Universidade Federal de Pernambuco

Departamento de Eletrônica e Sistemas

Curso de Engenharia Eletrônica

Análise de gerador de números pseudoaleatórios baseado em testes de divisibilidade

Trabalho de Conclusão de Curso de Graduação

por

Vágner Tôrres do Couto

Orientador: Ricardo Menezes Campello de Souza

Recife, Outubro / 2025

Vágner Tôrres do Couto

Análise de gerador de números pseudoaleatórios baseado em testes de divisibilidade

Monografia apresentada ao Curso de Engenharia Eletrônica, como requisito parcial para a obtenção do Título de Bacharel em Engenharia Eletrônica, Centro de Tecnologia e Geociências da Universidade Federal de Pernambuco.

Orientador: Ricardo Menezes Campello de Souza

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Couto, Vágner Torres do.
Análise de gerador de números pseudoaleatórios baseado em testes de divisibilidade / Vágner Torres do Couto. - Recife, 2025.
42 p. : il., tab.

Orientador(a): Ricardo Menezes Campello de Souza
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Eletrônica - Bacharelado, 2025.
Inclui referências, apêndices.

1. Gerador de números pseudoaleatórios. 2. Estatística. 3. Testes de divisibilidade. 4. Python. I. Souza, Ricardo Menezes Campello de. (Orientação). II. Título.

620 CDD (22.ed.)

Vágner Tôrres do Couto

Análise de gerador de números pseudoaleatórios baseado em testes de divisibilidade

Trabalho de Conclusão apresentado ao Curso de Graduação em Engenharia Eletrônica do Departamento de Eletrônica e Sistemas da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia Eletrônica.

Aprovado em: 09 de Setembro de 2025

Banca Examinadora:

Prof. Dr. Ricardo Menezes Campello de Souza
Departamento de Eletrônica e Sistemas da UFPE

Prof. Dr. Gilson Jeronimo da Silva Junior
Departamento de Eletrônica e Sistemas da UFPE

Prof. Dr. José Sampaio de Lemos Neto
Departamento de Eletrônica e Sistemas da UFPE

Recife

2025

Agradecimentos

Obrigado aos meus pais, Valter Alves do Couto Filho e Maria Elmiza Torres do Couto, pelo suporte ao longo do extenso curso.

Agradeço também a Prof. Ricardo Menezes Campello de Souza por toda a ajuda durante o curso e por aceitar ser o orientador deste trabalho e a Prof. Gilson Jerônimo da Silva Jr. por apresentar a ideia que deu origem ao mesmo.

Enfim, a todos os professores do DES e amigos do curso pelos anos de aprendizado, superação e irmandade necessários para completar esta experiência única.

Qualquer tecnologia suficientemente avançada é indistinguível de magia.

Arthur C. Clarke

RESUMO

Neste trabalho é apresentada uma nova classe de gerador de números pseudoaleatórios e a lógica para sua criação. Para contextualizar seu estudo, inicialmente, são revisadas a matemática subjacente e os testes utilizados para entender o comportamento das sequências geradas. Os testes são histograma, análise espectral, bondade de ajuste de Kolmogorov-Smirnov e a bateria de testes do NIST-STS. Em seguida são explicadas, de forma empírica, algumas características das sequências obtidas, buscando-se também entender como os parâmetros utilizados no gerador, como valores de entrada, limite de iterações e o número primo escolhido para o teste de divisibilidade, afetam a sequência gerada. Os geradores são implementados utilizando *Python*, através de notebooks Jupyter.

Os resultados são mostrados por meio de tabelas e gráficos que indicam fraquezas na aleatoriedade das sequências, que podem ser reduzidas com a mudança para um número primo maior, implicando que primos maiores podem aumentar o grau de aleatoriedade na sequência gerada.

Ao fim é feita a análise que interpreta que estes geradores apresentam falhas demais para uso com aplicações que exigem segurança criptográfica, mas possuem potencial para mais estudos sobre seu comportamento e possível uso em outros cenários.

Palavras-chave: Gerador de números pseudoaleatórios, Estatística, Testes de divisibilidade, Python.

ABSTRACT

This work presents a new class of pseudo-random number generators and the logic behind its creation. To contextualize the study, a review of the underlying mathematics and the tests used to understand the behavior of the generated sequences are initially presented. The tests include histograms, spectral analysis, Kolmogorov-Smirnov goodness-of-fit test, and the NIST-STS battery of tests.

Subsequently, an empirical explanation of the characteristics of the sequences is provided, seeking to understand how the parameters used in the generator, such as input values, iteration limits, and the chosen prime number for the divisibility test, affect the generated sequence. The generators are implemented using Python and Jupyter notebooks.

Finally, the results of the chosen tests are shown through tables and graphs, which indicate weaknesses in the randomness of the sequences, but which can be reduced with the change to a larger prime number, implying that larger primes can increase the degree of randomness in the generated sequence.

In conclusion, an analysis is made that interprets that these generators present too many limitations for use in applications that require cryptographic security, but have potential for further studies on their behavior and possible use in other scenarios.

Keywords: Pseudorandom Number Generator, Statistics, Divisibility tests, Python.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

FDC	Função Distribuição Contínua
FDE	Função Distribuição Empírica
GCL	Gerador de Congruência Linear
GTD	Gerador de Teste de Divisibilade
MDC	Maior Divisor Comum
NIST	*National Institute of Standards and Technology*
PRNG	*Pseudo Random Number Generator*
STS	*Statistical Test Suite*
UFPE	Universidade Federal de Pernambuco

LISTA DE SÍMBOLOS

$\Leftrightarrow$	Se e somente se
$\equiv$	Congruente a

SUMÁRIO

1 INTRODUÇÃO

Ao longo da história da matemática, testes de divisibilidade foram criados para facilitar a fatoração e servirem de testes de composição para números inteiros. Recentemente, um garoto na Inglaterra, Chika Ofili, encontrou um algoritmo recursivo para encontrar divisibilidade por 7 [1]. O resultado é conhecido há bastante tempo, mas o fato de uma mente tão jovem encontrar este resultado independentemente é ainda um feito admirável. Este resultado apresenta não só um resultado para a divisibilidade por 7, mas também uma forma de gerar algoritmos de teste de divisibilidade utilizando uma base de representação numérica, por exemplo, decimal, e aritmética básica.

Este trabalho irá analisar geradores de números pseudoaleatórios (PRNGs) criados a partir deste processo utilizando testes comuns, como o de Kolmogorov-Smirnov, e testes mais visuais como o teste espectral.

1.1 Objetivo Geral

Apresentar uma nova classe de geradores pseudo-aleatórios, analisando-se a complexidade de geração e realizando-se uma bateria de testes estatísticos para avaliar as características de aleatoriedade das sequências geradas.

1.2 Organização do texto

O trabalho está dividido em cinco capítulos e um apêndice. As referências bibliográficas encontram-se final do documento.

- O capítulo 2 é uma apresentação da teoria utilizada no resto do trabalho.
- O capítulo 3 introduz os geradores, mostra alguns exemplos deles e descreve o algoritmo generalizado e suas limitações.
- O capítulo 4 mostra um estudo melhor sobre o gerador com p=7, testagem do efeito dos parâmetros do gerador e uma bateria de testes aplicada sobre vários geradores desta classe.
- O capítulo 5 traz a conclusão do estudo e possíveis trabalhos futuros feitos sobre as ideias apresentadas neste.

- O apêndice A contém o URL com o código fonte utilizado para implementar o gerador.

2 PRELIMINARES MATEMÁTICOS

A divisão de a, inteiro não-negativo, por um número, inteiro positivo, n é descrita como

$$a = qn + r, \tag{2.1}$$

em que q e r, denominados respectivamente o quociente e o resto da divisão, são únicos [2]. Seja x um número inteiro, descrito numa base B, tal que

$$D_B(x) = d_0 + d_1B^1 + d_2B^2 + ... + d_lB^l, \tag{2.2}$$

em que os números $0 \leq d_i < B$, para $0 \leq i \leq l$. A notação $D_B(x)$ será utilizada para descrever a representação do número x na base B. Dividindo $D_B(x)$ pela base B e aplicando (2.1), tem-se

$$D_B(x) = qB + d_0, \tag{2.3}$$

separando assim o dígito menos significativo d_0 da representação. Toma-se um número primo p diferente de B tal que, por definição, $\text{MDC}(B, p) = 1$. Isto implica na existência de um inverso de $B \pmod p$, f, tal que $B \times f \equiv 1 \pmod p$. Multiplicando (2.3) por f e reduzindo módulo p, tem-se

$$D_B(x)f \equiv q + d_0f \pmod p. \tag{2.4}$$

Utilizando esta congruência, pode-se criar um algoritmo para teste de divisibilidade por primos, baseado em que

$$D_B(x) \equiv 0 \pmod p \Leftrightarrow q + d_0f \equiv 0 \pmod p. \tag{2.5}$$

O que esta equação significa é que se $p|x$, então $q + d_0f$ também é divisível por p. Daí surge o algoritmo a seguir.

2.1 Algoritmo para p=7 com base decimal

O teste que Chika Ofili descobriu utiliza $B = 10$ e $p = 7$ [1]. O que resulta disto é

$$10^{-1} \equiv 5 \pmod 7 \Rightarrow q + 5d_0 \equiv 0 \pmod 7.$$

Inicialmente, uma consideração importante é a seguinte: este algoritmo foi feito para diminuir a dificuldade de calcular se a congruência é zero, ou seja, o valor é divisível pelo divisorar à esquerda da ilustração sem destaque gráfico;. Uma explicação melhor sobre estes testes pode ser encontrada em [3]. A cada iteração, o número vai diminuir em um dígito até ficar de ordem baixa o suficiente para se caracterizar a divisibilidade. A seguir se encontram exemplos deste processo nas Tabelas 1 e 2.

Tabela 1: Divisibilidade por 7 de N=1578.

d_3	d_2	d_1	d_0	
1	5	7	8	$157 + 5 \times 8 = 197$
	1	9	7	$19 + 5 \times 7 = 54$
		5	4	Não divisível por 7

Fonte: Autor

Tabela 2: Divisibilidade por 7 de N=1519.

d_3	d_2	d_1	d_0	
1	5	1	9	$151 + 5 \times 9 = 196$
	1	9	6	$19 + 5 \times 6 = 49$
		4	9	Divisível por 7

Fonte: Autor

2.2 O resultado de cada iteração

Este processo pode ser iterado um número arbitrário de vezes, sendo necessário sempre haver um critério de parada. No exemplo da Tabela 3, está demonstrado um processo com o critério de parada quando se alcança um resultado de um dígito.

Tabela 3: Ilustrando o critério de parada para o processo iterativo com N = 1578.

d_3	d_2	d_1	d_0	
1	5	7	8	$157 + 5 \times 8 = 197$
	1	9	7	$19 + 5 \times 7 = 54$
		5	4	$5 + 5 \times 4 = 25$
		2	5	$2 + 5 \times 5 = 27$
		2	7	$2 + 5 \times 7 = 37$
		3	7	$3 + 5 \times 7 = 38$
		3	8	$3 + 5 \times 8 = 43$
		4	3	$4 + 5 \times 3 = 19$
		1	9	$1 + 5 \times 9 = 46$
		4	6	$4 + 5 \times 6 = 34$
		3	4	$3 + 5 \times 4 = 23$
		2	3	$2 + 5 \times 3 = 17$
		1	7	$1 + 5 \times 7 = 36$
		3	6	$3 + 5 \times 6 = 33$
		3	3	$3 + 5 \times 3 = 18$
		1	8	$1 + 5 \times 8 = 41$
		4	1	$4 + 5 \times 1 = 9$
			9	Resultado de um dígito

Fonte: Autor

A primeira observação em relação ao exemplo anterior é sobre o número de iterações que resultou em apenas dois digitos, o que é importante na complexidade do algoritmo para o cálculo deste resultado de um dígito, caso este seja o critério de parada. A segunda observação é que este resultado, módulo 7, é diferente do resto de 1578 dividido por 7, ou seja, $1578 \equiv 3 \pmod 7 \neq 9 \equiv 2 \pmod 7$. A terceira observação é que claramente o processo inteiro é determinístico, então se um número N, durante seu processamento, resultar em 36, por exemplo, o resultado final é 9.

2.3 Teste de hipótese nula

Para avaliar processos de natureza estatística, o teste de hipótese nula foi formulado. A versão moderna deste tipo de teste é uma junção, controversa e debatida no mundo da estatística [4], dos métodos de Ronald Fisher [5] e de Jerzy Neyman e Egon Pearson [6].

A hipótese nula, H_0, assume que a suposição testada é verdade. Para representar o possível erro na suposição utiliza-se a hipótese alternativa, H_a, que é o caso de haver provas para contradizer a hipótese nula. Deste fato, herdado de Neyman e Pearson, surge a discussão dos possíveis erros na testagem, o erro do tipo 1, onde H_0 é aceita sendo falsa e o erro do tipo 2, onde H_a é aceita sendo falsa. Os erros possíveis quando se utiliza o método de Neyman e Pearson têm suas probabilidades limitadas durante a formulação do teste.

Para cada teste, o valor crítico de p será decidido como um nível de significância, α. Esta parte da definição gera bastante confusão, pois nas teorias de Fisher e Neyman-Pearson ambas possuem níveis de significância, um valor α e valores-p. A confusão dos problemas filosóficos e originários destas grandezas e nomes é melhor visto em [4]. Para a abordagem atual, originada da mistura destas ideias, α será apenas um valor crítico para a estatística p de cada teste onde, caso $p > \alpha$, o teste é evidência que H_0 é verdade. Caso contrário, é evidência que H_a é verdade.

O valor do nível de significância, apesar de não ter um valor fixo convencionalmente, em pesquisa para aplicações criptográficas, é escolhido como $\alpha = 0,01$ [7]. Isto implica que para aceitar H_0 não só o valor de p tem que ser menor que α em um teste, mas que é esperado em testagem múltipla que no máximo 1% dos resultados sejam a rejeição de H_0 para esta hipótese ainda ser aceita como verdadeira. Este será o critério utilizado onde for aplicável.

2.4 Geradores congruentes lineares

Geradores congruentes lineares, GCLs, são uma classe de geradores semelhantes aos sugeridos neste trabalho [8]. A operação que define estes geradores é

$$X_{n+1} = (aX_n + c) \pmod{m} \tag{2.6}$$

em que a, c e m são os parâmetros do gerador. O único caso utilizado neste trabalho é o caso em que $c \neq 0$, cujo GCL segue o teorema de Hull-Dobel [8]. Uma vez utilizado este critério para geração, sabe-se que estes geradores possuem o período igual a m. As dificuldades da montagem e execução de GCLs são discutidas em [8]. Estes geradores possuem teorias bem definidas quanto à sua aleatoriedade e segurança, como a potência do gerador, mas são teorias matemáticas aplicadas sobre a operação específica da congruência linear descrita em (2.6). Uma vez superados os problemas que surgem com os GCLs, sabe-se que eles são geradores que funcionam excepcionalmente bem para geração de distribuições uniformes. Assim sendo, eles são usados para comparar com os resultados dos geradores propostos.

2.5 O teste espectral

O teste espectral é uma forma simples de detectar subsequências em uma sequência. Assumindo que a sequência é x, este teste consiste em desenhar pontos nas coordenadas $(x[n], x[n+1])$. O que isso demonstra são possíveis relações simples entre amostras consecutivas, em que, por exemplo, resultados linearmente relacionados aparecem como retas neste gráfico. Também pode-se procurar padrões em três dimensões com mais um atraso no terceiro eixo com os pontos aparecendo dispostos em qualquer hiperplano do sistema de coordenadas utilizado: uma reta para duas dimensões, um plano para 3 dimensões.

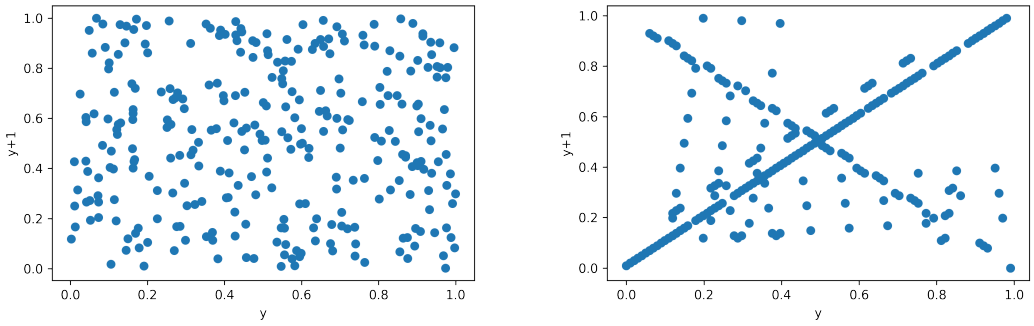
Este teste foi criado para GCLs, já que é feito para buscar a distância menor entre os hiperplanos que contêm todos os pontos com as entradas geradas [8]. Porém, o mesmo também funcionou visualmente para alguns geradores da classe apresentada, mostrando relações muito simples entre as suas saídas. Um exemplo de um gerador ruim é mostrado na Figura 1(b).

2.6 O teste de bondade de ajuste de Kolmogorov-Smirnov

A forma mais recomendada de estudar geradores de sequências pseudo-aleatórias é estudar suas falhas e vícios. Uma dessas formas, como é mostrado em [8], é um teste relacionando as Funções de Distribuição Cumulativa de uma sequência "alvo" e a sequência sendo analisada.

Kolmogorov sugeriu o teste em 1933 [9], inspirado por uma medida criada por Mises

Fonte: Autor



(a) (b)

Figura 1: Gráfico espectral para sequências de fontes diferentes: a) GCL bom . b) Gerador baseado em teste de divisibilidade com $p = 101$.

em 1931 para julgar se uma Função Distribuição Empírica (FDE) pode pertencer a uma Função Distribuição Contínua (FDC). Smirnov expandiu este teste para a comparação entre duas FDEs e para testes unilaterais, observando apenas a diferença superior ou inferior entre as duas funções [10]. A definição formal da estatística de Kolmogorov-Smirnov é dada por

$$D = \sup_x |F(x) - F_n(x)|, \tag{2.7}$$

em que D é a maior diferença absoluta entre a FDE obtida, $F_n(x)$, e a FDC, $F(x)$, para todo x [11]. As tabelas a seguir dão o limite superior de D para julgar um resultado como aceitável, com um certo grau de significância α. Observa-se que estes resultados não implicam em respostas definitivas sobre uma dada FDE, mas sim em mais uma prova na defesa da hipótese nula ou hipótese alternativa. Ainda sobre as tabelas, nota-se que quanto menor o valor de α, maior o limite superior de D; ou seja, se o teste prevê mais falhas do tipo 2, então o limite superior da estatística é maior.

Tabela 4: Limite superior de D para um dado α e tamanho de sequência n.

n \ α	0.1	0.05	0.025	0.01	0.005
1	0.900	0.925	0.950	0.975	0.995
2	0.684	0.726	0.776	0.842	0.929
3	0.565	0.597	0.642	0.708	0.823
4	0.494	0.525	0.564	0.624	0.733
5	0.446	0.474	0.510	0.565	0.669
6	0.410	0.436	0.470	0.521	0.618
7	0.381	0.405	0.438	0.486	0.577
8	0.358	0.381	0.411	0.457	0.543
9	0.339	0.360	0.388	0.432	0.514
10	0.322	0.342	0.368	0.410	0.490
11	0.307	0.326	0.352	0.391	0.468
12	0.295	0.313	0.338	0.375	0.450
13	0.284	0.302	0.325	0.361	0.433
14	0.274	0.292	0.314	0.349	0.418
15	0.266	0.283	0.304	0.338	0.404
16	0.258	0.274	0.295	0.328	0.392
17	0.250	0.266	0.286	0.318	0.381
18	0.244	0.259	0.278	0.309	0.371
19	0.237	0.252	0.272	0.301	0.363
20	0.231	0.246	0.264	0.294	0.356
25	0.21	0.22	0.24	0.27	0.32
30	0.19	0.20	0.22	0.24	0.29
35	0.18	0.19	0.21	0.23	0.27
>35	$\frac{1.07}{\sqrt{n}}$	$\frac{1.14}{\sqrt{n}}$	$\frac{1.22}{\sqrt{n}}$	$\frac{1.36}{\sqrt{n}}$	$\frac{1.63}{\sqrt{n}}$

Fonte: https://real-statistics.com/statistics-tables/kolmogorov-smirnov-table/

3 SEQUÊNCIAS GERADAS COM $p = 7$

Como descrito na Seção 1.1.2, o algoritmo pode ser utilizado para alcançar o resultado de apenas um dígito na base escolhida. É importante destacar que o resultado de um dígito nem sempre existe, havendo em alguns casos laços infinitos de resultados. Isto ocorre ocasionalmente dependendo do número primo p e base B escolhidos, tendo que ser detectado caso a caso. A solução para estes casos é limitar o número de iterações com um valor de iterações condizente à ordem de grandeza dos números na entrada ou uma condição específica de parada. Um caso observado, que inspirou o estudo, está na tabela a seguir:

Tabela 5: Entradas de 0 a 104 com suas respectivas saídas de um dígito.

Entradas	Saídas
0-6	0 1 2 3 4 5 6
7-13	7 8 9 1 6 6 8
14-20	7 8 8 9 9 9 2
21-27	7 6 9 6 9 8 9
28-34	7 6 3 8 8 9 9
35-41	7 9 9 9 6 4 9
42-48	7 9 6 6 9 6 6
49-55	7 5 1 8 2 9 3
56-62	7 4 6 5 6 6 8
63-69	7 8 8 9 9 9 1
70-76	7 6 9 6 9 8 9
77-83	7 6 8 8 8 9 9
84-90	7 9 9 9 6 2 9
91-97	7 9 6 6 9 6 6
98-104	7 9 1 8 2 9 3

Fonte: Autor

Apesar de todo múltiplo de 7 resultar na saída 7, notou-se inicialmente que os 6 dígitos entre dois múltiplos aparentavam não possuir padrão ou ordem. Estas sequências de $p - 1$ dígitos são chamadas de “palavras” a partir deste ponto.

3.1 **O algoritmo generalizado**

Como mencionado, nem sempre o resultado de um dígito é alcançável, pois com números primos maiores, os inversos multiplicativos também são de mais de um algarismo. Por exemplo, o número 10 para p=11 possui inverso $10 \equiv -1 \mod p$, para p=13 possui inverso 4 e para p=17 possui inverso 12.

O algoritmo proposto consiste de três parâmetros fixos: o primo p a ser utilizado no módulo, a base B da representação numérica e o limite de iterações S escolhido para o processo. Com isto o algoritmo itera sobre a entrada até atingir uma das duas condições de parada: Alcançar um número menor que o primo p ou alcançar o limite de iterações S, que deve ser escolhido de acordo com o número de dígitos da entrada, maior ou igual a este número. A saída final aqui então é módulo p, para assim haver p saídas possíveis. O fluxograma para o algoritmo se encontra na Figura 2.

Visando estudar as propriedades destas sequências, foram testados outros algoritmos de divisibilidade. A ideia das palavras existe independente do valor de p escolhido, no sentido de haver periodicamente um resultado $\equiv 0 \pmod p$.

Fonte: Autor

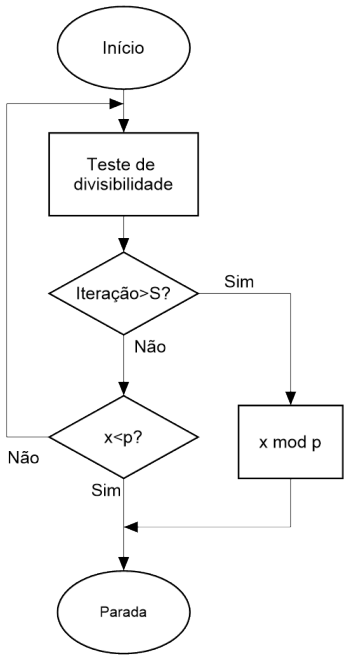


Figura 2: Fluxograma para uma entrada x do algoritmo.

3.2 Problemas encontrados com o algoritmo

Número de iterações

O número de iterações para uma entrada cresce linearmente com a sua ordem de grandeza. A exceção são certos números, como os da Tabela 3, que quando alcançam uma magnitude mais baixa, neste caso dois dígitos, continuam iterando com resultados de dois dígitos sem chegar em apenas um dígito. Para evitar casos inconsistentes como este, um limite de iterações é utilizado.

Ciclos infinitos

Há casos, como o número 49 no teste de divisibilidade por 7 em base decimal, em que o procedimento leva a um ciclo infinito, pois $4 + 5 \times 9 = 49$. Estes problemas só podem acontecer quando o número possuir menos dígitos que $9 \times f$, pois $d_0 \times f < B^2$ já que $d_0 < B$ e $f \leq B \pmod{p}$. Outro exemplo encontrado foi com o inverso $f = 4$, resultante da escolha $p = 13$. Com esta escolha, acontecem dois casos de enlace infinito com o critério de parada ser um resultado menor ou igual a 13, sem limite de iterações: 26, que resulta em 26, e a série 38-35-23-14-17-29-38 que, caso um resultado seja qualquer um destes, se repete indefinidamente. Resultados como estes afetam os histogramas da saída, pois terminam concentrando o histograma nestes grupos de números cíclicos. Por causa disto não seria recomendável utilizar primos que apresentassem este comportamento. Para encontrar se isto é comum no primo que foi utilizado, basta observar o número de iterações por entrada. Quanto mais este valor atingir o limite estabelecido para controlar os enlaces, mais este fenômeno está acontecendo caso o limite de iterações seja maior ou igual à ordem da entrada.

No código escrito em *Python* mostrado no anexo A, este limitante é calculado com o logaritmo na base 10 da primeira entrada somada a um número arbitrário cuja escolha é discutida mais à frente neste trabalho. Isto é para garantir que o algoritmo tenha iterações suficientes para chegar ao resultado com menor número de dígitos possível, o equivalente do "resultado de um dígito" no caso de $p = 7$, para qualquer número primo escolhido.

4 RESULTADOS

4.1 A regra da sequência $p = 7$

Tabela 6: Saídas de 5061 a 5155.

Linha	Saídas	Módulo 7
1	7 8 4 9 6 9 5	0 1 4 2 6 2 5
2	7 6 9 6 9 8 9	0 6 2 6 2 1 2
3	7 6 8 2 8 9 9	0 6 1 2 1 2 2
4	7 9 9 9 6 1 6	0 2 2 2 6 1 6
5	7 9 6 6 9 6 6	0 2 6 6 2 6 6
6	7 9 9 8 1 9 8	0 2 2 1 1 2 1
7	7 2 6 9 8 3 8	0 2 6 2 1 3 1
8	7 8 4 9 6 9 5	0 1 4 2 6 2 5
9	7 6 9 6 9 8 9	0 6 2 6 2 1 2
10	7 6 8 2 8 9 9	0 6 1 2 1 2 2
11	7 9 9 9 6 1 6	0 2 2 2 6 1 6
12	7 9 6 6 9 6 6	0 2 6 6 2 6 6
13	7 9 9 8 1 9 8	0 2 2 1 1 2 1
14	7 2 6 9 8 3 8	0 2 6 2 1 3 1
15	7 8 4 9 6 9 5	0 1 4 2 6 2 5

Fonte: Autor

O código, como descrito anteriormente, é feito na linguagem *Python* para gerar sequências com este processo. Implementado inicialmente sem limite de iterações, mas com detecção para o resultado 49 que resulta em ciclos infinitos, foi gerada uma sequência com valores na entrada de 0 a 1000000 com passo 1. Foi, então, utilizado um programa que buscava diferenças entre os blocos de 49 saídas que resultou nos seguintes números cujos ocorrem as mudanças: 50, 51, 52, 53, 54, 55, 57, 58, 59, 69, 79, 89, 99, 109, 129, 139, 149, 249, 349, 449, 549, 649, 849, 949, 1049, 2049, 3049, 4049, 5049, 6049, 8049, 9049, 10049, 20049, 30049, 40049, 50049, 60049, 80049, 90049, 100049, 200049, 300049, 400049, 500049, 600049, 800049, 900049. Após o número 149, nota-se que surge um padrão nestes

números. São sempre números que terminam em 49 e que não possuem dígitos não-nulos exceto pelo mais à esquerda diferente de 7. Além disto, percebeu-se onde acontecem as mudanças nas palavras, que é a entrada módulo 49. Assumindo a palavra [2, 6, 2, 1, 3, 1] ou a mais semelhante e ela como a primeira do bloco, é possível entender onde ocorre a mudança de acordo com esta regra. A observação deste fato é apenas experimental, sendo pura coincidência os locais das mudanças, já que nenhum comportamento similar foi observado com outros números primos.

4.2 **Efeitos de parâmetros no gerador**

Inicialmente, buscou-se entender a distribuição de números que surgem na saída com $p = 7$. Para isto vários histogramas foram feitos (Figura 3) junto com a análise espectral destas mesmas subsequências (Figura 4). Observou-se que, à medida que a sequência gerada se torna mais uniforme no histograma, ela também aparentava perder características de aleatoriedade com entradas na magnitude de 10^{99}. Ambos os gráficos demonstram o que já havia sido mencionado anteriormente: a distribuição das saídas varia de acordo com a entrada, reforçando a ideia de que este algoritmo pode ser utilizado para criar PRNGs. Posteriormente, notou-se que o limite de iterações estava definido como 100, o que causava a saída ordenada vista em (Figura 3(d)). Assim, tornou-se interessante testar como a variação desse valor afeta a distribuição da saída.

O teste espectral (Figura 4) demonstra a variação da saída com o aumento da magnitude da entrada. É também uma demonstração prática de que a homogeinidade do histograma não é indicador para a aleatoriedade da sequência gerada.

4.3 **O limite de iterações**

O limite de iterações tem forte influência na distribuição da saída, como se observa nos testes das Figuras 5 e 6. O histograma e teste espectral destes gráficos ambos foram gerados com uma sequência gerada com entrada iniciando em 10^{10} com passo 1 e 1000 amostras, variando apenas o limite de iterações. Nota-se que uma escolha adequada do limite de iterações irá afetar completamente a saída do gerador. Pelos testes aqui conduzidos, limites abaixo do número de dígitos no valor utilizado na entrada retorna saídas sem características de aleatoriedade.

Fonte: Autor

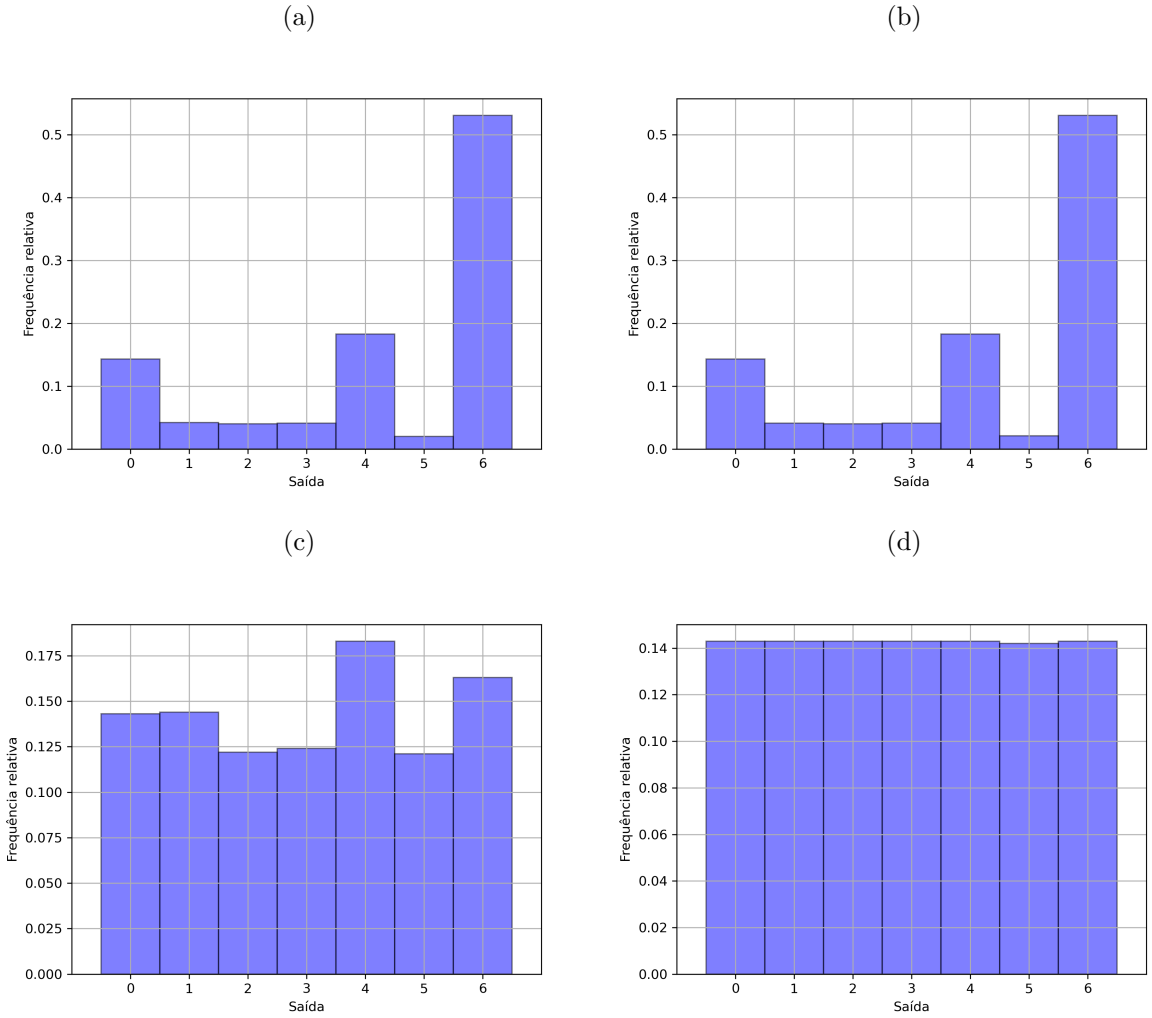


Figura 3: Frequência relativa de gerador $p = 7$ para 1000 entradas sequenciais com valor inicial: a) 10^{30}. b) 10^{60}. c) 10^{95}. d) 10^{99}.

Pode-se deduzir deste resultado, então, que a aleatoriedade da saída não está relacionada tão fortemente ao número na entrada do gerador e mais às iterações finais sobre os números de menos dígitos, mas não quer dizer que o valor na entrada não tem efeito, já que os testes espectrais variam mantendo o limite de iterações e variando os valores de entrada como visto nos histogramas da Figura 3.

4.4 **Teste de sequências**

Todo teste para sequências randômicas é pensado para procurar uma falha específica na sequência, por exemplo, blocos de valores repetidos ou desbalanceamentos na distribuição de zeros e uns numa sequência binária. Os testes estatísticos utilizando nú-

Fonte: Autor

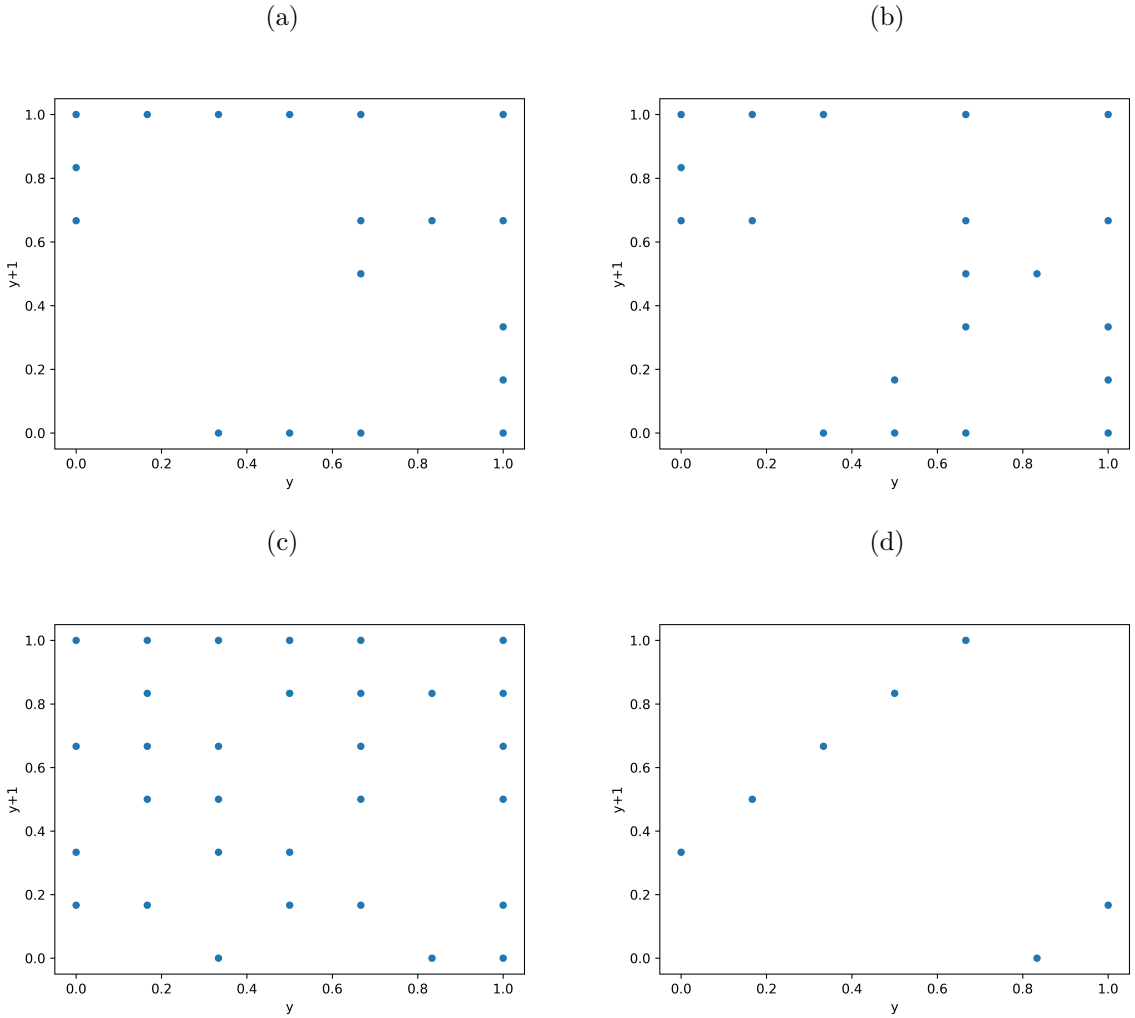


Figura 4: Teste espectral com gerador $p = 7$ com entradas de magnitude: a) 10^{30}. b) 10^{60}. c) 10^{95}. d) 10^{99} .

meros binários serão feitos convertendo as saídas módulo 2. Os testes aqui utilizados para fazer isto são: o teste espectral, o teste de bondade de ajuste de Kolmogorov-Smirnov e testes de sequência binária do NIST, como explicados em [7].

Para p = 7, todo teste é uma falha para sequências maiores de 49 elementos justo pela forma que as palavras se repetem como explicado anteriormente, então se torna redundante insistir na testagem com esta escolha de número primo.

4.5 **Teste com um GCL**

Nesta seção será submetido aos mesmos testes um GCL bom seguindo o teorema de Hull-Dobel. Isto servirá como uma referência de desempenho de um bom PRNG.

Fonte: Autor

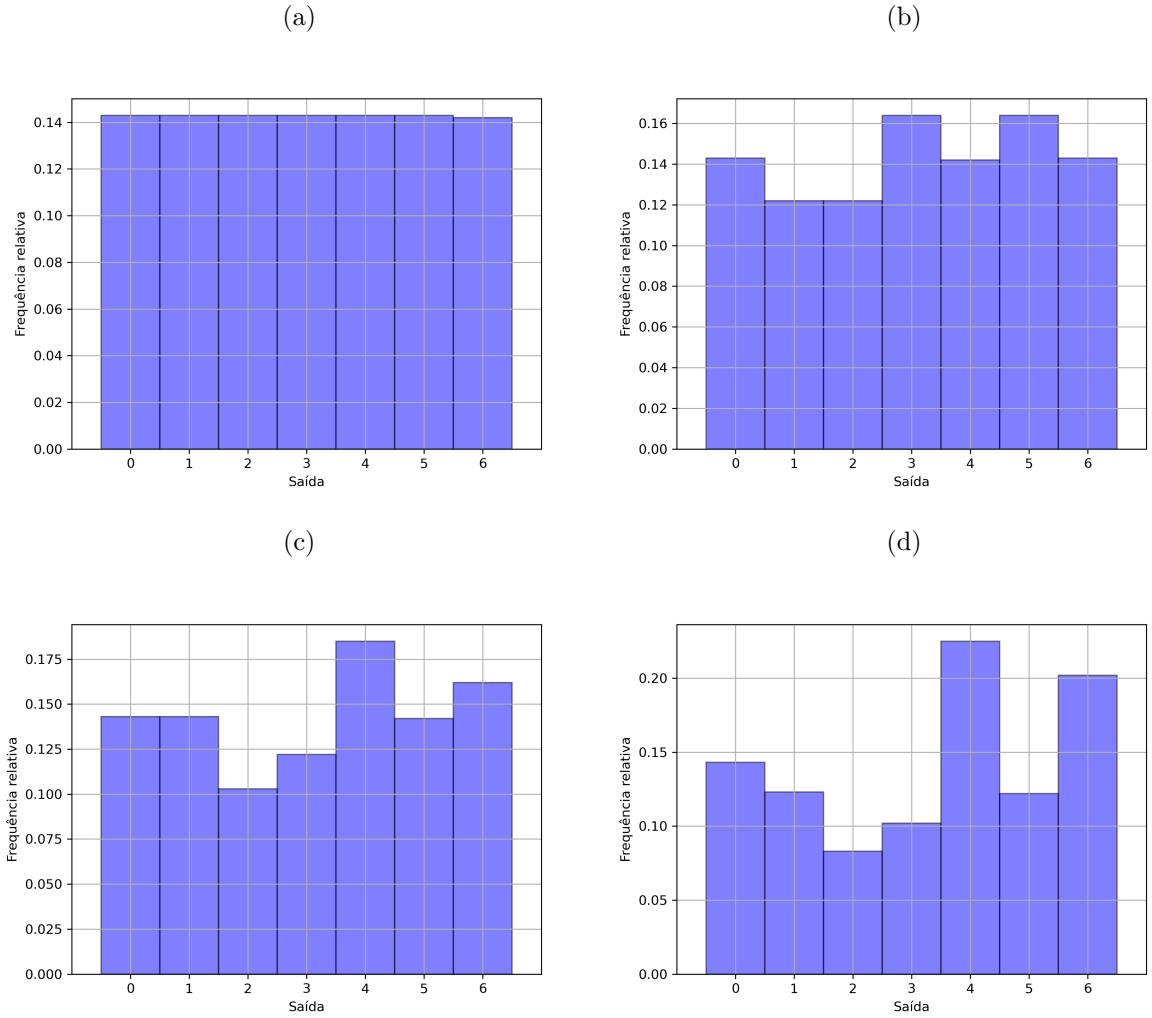


Figura 5: Histogramas com gerador $p = 7$ com entradas de magnitude 10^{10} com limite de iterações S: a) 10. b) 11. c)14. d) 16.

Para os parâmetros do gerador foram utilizados valores comuns padrão do GCC, com a = 1103515245, c = 12345 e $m = 2^{32}$, com a observação que a cumpre todos os requisitos do teorema, então com $c \neq 0$: c é coprimo com m, já que é ímpar e m é apenas uma potência de 2, $a - 1$ é divisível por 4 e sendo divisível por 4 também é divisível pelo único fator primo de m, 2. Foi gerado uma sequência de 10 números para repetir o teste que mais adiante é feito com o gerador apresentado neste trabalho. No histograma e para transformação em binário os valores da sequência foram multiplicados por 1000 por questões computacionais. Para os testes binários além disso foram arredondados para o inteiro mais próximo antes de ser aplicado o resto da divisão por 2.

Todo resultado observado condiz com a teoria deste tipo de gerador ser forte como

Fonte: Autor

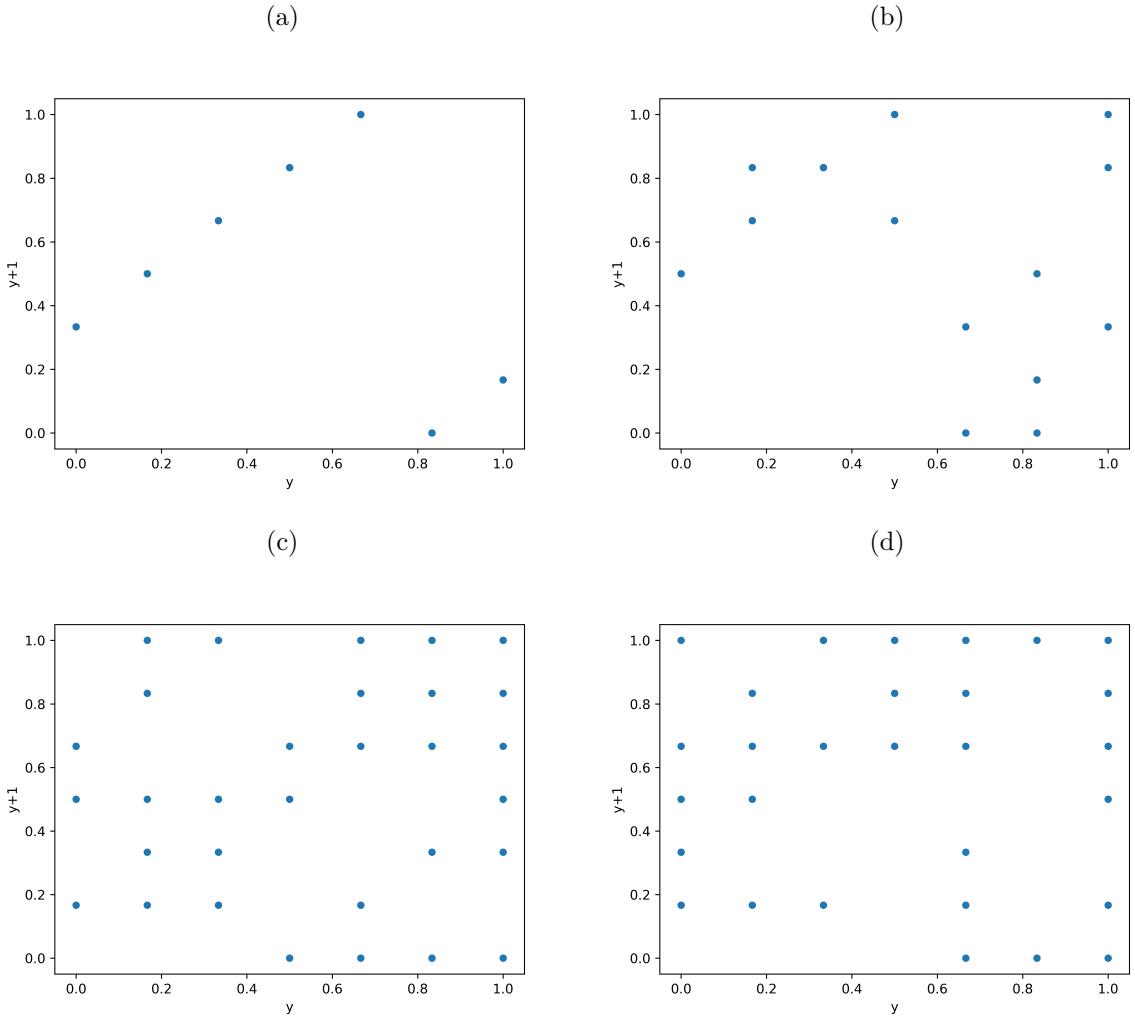


Figura 6: Teste espectral com om gerador $p = 7$ com entradas de magnitude 10^{10} com limite de iterações S: a) 10. b) 11. c)14. d) 16.

um PRNG. O histograma é uniforme como visto na (Figura 7a), o teste espectral não apresenta retas no caso com 2 dimensões visto na (Figura 7b). Os testes de Kolmogorov-Smirnov resultam em aprovações até com sequências mais longas de 1000 elementos, não só passando no teste que esperava $D \leq 0.043$, mas também com valores-p altos (Tabela 7). Os testes no NIST-STS em grande maioria resultam em aprovações, tanto na uniformidade dos valores-p como nos testes em si. Os únicos testes que reprovaram este gerador foram o *Universal*, que testa a compressibilidade dos dados e resulta em total falha, podendo ser causado pelo tamanho de sequência 10000 utilizado, e no *Aproximate Entropy* que testa a entropia de subsequências e resulta em 25 falhas em 100 sequências além da não-uniformidade dos valores-p (Tabela 8).

Fonte: Autor

(a) (b)

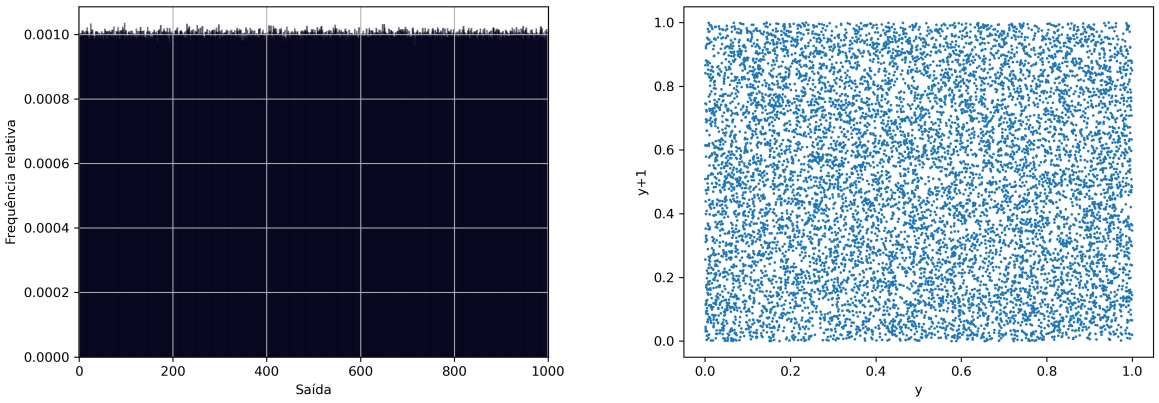


Figura 7: Testes gráficos com sequência gerada a partir de GCL de parâmetros a = 1103515245, c = 12345 e $m = 2^{32}$: a) Histograma. b) Teste espectral.

4.6 **Testando com outros números primos**

Ao observar que com $p = 7$ os resultados não eram favoráveis ao gerador, foi decidido fazer o mesmo teste com outros primos para observar o efeito disto nas sequências. Começando por primos de ordem de grandeza menor, os gráficos na Figura 8 mostram os histogramas para p = 11, p = 13, p = 37 e p = 6047.

Nota-se que primos diferentes geram sequências com histogramas diferentes, um bom sinal para a independência entre geradores que os utilizam. Apesar dos histogramas serem uma boa forma de detectar tendências da saída, eles não dão informação sobre padrões nas sequências. O teste espectral por outro lado demonstra isto claramente em algumas sequências, como na (Figura 9(a)), onde toda saída pode ser vista como três retas no teste espectral. Isso demonstra uma clara falha neste gerador. Apenas com o número primo maior, 6047, que aparece um teste espectral com menos problemas óbvios, visto em (Figura 9(d)). Então para os testes mais complexos será mais interessante aplicar apenas sobre geradores com p = 6047 ou primos de magnitude maior.

4.6.1 **Testes de Kolmogorov-Smirnov**

Este teste não é aplicável para p = 7, pois ele espera que a saída do gerador seja contínua, então apenas 7 possíveis resultados não é uma aproximação suficientemente

Tabela 7: Resultados de testes de Kolmogorov-Smirnov comparando a saída de um GCL a uma distribuição uniforme com sequências de 1000 elementos.

n	**Estatística**	**Valor-p**
0	0.0060	0.8536
1	0.0053	0.9345
2	0.0071	0.6782
3	0.0183	0.0023
4	0.0084	0.4773
5	0.0124	0.0916
6	0.0042	0.9937
7	0.0071	0.6892
8	0.0107	0.1990
9	0.0098	0.2904

Fonte: Autor

boa para uma saída contínua. Utilizando números primos maiores como parâmetro do gerador, é possível aproximar com uma função contínua. Um exemplo é p = 6047, S = 15, entradas começando de 11^{10}, como visto na Tabela 9.

Pela Tabela 4, o valor máximo de D para considerar que uma sequência com tamanho 100 passou no teste com $\alpha = 0.01$ é $1,36/\sqrt{n}$, que resulta em 0,136. Observando os resultados, todas as 10 sequências testadas passariam como uma distribuição uniforme neste teste com valores-p consideravelmente altos, todos ficando consideravelmente acima do α. Sendo assim para estes testes seria aceito que a hipótese nula é verdadeira.

Aumentando o comprimento da sequência testada para 500 com o mesmo gerador, o limite superior de D é 0,0608 e todos menos um dos testes passaram pelo Valor-p como visto na Tabela 10 em n = 4, apesar de todos superarem o valor máximo teórico da estatística.

Tabela 8: Resultados parciais gerados pelos testes do NIST-STS com 100 sequências de 10000 bits geradas a partir de um GCL com parâmetros a = 1103515245, c = 12345 e $m = 2^{32}$.

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
9	12	9	14	10	5	7	9	12	13	0.637119	99/100	Frequency
8	9	10	13	11	9	8	11	13	8	0.946308	100/100	BlockFrequency
12	7	11	10	8	9	7	10	10	16	0.699313	100/100	CumulativeSums
12	13	7	10	9	7	8	11	7	16	0.514124	99/100	CumulativeSums
9	10	11	11	11	10	11	11	3	13	0.699313	99/100	Runs
7	12	13	10	13	10	7	19	6	3	0.028817	100/100	LongestRun
9	8	7	16	17	7	6	7	10	13	0.115387	100/100	Rank
10	15	8	8	13	11	7	15	5	8	0.304126	98/100	FFT
8	13	7	9	10	13	14	10	7	9	0.759756	97/100	NonOverlappingTemplate
12	8	6	12	8	8	7	14	10	15	0.474986	96/100	NonOverlappingTemplate
9	8	4	12	13	7	13	9	10	15	0.366918	96/100	NonOverlappingTemplate
8	11	6	16	10	8	13	9	6	13	0.383827	99/100	OverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	Universal
67	12	5	4	4	2	1	3	2	0	0.000000 *	75/100 *	ApproximateEntropy
7	7	12	15	13	8	7	10	9	12	0.595549	100/100	Serial
9	7	12	11	8	11	14	7	8	13	0.759756	99/100	Serial
8	3	4	7	16	9	13	11	10	19	0.007160	96/100	LinearComplexity

Fonte: Autor

4.6.2 **Testes para sequências binárias**

Para testes utilizando-se métodos para sequências binárias, a saída da sequência será convertida pela sua paridade, ou seja, módulo 2. A análise neste setor é, então, sobre a paridade da sequência gerada. Foi gerada uma sequência de 10^7 elementos, com p = 6047, entrada inicial 11^{15}, passo da entrada 1, limite de iterações 20. Utilizando os testes do NIST-STS com os parâmetros padrão [7], utilizando 100 blocos de 10000 bits para a testagem. Vistos na Tabela 11, os resultados dos testes mais simples já demonstram que apesar de possuir características de aleatoriedade, a não uniformidade dos Valores-p reprova a maioria dos resultados, reprovando este gerador como um PRNG criptograficamente seguro.

Como pode ser visto, por exemplo, no teste "Frequency", que apenas julga a proporção de 0s para 1s, em que, apesar de todo valor-p ser acima de 0.4, por não possuir nenhuma ocorrência nas categorias C1 a C4 acaba reprovando este gerador como satisfatoriamente aleatório, pela falta de uniformidade nos valores-p. Apesar de falhar num teste tão simples, acabou sendo aprovado em testes mais complexos como o de "Rank", que visa encontrar dependência linear entre subsequências da sequência original.

Testado então com $p = 2^{107} - 1$, um número primo de 33 dígitos, não foram obtidos resultados relevantemente diferentes nos testes de Kolmogorov-Smirnov e espectral do que

Fonte: Autor

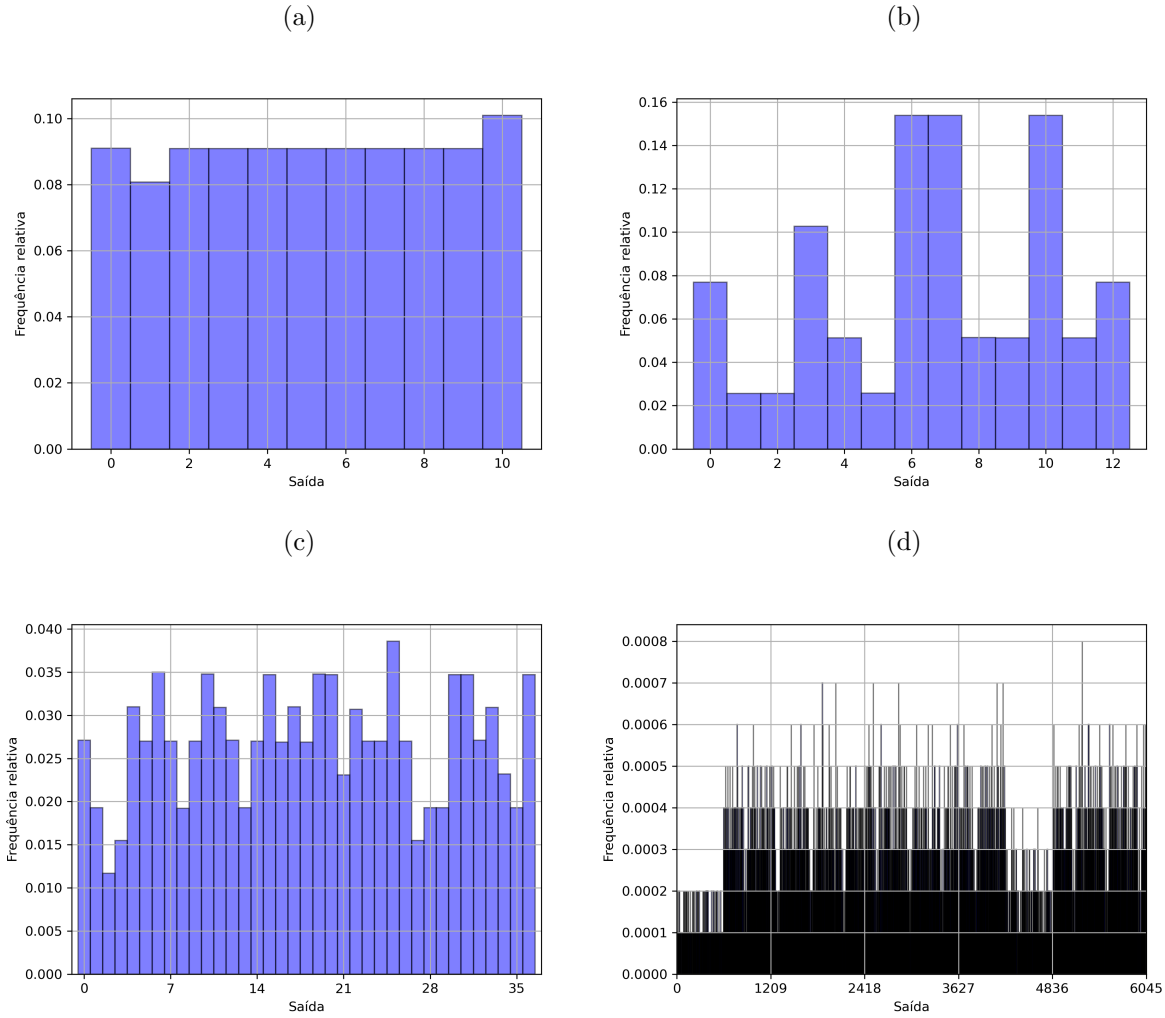


Figura 8: Histogramas para vários números primos diferentes com sequência de 10000 elementos, entrada sequencial iniciando em 10^{15}, S = 20: a)11. b)13. c)37. d)6047.

foi antes visto com p = 6047, mas nos testes do NIST-STS, como pode ser visto na Tabela 12, observam-se resultados significativamente melhores. Não só os resultados tendem a ser mais uniformes, populando melhor as categorias C1 a C10, mas também é aprovado em mais testes como os *Runs* e *LongestRun*.

Fonte: Autor

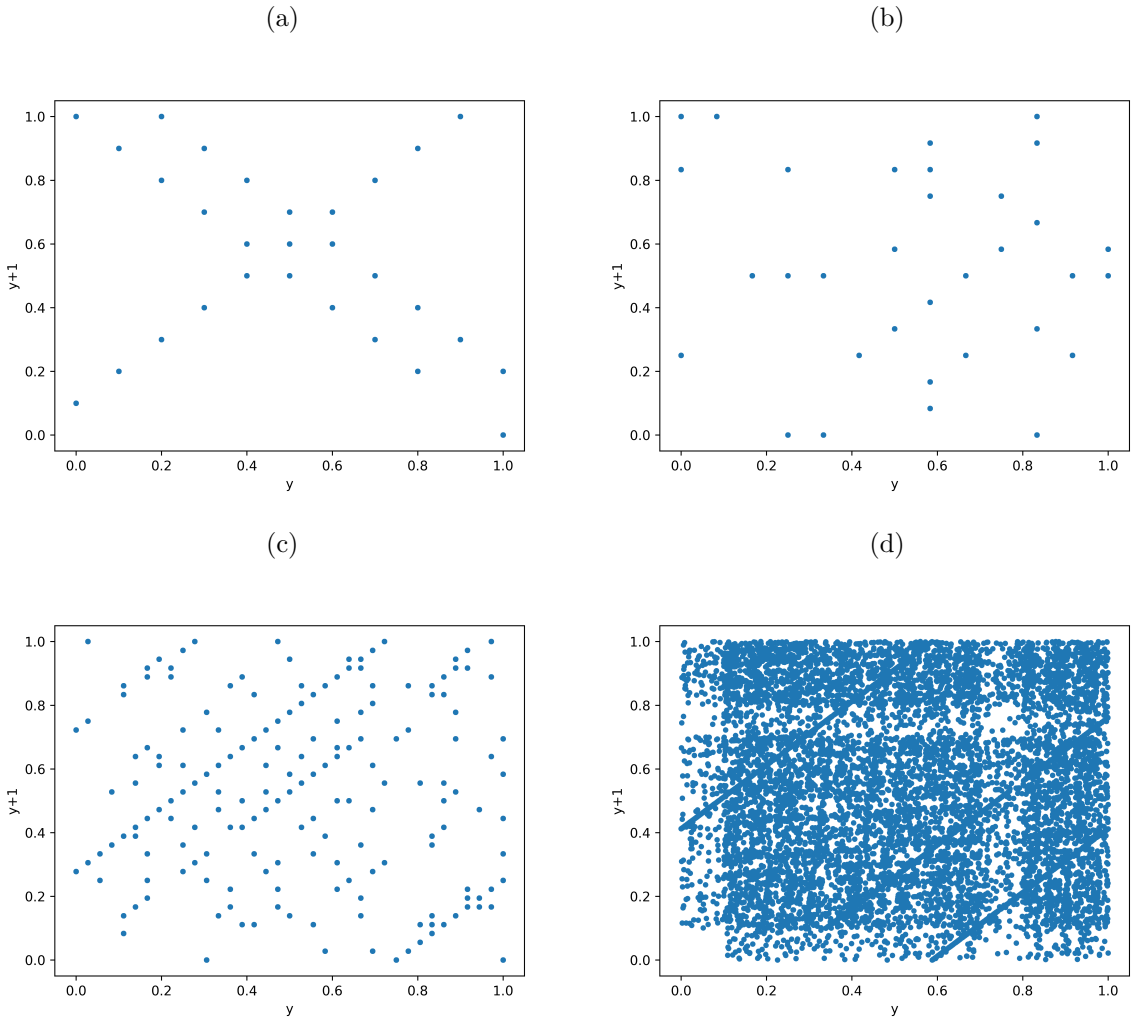


Figura 9: Teste espectral para vários números primos diferentes com sequência de 10000 elementos, entrada sequencial iniciando de 10^{15}, S = 20: a)11. b)13. c)37. d)6047.

Tabela 9: Resultados de testes de Kolmogorov-Smirnov comparando a saída de gerador p = 6047 a uma distribuição uniforme com sequências de 100 elementos.

n	**Estatística**	**Valor-p**
0	0.07880	0.5374
1	0.1030	0.2231
2	0.0753	0.5943
3	0.0937	0.3232
4	0.1008	0.2439
5	0.0766	0.5720
6	0.0645	0.7737
7	0.0934	0.3262
8	0.0934	0.3260
9	0.0705	0.6751

Fonte: Autor

Tabela 10: Resultados de testes de Kolmogorov-Smirnov comparando a saída de gerador p = 6047 a uma distribuição uniforme com sequências de 500 elementos.

n	**Estatística**	**Valor-p**
0	0.0702	0.0137
1	0.0626	0.0380
2	0.0620	0.0409
3	0.0669	0.0216
4	0.0733	0.0087
5	0.0717	0.0110
6	0.0707	0.0127
7	0.0645	0.0297
8	0.0631	0.0355
9	0.0642	0.0308

Fonte: Autor

Tabela 11: Resultados parciais gerados pelos testes do NIST-STS com 100 sequências de 10000 bits geradas a partir de um gerador com parâmetros p = 6047, entrada na magnitude 10^{15}, passo 1 e limite de iterações S = 20.

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0	0	0	0	2	10	17	17	20	34	0.000000 *	100/100	Frequency
0	0	0	0	0	0	0	0	0	100	0.000000 *	100/100	BlockFrequency
0	0	0	0	0	0	0	4	12	84	0.000000 *	100/100	CumulativeSums
0	0	0	0	0	0	0	1	14	85	0.000000 *	100/100	CumulativeSums
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	Runs
99	1	0	0	0	0	0	0	0	0	0.000000 *	4/100 *	LongestRun
10	7	3	20	19	12	5	8	7	9	0.000883	100/100	Rank
100	0	0	0	0	0	0	0	0	0	0.000000 *	4/100 *	FFT
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	NonOverlappingTemplate
8	5	13	9	10	9	9	13	11	13	0.739918	99/100	NonOverlappingTemplate
97	3	0	0	0	0	0	0	0	0	0.000000 *	9/100 *	NonOverlappingTemplate
36	24	12	10	6	6	4	2	0	0	0.000000 *	90/100 *	OverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	Universal
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	ApproximateEntropy
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	Serial
100	0	0	0	0	0	0	0	0	0	0.000000 *	10/100 *	Serial
7	4	8	9	10	17	8	12	11	14	0.191687	100/100	LinearComplexity

Fonte: Autor

Tabela 12: Resultados parciais gerados pelos testes do NIST-STS com 100 sequências de 10000 bits geradas a partir de um gerador com parâmetros p = $2^{107} - 1$, entrada na magnitude 10^{41}, passo 1 e limite de iterações S = 46.

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
1	1	5	12	10	12	11	18	15	15	0.000296	100/100	Frequency
0	0	0	0	0	0	0	3	2	95	0.000000 *	100/100	BlockFrequency
1	4	8	9	12	13	8	13	12	20	0.002758	100/100	CumulativeSums
0	2	5	11	11	16	9	20	7	19	0.000004 *	100/100	CumulativeSums
13	12	8	5	7	15	7	12	9	12	0.401199	98/100	Runs
11	16	13	11	10	10	7	9	7	6	0.514124	99/100	LongestRun
4	14	6	13	20	9	7	6	14	7	0.006661	99/100	Rank
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	FFT
47	15	10	8	3	8	4	1	2	2	0.000000 *	77/100 *	NonOverlappingTemplate
16	17	10	6	10	5	15	7	6	8	0.035174	97/100	NonOverlappingTemplate
11	9	7	8	6	14	11	8	13	13	0.637119	96/100	NonOverlappingTemplate
6	11	14	9	12	9	14	7	10	8	0.657933	100/100	OverlappingTemplate
100	0	0	0	0	0	0	0	0	0	0.000000 *	0/100 *	Universal
100	0	0	0	0	0	0	0	0	0	0.000000 *	1/100 *	ApproximateEntropy
52	21	9	11	3	2	2	0	0	0	0.000000 *	81/100 *	Serial
18	7	17	8	13	9	10	7	8	3	0.019188	98/100	Serial
12	3	12	5	15	13	8	10	13	9	0.162606	94/100 *	LinearComplexity

Fonte: Autor

5 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi apresentado um novo tipo de gerador de números pseudoaleatório a partir de um processo completamente determinístico, baseado em testes de divisibilidade. Comparado a um GCL nos testes mais analógicos, os GTDs aqui apresentados não parecem particularmente impressionantes nos testes visados para geradores contínuos, mas nos testes como um gerador de números binários conseguiu apresentar resultados interessantes no NIST-STS. Principalmente os resultados vistos na Tabela 12 demonstram que esta classe de geradores possui potencial.

Para trabalhos futuros, é necessário entender melhor a escolha do número primo para desenvolver o gerador. Possivelmente também entender se há alguma regra equivalente ao teorema de Hull-Dobel para GCLs para determinar alguma relação entre número primo, limite de iterações e valores de entrada ótimos para criar um bom gerador a partir deste processo. O processo também aparenta ter um mecanismo interno de *trapdoor*, ou seja, uma transformação irreversível, que poderia ser utilizado para hashing de dados, por exemplo. Uma possível alteração interessante também pode ser mudar a base dos testes para algo que o processador empregado utiliza otimamente nas operações de multiplicação.

6 APÊNDICE A - CÓDIGO

O código utilizado pode ser encontrado clicando neste link ou no site `https://github.com/vagner995/Gerador-teste-divisibilidade`.

REFERÊNCIAS

[1] Correio Braziliense. *Menino de 12 anos descobre fórmula matemática que ajuda o estudo da divisão.* Available at: <https://www.correiobraziliense.com.br/app/noticia/mundo/2019/11/19/interna_mundo,807535/menino-de-12-anos-descobre-formula-matematica-que-ajuda-o-estudo-da-di.shtml>. Accessed in: 09/03/2022.

[2] BURTON, D. *Teoria elementar dos números* . [S.l.]: Grupo Gen-LTC, 2016.

[3] do Couto, V. T.; da Silva Jr, G. J.; Campello de Souza, R. M. Construindo testes de divisibilidade para inteiros representados numa base qualquer. *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, v. 9, n. 1, 2022.

[4] HUBBARD, R.; BAYARRI, M. P values are not error probabilities. *Institute of Statistics and Decision Sciences, Working Paper*, n. 03-26, p. 27708–0251, 2003.

[5] FISHER, R. A. et al. The design of experiments. *The design of experiments.*, Oliver and Boyd. London and Edinburgh, n. 7th Ed, 1960.

[6] NEYMAN, J.; PEARSON, E. S. Ix. on the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, The Royal Society London, v. 231, n. 694-706, p. 289–337, 1933.

[7] SýS, M. et al. On the interpretation of results from the NIST statistical test suite. v. 18, p. 18–32, 01 2015.

[8] KNUTH, D. *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Pearson Education, 2014. ISBN 9780321635761. Available at: <https://books.google.com.br/books?id=Zu-HAwAAQBAJ>.

[9] AN, K. Sulla determinazione empirica di una legge didistribuzione. *Giorn Dell'inst Ital Degli Att*, v. 4, p. 89–91, 1933.

[10] DODGE, Y. *The Concise Encyclopedia of Statistics*. Springer New York, 2008. (The Concise Encyclopedia of Statistics). ISBN 9780387317427. Available at: <https://books.google.com.br/books?id=k2zklGOBRDwC>.

[11] SILVEY, S. *Statistical Inference*. Taylor & Francis, 1975. (Chapman & Hall/CRC Monographs on Statistics & Applied Probability). ISBN 9780412138201. Available at: <https://books.google.com.br/books?id=qIKLejbVMf4C>.