

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
           inversa com q
72
73     Returns:
74         a modular inversa de um numero x mod q
75
76     Raises:
77         Exception: quando nao existe modular inversa
78     """

```

```

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     psng = icg(1, 5, 2, 3)
84     print(psng.generate(30))

```

lfg.py

```

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

```

```

31         aleatorios sejam utilizados como semente. Nessa
           implementacao isso e
32         feito da seguinte forma:
33             Utiliza-se o modulo random do python para gerar
                 um numero entre 0 e
34             1, para assim multiplicar esse valor por 10000
35             (um valor arbitrario) e por fim pegar a parte
                 inteira desse
36             resultado e adicionar na lista de sementes
37         """
38         i = 0
39         while i < self.k:
40             self.seed.append((int(random.random() * 10000)))
41             i += 1
42
43     def valid_amount(self, n):
44         """Verifica a validade dos argumentos utilizados na
           geracao dos numeros
45
46         Argumentos serao validos quando:
47             j < k
48             n - j < k
49             n - k < k
50
51         Args:
52             n: quantidade de numeros que serao gerados
53
54         Returns:
55             True se os argumentos sao validos, senao False
56         """
57         return True if n - self.j < self.k and self.j <
           self.k else False
58
59     def alfg_generate(self, n):
60         """Gerador de numeros pseudo-aleatorios com a
           operacao de adicao
61
62         Gera uma lista com n numeros pseudo-aleatorios
           conforme o algoritmo
63         alfg (additive lagged fibonacci generator), ou seja,
           ha outras versoes
64         que ao inves da adicao para montar o p do algoritmo
           utilizam da:
65             subtracao
66             multiplicacao
67             xor
68         O algoritmo utiliza da seguinte relacao:
69         Para cada numero que sera gerado a lista de
           sementes e iterada:

```

```

70         Quando e o primeiro elemento da lista de
           sementes:
71         Calcula-se o pseudo-aleatorio p:
72         p = (seed_{n - j} + seed_{n - k})
           mod m
73         Onde seed e a lista de sementes
74         Quando e o ultimo elemento da lista de
           sementes:
75         A posicao recebe o pseudo-aleatorio
           calculado
76         Nos outros casos:
77         O elemento a direita da lista vem para a
           posicao atual
78
79     Args:
80         n: quantidade de numeros que serao gerados
81
82     Returns:
83         lista com os n numeros gerados
84
85     Raises:
86         Exception: quando os argumentos sao considerados
           invalidos
87
88     """
89     if not self.valid_amount(n):
90         raise Exception("Valores invalidos!")
91
92     result = []
93     self.make_seed()
94     i = 0
95     while i < n:
96         for j in range(self.k):
97             if j == 0:
98                 x = self.seed[n - self.j] + self.seed[n
99                     - self.k]
100                 number = x % self.m
101             elif 0 < j < self.k - 1:
102                 self.seed[j] = self.seed[j + 1]
103             else:
104                 self.seed[j] = number
105                 result.append(number)
106         i += 1
107
108     return result
109
110 if __name__ == '__main__':
111     psng = lfg((70, 100), 4)
112     print(psng.alfg_generate(150))

```

```

1  # -*- coding: utf-8 -*-
2  """Implementacao do 'Linear congruential generator'"""
3
4
5  class lcg:
6      """Classe que constroi o gerador de numeros
7          pseudo-aleatorios
8
9      Atributos:
10         multiplier: numero multiplicador da formula
11         increment: numero que faz a adicao da formula
12         seed: semente do gerador
13         modulus: numero que e feito o modulo da formula
14
15     def __init__(self, multiplier, increment, seed, modulus):
16         """Construtor com todos os atributos obrigatorios
17
18         So constroi o gerador se os atributos sao
19         considerados validos para a
20         geracao dos numeros
21
22         Args:
23             ja especificado na classe
24
25         Raises:
26             Exception: quando os argumentos sao considerados
27             invalidos
28
29         """
30         if modulus > 0 and 0 < multiplier < modulus and \
31             0 <= increment < modulus and 0 <= seed < modulus:
32             self.a = multiplier
33             self.c = increment
34             self.r0 = seed
35             self.m = modulus
36         else:
37             raise Exception("Argumentos invalidos!")
38
39     def generate(self, n):
40         """Gerador de numeros pseudo-aleatorios
41
42         Gera uma lista com n numeros pseudo-aleatorios
43         conforme o algoritmo lcg,
44         ou seja, utiliza da seguinte relacao:
45              $X_{n+1} = (a * X_n + c) \bmod m$ 
46             Quando  $n = 0$ , entao  $X_n$  = semente do gerador
47
48         Args:

```

```

45         n: quantidade de numeros que serao gerados
46
47     Returns:
48         lista com os n numeros gerados
49     """
50     result = []
51     for i in range(n):
52         if i == 0:
53             previous = self.r0
54         else:
55             previous = result[i-1]
56
57         result.append((self.a * previous + self.c) %
58                     self.m)
59
60     return result
61
62 if __name__ == '__main__':
63     psng = lcg(1664525, 1013904223, 10, 2**32)
64     print(psng.generate(1000))

```

2 Explicação dos algoritmos

3 Tabelas

4 Comparação entre os algoritmos

5 Complexidade dos algoritmos

- Inversive congruential generator: $O(n * q)$
- Lagged Fibonacci generator: $O(n * k)$
- Linear congruential generator: $O(n)$

6 Os números são aleatórios?

7 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](#)