# Algoritmos congruenciais

Estes são os metodos mais communs onde os geradores geram uma sequencia de numeros recursivamente

$$X_i = (aX_i + c) \bmod \mathrm{m}$$

Sendo X0,a,c e m a semente de geração designados de multiplicador, incremento e módulo, respetivamente

Quando $c$ = 0 o algoritmo designa-se por congruencial multiplicativo

Xi pode estar no intervalo [0,m] pelo que que aos numeros resultantes de

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

chamamos de elementos pseudo aleatórios que correspondem a uma sequencia aleatória uniformemente distribuida

## Processo de calculo

1. Escolher os valores de a, c e m
2. Escolher a semente $X0$ (tal que 1<= $X0$ <=m)
3. Calcular o próximo número aleatório usando a expressão $X1$ = $aX0$ + $c\ mod\ m$
4. Substituir $X0$ por $X1$ e voltar ao ponto anterior

## Como escolher os parâmetros ?

A recursão dos numeros é ciclica pelo que de **m** em **m** os numeros gerados começam a ser iguais

Ex:

$$a = c = X_0 = 3 \text{ e } m = 5 \text{ gera a sequência } \{3, 2, 4, 0, 3...\}$$

Pelo que apenas algumas combinações produzem resultados significativos

Uma implementação em matlab seria do tipo

```
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
end
```

## Outros algoritmos congruenciais

Outras estratégias baseiam-se na combinação de dois ou mais geradores congruenciais

**Ex:**

$$x_i = (xi - 31 + x_{i-63}) \bmod \mathrm{m}$$

Neste caso temos um período de 2^124 elmentos

## FSR - Feedback Shift Register

Esta forma está relaciona com os geradores recursivos que vimos anteriormente

A forma recursiva é aplicada a bits deslocando para a direita ou para a esquerda um bit de cada vez

Este gerador recorre a um shift register programando parcialmente ou totalmente o gerador em linguagem máquina

**Ex:**

- [Mersenne Twister]
    - Desenvolvido para resolver problemas de uniformidade do FSR
    - Apresenta um período extraordinário de 219937 − 1

## Wichman-Hill I

Este algoritmo usa uma combinação de quatro geradores congruenciais

$$\begin{aligned} w_i &= a_1 * w_{i-1}\ mod\ m_1 \\ x_i &= a_2 * x_{i-1}\ mod\ m_2 \end{aligned}$$

$$y_i = a_3 * x_{i-1} \; mod \; m_3$$
$$z_i = a_4 * z_{i-1} \; mod \; m_4$$

$$U_i = (\frac{w_i}{m_1} + \frac{x_i}{m_2} + \frac{y_i}{m_3} + \frac{z_i}{m_4}) \; mod \; 1$$

# Na prática ...

A maioria das linguagens de computador disponibilizam n geradores pseudo-aleatórios

### rand()

Gera números numa distribuição uniforme de 0 a 1, este utiliza os algoritmo de **Mersene twister** no entanto este pode ser alterado através da função **rng()**

## 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 multplicativo com atraso | `130x1 uint64` |
| `'v5uniform'` | Gerador uniforme do MATLAB® 5.0 | `35x1 double` |
| `'v5normal'` | Gerador normal do MATLAB 5.0 | `2x1 double` |
| `'v4'` | Gerador do MATLAB 4.0 | `1 uint=seed` |

# Transformações

### Transformação linear

**Y = a*U*+*b*** é uma forma simples de obter uma distribuição uniforme no intervalo de [b,a+b]

# Métodos geométricos para gerar variáveis aleatórias de distribuições não uniformes

Existem 3 tipos de métodos:

- Métodos de transformação
- Métodos de rejeição
- Procura em tabelas

### Método da transformação inversa

Dada uma função de distribuição acumulada **F(x)** podemos obter variáveis aleatórias dessa distribuição fazendo a inversa da função para uma variável aleatória **U** de distribuição aleatória

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

Este método é apenas num conjunto de casos como a **distribuição exponencial**, além disso em muitas distribuições não é possível calcular o seu inverso

#### Exemplo

#### Simulação de uma variável aleatória exponencial

Sendo

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

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



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

# Método de procura numa tabela

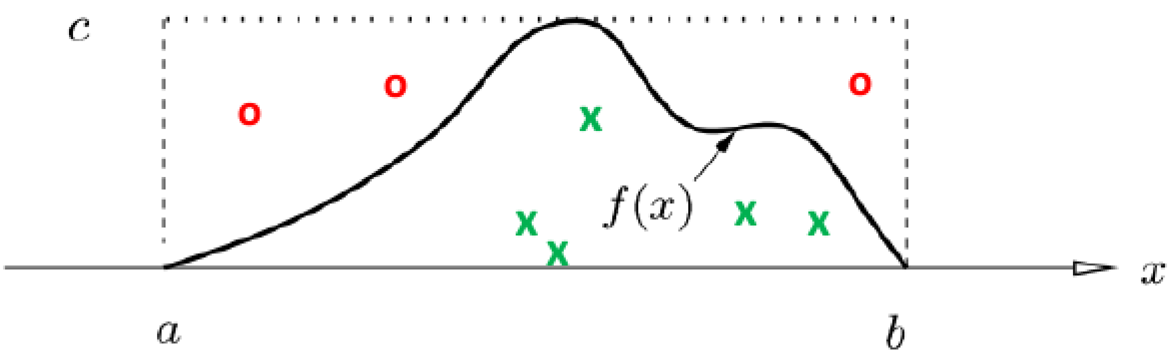
Se a função cumulativa for guardada numa tabela, então este algoritmo pode ser visto como uma simples procura numa tabela de

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

```
p=[0.800 0.01 0.01 0.01 0.17];
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
```

## Algoritmos baseados na rejeição

Neste algoritmo basicamente defenimos uma zona que contém todos os valores da **função densidade de probabilidade** no intervalo em que está definida, geramos números com uma distribuição uniforme nessa zona e rejeitam-se todos que fiquem acima de **f(X)**




1. Gerar X com distribuição U(a, b)

2. Gerar Y com distribuição U(0, c) independente de X

3. Se Y ≤ f X devolver Z = X; Caso contrário ir para o passo 1

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

# Algoritmos para distribuições comuns

## Bernolli

Gerar distribuição U(0,1)

Se U <= p, X = 1 caso contrário X = 0

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


% usando
N=1e6
X=Bernoulli(0.3,N);
```

## Binomial

Pode obter-se uma variável aleatória Binomial **através da soma de n variáveis de de Bernolli**

$$X = \sum_{i=1}^{n} X_i$$

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


N=1e6; n=20; p=0.3;
 0.05
X=binomial(n,p, N);
myhist(X,‘Binomial n=20 p=0.3')
```

## Normal

**Box Müller**

1. Gerar duas variáveis independentes U1 e U2
2. Obter 2 variáveis com distribuição Normal X e Y através de

$$X = (-2lnU_1)^{\frac{1}{2}} cos(2\pi U_2)$$
$$Y = (-2lnU_1)^{\frac{1}{2}} sin(2\pi U_2)$$

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

No entanto em matlab temos uma função nativa para a geração de numeros segundo uma distribuição normal **randn**