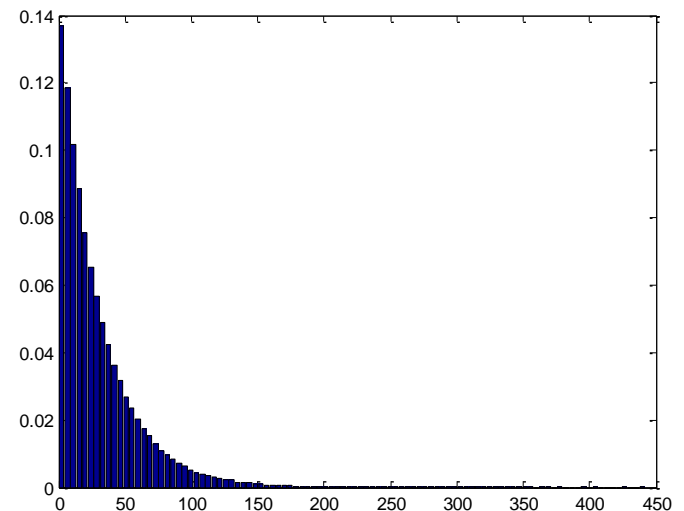
```
%
```

```
N=1e6
```

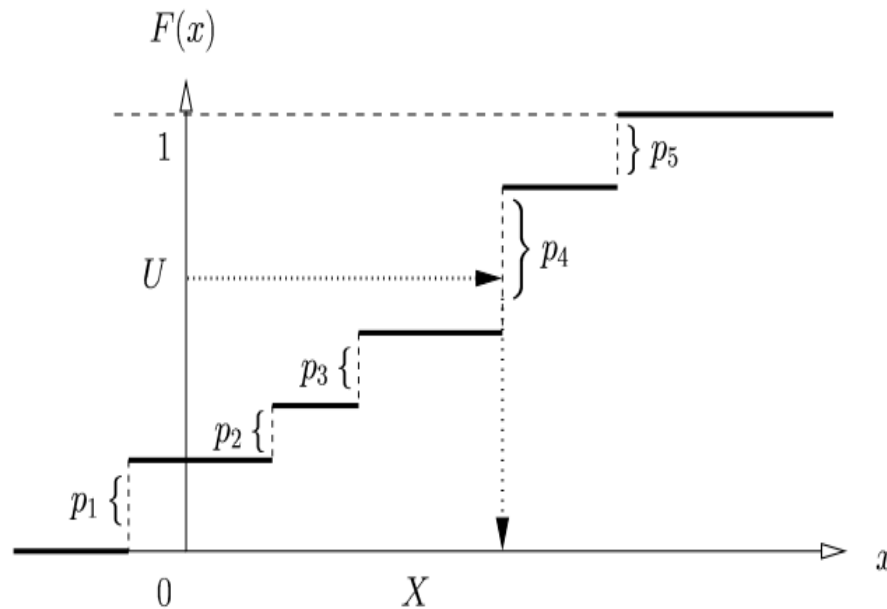
```
X=exponencial(10,N);
```

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

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



Algoritmo para caso discreto



Usado
no
crawler

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

Soma acumulativa

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) );
```

→ soma acumulativa!

```
    % out sera valor entre 1 e 5
```

```
    % de acordo com as probabilidades p
```

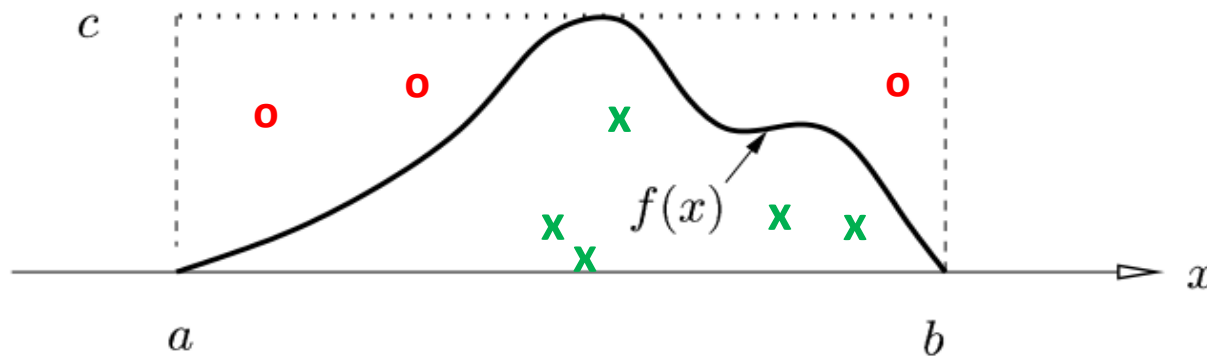
```
    X(j)= letters(i);
```

```
end
```

```
char(X)
```

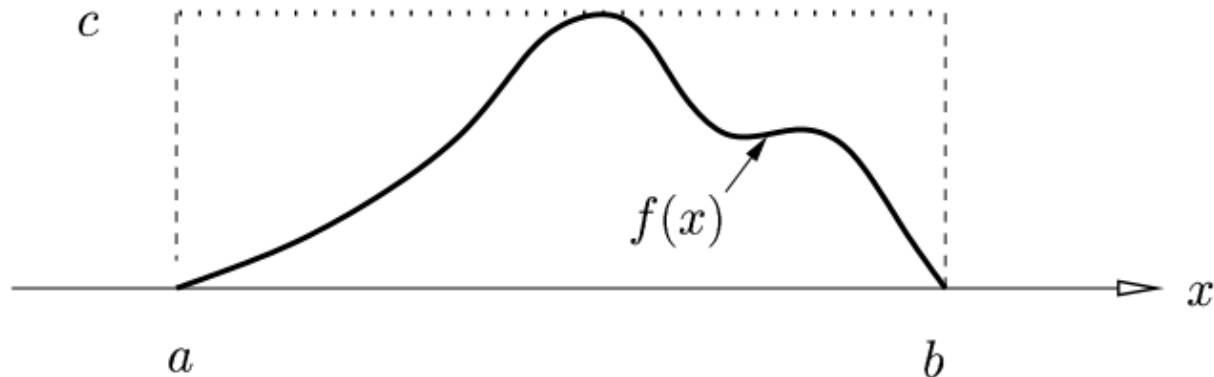

Métodos baseados em Rejeição

- 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 $f(x)$



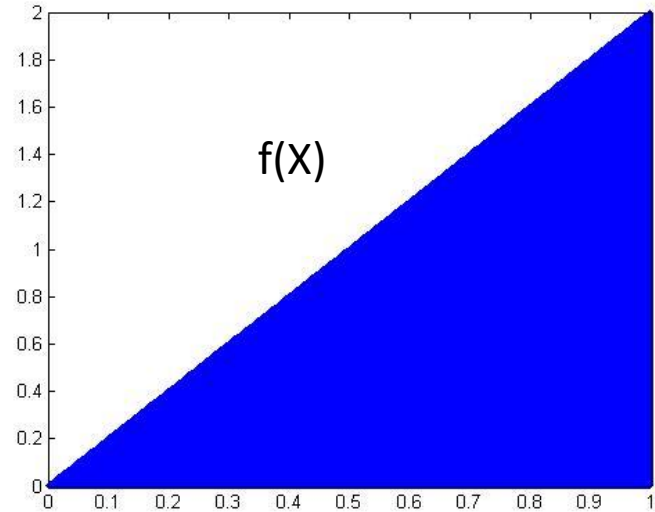
Algoritmo

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

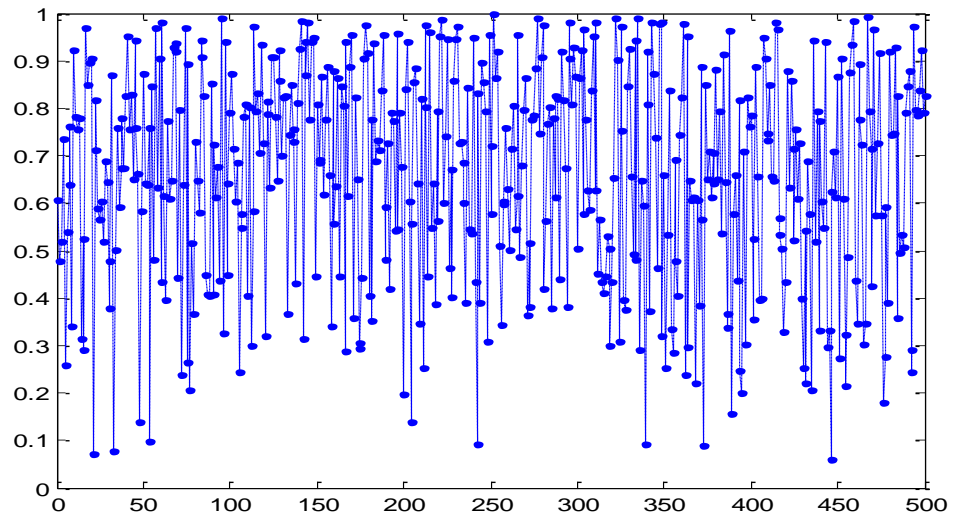
- $f(x) = \begin{cases} 2x & 0 \leq x \leq 1 \\ 0 & \text{outros valores} \end{cases}$
- Temos de usar $c=2$, $a=0$ e $b=1$



```
%  
N=1e6;  
X=rand(1,N);  
Y=rand(1,N)*2;
```

```
Z=X(Y<=2*X);
```

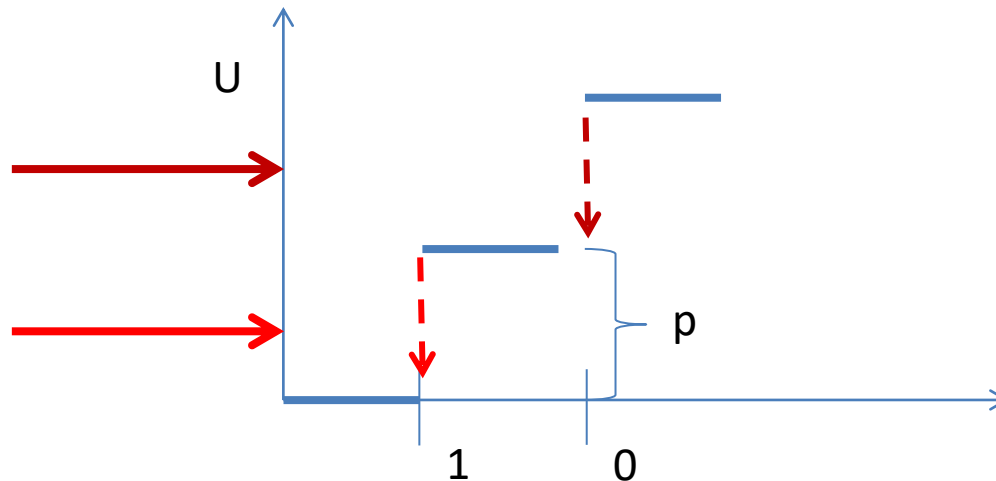
```
% grafico  
Y2= Y(Y<=2*X);  
plot(Z,Y2,'.')
```



Algoritmos específicos para distribuições discretas mais comuns

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 $X=0$

USAM DESDE
A PRIMEIRA
AULA
PRÁTICA

Técnicas especiais - Obter Binomial

- Pode obter-se uma variável aleatória Binomial 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

Algoritmo

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

Exemplo em Matlab - Binomial

```
function X=binomial(n,p, N)
```

```
Bern=rand(n,N)<=p; % n Bernoulli(p)
```

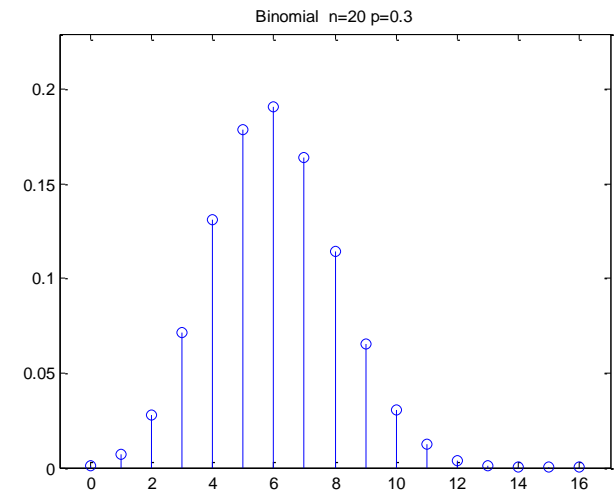
```
X=sum(Bern);
```

% usando

```
N=1e6; n=20; p=0.3;
```

```
X=binomial(n,p, N);
```

```
myhist(X,'Binomial n=20 p=0.3')
```



Algoritmos específicos para distribuição contínua mais comum

Distrib. Normal – Alg. Box Müller

- Algoritmo de **Box e Müller**:

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

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

$$\begin{aligned} X &= (-2 \ln U_1)^{1/2} \cos(2\pi U_2) , \\ Y &= (-2 \ln U_1)^{1/2} \sin(2\pi U_2) . \end{aligned}$$

e^x da fórmula de distribuição normal

→ termos de garantir ortogonalidade

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
```

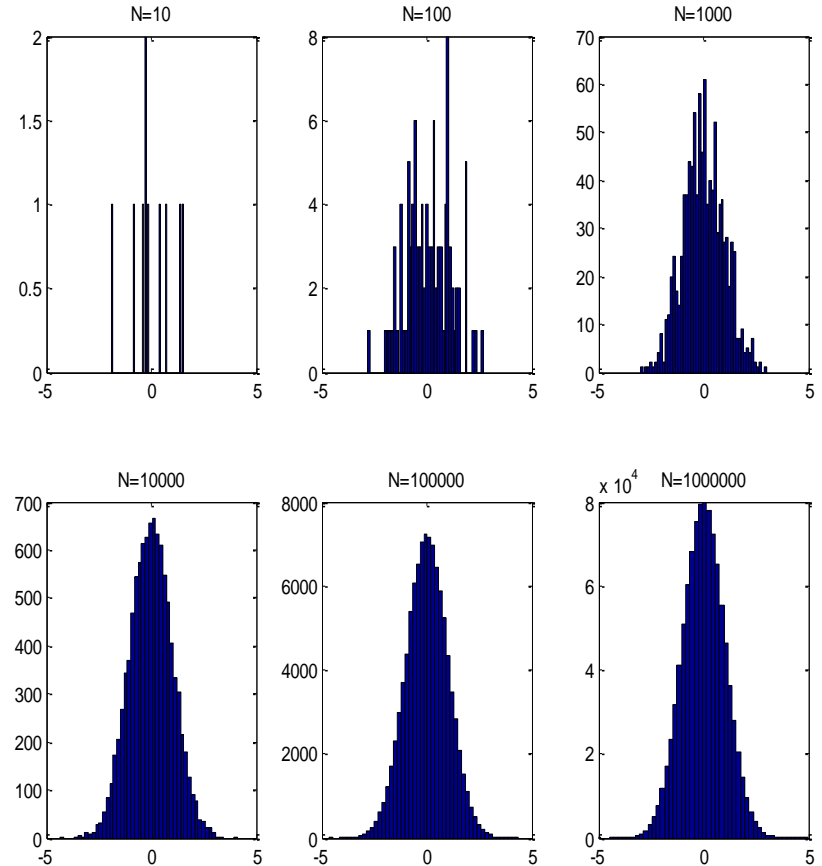
```
X=(-2*log(U1)).^(1/2).* cos(2*pi*U2);
```

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

- Atenção ao uso de .^ e .*

Demonstração em Matlab

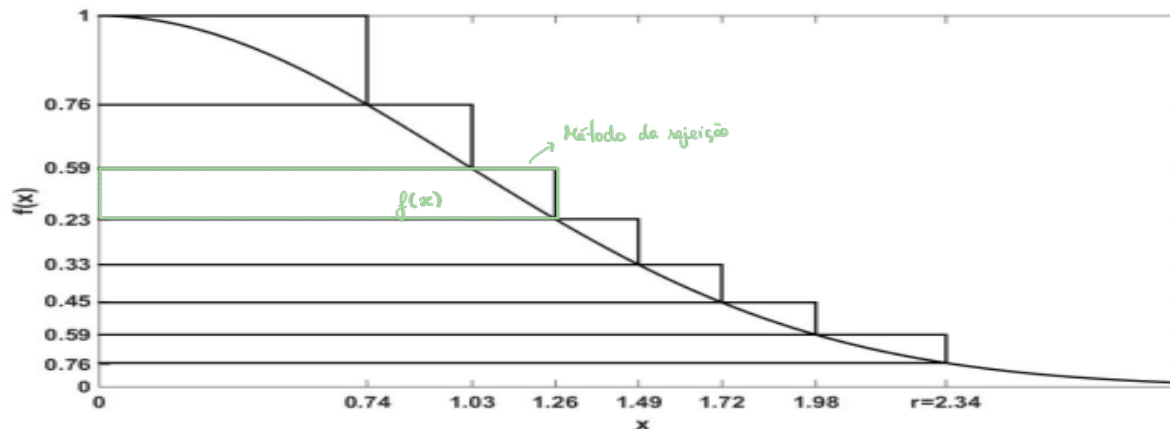
```
for i=1:6
    subplot(2,3,i)
    N=10^i;
    [X,Y]=BoxMuller(N);
    hist(X,50)
    title(['N=' num2str(N)]);
    ax=axis;
    ax(1)=-5; ax(2)=5;
    axis(ax)
end
```



Distribuição Normal – Algoritmo Ziggurat

- Desenvolvido por Marsaglia em 2000
- É um **método de rejeição**
- Utiliza a curva $y = f(x) = e^{-x^2/2}$ para $x > 0$
 - Devido a simetria
- Utiliza um conjunto de tiras com a mesma área e geração de números com distrib. Uniforme

→ Basicamente dividimos em fatias a "pirâmide"



– Figura faz lembrar um Ziggurate (antiga Mesopotâmia)

Para aprender mais

- Cap. 9 do livro “Métodos Probabilísticos para Engenharia Informática”,
 - F. Vaz e A Teixeira, Edições Sílabo
- [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