CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

João Pedro Pereira de Assis Castro Santos
Juliana Silva Cruz Sartori

PRNG
(PSEUDORANDOM NUMBER GENERATORS)

Timóteo

2022

1 Introdução

A geração de números aleatórios tem muitos usos (em sua maioria em estatística, para amostragem aleatória e simulação). Antes da computação moderna, pesquisadores que precisavam de números aleatórios os gerava através de vários meios (dado, cartas, roleta, etc.), ou utilizavam as tabelas de números aleatórios existentes. A primeira tentativa de prover para os pesquisadores um suprimento pronto de dígitos aleatórios foi feita em 1927, quando a Cambridge University Press publicou uma tabela de 41.600 dígitos desenvolvida por Leonard H.C. Tippet. Em 1947, a RAND Corporation gerou números por meio de uma simulação eletrônica de uma roleta; os resultados foram eventualmente publicados em 1955 como A Million Random Digits with 100,000 Normal Deviates (Um milhão de dígitos aleatórios com 100.000 desvios normais).(Wikipedia, 2021a)

John von Neumann foi um pioneiro dos geradores de números aleatórios baseados em computadores. Um contribuidor notável no campo da geração de números pseudoa leatórios para uso prático, é um matemático paquistanês Dr. Arif Zaman. Em 1951, Derrick Henry Lehmer inventou o gerador linear congruente, utilizado na maioria dos geradores de números pseudoa leatórios atuais. Com a disseminação do uso dos computadores, geradores de números pseudoa leatórios substituíram as tabelas numéricas, e "verdadeiros"geradores aleatórios (hardwares geradores de números pseudoa leatórios) são utilizados apenas em alguns casos.(Wikipedia, 2021a)

Uma variável pseudo aleatória é uma variável que é criada por um procedimento determinístico (frequentemente um programa de computador ou uma subrotina) que (geralmente) recebe bits aleatórios como entrada. A cadeia pseudoaleatória irá, tipicamente, ser maior do que a cadeia aleatória original, porém menos aleatória (menor entropia, no sentido aplicado na teoria da informação). Isto pode ser útil para algoritmos aleatórios. Geradores de números pseudoa leatórios são amplamente utilizados em aplicações como modelagem computacional (e.g., Cadeias de Markov), estatística, design experimental, etc. Alguns deles são suficientemente aleatórios para serem úteis nestas aplicações; muitos não são, e uma sofisticação considerável é necessária para determinar corretamente a diferença para qualquer propósito em particular. O uso não-precavido de geradores de números pseudoa leatórios prontamente disponíveis tem causado danos consideráveis, e por muito tempo sustentados, no valor de um grande número de projetos de pesquisas por muitos anos.(Wikipedia, 2021a)

2 Desenvolvimento

O código XorShift é baseado em mudanças de posição nos bits ou bit shifts. O algoritmo implementado usa a operação de shift para fazer três movimentações nos bits do valor binário da semente, em seguida soma o resultado de tais shifts a um incremento, também advindo da semente somada ao valor Long 123456789L, e faz sua divisão pelo valor máx(número máximo de bits do valor “randomizado”) pela operação %(módulo), a fim de entregar o resto da divisão como valor final. Caso o valor seja negativo, ele é multiplicado por -1 antes de ser retornado.
Abaixo segue o código implementado:

```
package rngtests;

/**
 *
 * @author Juliana e João Pedro
 */
public class RNGTests {


  public static void main(String[] args) {
      PRNG ale = new PRNG(1);
      int n = ale.nextInt(1000); //quantidade maxima de bits
      for (int i = 0; i < 1000; i++) {
          n = ale.nextInt(1000);
         // System.out.println(i+":");
          System.out.println(n);
      }
  }

  }
public class PRNG {

   private long last;
   private long inc;

   public PRNG(long seed) {
      seed = System.nanoTime();
      this.last = seed | 1;
      inc = seed;
   }

   public int nextInt(int max) {
      last ^= (last << 400);
      last ^= (last >>> 700);
      last ^= (last << 100);
      inc += 1234567891234567891L;  //num magico(incremento)
      int out = (int) ((last + inc) % max);
      return (out < 0) ? -out : out;
   }
}
```

Figura 1 – XorShift produzido

A fim de responder a pergunta “O que difere o algoritmo em estudo do Java.util.Random”, devemos compará-los.

O algoritmo da biblioteca java combina o código de gerador do tipo Xorshift com um gerador congruencial a fim de diversificar os resultados o máximo possível:

```
protected int next(int bits) {
 long oldseed, nextseed;
 AtomicLong seed = this.seed;
   do {
  oldseed = seed.get();
  nextseed = (oldseed * multiplier + addend) & mask;  (fórmula do gerador congruencial)
       } while (!seed.compareAndSet(oldseed, nextseed));
    return (int)(nextseed >>> (48 - bits)); (operação de shift)
  }
```

Figura 2 – Java.util.Random(Gerador pseudo aleatório)

Ambos utilizam como seed(semente) o valor obtido usando System.nanoTime. O código implementado tem ordem de complexidade de tempo O(n) já que executa as chamadas de uma função O(1) n vezes dentro de um “for”. O código util.Random do java tem complexidade de O(1) por fazer chamadas do método nextInt que também é O(1) -ele só é executado uma vez, a não ser que o valor resultante seja igual à semente que o originou-, se for utilizado também n vezes em um laço de repetição, terá também complexidade O(n), logo suas complexidades de tempo são iguais. Após testes imprimindo vários valores com o loop até o algoritmo começar a repetir uma mesma sequência “randomizada”, encontramos o período de 144 com a execução do código xorshift, o que é igual a 2^7+16, enquanto segundo a wikipédia (Wikipedia, 2021b) o período do algoritmo util.Random é de cerca de 2^{32} o que indica que o código util.Random pode retornar valores mais variados e menos previsíveis, devido ao período maior de variabilidade antes de sofrer repetições garantidas.(Wikipedia, 2021b)

TABELA: XORSHIFT

100%	X	Y	75%	X	Y	50%	X	Y	25%	X	Y	10%	X	Y	5%	X	Y
559	559	300	559	559	300	559	559	300	559	559	300	559	559	300	559	559	300
300	562	533	300	562	533	300	562	533	300	562	533	300	562	533	300	562	533
562	74	571	562	74	571	562	74	571	562	74	571	562	74	571	562	74	571
533	913	470	533	913	470	533	913	470	533	913	470	533	913	470	533		
74	247	988	74	247	988	74	247	988	74	247	988	74	247	988	74		
571	874	221	571	874	221	571	874	221	571	874	221	571	874	221	571		
913	762	259	913	762	259	913	762	259	913	762	259	913	762	259			
470	225	158	470	225	935	470	225	935	470	225	935	470					
247	935	676	247	676	186	247	676	186	247	676	186	247					
988	186	909	988	909	450	988	909	450	988	909	450	988					
874	450	947	874	947	537	874	947	537	874	947	537	874					
221	537	846	221	846	623	221	846	623	221	846	623	221					
762	623	364	762	364	498	762	364	498	762	364	498	762					
259	498	597	259	597	138	259	597	138	259	597	138	259					
225	138	635	225	635	849	225	635	849	225	635	849						
935	311	948	935	534	311	935	534	311	935	534	311						
676	810	285	676	948	810	676	948	810	676	948	810						
186	826	677	186	285	826	186	285	826	186	285	826						
909	161	222	909	677	161	909	677	161	909								
450	999	260	450	222	999	450	222	999	450								
947	122	973	947	260	122	947	260	122	947								
537	514	989	537	973	514	537	973	514	537								
846	473	910	846	989	473	846	989	473	846								
623	687	572	623	910	687	623	910	687	623								
364	434	339	364	572	434	364	572	434	364								
498	202	301	498	339	202	498	339	202	498								
597	785	402	597	301	785	597	301	785	597								
138	375	884	138	402	375	138	402	375	138								
635	746	651	635	884	746	635	884	746	635								
849	890	613	849	651	890	849	651	890	849								
534	97	714	534	613	97	534	613	97	534								
311	937	196	311	714	937	311	714	937	311								
948	58	963	948	196	58	948	196	58	948								
810	422	925	810	963	422	810	963	422	810								
285	409	26	285	925	409	285	925	409	285								
826	249	508	826	26	249	826	26	249	826								
677	246	275	677	508	246	677											
161	734	237	161	275	734	161											
222	895	338	222	237	895	222											
999	561	820	999	338	561	999											
260	934	587	260	820	934	260											
122	46	549	122	587	46	122											
973	583	650	973	549	583	973											
514	873	132	514	650	873	514											
989	622	899	989	132	622	989											
473	358	861	473	899	358	473											
910	271	962	910	861	271	910											
687	185	444	687	962	185	687											
572	310	211	572	444	310	572											
434	670	173	434	211	670	434											
339	959	274	339	173	959	339											
202	497	756	202	274	497	202											
301	998	523	301	756	998	301											
785	982	131	785	523	982	785											
402	647	586	402			402											
375	809	548	375			375											
884	686	835	884			884											
746	294	819	746			746											
651	335	898	651			651											
890	121	236	890			890											
613	374	469	613			613											
97	606	507	97			97											
714	23	406	714			714											
937	433	924	937			937											
196	62	157	196			196											
58	918	195	58			58											
963	711	94	963			963											
422	871	612	422			422											
925	750	845	925			925											
409	386	883	409			409											
26	399	782	26			26											
249			249			249											
508			508														
246			246														
275			275														
734			734														
237			237														
895			895														
338			338														
561			561														
820			820														
934			934														
587			587														
46			46														
549			549														
583			583														
650			650														
873			873														
132			132														
622			622														
899			899														
358			358														
861			861														
271			271														
962			962														
185			185														
444			444														
310			310														
211			211														
670			670														
173			173														
959			959														
274			274														
497			497														
756			756														
998			998														
523			523														
982			982														
131																	
647																	
586																	
809																	
548																	
686																	
835																	
294																	
819																	
335																	
898																	
121																	
236																	
374																	
469																	
606																	
507																	
23																	
406																	
433																	
924																	
62																	
157																	
918																	
195																	
711																	
94																	
871																	
612																	
750																	
845																	
386																	
883																	
399																	
782																	

Figura 3 – Valores obtidos em um período

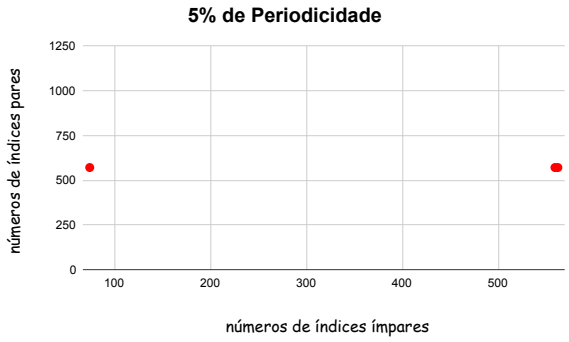
TABELA: JAVA.UTIL.RANDOM

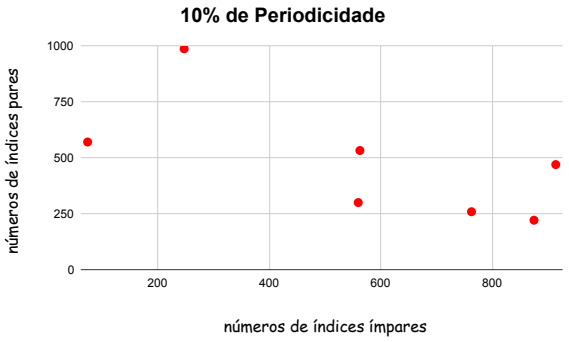
100%	X	Y	75%	X	Y	50%	X	Y	25%	X	Y	10%	X	Y	5%	X	Y
722	722	647	722	722	647	722	722	647	722	722	647	722	722	647	722	722	647
647	153	399	647	153	399	647	153	399	647	153	399	647	153	399	647	153	399
153	939	794	153	939	794	153	939	794	153	939	794	153	939	794	153	939	794
399	98	5	399	98	5	399	98	5	399	98	5	399	98	5	399		
939	668	853	939	668	853	939	668	853	939	668	853	939	668	853	939		
794	525	370	794	525	370	794	525	370	794	525	370	794	525	370	794		
98	772	487	98	772	487	98	772	487	98	772	487	98	772	487			
5	764	39	5	764	39	5	764	39	5	764	39	5					
668	833	636	668	833	636	668	833	636	668	833	636	668					
853	952	983	853	952	983	853	952	983	853	952	983	853					
525	635	767	525	635	767	525	635	767	525	635	767	525					
370	795	785	370	795	785	370	795	785	370	795	785	370					
772	497	947	772	497	947	772	497	947	772	497	947	772					
487	457	289	487	457	289	487	457	289	487	457	289	487					
764	391	887	764	391	887	764	391	887	764	391	887						
39	940	918	39	940	918	39	940	918	39	940	918						
833	845	79	833	845	79	833	845	79	833	845	79						
636	579	582	636	579	582	636	579	582	636	579	582						
952	21	55	952	21	55	952	21	55	952								
983	679	990	983	679	990	983	679	990	983								
635	288	331	635	288	331	635	288	331	635								
767	946	670	767	946	670	767	946	670	767								
795	461	9	795	461	9	795	461	9	795								
785	892	1	785	892	1	785	892	1	785								
497	927	648	497	927	648	497	927	648	497								
947	307	138	947	307	138	947	307	138	947								
457	90	978	457	90	978	457	90	978	457								
289	974	386	289	974	386	289	974	386	289								
391	530	421	391	530	421	391	530	421	391								
887	302	774	887	302	774	887	302	774	887								
940	888	855	940	888	855	940	888	855	940								
918	778	268	918	778	268	918	778	268	918								
845	778	271	845	778	271	845	778	271	845								
79	645	646	79	645	646	79	645	646	79								
579	550	113	579	550	113	579	550	113	579								
582	502	368	582	502	368	582	502	368	582								
21	411	537	21	411	537	21											
55	188	716	55	188	716	55											
679	660	281	679	660	281	679											
990	652	861	990	652	861	990											
288	417	782	288	417	782	288											
331	416	408	331	416	408	331											
946	675	108	946	675	108	946											
670	454	934	670	454	934	670											
461	548	129	461	548	129	461											
9	546	62	9	546	62	9											
892	689	575	892	689	575	892											
1	265	743	1	265	743	1											
927	254	915	927	254	915	927											
648	811	163	648	811	163	648											
307	412	827	307	412	827	307											
138	556	392	138	556	392	138											
90	721	109	90	721	109	90											
978	967	216	978	967	216	978											
974	452	794	974			974											
386	951	856	386			386											
530	187	585	530			530											
421	220	317	421			421											
302	523	319	302			302											
774	710	442	774			774											
888	171	724	888			888											
855	978	606	855			855											
778	670	18	778			778											
268	606	819	268			268											
778	787	119	778			778											
271	938	721	271			271											
645	194	949	645			645											
646	280	777	646			646											
550	917	257	550			550											
113	568	70	113			113											
502	153	760	502			502											
368	347	165	368			368											
411			411			411											
537			537			537											
188			188			188											
716			716			716											
660			660			660											
281			281			281											
652			652			652											
861			861			861											
417			417			417											
782			782			782											
416			416			416											
408			408			408											
675			675			675											
108			108			108											
454			454			454											
934			934			934											
548			548			548											
129			129			129											
546			546			546											
62			62			62											
689			689			689											
575			575			575											
265			265			265											
743			743			743											
254			254			254											
915			915			915											
811			811			811											
163			163			163											
412			412			412											
827			827			827											
556			556			556											
392			392			392											
721			721			721											
109			109			109											
967			967			967											
216			216			216											
452																	
794																	
951																	
856																	
187																	
585																	
220																	
317																	
523																	
319																	
710																	
442																	
171																	
724																	
978																	
606																	
670																	
18																	
606																	
819																	
787																	
119																	
938																	
721																	
194																	
949																	
280																	
777																	
917																	
257																	
568																	
70																	
153																	
760																	
347																	
165																	

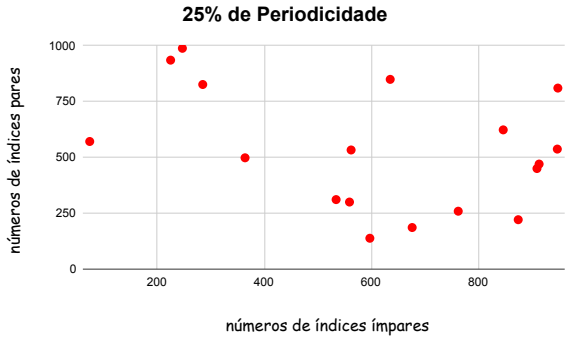
Figura 4 – Valores obtidos em um período

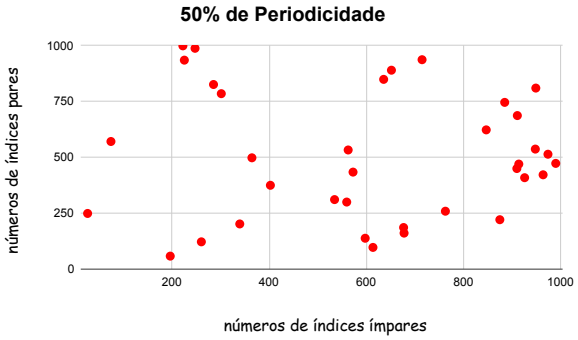
Para a melhor compreensão de como é a aleatoriedade dos gráficos tanto o XorShift implementado, quanto o gerador do java, foi realizado gráficos em duas dimensões, com um par de coordenadas x e y, definido pelas seguintes periodicidades: 5%, 10%,25%,50%, 75% e 100%.

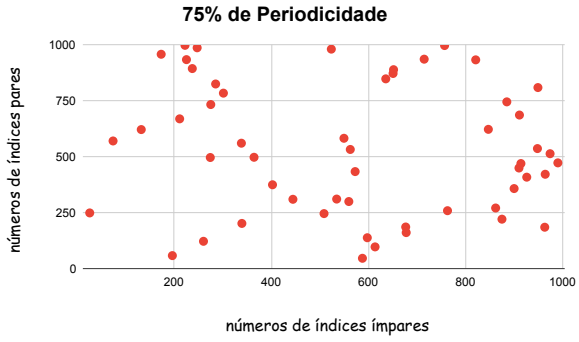
GRÁFICO EM XORSHIFT:











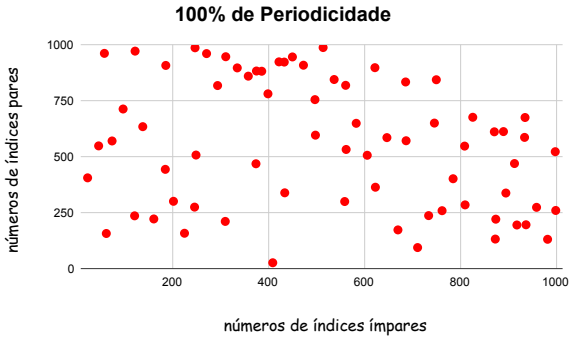
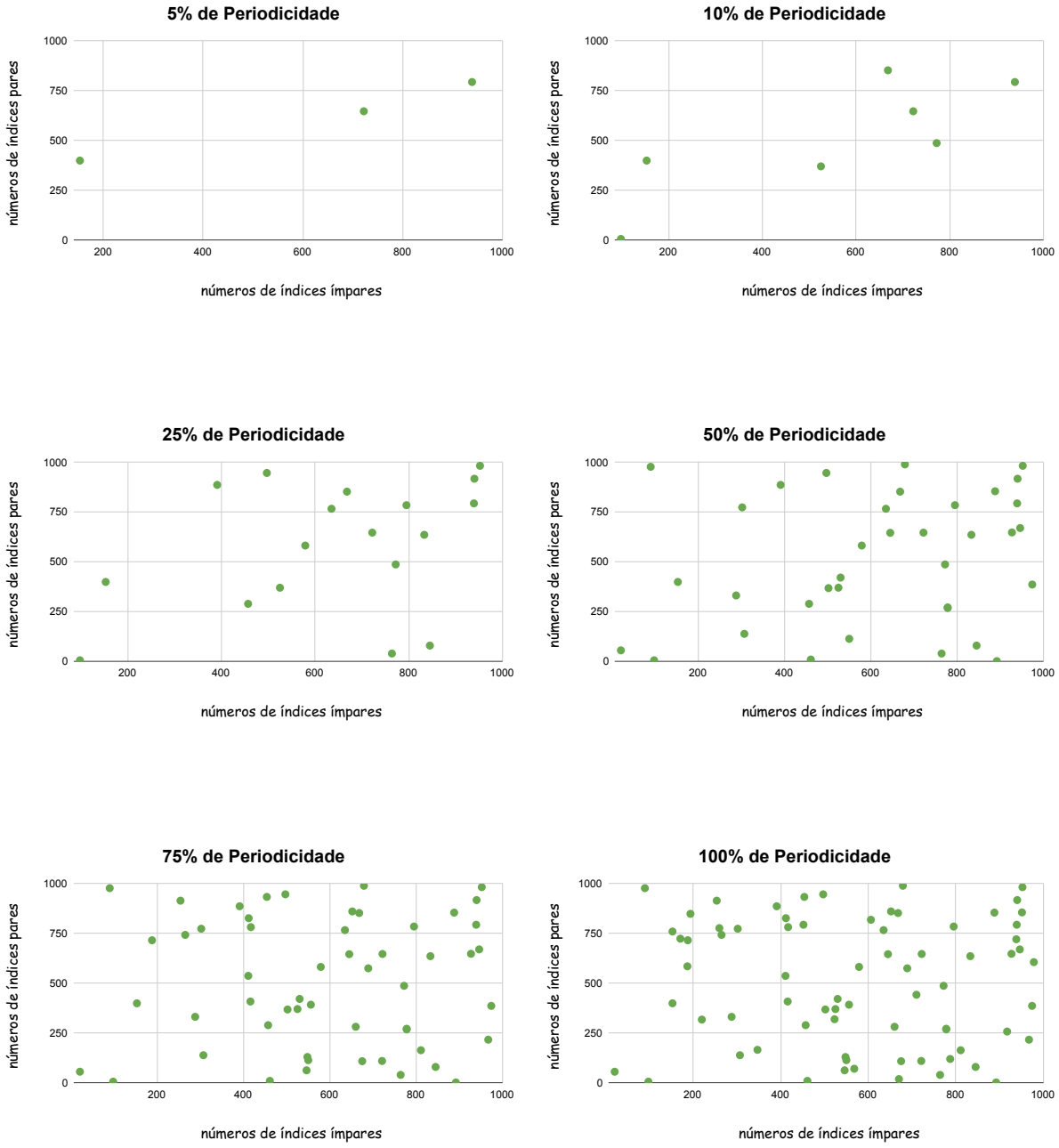


GRÁFICO EM JAVA:



Ao compilar os códigos, tanto o Xorshift elaborado, quanto o Random do Java, pode-se perceber a aleatoriedade gerada por ambos, e a partir disso foi realizado os gráficos juntamente as tabelas com os devidos períodos, na execução das coordenadas, foi empregue para os números em X os índices ímpares da tabela - parte horizontal desta, já para o Y os índices pares da tabela - parte vertical desta. O gráfico do java.util.Random foi feito baseado no período do codigo xorshift, pois o período dele próprio seria grande demais.

3 Conclusão

Contudo, pode-se concluir que o estudo dos geradores de valores pseudo aleatórios enfatizou a noção de que valores gerados computacionalmente nunca são verdadeiramente aleatórios, pois são advindos de operações lógicas e matemáticas. Destarte, a comparação do método mais simples, Xorshift, com o código mais sofisticado de java.util.Random demonstrou que um gerador complexo, com mais operações, é melhor pois gera sequências de valores com menos padrões e coincidências.

Referências

Wikipedia. *Pseudoaleatoriedade — Wikipédia, a enciclopédia livre*. 2021. Disponível em: <https://pt.wikipedia.org/wiki/Pseudoaleatoriedade#:>. Acesso em: 9 de setembro 2022. Citado na página 1.

Wikipedia. *Pseudoaleatoriedade — Wikipédia, a enciclopédia livre*. 2021. Disponível em: <https://en.wikipedia.org/wiki/Linear_congruential_generator>. Acesso em: 10 de setembro 2022. Citado na página 3.