

# Geradores de números pseudo-aleatórios: Inversive congruential, Lagged Fibonacci e Linear congruential

Lucas João Martins

## 1 Códigos das implementações

icg.py

```
1 # -*- coding: utf-8 -*-
2 """Implementacao do 'Inversive congruential generator'"""
3
4
5 class icg:
6     """Classe que constroi o gerador de numeros
7     pseudo-aleatorios
8
9     Atributos:
10         seed: semente do gerador
11         q: numero que e feito o modulo da formula
12         a: numero multiplicador da formula
13         c: numero que faz a adicao da formula, ou, pode ser
14             utilizado como um
15             resultado
16
17     """
18
19     def __init__(self, seed, q, a, c):
20         """Construtor com todos os atributos obrigatorios
21
22         Args:
23             ja especificado na classe
24
25         """
26         self.seed = seed
27         self.q = q
28         self.a = a
29         self.c = c
30
31     def generate(self, n):
32         """Gerador de numeros pseudo-aleatorios
33
34         Gera uma lista com n-1 numeros pseudo-aleatorios
35         conforme o algoritmo
36         icg ou seja, utiliza da seguinte relacao:
```

```

32         x_0 = seed
33         x_{i+1} = (a * x^{-1}_i + c) mod q if x_i != 0
34         x_{i+1} = c if x_i == 0
35         Onde  $x^{-1}_i$  e a funcao modular inversa
36
37     Args:
38         n: valor representa quantidade de numeros que
           serao gerados - 1
39
40     Returns:
41         lista com os n + 1 numeros gerados
42     """
43     result = [self.seed]
44     for i in range(n):
45         number = 0
46         if result[i] == 0:
47             number = self.c
48         else:
49             x = (self.a *
50                 self.modular_inverse(result[i]) + self.c)
51             number = x % self.q
52
53         result.append(number)
54
55     return result
56
57 def modular_inverse(self, x):
58     """Funcao que calcula a modular inversa de um numero
59
60     Forma 'naive' de se calcular a modular inversa de um
61     numero x mod q,
62     onde q e informado na construcao do gerador.
63     Considerada 'naive' por nao
64     usar do algoritmo de euclides estendido. Funciona da
65     seguinte maneira:
66     Calcula  $x * i \bmod q$  para todo i entre 0 e q - 1,
67     o i que gerar o
68     resultado 1 sera a modular inversa
69
70     Args:
71         x: numero do qual se quer saber a modular
72             inversa com q
73
74     Returns:
75         a modular inversa de um numero x mod q
76
77     Raises:
78         Exception: quando nao existe modular inversa
79     """

```

```

74         for i in range(self.q):
75             result = (x * i) % self.q
76             if result == 1:
77                 return i
78
79         raise Exception("Nao foi possivel calcular a
80                         'inversa modular'")
81
82 if __name__ == '__main__':
83     # nao usar sys.argv porque demora para gerar numeros com
84     # mais de 30 bits
85     psng = icg(109951162775, 100110021, 10, 109951162775)
86     print(psng.generate(5))

```

lfg.py

```

1  # -*- coding: utf-8 -*-
2  """Implementacao do 'Lagged Fibonacci generator'"""
3
4  import random
5  import sys
6
7
8  class lfg:
9      """Classe que constroi o gerador de numeros
10         pseudo-aleatorios
11
12         Atributos:
13             lags: tupla de dois valores que irao representar os
14                 'lags' da formula
15             exponent: expoente que ira na potencia de 2 que e
16                 feita de modulo da
17                 formula
18         """
19
20     def __init__(self, lags, exponent):
21         """Construtor com todos os atributos obrigatorios
22
23         Args:
24             seed: lista vazia que ira conter a semente do
25                 gerador
26             random: valor utilizado na montagem da semente
27                 alem dos ja especificados na classe
28         """
29         self.j = lags[0]
30         self.k = lags[1]
31         self.m = 2**exponent
32         self.seed = []
33         self.random = 10000

```

```

31     def make_seed(self):
32         """Monta a lista de numeros que sera utilizada como
33             semente no gerador
34
35         Para o funcionamento correto do algoritmo e
36             necessario que k valores
37             aleatorios sejam utilizados como semente. Nessa
38             implementacao isso e
39             feito da seguinte forma:
40             Utiliza-se o modulo random do python para gerar
41             um numero entre 0 e
42             1, para assim multiplicar esse valor por 10000
43             (um valor arbitrario) e por fim pegar a parte
44             inteira desse
45             resultado e adicionar na lista de sementes
46         """
47         i = 0
48         while i < self.k:
49             self.seed.append((int(random.random() *
50                 self.random)))
51             i += 1
52
53     def valid_amount(self, n):
54         """Verifica a validade dos argumentos utilizados na
55             geracao dos numeros
56
57         Argumentos serao validos quando:
58             j < k
59             n - j < k
60             n - k < k
61
62         Args:
63             n: quantidade de numeros que serao gerados
64
65         Returns:
66             True se os argumentos sao validos, senao False
67         """
68         return True if n - self.j < self.k and self.j <
69             self.k else False
70
71     def alfg_generate(self, n):
72         """Gerador de numeros pseudo-aleatorios com a
73             operacao de adicao
74
75         Gera uma lista com n numeros pseudo-aleatorios
76             conforme o algoritmo
77             alfg (additive lagged fibonacci generator), ou seja,
78             ha outras versoes
79             que ao inves da adicao para montar o p do algoritmo

```

```

        utilizam da:
        subtracao
        multiplicacao
        xor
O algoritmo utiliza da seguinte relacao:
    Para cada numero que sera gerado a lista de
    sementes e iterada:
        Quando e o primeiro elemento da lista de
        sementes:
            Calcula-se o pseudo-aleatorio p:
                p = (seed_{n - j} + seed_{n - k})
                  mod m
                Onde seed e a lista de sementes
        Quando e o ultimo elemento da lista de
        sementes:
            A posicao recebe o pseudo-aleatorio
            calculado
        Nos outros casos:
            O elemento a direita da lista vem para a
            posicao atual

Args:
    n: quantidade de numeros que serao gerados

Returns:
    lista com os n numeros gerados

Raises:
    Exception: quando os argumentos sao considerados
    invalidos
"""
if not self.valid_amount(n):
    raise Exception("Valores invalidos!")

result = []
self.make_seed()
i = 0
while i < n:
    for j in range(self.k):
        if j == 0:
            x = self.seed[n - self.j] + self.seed[n
                - self.k]
            number = x % self.m
        elif 0 < j < self.k - 1:
            self.seed[j] = self.seed[j + 1]
        else:
            self.seed[j] = number
            result.append(number)
    i += 1

```

```

109         return result
110
111
112 if __name__ == '__main__':
113     psng = lfg((70, 100), int(sys.argv[2]))
114     psng.random = int(sys.argv[1])
115     print(psng.alfg_generate(5))

```

#### lcg.py

```

1  # -*- coding: utf-8 -*-
2  """Implementacao do 'Linear congruential generator'"""
3
4  import sys
5
6
7  class lcg:
8      """Classe que constroi o gerador de numeros
9          pseudo-aleatorios
10
11      Atributos:
12          multiplier: numero multiplicador da formula
13          increment: numero que faz a adicao da formula
14          seed: semente do gerador
15          modulus: numero que e feito o modulo da formula
16      """
17
18      def __init__(self, multiplier, increment, seed, modulus):
19          """Construtor com todos os atributos obrigatorios
20
21          So constroi o gerador se os atributos sao
22          considerados validos para a
23          geracao dos numeros
24
25          Args:
26              ja especificado na classe
27
28          Raises:
29              Exception: quando os argumentos sao considerados
30              invalidos
31      """
32
33      if modulus > 0 and 0 < multiplier < modulus and \
34          0 <= increment < modulus and 0 <= seed < modulus:
35          self.a = multiplier
36          self.c = increment
37          self.r0 = seed
38          self.m = modulus
39      else:
40          raise Exception("Argumentos invalidos!")

```

```

38     def generate(self, n):
39         """Gerador de numeros pseudo-aleatorios
40
41         Gera uma lista com n numeros pseudo-aleatorios
42         conforme o algoritmo lcg,
43         ou seja, utiliza da seguinte relacao:
44             X_{n+1} = (a * X_n + c) mod m
45             Quando n = 0, entao X_n = semente do gerador
46
47         Args:
48             n: quantidade de numeros que serao gerados
49
50         Returns:
51             lista com os n numeros gerados
52         """
53         result = []
54         for i in range(n):
55             if i == 0:
56                 previous = self.r0
57             else:
58                 previous = result[i-1]
59
60             result.append((self.a * previous + self.c) %
61                           self.m)
62
63         return result
64
65 if __name__ == '__main__':
66     psng = lcg(1664525, 1013904223, int(sys.argv[1]),
67               2**int(sys.argv[2]))
68     print(psng.generate(5))

```

## 2 Explicação dos algoritmos

O inversive congruential generator, também conhecido como ICG, foi desenvolvido por Eichenauer e Lehn em 1986 e trata-se de um gerador de números pseudo-aleatórios que é baseado em recursão não linear. A sua congruência é definida por:

$$x_{i+1} \equiv ax_i^{-1} + c \pmod{q} \quad (1)$$

Onde  $x_i^{-1}$  é a função modular inversa entre  $x$  e  $q$ . Os outros componentes dessa relação já foram explicados na documentação do código. Importante citar que para um correto funcionamento da relação  $x$  e  $q$  precisam ser coprimos, e, que essa verificação não é realizada no código implementado. O máximo período que o ICG pode alcançar é de  $q$  unidades.

O lagged fibonacci generator, também conhecido como LFG, é um gerador de números pseudo-aleatórios que é baseado em uma generalização da sequência de

Fibonacci. A sua congruência é definida por:

$$p_n \equiv p_{n-j} \star p_{n-k} \pmod{m} \quad (2)$$

Onde  $\star$  denota uma operação binária qualquer, ou seja, pode ser adição, subtração, multiplicação ou um xor. Os outros atributos dessa relação já foram explicados na documentação do código. A implementação do gerador nesse trabalho faz uso do additive lagged fibonacci generator, também chamado de ALFG, ou seja, a operação binária escolhida foi a adição. O máximo período que o ALFG pode alcançar é de  $(2^k - 1)2^{p-1}$  unidades.

O linear congruential generator, também conhecido como LCG, foi desenvolvido por D. H. Lehmer em 1948 e trata-se de um popular gerador de números pseudo-aleatórios que é baseado em uma equação linear. A sua relação de recorrência é definida por:

$$X_{n+1} = (aX_n + c) \pmod{m} \quad (3)$$

Os atributos apresentados já foram explicados na documentação do código. O máximo período que o LCG pode alcançar é de  $m$  unidades.

Todos os três geradores possuem regras associadas as suas variáveis e também as suas formas de operações que podem ser classificadas, de maneira informal, em dois tipos:

1. Essenciais para o funcionamento do gerador;
2. Responsáveis por maximizar o desempenho do gerador.

Optou-se por não implementar regras do segundo tipo, mas mais informações sobre elas podem ser obtidas nas referências citadas. Já com relação às regras do primeiro tipo, experimentou-se utilizar três estratégias diferentes:

- Não permitir a instanciação do gerador, o que foi feito no LCG;
- Não fazer nada em nível de código e esperar que quem o utilize saiba das regras, aplicado no ICG;
- Permitir a instanciação do gerador, mas não permitir a geração dos números, utilizado no LFG.

### 3 Comparação entre os algoritmos

Ao comparar o LCG com o ICG, percebe-se o seguinte:

- O ICG é a versão não linear do LCG;
- O LCG apresenta diversas regularidades não desejadas devido a sua linearidade;
- O resultado do LCG é reticulado, enquanto que o do ICG não é,

Já ao comparar o LFG contra o LCG e o ICG, pode-se destacar:



Tabela 1: 5 números pseudo-aleatórios grandes

Gerador	Tamanho do número em bits	Tempo gasto
ICG	27	1m15,687s
ICG	mais que 30	Muito tempo, impraticável
LCG	40	0m0,023s
LFG	40	0m0,037s
LCG	56	0m0,027s
LFG	56	0m0,043s
LCG	80	0m0,030s
LFG	80	0m0,035s
LCG	128	0m0,023s
LFG	128	0m0,033s
LCG	168	0m0,027s
LFG	168	0m0,040s
LCG	512	0m0,027s
LFG	512	0m0,043s
LCG	2048	0m0,023s
LFG	2048	Overflow
LCG	4096	0m0,023s

- O LFG requer que  $k$  unidades sejam armazenadas para poder gerar um único número, enquanto que os outros dois só necessitam do armazenamento de uma unidade
- O período máximo do LCG e do ICG é limitado ao valor do seu módulo, enquanto que o LFG não é, e, por isso pode atingir um valor bem maior do que o dos outros dois já citados

## 4 Complexidade dos algoritmos

- Inverse congruential generator:  $O(n * q)$ 
  - Devido aos laços nas linhas 44 e 74 do código.
- Lagged Fibonacci generator:  $O(n * k)$ 
  - Devido aos laços nas linhas 94 e 95 do código.
- Linear congruential generator:  $O(n)$ 
  - Devido ao laço na linha 51 do código.

## 5 Os números são aleatórios?

Uma resposta curta: você não consegue afirmar se algo é aleatório ou não, conforme Dilbert já falou um dia.



Uma resposta longa: há diferentes maneiras de interpretar o significado da aleatoriedade, pois isso pode ser feito do ponto de vista estatístico, ou do ponto de vista da complexidade de Kolmogorov, ou até mesmo do ponto de vista criptográfico. Entretanto, nenhum desses pontos de vista conseguem afirmar se algo é aleatório ou não.

Uma possível solução: se o ponto de vista estatístico for o escolhido, então há diversos testes que podem mensurar a qualidade de números aleatórios. Testes de Diehard, desenvolvido em 1995 por George Marsaglia, e os testes de Charmaine Kenny, desenvolvido em 2005, são alguns exemplos de conhecidas baterias de testes que realizam essa verificação de qualidade.

## 6 Referências

Inversive congruential generator:

- [Wikipedia](#)
- [Khan Academy](#)
- [The Mathematical-Function Computation Handbook](#), Nelson H.F. Beebe
- [pLab](#)
- [Construction of inversive congruential pseudorandom number generators with maximal period length](#), Jürgen Eichenauer-Herrmann, 2002
- [Parallel inversive congruential generators: Software and field-programmable gate array implementations](#), Michael Mascagni e Shahram Rahimi, 2001

Lagged Fibonacci generator:

- [Lagged Fibonacci Random Number Generators for Distributed Memory Parallel Computers](#), Srinivas Aluru, 1997
- [Aaron Toponce](#)
- [Wikipedia](#)

- [Bernie Pope](#)
- [Oak Ridge National Laboratory](#)

Linear congruential generator:

- [Wikipedia](#)
- [Eternally Confuzzled](#)
- [Rosetta Code](#)

Aleatorieadade:

- [Jeremy Kun](#)
- [Wikipedia](#)
- [Stack overflow](#)