PUC-RIO
INF1036 - Probabilidade Computacional

Material 2 - Geração de Números Pseudo-aleatórios

Professora - Ana Carolina Letichevsky*

2022.1

*Material Adaptado de Professor Hélio Lopes

Geração de números pseudo-aleatórios

Geradores de números aleatórios:

- Linear Congruential Generator (LCG)
- Mersene Twister

Linear Congruential Generator (LCG)

- Gera uma sequência de números inteiros através da seguinte fórmula de recorrência:

$$x_k = (a \times x_{k-1} + c) \, mod \, M$$

onde M, a e c são inteiros dados.

- A condição inicial x_0 é chamada semente do gerador.
- O inteiro M é aproximadamente o maior inteiro representável na máquina.
- A qualidade de tal gerador depende da escolha de a e c, e em qualquer caso o período é menor do que M.

Linear Congruential Generator (LCG)

Mostre que a sequência de inteiros gerada pelo método LCG, usando $x_0 = 1, a = 6, c = 0$ e $M = 11$ é:

i	0	1	2	3	4	5	6	7	8	9	10	11
x_i	1	6	3	7	9	10	5	8	4	2	1	6

```
x0 = 1
a = 6
c = 0
M = 11

def LCG(seed, a, c, M):
    x = seed
    u = []
    u.append(x)
    for i in range(11):
        nx = (a * x + c) % M
        u.append(nx)
        x  = nx
    return u

U = LCG(x0, a, c, M)
print(len(U))
print(U)
```

```
12
[1, 6, 3, 7, 9, 10, 5, 8, 4, 2, 1, 6]
```

Linear Congruential Generator (LCG)

Mostre que a sequência de inteiros gerada pelo método LCG, usando $x_0 = 1$, $a = 6$, $c = 0$ e $M = 11$ é:

i	0	1	2	3	4	5	6	7	8	9	10	11
x_i	1	6	3	7	9	10	5	8	4	2	1	6

Repita o exercício usando o mesmo valor para c e M, e alterando a para 3. Considere $x_0 = 1$.

```
x0 = 1
a = 3
c = 0
M = 11

def LCG(seed, a, c, M):
    x = seed
    u = []
    u.append(x)
    for i in range(11):
        nx = (a * x + c) % M
        u.append(nx)
        x  = nx
    return u

U = LCG(x0, a, c, M)
print(len(U))
print(U)
```

```
12
[1, 3, 9, 5, 4, 1, 3, 9, 5, 4, 1, 3]
```

Linear Congruential Generator (LCG)

Mostre que a sequência de inteiros gerada pelo método LCG, usando $x_0 = 1, a = 6, c = 0$ e $M = 11$ é:

i	0	1	2	3	4	5	6	7	8	9	10	11
x_i	1	6	3	7	9	10	5	8	4	2	1	6

Repita o exercício usando o mesmo valor para c e M, e alterando a para 3. Considere $x_0 = 2$.

```
x0 = 2
a = 3
c = 0
M = 11

def LCG(seed, a, c, M):
    x = seed
    u = []
    u.append(x)
    for i in range(11):
        nx = (a * x + c) % M
        u.append(nx)
        x  = nx
    return u

U = LCG(x0, a, c, M)
print(len(U))
print(U)
```

```
12
[2, 6, 7, 10, 8, 2, 6, 7, 10, 8, 2, 6]
```

Linear Congruential Generator (LCG)

M	a	Referência
$2^{31} - 1 = 2147483647$	16807	Park & Miller
2147483647	39373	L'Ecuyer
2147483647	742938285	Fishman & Moore
2147483647	950706376	Fishman & Moore
2147483647	1226874159	Fishman & Moore
2147483647	40692	L'Ecuyer
2147483647	40014	L'Ecuyer

Para todos os casos acima, $c = 0$.

Linear Congruential Generator (LCG)

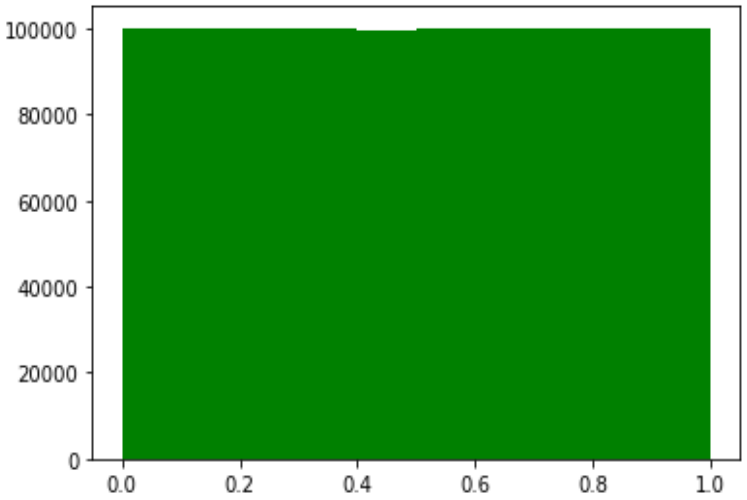
Em simulação estocástica, nos interessa ter uma sequência de números pseudo-aleatórios distribuídos de forma uniforme entre 0 e 1.

Assim, cria-se uma outra sequência a partir da sequência gerada pelo LCG, por exemplo:

$$x_k = (a \times x_{k-1} + c) \, mod \, M$$

$$u_k = x_k / M$$

```
#Referência L'Ecuyer, M = 2^31 - 1
import matplotlib.pyplot as plt

a = 39373
c = 0
M = 2147483647

def LCG(seed, a, c, M, nsamples):
    x = seed
    u = []
    for i in range(nsamples):
        nx = (a * x + c) % M
        u.append(float(nx) / float(M))
        x  = nx
    return u

U = LCG(4, a, c, M, 1000000)
print(len(U))
plt.hist(U, facecolor = 'green')
plt.show()
```



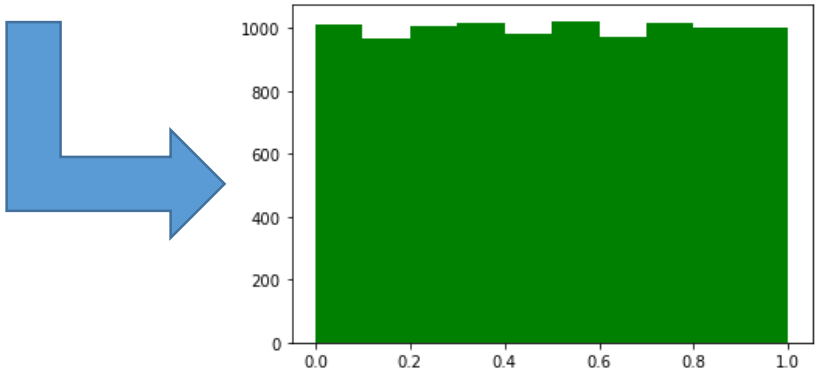
Explorando o LCG em Python

Exemplo 1) Lançamento de moeda (cara ou coroa).

```
a = 39373
c = 0
M = 2147483647

def LCG(seed, a, c, M, nsamples):
    x = seed
    u = []
    for i in range(nsamples):
        nx = (a * x + c) % M
        u.append(float(nx) / float(M))
        x  = nx
    return u

U = LCG(3, a, c, M, 10000)
```

p = 0.5
A sequência é dividida em cara e coroa

```
def CARA_OU_COROA(U, p):
    n = len(U)
    CC = []
    for i in range(n):
        if (U[i] < (1.0 - p)):
            CC.append(0) #cara
        else:
            CC.append(1) #coroa
    return CC

CC = CARA_OU_COROA(U, 0.5)
print(sum(CC))
```




```
5016
```

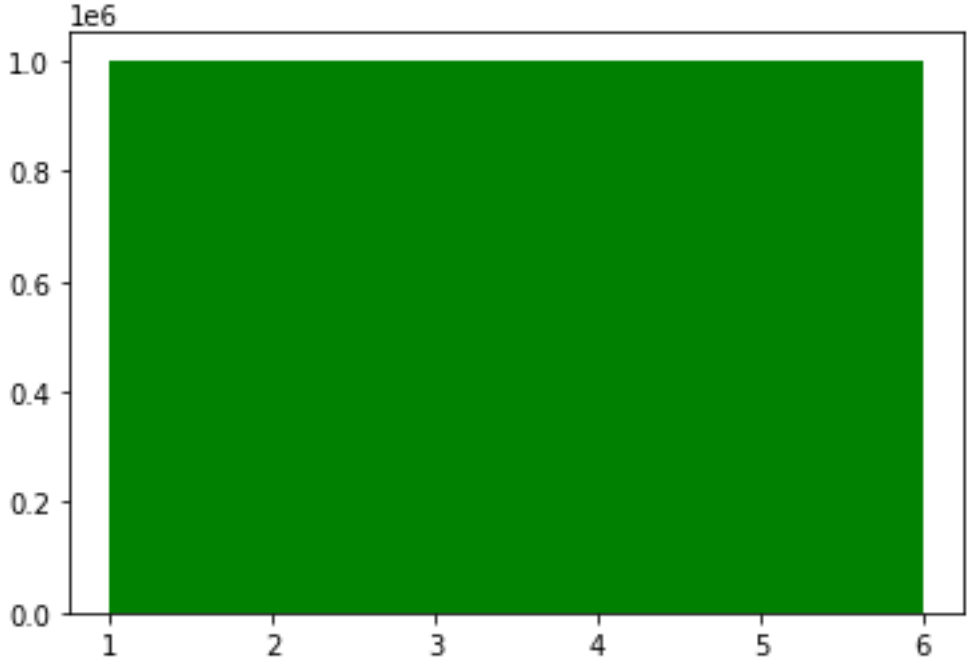
10.000 lançamentos, resultando em 5016 coroas e 4984 caras.

Explorando o LCG em Python

Exemplo 2) Lançamento de dado.

```
def DADO(U):
    n = len(U)
    dado = []
    for i in range(n):
        if   (U[i] < 1.0 / 6.0):
            dado.append(1)
        elif (U[i] < 2.0 / 6.0):
            dado.append(2)
        elif (U[i] < 3.0 / 6.0):
            dado.append(3)
        elif (U[i] < 4.0 / 6.0):
            dado.append(4)
        elif (U[i] < 5.0 / 6.0):
            dado.append(5)
        else:
            dado.append(6)
    return dado

U = LCG(3, a, c, M, 6000000)
dado = DADO(U)
plt.hist(dado, 6, facecolor = 'green')
plt.show()
```

Um conjunto para cada face do dado.



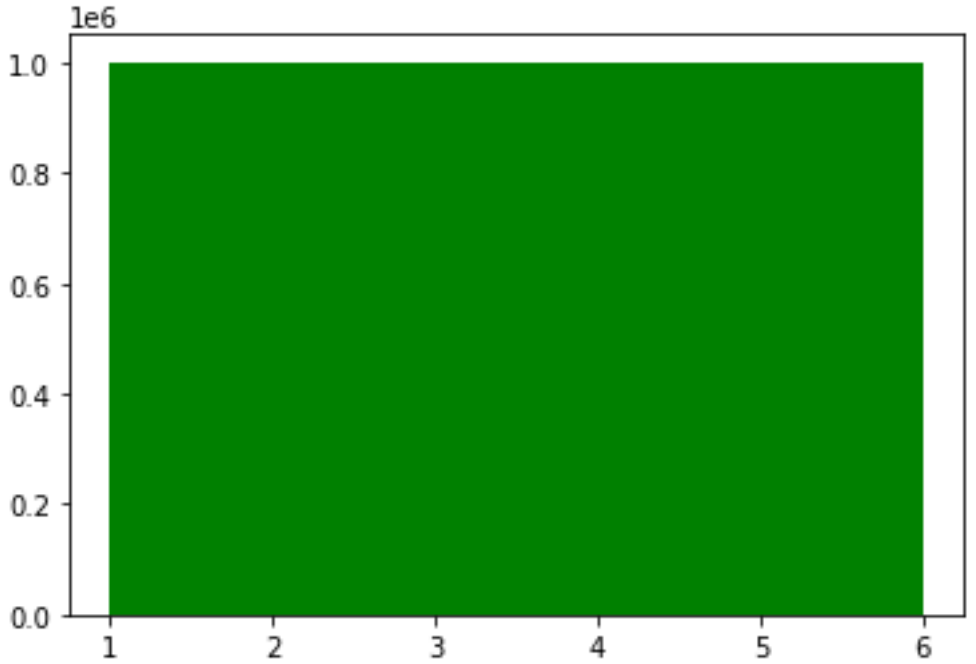
Explorando o LCG em Python

Exemplo 3) Lançamento de dado modificado.

```
def DADO(U):
    n = len(U)
    dado = []
    for i in range(n):
        dado.append(int(U[i] * 6.0) + 1)
    return dado

U = LCG(3, a, c, M, 6000)
dado = DADO(U)
print(dado)

plt.hist(dado,6,facecolor='green')
plt.show()
```

A sequência apresenta resultados de 1 a 6.

```
[1, 1, 1, 6, 2, 5, 6, 5, 6, 3, 5, 6, 2, 2, 3, 4, 4, 4,
3, 4, 4, 5, 2, 5, 5, 6, 6, 5, 2, 6, 2, 3, 6, 5, 5, 5,
4, 4, 2, 4, 3, 3, 1, 4, 4, 4, 4, 3, 1, 3, 4, 5, 6, 1,
3, 5, 4, 2, 2, 4, 2, 5, 4, 6, 6, 2, 2, 5, 3, 2, 2, 5,
5, 3, 3, 2, 3, 1, 2, 4, 4, 5, 3, 4, 4, 5, 5, 1, 6, 6,
1, 1, 2, 6, 6, 2, 6, 1, 6, 4, 3, 2, 6, 2, 5, 2, 5, 6,
```



Mersenne Twister

- Gera uma sequência de números pseudo-aleatórios com base em uma recursão linear.
- Fornece uma geração rápida com alta qualidade de aleatoriedade.
- Tem período de $2^{19937} - 1$.
- Está presente em várias linguagens inclusive Python e R.

```
import numpy as np
u = np.random.sample(10000) # Retorna 10.000 valores no intervalo semiaberto [0.0, 1.0).
print(u)
```

```
[0.84323793 0.69972057 0.39446578 ... 0.52126064 0.85923756 0.37630521]
```

```
v = runif(10000)   # Retorna 10.000 valores no intervalo [0.0, 1.0).
print(v)
```

```
2.979446e-02 1.167465e-01 1.951053e-01 4.742841e-01
```

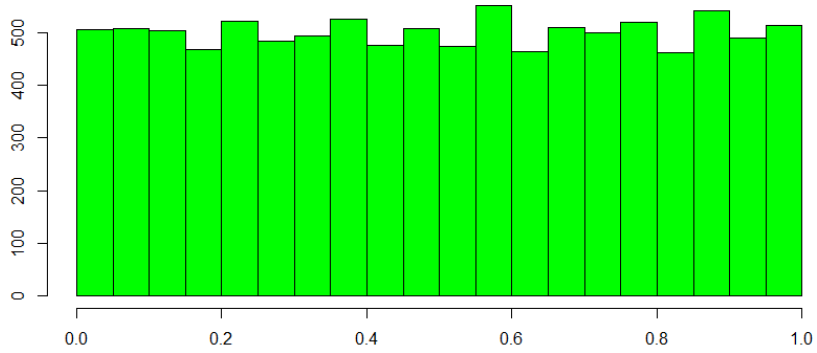
Explorando o LCG em R

Exemplo 1) Lançamento de moeda (cara ou coroa).

```
a = 39373
c = 0
M = 2147483647

LCG <- function (seed, a, c, M, nsamples) {
  x = seed
  u = NULL
  for (i in 1:nsamples) {
    nx = (a * x + c) %% M
    u = c(u, as.double(nx) / as.double(M))
    x = nx
  }
  return (u)
}
```

A sequência é dividida em cara e coroa.

```
CARA_OU_COROA <- function (U, p) {
  n = length(U)
  CC = NULL
  for (i in 1:n)
    if (U[i] < (1.0 - p))
      CC = c(CC, 0) #cara
    else
      CC = c(CC, 1) #coroa
  return (CC)
}

U = LCG(3, a, c, M, 10000)
hist(U, col = 'green')
CC = CARA_OU_COROA(U, 0.5)
print(sum(CC))
```



10.000 lançamentos, resultando em 5016 para cara e 4984 para coroa.

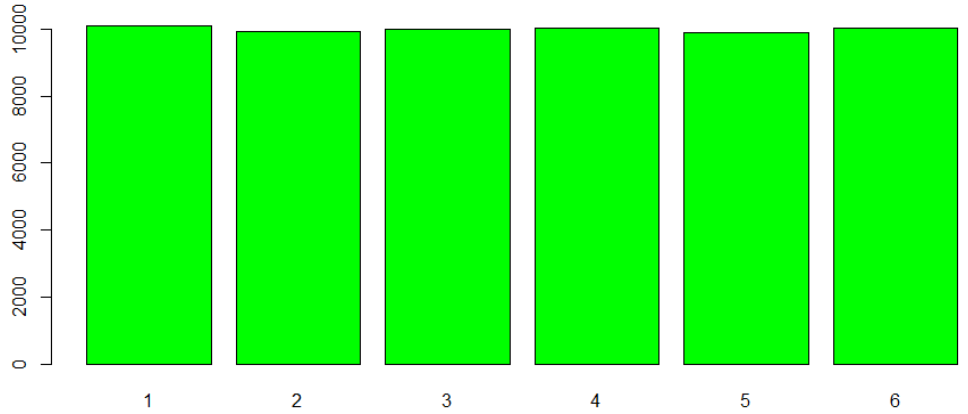
5016

Explorando o LCG em R

Exemplo 2) Lançamento de dado.

```
DADO <- function (U) {
  n = length(U)
  dado = NULL
  for (i in 1:n) {
    if (U[i] < 1.0 / 6.0)
      dado = c(dado, 1)
    else if (U[i] < 2.0 / 6.0)
      dado = c(dado, 2)
    else if (U[i] < 3.0 / 6.0)
      dado = c(dado, 3)
    else if (U[i] < 4.0 / 6.0)
      dado = c(dado, 4)
    else if (U[i] < 5.0 / 6.0)
      dado = c(dado, 5)
    else
      dado = c(dado, 6)
  }
  return (dado)
}

U = LCG(3, a, c, M, 60000)
dado = DADO(U)
barplot(table(dado), col = 'green')
```

Um conjunto para cada face do dado.




```
    1     2     3     4     5     6
10097  9929  9999 10035  9898 10042
```

Explorando o LCG em R

Exemplo 3) Lançamento de dado modificado.

```
DADO <- function (U) {
  n = length(U)
  dado = NULL
  for (i in 1:n)
    dado = c(dado, as.integer(U[i] * 6.0) + 1)
  return (dado)
}

U = LCG(3, a, c, M, 60000)

dado = DADO(U)
print(dado)

barplot(table(dado), col = 'green')
```

A sequência apresenta resultados de 1 a 6.

```
 [1] 1 1 1 6 2 5 6 5 6 3 5 6 2 2 3 4 4 4 4 3 1 4 6 5 2 6 1 4 5 6 5 5
[33] 4 5 3 6 2 1 1 1 2 1 4 3 4 4 5 2 5 5 6 6 5 2 6 2 3 6 5 5 5 4 6 5
[65] 4 4 3 2 1 4 2 3 2 6 6 3 1 1 5 1 6 3 3 1 1 2 6 4 4 2 4 3 3 1 4 4
[97] 4 4 3 1 3 4 5 6 1 5 5 5 4 1 6 6 4 6 5 2 3 4 5 2 6 1 6 4 3 1 2 5
```

```
    1     2     3     4     5     6
10097  9929  9999 10035  9898 10042
```