Universidade Federal do Amazonas
Instituto de Computação
PPGI - Programa de Pós-Graduação em Informática

Um método de encriptação simétrica baseada em caos

Angelo de Oliveira

Manaus (AM), Julho de 2019

Angelo de Oliveira

Um método de encriptação simétrica baseado em caos

Dissertação apresentada ao Programa de Pós-Graduação em Informática (PPGI), do Instituto de Computação da Universidade Federal do Amazonas (ICOMP), como pré-requisito parcial para a obtenção do título de mestre em Informática.

Orientador: Prof. Dr. Eduardo Luzeiro Feitosa.

Manaus (AM), Julho de 2019

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

O48u

Oliveira, Angelo de
Um método de encriptação simétrica baseado em caos / Angelo de Oliveira. 2019
143 f.: il. color; 31 cm.

Orientador: Eduardo Luzeiro Feitosa
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Criptografia baseada em caos. 2. Aleatoriedade. 3. Jogo do caos. 4. One-time pad. I. Feitosa, Eduardo Luzeiro II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Um Método de Encriptação Simétrica baseada em Caos"

ANGELO DE OLIVEIRA

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Eduardo Luzeiro Feitosa - PRESIDENTE

Prof. Eduardo James Pereira Souto - MEMBRO INTERNO

Prof. Mário Salvatierra Júnior - MEMBRO EXTERNO

Manaus, 29 de Agosto de 2019

DEDICATÓRIA

PUXI ZTTD GQOL FLIK XPAM QYKA ZJPR
DJSY JNZX AJTQ QNOX KBHE QDMF JXPE
ZPED DOBB JPGZ AWGH IRVK NIQQ SPEB
HYWI PXUD VPRB YLWH KYJC SXLK YVSH
BBFM LUDT VUGE KFEE HRUJ VRXC JUSY
VWDC OHIA RKFT RGXG YPWI BPBR MAUA
BTJL

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Eduardo Luzeiro Feitosa, por me aceitar como orientando e por me dar liberdade e apoio no processo de escrita do presente trabalho.

Aos meus pais, pelo amor incondicional, apoio, torcida, e principalmente, pelo exemplo de vida.

Aos meus dois queridos irmãos, Marcus e Eduardo, por compartilhar ideias e pela sempre boa conversa.

Aos meus queridos filhos, Marina e Polaco, por alegrarem minha vida.

Ao Diego (in memoriam), Fofinha (in memoriam), Johnny (in memorian), Safira, Prince, Robert, Feioso, Valquíria, Frederico (in memoriam), James, Margareth (Meg), Henry, Dafne, Jorge, Joaquim (gato preto), Joana, Abgail, Pretinho, Floquinha, João-zinho, e agora a Lindinha, pela companhia e amizade.

Por fim, mas não menos, a minha Prof^{a}. Dilcélia Heckmann Barbalho, por ter me dado a edição 80 da revista Ciência Hoje, cujo conteúdo era voltado inteiramente para a ciência do caos.

Ich sage euch:
man muss noch Chaos in sich haben,
um einen tanzenden Stern gebären zu können.
Ich sage euch:
ihr habt noch Chaos in euch.

Friedrich Nietzsche, *Also sprach Zarathustra*

RESUMO

Historicamente, one-time pad é a única cifra que pode ser matematicamente provada ser inquebrável. Contanto forneça o mais alto nível de segurança caso adequadamente utilizada, one-time pad tem algumas desvantagens que inibem o seu amplo uso, sendo a mais proeminente a distribuição de sequências (chaves) verdadeiramente aleatórias que devem possuir no mínimo o mesmo tamanho da mensagem a ser encriptada e que devem ser utilizadas uma única vez (daí a alcunha one-time). Para contravir esta dificuldade pode-se a princípio fazer uso de sequências pseudo-aleatórias (que são passíveis de reprodução). Ora, para se fornecer tais sequências, pode-se recorrer a funções caóticas de modo em que não se faça necessário a transmissão a priori da sequência (chave) inteira antes do processo de comunicação ter início, sendo necessária somente poucas informações e/ou parâmetros, tais como: mapa caótico a ser utilizado e valor inicial. Desta feita, o presente trabalho tem como proposta uma nova cifra de fluxo simétrico baseado em funções caóticas (unidimensionais) e no jogo do caos. A fim de verificar se a metologia ora empregada é adequada para uso criptográfico, várias suítes de testes estatísticos são empregadas dado que criptoanálise será deixada para outro momento.

Palavras-chave: Criptografia baseada em caos; aleatoriedade; jogo do caos, one-time pad.

ABSTRACT

Historically, one-time pad is the only cipher that can be mathematically proven to be unbreakable. However provide the highest level of security if properly used, one-time pad has some drawbacks that inhibit its widespread use, the most prominent being the distribution of truly random sequences (keys) that should be at least as long as the message to be encrypted and should be used only once (hence the origin the nickname one-time). To counteract this difficulty, one can at first make use of pseudo-random sequences (which are reproducible). In order to provide such sequences, chaotic functions may be used, so that a priori transmission of the entire (key) sequence is not necessary before the communication process begins, requiring only little information and/or parameters such as: chaotic map to be used and initial value. Thus, the present work proposes a new symmetric stream cipher based on chaotic functions (one-dimensional) and the game of chaos. In order to verify if the methodology employed is suitable for cryptographic use, several statistical test suites are employed since cryptanalysis will be left for another moment.

Keywords: Chaos based cryptography; randomness; chaos game; one-time pad.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

1.1 Problema, Objetivos e Contribuições

1.1.1 Problema

Nos idos dos anos 40 do século passado, Claude E. Shannon mostrou que a cifra *one-time pad* (OTP), proposta no começo do século XX[1], era o único sistema (até hoje conhecido) que pode prover o que se define como sigilo perfeito. Aqui não cabe (e nem será dada) uma discussão pormenorizada sobre este sistema; contudo, de antemão, pode-se dizer o mesmo é simples e utiliza sequências verdadeiramente aleatórias de caracteres (ou números, hodiernamente), os quais são pareados com o texto plano, sendo este encriptado com a sequência por meio de adição modular.

Para que OTP seja incapaz de ser decifrada/quebrada, os seguintes quesitos devem ser cumpridos:

1. A chave deve ser *verdadeiramente aleatória*.
2. A chave deve ter no *mínimo* o comprimento do texto plano.
3. A chave nunca deve ser *reutilizada* em todo ou em parte.
4. A chave deve ser mantida completamente em segredo.

Observe que os requisitos de OTP referem-se única e exclusivamente às chaves. Isso não é de modo algum inesperado, pois dentre os vários problemas encontrados ao se utilizar sistemas criptográficos, pode-se destacar:

- **Problema de distribuição de chave:** como distribuir com segurança as *chaves* entre os usuários de um sistema criptográfico.

[1] One-time pad foi primeiramente descrita por Frank Miller (Miller, 1882) e redescoberta/reinventada independentemente por Gilbert S. Vernam (Vernam, 1919). Para uma discussão a respeito da origem de OTP, ver (Bellovin, 2011).

- **Problema de manutenção:** Como gerenciar com segurança todas as *chaves* distribuídas de modo a assegurar que os objetivos são alcançados.
- **Problema do protocolo criptográfico:** Como trocar informação de modo seguro usando o sistema criptográfico e o meio disponível.
- **Problema do agendamento da chave:** Como agendar o uso, reuso e eliminação de *chaves*.
- **Problema da geração/seleção da chave:** Como gerar de maneira confiável, *chaves* difíceis de adivinhar.

Mais uma vez, observa-se que o principal fator que pragmaticamente se faz presente é a questão de como lidar com as chaves. Isso é predominante porque, conforme o segundo princípio de Kerckhoffs (Kerckhoffs, 1883, p. 12), tudo é presumido ser de conhecimento do *atacante*, com exceção das chaves (ou ou nas palavras de Shannon (Shannon, 1949, p. 662): "*the enemy knows the system being used*"). Assim, em qualquer sistema criptográfico, o principal problema jaz em como lidar com chaves.

Por sua vez, em contraponto a oferta da maior segurança possível (se adequadamente utilizada), one-time pad apresenta vários inconvenientes, dentre os quais:

- Números verdadeiramente aleatórios não podem ser gerados por algoritmos. Assim, computadores não podem gerar sequências de números verdadeiramente aleatórios, devendo a geração de chaves ser entregue a um TRNG (True Random Number Generator), os quais se constituem em dispositivos especializados para tal faina e geralmente externos ao sistema criptográfico.
- A troca segura das chaves, que devem ser no mínimo do mesmo tamanho da mensagem, é um importuno.
- One-time pad não fornece nenhum meio de autenticação.

Alguns destes estorvos no uso de OTP não se constituem como tal, por exemplo, a geração de chaves verdadeiramente aleatórias é um item externo a OTP. Por seu turno, o gerenciamento das chaves, de modo que nenhuma delas seja reutilizada em todo ou em parte, faz parte do protocolo criptográfico, ou seja, o modo de utilizar o sistema, bem como o é a correta manutenção *em segredo* das chaves e a questão de autenticação.

Entretanto, o segundo item é considerado o maior empecilho ao uso de OTP. Dado que OTP é uma cifra de encriptação simétrica, a mesma chave é utilizada tanto para encriptação quanto para decriptação, e a partilha antecipada de longas chaves pode

ser inseguro.

Uma questão pertinente é: se OTP possui tantas desvantagens e considerando que outras cifras oferecem melhor gerenciamento e manuseabilidade, bem como algumas podem prover autenticação, por que usar OTP? A principal razão jaz no fato, já anteriormente citado, que OTP provê o meio mais seguro de troca de comunicação (isso em termos estritamente matemáticos). Aliás, OTP é tão segura que nem mesmo o advento de computadores quânticos podem quebrá-la (ver (Bennett et al., 2014)).

Assim, considerando a inerente segurança ofertada por OTP, não é surpreendente que uma miríade de métodos tem sido propostos com o objetivo de retrucar adequadamente as desvantagens de OTP[2].

Dado que a principal força de OTP também é o maior empecilho em sua utilização, a saber, o uso de (longas) sequências verdadeiramente aleatórias, muito do que se tem feito tem como escopo a utilização de sequências *pseudo-aleatórias* que são passíveis de reprodução, bem como sua análise[3]. Por exemplo, (Bright e Enison, 1979), (Blum et al., 1986), (Wolfram, 1986), (Meier e Stdelbach, 1991) (Crounse et al., 1996), (Matsumoto e Nishimura, 1998a), (Tomassini et al., 1999), (Soto, 1999), (Shujuna et al., 2001), (Röck, 2005), (Pareschi, 2006), (Röck, 2009), (Guyeux et al., 2010), (Luizi et al., 2010), (Hu et al., 2013), (Niu et al., 2014) e (Raza e Satpute, 2018).

Existe uma pletora de métodos para a geração de números pseudo-aleatórios, e embora estes métodos possam ser objeto de pesquisa, esse não será o objetivo desta dissertação. Convém destacar, porém, que as iterações de funções caóticas podem (em principio) fornecer sequências de tais números, e como tal, bem servir para esquemas criptográficos.

Embora neste ponto não seja dado uma explicação mais amiúde do que é *chaos* e como o mesmo se manifesta (isso é deixado para o Capítulo 3), aqui será dado um vislumbre sobre funções caóticas e como sequências binárias pseudo-aleatórias podem ser geradas a partir das mesmas. Para tanto, considere o mapa logístico[4]: $x_{n+1} = r.x_n.(1 - x_n)$. Ora, as sucessivas iterações do mapa logístico a partir de um valor inicial x_0 tem comportamentos diferentes conforme os valores do parâmetro r (Ver as Figuras 1.1 a 1.4).

[2]Mas nem tudo é assim. Em (Rubin, 1996) é aventado como utilizar OTP de modo efetivo.

[3]A propósito, a ordem cronológica apresentada não é arbitrária, mas intencional, e tem como fito mostrar que a geração de sequências pseudo-aleatórias foi e continua sendo um tema de interesse científico.

[4]A propósito: o método proposto neste trabalha utiliza este mapeamento.

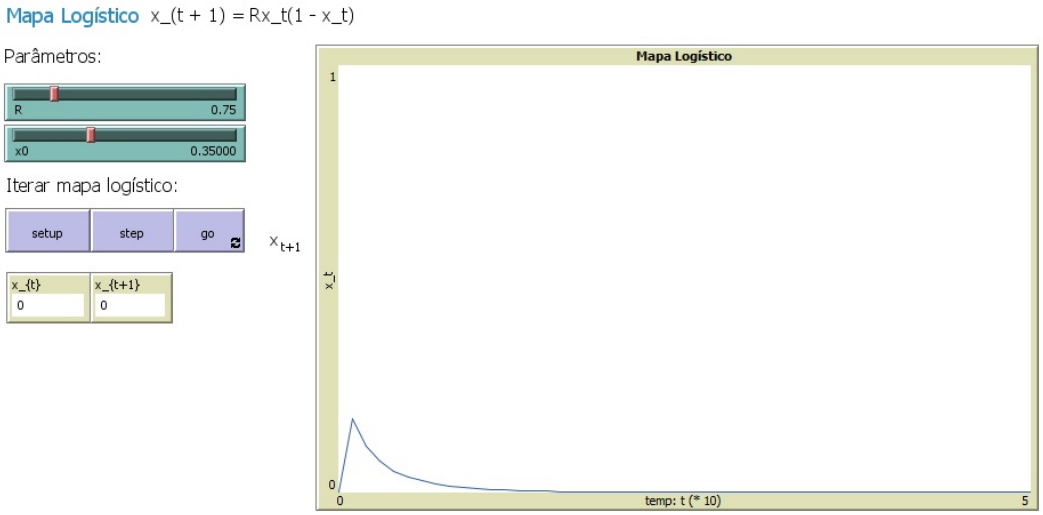


Figura 1.1: Iterações do mapa logístic: $r = 0.75$, $x_0 = 0.35$.

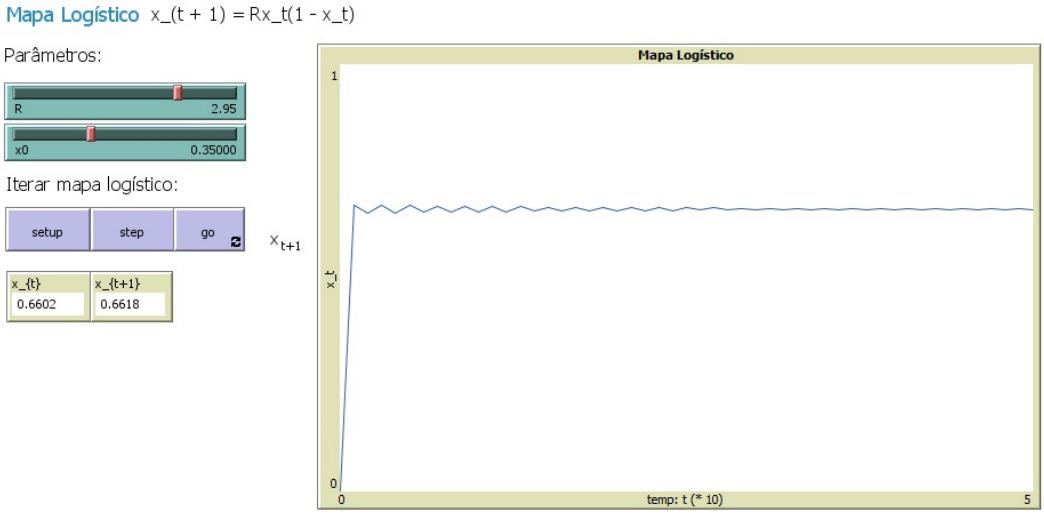


Figura 1.2: Iterações do mapa logístic: $r = 2.95$, $x_0 = 0.35$.

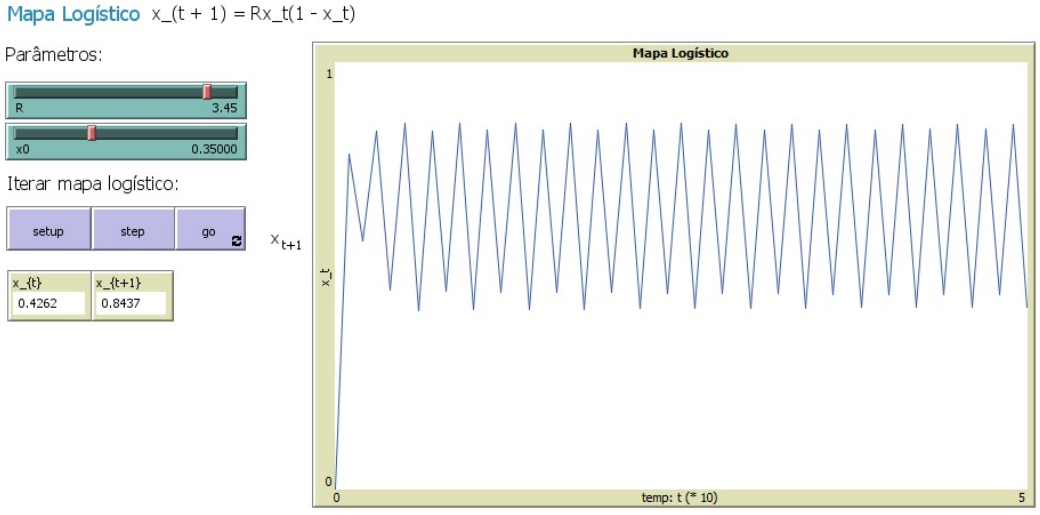


Figura 1.3: Iterações do mapa logístic: $r = 3.45$, $x_0 = 0.35$.

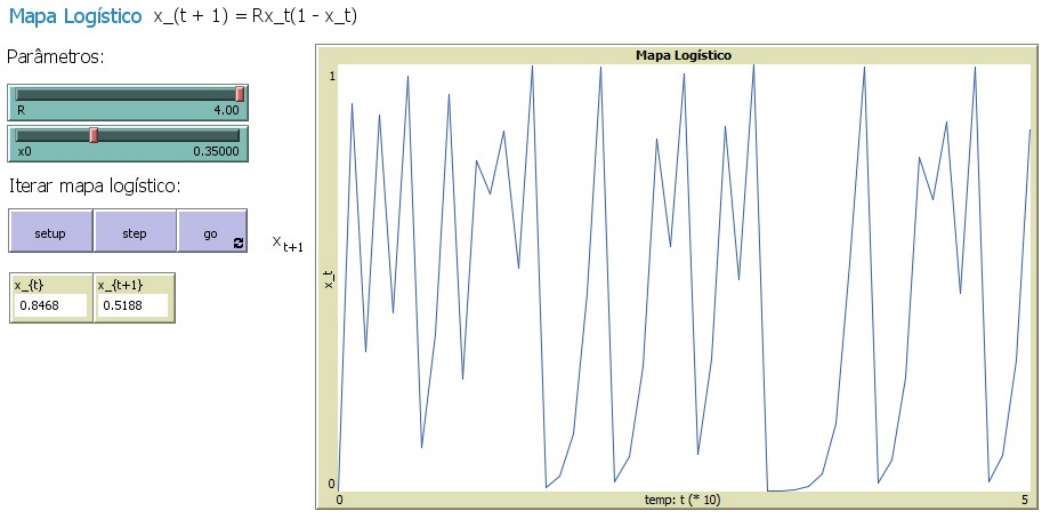


Figura 1.4: Iterações do mapa logístic: $r = 4$, $x_0 = 0.35$.

O que as figuras mostram é que conforme o valor do parâmetro r, o comportamento de longo prazo da iterações pode tender a um ponto fixo (Figuras 1.1 e 1.2), pode ter um comportamento periódico (Figura 1.3) ou aperiódico (Figura 1.4). Ademais, o mapa logístico apresenta o que se convenciona chamar de *dependência sensível sobre as condições iniciais* (Figura 1.5). O que isto quer dizer? Bem, significa que valores iniciais muito próximos quando iterados afastam-se assintoticamente, divergindo de forma exponencial.

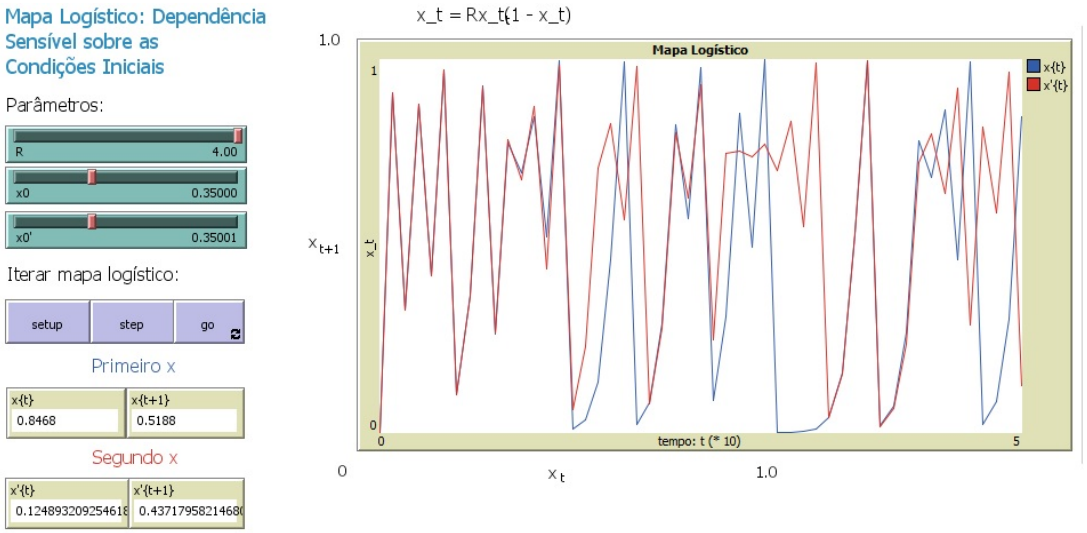


Figura 1.5: Iterações do mapa logístic: $r = 4$, $x_0 = 0.35000$, $x_0' = 0.35001$.

Agora é chegada a hora de dizer como essas iterações podem produzir sequências de números pseudo-aleatórios. Para tanto, faça uso da assim chamada *dinâmica simbólica*. Nesta, um sistema dinâmico é modelado por um espaço discreto consistindo de infinitas sequências de símbolos abstratos, cada um dos quais correspondendo a um estado do sistema. Mas para o presente caso, os símbolos adotados serão 0 e 1. Use a seguinte convenção: sempre que o valor da iteração for maior ou igual a $\frac{1}{2}$, associe 1;

caso contrário, associe 0[5]. Assim procedendo, as sucessivas iterações do mapa logístico produzem sequências de dígitos binários. Considerando que o mapa logístico é capaz de comportamento aperiódico e que apresenta dependência sensível sobre as condições iniciais, teoricamente o número de sequências binárias passíveis de serem produzidas é infinito. Mas tão importante quanto isto, a densidade de informação da dinâmica simbólica assim constituída possui entropia 1 (Figura 1.6).

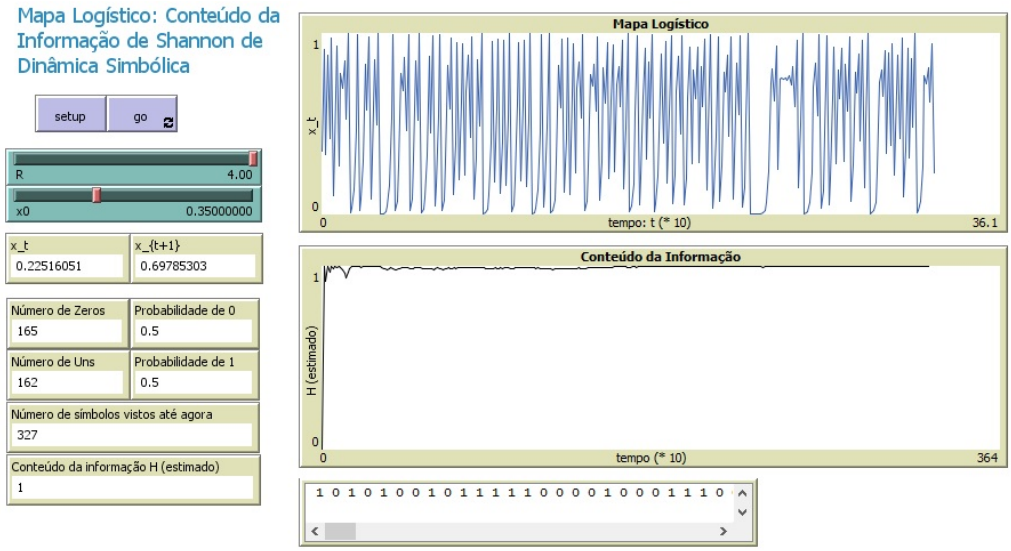


Figura 1.6: Mapa logístico: dinâmica simbólica e entropia.

Agora, volvendo-se para novamente para OTP, que tal a ideia de utilizar funções caóticas para a produção de sequências pseudo-aleatórias em lugar das verdadeiras? Seria isto de alguma feita satisfatório? Antes de se emitir um juízo, é mister ponderar quais seriam as vantagens e desvantagens.

Destarte, dentre as vantagens, tem-se:

- As sequências (chaves) não necessitariam ser encaminhadas em sua totalidade entre as partes para que o processo de comunicação seja iniciado, sendo necessário tão somente o conhecimento da(s) função(ões) a serem utilizadas e o valor inicial, isso porque a parte que recepcionará o criptograma construirá uma sequência equivalente a fim de decriptar a mensagem. Isso no mínimo soa prático.
- Idealmente, sequências de tamanho ilimitado são suscetíveis de construção.
- Teoricamente, o número possível de valores iniciais é infinito. Ora, um número infinito de chaves permite, com o devido gerenciamento, que nenhuma chave seja reutilizada.

[5] As iterações do mapa logístico estão limitadas ao intervalo $[0, 1]$.

Observe que a primeira vantagem supra mencionada soluciona em quase sua totalidade o maior empecilho à utilização de OTP. Já a segunda, satisfaz ao segundo requisito para o correto uso de OTP. Por derradeiro, a terceira vantagem atende ao terceiro quesito para o preciso emprego de OTP.

Mas e quanto as desvantagens? Elas se contrapõem em relação as vantagens:

- Como um item primaz e crucial para que a segurança de OTP seja alcançada, deve-se utilizar sequências verdadeiramente aleatórias.
- Pragmaticamente não se pode construir sequências de tamanho ilimitado.
- A aritmética computacional, ou mais especificamente, o modo como números reais são representados em computadores, é restrito.

A vista disso, considere:

- Para a primeira desvantagem, tome e conta que formalmente, ou seja, dentro do arcabouço matemático existente, não se é possível provar que uma determinada sequência é aleatória, quer seja ela proveniente de uma fonte entrópica ou não. Quando muito, pode-se mostrar que ela não é aleatória, mas definitivamente a prova afirmativa é impossível de ser alcançada. Assim, sequências pseudo-aleatórias são consideradas aleatórias até prova em contrário, desde que, é claro, tenham as mesmas propriedades esperadas para sequências verdadeiramente aleatórias.
- A segunda desvantagem é verdadeira tanto para sequências verdadeiramente aleatórias quanto para as pseudo-aleatórias. Assim, não se constitui em uma verdadeira desvantagem.
- A terceira desvantagem é impossível de ser rechaçada (quando muito, pode-se aumentar a magnitude da representação), significando que o número efetivo de valores iniciais é limitado, muito embora amplo[6].

Agora, considerando o exposto nos parágrafos precedentes, pode-se estatuir o problema a ser perseguido neste trabalho.

[6]Por exemplo, existem 36857459350400139265 números ponto flutuante passíveis de representação no formato de precisão dupla no padrão IEEE.

Este trabalho tem como objetivo aventar um método que procure minorar o problema mais proeminente na utilização da cifra one-time pad, a saber, o *problema de distribuição de chaves*. Para tanto, será proposta uma *nova* cifra de encriptação simétrica que fará uso de funções caóticas unidimensionais e do jogo do caos.

Muito embora aqui, neste ponto, pudesse ser dado um esboço do método proposto, tal descrição teria como desfecho a necessidade de esmiuçar aspectos teóricos e/ou pragmáticos que demandariam inúmeras páginas, o que certamente aumentaria em demasia o presente capítulo; portanto, isso é deixado para o capítulo 4. Contudo, antecipadamente, pode-se afirmar que técnica sugerida será composta por duas etapas nas quais serão gerados números pseudo-aleatórios, sendo que para tanto, serão empregadas funções caóticas unidimensionais (1ª etapa) e o *jogo do caos* (2ª etapa).

Antes de prosseguir para a próxima subseção, um adendo na forma de pergunta/resposta. Em que aspecto a ideia apresentada é superior/diferente ao que já foi proposto? Para esta inquirição, pode-se argumentar que a resposta demanda que tudo (ou quase) o que foi apresentado deveria ser objeto de comparação (por meio de implementação e testes), o que evidentemente é impraticável considerando escopo limitado. Ademais, em termos teóricos, somente o uso de sequências verdadeiramente aleatórias provê as qualidades desejadas para one-time pad. Por fim, o autor não encontrou na literatura nenhum método de encriptação/decriptação que utilize o jogo do caos para a geração de sequências binárias pseudo-aleatórias, muito embora tenha detectado um único trabalho (ver (P. e S., 2011)) que utiliza o triângulo de Sierpinsk como método de encriptação; como o exemplo mais conhecido do jogo do caos produz algo símile ao triângulo de Sierpinsk, isso remeteu à ideia neste trabalho sugestionada, mas é completamente distinta.

1.1.2 Objetivos

1.1.2.1 Objetivo Geral

Construir uma cifra de encriptação simétrica baseada em funções caóticas unidimensionais e no jogo do caos adequada para utilização em criptografia.

1.1.2.2 Objetivos Específicos

- Verificar por meio de testes estatísticos se as sequências de números *pseudo*-aleatórios produzidas possuem propriedades adequadas para uso criptográfico.

- Verificar que texto plano e texto cifrado diferem em natureza, asseverando se o texto cifrado apresenta-se tão aleatório quanto possível.

1.1.3 Contribuições

A presente dissertação apresenta as seguintes contribuições:

- Análise de sistemas dinâmicos, em particular, suas propriedades matemáticas, que podem ser empregadas no projeto de sistemas criptográficos.
- Produção de sequências pseudo-aleatórias usando funções caóticas, verificando por meio de testes estatísticos se as mesmas apresentam-se suficientemente aleatórias para uso em criptografia[7].
- Uma nova proposta de cifra baseada em funções caóticas e no "jogo do caos".

1.2 Estrutura do Trabalho

O presente trabalho é estruturado segundo os capítulos:

- O capítulo 1 apresenta de forma resumida o assunto e o problema a ser perseguido.
- O capítulo 2 introduz conceitos relacionados à ciência da criptografia, aleatoriedade e criptografia baseada em caos que são basilares para o entendimento do restante da dissertação.
- O capítulo 3 introduz conceitos relacionados à ciência do caos, em particular, discorre sobre sistemas dinâmicos e suas propriedades.
- O capítulo 4 descreve o método proposto de forma detalhada, discutindo quais são os pressupostos teóricos assumidos em cada etapa da implementação, finalizando com uma descrição dos experimentos de ordem estatística a serem realizados, bem como uma descrição das suítes de testes utilizadas.
- O capítulo 5 expõe o funcionamento do programa protótipo e o protocolo experimental, seguido pelos resultados dos testes estatísticos realizados.
- O capítulo 6 apresenta a conclusão geral do presente dissertação e sugere outras direções a serem perseguidas em outros trabalhos.

[7] Sequências aleatórias (e pseudo-aleatórias) possuem uma ampla gama de aplicações que não a criptografia, por exemplo, jogos de computador (para tentar dar mais realismo), amostragem estatística, simulações computacionais de fenômenos físicos, etc. Assim, as sequências geradas neste lavor bem podem, auspiciosamente, servirem para outros tipos de aplicações.

Capítulo 2

Criptografia, Aleatoriedade e Caos

O presente Capítulo tem por objetivo introduzir conceitos relacionados à ciência da criptografia, aleatoriedade e criptografia baseada em caos. Considerando que o escopo dos assuntos aqui abordados são "abrangentes", o tratamento dado aqui é de caráter elementar. Entretanto, ao menos para os dois primeiros tópicos, existe farta e extensa literatura, a qual o leitor mais interessado poderá encontrar material com escopo mais abrangente.

2.1 Criptografia

Como é comum a muitos termos científicos, o conceito de *criptografia* varia conforme os autores, considerando, é claro, que cada um possui um viés que os dirige e norteia. Não sendo desejável (e talvez nem mesmo possível) mencionar todas as possíveis derivações para o termo, e sendo premente que se tenha um ponto de partida, a definição seguinte tenciona ser este marco inicial.

Definição 2.1.1. *Criptografia é o estudo de técnicas matemáticas relacionadas a aspectos da segurança da informação, tais como confidencialidade, integridade de dados, autenticação de entidade e autenticação da origem dos dados. (Menezes et al., 1996, p. 4)*

A definição 2.1.1 sem dúvida é adequada para os muitos papéis encenados pela criptografia, sendo realmente uma declaração abrangente e adequada, ligando a mesma à segurança da informação, mas certamente deixa de lado a origem do termo.

Etimologicamente, o vocábulo provém do grego *κρυπτός* (kryptós = escondido) + *γράφειν* (gráphein = escrita), literalmente, escrita escondida, o que permitiria considerar que a expressão poderia significar o estudo de métodos para o envio de mensagens em segredo (que em tempos idos seriam restritos unicamente à comunicação escrita em

meio estritamente físico).

Dada a amplitude de possíveis significados, em lugar de se tentar definir "criptografia", é melhor é explicar um típico processo criptográfico (Figura 2.1) envolvendo três atores: Alice, Bob e Eva.

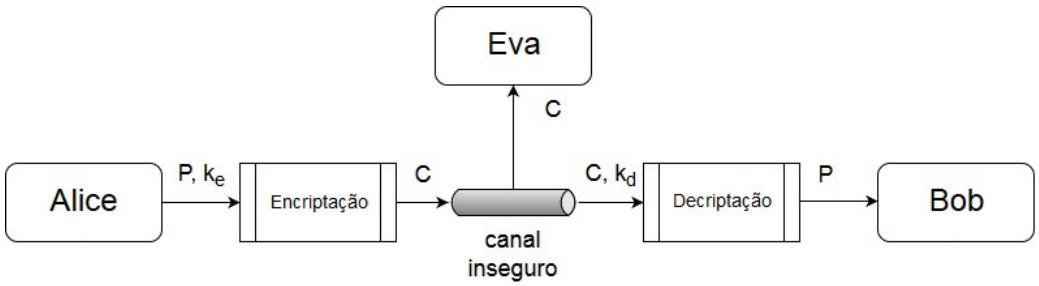


Figura 2.1: Típico processo de comunicação secreta utilizando criptografia.

Alice e Bob são dois indivíduos que desejam comunicar-se em segredo, e para tanto, empregam técnicas que permitem que a comunicação (mais especificamente o conteúdo) seja oculto para qualquer outrem. Por sua vez, Eva tem interesse em conhecer qual é o teor das mensagens partilhadas por Alice e Bob. Com o intuito de garantir o sigilo sobre o conteúdo de uma *mensagem*[1] (genericamente denominada *texto plano*), Alice utiliza um procedimento que embaralha a mensagem encaminhada para Bob, de modo que a interceptação da mesma por parte de Eva não tenha efeito. O processo utilizado por Alice chama-se *encriptação*, sendo produzido com base em uma *chave*, resultando no que se chama de *criptograma* ou *texto cifrado*.

Na outra ponta da linha, Bob deve coletar a mensagem enviada por Alice (o criptograma ou texto cifrado) e utilizar um método, genericamente chamado de *decriptação*, para transformá-lo novamente no texto plano, utilizando para tanto uma chave que, dependendo do método utilizado, pode tanto ser *igual* à utilizada originalmente por Alice ou uma *diferente*. O mundo de Alice e Bob prescindiria destas operações se não fosse a existência de Eva, cujo intento é descobrir o segredo partilhado entre eles. A ação tomada por Eva para revelar o conteúdo compartilhado por Alice e Bob é chamado de *criptoanálise*.

Detalhando o processo ilustrado na Figura 2.1 tem-se:

- Alice encripta a mensagem a ser encaminhada para Bob, fornecendo os seguintes argumentos para o processo (função) de encriptação E: o *texto plano* $\mathcal{P}$ e a *chave de encriptação* k_e, resultando no *criptograma* (*texto cifrado*) $\mathcal{C}$, o qual é

[1] O significado do termo mensagem aqui deve ser tomado como sendo informação não encriptada.

encaminhado a Bob por meio de um canal inseguro (que está sujeito a "escuta" de terceiros).

- Bob decripta a mensagem encaminhada por Alice, fornecendo os seguintes argumentos para o processo (função) de decriptação D: o *texto cifrado* $\mathcal{C}$ (recebido de Alice) e a *chave de decriptação* k_d, produzindo como resultado o texto plano $\mathcal{P}$.

- A tarefa de Eva consiste em interceptar o *texto cifrado* $\mathcal{C}$ enviado pelo canal inseguro e tentar reconstruir o *texto plano* original $\mathcal{P}$, ou alternativamente, descobrir a chave de decriptação k_d a partir do *texto cifrado* $\mathcal{C}$.

O modelo apresentado na Figura 2.1 pode ser chamado, genericamente, de *modelo padrão de comunicação criptográfica*, podendo ser simbolicamente descrito pelas seguintes equações:

$$\begin{cases} A : E(\mathcal{P}, k_e) = \mathcal{C} \\ B : D(\mathcal{C}, k_d) = \mathcal{P} \end{cases} \tag{2.1}$$

Nas equações, os componentes $\mathcal{C}$, E e D são de conhecimento público, ao passo que $\mathcal{P}$, k_e e k_d são secretos[2]. Desta feita, a ocultação da informação está baseada no sigilo de k_e e k_d e nas propriedades de E e D.

A fim de que o sistema possibilite a comunicação segura entre as partes envolvidas (Alice e Bob), certos requisitos devem ser cumpridos, dentre os quais se pode destacar:

- O número de possíveis diferentes chaves deve ser grande o suficiente, de modo a assegurar que uma pesquisa exaustiva (*ataque de força bruta*) por parte de Eva não é verossímel.

- O número de diferentes textos cifrados gerados pela combinação de um texto plano com diferentes chaves de encriptação deve ser grande o suficiente para assegurar que $\mathcal{P}$ depende em alto grau de chave de encriptação k_e utilizada.

- A derivação das chaves de encriptação (k_e)/decriptação (k_d) e do texto plano ($\mathcal{P}$) a partir do texto cifrado $\mathcal{C}$ e das funções de encriptação (E)/decriptação (D) deve ser muito difícil.

Os processos que se incumbem das tarefas de encriptação e decriptação são chamados de *sistemas criptográficos*.

Por sua vez, o modelo representado simbolicamente pelas equações 2.1 (e pictoricamente pela Figura 2.1), é genérico o suficiente para representar diferentes aplicações,

[2]Como será visto adiante, existem modelos nos quais uma das chaves podem ser tornada pública.

mas é insuficientemente detalhado na descrição do preciso entendimento sobre como alguns sistemas criptográficos operam pragmaticamente. Desta feita, seguem algumas variações do modelo geral exibindo características de possíveis sistemas criptográficos:

- **Sistema de chave única:** Sistema onde remetente e receptor compartilham uma única chave, ou seja, as chaves de encriptação k_e e decriptação k_d são iguais.

$$\begin{cases} k_e = k_d = K \\ A : E(K, \mathcal{P}) = \mathcal{C} \\ B : D(K, \mathcal{C}) = \mathcal{P} \end{cases} \tag{2.2}$$

- **Sistema de chave dupla:** Sistema onde as chaves do remetente e receptor são diferentes, ou seja, $k_e \neq k_d$, sendo que $k_e = k_d^{-1}$ e $k_d = k_e^{-1}$, ou seja, uma é a inversa da outra. Um requisito de segurança é que o conhecimento de k_e não deve levar ao conhecimento de k_d; o mesmo valendo para k_d.

$$\begin{cases} k_e \neq k_d \\ A : E(k_e, \mathcal{P}) = \mathcal{C} \\ B : D(k_d, \mathcal{C}) = \mathcal{P} \end{cases} \tag{2.3}$$

- **Sistema de chave privada:** Sistema cuja segurança depende do segredo de todas as chaves, ou seja, k_e e k_d devem ser mantidas em sigilo.

$$\begin{cases} k_e = k_d \text{ ou } k_e \neq k_d \\ A : E(k_e, \mathcal{P}) = \mathcal{C} \\ B : D(k_d, \mathcal{C}) = \mathcal{P} \end{cases} \tag{2.4}$$

- **Sistema de chave pública:** Sistema onde alguma chave pode ser distribuída ao público. A chave de encriptação k_e é tornada pública, servindo, adicionalmente, como forma de autenticação. A chave de decriptação k_d não deve ser de conhecimento público e garante o sigilo.

$$\begin{cases} k_e \neq k_d \\ A : E(k_e, \mathcal{P}) = \mathcal{C} \\ B : D(k_d, \mathcal{C}) = \mathcal{P} \end{cases} \tag{2.5}$$

- **One Time Pad:** Sistema de chave privada na qual cada bit da chave é aleatório e usado para cifrar somente um bit da mensagem.

$$\begin{cases} k_e = k_d = K \\ A : E(k_e, \mathcal{P}) = \mathcal{C} \\ B : D(k_d, \mathcal{C}) = \mathcal{P} \end{cases} \tag{2.6}$$

- **Sistema sem chaves:** A designação 'sem chaves' é enganosa, dado que na verdade neste sistema existem chaves de encriptação e decriptação, mas elas não são compartilhadas.

$$\begin{cases} A : E(K_{e_1}, \mathcal{P}) = \mathcal{C}_1 \\ B : E(K_{e_2}, \mathcal{C}_1) = \mathcal{C}_2 \\ A : D(K_{d_1}, \mathcal{C}_2) = \mathcal{C}_1 \\ B : D(K_{d_2}, \mathcal{C}_1) = \mathcal{P} \end{cases} \tag{2.7}$$

2.1.1 Sistemas Criptográficos

Nas linhas precedentes, as palavras *sistemas criptográficos* foram mencionadas diversas vezes, mas foram feitas de modo, diga-se de passagem, informal. A definição que segue pretende ser uma base para a discussão que se seguirá.

Definição 2.1.2 (Sistema Criptográfico)**.** *Um sistema criptográfico (Stinson, 2001, p. 1) é uma quíntupla* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$*, onde:*

1. $\mathcal{P}$ *é um conjunto finito de possíveis textos planos.*
2. $\mathcal{C}$ *é um conjunto finito de possíveis textos cifrados.*
3. $\mathcal{K}$ *é é um conjunto finito de possíveis chaves (espaço de chaves).*
4. *Para todo* $k \in \mathcal{K}$*, existe uma regra de encriptação* $e_k \in \mathcal{K}$ *e uma correspondente regra de decriptação* $d_k \in \mathcal{D}$*;* $e_k : \mathcal{P} \to \mathcal{C}$ *e* $d_k : \mathcal{C} \to \mathcal{P}$ *são funções tais que* $d_k(e_k(x)) = x$ *para todo texto plano.*

> **Nota:** O item 4 é a principal propriedade de um sistema criptográfico, estatuindo que se um texto plano x é encriptado usando e_k e o texto cifrado resultante y é decriptado usando d_k, então o resultado é o texto plano original x.

Por sua vez, em relação aos algoritmos que são usados na implementação de sistemas criptográficos, tem-se a seguinte classificação:

- Algoritmos simétricos: as chaves de encriptação k_e e decriptação k_d são iguais.

- Algoritmos assimétricos: as chaves de encriptação k_e e decriptação k_d são diferentes).

2.1.1.1 Algoritmos Simétricos

Algoritmos criptográficos simétricos são aqueles no qual tanto a chave de encriptação k_e quando a chave de decriptação k_d são iguais, conforme as equações 2.2, ou alternativamente, quando a chave de decriptação k_d é "facilmente computável" a partir de k_e, e vice-versa. A maior desvantagem deste sistema é que a chaves de encriptação/decriptação ($k_e = k_d = k$ ou $k_e \neq k_d$) devem ser trocadas entre as partes antes do processo de comunicação poder ser iniciado, sendo que, evidentemente, isso não pode em hipótese alguma ser de conhecimento de terceiros.

Em relação a implementação, algoritmos criptográficos simétricos são classificados em *cifras de bloco* e *cifras de fluxo*.

2.1.1.2 Algoritmos Assimétricos

Algoritmos assimétricos, também chamados de *algoritmos de chave pública*, conforme as equações 2.5, são quaisquer sistemas criptográficos no qual a chave de encriptação k_e é tornada pública e a chave de decriptação k_d é mantida privada, apresentando como vantagem em relação aos sistemas simétricos a não necessidade de um canal seguro para a troca inicial de uma ou mais chaves.

A segurança de um sistema criptográfico implementado por meio de um algoritmo assimétrico jaz no *esforço computacional* que é necessário para encontrar a chave privada a partir da chave pública, mas especificamente, os algoritmos assimétricos estão baseados em problemas que não possuem solução eficiente (em termos computacionais), tais como fatoração de inteiros, cálculo de logaritmos discretos e curvas elípticas.

O primeiro sistema de chave público tornado conhecido foi a implementação do que é conhecido como *Diffie–Hellman key exchange* (troca de chaves Diffie–Hellman) (Diffie e Hellman, 1976) com base nos conceitos apresentados por Ralph Merlke (Merkle, 1978)[3]. Em seguida surgiram outros sistemas criptográficos, tais como RSA[4](Rivest et al., 1978), ElGamal (ElGamal, 1985a)(ElGamal, 1985b), e em tempos mais recentes, métodos possivelmente considerados menos ortodoxos, p. ex., (Kocarev e Tasev, 2003),

[3]Independentemente, James H. Ellis (Ellis, 1970), Clifford Cocks e Malcolm J. Williamson mostraram previamente como poderia ser realizada criptografia de chave pública, embora isto tenha permanecido classificado pelo governo britânico até 1997 (a história *secreta* da criptografia pública pode ser lida em (Singh, 2001, p. 211-220))

[4]Um sistema equivalente foi descrito por Clifford Cooks (Cocks, 1973).

(Kumar, 2006), (Alia e Samsudin, 2007), (Mishkovski e Kocarev, 2011).

Detalhes em maior profundidade sobre os sistemas RSA e ElGamal podem ser encontrados na literatura, p. ex., (Menezes et al., 1996, cap. 8), (Mollin, 2007, cap. 4) e (Delfs e Knebl, 2007, cap. 3), apenas para citar alguns.

2.1.2 Sigilo Perfeito

A segurança de um sistema criptográfico pode ser avaliada com base em dois critérios: segurança computacional e segurança incondicional.

Definição 2.1.3 (Segurança Computacional)**.** *Um sistema criptográfico é dito ser computacionalmente seguro se o melhor algoritmo para quebrá-lo toma no mínimo n passos, onde n é um número inteiro muito grande.*

Definição 2.1.4 (Segurança Incondicional)**.** *Um sistema criptográfico é dito se incondicionalmente seguro se não pode ser quebrado mesmo com a disponibilidade infinita de recursos computacionais.*

A definição de *segurança computacional* está relacionada ao esforço computacional requerido para quebrar o sistema. O problema é que dificilmente algum sistema criptográfico pode ser considerado seguro com esta definição, dado que em termos pragmáticos, a segurança do sistema criptográfico é testada com relação a certos tipos de ataques, p. ex., *busca exaustiva de chave*, mas segurança em relação a um tipo de ataque não garante a segurança contra outros tipos de ataques.

Já a segurança incondicional afirma que um sistema criptográfico é seguro quando não existe limite para a quantidade de poder computacional que um atacante (Eva) pode fazer uso. É o tipo de segurança *ideal* para qualquer sistema criptográfico, e como será visto em breve, apenas um sistema, de todos os já inventados, preenche e atende este requisito, a saber, *one-time pad*.

Outro critério de avaliação é a *segurança provável*. Sob a égide deste critério, o que se faz é tentar provar a evidência da segurança de um sistema criptográfico por meio de uma redução. Mais especificamente, tenta se mostrar que se é possível quebrar o sistema de um modo específico, então é possível resolver de modo eficiente algum problema bem conhecido e estudado que é tido como difícil. Deve ser entendido que esta abordagem somente provê uma prova de segurança em relação a algum outro problema, não sendo de forma alguma um prova irrestrita de segurança[5].

[5]Uma analogia é possível ao se considerar um problema que pode ser provado ser NP-Completo; ora, esta prova mostra apenas que o problema considerado é no mínimo tão difícil quanto qualquer

Definição 2.1.5 (Sigilo Perfeito)**.** *Um sistema criptográfico é dito ter* sigilo perfeito *se para qualquer mensagem x e qualquer cifra y:*

$$p(x|y) = p(x) \tag{2.8}$$

onde:

- $p(x|y)$ *é a probabilidade condicional do texto plano x e do texto cifrado y.*
- $p(x)$ *é a probabilidade de ocorrência do texto plano x.*

O que a equação 2.8 afirma é que a *probabilidade a posteriori* de um texto plano x, dado que se conhece (ou que é observado) o texto cifrado y, é idêntico a probabilidade a priori que o texto plano é x[6].

Uma implicação da equação 2.8 é que para qualquer par de mensagem/cifra, existe no mínimo uma chave os conecta. Portanto,

$$|\mathcal{P}| \leq |\mathcal{C}| \leq |\mathcal{K}| \tag{2.9}$$

onde $|\mathcal{P}|, |\mathcal{C}|$ e $|\mathcal{K}|$ são, respectivamente, os espaços de textos planos, cifras e chaves.

O teorema seguinte demonstra que um sistema criptográfico na qual $|\mathcal{P}| = |\mathcal{C}| = |\mathcal{K}|$ possui sigilo perfeito.

Teorema 2.1.1. *Suponha um sistema criptográfico com $|\mathcal{P}| = |\mathcal{C}| = |\mathcal{K}|$. O sistema criptográfico tem sigilo perfeito se, e somente se:*

- *Cada chave é usada com igual probabilidade* $\frac{1}{|\mathcal{K}|}$.
- *Para todo texto plano x e texto cifrado y, existe uma única chave $k \in \mathcal{K}$ tal que $e_k(x) = y$.*

Demonstração. Suponha sigilo perfeito, i. é, $p(x|y) = p(x), \forall x, y$. A menos que $p(x) = 0$, deve haver chaves suficientes de modo que um texto cifrado possa ser decodificado como um texto plano, i. é, $|C| \leq |K|$, mas por suposição, a igualdade deve se manter. Portanto, existe uma única chave para todo par de mensagem/cifra.

---

outro problema NP-Completo, mas definitivamente não é uma prova irrefutável da complexidade computacional do problema.

[6]Isto também pode ser pensado em termos de eventos independentes, ou seja, texto plano e texto cifrado são eventos independentes.

Suponha agora que as chaves $k_1, k_2, \ldots$ são únicas e são tais que $d_{k_i}(y) = x_i$. Usando a regra de Bayes:

$$p(x_i|y) = \frac{p(y|x_i)p(x_i)}{p(y)} \tag{2.10}$$

Usando a assunção de sigilo perfeito, tem-se:

$$p(y|x_i) = p(y) \tag{2.11}$$

e portanto, cada k_i deve ocorrer com a mesma probabilidade.

Agora, assuma que $p(k) = \frac{1}{|K|}$ e que existe uma chave única relacionada com qualquer par de texto plano/texto cifrado

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \tag{2.12}$$

Pela unicidade das chaves, $p(y|x) = \frac{1}{|\mathcal{K}|}$.

Calculando $p(y)$, tem-se:

$$p(y) = \sum_k p(k)p(d_k(y)) \tag{2.13}$$

$$p(y) = p(k)\sum_k p(d_k(y)) \tag{2.14}$$

$$p(y) = \frac{1}{|\mathcal{K}|}\sum_k p(d_k(y)) \tag{2.15}$$

$$p(y) = \frac{1}{|\mathcal{K}|}\sum_x p(x) \tag{2.16}$$

$$p(y) = \frac{1}{|\mathcal{K}|}.1 \tag{2.17}$$

$$p(y) = \frac{1}{|\mathcal{K}|} \tag{2.18}$$

Voltando agora para

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}.$$

e substituindo $p(y|x)$ e $p(y)$ por $\frac{1}{|\mathcal{K}|}$, tem-se:

$$p(x|y) = p(x)$$

isto é, *sigilo perfeito*. □

Como citado anteriormente, *one-time pad* é o único sistema criptográfico que

pode (matematicamente) ser provado ser incondicionalmente seguro. Dada a discussão nesta seção, agora se tem as condições necessárias para mostrar o porquê.

Definição 2.1.6 (One-Time Pad)**.** *Faça* $\mathcal{P} = \mathcal{C} = \mathcal{K} = \{0,1\}^n$*, onde* $n \geq 1$*. Seja* $x \in \mathcal{P}$ *e* $y \in \mathcal{C}$*, sendo* $e_k(x) = x \oplus k$ *e* $d_k(y) = y \oplus k$*, respectivamente, as funções de encriptação e decriptação (*$k \in \mathcal{K}$*)*[7]*.*

Teorema 2.1.2. *A cifra dada pela definição 2.1.6 provê sigilo perfeito.*

Demonstração. Deve-se mostrar *sigilo perfeito*, ou seja, que $p(x|y) = p(x), \forall x \in \mathcal{P}, y \in \mathcal{C}$.

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(k)p(x)}{p(y)} = \frac{\frac{1}{2}p(x)}{\sum_{k\in\mathcal{K}} p(k)p(d_k(y))} = \frac{\frac{1}{2}p(x)}{p(k)\sum_{k\in\mathcal{K}} p(x)} = \frac{\frac{1}{2}p(x)}{\frac{1}{2}.1} = p(x) \tag{2.19}$$

□

2.2 Aleatoriedade

Uma boa questão filosófica sempre se resume a perguntas que podem ser formuladas facilmente(?), mas dificilmente respondidas. Tal é o caso da aleatoriedade. Perguntas tais como, O que é aleatoriedade? Existem eventos aleatórios na natureza? Existem leis da aleatoriedade?, geralmente levam a mais perguntas. A presente seção não pretende dar cabo definitivamente da questão, sendo apenas uma ligeira discussão a respeito.

Informalmente (e intuitivamente), aleatoriedade pode ser descrita como a ausência de padrão, ordem ou previsibilidade, ou ainda, quando se trata de um experimento, incerteza em relação ao resultado. Apesar da intuição ser (em geral é) um ponto de partida, tal (ou tais) conceito(s) não servem de baliza. Assim, desde tempos remotos, um sentido (ou uma definição) do que é aleatoriedade tem sido procurado; não obstante, somente recentemente surgiram aprimoramentos no conceito.

Pelo menos três abordagens a aleatoriedade foram propostas: *Estocasticidade*, *Incompressibilidade* e *Tipicalidade*.

2.2.1 Estocasticidade

Richard von Mises propôs em 1919 sua noção de sequência aleatória, por ele chamada de *Kollectivs* = *coletivos* (von Mises, 1919)[8]. O que von Mises denomina de "*kollektiv*"

[7] O operador $\oplus$ significa a operação de *ou exclusivo*.
[8] Também em disponível em (Mises, 1964).

são sequências de possíveis resultados, que supostamente se estendem indefinidamente (Löf, 1969, p. 15)

$$x_1, x_2, \ldots, x_n, \ldots$$

que devem obedecer a dois requisitos (Löf, 1969, p. 16).

1. Faça n_A denotar a frequência com a qual um evento A tem ocorrido nas n primeiras tentativas, i. é, o número de pontos x_m, $1 \leq m \leq n$, que pertencem a um subconjunto A do espaço amostral. Para todo "*angebbare Punktmenge*" A, o limite da frequência relativa deve existir

$$\lim_{\to\infty} \frac{n_A}{n} = \pi(A)$$

   Por *angebbare*, aparentemente von Mises que dizer qualquer subconjunto de um espaço amostral finito. O limite supra definido é chamado de probabilidade do evento A em relação ao coletivo dado.

2. Se selecionamos uma subsequência $x_1, x_2, \ldots, x_n, \ldots$ de tal modo que a decisão de se x_n deveria ser (ou não) selecionada independentemente de x_n, então a frequência relativa limitante da subsequência deveria existir e ser igual a sequência original. A regra de seleção deveria, é claro, ser tal que a sequência selecionado seria infinita.

O primeiro requisito diz respeito a *estabilidade da frequência relativa* ou ainda, representa a *Lei dos Grandes Números*. O segundo requisito afirma que tal a estabilidade da frequência relativa é preservada para subsequências extraídas (ou selecionadas), o que elimina subsequências não aleatórias triviais.

A proposta de Mises, entretanto, apresenta problemas: primeiro, a partir da formulação dada, um *kolleltiv* não pode ser dado por uma lei matemática, a menos para casos triviais, e se não existe uma prescrição que diz como computar os elementos, como se pode saber se tal sequência existe?

O segundo requisito, evidentemente, não pode pode ser válido para qualquer sequência, e desta feita, é mister restringir as seleções a uma classe admissível. Tais questões começaram a serem resolvidas por Abraham Wald e Alonzo Church. Em particular, Abraham Wald (Wald, 1936) mostrou que se a classe admissível for enumerável, então o conjunto dos kollektivs é não enumerável. Por sua vez, Alonzo Church (Church, 1940) propôs identificar as seleções admissíveis como sendo as efetivas, particularmente, funções computáveis ou funções parciais recursivas.

Contudo, o conceito tomou um golpe derradeiro com o trabalho de Ville (Ville, 1939), que mostrou que existem coletivos que não são suficientemente aleatórios; por

exemplo, existem sequências binárias nas quais $\frac{n_A}{n} \geq \frac{1}{2}$, ou seja, na qual aparecem mais uns do que zeros (ou vice-versa).

2.2.2 Incompressibilidade

O conceito de aleatoriedade como incompressibilidade pode ser ilustrado por meio de uma analogia sobre o envio de mensagens. Considere as seguintes strings binárias (dos quais são exibidos os primeiros 64 dígitos):

(a) 0000111100001111000011110000111100001111000011110000111100001111...

(b) 0010010000111111011010101000100010000101101000110000100011010011...

(c) 0001001001001101110001010001101000101010100001000110100001011011...

A string binária (a) não somente "aparenta-se", mas é altamente ordenada. Ela é constituída por sub-sequências compostas de 4 zeros seguidos por 4 uns, assim, pode-se dizer que a mesma poderia ser gerada por uma "*lei de formação*": primeiro 4 zeros, depois 4 uns. Alternativamente, poderia ser escrito um programa de computador (ou um *algoritmo*) capaz de gerar a sequência, por exemplo:

```
void generate(int n)
{
    for(int i = 0; i < n; i++)
    {
        if((n % 0) == 0)
            printf("0000");
        else
            printf("1111");
     }
}
```

Por sua vez, a string (b) à primeira vista não possui uma lei de formação, mas a bem da verdade ela corresponde a expansão binária dos dígitos decimais de π, o qual também possui vários algoritmos (receitas, prescrições, ...) para a sua geração, por exemplo, a fórmula de Bailey–Borwein–Plouffe:

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

Por fim, a string (c) foi produzida por meio de um TRNG (True Random Number Generator), mais especificamente, a partir do ANU Quantum Random Numbers Server.

Suponha agora que se deseja encaminhar as strings binárias para outrem. Qual delas seria menos/mais custosa (de enviar)? Considerando o que foi já comentado a respeito das strings exemplo, tanto as strings (a) quanto (b) seriam menos custosas de encaminhar por mensagem do que a string (c). Por quê? Bem, considere que um modo mais econômico de envio da mensagem seria não encaminhar a própria string, mas uma fórmula/algoritmo que descreve como a mesma pode ser construída. Neste caso, o conteúdo da mensagem seria bem menor do que o comprimento da string (considerando, é claro, que a string é grande o suficiente). Isso acontece para as strings (a) e (b). Por outro lado, a string (c) pode ser considerada "sem lei", significando que não existe uma forma de condensar sob uma fórmula/algoritmo a informação necessária para produzir a string, não restando alternativa a não ser transmitir toda a string. Nos termos da *teoria algoritmica da informação*, a string (c) é incompressível, sendo esta uma característica inerente a todas as sequências aleatórias.

Definição 2.2.1. *(Chaitin, 1975, p. 48) Uma série de números é aleatória se o menor programa capaz de especificá-lo para um computador tem aproximadamente o mesmo número de bits de informação que a própria série.*

Na definição, a palavra *programa* deve ser entendido como o código binário de uma MT (Máquina de Turing) que quando apresentada com uma entrada nula, imprime sucessivamente os dígitos da sequência.

Um fato comum na ciência, a definição acima foi desenvolvida independentemente por Gregori J. Chaitin (Chaitin, 1966), Andrey Nikolaevich Kolmogorov (Kolmogorov, 1965) e Ray J. Solomonoff (Solomonoff, 1964) em meados dos anos 60.

Interessantemente, uma consequência da proposta da aleatoriedade como incompressibilidade é que para provar que uma série de dígitos não é aleatória, basta encontrar um programa que gere a série e que seja substancialmente menor do que a própria série. Por outro lado, para provar que uma dada sequência é aleatória, deve-se provar que nenhum programa menor do que a própria série existe.

Outro conceito é o de *complexidade*, sendo que para uma dada série de dígitos, complexidade é o número de bits que devem ser inseridos em um computador de modo a obter a série como saída, o que equivale a dizer que complexidade é igual ao tamanho de bits do programa minimal da série. Desta feita, complexidade pode ser definida como uma medida de aleatoriedade.

Finalizando, uma consequência *notável*, derivação dos teoremas da incompletude de Gödel, é que em um sistema formal, como o proposto por Chaitin, nenhum número

pode ser provado ser aleatório.

2.2.3 Tipicalidade

A proposta de Martin-Löf (Löf, 1966) foi inicialmente aventada como alternativa a incompressibilidade. A ideia de *sequência típica* é de uma que não apresente padrões reconhecíveis. Mais especificamente, uma sequência é dita ser típica se ela satisfaz todas as propriedades de aleatoriedade, ou ainda, se as propriedades de probabilidade 1 no conjunto das sequências infinitas são atendidas. Alternativamente, uma sequência x_n é dita ser aleatória se ela passa em todos os testes estatísticos efetivos.

2.3 Testes Estatísticos de Aleatoriedade

Testes estatísticos de aleatoriedade tem como objetivo determinar se uma sequência particular de números foi produzida por um gerador de números aleatório, sendo que a abordagem consiste no cálculo de certas quantidades estatísticas. Estas quantidades são então comparadas com os valores médios que seriam obtidos no caso de uma sequência aleatória. Per si, os valores médios são obtidos a partir de computações executadas em um modelo ideal de gerador de números aleatório.

Sequências de números aleatórios são extensivamente usados em aplicações criptográficas, por exemplo, para a geração de chaves de encriptação/decriptação e para melhorar a segurança do protocolo criptográfico, tanto para algoritmos simétricos quanto assimétricos.

Agora, como uma sequência de números aleatórios pode ser obtida e/ou gerada? Dependendo da forma como os números são gerados, os geradores podem ser classificados em TRNG (*True Random Number Generator*) e PRNG (*Pseudo Random Number Generator*).

Um TRNG utiliza medições de algum fenômeno físico que é conjecturado ser aleatório, compensando possíveis desvios no processo de medição. Fontes de entropia natural incluem: ruído atmosférico, decaimento radioativo, flutuações quânticas do vácuo, ruído branco gerado por meio de uma junção semicondutora, aleatoriedade quântica baseada na contagem de fótons, entre outros.

Um PRNG, por outro lado, usa métodos computacionais para produzir longas sequências de números aparentemente aleatórios, mas que são unicamente determinados pelo algoritmo e chave utilizados. Números gerados por um PRNG são chamados

de pseudo-aleatórios, dado que a geração dos mesmos é completamente determinística e reproduzível. Ao longo do tempo, vários métodos tem sido empregados, entre os quais: método do meio do quadrado (inventado por John von Neumann (von Neumann, 1951)), LCG (*Linear Congruential Generator*), CLCG (*Combined Linear Congruential Generator* (L'Ecuyer, 1988)), LFSR (*Linear-Feedback Shift Register*), Mersenne Twister (Matsumoto e Nishimura, 1998b), caos (Bucci e Luzzi, 2016).

Idealmente, toda sequência de números aleatórios gerada deve possuir certas propriedades estatísticas. Em particular, espera-se que uma sequência aleatória não apresente padrões e que os valores gerados se conformem a uma distribuição uniforme. A pergunta é: como é que se pode garantir que uma sequência é verdadeiramente aleatória? A resposta a esta inquirição é dada na Figura 2.2.

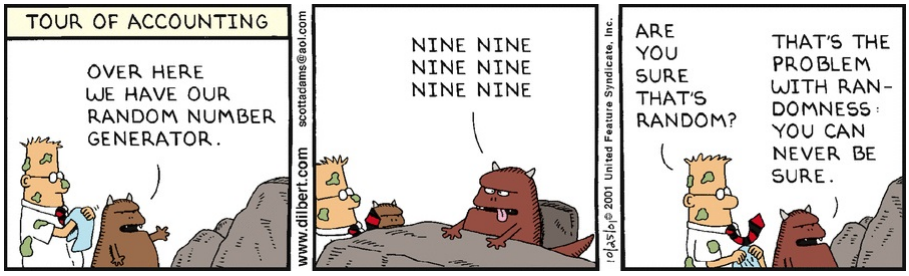


Figura 2.2: Aleatoriedade - Dilbert, por Scott Adams.

Mas se assim o é, que nunca se pode decidir com absoluta certeza que uma sequência é aleatória, como alguém pode estar seguro na sua utilização (para qualquer fim)? A réplica a esta inquirição é que embora não se possa "provar" que uma sequência é aleatória, testes estatísticos podem asseverar se a mesma é aleatória em certo nível de significância.

Para realizar os testes de aleatoriedade, diversas suítes de testes estatísticos estão disponíveis. Per si, os testes verificam algum aspecto que supostamente a sequência deveria possuir para ser considerada aleatória. Exemplos de testes são entropia aproximada, coeficiente de correlação serial, teste χ-quadrado, cálculo do valor de π pelo método de Monte Carlo, teste de frequência (monobit/bloco), teste Runs, teste de soma cumulativa, espaçamento de aniversário, sobreposição de permutações, apenas para citar alguns. Entretanto, nenhum conjunto de testes estatísticos pode ser considerado completo, e novas anomalias estatísticas passíveis de teste podem ser encontradas.

2.4 Criptografia baseada em Caos

Tradicionalmente, algoritmos de criptografia empregam técnicas baseadas em anéis sobre números inteiros, logaritmo discreto, curvas elípticas, curvas hiperelípticas, empa-

relhamentos bilineares, etc. Mas existem outras possibilidades? A fim de responder a esta pergunta, deve-se voltar no tempo.

Antes do nascimento do que hodiernamente é conhecido como *caos determinístico*, Claude E. Shannon (Shannon, 1949, p. 711-712) descreveu uma característica essencialmente ligada ao mesmo, a saber, *mixagem topológica*. Shannon utilizou o termo *mixing transformations* (transformações de mistura), indicando que as mesmas poderiam ser utilizadas para espalhar mensagens de alta probabilidade de forma uniforme por todo o espaço de mensagens. Em adição, nas mesmas linhas é mencionado (embora não diretamente) outro conceito característico de sistemas caóticos, a saber, *sensibilidade às condições iniciais*.

Como exemplo de boas transformações de mistura, Shannon citou o exemplo tipificado por uma massa folhada que é esticada e depois dobrada em si mesma muitas e muitas vezes[9]. Imagine um mancha de corante alimentar adicionada a uma folha de massa folhada. Após algumas repetições de esticar e dobrar, a pequena mancha acabaria tão esticada que ocuparia uma área muito maior e então, após ser dobrada muitas vezes, ela ocuparia aproximadamente todas as camadas. Dois pedaços de massa colorida que estavam inicialmente próximos poderiam, após o número apropriado de repetições, ter qualquer separação entre elas permitida pela dimensão da massa. Operações tais como esticar e dobrar são características de muitos mapeamentos caóticos, tais como o Mapa da Ferradura (Horseshoe Map), embora isso não seja uma condição necessária ou suficiente (Ruelle, 2006).

O que torna o caos (determinístico) atraente para a construção de sistemas criptográficos é a natureza altamente imprevisível e aparentemente aleatória dos sinais produzidos, que pode ser caracterizada pela propriedade de sensibilidade às condições iniciais, embora esta não garanta que um sistema é caótico. As características que tornam um mapeamento caótico, e que em última instância são interessantes para aplicações criptográficas, serão abordadas em profundidade no capítulo 3.

A princípio, a criptografia baseada em caos, cujo aparecimento se deu completamente fora do contexto da criptografia convencional, surgiu como uma aplicação da teoria do caos e como uma aplicação de sincronização do caos (Pecora e Carroll, 1990)[10]. Como exemplo da primeira abordagem, tem-se: (1) (Matthews, 1989), cuja ideia é que mapas iterativos não lineares são capazes de gerar sequências caóticas de números ale-

[9]Na verdade Shannon referenciou o exemplo dado por Hopf (Hopf, 1934, p. 73).

[10]Sincronização é uma propriedade de osciladores caóticos acoplados estudada no âmbito das técnicas de comunicação cuja a ideia básica é mascarar a mensagem (a ser enviada) por meio de um sinal caótico e usar sincronização no receptor para filtrar o sinal caótico.

atórios, de modo que que tal função pode ser adequada para uso criptográfico, e em particular, servir como um substituto para o sistema one-time pad; (2) (Habutsu et al., 1991), cuja ideia consiste em utilizar as propriedades inerentes ao caos para a construção de um sistema criptográfico no qual o texto plano é representado por meio da iteração de um mapa caótico inverso (em particular, os autores usaram como exemplo o mapa tenda - tent map). Já como segundo exemplo da segunda abordagem tem-se, por exemplo: (a) (Cuomo et al., 1993), que investigam a implementação do sistema de Lorenz com vistas a aplicação segura em comunicações por meio de sincronização caótica de sistemas; (b) (Parlitz et al., 1992), que tratam da transmissão de sinais digitais por meio de sincronização caótica usando o circuito de Chua.

Em relação aos métodos de encriptação caótica, eles podem ser classificados em digitais ou analógicos, e de forma um tanto quanto livre, pode-se projetar cifras caóticas digitais de dois modos: (1) Usar sistemas caóticos para gerar um chave de fluxo (contínua) para encriptar o texto plano (cifras de fluxo), por exemplo, (Matthews, 1989)(jun Huang e qian Zhao, 2013); (2) Usar o texto plano/chave secreta como condições iniciais e/ou parâmetros de controle, iterando sistemas caóticos n vezes de modo a obter o texto cifrado (cifras de bloco), por exemplo, (Habutsu et al., 1991).

Contudo, devido ao fato da criptografia baseada em caos ter surgido fora do ambiente da criptografia tradicional, aliada ao fato de que a mesma foi desenvolvida por engenheiros e físicos (possivelmente não cientes dos conceitos e padrões usados por criptógrafos), grande parte das propostas foi publicada em jornais de física e engenharia, reiteradamente com falta de rigor técnico, aliado ao fato de serem criptograficamente fracas (inseguras)[11] e computacionalmente ineficientes. Seja como for, pode ser apontado que existe uma relação estreita entre caos e criptografia (Brown e Chua, 1996, p. 238-239)(Fridrich, 1998, p. 1280-1281).

Alguns aspectos comparativos entre propriedades criptográficas e caos pode ser vistos na Tabela 2.1. Já a Tabela 2.2 descreve diferenças e similaridades entre sistemas caóticos e algoritmos criptográficos.

A partir da Tabela 2.2 verifica-se que algumas diferenças são proeminentes, enquanto que outras não se constituem verdadeiramente como tal:

1. Sistemas caóticos funcionam em conjuntos (subconjuntos) de números reais ($\mathbb{R}$), ao passo que algoritmos criptográficos (tradicionais) estão baseados em um sub-

[11] Por exemplo, as propostas de (Baptista, 1998) e (Alvarez et al., 1999) foram analisadas por (Jakimoski e Kocarev, 2001), que provaram que as mesmas eram sujeitas a serem "quebradas" por meio do conhecido ataque de texto plano conhecido.

Tabela 2.1: Propriedades caóticos versus propriedades criptográficas (Alvarez, 2006).

Propriedade caótica	Propriedade Criptográfica	Descrição
Ergodicidade	Confusão	A saída tem a mesma distribuição para qualquer entrada.
Sensibilidade às condições iniciais/parâmetros	Difusão com uma pequena mudança no texto plano/chave secreta	Um pequeno desvio na entrada pode causar uma grande mudança na saída.
Mistura	Difusão com uma pequena mudança em um bloco plano do texto inteiro	Um pequeno desvio local pode causar uma grande mudança no espaço inteiro.
Dinâmica determinística	Aleatoriedade pseudo-aleatória	Um processo determinístico pode causar um comportamento pseudo-aleatório.
Complexidade estrutural	Complexidade algorítmica	Um simples processo tem uma alta complexidade.

Tabela 2.2: Diferenças e similaridades: sistemas caóticos versus algoritmos criptográficos tradicionais (Kocarev, 2001).

Sistemas Caóticos	**Algoritmos Criptográficos**
Espaço de Fase: Conjunto (subconjunto) de $\mathbb{R}$	Espaço de Fase: Subconjunto finito de $\mathbb{Z}$
Iterações	Rodadas
Parâmetros	Chave
Sensibilidade às condições iniciais	Difusão
?	Segurança e Performance

conjunto finito de números inteiros ($\mathbb{Z}$).

2. Um passo no algoritmo que constitui o sistema caótico é chamado de iteração, já em algoritmos criptográficos (tradicionais) isso é chamado de rodada[12].

3. Os valores repassados para um sistema caótico são chamados de parâmetros, já em um algoritmo criptográfico (tradicional) isso é chamado de chave[13].

4. Sistemas caóticos possuem (entre elas) a propriedade de sensibilidade às condições iniciais, significando que valores incialmente próximos divergem assintoticamente quando iterados; por sua vez, em algoritmos criptográficos (tradicionais) a propriedade de difusão diz respeito sobre à influência com que cada bit do texto plano e da chave de encriptação tem sobre muitos bits do texto cifrado.

5. Até o presente não existe um conjunto de técnicas que garantam o desempenho e segurança de algoritmos criptográficos baseados em caos, ao passo que para

[12] A diferença, obviamente, é apenas uma questão de nomenclatura.
[13] A diferença aqui, novamente, é mais uma questão de nomenclatura, dado que em um algoritmo de criptografia baseado em caos os parâmetros corrrespondem à noção de chave.

algoritmos criptográficos tradicionais esse conjunto existe.

Agora, antes de se concluir esta seção, uma pergunta: Quais são áreas da criptografia tem sido aplicado o caos, ou mais especificamente, quais são os tipos de aplicações criptográficas baseadas em caos? A resposta a esta inquirição é por demasiado longa para ser listada em sua totalidade em poucas linhas, contudo, dentre as muitas aplicações criptográficas baseadas em caos pode-se citar: *função hash baseada em caos* (Yi, 2005)(Rani et al., 2011)(Wu, 2015); *criptografia de chave-pública baseada em caos* (Kocarev e Tasev, 2003)(Tenny et al., 2003)(Tenny e Tsimring, 2005); *marca d'água digital baseada em caos*(Tefas et al., 2003)(Mooney et al., 2008); *encriptação multimídia* (imagens, áudio, vídeo)(Fu et al., 2007)(Sathishkumar et al., 2011).

Capítulo 3

Noções de Caos Determinístico

O presente Capítulo tem o objetivo de apresentar ao leitor os conceitos envolvidos na teoria do caos (determinístico) que são basilares para o presente trabalho. Contudo, embora seria de interesse que tais assuntos fossem abordados com profundidade (e vigor), tal objetivo escaparia do escopo do presente trabalho. Assim, sistemas dinâmicos e suas propriedades (nos quais o caos determinístico se manifesta) serão abordados somente na profundidade necessária para entendimento em caráter elementar (o leitor interessado pode sondar as referências constantes ao longo do presente capítulo).

3.1 Caos Determinístico

A origem do caos determinístico pode ser datada do final do século XIX, quando Henri Poincaré submeteu um trabalho para uma competição matemática patrocinada pelo rei sueco Oscar II, sob a seguinte epígrafe: "*Sur le problème des trois corps et les équations de la dynamique*" (Poincaré, 1890). Nesta pesquisa, Poincaré tornou-se a primeira pessoa a descobrir um sistema determinístico caótico. O enunciado do problema que levou ao trabalho de Poincaré (dado pelo eminente matemático alemão Karl Wilhelm Theodor Weierstrass) pode ser encontrado em (Cvitanović et al., 2013, p. 768).

Outro relevante trabalho da mesma época é o do matemático francês Jaques Hadamard, *Les surfaces à courbures opposées et leurs lignes géodésiques* (Hadamard, 1898), na qual é deixado explícito o conceito hodierno de *dependência sensível sobre as condições inicias*.

Seja como for, o atual conceito de caos determinístico começou a ser talhado somente a partir dos anos 60 do século XX, quando o ferramental matemático e computacional estava amadurecido o suficiente. Até este tempo, era usual entre os cientistas distinguir dois tipos de sistemas e/ou fenômenos: aqueles que perfeitamente descritos por leis matemáticas precisas e outros impossíveis de serem condensados em fórmulas.

Para os primeiros, era razoável conceber que soluções sempre são possíveis, muito embora pragmaticamente nem sempre seja este o caso. Para os segundos, a única possível ferramenta capaz de tentar captar a essência era a teoria das probabilidades. Dito de outra forma, era-se acreditado que sistemas simples são determinísticos e sempre (em teoria) terão uma resposta, ao passo que sistemas complexos comportam-se de forma complicada. Então, o que aconteceu foi uma mudança de paradigma: percebeu-se que sistemas complexos podem comporta-se de forma simples e que sistemas simples podem ter comportamento complexo. O que hoje costuma-se denominar de *caos*, é um conjunto de características que aparecem nos mais variados fenômenos. Não cabendo neste ponto uma descrição detalhada do assunto, isso é deixado para a seção 3.2. Contudo, há de se notar que nem mesmo dentre os especialistas não existe consenso, por exemplo, em (Tucker, 2009): [...] despite more than four decades of intense research in this area, there is still no general agreement as to what the word chaos should really mean. Para obras de conhecimento geral, o leitor podeira remeter-se a (Gleick, 1987), (Lorenz, 1993), (Ruelle, 1993), (Stewart, 1997), (Smith, 1998), (Prigogine, 2000) e Smith (2007), textos realmente excelentes, dois dos quais foram escritos por pioneiros na área do caos.

De qualquer forma, verifica-se que o caos se manifesta em sistemas dinâmicos. Assim, é mister que se examine o que é um sistema dinâmico[1]. A próxima seção é dedicada a esta discussão.

3.1.1 Sistemas Dinâmicos

Sinteticamente, um sistema dinâmico consiste de um conjunto de regras que mapeiam um estado presente no futuro. Dito de outra forma, um sistema dinâmico pode ser caracterizado como uma função f agindo em um conjunto X, chamado de *espaço de fase* (onde todos os possíveis estados do sistema são representados e na qual cada possível estado corresponde a um único ponto)[2]. Dado um estado inicial $x_0 \in X$, o estado do sistema em um tempo futuro é calculado iterando-se a função $f : X \to X$ a partir de $x_0 \in X$. Sua representação é:

$$x_{n+1} = f(x_n) \ (n \in \mathbb{N}) \tag{3.1}$$

embora também seja usual a seguinte representação:

$$\begin{cases} f^0(x_0) = x_0 \\ f^{n+1} = f(f^n(x_0)) \end{cases} \tag{3.2}$$

A sequência de iterações $f^n(x)_{n=0}^{\infty}$ é chamada de órbita (a frente) de x sobre f,

[1] Uma exposição elucidativa e não excessivamente formal é dada em (Palis, 1999).
[2] Em Nolte (2010) é dado um relato histórico para o origem do termo.

a qual pode ser representada por $x_0, f(x_0), f(f(x_0)), f(f(f(x_0))), \ldots$; se a função f for inversível, pode-se falar da *órbita reversa*, $f^{-n}(x)_{n=0}^{\infty}$. Outra forma de representação para as órbitas é considerar a variável n como sendo uma variável temporal discreta, isto é, faz-se $n = 0 = t_0, n = 1 = t_1, \cdots, n = k = t_k, \ldots$ Pareando-se a variável temporal com a variável espacial tem-se a seguinte sequência de pares ordenados:

$$(t_0, x_0), (t_1, f(x_0)), \ldots, (t_k, f^k(x_0)), \ldots$$

Agora, dada a sequência que representa o comportamento do sistema (órbita), a seguinte pergunta é de interesse maior: qual é o destino das órbitas do sistema quando $n = t_n \to \infty$ (em outras palavras, qual é o *comportamento assintótico* das órbitas)? Algumas possíveis respostas são:

(a) A órbita tende para um valor fixo;

(b) A órbita apresenta comportamento periódico;

(c) A órbita tende ao infinito;

(d) A órbita não apresenta nenhum dos comportamentos anteriores.

É precisamente o comportamento da órbita no caso (d) que representa imprevisibilidade no sistema. Neste caso não existe um padrão evidente (ou aparente), ou dito de outra forma, o comportamento é caótico.

Outra questão de interesse é: O comportamento da(s) órbita(s) é afetado pela escolha da condição inicial x_0?

Nota: Até aqui o trabalho se referiu somente a *sistemas dinâmicos discretos*, contudo, sistemas dinâmicos também podem ser descritos por *equações diferenciais* $\dot{x} = f(x)$; a diferença agora (em relação aos sistemas dinâmicos discretos) é que o tempo é tomado como sendo contínuo, sendo por isso denominados de *sistemas dinâmicos contínuos*. A notação usual para a solução (fluxo) de tais sistemas é:

$$\frac{d}{dt}\phi(x,t) = f(\phi(x,t)), \;\; \phi(x,0) = x \tag{3.3}$$

e para eles as perguntas com relação ao comportamento de longo prazo são as mesmas.

Continuando a discussão, a seguir são dados exemplos de sistemas dinâmicos bem conhecidos e estudados. Os três primeiros são sistemas contínuos (sistema Lorenz,

sistema Rössler e circuito de Chua) e os dois seguintes são sistemas discretos (mapa Hénon e mapa Logístico).

Em particular, o sistema Lorenz é inserido aqui por ser de interesse não somente histórico (foi o primeiro sistema de equações diferenciais na qual surge um atrator caótico), mas também porque outros sistemas foram propostos na tentativa de entender a dinâmica do mesmo, por exemplo, o sistema Rössler, o circuito de Chua e o mapa Hénon, apresentados ao longo do presente capítulo, foram originalmente propostos com este objetivo. Já o mapa Logístico é usado para exemplificar algumas propriedades dos sistemas dinâmicos.

3.1.1.1 O Sistema Lorenz

O sistema Lorenz consiste em um sistema de equações diferenciais ordinárias primeiramente estudado por Edward Lorenz (Lorenz, 1963) como um modelo matemático (simplificado) da atmosfera. Ele é notável por ter soluções caóticas para certos parâmetros de valores e certas condições iniciais. Em particular, o conjunto de soluções caóticas quando plotadas em um gráfico lembra a figura de uma borboleta (Figura 3.1).

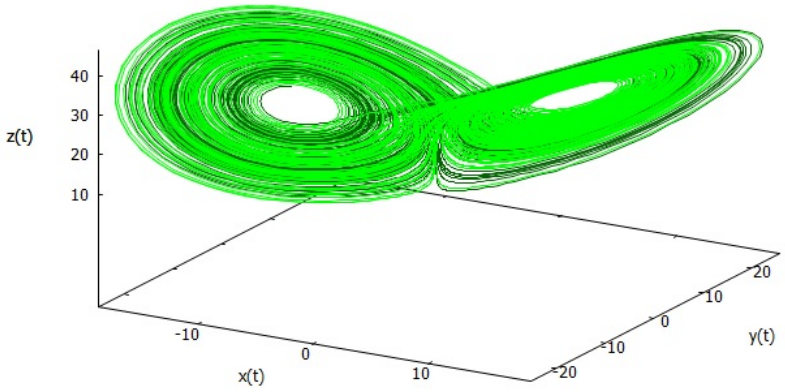


Figura 3.1: Atrator Lorenz para duas condições iniciais próximas, a saber, $(-8, 8, 27)$ (verde escuro) e $(-8.01, 8.01, 27.01)$ (verde claro).

Imagine uma fatia retangular de ar aquecido por baixo, esfriado acima e limitado por arestas mantidas em temperatura constante. Nesta descrição simples para a atmosfera, o fundo é aquecido pela terra e o topo é esfriado pelo vazio do espaço exterior. Dentro desta fatia, ar quente sobe e ar frio desce. Neste modelo, células de convecção se desenvolvem, transferindo calor de baixo para cima. Com base nesta abstração, o

sistema de equações proposto por Lorenz é descrito pelo seguinte conjunto de equações[3]:

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = \rho x - y - xz \\ \dot{z} = xy - \beta z \end{cases} \tag{3.4}$$

onde o estado da atmosfera pode ser representado pelas variáveis x, y e z, que representam, respectivamente, o fluxo convectivo, a distribuição de temperatura horizontal e a distribuição de temperatura vertical. Os parâmetros σ, ρ e β são, reciprocamente, a razão de viscosidade para a condutividade termal, a diferença de temperatura entre o topo e o fundo da fatia e a razão entre o comprimento pela altura da fatia.

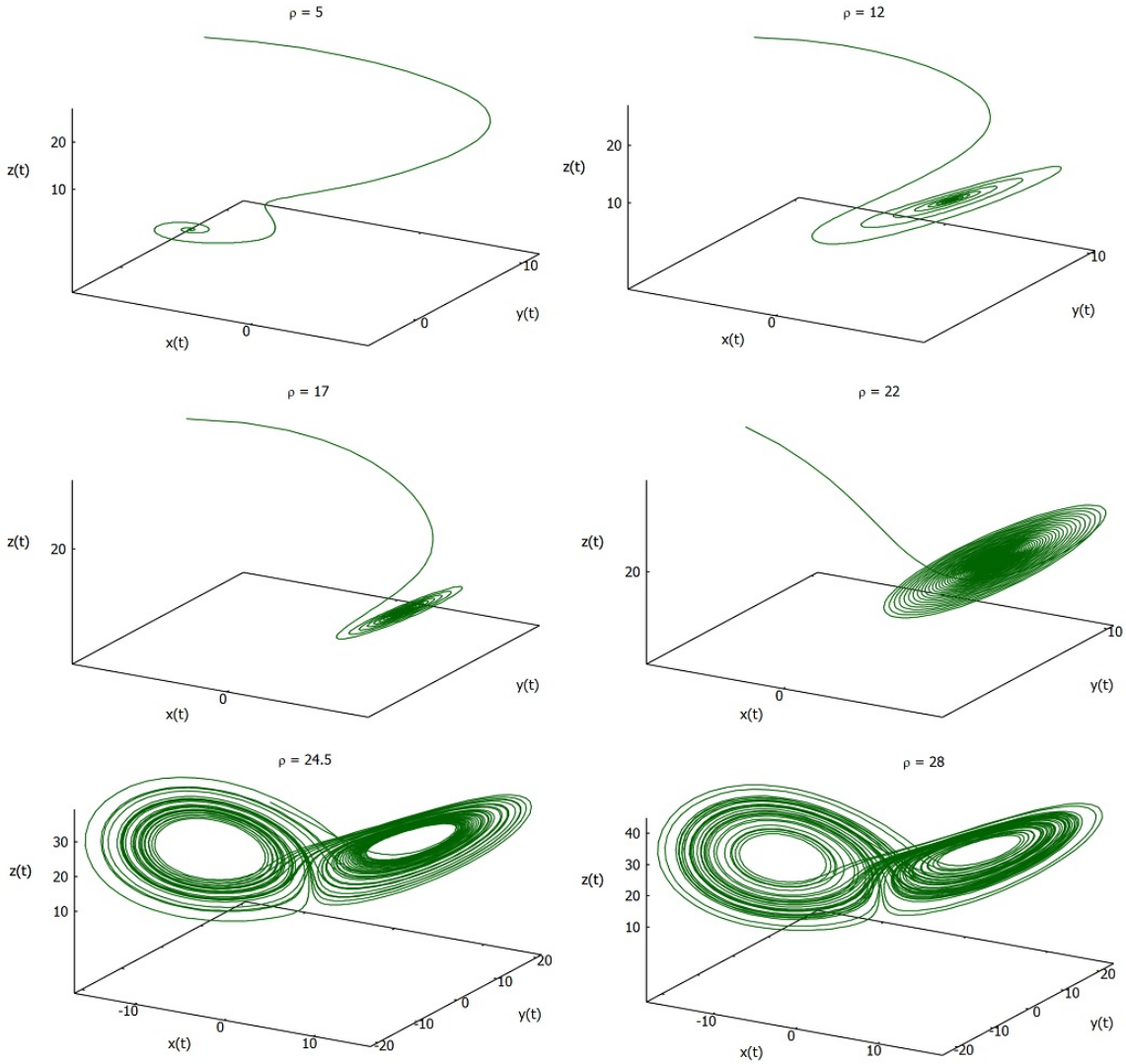


Figura 3.2: Desenvolvimento do atrator Lorenz.

Quando $\rho = 28$, $\sigma = 10$ e $\beta = \dfrac{8}{3}$, o sistema Lorenz tem soluções caóticas e quase

[3]Um relato mais detalhado pode ser encontrado em (Alligood et al., 1996, p. 359-365).

todos os pontos iniciais tendem para um conjunto invariante, o atrator Lorenz (Fig. 3.1), que foi o primeiro atrator caótico conhecido a partir de um conjunto de equações diferenciais. Em particular, o desenvolvimento do atrator Lorenz pode ser visto na Figura 3.2, sendo que para tanto variou-se o valor de ρ, mantidos constantes $\sigma = 10$ e $\beta = \frac{8}{3}$; a condição inicial é $(-8, 8, 27)$.

O estudo numérico das órbitas pode ser realizado com métodos de análise numérica para equações diferenciais, por exemplo, o método Runge-Kutta de 4ª ordem.

3.1.1.2 O Sistema Rössler

O sistema Rössler, originalmente proposto por Otto E. Rössler (Rössler, 1976), consiste em um sistema não linear composto por três equações diferenciais ordinárias, intentado para ser um modelo que produzisse fluxos no espaço de fase semelhantes aos gerados pelo sistema Lorenz, mas que permitisse análise qualitativa mais simples.

Uma das diferenças entre os sistemas Lorenz e Rössler é que o primeiro contém dois termos não lineares, a saber, os termos $-xz$ e xy (respectivamente para $\dot{y}$ e $\dot{z}$), ao passo que o segundo contém apenas um termo não linear zx (para $\dot{z}$).

As equações que descrevem o sistema Rössler são:

$$\begin{cases} \dot{x} = -y - x \\ \dot{y} = x + ay \\ \dot{z} = b + z(x - c) \end{cases} \tag{3.5}$$

onde $(x, y, z) \in \mathbb{R}^3$ são variáveis dinâmicas descrevendo o espaço de fase e $(a, b, c) \in \mathbb{R}^3$ são parâmetros.

Assim como no caso do sistema Lorenz, o sistema Rössler produz soluções caóticas para determinados valores dos parâmetros. Em particular, Rössler fixou os parâmetros em $a = 0.2$, $b = 0.2$ e $c = 5.7$.

Com a escolha certa de parâmetros, a trajetória correspondente às soluções numericamente computadas produzem, quando plotadas, uma figura chamada de atrator Rössler (Fig. 3.3)[4].

[4]Em particular, os parâmetros e valor inicial para a Figura 3.3 foram, respectivamente, $a = 0.1$, $b = 0.1$, $c = 14$ e $(0.1, 0.1, 0.1)$

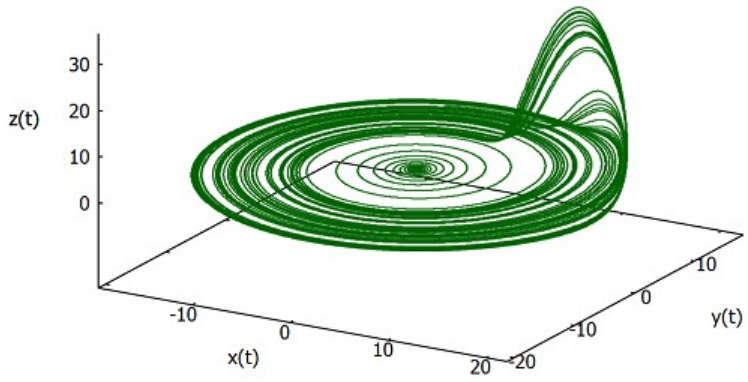


Figura 3.3: Atrator Rössler.

Por fim, o desenvolvimento do atrator Rössler pode ser visualizado na Figura 3.4, sendo que aqui variou-se o parâmetro c e manteve-se constantes os parâmetros $a = b = 0.1$; o valor inicial é igual a $(0.1, 0.1, 0.1)$.

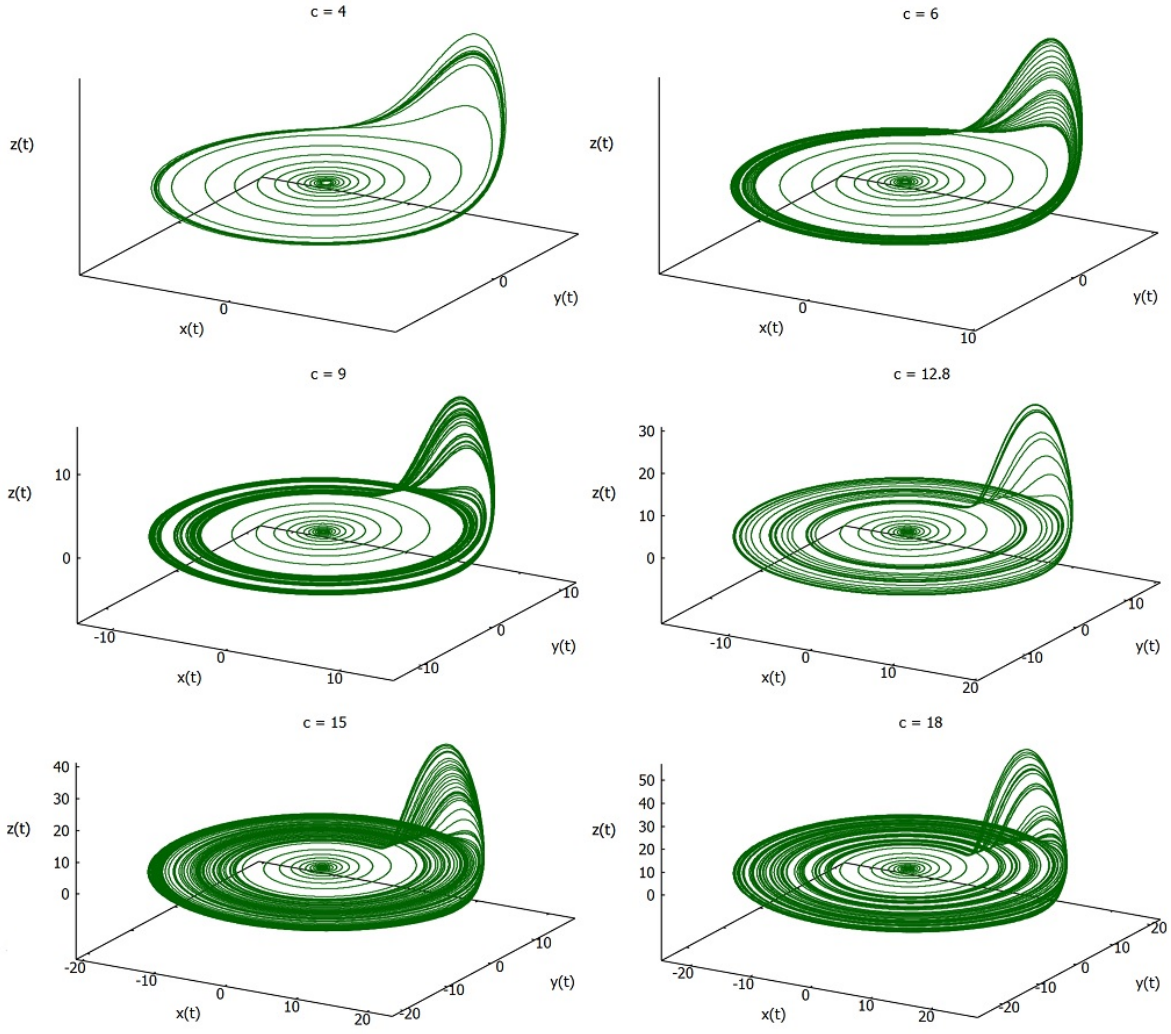


Figura 3.4: Desenvolvimento do atrator Rössler.

3.1.1.3 O Circuito de Chua

O circuito de Chua[5] (Fig. 3.5), concebido por Leon O. Chua quando o mesmo estava visitando a Universidade de Waseda (Japão) entre outubro/1983 a janeiro/1984 (Matsumoto, 1984), é o mais simples circuito eletrônico exibindo comportamento caótico (o mesmo pode ser considerado como um oscilador não periódico), sendo inventado como resposta a dois objetivos não alcançados entre os pesquisadores do caos em relação a dois aspectos das equações Lorenz, a saber: (1) Criar um sistema em laboratório que pudesse modelar realisticamente as equações Lorenz de modo a demonstrar o caos como um fenômeno físico robusto, e não meramente como o resultado de erros de arredondamento do computador; (2) Provar que o atrator Lorenz, obtido por meio de uma simulação computacional, é realmente caótico em um sentido matemático rigoroso[6].

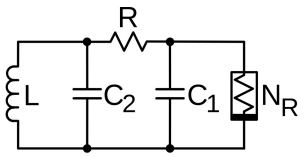


Figura 3.5: O circuito de Chua.

A partir da Figura 3.5 constata-se que o circuito de Chua é composto por dois capacitores (C_1 e C_2), um indutor L, um resistor R e uma resistência negativa não linear N_R, também chamada de diodo de Chua.

Matematicamente, o circuito de Chua pode ser modelado pelo seguinte conjunto de equações diferenciais:

$$\begin{cases} \dot{x} = \alpha(y - x - g(x)) \\ \dot{y} = x - y + z \\ \dot{z} = \beta y \end{cases} \tag{3.6}$$

que representam, respectivamente, as voltagem através dos capacitores e do indutor. Os parâmetros α e β dependem dos componentes do circuito. A função $g(x)$ é uma função linear por partes que representa a mudança de resistência versus corrente através do diodo de Chua. Sua formulação é dada por:

$$g(x) = \begin{cases} m_0 x + m_0 - m_1, & \text{se } x \leq 1 \\ m_1 x, & \text{se } -1 \leq x \leq 1 \\ m_0 x + m_1 - m_0, & \text{se } 1 \leq x \end{cases} \tag{3.7}$$

[5] Ver (Chua, 1992) para um relato contendo os passos envolvidos no design do circuito.

[6] A propósito, a prova da existência do atrator Lorenz foi dada por (Tucker, 1999) (em (Stewart, 2000) tem-se uma discussão geral sobre a prova de Tucker, enquanto que em (Viana, 2000) encontra-se uma exposição mais técnica).

Outrossim, as mesmas equações podem ser representadas explicitamente como funções dos componentes, voltagens e resistências:

$$\begin{cases} v_1' = \dfrac{1}{R * C_1}((v_2 - v_1) - R * g(v_1)) \\ v_2' = \dfrac{1}{R * C_2}(v_1 - v_2 + R * i_L) \\ i_L' = \dfrac{1}{L}(-v_2) \end{cases} \tag{3.8}$$

$$g(v_1) = \begin{cases} m_0 v_1 + (m_0 - m_1)E_1 & , \text{ se } v_1 \leq -E_1 \\ m_1 v_1 & , \text{ se } -E_1 < v_1 < E_1 \\ m_0 v_1 + (m_1 - m_0)E_1 & , \text{ se } E_1 \leq v_1 \end{cases} \tag{3.9}$$

Tal como os dois sistemas anteriores, o circuito de Chua produz, com a escolha adequada de parâmetros, soluções caóticas que quando plotadas produzem um atrator caótico (estranho) (Figura 3.6) (Matsumoto et al., 1985). O desenvolvimento do atrator de Chua pode ser visto na Figura 3.7.

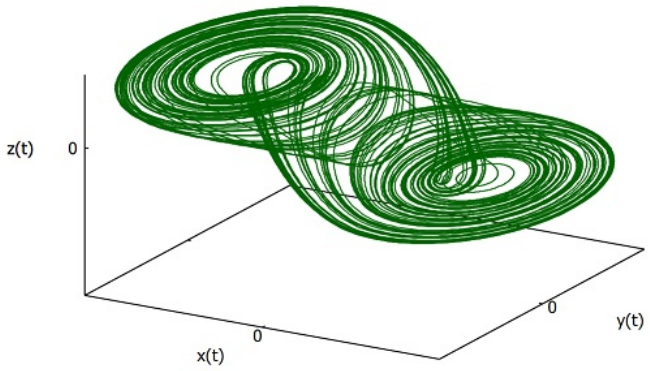


Figura 3.6: Atrator de Chua, utilizando $\alpha = 15.6$, $\beta = 28$, $m_0 = -1.143$, $m_1 = -0.714$ e condição inicial $(0.7, 0.0, 0.0)$.

3.1.1.4 O Mapa Hénon

O mapa Hénon (Hénon, 1976) é um modelo simplificado de uma seção de Poincaré, intentado para ser um modelo tão simples o quanto possível, mas que apresentasse as mesmas propriedades do sistema Lorenz. O modelo proposto tinha entre seus objetivos: (i) tornar a exploração numérica mas rápida e precisa; (ii) fornecer um modelo que poderia se prestar mais facilmente a análise matemática.

Per si, o modelo começa considerando não a totalidade das trajetórias no espaço tridimensional, mas tão somente suas sucessivas intersecções com uma superfície de seção S, as quais são obtidas definindo-se um mapeamento $T : S \to S$, de modo que

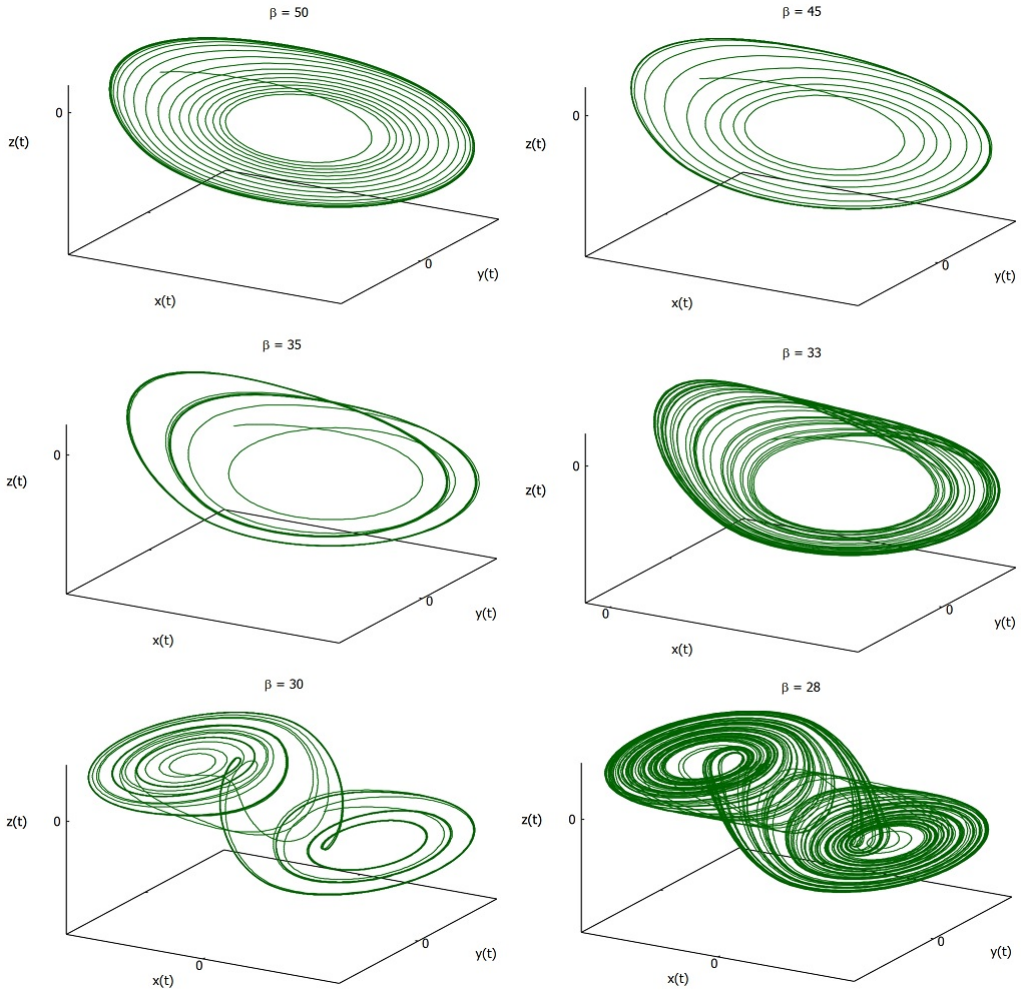


Figura 3.7: Desenvolvimento do atrator de Chua, variando-se β e mantidos constantes $\alpha = 15.6$, $m_0 = -1.143$, $m_1 = -0.714$.

dado um ponto $A \in S$, segue-se a trajetória que se origina em A até ela intersectar a superfície S novamente; este ponto é então chamado de $T(A)$[7]. Assim procedendo, a trajetória é substituída por um conjunto infinito de pontos em S, e as propriedades da trajetória são refletidas nas correspondentes propriedades do conjunto de pontos. Feito isso, a computação do mapeamento requer ainda a integração numérica de equações diferenciais. Assim, o próximo passo, cujo escopo é simplificar a computação, consiste em definir explicitamente o mapeamento T por meio de equações, produzindo $T(A)$ sempre que se é conhecido A. Finalmente, o mapeamento T é especificado de modo a ser esticado em uma direção e dobrado sobre si mesmo. Mais especificamente, a transformação

$$T' : x' = x, y' = y + 1 - ax^2 \tag{3.10}$$

[7]O mapeamento T é conhecido como *mapa de Poincaré*.

dobra uma área (Figura 3.8-(a)) produzindo a Figura 3.8-(b). Já a transformação

$$T'' : x'' = bx', y'' = y' \tag{3.11}$$

produz uma contração ao longo do eixo x, produzindo a Figura 3.8-(c). Finalmente, a transformação

$$T''' : x''' = y'', y''' = x'' \tag{3.12}$$

faz com que a orientação volte a ser alinhada em relação ao eixo x (Figura 3.8-(d)).

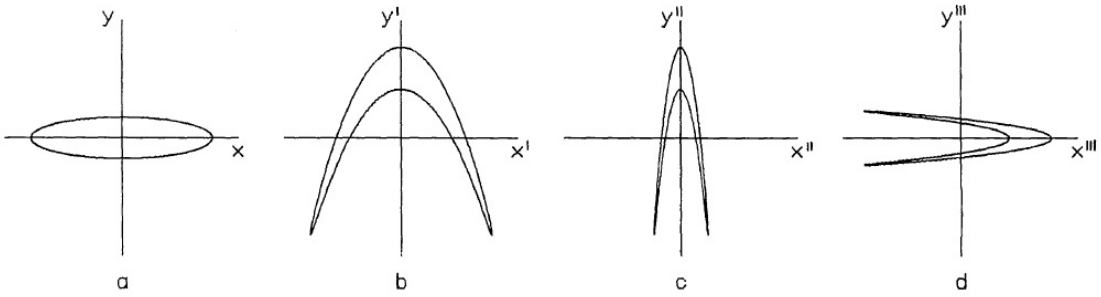


Figura 3.8: A área inicial a é mapeada por T' em b, então por T'' em c, e finalmente por T''' em d (Hénon, 1976).

O mapeamento T é então definido para ser $T = T'T''T'''$, de modo que se pode escrever

$$T : \begin{cases} x_{i+1} = y_i + 1 - ax^2 \\ y_{i+1} = bx_i \end{cases} \tag{3.13}$$

Digno de nota é observar que o mapeamento T é inversível:

$$T^{-1} : \begin{cases} x_i = b^{-1} y_{i+1} \\ y_i = x_{i+1} - 1 + ab^{-2} y_{i+1}^2 \end{cases} \tag{3.14}$$

o que atesta que que T é um mapeamento de bijetor do plano em si mesmo.

O mapa depende de dois parâmetros, a e b. Quando $a = 1.4$ e $b = 0.3$, qualquer ponto inicial contido no plano tende a um conjunto de pontos conhecido como atrator Hénon (Fig. 3.9) ou diverge para o infinito.

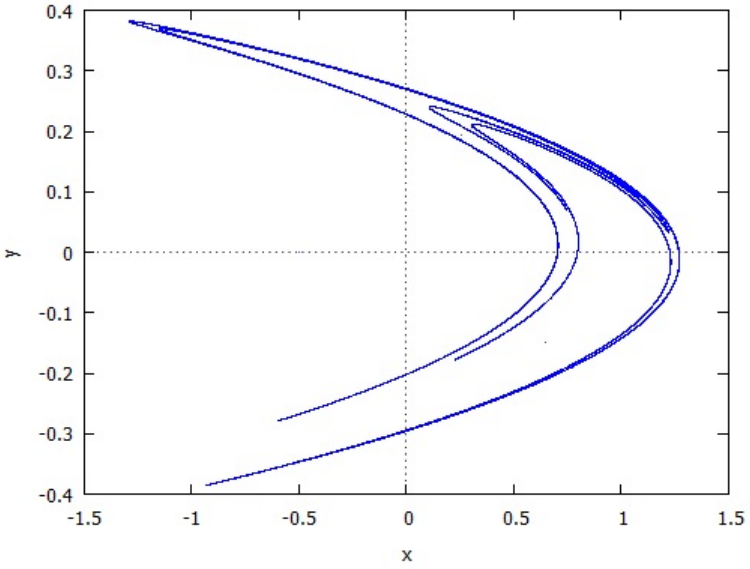


Figura 3.9: Atrator Hénon.

Para outros valores de a e b, o mapa pode ser caótico, intermitente ou convergir para uma órbita periódica. Nas Figuras 3.10-(a) e 3.10-(b) são exibidos mapeamentos para diferentes valores de a e b.

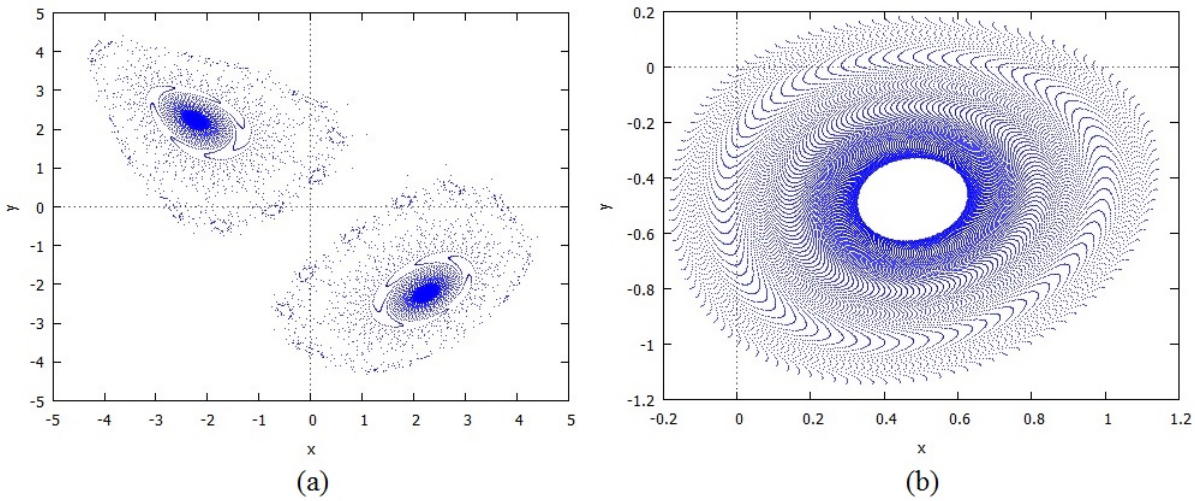


Figura 3.10: Mapa Hénon para: (a) $a = 0.2$ e $b = 0.9991$; (b) $a = 0.2$ e $b = -0.9991$).

3.1.1.5 Mapa Logístico

O mapa logístico[8] se constitui em um dos mais estudados sistemas unidimensionais capazes de vários regimes dinâmicos, incluindo caos, sendo a derivação discreta análoga de um modelo contínuo de crescimento populacional, a equação logística (introduzida por Pierre François Verhulst (Verhulst, 1838)).

[8]O artigo de Robert M. May (May, 1976) disserta em grandes detalhes sobre as principais características do mapa logístico.

Per si, o mapa logístico tem sua formulação dada pela equação de diferenças

$$x_{n+1} = r.x_n.(1 - x_n) \tag{3.15}$$

onde r é chamado de parâmetro de controle. Diferentemente do mapa Hénon, o mapa logístico não é inversível.

De forma mais técnica, o mapa logístico constitui-se em uma família de funções quadráticas, $F(r, x) = rx(1-x)$; basta variar o valor do parâmetro r e tem-se uma nova função (ou mapeamento).

Da definição do mapa logístico, pode ser verificado que para um dado r, os pontos fixos[9] de $F(r, x)$ são soluções constantes da forma $x_n = c$. Quando calculados, estes estados estacionários apresentam-se como $c = \dfrac{r-1}{r}$ e $c = 0$.

Por sua vez, o estudo da dinâmica do mapa logístico pode ser feito nos aspectos analítico, numérico e qualitativo (análise gráfica).

3.1.2 Propriedades de Sistemas Dinâmicos

3.1.2.1 Órbitas

Já foi dito neste capítulo que um dos interesses maiores concernentes a sistemas dinâmicos era conhecer o destino das órbitas para um dado valor inicial sob interação, ou de forma mais pomposa, determinar o comportamento assintótico das órbitas. Como também já mencionado, as órbitas podem: (a) tender a um valor fixo; (b) apresentar um comportamento periódico; (c) tender ao infinito; (d) apresentar um comportamento aperiódico. O mapa logístico apresenta todos os comportamentos mencionados.

Uma das formas de se estudar o comportamento das órbitas é representando-as por meio de uma *série temporal* ou por meio dos chamados *diagramas de teia de aranha*. Em particular, os diagramas de teia de aranha permitem o estudo da emergência de pontos periódicos "atratores" com base na forma particular da equação de diferença: $x_{n+1} = f(x_n)$. Mais especificamente, se as coordenadas dos pontos no plano representam dois valores consecutivos na órbita de x_0, o ponto $(x, y) = (x_n, x_{n+1})$ pode ser obtido graficamente como a intersecção da linha vertical $x = x_n$ e o gráfico da função $y = f(x)$. Uma vez que x_{n+1} é conhecido, para poder obter x_{n+2} deve-se fazer com que x_{n+1} assuma o papel de x_n no passo anterior. Desta feita, toma-se a intersecção da linha horizontal $y = x_{n+1}$ com a diagonal $y = x$. Agora, $x_{n+2} = f(x_{n+1})$, e iterando o processo um número

[9] Pontos fixos são valores que permanecem constantes sob iteração.

determinado de vezes, obtém-se um segmento da órbita de x_0.

Os quatro exemplos seguintes mostram por meio séries temporais (e diagramas de teia de aranha) o comportamento do mapa logístico para diversos valores de r; o valor inicial é o mesmo para todos: $x_0 = 0.8886589561406515$.

Exemplo 1: A Figura 3.11 exibe as 25 primeiras iterações do mapa logístico quando $r = 0.25$, podendo se constatar que a órbita de x_0 tende a um *valor fixo* (neste caso, zero).

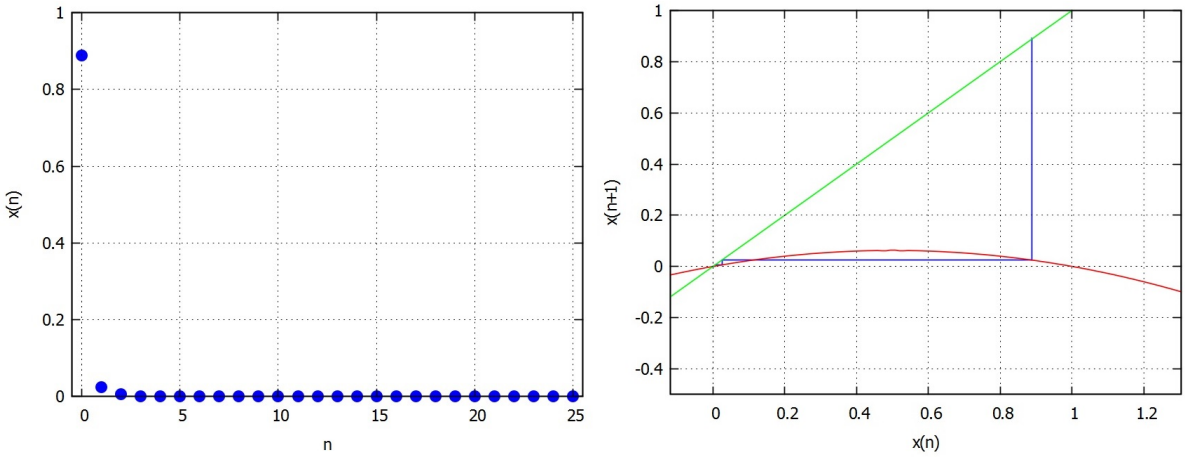


Figura 3.11: Gráfico da órbita de $x_0 = 0.8886589561406510$ para $f(x) = 0.25x(1-x)$.

Exemplo 2: A Figura 3.12 exibe as 25 primeiras iterações do mapa logístico quando $r = 1.3$, e tal como no exemplo precedente, pode ser visto que a órbita de x_0 tende a um valor fixo.

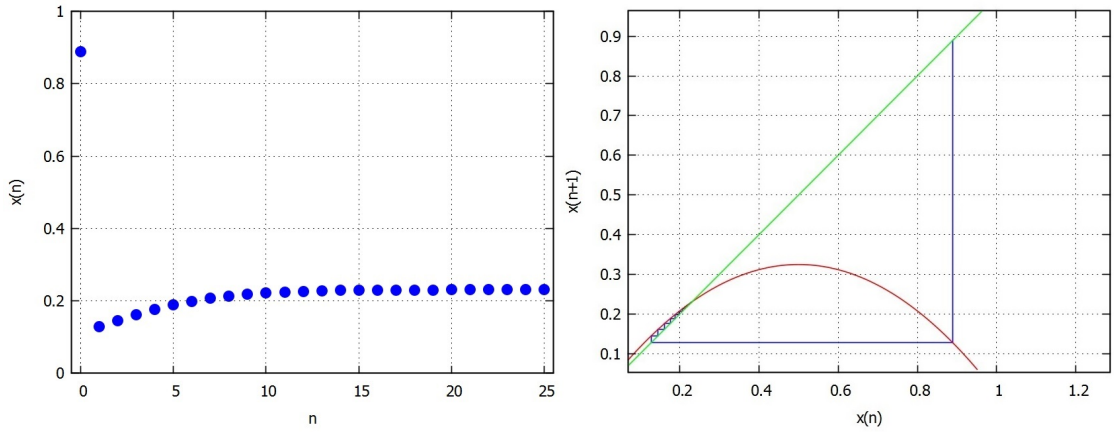


Figura 3.12: Gráfico da órbita de $x_0 = 0.8886589561406510$ para $f(x) = 1.3x(1-x)$.

Exemplo 3: A Figura 3.13 exibe as 70 primeiras iterações do mapa logístico quando $r = 3.5$, na qual se pode observar um transiente e um ciclo de período quatro.

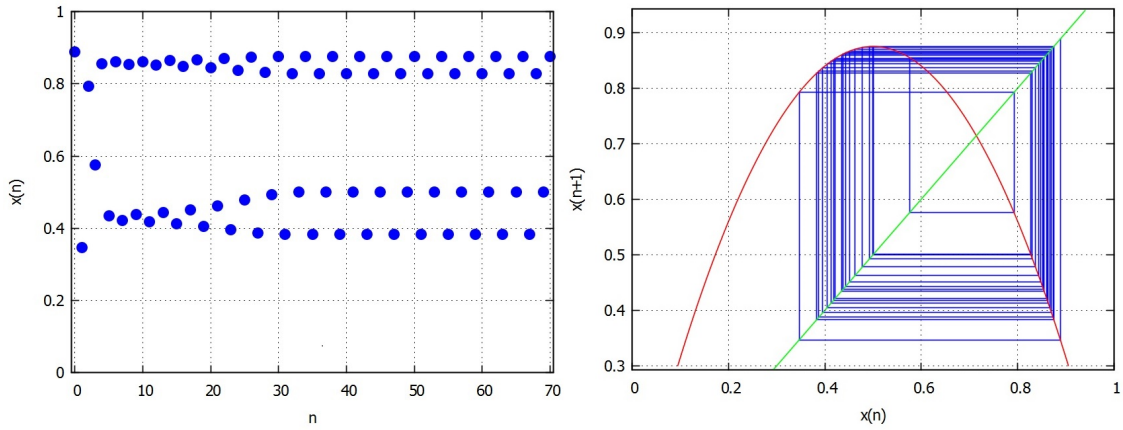


Figura 3.13: Gráfico da órbita de $x_0 = 0.8886589561406510$ para $f(x) = 3.5x(1-x)$.

Exemplo 4: A Figura 3.14 exibe as 75 primeiras iterações do mapa logístico quando $r = 3.95$, na qual se pode observar a inexistência de comportamento periódico.

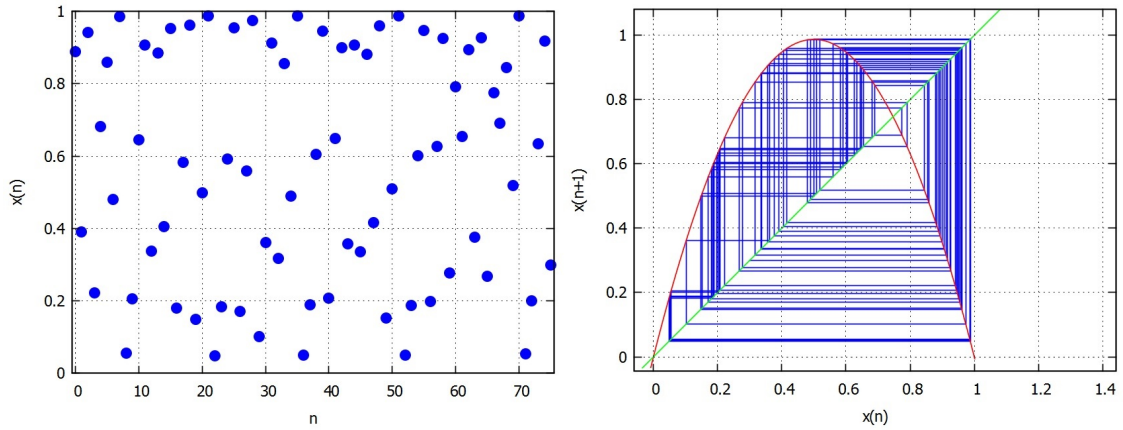
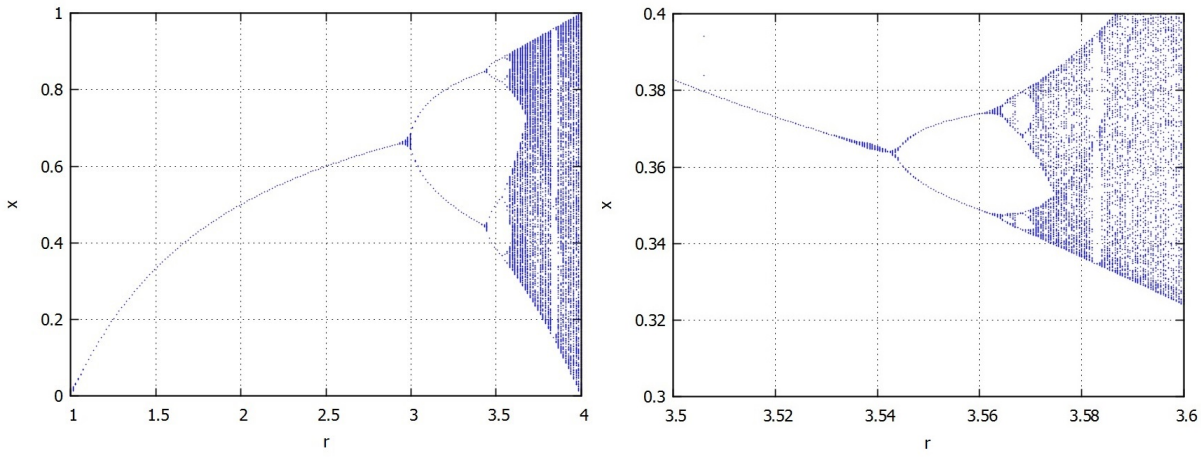


Figura 3.14: Gráfico da órbita de $x_0 = 0.8886589561406510$ para $f(x) = 3.95x(1-x)$.

Os exemplos dados mostram que o comportamento das órbitas para o mapa logístico depende do parâmetro r. De fato, a partir de $r > 3$, o número de órbitas periódicas aumenta no que se convenciona chamar de *cascata de duplicação de período*.

É possível estudar a mudança na estrutura das órbitas periódicas usando os assim chamados *diagramas de bifurcação*. Para o mapa logístico, representa-se os valores do parâmetro r no eixo horizontal, marcando-se para cada um destes valores os correspondentes pontos periódicos "atratores" (ver Fig. 3.15-(a)).

O diagrama da Figura 3.15-(a) tem uma propriedade excepcional: ele é auto-similar (uma propriedade comum aos fractais). Isto significa que após uma mudança de escala, a figura resultante tem a mesma estrutura que a original (ver Fig. 3.15-(b)).



(a) (b)

Figura 3.15: Diagrama de bifurcação para o mapa logístico.

Uma medida da auto-similaridade de um conjunto é sua *dimensão fractal* que, diferente da dimensão de um espaço vetorial, pode ser um número não inteiro. A cascata de duplicação de período e a aparência dos conjuntos com dimensão fractal são usualmente sinais do comportamento caótico.

Resumidamente, o comportamento do mapa logístico segue as seguintes regras:

- Se $0 < r \leq 1$ e $x_0 \in [0, 1]$, as órbitas tendem para zero, independente da condição inicial.
- Se $1 < r \leq 2$ e $x_0 \in]0, 1[$, as órbitas se aproximam rapidamente do valor $(r-1)/r$, independentemente da condição inicial.
- Se $2 < r \leq 3$ e $x_0 \in]0, 1[$, as órbitas tendem novamente para o valor $(r-1)/r$, mas de um modo oscilante (talvez após um transiente curto). A taxa de convergência é linear, exceto para $r = 3$, onde a taxa de convergência é muito lenta, de fato ela é sublinear.
- Se $3 < r < 1 + \sqrt{6}$ (com $1 + \sqrt{6} \approx 3.45$), as órbitas oscilam entre dois valores, quase independentemente das condições iniciais. Estes dois valores, que dependem de r, são ditos terem período primário dois.
- Se $3.45 < r < 3.54$ (aproximadamente), as órbitas oscilam entre quatro pontos periódicos para quase todas as condições iniciais.
- Se r aumenta mais do que 3.54, para quase todas as condições iniciais as órbitas oscilam entre 8 pontos periódicos, então 16, 32, etc. O tamanho dos intervalos formados pelos valores do parâmetro produzindo oscilações torna-se pequeno, e o quociente do tamanho de dois intervalos consecutivos de duplicação de período se aproxima da assim chamada *constante de Feigenbaum* δ =

4.669201609102990671853203820466201617258185577475768632745 . . . Este comportamento é chamado de cascata de duplicação de período.

A propósito, a constante de Feigenbaum (Feigenbaum, 1978) representa uma característica universal para diagramas de bifurcação em mapas não lineares, a saber, ela é a razão limite de cada intervalo entre uma bifurcação e a seguinte em um mapeamento unidimensional $x_{n+1} = f(x)$. Mais especificamente, considere que o n-ésimo período de bifurcação ocorre em $\lambda = \lambda_n$, então

$$\lim_{n\to\infty} \frac{\lambda_{n-1} - \lambda_{n-2}}{\lambda_n - \lambda_{n-1}} = 4.669201609\ldots \tag{3.16}$$

A tabelas 3.1 e 3.2 mostram os valores dos parâmetros de bifurcação para o mapa logístico e para o mapa Hénon (Alligood et al., 1996, p. 504).

Tabela 3.1: Valores de parâmetros de bifurcação para o mapa logístico.

Período	**Parâmetro λ**	**Razão**
2	3.0000000	-
4	3.4494896	-
8	3.5440903	4.7514
16	3.5644073	4.6562
32	3.5687594	4.6683
64	3.5696916	4.6686
128	3.5698913	4.6692
256	3.5699340	4.6694

Tabela 3.2: Valores de parâmetros de bifurcação para o mapa Hénon.

Período	**Parâmetro λ**	**Razão**
2	1.2675000	-
4	1.8125000	
8	1.9216456	4.9933
16	1.9452006	4.6337
32	1.9502644	4.6516
64	1.9513504	4.6630
128	1.9515830	4.6678
256	1.9516329	4.6688

3.1.2.2 Sistema Caótico

Um sistema dinâmico, cuja evolução é descrita por uma função $f : V \to V$, é dito ser caótico em V (Devaney, 2003, p. 50) se:

(a) Ele tem dependência sensível sobre as condições iniciais: uma pequena variação na condição inicial x_0 de uma órbita pode produzir, em longo prazo, outra órbita muito distante da original.

(b) Pontos periódicos são "densos" no espaço de fase do sistema: em geral o conjunto S é denso no conjunto $V \subset \mathbb{R}$ se para todo ponto $p \in V$ e qualquer $\epsilon > 0$, o intervalo $]p - \epsilon, p + \epsilon[$ intersecta S.

(c) Ele tem a propriedade de mistura (diz-se também que o sistema é topologicamente transitivo): para quaisquer dois intervalos $J, K \subset V$ existem pontos de J cujas órbitas eventualmente entram em K, isto é, existe $n > 1$ tal que $f_n(J) \cap K \neq 0$ com $f^n = f \circ \cdots \circ f$. Esta propriedade é satisfeita se, e somente se, o sistema tem uma órbita $\{f_n(x_0) : n \in \mathbb{N} \cap \{0\}\}$ que é densa em I.

Estas propriedades podem ser ilustradas por meio do mapa logístico.

(a) Dependência sensível sobre as condições inciais: Considere o mapa logístico $f(x) = 4x(1 - x)$. O valor $x_0 = \frac{3}{4}$ é um ponto fixo, dado que a órbita do mesmo sob iteração se mantém invariante, conforme pode ser visto na Figura 3.16-(a). O que acontece quando se escolhe um valor inicial próximo a 3/4, digamos $x_0 = 3/4 + \epsilon$, onde $\epsilon = 9.13809599612896 \times 10^{-13}$, por exemplo? A resposta é dada na Figura 3.16-(b), onde verifica-se que após menos de 40 iterações a órbita de $3/4+\epsilon$ diverge da órbita de 3/4.

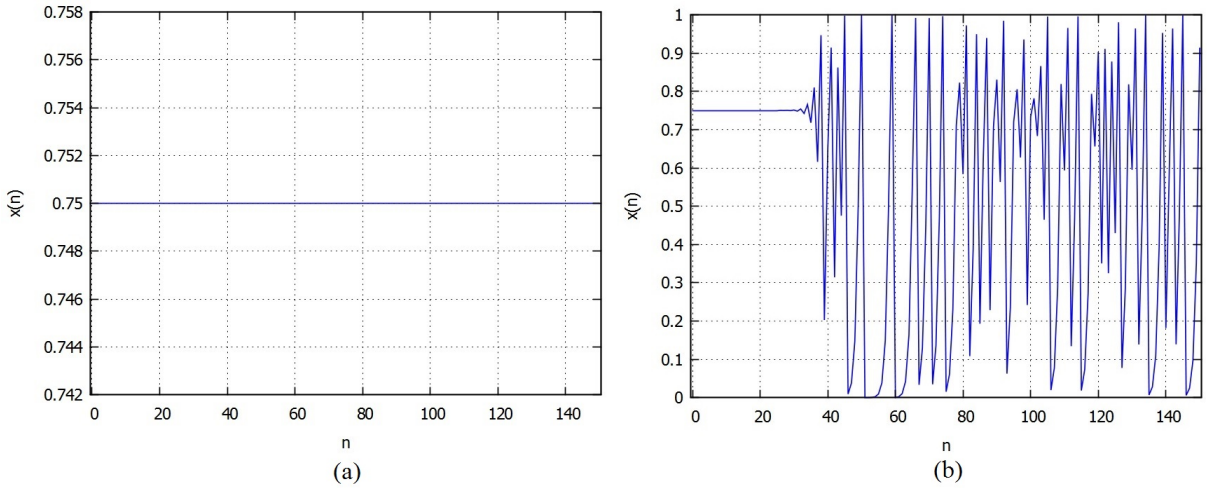


Figura 3.16: Dependência sensível sobre as condições iniciais.

(b) Pontos periódicos são "densos" no espaço de fase do sistema: Considere novamente o mapa logístico. Pode-se verificar que o conjunto de pontos periódicos é denso no intervalo $I = [0, 1]$ por meio da Figura 3.15-(a), onde todo o lado direito do quadrado é "preenchido" com pontos periódicos.

(c) Propriedade de mistura (transitividade topológica): Considere outra vez o mapa logístico $f(x) = 4x(1-x)$ e faça $x_0 = 0.2$, tomando agora 50000 iterações. Pode-se constatar (cfm. Figura 3.17) que esta órbita preenche o espaço de fase.

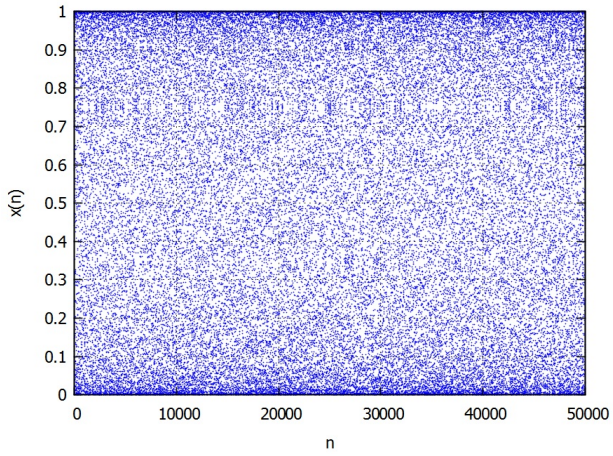


Figura 3.17: Mistura - Transitividade topológica.

3.1.2.3 Expoentes de Lyapunov

Foi visto na subseção anterior que uma característica de sistemas caóticos é a dependência sensível sobre as condições iniciais. Contudo, existe um método para medir esta propriedade? A resposta a esta inquirição é positiva, sendo encontrada no conceito de expoentes de Lyanunov, cujo termo deriva de Aleksandr Mikhailovich Lyapunov, matemático russo que em sua tese de doutoramento de 1892 (Liapounoff, 1907)(Lyapunov, 1992) sobre estabilidade de uma massa de fluido em rotação, introduziu dois métodos, o primeiro dos quais foi posteriormente chamado por esta designação.

A ideia básica por detrás dos expoentes de Lyapunov consiste em escolher uma órbita fixa e compará-la com outras órbitas que possuem condições iniciais próximas, medindo então a distância entre elas com um fator da forma $e^{(\lambda t)}$, onde λ é o expoente de Lyapunov. Se ele é positivo, as órbitas divergem assintoticamente e existe sensibilidade às condições iniciais (tanto maior quando maior é λ). Se o expoente λ é negativo, as órbitas devem se aproximar assintoticamente uma da outra e não existe sensibilidade às condições iniciais. Um valor de $\lambda = 0$ indica que a órbita considerada é um ponto fixo estável.

A ideia acima, de forma mais técnica, é considerar um sistema dinâmico $x_{n+1} = f(x_n)$, com $f : \mathbb{R} \to \mathbb{R}$ sendo derivável, com exceção de um número finito de pontos, e tomar duas órbitas, uma começando no ponto x_0 e outra começando no ponto $x_0 + \epsilon$. Após n interações os pontos nas órbitas serão $f^n(x_0)$ e $f^n(x_0 + \epsilon)$, onde $f^n = f \circ \ldots \circ f$, de modo que a distância entre as órbitas é $|f_n(x_0 + \epsilon) - f_n(x_0)|$, o que pode ser escrito

como

$$|f_n(x_0+\epsilon) - f_n(x_0)| = \epsilon e^{n\lambda(x_0)} \tag{3.17}$$

onde $\lambda(x_0) \in \mathbb{R}$ é chamado de *expoente de Lyapunov* no ponto x_0.

Resolvendo a equação para $\lambda(x_0)$, tem-se:

$$\lambda = \frac{\ln(\frac{|f_n(x_0+\epsilon) - f_n(x_0)|}{\epsilon})}{n}. \tag{3.18}$$

Agora, tomando limites com $\epsilon \to 0$ e $n \to \infty$, tem-se:

$$\lambda(x_0) = \lim_{n\to\infty} \frac{1}{n} \ln \left| \frac{df_n}{dx}(x_0) \right|. \tag{3.19}$$

Dado que

$$\frac{df}{dx}(x_0) = \prod_{i=0}^{n-1} f'(x_i), \tag{3.20}$$

então

$$\lambda(x_0) = \lim_{n\to\infty} \frac{1}{n} \ln \left| \prod_{i=0}^{n-1} f'(x_i) \right| = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} \ln |f'(x_i)|. \tag{3.21}$$

A última equação permite o cômputo de exponentes de Lyapunov numericamente, aproximando o limite por uma soma finita para n for suficientemente grande.

3.1.2.4 Inversibilidade

De passagem foi dito (ainda que um tanto quanto implicitamente) que o mapa Hénon é inversível, ao passo de que o mapa logístico não o é. Para entender o porque desta afirmação, algumas definições e explicações se fazem necessárias.

Definição 3.1.1 (Homeomorfismo)**.** *Um mapeamento $M : X \to X$ é dito ser um homeomorfismo se ele é bijetor (ou seja, injetor e sobrejetor), contínuo e seu mapeamento inverso M^{-1} também é contínuo.*

Definição 3.1.2 (Difeomorfismo)**.** *Um mapeamento $M : X \to X$ é dito ser um difeomorfismo se M é um homeomorfismo que é diferenciável, sendo seu mapeamento inverso M^{-1} também diferenciável.*[10]

Definição 3.1.3 (Órbita adiante/reversa)**.** *Dado um mapa $x_{n+1} = f(x_n)$, define-se como órbita adiante do estado x deste mapa, aqui denotado por $\mathcal{O}^+(x)$, o conjunto de pontos*

$$\mathcal{O}^+ = \{x, f(x), f^2(x), \cdots\} \tag{3.22}$$

[10] Uma observação interessante é que se existe um difeomorfismo entre dois sistemas dinâmicos, então eles são equivalentes.

Se o mapeamento é um homeomorfismo, então define-se como órbita reversa do estado x deste mapa, aqui denotado por $\mathcal{O}^-(x)$, o conjunto de pontos

$$\mathcal{O}^- = \{x, f^{-1}(x), f^{-2}(x), \cdots\} \tag{3.23}$$

A razão para a distinção entre estes dois tipos de órbitas (adiante/reversa) é que nem sempre se é possível seguir órbitas reversas (voltar no tempo), caso que acontece com mapeamentos não inversíveis. Mais especificadamente, um mapa $f(x_n)$ é dito ser inversível se, para um dado estado x_{n+1}, existe uma única pré-imagem x_n tal que $x_n = f^{-1}(x_{n+1})$, onde f^{-1} é a inversa de f.

Considere agora o mapa logístico. A partir da relação $x_{n+1} = rx_n(1-x_n)$ obtém-se:

$$x_{n+1} = rx_n(1-x_n) \Leftrightarrow x_{n+1} = rx_n - rx_n^2 \Leftrightarrow rx_n^2 - rx_n + x_{n+1} = 0 \Leftrightarrow$$

$$x_n = \frac{r \pm \sqrt{r^2 - 4rx_{n+1}}}{2r} \tag{3.24}$$

o que significa que existem duas pré-imagens[11] para x_{n+1}, o que confirma que o mapa logístico claramente não é inversível.

Considere agora o mapa Hénon

$$\begin{cases} x_{n+1} = 1 - ax_n^2 + y_n \\ y_{n+1} = bx_n \end{cases} \tag{3.25}$$

efetuando uma substituição:

$$\begin{cases} x_{n+1} = f(x_n) - Jy_n \\ y_{n+1} = bx_n \end{cases} \tag{3.26}$$

onde $f(x_n) = a - x_n^2$ e $J = -1$, pode-se expressar as pré-imagens como:

$$\begin{cases} x_n = \dfrac{y_{n+1}}{b} \\ y_n = J^{-1}\left[f\left(\dfrac{y_{n+1}}{b}\right) - x_{n+1}\right] \end{cases} \tag{3.27}$$

o que mostra que o mapa Hénon é inversível.

[11] Com exceção do ponto $x = 0.5$. Por quê? Porque o ponto $x = 0.5$ é o vértice da parábola, que é a curva que representa um mapeamento quadrático (como é o caso do mapa logístico), e como tal, é um ponto único.

3.1.2.5 Tipos de Atratores

Um atrator é um subconjunto do espaço de fase do sistema dinâmico para o qual as órbitas tendem conforme o sistema evolui. Existem vários tipos de atratores, dependendo do tipo de sistema e de sua dimensão, mas genericamente os mesmos podem ser:

- Um ponto fixo.
- Um conjunto finito de pontos.
- Uma curva.
- Uma superfície.
- Um atrator estranho.

Como exemplo de atratores que são pontos fixos, pode-se considerar o destino das órbitas para o mapa logistico quando $r \leq 3$ (Figura 3.18)[12]. Considere também o sistema Lorenz com os parâmetros $\sigma = 10$, $\rho = 9$ e $\beta = 2.66$, condição inicial $(0.5, 1.4, 2)$, representado na Figura 3.19[13]. A partir da mesma, constata-se que o estado do sistema evolui para um único ponto, a saber, o valor numericamente computado $(4.613025037868326, 4.61302503786833, 8.0)$.

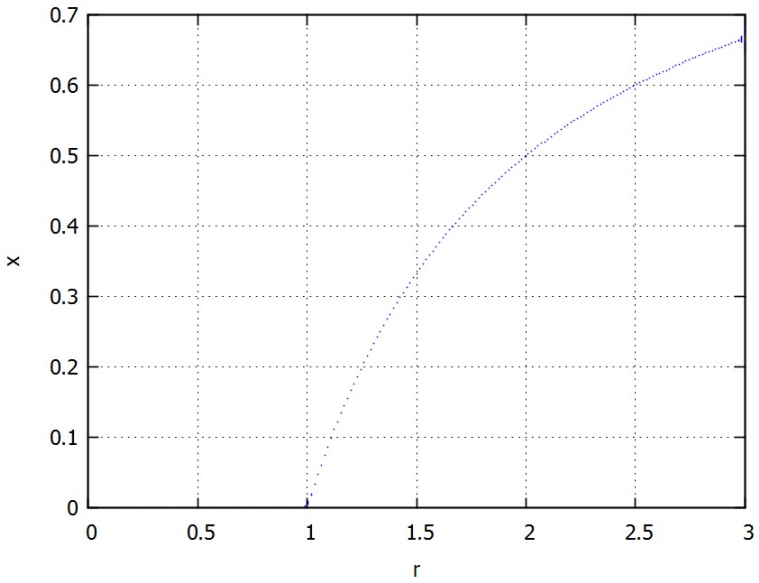


Figura 3.18: Diagrama de bifurcação para o mapa logístico exibindo atratores pontuais.

[12] O leitor mais cuidadoso irá observar os atratores se constituem no ponto $x_n = 0$ quando $r \in (0, 1)$.

[13] Como são mostrados todos os pontos da trajetória a partir da condição inicial, verifica-se que a mesma "espirala" em direção ao atrator pontual.

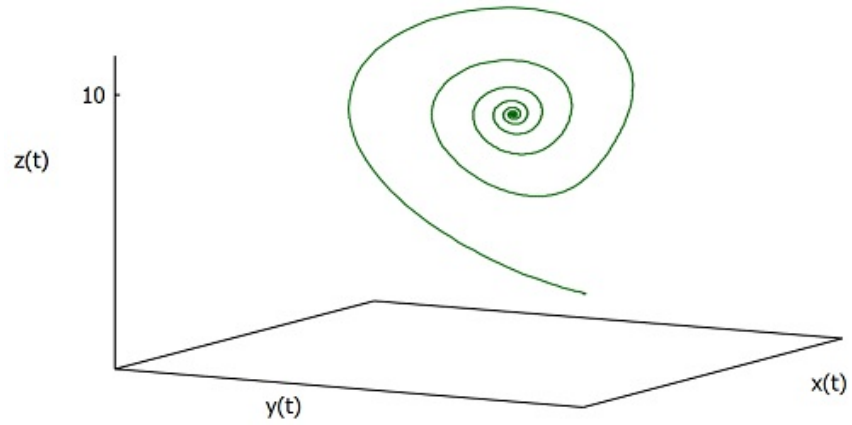


Figura 3.19: Atrator pontual no sistema Lorenz.

Como um exemplo final, considere o pêndulo amortecido, cujo espaço de fase é representado pelas seguintes equações diferenciais:

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = -\dfrac{\beta}{m}\omega - \dfrac{g}{l}\sin(\theta) \end{cases} \tag{3.28}$$

Na Figura 3.20[14] pode-se constatar que o ponto $(0, 0)$ constitui-se um ponto de equilíbrio, ou em outras palavras, um ponto fixo.

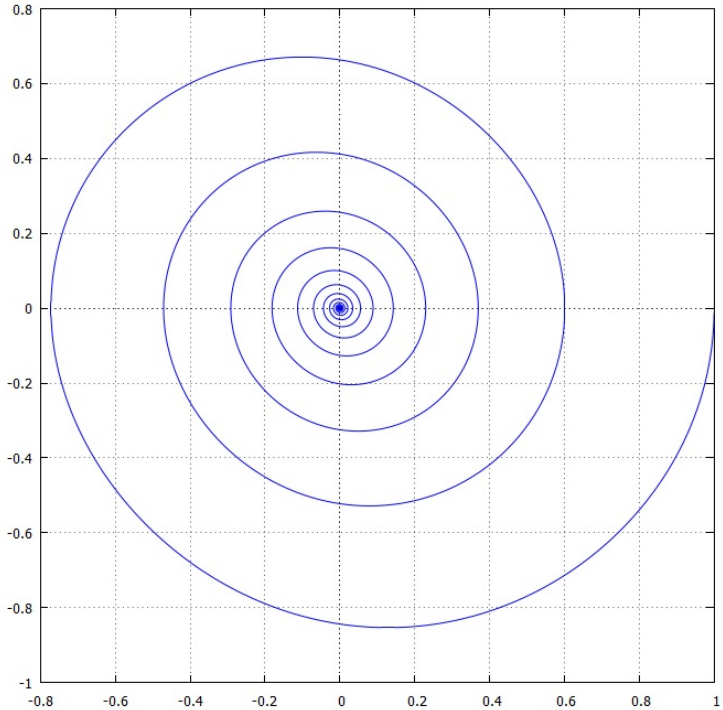


Figura 3.20: Atrator pontual para o pêndulo amortecido.

[14]Condições iniciais: $\theta = 1$ e $\omega = 0$; parâmetros: $\beta = 0.15$, $m = g = l = 1$.

Como exemplo de atratores que são conjuntos de pontos finitos, considere o mapa Hénon com os seguintes parâmetros $a = 1.25$ e $b = 0.3$, representado na Figura 3.21, na qual são observados 7 pontos atratores (a bem da verdade, são exibidos bem mais do que 7 pontos, contudo, isso acontece porque nesta representação os pontos atratores são exibidos juntamente com vários transientes, pontos pela qual o sistema passa antes de alcançar o atrator. Mas a mesma situação pode ser tornada mais explícita na Figura 3.22). Para o mapa logístico, considere a região do diagrama de bifurcação para $3 \leq r \approx 3.54$ (Figura 3.15-(a)) (embora deva-se destacar que após este limiar existem regiões nas quais caos e órbitas periódicas se misturam, cfm. Figura 3.23).

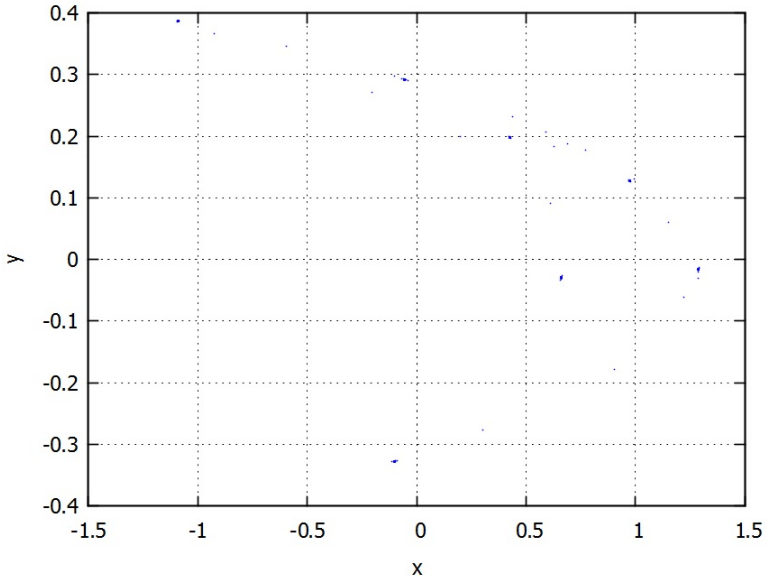


Figura 3.21: Atratores pontuais para o mapa de Hénon.

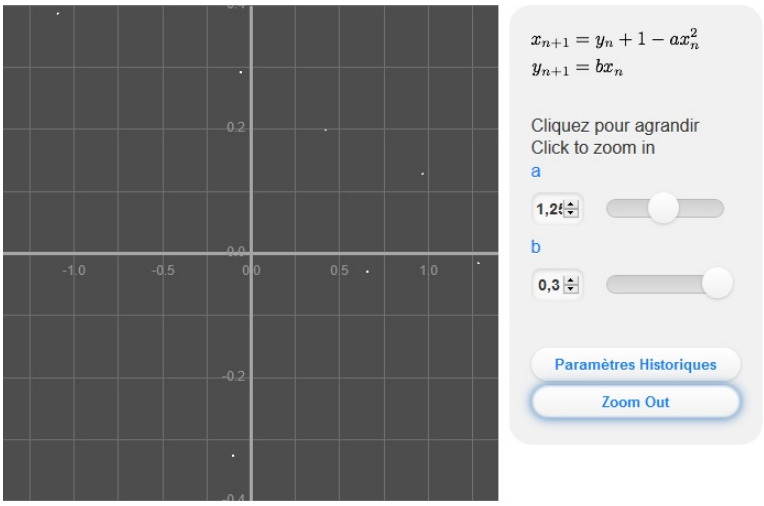


Figura 3.22: Mesmos atratores pontuais para o mapa de Hénon da figura 3.21 (ver http://experiences.math.cnrs.fr/L-attracteur-de-Henon.html).

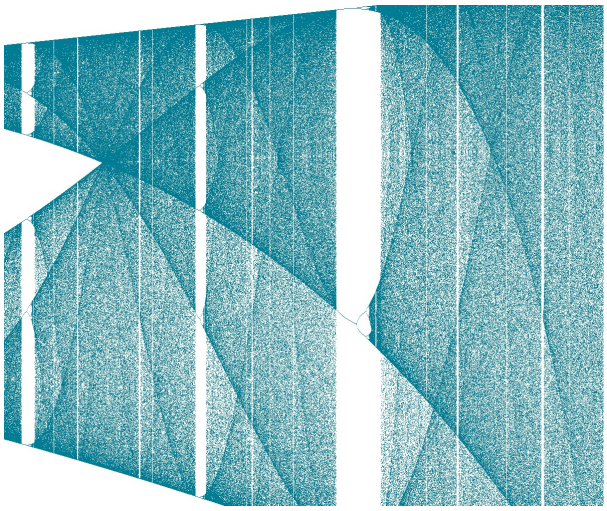


Figura 3.23: Diagrama de bifurcação para o mapa logístico ($3.62 \leq r \leq 4$) exibindo regiões entremeadas de caos e órbitas periódicas.

Como exemplo de atratores que são curvas, considere o *Brusselator* (ver Glansdorff e Prigogine (1971); Nicolis (1971)), um modelo de uma reação química autocatalítica oscilante proposto por Ilya Prigogine e colaboradores. Neste arquétipo, são consideradas duas espécies de reagentes autocatalíticos, X e Y. As equações diferenciais são:

$$\begin{cases} \dot{X} = 1 - (b+1)X + aX^2Y \\ \dot{Y} = bX - aX^2Y \end{cases} \tag{3.29}$$

onde: $X, Y \in \mathbb{R}$ são as concentrações adimensionais de dois reagentes; $a, b \in \mathbb{R}$ $(a, b > 0)$ são constantes.

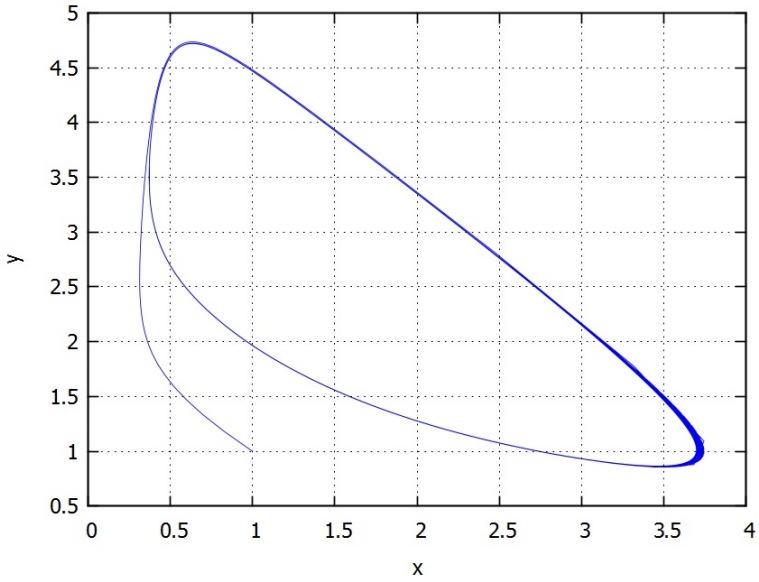


Figura 3.24: Ciclo limite para o modelo Brusselator.

O sistema em questão pode ter comportamento estável ou instável, sendo que neste último caso o sistema se aproxima de uma curva (mais especificamente, um *ciclo limite*), conforme a Figura 3.24.

Outro exemplo é a equação *replicator*, introduzida por (Taylor e Jonker, 1978) e assim batizada em (Schuster e Sigmund, 1983):

$$\dot{x}_i = x_i((Ax)_i - \mathrm{x}^T Ax) \tag{3.30}$$

onde:

- $\dot{x}_i$ é a taxa de crescimento da população i.
- x_i é a razão entre o tamanho da população i pela população total.
- $((Ax)_i - \mathrm{x}^T Ax)$ é taxa da população comparado com a população média.
- x é um vetor $N \times 1$ (N é o número da população) na qual cada linha representa a frequência de cada população.
- A é uma matriz que representa como cada população relaciona-se quando encontra a si mesma ou um membro de outra população.
- $(Ax)_i$ é a linha da matriz A multiplicada pelo vetor x, usada para encontrar o resultado médio da população i quando ela encontra outras espécies.
- $x^T Ax$ é a matriz que calcula o resultado do encontro médio.

Quando existem três espécies interagindo segundo estratégias próprias, o jogo *rock-paper-scissors*[15] pode ser utilizado. Neste *jogo* a estratégia 1 domina a estratégia 2 (na ausência da estratégia 3); similarmente, a estratégia 2 domina 3 (na ausência de 1) e a estratégia 3 domina a estratégia 1 (na ausência de 2). A matriz de recompensa para o jogo (que representa o que acontece quando cada população interage com outra) é representada como:

$$A = \begin{pmatrix} 0 & -a & b \\ b & 0 & -a \\ -a & b & 0 \end{pmatrix} \tag{3.31}$$

onde: 0 significa que nada acontece quando uma população interage consigo mesma; $-a$ ocorre quando uma população é dominada por outra (p. ex., a linha 1 tem um $-a$ na coluna 2, indicando que a população 2 é dominada pela população 1); b ocorre quando uma população domina a outra (p. ex., a linha 2 tem um b na coluna 1, indicando

[15] Literalmente isso mesmo... *pedra-papel-tesoura*... um jogo do mal.

que a população 2 é dominada pela população 1). A propósito, a e b são coeficientes dos resultados dos encontros de cada população. Fazendo $a = 1$, $b = 0.55$, e tomando $(0.25, 0.25, 0{,}5)$ como condições iniciais, o espaço de fase para as soluções deste caso específico dão origem a curva mostrada na Figura 3.25.

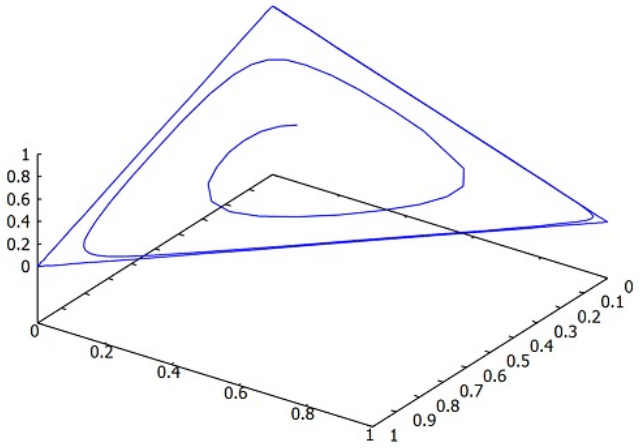


Figura 3.25: Retrato de fase para a solução para o jogo *rock-paper-scissors* usando a equação *replicator*.

Como um último exemplo de atrator que é uma curva, considere o seguinte sistema de equações diferenciais:

$$\begin{cases} \dot{x} &= -y + x(1 - x^2 - y^2) \\ \dot{y} &= x + y(1 - x^2 - y^2) \end{cases} \tag{3.32}$$

Na Figura 3.26 pode-se constatar que as órbitas de diferentes pontos convergem (ou se preferir, são "atraídas") para um curva fechada, neste caso um círculo.

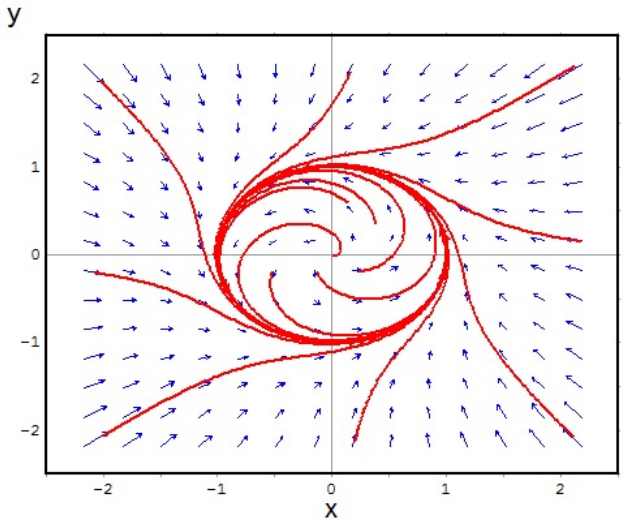


Figura 3.26: Exemplo de atrator que é uma curva (círculo).

Como exemplo de atrator que é uma superfície, considere o seguinte sistema:

$$\begin{cases} \dot{x} = -y + x(1 - x^2 - y^2) \\ \dot{y} = x + y(1 - x^2 - y^2) \\ \dot{z} = \alpha \end{cases} \tag{3.33}$$

Para este sistema o eixo z e cilindro $x^2 + y^2 = 1$ são conjuntos invariantes, e o cilindro é o conjunto atrator (Figura 3.27).

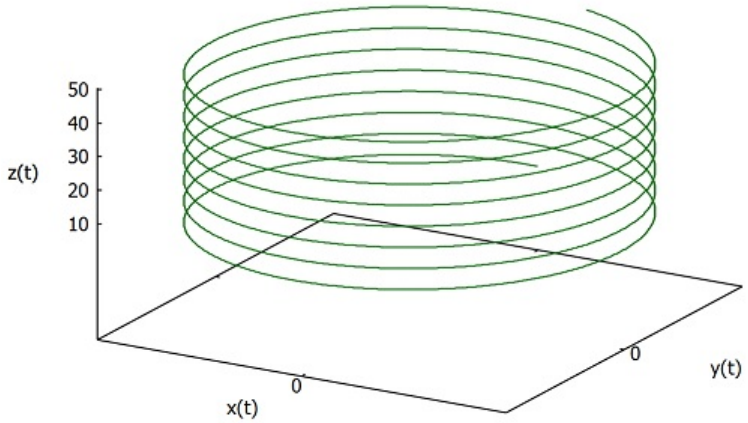


Figura 3.27: Exemplo de conjunto atrator que é uma superfície (cilindro).

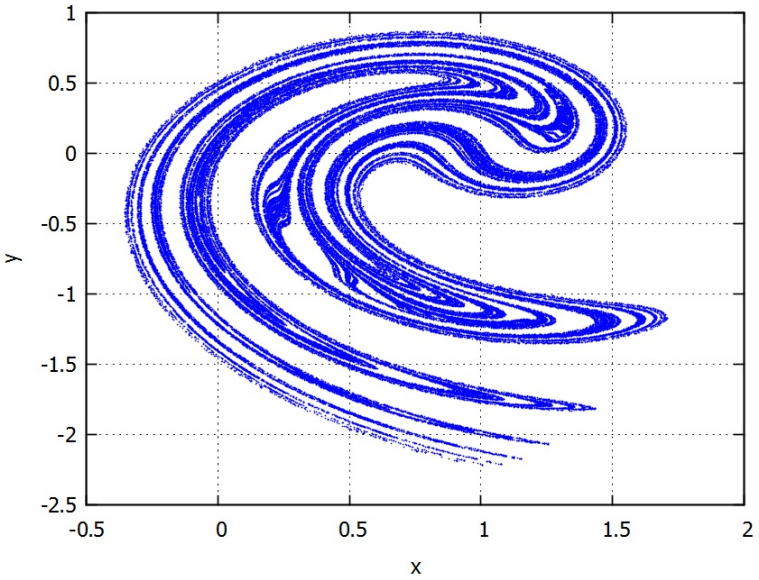


Figura 3.28: Atrator Ikeda.

Conquanto de uma forma não muito explícita, o presente capítulo apresentou alguns exemplos de atratores estranhos, a saber, o atrator Lorenz (Figura 3.1), o atrator Rössler (Figura 3.3), atrator de Chua (Figura 3.6) e o atrator Hénon (Figura 3.9), embora deva ser lembrado que existe uma ampla multiplicidade dos mesmos, por exemplo,

o atrator Ikeda, ilustrado na Figura 3.28 (Ikeda (1979), Ikeda et al. (1980)). Sendo uma característica marcante de muitos sistemas dinâmicos caóticos[16], uma pequena explanação sobre os mesmos se faz necessária. Assim, para começar, vem ao ar a pergunta: o que é um atrator estranho[17] (ver Ruelle (2006))? Antes de responder a esta inquirição, contudo, far-se-á uma breve exposição.

Inicialmente, pode ser observado que para muitos sistemas dinâmicos, órbitas próximas divergem exponencialmente com o passar do tempo (ver o item (a) da subseção 3.1.2.2 e a discussão contida na subseção 3.1.2.3) ou equivalentemente, que o sistema apresenta dependência sensível sobre as condições iniciais. Como já previamente mencionado no começo desta subseção, o conjunto do espaço de fase para o qual as órbitas tendem é chamado de atrator, e se é o caso que o mesmo apresenta dependência sensível sobre as condições iniciais, então diz-se que o atrator é caótico.

Frequentemente, também se verifica que alguns atratores possuem propriedades geométricas bem distintas: dimensão fractal não inteira, estrutura semelhante a um conjunto de Cantor e não diferenciabilidade em nenhum lugar. Quando este fato está presente, o atrator ganha a alcunha de "estranho".

Dos dois parágrafos anteriores, vê-se que se pode distinguir entre a dinâmica sobre o atrator e sua estrutura geométrica. Assim, a palavra *caótico* refere-se a dinâmica do sistema sobre o atrator, ao passo que a palavra *estranho* se refere a estrutura geométrica do atrator.

Considere o atrator caótico para o mapa Hénon. Ora, pode ser constatado que o mesmo exibe dependência sensível sobre as condições iniciais e que este atrator tem uma estrutura do tipo conjunto de Cantor (ver Figura 3.29).

Agora, como um contra-exemplo, considere o mapa logístico. Pode ser comprovado que o mesmo possui atratores caóticos para valores do parâmetro r em um conjunto de medida positiva (Jakobson, 1981). Não obstante, estes atratores consistem de um número finito de intervalos disjuntos em $[0, 1]$[18], de forma que estes atratores não são estranhos.

Neste ponto, pode ser dada uma definição de atrator estranho: um atrator é dito ser "estranho" se possui estrutura fractal (ver Boeing (2016)).

[16] Existem atratores estranhos que não são caóticos (ver Grebogi et al. (1984)).
[17] O termo estranho foi introduzido por (Ruelle e Takens, 1971).
[18] Em (Li e Yorke, 1978) é mostrado que os atratores são sempre mapas diferenciáveis por partes do intervalo unitário em si mesmo.

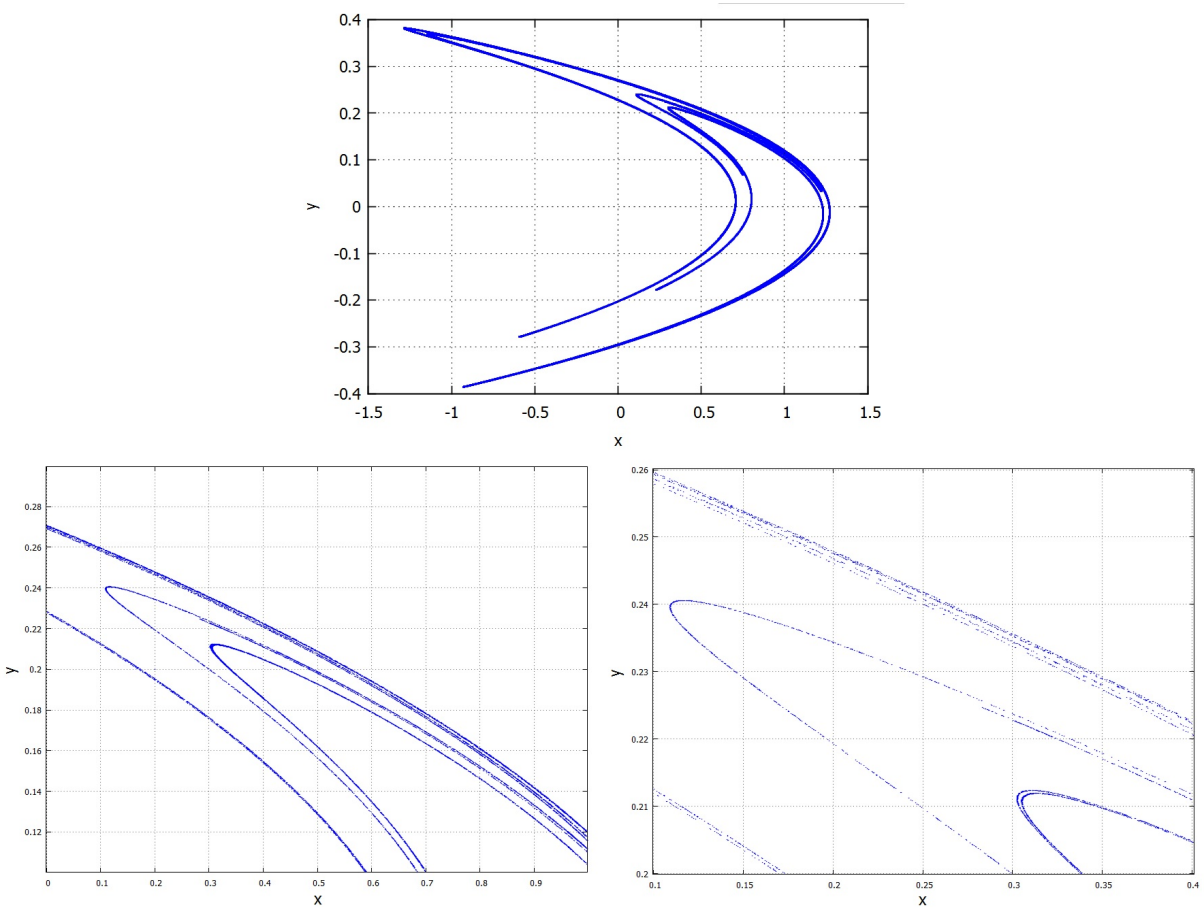


Figura 3.29: Atrator de Hénon e ampliações mostrando estrutura do tipo conjunto de Cantor.

A propósito, o conjunto de Cantor, nomeado em homenagem ao criador da Teoria dos Conjuntos, Georg Ferdinand Ludwig Philipp Cantor (1845-1918) (Cantor, 1883), é um conjunto fractal obtido através da fragmentação do intervalo $[0, 1]$. O procedimento consiste em dividir o intervalo $[0, 1]$ em três partes iguais e em seguida retirar o intervalo aberto mediano; repete-se o processo agora com os intervalos restantes, retirando-se deles o intervalo aberto mediano. Assim, após cada iteração, obtém-se dois novos intervalos (N = 2) após a divisão do intervalo original por três (r = 1/3). Repetindo-se o processo indefinidamente, chega-se ao conjunto de Cantor (Fig. 3.30).

Observe que o conjunto tem comprimento zero, posto que a cada etapa do processo seu comprimento é reduzido por um fator de 1/3. Portanto, seu comprimento quando $L(n)$ quando $n \to \infty$ (número de iterações) será:

$$L(n) = \lim_{n\to\infty} (\frac{2}{3})^n = 0 \tag{3.34}$$

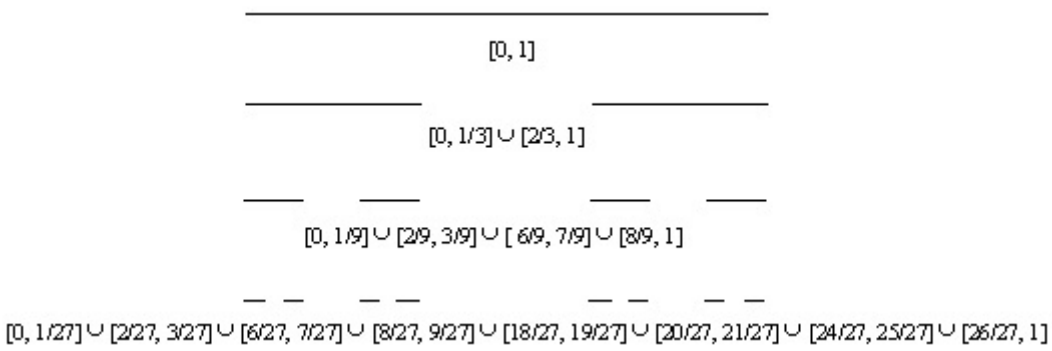


Figura 3.30: Processo de construção do conjunto de Cantor.

Contudo, apesar de ter comprimento igual a zero, o conjunto de Cantor é mais do um único ponto (na verdade uma infinidade de pontos), bem como também é bem menos do que um segmento de reta, tendo sua dimensão D igual a $\frac{\log 2}{\log 3} = 0{,}63092975...$

3.1.2.6 Dimensionalidade

Até o momento foram apresentados exemplos de sistemas dinâmicos que possuem propriedades caóticas, sendo alguns deles sistemas contínuos e outros discretos; contudo, nada foi estatuído sobre a dimensionalidade dos mesmos. Sendo razoável conjecturar que a complexidade da possível estrutura das órbitas deve ser maior quanto maior for a dimensionalidade do sistema, então é razoável (bem como pertinente) perguntar: *qual deve ser a dimensão N mínima que um sistema (dinâmico) deve possuir para que a manifestação do caos seja possível*?

Para sistemas dinâmicos contínuos (definidos por equações diferenciais) a resposta jaz no teorema de Poincaré-Bendixson (Poincaré (1881), Bendixson (1901)). Sem entrar em detalhes de ordem técnica, o teorema de Poincaré-Bendixson (ver (Hirsch et al., 2004, p. 225-227)) limita as possibilidades para uma trajetória no plano. Mais especificamente, se a trajetória é confinada a uma região fechada e limitada, sem pontos fixos, então ela deve se aproximar de uma órbita fechada (ciclo limite). Em última instância isto implica que sistemas dinâmicos contínuos de dimensão $N = 2$ não podem exibir caos. Contudo, quando a dimensão $N \geq 3$ e o sistema é não linear, caos é possível.

Agora, para sistemas dinâmicos discretos (definidos por equações de diferença), os requisitos de dimensionalidade são: (a) se o mapa é invertível, então não pode haver caos a menos que $N \geq 2$ (por exemplo o mapa Hénon é bidimensional); (b) se o mapa não é invertível, caos é possível mesmo em uma dimensão (por exemplo, o mapa logístico).

3.2 O que é realmente Caos?

O presente Capítulo introduziu algumas características do caos determinístico e como o mesmo se manifesta em sistema dinâmicos, contudo nenhuma definição "forte" (se for permitido o uso desta palavra) foi dada. Isto de certa forma é proposital, dado que mesmo entre especialistas não existe consenso do que o termo significa.

Considerando que o fenômeno de caos se manifesta em muitos campos: fluídos, plasmas, dispositivos de estado sólido, circuitos, lasers, dispositivos mecânicos, biologia, química, acústica, mecânica celeste, etc., com características e comportamentos diversos (o que poderia levar alguém a pensar: é a mãe natureza um atrator estranho? (Hastings et al. (1993)[19]), uma definição formal deveria deveria dar cabo desta totalidade de fenômenos.

Assim, na falta de uma definição formal, o termo é usado de forma um tanto quanto vaga (Smith, 1998, p. 165). O que se percebe é que a questão da definição de caos tem preponderantemente cunho de ordem filosófica, de modo que a busca pelo significado deve partir para este campo. Todavia, considerando que a presente faina não tem este intento (e nem poderia), não será realizado um aprofundamento neste aspecto.

Não sendo conveniente deixar de lado a questão, serão mencionados aqui os cinco critérios para o caos analisados em Zuchowski (2017): determinismo, transitividade, aperiodicidade, periodicidade e dependência sensível sobre as condições iniciais. A definição de caos dada na seção 3.1.2.2 (Devaney, 2003, p. 50)[20] não enfatiza determinismo como um critério para caos, mas destaca propriedades de mapeamentos determinísticos[21] (dependência sensível sobre as condições iniciais, densidade de pontos periódicos no espaço de fase e transitividade), sendo esta a única[22] usada até o presente. Contudo, outras definições caracterizam o caos em termos de mistura, expoentes de Lyapunov positivos, estocástica e atratores estranhos.

A tabela constante na página 67 de Zuchowski (2017), reproduzida a seguir (Tabela 3.3), mostra que para as várias definições dadas, vários critérios estão presentes de forma implícita, ao passo que outros estão ausentes ou são esporadicamente necessários.

[19] O artigo em questão faz uma revisão do caos e do papel encenado pelo mesmo em ecologia.
[20] Ver Banks et al. (1992) e Lardjane (2006) para alguns resultados sobre a definição dada por Devaney.
[21] Assim, determinismo aparentemente é um quesito implícito.
[22] Além daquela dada por Stewart na página **??**.

Tabela 3.3: Cinco definições de caos e seus critérios para diagnosticar caos.

	Determinismo	Periodicidade	Transitividade	DSCI	Aperiodicidade
Devaney	X	X	X	(X)	-
Mistura	X	-	X	X	-
Exp. de Lyapunov	X	-	-	X	-
Estocástica	X	-	(X)	(X)	X
Atratores Estranhos	X	X	[X]	X	[X]

Nota: Quando um critério é implicado por outro critério, mas não é afirmado explicitamente, ele é mostrado entre parêntesis; se o critério é ocasionalmente requerido, ele é mostrado entre colchetes.

Como não se pretende aqui dar cabo do assunto em tela, pode-se adotar o ponto de vista de (Zuchowski, 2017, p. 78), que afirma que a coexistência de diferentes definições de caos não parece ser uma consequência de desacordos fundamentais sobre como a noção deveria ser contextualizada, mas são uma consequência da complexidade do relacionamento dos diferentes modelos usados na teoria do caos, o que pode levar a concluir que as mesmas são usualmente aplicáveis a uma classe de específica de modelos que destacam precisamente as propriedades destes modelos que são importantes para o uso investigativo do modelo.

3.3 Considerações Finais

O presente capítulo dissertou a respeito do caos determinístico e como o mesmo se manifesta em sistemas dinâmicos, sendo apresentadas propriedades dos mesmos (embora não exaustivamente, considerando o escopo do presente trabalho e a magnitude do que seria necessário para uma apresentação holística)[23]. Finalmente, foi explanado resumidamente que uma definição única para caos até o presente momento não encontrou consenso.

[23]Faltou, por exemplo, o teorema de Takens, que dá as condições para a qual um sistema dinâmico caótico pode ser reconstruído a partir de observações do estado do sistema dinâmico.

Capítulo 4

Método

O presente Capítulo tem os seguintes os objetivos: (1) Apresentar o método proposto, descrevendo minuciosamente os passos envolvidos e os pressupostos teóricos assumidos; (2) Explicar a razão do uso de testes estatísticos de aleatoriedade e quais suítes de testes serão utilizados.

4.1 Método

O método proposto é composto por três etapas, cuja a descrição textual segue e a pictórica é dada pelo fluxograma constante na Figura 4.1. Os detalhes são:

Etapa 1: Nesta etapa é construída a sequência binária[1] S_1. Os passos são os seguintes:

- Inicialmente é dado entrada ao arquivo de *texto plano* $\mathcal{P}$.
- Dado $\mathcal{P}$, calcula-se o tamanho (em bytes) do arquivo $\#(\mathcal{P})$, multiplica-se esse valor por 64 e atribui-se o resultado à variável n (número de iterações).
- São inseridos os seguintes parâmetros: f (mapa caótico) e x_0 (valor inicial).
- Os parâmetros f, x_0 e n (número de iterações) é oferecido ao procedimento que gera a sequência binária S_1.
- A sequência binária S_1 é gravada em disco.

A sequência binária S_1 é gerada a partir das iterações do mapa caótico f. O mapa caótico, quando iterado, produz uma sequência de números x_0, $f(x_0)$, $f(f(x_0))$, $f(f(f(x_0)))\ldots$, e para um determinado mapa caótico, pode-se verificar que os valores produzidos estão limitados a um determinado intervalo. Toma-se o ponto médio x_m do intervalo. Se o valor gerado for maior ou igual ao ponto médio $x_n \geq x_m$, então é

[1] Sequências binárias são strings compostas unicamente por zeros e uns.

produzido o valor um; caso contrário, isto é, se $x_n < x_m$, então é produzido o valor 0[2] (veja a explicação sobre dinâmica simbólica contida no final da página 19 e a Figura 1.6 para entender como são geradas as sequências.).

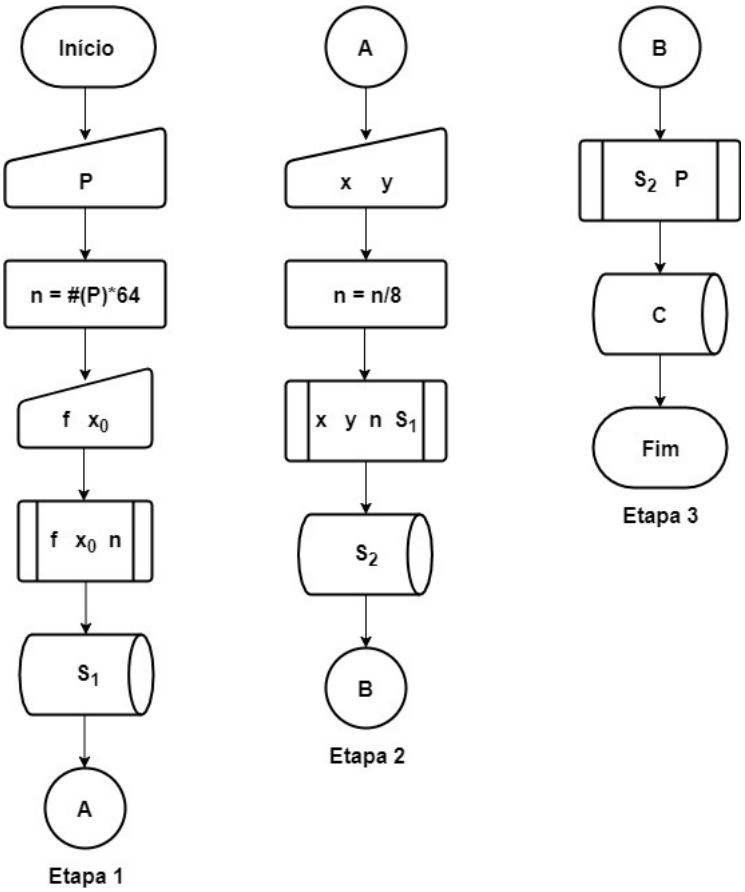


Figura 4.1: Fluxograma do método criptográfico proposto.

Etapa 2: Nesta etapa é construída a sequência binária S_2. Os passos são os seguintes:

- São fornecidas as coordenadas x e y $(x, y \in [0, 1])$.
- O número de iterações n (obtido na etapa anterior) é divido por 8.
- Os parâmetros x, y, n e a sequência binária S_1 são fornecidos ao procedimento que gera a sequência binária S_2.
- A sequência binária S_2 é gravada em disco.

A sequência binária S_2 é gerada da seguinte forma: Primeiro, os pontos x e y se constituem no ponto inicial dentro do quadrado unitário de coordenadas $(0, 0)$, $(1, 0)$, $(1, 1)$ e $(0, 1)$[3]. Em seguida é lido um byte do arquivo S_1 produzido na etapa anterior. A partir deste byte é calculado o módulo do mesmo por 4, o qual produzirá 0, 1, 2 ou

[2]Obviamente esta é uma convenção. Bem poderia ser o contrário.

[3]O protótipo do programa de encriptação/decriptação construído utiliza estas coordenadas. A razão para o porquê disto será explanada na subseção 4.1.1.

3 (resto da divisão por 4). Se o valor produzido for igual a zero, é calculado um novo ponto, correspondente ao ponto médio entre o ponto inicial e o vértice de coordenadas $(0, 0)$; se o valor for igual a 1, então o novo ponto produzido corresponderá ao ponto médio entre o ponto inicial e o vértice de coordenadas $(1, 0)$; o mesmo se dá para os valores 3 e 4, mas neste caso o vértices associados são $(1, 1)$ e $(0, 1)$. O ponto médio ora calculado passa a ser o ponto atual. Um novo byte é lido e o processo continua (um total de n iterações). Em cada passo, verifica-se o valor da coordenada vertical y_n; se $y_n \geq y_m$ (onde y_m é o valor mediano das coordenadas verticais), então é produzido o valor 1; caso contrário, é produzido o valor 0[4] (o método de criação é tecnicamente o mesmo utilizado para a produção da sequência S_1, embora o processo de geração dos pontos seja diferente).

Etapa 3: Nesta etapa é realizada a encriptação do texto plano (ou decriptação do texto cifrado, dado que o algoritmo é simétrico). O funcionamento se dá como segue:

- A sequência binária S_2 e o arquivo de texto plano são fornecidos ao procedimento de encriptação.
- O arquivo de *texto crifrado* é gravado em disco.

O processo de encriptação é simples. São lidos bytes consecutivos dos arquivos *texto plano* e da sequência S_2, os quais são combinados por meio de uma operação XOR (eXclusive OR) e escritos no arquivo de *texto cifrado*.

Não é demais observar que a descrição ora dada não descreve detalhes específicos de implementação em um computador, p. ex., que (e como) os arquivos devem ser abertos, que os parâmetros devem estar em determinadas faixas, que pode acontecer erros de entrada/saída, etc.

Antes de passar para a subseção seguinte, alguns pequenos esclarecimentos. Na etapa 1 o número de iterações n é calculado para ser 64 vezes o número de bytes do arquivo de texto plano. Isso tem uma explicação simples, mas para tanto, considere o seguinte: cada bit do texto plano é cifrado bit a bit com um byte da sequência S_2, porém, cada bit da sequência S_2 demanda um byte para sua geração, o que equivale a dizer que são necessários 8 bytes na sequência S_2 para cada byte no texto plano. Por sua vez, cada bit na sequência S_2 demanda um byte para sua geração, o que significa que cada byte na sequência S_2 requer 8 bytes na sequência S_1 para sua criação; assim, o número total de iterações é um múltiplo de 64 (8×8).

[4] Mais uma vez, trata-se apenas de uma convenção.

Na etapa 2, o número de iterações n é dividido por 8. Neste caso, a explicação é dada nas linhas precedentes (cada byte de texto plano corresponde a 8 bytes da sequência S_2. Ainda na etapa 2, a razão para o qual é tomado o módulo por 4 do byte lido a partir da sequência S_2 é que o resultado serve para associar o ponto atual a um dos vértices do quadrado, desta forma: $0 \rightarrow (0,0)$, $1 \rightarrow (1,9)$, $2 \rightarrow (1,1)$, $3 \rightarrow (0,1)$[5].

4.1.1 Pressupostos Teóricos

Como pode ser visto na descrição do método, o processo de encriptação/decriptação engloba três etapas que possuem pressupostos teóricos que são os fundamentos que hipoteticamente devem (ou deveriam) garantir o êxito (ou em outras palavras, a segurança criptográfica) do procedimento aventado. A seguir é exposto tais hipóteses.

Etapa 1: Nesta etapa é construída a sequência binária S_1, a qual quando concluída servirá como uma das entradas para a etapa 2. A sequência S_1 constitui-se de números (pseudo)-aleatórios que são gerados a partir de uma função que apresenta comportamento caótico quando iterada sob uma condição inicial restrita a um intervalo (base de atração) com parâmetros adequadamente escolhidos. Aqui não serão dados detalhes sobre tais funções, bem como não será discutida a questão do caos determinístico, uma vez que a discussão destes aspectos foi presenteada no capítulo 3. A única observação aqui pertinente é que neste trabalho serão utilizados apenas mapeamentos discretos unidimensionais. Cumpre observar que o valor inicial entregue a função caótica poderia ser pensado como um chave (no sentido usual) em um sistema criptográfico. Ora, esta chave (se assim é permitido dizer), constitui-se de um valor pontual em um intervalo real (para o mapeamento logístico, a faixa de valores admissíveis constitui-se no intervalo real unitário), e consequentemente e idealmente[6], possui infinitos valores, ou ainda, possui a cardinalidade dos números reais $2^{\aleph_0}$[7].

Etapa 2: Nesta fase é construída a sequência binária S_2, que devidamente finalizada servirá como uma das entradas para a etapa 3. Na descrição do método foi mostrado como o mesmo funciona, entretanto, o que o mesmo é? O processo em questão chama-se *jogo do caos*[8] (*chaos game*) e foi utilizado Michael F. Barnsley (Barnsley, 2006, p. 1) para exemplificar a ideia de Sistemas de Funções Iteradas (IFS) e o atrator a ele associado. Per si, o jogo do caos é um algoritmo chamado *Markov Chain Monte Carlo*

[5] Outra vez, a ordem é arbitrária.

[6] Idealmente porque qualquer implementação via computador de um número real está sujeita a inúmeras limitações.

[7] $\aleph_0$ (aleph zero) é a cardinalidade do conjunto de todos os números naturais

[8] Considerando a utilização do jogo do caos nesta etapa do método, aliado ao fato de que em nenhuma parte do presente documento existe uma descrição detalhada do mesmo, no que segue será dada uma descrição mais detalhada do mesmo.

(MCMC) ou *algoritmo de iteração aleatório*, sendo descrito em (Örjan Stenflo, 2002, p.13-32) e (Kaijser, 1981). No contexto de imagens fractais, o algoritmo foi primeiramente descrito em (Barnsley e Demko, 1985, 243–275) baseada nas ideias descritas em (Mandelbrot, 1983). A razão para o seu uso será dada mais adiante (ver o parágrafo no fim da página 83).

O exemplo mais comumente ilustrado do jogo do caos possui os seguintes passos (Barnsley, 2006):

- Marque quatro pontos em uma folha de papel. Rotule três deles como A, B e C, e rotule o ponto remanescente como X_0. Rotule duas faces de um dado de seis lados de A, duas outras faces de B e as duas faces remanescentes de C, ou conceba seu próprio modo de produzir uma sequência aleatória de símbolos A, B e C.

- Role o dado para escolher aleatoriamente um símbolo A, B ou C. No papel, marque o ponto médio entre X_0 e o ponto rotulado pelo símbolo. Chame este ponto médio de X_1. Por exemplo, se o resultado de rolar o dado é B, então X_1 é o ponto médio entre X_0 e B.

- Role o dado novamente. Plote o ponto médio entre X_1 e o ponto cujo rótulo é mostrado no dado. Chame este novo ponto de X_2. Role o dado novamente, e novamente... e plote o novo ponto médio no papel a cada vez. O resultado, na folha de papel, é muito provavelmente algo semelhante a uma figura aproximada de um triângulo de Sierpinsk (Figura 4.2), mas com alguns pontos extras discrepantes.

O algoritmo supra pode ser implementado via computador. Por exemplo, a Figura 4.3 foi produzida utilizando o software Maxima por meio do package dynamics[9].

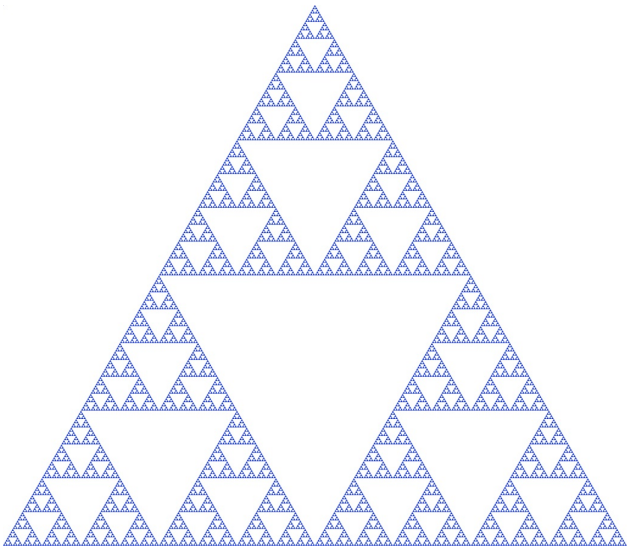


Figura 4.2: Triangulo de Sierpinsk.

[9] A título de escólio, neste gráfico são apresentadas 30000 iterações do jogo do caos.

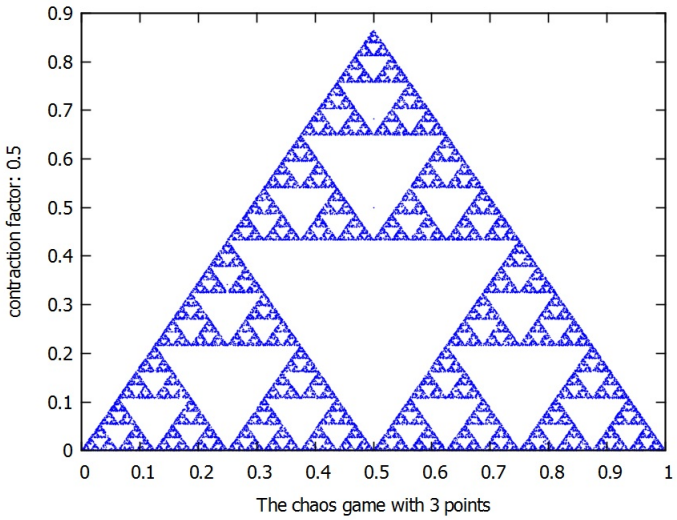


Figura 4.3: Triangulo de Sierpinsk produzido pelo software Maxima.

Observe que a legenda vertical exibe os seguintes dizeres: *contraction factor: 0.5*. Bem, este fator de contração nada mais é do que a instrução para tomar *o ponto médio* descrita pelo algoritmo.

O que acontece ao se tomar outros fatores de contração? Para o jogo do caos com três pontos e coordenadas iguais às que foram utilizadas, a resposta é dada pela Figura 4.4, onde é apresentado os *atratores* resultantes do processo iterativo.

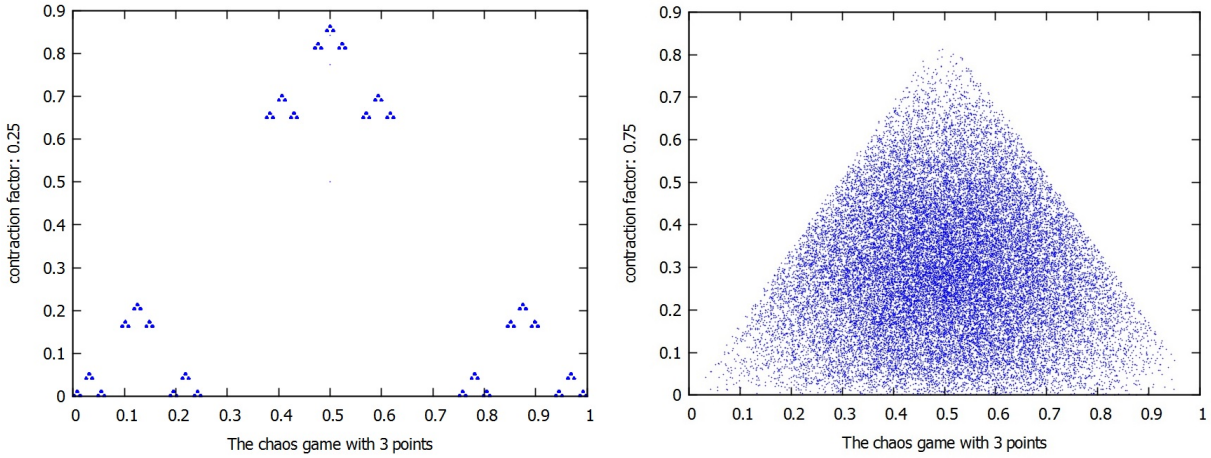


Figura 4.4: Jogo do caos com três pontos e fatores de escala iguais a 1/4 e 3/4.

Entretanto, o jogo do caos não está limitado a três pontos apenas[10], outros pontos (vértices) podem ser adicionados. As Figuras 4.5, 4.6, 4.7, 4.8, 4.9 e 4.10 exibem o jogo do caos com maior números de pontos.

[10] A propósito, o "jogo" em questão pode ser estendido para três ou mais dimensões.

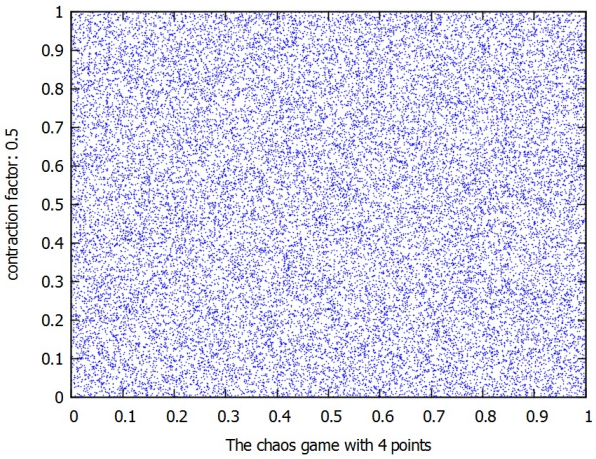


Figura 4.5: Jogo do caos com quatro pontos e fator de escala 1/2.

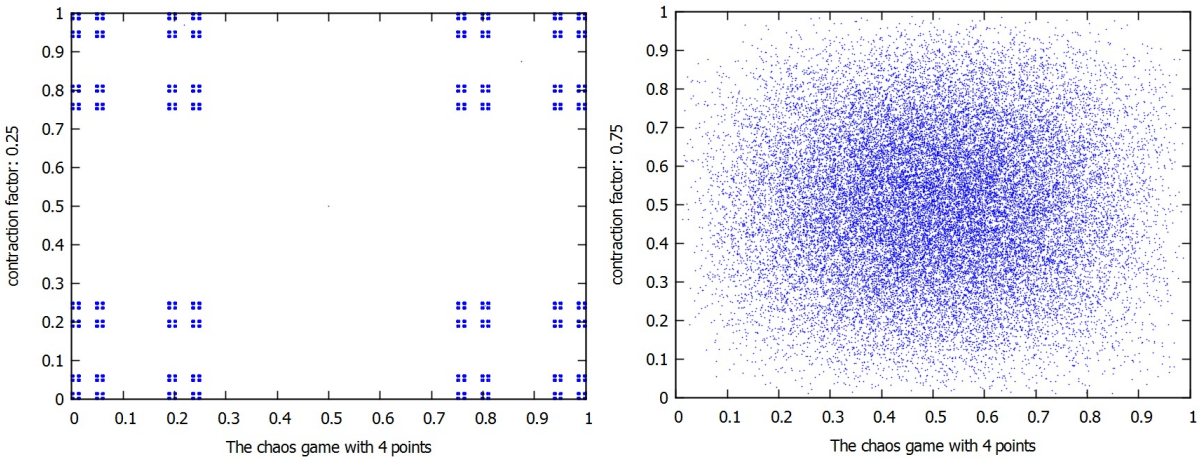


Figura 4.6: Jogo do caos com quatro pontos e fatores de escala iguais a 1/4 e 3/4.

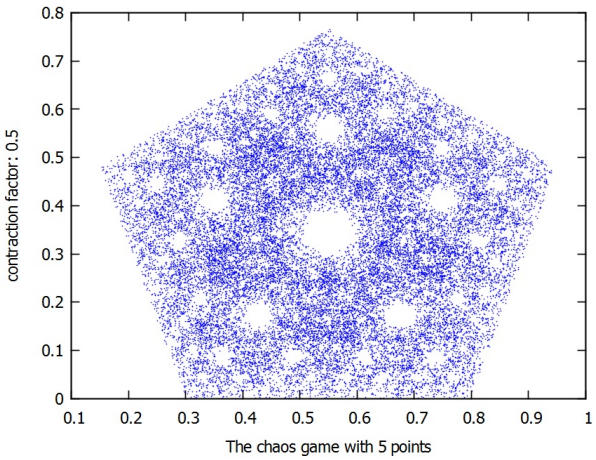


Figura 4.7: Jogo do caos com cinco pontos e fator de escala igual a 1/2.

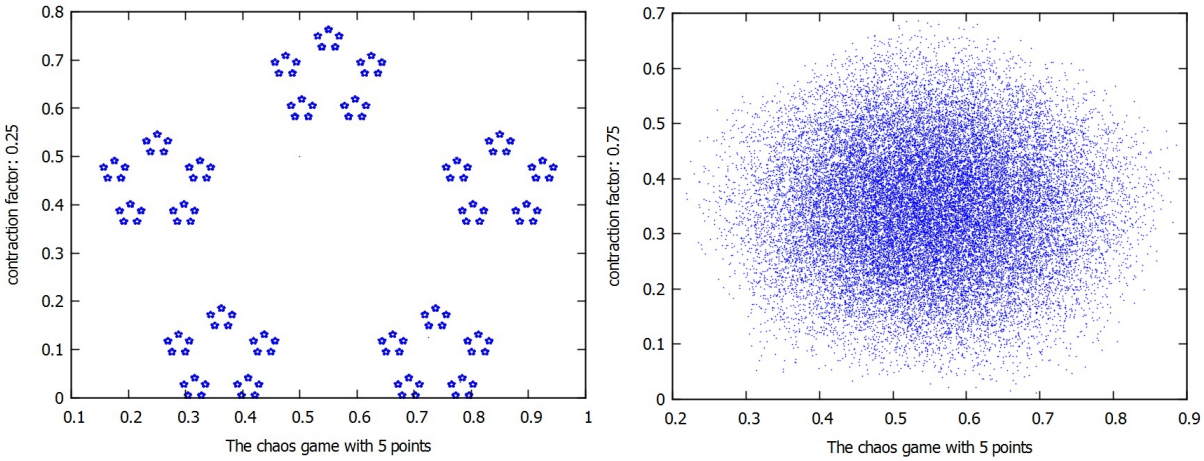


Figura 4.8: Jogo do caos com cinco pontos e fatores de escala iguais a 1/4 e 3/4.

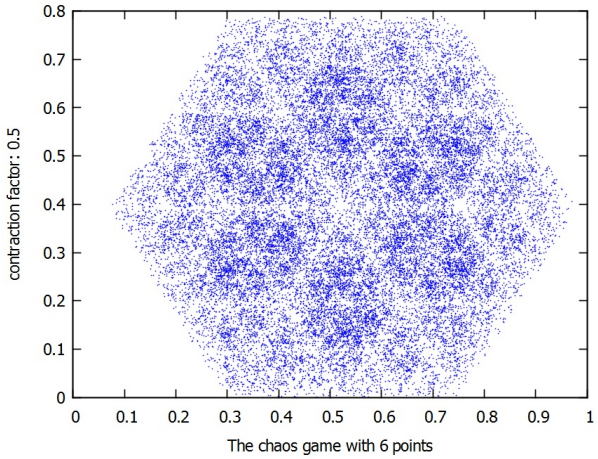


Figura 4.9: Jogo do caos com seis pontos e fatores de escala iguais a 1/2.

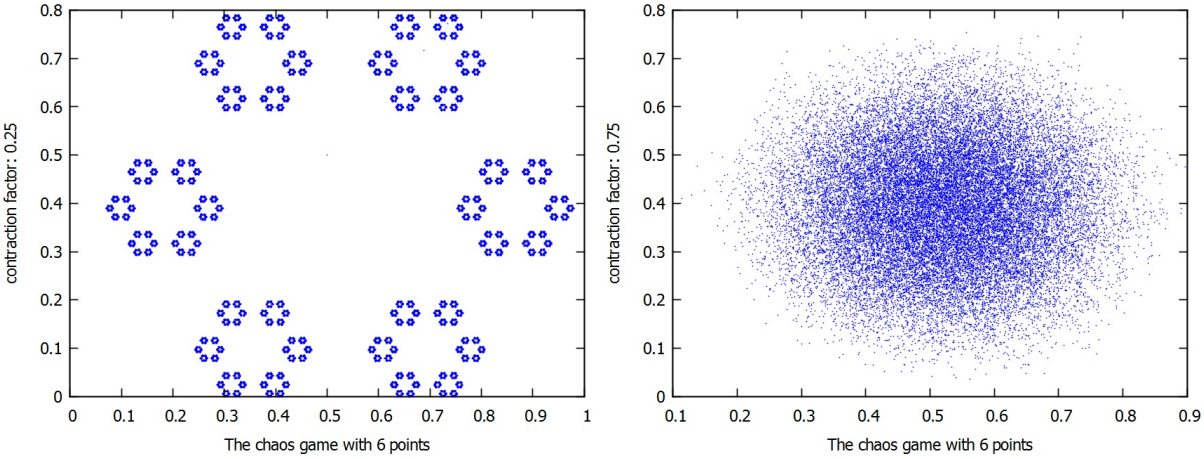


Figura 4.10: Jogo do caos com seis pontos e fatores de escala iguais a 1/4 e 3/4.

Agora, é chegada a hora de dizer o porque do uso do jogo do caos no método aventado. Observe os gráficos produzidos por sucessivas "rodadas" do jogos do caos, em

particular, a dispersão dos pontos no quadrado unitário no plano. Dentre estes, o que apresenta a maior dispersão e "preenche" o *espaço de fase* é o representado pela Figura 4.5. O que isto significa? Bem, isto pode significar que no procedimento em questão os pontos estão distribuídos aleatoriamente e de maneira uniforme[11].

Ora, a simples aparição desta característica (visualmente exibida na Figura 4.5) não é *garantia* que a sequência de pontos gerada pelo jogo do caos, e consequentemente a sequência S_2 gerada a partir deste pontos, é verdadeiramente aleatória. Contudo, como explicitado na subseção 2.3, nunca se é possível asseverar que uma sequência é verdadeiramente aleatória, de modo que será tomado como *hipótese* que as sequências S_1 e S_2 são aleatórias.

Ademais, não é surpresa (ou pelo menos não deveria) notar que esquema exposto apresenta *dependência sensível sobre as condições iniciais*, considerando que pontos iniciais diferentes (o ponto rotulado como X_0 no algoritmo), embora conduzam para o mesmo atrator, o fazem de modo diferente, ou mais especificamente, as órbitas convergem para o atrator, mas o fazem por caminhos (se for permitido utilizar esta palavra) diferentes.

Ainda, com relação a condição inicial X_0, note que teoricamente ele (não esquecendo das limitações da representação de números reais em computador), pode assumir qualquer valor no conjunto $[0,1] \times [0,1]$, o que equivale a dizer *infinitos valores*[12].

Etapa 3: Esta fase, na qual é verdadeiramente realizada as operações de encriptação/decriptação, é a mais simples de todas, utilizando apenas uma operação $\oplus$ (XOR - eXclusive OR), que pode a primeira vista parecer um tanto quanto simples. Entretanto, uma explicação elementar pode demonstrar que a mesma é adequada para o método proposto. Para tanto, considere a Figura 4.11.

Nela uma imagem é apresentada em sua forma original no canto esquerdo superior; já as imagens seguintes representam a mesma imagem combinada, pixel a pixel, com uma mesma sequência binária aleatória por meio das operações lógicas elementares AND (a), OR (b) e XOR (c). Pode ser observado que a imagem obtida por meio da operação AND (no canto superior direito) apresenta-se mais escura. Por sua vez a imagem obtida por meio da operação OR (no canto inferior esquerdo) apresenta-se mais clara. Finalmente, a imagem obtida por meio da operação XOR apresenta-se, pode-se

[11] E em razão disto é utilizado no procedimento proposto.

[12] Embora talvez possa parecer um contra-senso, $\#[0,1]^2 = \#[0,1] = 2^{\aleph_0}$, ou seja, os conjuntos possuem a mesma cardinalidade (mesmo número de elementos).

dizer, totalmente indiscernível.

A razão para estes resultados fundamenta-se tanto na forma de representação de cor de cada pixel quanto nas definições dos operadores lógicos. Em uma imagem a cor de um pixel é codificada como uma sequência binária de 24-bits ou 3 bytes, sendo que cada byte representa, nesta ordem, um tom de vermelho, verde e azul. Por sua vez, cada uma desta cores pode ser representada no intervalo numérico compreendido entre 0 e 255, sendo a cor preta representada como (0, 0, 0) e a cor branca como (255, 255, 255), o que na representação binária corresponde a 000000000000000000000000 e 111111111111111111111111. Já os operadores lógicos tem seus comportamentos discriminados nas Tabela 4.1.

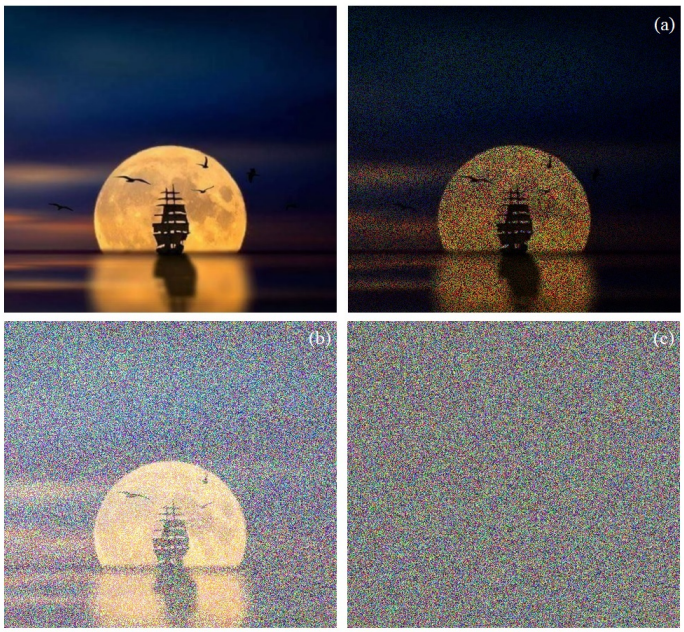


Figura 4.11: Encriptação usando as operações lógicas de conjunção (a), disjunção (b) e ou-exclusivo (c).

Tabela 4.1: Tabela verdade dos operadores AND, OR e XOR.

S_1	S_2	$S_1 \wedge S_2$	$S_1 \vee S_2$	$S_1 \oplus S_2$
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Das tabelas constata-se que uma operação de conjunção binária resulta em 1 quando ambos os seus operandos são iguais a 1; uma operação de disjunção binária

resulta em 0 quando os seus operandos são iguais a 0 e, finalmente, uma operação de disjunção exclusiva binária resulta em 1 apenas quando os seus operandos possuem valores diferentes.

Considere agora a sequência binária S_1 = 100111001011010100111010 que representa um cor qualquer[13] e a sequência S_2 = 010110100001101111011000, tomada aleatoriamente. Quais seriam os resultados da aplicação das operações lógicas englobando as duas sequências (Tabela 4.2)?

Tabela 4.2: Aplicação das operações $\wedge$, $\vee$ e $\oplus$ a duas sequências binárias.

S_1	100111001011010100111010
S_2	010110100001101111011000
$S_1 \wedge S_2$	000110000001000100011000
$S_1 \vee S_2$	110111101011111111111010
$S_1 \oplus S_2$	110001101010111011100010

Da Tabela 4.2 constata-se que a operação de conjunção produz maior número de zeros, de modo que a cor do pixel tende a preto, o que explica o porquê da Figura 4.11-(a) no canto superior esquerdo ficar escurecida. Por sua vez a operação de disjunção produz maior número de uns, de forma que a cor do pixel tende ao branco, o que explica porque a Figura 4.11-(b) no canto inferior esquerdo apresenta-se em tom mais claro. Já na operação de disjunção exclusiva, pode ser verificado que a mesma pode produzir qualquer sequência possível (considerando que a sequência S_2 é aleatória), e isso implica que o pixel em questão poderia ter qualquer combinação de cor possível entre 16 milhões de possibilidades!

Posto de outra maneira, em uma conjunção o resultado 1 tem probabilidade de ocorrência igual a 1/4 e o resultado 0 tem probabilidade de ocorrência igual a 3/4 (o resultado nunca pode ter valor numérico maior do que a sequência de menor valor); em uma operação de disjunção o resultado 1 acontece na razão 3/4, enquanto que o resultado 0 acontece na razão 1/4 (o resultado nunca pode ter valor numérico menor do que a sequência de maior valor); já em uma disjunção exclusiva, os resultados 1 e 0 são igualmente prováveis (ou equiprováveis) e numericamente iguais a 1/2.

Os apontamentos anteriores podem ser resumidos dizendo que se uma operação de disjunção exclusiva é aplicada combinando uma sequência binária com uma outra sequência binária aleatória, o resultado pode ser qualquer sequência binária possível,

[13] Na verdade um tom de verde.

o que do ponto de vista criptográfico significa que nenhuma informação é passível de extração.

4.1.2 Experimentos de Ordem Estatística

Considerando que sequências geradas (S_1 e S_2) nas duas primeiras etapas não são verdadeiramente aleatórias (no sentido de que são reproduzíveis), seria possível asseverar se elas se assemelham (em sentido estatístico) a sequências verdadeiramente aleatórias?

A resposta é afirmativa em certo grau. Deve-se dizer em medida não exata porque nem mesmo as sequências verdadeiramente aleatórias podem ser provadas serem aleatórias, quer analiticamente, quer por meios estatísticos. O que se faz, quer com a sequências aleatórias verdeiras, quer com as pseudo-aleatórias, é assumir certas hipóteses sobre aleatoriedade, aplicando-se então a elas testes estatísticos para aferir se essas assunções se sustentam[14].

Existe um teste estatístico (ou conjunto de testes estatísticos) que pode asseverar a certeza que uma determinada sequência é aleatória? A resposta a esta inquirição é negativa[15]. Nenhum conjunto de testes estatísticos pode com absoluta certeza certificar que uma sequência é verdadeiramente aleatória e, portanto, testes estatísticos não podem servir como substitutos para *criptoanálise*.

A bem da verdade, a cifra aventada neste trabalho não será objeto de análise criptográfica. Então: (1) Por que não será realizada análise criptográfica; (2) Como será asseverada se o a cifra é segura?

Para a primeira pergunta, a resposta é que o processo de criptoanálise é algo que deve ser realizado posteriormente a etapa de construção da cifra, havendo ainda de se argumentar que embora a criptoanálise seja complementar ao trabalho de concepção, a mesma se constitui em outro trabalho, o que inequivocamente está fora de escopo para esta dissertação.

Para a segunda pergunta, a réplica é simples. Considerando o que já foi exposto nos poucos parágrafos precedentes, não será dada nenhuma prova cabal (e formal) da

[14] If a sequence behaves randomly with respect to tests $T_1, T_2, \ldots, T_n$, we cannot be sure in general that it will not be a miserable failure when it is subjected to a further test T_{n+1}. Yet each test gives us more and more confidence in the randomness of the sequence. In practice, we apply about half a dozen different kinds of statistical tests to a sequence, and if it passes them satisfactorily we consider it to be random – it is then presumed innocent until proven guilty. (Knuth, 1998, p. 41)

[15] There is literally no end to the number of tests that can be conceived;... (Knuth, 1998, p. 41)

segurança da cifra (isso deve ser feito por meio de criptoanálise)[16], contudo, pretende-se demonstrar que as propriedades estatísticas dos criptogramas gerados pelo método "sugerem" confiabilidade para uso criptográfico. Dito isto, deve estar patente que serão empregados métodos estatísticos. Pergunta: quais métodos estatísticos? Bem, serão empregadas diferentes suítes de testes estatísticos para avaliar a qualidade, mais especificamente, a aleatoriedade dos criptogramas gerados.

Por sua vez, resta agora saber quais suítes de testes estatísticos serão escolhidas. Para o presente trabalho serão utilizadas as seguintes: ENT, Dieharder e NIST STS. Aqui não será dada uma descrição mais profunda das suítes de testes empregadas, sendo isto deixado para o Apêndice A. Em lugar disso, será explanado o porquê da escolha das tais.

A bateria de testes ENT será utilizada por apresentar rápido processamento de grandes sequências binárias de dados ($\geq$ 100MB). Apesar de ser constituída de apenas 5 testes que podem ser considerados básicos, estes testes conseguem capturar aspectos que toda sequência aleatória deveria possuir, indicando se a sequência deve ser rejeitada sob o aspecto da aleatoriedade ou deve ser submetida a outros testes.

A bateria de testes Dieharder[17], desenvolvida por Robert G. Brown, não tem como objetivo primário o teste de arquivos compostos por possíveis números aleatórios, mas trata-se de uma suíte de testes de geradores de números aleatórios. Contudo, o teste de arquivos contendo possíveis números aleatórios é possível, e esta opção será aqui utilizada. Dieharder é nomeado em homenagem a bateria de testes de geradores de números aleatórios *diehard*, desenvolvida por George Marsaglia, contendo todos os testes desta última. Contudo, mais do que uma atualização, dieharder tem código-fonte aberto, é escrito em C, e vem incorporando diversos novos testes ao longo de seu desenvolvimento. A principal razão para o uso de *dieharder* (em detrimento de *diehard*) é que o primeiro provê um relatório contendo informações sobre o sucesso ou fracasso em testes específicos, o que não demanda do utilizador uma análise estatística dos resultados.

NIST STS (Stastistical Test Suite)[18] (Rukhin et al., 2010) é um pacote de software cujo escopo é fornecer um conjunto de testes de estatísticos visando avaliar a qualidade de geradores de números aleatórios e pseudo-aleatórios para aplicações criptográficas. Desenvolvido e disponibilizado pelo NIST (National Institute of Standards

[16]Há de se considerar também que métodos de criptoanálise para criptografia baseada em caos ainda não estão suficientemente amadurecidos, ao contrário do que acontece com a *criptografia convencional*.
[17]A versão utilizada é a 3.31.1.
[18]A versão utilizada é 2.1.2.

and Technology), é considerado como o *padrão* aceito pela indústria[19], sendo empregado tanto por empresas que produzem hardware gerador de números aleatórios, p. ex., ID Quantique SA, qutools GmbH e Araneus Information Systems Oy, quanto por fornecedores de números aleatórios via Internet.

Per si, o NIST STS oferece um total de 15 testes estatísticos. Estes testes avaliam a presença de um padrão que, se detectado, poderia indicar que a sequência não é aleatória. As propriedades de uma sequência aleatória podem ser descritas em termos de probabilidades, e em cada teste é extraído um p-value (probabilidade) que indica a força da evidência contra a hipótese de aleatoriedade perfeita. Um p-value igual a 0 indica que a sequência aparenta ser completamente não aleatória; já um p-value maior do 0.01 significa que a sequência é considerada ser aleatória com confiança de 99%.

Por fim, deve ter sido notado que até aqui nada foi informado sobre os detalhes dos experimentos estatísticos a serem realizados, muito embora tenha sido explicado o porquê da utilização dos mesmos e quais seriam as ferramentas utilizadas. Tais detalhes são deixados para o próximo capítulo, onde será explicado o *protocolo experimental*.

[19] E esta é a principal razão para a sua utilização neste trabalho.

Capítulo 5

Experimentos e Resultados

O presente capítulo pretende ser a finalização dos capítulos precedentes, em especial do último. Aqui, serão apresentados o protocolo experimental e os resultados dos experimentos estatísticos. Entretanto, antes de iniciar estes tópicos, será dado um vislumbre do protótipo do programa de encriptação/decriptação. Isso se faz necessário? Sim, e a razão para tanto é que isto permitirá ver o método proposto em ação e como o protocolo experimental utilizará os arquivos produzidos no processo de encriptação/decriptação com vistas a validação da técnica.

5.1 Programa Protótipo

O programa protótipo foi nomeado como `ChaosCrypt`, tendo sido desenvolvido na linguagem de programação C para a plataforma MS-Windows com o uso do compilador `gcc`. Em relação a interface, não foi desenvolvida uma GUI, mas em lugar disso é ofertada uma interface de linha de comandos (CLI).

Em razão da interface, o programa é chamado digitando seu nome em uma janela terminal. Quando executado, o programa apresenta uma série de mensagens, algumas das quais devem ser respondidas. Um exemplo de execução é como segue:

```
ChaosCrypt(R) by Ivan Ivanovich Goratchim

Please, type the name of the file to be encrypted/decrypted: Moon-Ship.bmp
Please, type the name of the file to be outputed: Moon-Ship-e.bmp

Choose the chaotic map:
(1) - quadratic map: x -> x^2 + r
(2) - quadratic map: x -> -x^2 + r
(3) - logistic map:  x -> r*x*(x - 1)
(4) - sine map:      x -> r*sin(x)
(5) - cosine map:    x -> r*cos(x)
```

```
Your option: 1
You choose the quadratic map x -> x^2 + r
Please, choose the initial value (-2, 2): 1.2
Please, enter the x-coordinate (value must be in [0, 1]): 0.4
Please, enter the y-coordinate (value must be in [0, 1]): 0.83
Done!
```

Subsequente à mensagem de copyright[1], é solicitado ao usuário o nome do arquivo a ser encriptado/decriptado e em seguida, o nome do arquivo de saída (texto cifrado/plano nos casos de encriptação/decriptação).

Imediatamente após, é apresentada ao usuário as opções de escolha para os mapas caóticos. Feita esta escolha, é apresentado um aviso informando o mapa escolhido, sendo então solicitado a escolha do valor inicial, a qual deve estar na faixa de valores apresentada. Neste ponto, o programa cria a sequência S_1, mencionada na discussão do método constante no capítulo 4.

Em seguida, são solicitadas as coordenadas (abscissa (x) e ordenada (y)) que serão utilizadas pelo "jogo do caos" a fim de produzir a sequência S_2.

A partir daqui não se é mais solicitado dados ao usuário, e o programa executa a etapa de encriptação/decriptação já discutida. Quando finalizado, o programa imprime a mensagem `Done!` para indicar o encerramento dos procedimentos e que o arquivo de saída foi produzido.

Mensagens de erro são emitidas (caso aconteça alguma falha) somente após a inserção das coordenadas descritas no parágrafo precedente.

5.2 Protocolo Experimental

Para fins de testes, foram escolhidos 4 arquivos de tipos diferentes:

- 01 arquivo de texto: Solaris.txt (399962 bytes) - romance de ficção científica escrito em 1961 pelo autor polonês Stanisław Lem (no qual não foi aplicado nenhuma tratamento específico, significando que todos os caracteres originais do texto foram deixados "intocados").
- 01 arquivo de imagem: Universum.bmp (1387254 bytes) - xilogravura, de autoria desconhecida que teve sua primeira aparição no livro do astrônomo francês Camille

[1] Did you know who is Ivan Ivanovich Goratchim? Na ja, er ist mein doppelgänger.

Flammarion, intitulado "L'atmosphère: météorologie populaire" e publicado em 1888.

- 01 arquivo de áudio: Stalker_Theme.mp3 (10650354 bytes) - música tema do filme Stalker (1979), de Andrei Tarkovsky.
- 01 arquivo de vídeo: Stalker.mp4 (1232553944 bytes) - Filme de Andrei Tarkovsky (1979).

Cada um destes arquivos foi objeto de 30 operações de encriptação utilizando 30 chaves diferentes (condição inicial) para os mapeamentos caóticos[2] e uma chave (coordenada) única para o *jogo do caos*[3]. O arquivo de *texto plano* e os arquivos resultantes do processo de encriptação (sequência binária S_1, sequência binária S_2 e *texto cifrado*) formam então submetidos aos testes disponibilizados pelas suítes ENT, Dieharder e NIST STS.

Agora, considerando que cada um dos mapeamentos disponibilizados pelo programa possuem faixas diferentes de valores admissíveis como condição inicial, como estes valores foram escolhidos?

Bem, para os mapeamentos (1) e (2), mapas quadráticos na forma $f(x) = \pm\, x^2 + c$, a faixa de valores admissíveis está contida no intervalo $(-2, 2)$, conforme pode-se observar a partir dos diagramas de bifurcação constantes na Figura 5.1.

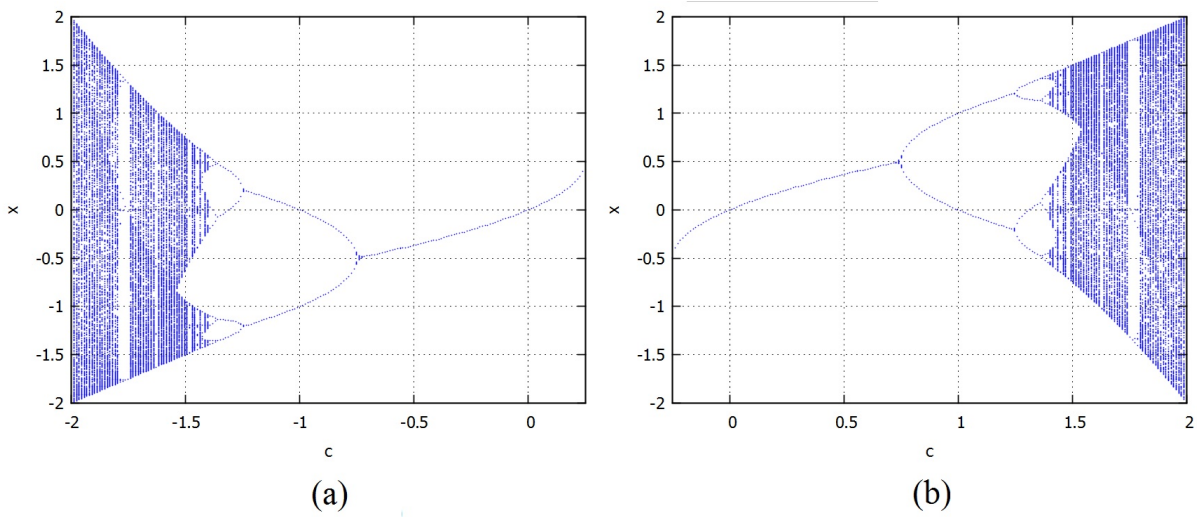


Figura 5.1: Diagramas de bifurcação: (a) $f(x) = x^2 + c$; (b) $f(x) = -x^2 + c$.

[2]Somente os três primeiros mapeamentos foram objetos de teste. Os mapeamentos restantes necessitam de refinamentos antes de serem liberados para teste.

[3]A razão para isso é são realizados um total de 51 testes por arquivo com esta configuração. Se mais de uma *chave* fosse utilizada para o jogo do caos, esse número seria multiplicado pelo total de chaves usadas, o que embora factível, está aquém dos recursos computacionais e temporais disponíveis.

Com vistas a evitar qualquer possível viés na escolha de valores, foi-se adotado a seguinte opção: os valores são provenientes de uma fonte aleatória. Mais especificamente, foram tomados 10000 números aleatórios contidos no intervalo $[0,1]$ a partir de Random Decimal Fraction Generator, um dos serviços gratuitos oferecidos por Random.org. Destes, foram escolhidos os 30 primeiros números. Como estes números se situam no intervalo $[0,1]$, eles foram multiplicados por 2 para que os mesmos se situassem na faixa do intervalo $[0,2]$. A fim de que estes números ficassem no intervalo desejado de $[-2,2]$, foram tomados os 30 primeiros bits dos bytes (de um total de 16384) gerados a partir de Random Byte Generator, outro serviço gratuito de Random.org. Estes bits foram pareados com os números fracionários anteriormente obtidos e tratados como bits de sinal da seguinte forma: se o bit é 1, o número fracionário em questão é positivo; se o bit é 0, o número fracionário é negativo. Os números resultantes pertencem agora ao intervalo $[-2,2]$, mas antes disso sua magnitude foi reduzida de 20 casas decimais para duas, rendendo: -1,95; -1,86; -1,83; -1,66; -0,76; -0,59; -0,52; -0,46; -0,44; -0,36; -0,18; -0,14; -0,05; 0,01; 0,04; 0,07; 0,23; 0,33; 0,50; 0,61; 0,79; 0,88; 1,14; 1,37; 1,39; 1,58; 1,71; 1,80; 1,82 e 1,92.

Para o terceiro mapeamento, o mapa logístico $f(x) = rx(1-x)$, a faixa de valores admissíveis se situa no intervalo real unitário $[0,1]$ como pode ser visto por meio do diagrama de bifurcação (eixo vertical) da Figura 5.2.

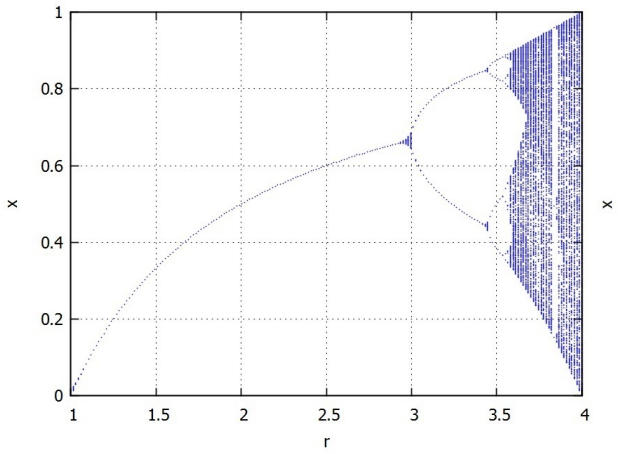


Figura 5.2: Diagrama de bifurcação para $f(x) = rx(1-x)$.

Para este mapeamento, os valores tomados como condições iniciais são os originalmente obtidos de Random Decimal Fraction Generator. Como estes números já estão normalizados no intervalo $[0,1]$, não se foi necessário nenhuma manipulação, com exceção de que os mesmos foram arredondados para duas casas decimais. Eles são: 0,01; 0,05; 0,06; 0,07; 0,14; 0,18; 0,23; 0,33; 0,36; 0,44; 0,46; 0,51; 0,52; 0,59; 0,61; 0,76; 0,79; 0,88; 0,14; 0,37; 0,39; 0,58; 0,66; 0,71; 0,80; 0,82; 0,83; 0,86; 0,92 e 0,95.

Como valores iniciais (coordenadas) a serem passados para o *jogo do caos*, foram escolhidos os dois valores consecutivos aos 30 primeiros tomados para o mapeamento anterior, os quais também foram arredondados para duas casas decimais: 0,72 e 0,16.

As suítes de testes Dieharder e NIST STS produzem um relatório analítico relatando os resultados dos testes (sucesso ou fracasso). Já a suíte ENT, apesar de relatar os valores dos resultados, não emite nenhum relatório concernente ao sucesso ou fracasso dos testes, e desta feita, é deixado para o experimentador a tarefa de analisar os valores reportados.

Considerando isto, para os cinco testes da suíte ENT foram considerados os seguintes parâmetros:

- **Entropia:** Sucesso se o valor observado for maior ou igual a 0,95; caso contrário, fracasso.
- **Teste χ^2:** Sucesso se o valor observado estiver no intervalo $[1, 99]$; caso contrário, fracasso.
- **Média Aritmética:** Sucesso se o módulo do valor observado for menor ou igual a 0,0005; caso contrário, fracasso.
- **Cálculo de Pi pelo Método de Monte Carlo:** Sucesso se o valor observado for percentualmente menor ou igual do que 0,5; caso contrário, fracasso.
- **Coeficiente de Correlação Linear:** Sucesso se o módulo do valor observado for menor ou igual a 0,001; caso contrário, fracasso.

Para as demais suítes de testes foram adotados os valores default.

5.3 Resultados

5.3.1 ENT

5.3.1.1 Mapeamento Quadrático $f(x) = x^2 + r$

Todos os arquivos de texto plano submetidos aos testes (texto, imagem, áudio e vídeo) da suíte ENT falharam em todos os testes, com exceção do teste de entropia (ver aba "*Plain Text*" da planilha Quadratic 1 - Analysis (Ent) - Final).

Os arquivos contendo as sequências S_1 geradas passaram em quase a totalidade dos testes, com exceções pontuais (ver aba "*Orbits*" da planilha Quadratic 1 - Analysis

(Ent) - Final).

Os arquivos contendo as sequências S_2 geradas passaram em quase a totalidade dos testes, com exceções pontuais (ver aba "*ChaosGame*" da planilha Quadratic 1 - Analysis (Ent) - Final).

Os arquivos contendo os textos cifrados passaram em quase a totalidade dos testes, com exceções pontuais (ver aba "*Encrypted*" da planilha Quadratic 1 - Analysis (Ent) - Final).

A planilha contendo os resultados pode ser acessada neste link: Quadratic 1 - Analysis (Ent) - Final.

5.3.1.2 Mapeamento Quadrático $f(x) = -x^2 + r$

Todos os arquivos de texto plano submetidos aos testes (texto, imagem, áudio e vídeo) da suíte ENT falharam em todos os testes, com exceção do teste de entropia (ver aba "*Plain Text*" da planilha Quadratic 2 - Analysis (Ent) - Final).

Os arquivos contendo as sequências S_1 geradas passaram em quase a totalidade dos testes, com exceções pontuais (ver aba "*Orbits*" da planilha Quadratic 2 - Analysis (Ent) - Final).

Os arquivos contendo as sequências S_2 geradas passaram em quase a totalidade dos testes, com exceções pontuais (ver aba "*Chaos Game*" da planilha Quadratic 2 - Analysis (Ent) - Final).

Os arquivos contendo os textos cifrados passaram em quase a totalidade dos testes, com exceções pontuais (ver aba "*Encrypted*" da planilha Quadratic 2 - Analysis (Ent) - Final).

A planilha contendo os resultados pode ser acessada neste link: Quadratic 2 - Analysis (Ent) - Final.

5.3.1.3 Mapeamento Logístico $f(x) = rx(1 - x)$

Todos os arquivos de texto plano submetidos aos testes (texto, imagem, áudio e vídeo) da suíte ENT falharam em todos os testes, com exceção do teste de entropia (ver aba "*Plain Text*" da planilha Logistic - Analysis (Ent) - Final).

Os arquivos contendo as sequências S_1 geradas passaram em quase a totalidade dos testes, com exceções pontuais (ver aba "*Orbits*" da planilha Logistic - Analysis (Ent) - Final).

Os arquivos contendo as sequências S_2 geradas passaram em quase a totalidade dos testes, com exceções pontuais (ver aba "*Chaos Game*" da planilha Logistic - Analysis (Ent) - Final).

Os arquivos contendo os texto cifrados passaram em quase a totalidade dos testes, com exceções pontuais (ver aba "*Encrypted*" da planilha Logistic - Analysis (Ent) - Final).

A planilha contendo os resultados pode ser acessada neste link: Logistic - Analysis (Ent) - Final.

5.3.1.4 Análise

Dos resultados apresentados pelas tabelas, constata-se:

- Os textos planos falham em todos os testes da suíte ENT, com exceção da entropia. Como o valor para sucesso no teste de entropia foi fixado em 0,95, poder-se-ia esperar que mais arquivos e sequências falhassem no teste, mas numericamente todos os arquivos e sequências submetidas ao teste apresentaram um valor próximo de 1, o que indica que a nível binário estes arquivos apresentam alta entropia[4].
- Com exceção de casos pontuais, todas as sequências pseudo-aleatórias S_1 e S_2 passaram nos testes, mas mais importante, o *texto cifrado* resultante também passou nos testes, o que contrasta de forma bastante contundente com o *texto cifrado*, sendo que idealmente o texto cifrado deve-se aproximar de uma distribuição uniforme, ou em outras palavras, deve apresentar-se o mais aleatório possível.

5.3.2 Dieharder

5.3.2.1 Mapeamento Quadrático $f(x) = x^2 + r$

De forma semelhante a que aconteceu com os arquivos de texto plano apresentados a suíte de testes ENT, os mesmos arquivos quando submetidos ao escrutínio dos testes da suíte Dieharder falharam, salvo raras exceções (ver aba "*Plain Text*" da planilha

[4]Os resultados indicam que uma sequência binária aleatória deve possuir alta entropia, a saber, 1; contudo, a recíproca não é verdadeira - um arquivo em formato binário com alta entropia não necessariamente se constitui em uma sequência binária aleatória.

Quadratic 1 - Analysis (Diehard) - Final).

Já os arquivos contendo as sequências S_1, S_2 e o texto cifrado apresentaram o seguinte resultado: conforme o tamanho dos arquivos foi aumentando, a quantidade de testes que foram bem sucedidos aumentou equivalentemente, passando em todos os testes (com raras exceções) para as sequências S_1, S_2 e texto cifrado para o arquivo de vídeo (ver as abas 01 a 30 da planilha Quadratic 1 - Analysis (Diehard) - Final).

A planilha contendo os resultados pode ser acessada neste link: Quadratic 1 - Analysis (Diehard) - Final.

5.3.2.2 Mapeamento Quadrático $f(x) = -x^2 + r$

Similarmente aos arquivos de texto plano submetidos a suíte de testes ENT, os mesmos arquivos quando submetidos ao escrutínio dos testes da suíte Dieharder falharam, salvo raras exceções (ver aba "*Plain Text*" da planilha Quadratic 2 - Analysis (Diehard) - Final).

Já os arquivos contendo as sequências S_1, S_2 e o texto cifrado apresentaram o seguinte resultado: conforme o tamanho dos arquivos foi aumentando, a quantidade de testes que foram bem sucedidos aumentou equivalentemente, passando em todos os testes (com raras exceções) para as sequências S_1, S_2 e texto cifrado para o arquivo de vídeo (ver as abas 01 a 30 da planilha Quadratic 2 - Analysis (Diehard) - Final).

A planilha contendo os resultados pode ser acessada neste link: Quadratic 2 - Analysis (Diehard) - Final.

5.3.2.3 Mapeamento Logístico $f(x) = rx(1 - x)$

Identicamente aos arquivos de texto plano submetidos a suíte de testes ENT, os mesmos arquivos quando submetidos ao escrutínio dos testes da suíte Dieharder falharam, salvo raras exceções (ver aba "*Plain Text*" da planilha Logistic - Analysis (Diehard) - Final).

Por sua vez, os arquivos contendo as sequências S_1, S_2 e o texto cifrado apresentaram o seguinte resultado: conforme o tamanho dos arquivos foi aumentando, a quantidade de testes que foram bem sucedidos aumentou equivalentemente, passando em todos os testes (com raras exceções) para as sequências S_1, S_2 e texto cifrado para o arquivo de vídeo (ver as abas 01 a 30 da planilha Logistic - Analysis (Diehard) - Final).

A planilha contendo os resultados pode ser acessada neste link: Logistic - Analysis (Diehard) - Final.

5.3.2.4 Análise

A suíte de testes Dieharder possui um conjunto mais amplos de testes do que ENT, com testes mais estritos e específicos, que podem muito bem captar desvios das sequências submetidas em relação à aleatoriedade.

Ademais, dos resultados obtidos, digno é de se notar que um aumento no tamanho das sequências corresponde a um incremento proporcional da quantidade de testes bem sucedidos, alcançando bons resultados para as sequências S_1, S_2 e *texto cifrado* para o arquivo de vídeo correspondente. Idealmente, uma sequência aleatória deve possuir tamanho ilimitado, mas isto não é pragmaticamente possível. Entretanto, há de se esperar que sequências de tamanho crescente apresentem características estatísticas que permitam classificá-la como aleatória. Desta feita, os testes aparentam corroborar este aspecto.

Por derradeiro, da mesma forma como apresentada pelos resultados dos testes da suíte ENT, comprova-se que o *texto cifrado* difere em muito do *texto plano*, algo que além de esperado, é o almejado.

5.3.3 STS

5.3.3.1 Mapeamento Quadrático $f(x) = x^2 + r$

Os textos planos correspondentes aos arquivos de texto, imagem e áudio não puderam ser avaliados pela suíte de teste STS. Quando submetidos, o programa emitiu a seguinte informação: *READ ERROR: Insufficient data in file*. Desta feita, não foi possível avaliar a aleatoriedade dos mesmos. Já o arquivo de vídeo foi avaliado, mas falhou em todas as instâncias dos testes (ver aba "Plain Text"da planilha Quadratic 1 - Analysis (STS) - Final).

Por sua vez, todas as sequências S_1 geradas foram submetidas com sucesso, passando em quase a totalidade dos testes, com exceções pontuais. A sequência S_1 que apresentou mais falhas foi aquela correspondente ao arquivo de vídeo, tendo também de ser destacado que para a mesma não foi possível realizar o teste FFT (Fast Fourier Transform), que realiza análise espectral, o que provavelmente foi causado pela falta de memória RAM disponível[5] (ver "*orbits*" nas abas 01 a 30 da planilha Quadratic 1 -

[5]O programa emitiu a seguinte mensagem: Unable to allocate working arrays for the DFT.

Analysis (STS) - Final).

Por seu turno, as sequências S_2 e texto cifrado correspondentes aos arquivos de texto, imagem e vídeo também não foram aptas de serem avaliadas, e a mensagem *READ ERROR: Insufficient data in file* mais uma vez se fez presente. No que lhe concerne, as sequências S_2 e texto cifrado correlatas ao arquivo de vídeo foram foram analisadas com êxito, passando na maioria dos testes, salvo algumas exceções, embora o teste FFT não tenha sido realizado em função do mesmo erro apontado no parágrafo supra.

A planilha contendo os resultados pode ser acessada neste link: Quadratic 1 - Analysis (STS) - Final.

5.3.3.2 Mapeamento Quadrático $f(x) = -x^2 + r$

Os textos planos correspondentes aos arquivos de texto, imagem e áudio não puderam ser avaliados pela suíte de teste STS. Quando submetidos, o programa emitiu a seguinte informação: *READ ERROR: Insufficient data in file*. Desta feita, não foi possível avaliar a aleatoriedade dos mesmos. Já o arquivo de vídeo foi avaliado, mas falhou em todas as instâncias dos testes (ver aba "Plain Text"da planilha Quadratic 2 - Analysis (STS) - Final).

Por sua vez, todas as sequências S_1 geradas foram submetidas com sucesso, passando em quase a totalidade dos testes, com exceções pontuais. A sequência S_1 que apresentou mais falhas foi aquela correspondente ao arquivo de vídeo, tendo também de ser destacado que para a mesma não foi possível realizar o teste FFT (Fast Fourier Transform), que realiza análise espectral, o que provavelmente foi causado pela falta de memória RAM disponível[6] (ver "*orbits*" nas abas 01 a 30 da planilha Quadratic 2 - Analysis (STS) - Final).

Por seu turno, as sequências S_2 e texto cifrado correspondentes aos arquivos de texto, imagem e vídeo também não foram aptas de serem avaliadas, e a mensagem *READ ERROR: Insufficient data in file* mais uma vez se fez presente. No que lhe concerne, as sequências S_2 e texto cifrado correlatas ao arquivo de vídeo foram foram analisadas com êxito, passando na maioria dos testes, salvo algumas exceções, embora o teste FFT não tenha sido realizado em função do mesmo erro apontado no parágrafo supra (ver "*orbits*" nas abas 01 a 30 da planilha Quadratic 1 - Analysis (STS) - Final).

[6] O programa emitiu a seguinte mensagem: Unable to allocate working arrays for the DFT.

Para algumas sequências, os testes de *RandomExcursions* e *RandomExcursionsVariant* não foram realizados. O programa emitiu a seguinte mensagem: *WARNING: TEST NOT APPLICABLE. THERE ARE AN INSUFFICIENT NUMBER OF CYCLES.*

A planilha contendo os resultados pode ser acessada neste link: Quadratic 2 - Analysis (STS) - Final.

5.3.3.3 Mapeamento Logístico $f(x) = f(x) = rx(1-x)$

Os textos planos correspondentes aos arquivos de texto, imagem e áudio não puderam ser avaliados pela suíte de teste STS. Quando submetidos, o programa emitiu a seguinte informação: *READ ERROR: Insufficient data in file.* Desta feita, não foi possível avaliar a aleatoriedade dos mesmos. Já o arquivo de vídeo foi avaliado, mas falhou em todas as instâncias dos testes (ver aba "Plain Text"da planilha Logistic - Analysis (STS) - Final).

Por sua vez, todas as sequências S_1 geradas foram submetidas com sucesso, passando em quase a totalidade dos testes, com exceções pontuais. A sequência S_1 que apresentou mais falhas foi aquela correspondente ao arquivo de vídeo, tendo também de ser destacado que para a mesma não foi possível realizar o teste FFT (Fast Fourier Transform), que realiza análise espectral, o que provavelmente foi causado pela falta de memória RAM disponível[7] (ver "*orbits*" nas abas 01 a 30 da planilha Logistic - Analysis (STS) - Final).

Por seu turno, as sequências S_2 e texto cifrado correspondentes aos arquivos de texto, imagem e vídeo também não foram aptas de serem avaliadas, e a mensagem *READ ERROR: Insufficient data in file* mais uma vez se fez presente. No que lhe concerne, as sequências S_2 e texto cifrado correlatas ao arquivo de vídeo foram foram analisadas com êxito, passando na maioria dos testes, salvo algumas exceções, embora o teste FFT não tenha sido realizado em função do mesmo erro apontado no parágrafo supra (ver "*orbits*" e "*Encrypted*" nas abas 01 a 30 da planilha Logistic - Analysis (STS) - Final).

A planilha contendo os resultados pode ser acessada neste link: Logistic - Analysis (STS) - Final).

[7] O programa emitiu a seguinte mensagem: Unable to allocate working arrays for the DFT.

5.3.3.4 Análise

A suíte de testes NIST STS, possivelmente por ter sua origem é uma agência governamental dos EUA, é considerada a suíte de testes padrão para a indústria. Mas como os resultados coligidos nas planilhas deixam patente, certos arquivos não puderam ser testados devido ao tamanho das sequências. Ademais, das suítes de testes utilizadas, esta é a que apresenta (para alguns testes, obviamente) um elevado grau de exigência de memória.

No geral, observa-se que os arquivos que foram submetidos com sucesso apresentaram poucos testes que falharam, o que corrobora os testes anteriores realizados pelas suítes de testes ENT e Dieharder.

Por derradeiro, da mesma forma como apresentada pelos resultados dos testes das suíte ENT e Dieharder, comprova-se que o *texto cifrado* difere em muito do *texto plano*.

5.4 Comparação

Nesta seção, serão objeto de testes 3 arquivos de sequências binárias consideradas verdadeiramente aleatórias. Esses arquivos são provenientes das seguintes fontes: ANU Quantum Random Numbers Server, Random.Org e Araneus Alea II.

Os arquivos de teste são:

- `ANU_13Oct2017_100MB_1`: tamanho de 100MB, obtido no seguinte link: ANU - 13Oct2017.
- `RandomNumbers`: tamanho de 16384 bytes, obtido no seguinte link: Random Byte Generator.
- `alea-sample-10M.bin`: tamanho 10MB, obtido no seguinte link: Raw Binary Data.

Os resultados da aplicações dos testes foram os seguintes:

- **ENT**: O arquivo proveniente do serviço *ANU QUANTUM RANDOM NUMBERS SERVER* passou em todos os testes; o arquivo proveniente do serviço *Random Byte Generator* falhou no teste de média; o arquivo de amostra do gerador Araneus Alea II passou em todos os testes (Tabela 5.1).

- **Dieharder**: O arquivo proveniente do serviço *ANU QUANTUM RANDOM NUMBERS SERVER* apresentou poucas falhas a saber, cinco em um total de 115 instâncias; o arquivo proveniente do serviço *Random Byte Generator* falhou em todos os testes; o arquivo de amostra do gerador Araneus Alea II apresentou um misto de sucessos e falhas (Tabela 5.2).

- **STS**: O arquivo proveniente do serviço *ANU QUANTUM RANDOM NUMBERS SERVER* apresentou somente uma falha; o arquivo proveniente do serviço *Random Byte Generator* apresentou duas falhas, bem como não foi possível realizar os testes *RandomExcursions* e *RandomExcursionsVariant*[8]; o arquivo de amostra do gerador Araneus Alea II apresentou apenas uma falha (Tabela 5.3).

Tabela 5.1: Resultados dos testes ENT para sequências binárias verdadeiramente aleatórias.

	Testes				
Gerador	**Entropia**	**Qui-Quadrado**	**Média**	**Pi**	**CCS**
ANU	Success	Success	Success	Success	
Random	Success	Success	Fail	Success	Success
Alea II	Success	Success	Success	Success	Success

Tabela 5.2: Resultados dos testes Dieharder para sequências binárias verdadeiramente aleatórias.

Teste	**Quantum**	**RandomNumbers**	**Alea II**
diehard_birthdays	Passed	Failed	Passed
diehard_operm5	Passed	Failed	Passed
diehard_rank_32x32	Passed	Failed	Failed
diehard_rank_6x8	Passed	Failed	Passed
diehard_bitstream	Passed	Failed	Weak
diehard_opso	Weak	Failed	Failed
diehard_oqso	Passed	Failed	Passed
diehard_dna	Passed	Failed	Failed
diehard_count_1s_str	Passed	Failed	Passed
diehard_count_1s_byt	Passed	Failed	Failed
diehard_parking_lot	Weak	Failed	Passed
diehard_2dsphere	Passed	Failed	Passed
diehard_3dsphere	Passed	Failed	Passed
diehard_squeeze	Passed	Failed	Weak
diehard_sums	Weak	Weak	Passed
diehard_runs	Passed	Failed	Passed
diehard_runs	Passed	Failed	Weak

[8]Warning: Test not applicable. There are an insufficient number of cycles.

diehard_craps	Passed	Failed	Weak
diehard_craps	Weak	Failed	Passed
marsaglia_tsang_gcd	Passed	Failed	Failed
marsaglia_tsang_gcd	Failed	Failed	Failed
sts_monobit	Passed	Failed	Passed
sts_runs	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Weak	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
sts_serial	Passed	Failed	Passed
rgb_bitdist	Passed	Failed	Passed
rgb_bitdist	Passed	Failed	Passed
rgb_bitdist	Passed	Failed	Passed
rgb_bitdist	Weak	Failed	Passed
rgb_bitdist	Passed	Failed	Passed
rgb_bitdist	Passed	Failed	Passed

rgb_bitdist	Passed	Failed	Passed
rgb_bitdist	Passed	Failed	Passed
rgb_bitdist	Passed	Failed	Passed
rgb_bitdist	Passed	Failed	Passed
rgb_bitdist	Passed	Failed	Passed
rgb_bitdist	Passed	Failed	Weak
rgb_minimum_distance	Passed	Failed	Passed
rgb_minimum_distance	Passed	Failed	Passed
rgb_minimum_distance	Passed	Failed	Failed
rgb_minimum_distance	Passed	Failed	Passed
rgb_permutations	Passed	Failed	Passed
rgb_permutations	Passed	Failed	Passed
rgb_permutations	Passed	Failed	Passed
rgb_permutations	Passed	Failed	Passed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Failed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Weak	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Weak	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Weak	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Failed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Weak	Failed	Failed

rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Weak	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_lagged_sum	Failed	Failed	Failed
rgb_lagged_sum	Passed	Failed	Failed
rgb_kstest_test	Passed	Failed	Passed
dab_bytedistrib	Failed	Failed	Failed
dab_dct	Passed	Failed	Weak
dab_filltree	Weak	Failed	Failed
dab_filltree	Passed	Failed	Failed
dab_filltree2	Passed	Failed	Failed
dab_filltree2	Weak	Failed	Failed
dab_monobit2	Failed	Failed	Failed

Tabela 5.3: Resultados dos testes STS para sequências binárias verdadeiramente aleatórias.

Resultados			
Testes	**Quantum**	**RandomNumbers**	**Alea II**
Frequency	Success	Success	Success
BlockFrequency	Success	Success	Success
CumulativeSums	Success	Success	Success
CumulativeSums	Success	Success	Success
Runs	Success	Success	Success
LongestRun	Success	Success	Success
Rank	Success	Success	Success
FFT	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success

NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success

NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success

NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Fail	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success

NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Success	Success	Success
NonOverlappingTemplate	Fail	Success	Fail
OverlappingTemplate	Success	Success	Success
Universal	Success	Success	Success
ApproximateEntropy	Success	Success	Success
RandomExcursions	Success	—	Success
RandomExcursions	Success	—	Success
RandomExcursions	Success	—	Success
RandomExcursions	Success	—	Success
RandomExcursions	Success	—	Success
RandomExcursions	Success	—	Success
RandomExcursions	Success	—	Success
RandomExcursions	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
RandomExcursionsVariant	Success	—	Success
Serial	Success	Success	Success
Serial	Success	Success	Success
LinearComplexity	Success	Success	Success

5.4.1 Análise

O objetivo desta seção é verificar por meio dos mesmos testes estatísticos, se sequências gerados por *TRNGs* apresentam falhas comparáveis às sequências geradas neste trabalho. Dado que a quantidade de testes realizados foi inexpressiva, as conclusões não são definitivas. Entretanto:

- Os resultados dos testes da suíte ENT aparentemente se mostraram melhores, com apenas um desvio.
- Os resultados dos testes da suíte Diehard exibiram possíveis fraquezas nas sequências geradas pelos TRNGs[9].
- Os resultados dos testes da suíte STS exibiram excelentes resultados.

A questão que agora pode ser levantada é: qual suíte de testes oferece o melhor conjunto de testes? Bom, essa é uma questão que não pode ser respondida aqui. Porque? Bem, primeiro considere que nenhum conjunto de testes é totalmente conclusivo a respeito (ou não) da aleatoriedade de uma dada sequência. Segundo, há a necessidade de se aferir quais testes são mais significativos, quais apresentam maior acurácia e precisão, etc. Ainda, há de se considerar as implementações dos mesmos. Assim, a tarefa de responder a esta pergunta constitui-se em outro objeto de pesquisa.

Por fim, há de se destacar que as sequência binárias produzidas neste labor, não foram objeto de pós-tratamento algum (pode-se dizer que foram usadas em estado bruto, se preferir), com vistas a eliminar/diminuir tendências que produzem desvios que podem levar a não uniformidade. Assim, futuramente, pode-se pensar em adicionar ao método os meios necessários para mitigar possíveis tendências.

[9]Contudo, considerando que sequências menores devem mostrar características estatísticas mais fracas do que sequências maiores, o ideal seria testar sequências de tamanhos crescentes e equivalentes, o que não foi aqui possível - mesmo porque este trabalho não tem como fim o teste de TRNGs.

Capítulo 6

Conclusão e Trabalhos Posteriores

6.1 Síntese e Análise

Neste trabalho, foi proposta (e implementada) uma cifra de encriptação simétrica baseada em caos[1], mais especificamente, foi empregado *funções caóticas unidimensionais* e o *jogo do caos* a fim de produzir sequências binárias *pseudo*-aleatórias. A primeira sequência S_1 é gerada por uma função caótica unidimensional que é utilizada pelo *jogo do caos* para produzir a sequência S_2. Por sua vez, esta última é combinada com o *texto plano* a fim de produzir o *texto cifrado*.

Qual a razão de duas etapas que produzem números pseudo-aleatórios? Existem duas. A primeira razão é o fato de que o jogo do caos necessita como entrada de um fluxo de números aleatórios. Entretanto, se tal entrada é realmente de números verdadeiramente aleatórios (que não são reproduzíveis), então pode-se utilizar one-time pad da forma original e nenhum benefício sobre suas desvantagens é ganho. Contudo, se uma sequência pseudo-aleatória possui propriedades similares a uma sequência verdadeiramente aleatória, a mesma pode servir de entrada para o jogo do caos de forma satisfatória, sendo efetivamente desconsiderada após seu uso. A segunda razão é o acoplamento de duas fases deveria (em princípio) melhorar a robustez[2] do sistema. Contudo, somente análise criptográfica pode afirmar ou negar isso.

Por sua vez, os pressupostos teóricos que embasam a produção das sequências S_1, S_2 e *texto cifrado* foram abordados ao longo do trabalho, mas mais especificamente na Subseção 4.1.1.

[1] Mais especificamente, nas propriedades de funções caóticas.

[2] Duas propriedades que um bom sistema criptográfico deveria possuir para dificultar a análise estatística são *difusão* e *confusão*. No método proposto, a difusão é alcançada pela *dependência sensível sobre as condições iniciais*, ao passo que a confusão por meio da *ergodicidade* (ver Tabela 2.1)

Por derradeiro, *texto plano*, *texto cifrado*, sequências S_1 e S_2, foram submetidas a testes de aleatoriedade, cujos resultados encontram-se coligidos no Capítulo 5. Estes resultados "parecem" indicar que as sequências geradas possuem boas qualidades em relação ao quesito aleatoriedade. Para tanto, considere que se a sequência S_1 não fosse suficientemente aleatória, então os vértices escolhidos para o jogo do caos teriam um viés, isto é, certos vértices seriam selecionados com mais frequência do que outros, o que por fim não produziria uma distribuição supostamente uniforme (ver Figura 4.5). Ainda, embora os resultados das suítes de testes estatísticos indiquem que as sequências S_1 são possivelmente mais aleatórias do que as sequências S_2 e texto cifrado, isso é apenas uma questão de tamanho, dado que as sequências S_1 são 8 vezes maiores do que S_2 e o texto cifrado. Por fim, embora algumas sequências não tenham sido passíveis de submissão a suíte STS, isso também é uma questão de tamanho (da sequência), pois as sequências geradas para os arquivos de vídeo são as mesmas para os outros arquivos (possuem as mesmas condições iniciais e usam o mesmo procedimento), diferindo apenas no comprimento.

6.2 Trabalhos Futuros

A bem da verdade, existem vários objetivos derivados do presente trabalho que podem ser perseguidos futuramente. Dentre os quais, pode-se destacar:

(1) Implementação de outras funções caóticas unidimensionais (ou até mesmo multidimensionais) que servirão para a geração da sequências S_1.

(2) Aumentar o número de modos no qual a sequência S_1 pode ser construída, verificando se estas sequências geradas possuem as propriedades estatísticas desejadas.

(3) Aumentar o número de modos no qual a sequência S_2 pode ser construída, verificando se estas sequências geradas possuem as propriedades estatísticas desejadas.

(4) Implementar um modo pelo qual as sequências S_1 e S_2 possam ser produzidas em paralelo (usando múltiplos threads).

(5) Realizar os mesmos testes estatísticos com outras cifras a fim de obter uma comparação entre estas e o método proposto.

(6) Submeter o método proposto a criptoanálise.

Com relação ao item (1), para principiar, poder-se-ia começar implementando as duas funções não utilizadas (embora listadas), a saber: (a) $f(x) = r.sin(x)$ e (b) $f(x) = r.cos(x)$. Ambas as funções, similarmente a aquelas implementadas, "*parecem*"[3]

[3] "Parecem" porque isso deve ser efetivamente comprovado.

exibir caos para certos valores de r, mas ao contrário daquelas, estes valores não são únicos (ver Figuras 6.1 e 6.2). Em adição, outras funções apresentando regiões caóticas podem ser encontradas e então utilizadas.

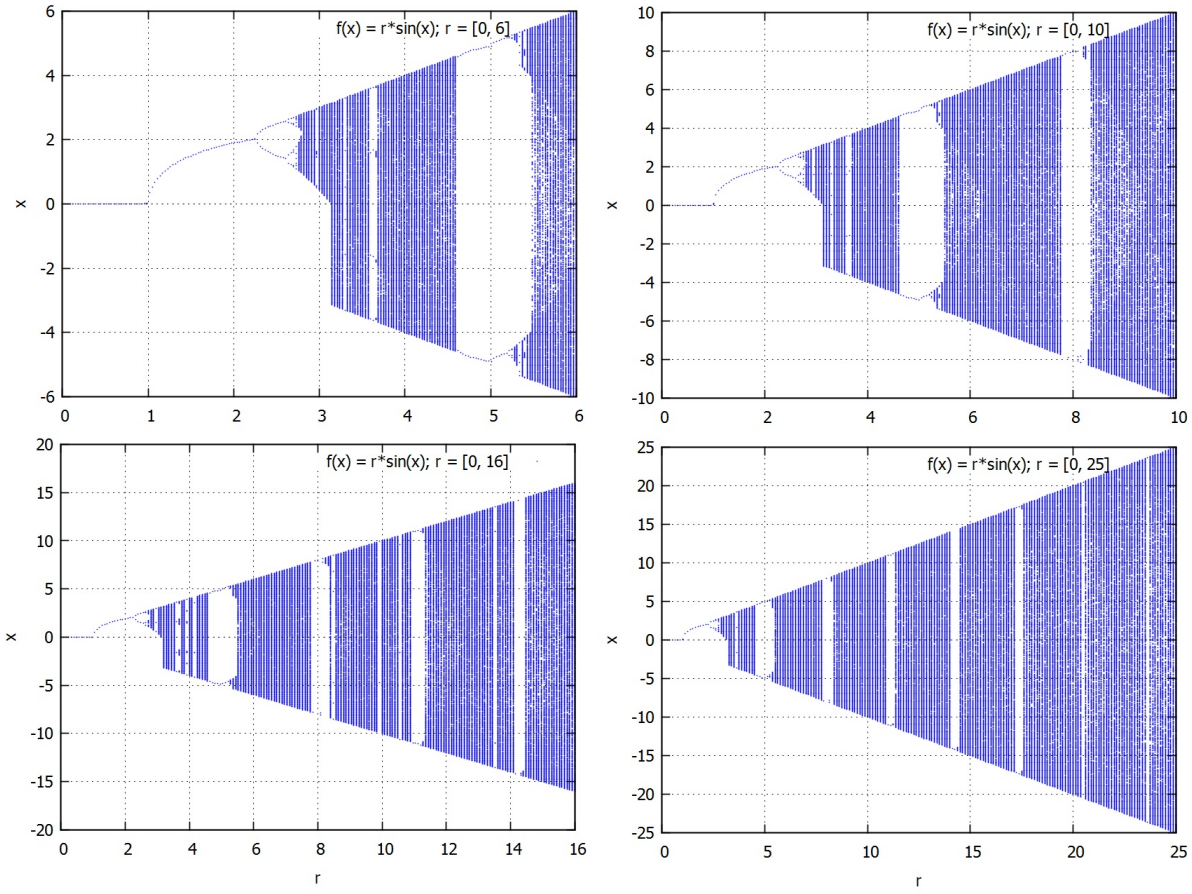


Figura 6.1: Diagramas de bifurcação para a função $f(x) = r.sin(x)$.

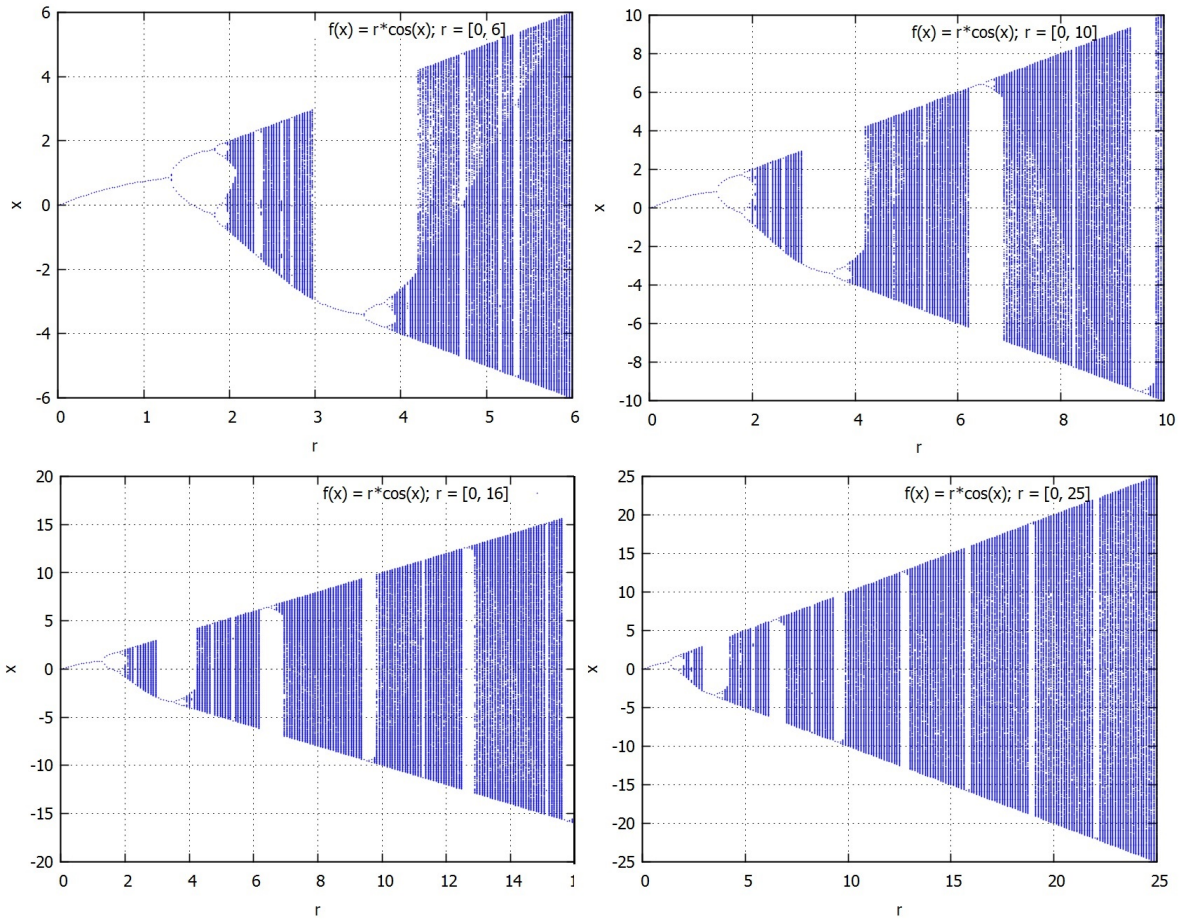


Figura 6.2: Diagramas de bifurcação para a função $f(x) = r.cos(x)$.

Com relação ao item (2), a sequência binária S_1 é construída byte a byte no formato *little endian*. Contudo, poderia ser escolhido o formato *big endian*. De toda a feita, como cada byte é constituído por 8 bits, existem 8! = 40320 modos diferentes de escolher a ordem do bits. Desta feita, poder-se-ia, a princípio, escolher uma destas ordens e produzir a sequência adequadamente.

Para o item (3), deve-se considerar a ordem no qual os vértices são escolhidos. Para um quadrado (que é o utilizado aqui), existe 4! = 24 formas, e deste modo, pode-se construir um número equivalente de sequências. Obviamente, tal disposição deve ser objeto de testes, assim como ocorre para o item (2).

Para o item (4), deve ser considerado que a construção das sequências binárias se dá basicamente de forma sequencial. Contudo, existe uma forma de implementar esta construção de forma semelhante a um *pipeline*. Se esta forma de realização proverá ganhos de velocidade, ou mesmo de segurança, isto deverá ser averiguado.

Para o item (5), considere que nem todos os métodos criptográficos se prestam para arquivos de mídia, por exemplo, compare as Figuras 6.3 e 6.4. A primeira foi

encriptada usando AES (Advanced Encryption Standard), já a segundo foi encriptada utilizando o método proposto neste trabalho[4]. Manifestamente, pelo menos em nível perceptual, o método proposto é superior. Contudo, como quando se trata de arquivos de mídia, outras métricas podem (e devem) ser utilizadas.

(a) Original (b) Encrypted

Figura 6.3: Imagem (a) encriptada diretamente (b) usando AES. Fonte: (Lian, 2009).

Figura 6.4: Imagem original e imagem encriptada usando o método proposto no trabalho.

O item remanescente, submissão do método proposto a criptoanálise, constitui-se em outro trabalho. Contudo, há de se notar que métodos de criptoanálise para esquemas criptográficos baseados em caos não se encontram no mesmo nível de amadurecimento (se for permitido o uso dessa palavra) dos métodos de criptoanálise para

[4]A bem da verdade, a figura teve de ter o cabeçalho recomposto de modo a ser passível de leitura.

métodos convencionais, assim, aparentemente, tais métodos precisam serem desenvolvidos ao perseguir a meta do presente item.

Apêndice A

Baterias de Testes (Estatísticos) Empregadas

A.1 ENT

ENT[1] é um programa que aplica vários testes a sequências de bytes armazenados em arquivos e relata os resultados destes testes. A finalidade destes testes é avaliar geradores de números pseudo-aleatórios para aplicações criptográficas, amostragem estatística e algoritmos de compressão, bem como outros programas onde a densidade da informação de um arquivo é de interesse.

ENT realiza um total de 5 testes no fluxo de bytes do arquivo fornecido como entrada (ou a partir da entrada padrão se nenhuma arquivo é especificado): entropia, teste Chi-quadrado, média aritmética, valor de π calculado pelo método de Monte Carlo e coeficiente de correlação serial. Os resultados dos testes são encaminhado para a saída padrão (tela), tendo a seguinte a aparência[2]:

```
Value Char Occurrences Fraction
  0        419415351   0.499982
  1        419445449   0.500018

Total:     838860800   1.000000

Entropy = 1.000000 bits per bit.

Optimum compression would reduce the size
of this 838860800 bit file by 0 percent.

Chi square distribution for 838860800 samples is 1.08, and randomly
would exceed this value 29.87 percent of the times.

Arithmetic mean value of data bits is 0.5000 (0.5 = random).
```

[1] Disponível em: http://www.fourmilab.ch/random/.

[2] Dependendo das opções escolhidas a impressão é exibida em outros formatos.

```
Monte Carlo value for Pi is 3.141692625 (error 0.00 percent).
Serial correlation coefficient is -0.000026 (totally uncorrelated = 0.0).
```

A propósito, o exemplo de saída exibido refere-se a um arquivo binário aleatório de 100MB obtido junto a ANU Quantum Random Numbers Server, que é um serviço online (gratuito!) que oferece números verdadeiramente aleatórios[3] a qualquer um.

Os cinco testes realizados por ENT possuem os seguintes significados:

- **Entropia:** Densidade de informação do conteúdo do arquivo expresso como o número de bits por caractere (a mensagem que segue informa o quanto o conteúdo em questão pode ser passível de compressão).
- **Teste Chi-quadrado:** Cálculo da distribuição χ-quadrado para o fluxo de bytes contidos no arquivo e expressa como um valor absoluto e uma porcentagem que indica o quão frequentemente uma sequência verdadeiramente aleatória deveria exceder o valor calculado. A porcentagem é interpretada como o grau para o qual a sequência testada é suspeita de ser não aleatória, como segue:
  - Se a porcentagem é maior do que 99% ou menor do que 1%, a sequência quase certamente não é aleatória.
  - Se a porcentagem está entre 95-99% ou entre 1-5%, a sequência é suspeita.
  - Se a porcentagem está entre 90-95% ou entre 5-10%, a sequência é quase suspeita.
- **Média aritmética:** Resultado da soma de todos os bits (ou bytes) no arquivo divididos pelo comprimento do arquivo. Se os dados estão próximos de serem aleatórios, a média aritmética deveria ser aproximadamente 0.5 para sequências binárias (127.5 considerando a sequência como sendo composta por bytes). Se a média se afasta deste valor, os valores são consistentemente altos ou baixos.
- **Valor de π calculado pelo método de Monte Carlo:** Cada sequência sucessiva de seis bytes é usada como coordenadas (x, y) dentro de um quadrado. Se a distância do ponto gerado aleatoriamente é menor do que o raio do círculo inscrito dentro do quadrado, então a sequência de seis bytes é considerada como um acerto. A porcentagem de acertos pode ser usada para calcular o valor de π, sendo que para fluxos muito grandes (a aproximação converge muito lentamente), o valor irá se aproximar do valor correto de π se a sequência está próxima de ser aleatória.

[3] Per si, os números aleatórios são gerados por meio da medição de flutuações quânticas do vácuo, que diferentemente do conceito de vácuo no contexto clássico (considerado um espaço vazio sem matéria ou fótons), é preenchido por partículas virtuais que surgem e desaparecem o tempo todo, o que faz com que o campo eletromagnético do vácuo exiba flutuações aleatórias de fase e amplitude em todas as frequências.

- **Coeficiente de correlação serial:** Mede a extensão para o qual cada byte no arquivo depende do byte anterior. Se a sequência é aleatória, este valor (que pode ser positivo ou negativo) será próximo de zero.

> **Nota:** A observação "optimal compression" mostrada para o arquivo é computada a partir da entropia do fluxo de bits (bytes), e portanto, reflete a compressibilidade baseada na leitura do comprimento da estrutura escolhida (bits ou bytes). Algoritmos que usam uma estrutura de leitura maior podem alcançar maior compressão se o arquivo contém sequências repetidas de múltiplos bytes.

A.2 Dieharder

O programa dieharder, desenvolvido por Robert G. Brown, tem como propósito é fornecer uma suíte de testes, tão sistemática quanto possível, no qual geradores de números aleatórios de todos os tipos podem ser avaliados. Pode-se dizer que dieharder é um derivado de *diehard*, uma bateria de testes estatísticos desenvolvida pelo Prof. George Marsaglia, objetivada a medir a qualidade de um gerador de números aleatório, cujo primeiro lançamento se deu no ano de 1995 por meio de um CD-ROM contendo várias sequências binárias de números aleatórios, bem com os programas correlatos para as plataformas DOS, Linux e SUN.

Por muito tempo considerada a suíte padrão para testes de aleatoriedade, diehard possuía várias fraquezas, dentre os quais a falta de bons testes em nível de bits e força criptográfica[4]. Outra falha em diehard era a falta de variabilidade paramétrica. Apesar de muitos testes em diehard serem sensíveis e capazes de demonstrar a falta de aleatoriedade em geradores com tipos particulares de correlações internas, falta na suíte o poder de discriminar claramente as falhas. Considerando isto, dieharder procura melhorar aonde possível os testes de diehard (que são todos implementados), bem como fornece alguns testes disponíveis em STS e alguns testes propostos por Donald E. Knuth.

Em dieharder, a comunicação entre o computador e o usuário é realizada por meio de uma interface baseada em linha de comandos. Ao ser emitido o comando `diehard`, é exibida a seguinte tela (exibida aqui de forma abreviada):

[4]Isto levou ao desenvolvimento do STS pelo NIST.

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#

Usage:

dieharder [-a] [-d dieharder test number] [-f filename] [-B]
          [-D output flag [-D output flag] ... ] [-F] [-c separator]
          [-g generator number or -1] [-h] [-k ks_flag] [-l]
          [-L overlap] [-m multiply_p] [-n ntuple]
          [-p number of p samples] [-P Xoff]
          [-o filename] [-s seed strategy] [-S random number seed]
          [-n ntuple] [-p number of p samples] [-o filename]
          [-s seed strategy] [-S random number seed]
          [-t number of test samples] [-v verbose flag]
          [-W weak] [-X fail] [-Y Xtrategy]
          [-x xvalue] [-y yvalue] [-z zvalue]
...
```

Dieharder disponibiliza um total de 31 testes, que podem ser listados por meio do comando `dieharder -l`:

```
#=============================================================================
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown
#=============================================================================
Installed dieharder tests:
 Test Number                          Test Name               Test Reliabi
=============================================================================
  -d 0                            Diehard Birthdays Test             Good
  -d 1                               Diehard OPERM5 Test             Good
  -d 2                     Diehard 32x32 Binary Rank Test            Good
  -d 3                       Diehard 6x8 Binary Rank Test            Good
  -d 4                            Diehard Bitstream Test             Good
  -d 5                                     Diehard OPSO           Suspect
  -d 6                                  Diehard OQSO Test         Suspect
  -d 7                                   Diehard DNA Test         Suspect
  -d 8                 Diehard Count the 1s (stream) Test            Good
  -d 9                   Diehard Count the 1s Test (byte)            Good
  -d 10                         Diehard Parking Lot Test             Good
  -d 11          Diehard Minimum Distance (2d Circle) Test           Good
  -d 12          Diehard 3d Sphere (Minimum Distance) Test           Good
  -d 13                              Diehard Squeeze Test            Good
  -d 14                                 Diehard Sums Test        Do Not Use
  -d 15                                 Diehard Runs Test            Good
  -d 16                                Diehard Craps Test            Good
  -d 17                     Marsaglia and Tsang GCD Test             Good
  -d 100                                 STS Monobit Test            Good
  -d 101                                    STS Runs Test            Good
```

```
-d 102                  STS Serial Test (Generalized)          Good
-d 200                      RGB Bit Distribution Test          Good
-d 201          RGB Generalized Minimum Distance Test          Good
-d 202                          RGB Permutations Test          Good
-d 203                            RGB Lagged Sum Test          Good
-d 204               RGB Kolmogorov-Smirnov Test Test          Good
-d 205                            Byte Distribution          Good
-d 206                                    DAB DCT            Good
-d 207                            DAB Fill Tree Test         Good
-d 208                          DAB Fill Tree 2 Test         Good
-d 209                            DAB Monobit 2 Test         Good
```

A partir desta lista, verifica-se que os testes disponibilizados pela suíte diehard estão implementados, bem como alguns da suíte STS e vários outros.

Uma descrição dos testes é como segue:

- **Diehard Birthdays Test:** Cada teste determina o número de intervalos correspondentes a partir de 512 aniversários retirados (por padrão) de um ano de 24 bits (por padrão). Isto é repetido 100 vezes (por padrão) e os resultados acumulados em um histograma. Se o gerador é suficientemente aleatório, os intervalos repetidos deveriam estar distribuídos conforme uma distribuição de Poisson. Um χ^2 e um p-value para o teste são avaliados com relação a hipótese nula.

- **Diehard OPERM5 Test:** O teste trabalha em uma sequência de um milhão de inteiros aleatórios de 32 bits. Cada conjunto de cinco inteiros consecutivos pode ser um dos 120 estados (existem 5! possíveis ordenações de cinco números). Assim, o quinto, sexto, sétimo, ... números, cada um, fornecem um estado. Conforme milhares de transições de estado são observadas, contagens cumulativas são feitas dos números de ocorrências de cada estado. Então, a forma quadrática da inversa fraca da matriz de covariância de dimensão 120×120 produz um teste equivalente ao teste da razão de verossimilhança que se se origina da contagem de 120 células (assintoticamente) de uma distribuição normal especificada com a matriz de covariância de dimensão 120×120 (com posto, ou característica, igual a 99).

- **Diehard 32x32 Binary Rank Test:** Uma matriz binária aleatória de ordem 32×32 é formada, sendo cada linha um inteiro de 32 bits aleatório. O posto é determinado, sendo que este posto pode estar entre 0 a 32. Postos menores do que 29 são raros, e suas contagens são coletadas junto com aquelas de posto 29. Postos são encontrados para 40000 de tais matrizes e o teste χ^2 é executado nas contagens dos postos 32, 31, 30 e ≤ 29. O teste é repetido e um teste KS é aplicado

aos p-values resultantes para verificar se eles se aproximam de uma distribuição uniforme.

- **Diehard 6x8 Binary Rank Test:** A partir de cada matriz de inteiros de 32 bits aleatórios do gerador sob teste, um byte específico é escolhido, e os seis bytes resultantes formam uma matriz binária de ordem 6×8 cujo posto é determinado. Aquele posto pode variar de 0 a 6, mas postos 0, 1, 2 e 3 são raros; suas contagens são coletadas junto com ao posto 4. Postos são encontrados para 100000 matrizes aleatórias, e o teste χ^2 é executado para os postos 6, 5 e ≤ 4. O teste é repetido e um teste KS é aplicado aos p-values resultantes para verificar se eles se aproximam de uma distribuição uniforme.

- **Diehard Bitstream Test:** O arquivo sob teste é visto como um fluxo de bits. Chame estes bits de $b_1, b_2, b_3, \ldots$ Considere um alfabeto com duas "letras", 0 e 1, e pense que o fluxo de bits é uma sucessão de "palavras" de tamanho 20, com sobreposição. Assim, a primeira palavra é $b_1 b_2, \ldots, b_{20}$, a segunda é $b_2 b_3, \ldots, b_{21}$, e assim por diante. O teste de fluxo de bits conta o número de palavras faltantes de 20 letras (20 bits) em uma string de 2^{21} palavras de 20 letras (20 bits) que se sobrepõem. Existem 2^{20} possíveis palavras de 20 letras. Para um string verdadeiramente aleatória de $2^{21} + 19$ bits, o número j de palavras faltantes deveria estar muito próximo a uma distribuição normal com $\overline{x}$ (média) igual a 141909 e σ (desvio-padrão) igual a 428. Assim, $(j - 141909)/428$ deveria ser a variação padrão normal (z score) que leva a um p-value uniforme $[0, 1)$. O teste é repetido 20 vezes.

- **Diehard Overlapping Pairs Sparse Occupance (OPSO):** O teste OPSO considera palavras de duas letras de um alfabeto de 1024 letras. Cada letra é determinada por 10 bits especificados de um inteiro de 32 bits em uma sequência a ser testada. O teste OPSO gera 2^{21} palavras de 2 letras (com sobreposição) e conta o número de palavras faltantes - isto é, palavras de duas letras que não aparecem na sequência inteira. Aquela contagem deveria estar muito próxima a uma distribuição normal com $\overline{x}$ (média) igual a 141909 e σ (desvio-padrão) 290. Assim, $(mw - 141909)/290$ (mw - número de palavras faltantes) deveria ser uma variável normal padrão. O teste OPSO toma 32 bits por vez de um arquivo de teste e usa um conjunto designado de dez bits consecutivos. Ele então reinicializa o arquivo para os próximos 10 bits designados, e assim por diante.

- **Diehard Overlapping Quadruples Sparce Occupancy (OQSO) Test:** Teste similar ao OPSO, exceto que ele considera palavras de 4 letras de um alfabeto de 32 letras, onde cada letra é determinada por uma string determinada de 5 bits consecutivos do arquivo de teste, elementos que são assumidos serem inteiros ale-

atórios de 32 bits. O número médio de palavras faltantes em uma sequência de 2^{21} palavras de 4 letras novamente é 141909, com $\sigma = 295$. A média é baseada em teoria; σ vem de uma simulação extensiva.

- **Diehard DNA Test:** O teste DNA considera um alfabeto de 4 letras, C, G, A e T, determinadas por dois bits designados na sequência de inteiros aleatórios sendo testados. O teste considera palavas de 10 letras, assim como nos testes OPSO e OQSO. Existem 2^{20} palavras possíveis, e o número médio de palavas faltantes que vem de uma string de 2^{21} palavras (com sobreposição) é 141909. O desvio padrão σ = 339 foi determinado pela simulação OQSO.

- **Diehard Count the 1s (stream) (modified) Test:** Considera o arquivo sob teste como um fluxo de bytes (quatro por inteiro de 32 bits). Cada byte pode conter de 0 a 8 uns com probabilidades 1, 8, 28, 56, 70, 56, 28, 8, 1 sobre 256. Faça agora o fluxo de bytes fornecer uma string de palavras de 5 letras sobrepostas, cada "letra" tomando os valores A, B, C, D, E. As letras são determinadas pelo número de uns em um byte: 0, 1 ou 2 produzem A, 3 produz B, 4 produz C, 5 produz D e 6, 7 ou 8 produzem E. Assim, temos um macaco digitando cinco teclas com várias probabilidades (37, 56, 70, 56, 37 sobre 256). Existem 5^5 possíveis palavras de 5 letras, e a partir de uma sequência com 256000 palavras de cinco letras (sobreposta), contagens são feitas sobre as frequências de cada palavra. A forma quadrática da inversa fraca da matriz de covariância da célula de contagem fornece um teste χ^2: Q5-Q4, a diferença da soma de Pearson empírica de (OBS-EXP)^2/EXP sobre a contagem para as células de contagem para 4 e 5 letras.

- **Diehard Count the 1s Test (byte) (modified):** Este teste é semelhante ao anterior, mas é especificado para bytes. Considere que o arquivo sob teste como um fluxo de inteiros de 32 bits. A partir de cada inteiro, um byte específico é escolhido, digamos, o mais a esquerda: bits de 1 a 8. Cada byte pode conter de 0 a 8 uns, com probabilidades 1, 8, 28, 56, 70, 56, 28, 8, 1 sobre 256. Agora os bytes especificados de inteiros sucessivos fornecem uma string (com sobreposição) de palavras de 5 letras, cada "letra" tomando os valores A, B, C, D, E. As letras são determinadas pelo número de uns naquele byte: 0, 1 ou 2 → A, 3 → B, 4 → C, 5 → D e 6, 7 ou 8 → E. Assim, temos um macaco apertando 5 teclas com várias probabilidades: 37, 56, 70, 56, 37 sobre 256. Existem 5^5 possíveis palavras de 5 letras, e de uma string de 256000 palavras com 5 letras (com sobreposição), contagens são feitas sobre as frequências para cada palavra. A forma quadrática na forma da inversa fraca da matriz de covariância de cada célula de contagem fornece um teste χ^2: Q5-Q4, a diferença da soma de Pearson empírica de (OBS - EXP)^2/EXP sobre a contagem para as células de contagem para 4 e 5 letras.

- **Diehard Parking Lot Test (modified):** Este teste verifica a distribuição de tentativas para estacionar aleatoriamente um carro quadrado de comprimento 1 em um estacionamento de dimensão 100×100 sem colisão. Plota-se n (número de tentativas) versus k (número de tentativas que não colidem), porque os carros quadrados se sobrepõem, e compara o resultado esperado de um conjunto aleatório de coordenadas de estacionamento. Isto é, aliás, não se conhece sólidas bases teóricas para tal, assim se é comparado n = 12000, onde k deveria ser em média 3523 e σ = 21.9, sendo muito próximo de uma distribuição normal. Assim, $(k - 3523)/21.9$ é a variável normal padrão, que quando convertida para um p-value uniforme, fornece entrada para um teste KS com 100 amostras padrão.

- **Diehard Minimum Distance (2d Circle) Test:** Isto é feito 100 vezes - escolha n = 8000 pontos aleatórios em um quadrado de lado 10000. Encontre d, a distância mínima entre os $(n^2 - n)/2$ pares de pontos. Se os pontos são verdadeiramente uniformemente independentes, o quadrado da distância mínima deveria ser (muito próximo) a uma distribuição exponencial com média 0.995. Assim, $1 - exp(-d^2/.995)$ deveria ser uniforme em $[0, 1)$ e o teste KS nos 100 valores serve como teste de uniformidade para pontos aleatórios no quadrado.

- **Diehard 3d Sphere (Minimum Distance) Test:** Escolha 4000 pontos aleatórios em um cubo de aresta 1000. A cada ponto, centre uma esfera grande o suficiente para alcançar o próximo ponto mais perto. Então o volume da menor de tais esfera é muito próximo a uma distribuição exponencial com média $120\pi/3$. Assim, o raio cúbico é exponencial com média 30. O teste gera 4000 destas esferas 20 vezes. Cada raio mínimo ao cubo leva a uma variável uniforme por meio de $1 - exp(-r^3/30)$, então um teste KS é feito em 20 p-values.

- **Diehard Squeeze Test:** Inteiros aleatórios são convertidos para ponto flutuante de modo a serem uniformes em $[0, 1)$. Começando com $k = 2^{31}$ = 2147483647, o teste encontra j, o número de iterações necessárias para reduzir k para 1, usando a redução $k = ceiling(k * U)$, com U fornecido pelos inteiros convertidos para float do arquivo sendo testado. Tais j's são encontrados 100000 vezes, então a contagem para o número de vezes que j foi $\leq$ 6, 7, ..., 47, $\geq$ 48 são usados para fornecer um teste χ^2 para frequência de células.

- **Diehard Sums Test:** Inteiros são convertidos para floats para obter uma sequência $U(1), U(2), \ldots$ de variáveis uniformes em [0,1). então são formadas somas sobrepostas, $S(1) = U(1) + \ldots + U(100)$, $S2 = U(2) + \ldots + U(101), \ldots$ Estes S's são virtualmente normais com uma certa matriz de covariância. Uma transformação linear dos S's os converte para uma sequência de variáveis normais padrão

independentes, que são convertidas para variáveis uniformes para o teste KS. Os p-values de 10 testes KS são dados para outro teste KS.

- **Diehard Runs Test:** Este teste conta as subidas e descidas em uma sequência de variáveis uniformes no intervalo $[0, 1)$, obtidas pela transformação em ponto flutuante de inteiros de 32 bits do arquivo especificado. As matrizes de covariância para as subidas e descidas são bem conhecidas, levando a testes χ^2 para formas quadráticas nas inversas fracas das matrizes de covariância. As subidas e descidas são contadas para sequências de comprimento 10000. Isso é feito 10 vezes, e então repetido.

- **Diehard Craps Test:** Este teste realiza 200000 jogos de craps, encontra o número de vitórias e o número de lançamentos necessários para terminar cada jogo. O número de vitórias deveria ser muito próximo de uma distribuição normal com média $200000p$ e variância $200000p(1 - p)$, com $p = 244/495$. Lançamentos necessários para completar o jogo podem variar de 1 a ∞, mas as contagens para todos os maiores do que 21 são agrupados com 21. Um teste χ^2 é feito sobre a contagem do número de lançamentos. Cada inteiro de 32 bits do arquivo de teste fornece o valor para o lançamento de um dado pela transformação do mesmo em um número ponto-flutuante no intervalo $[0, 1)$, multiplicando-o por 6 e tomando 1 mais a parte inteira do resultado.

- **Marsaglia and Tsang GCD Test:** 10^7 amostras (por padrão) dos inteiros não sinalizados u e v são gerados, bem como duas estatísticas, o maior divisor comum deles $(GCD)(w)$ e o número de passos no método de Euclides necessários para o encontrar (k). Duas tabelas de frequências são assim geradas: uma para o número de vezes para cada valor de k na faixa de 0 a 41 (com contagens maiores do que esta faixa agrupados nos pontos finais), e outra para a frequência de ocorrências de cada $GCD(w)$. k está distribuído de forma aproximadamente binomial, mais isto é inútil para os propósitos de executar um teste rigoroso. Em vez disso, quatro "bons" RNGs (gfsr4, mt19937_1999, rndlxs2, taus2) foram usados para construir uma tabela simulada de alta precisão de probabilidades para k (um processo que obviamente levanta a questão de que se estes geradores são "bons"ou não). De qualquer forma, eles produzem tabelas muito similares e passam no teste com as tabela um do outro (e são, de outro modo, RNGs muito diferentes). A tabela de probabilidades para a distribuição *gcd* é gerada dinamicamente por teste (é fácil de computar). Testes χ^2 em ambas destas distribuições produzem dois p-values por teste e 100 (por padrão) p-values de cada um são acumulados e sujeitos aos testes KS finais e exibidos em um histograma

- **STS Monobit Test:** Conta os bits 1 em uma longa string de inteiros aleatórios

não sinalizados. Compara o número esperado e gera um p-value diretamente a partir da função $erfc()$. Muito efetivo para revelar geradores abertamente fracos; não tão bom para determinar se geradores fortes eventualmente irão falhar.

- **STS Runs Test:** Conta o número total de runs de 0 mais o número total de runs de 1 através de uma amostra de bits (run é uma sequência ininterrupta de bits idênticos). Note que um run 0 deve começar com 10 e terminar com 01. Note que um run 1 deve começar com 01 e terminar com 10. Este teste, executado em uma string de bits com condições de contorno cíclicas, é absolutamente equivalente a apenas contar os pares de bits 01 + 10. Ele é, portanto, totalmente redundante, mas não tão bom quanto o teste rgb_bitdist() para 2-tuplas, que procura além, testando inteiramente o histograma de contagens 00, 01, 10 e 11 para ver se é binomialmente distribuído com $p = 0.25$.

- **STS Serial Test:** Acumula as frequências de n-tuplas sobrepostas de bits retiradas de uma fonte de inteiros aleatórios. A distribuição esperada dos padrões de n-bits é multinomial com $p = 2^{(-n)}$, p. ex., os quatro padrões de 2 bits

  00011011

  deveria ocorrer com igual probabilidade. A distribuição alvo é assim um χ^2 com $2^n - 1$ graus de liberdade, um perdido devido a restrição que: $p_{00} + p_{01} + p_{01} + p_{11} = 1$.

- **RGB Bit Distribution Test:** Acumula as frequências de todas as n-tuplas de bits em uma lista de inteiros aleatórios e compara a distribuição assim gerada com o histograma teórico (binomial), formando χ^2 e o associado p-value. Neste teste n-tuplas são selecionadas sem sobreposição (p. ex., 01|10|10|01|11|00|01|10), de modo que as amostras são independente.

- **The Generealized Minimum Distance Test:** Este teste é baseado em um artigo de M. Fischler. Este teste utiliza termos de correlação que estão essencialmente em ordem para o teste não falhar para um grande número de tentativas. Ele substitui tanto diehard_2dsphere.c quanto diehard_3dsphere.c, e generaliza o teste per si, de modo que ele pode ser executado para qualquer $d = 2, 3, 4, 5$.

- **RGB Permutations Test:** Este é um teste sem sobreposição que simplesmente conta as permutações de ordem de números aleatórios, retirados n de cada vez. Existem $n!$ permutações e todas são igualmente prováveis. As amostras são independentes, assim alguém pode fazer um simples teste χ^2 sobre o vetor de contagem com $n! - 1$ graus de liberdade.

- **RGB Lagged Sums Test:** Na verdade um pacote de vários testes. Alguns dos testes, contudo, testam correlações defasadas - a possibilidade que o gerador de

números aleatório tem um uma correlação a nível de bits após algum número fixo de bits intervenientes.

- **The Kolmogorov-Smirnov Test Test:** Este teste gera um vetor de tsamples que se desvia uniformemente de um RNG selecionado, aplicando então um teste Anderson-Darling ou Kuiper KS a ele para diretamente testar uniformidade.

- **DAB Byte Distribution Test:** Extrai n bytes independente a partir de k palavras consecutivas, incrementando os contadores indexados em cada uma das n tabelas. (Total de 256 * n contadores.) Correntemente, n = 3, fixo em tempo de compilação. Se $n \geq 2$, então o menor e maior bytes serão usados, junto com $n-2$ bytes do meio. Se o tamanho da palavra do gerador é muito pequena, bytes sobrepostos serão usados. Use o teste básico de ajuste básico χ^2 (chisq_pearson) para o p-value. A versão anterior também usava um teste de independência χ^2 (chisq2d) que foi encontrado ser ligeiramente menos sensível.

- **DCT (Frequency Analysis) Test:** Este teste executa uma Discrete Cosine Transform (DCT) na saída do RNG. Mais especificadamente, ele executa tsamples transformações, cada uma sobre um bloco independente de n-tuplas palavras. Se tsamples é grande o suficiente, as posições do valor máximo (absoluto) em cada transformação são registrados e sujeitos ao teste χ^2 para uniformidade/independência.

- **DAB Fill Tree Test:** Este teste preenche pequenas árvores binárias de profundidade fixa com palavras do RNG. Quando uma palavra não pode ser inserida na árvore, a contagem corrente de palavras na árvore é registrada, junto com a posição na qual a palavra seria inserida. As palavras do RNG são rotacionadas (em longos ciclos) para melhor detectar RNGs que podem influenciar somente os bytes altos, médios e baixos. O teste retorna dois p-values. O primeiro é novamente o teste Pearson χ^2 contra os valores esperados (que foram estimados empiricamente). O segundo é um teste Pearson χ^2 para uma distribuição uniforme das posições no qual a inserção falhou.

- **DAB Fill Tree 2 Test:** Versão para bits do teste Fill Tree.

- **DAB Monobit 2 Test:** Teste block-monobit.

A.3 NIST STS

NIST STS (Stastistical Test Suite) (Rukhin et al., 2010) é um pacote de software cujo escopo é fornecer um conjunto de testes de estatísticos visando avaliar a qualidade de geradores de números aleatórios e pseudo-aleatórios para aplicações criptográficas.

Desenvolvido e disponibilizado pelo NIST (National Institute of Standards and Technology), é considerado como o *padrão* aceito pela indústria, sendo composto por 15 testes estatísticos, que avaliam a presença de um padrão que, se detectado, poderia indicar que a sequência não é aleatória.

Da mesma feita que as outras suítes de testes, NIST STS é acessada por meio de uma interface de linha de comandos. A partir da linha de comando, deve ser emitido o seguinte comando: `assess <stream length>`, onde *stream length* é o comprimento do fluxo de bits individuais a serem processados, por exemplo, `assess 681622656`. Após isso, deve-se escolher o tipo de gerador a ser utilizado ou o arquivo contendo o fluxo de bits a ser analisado (opção `[0]`). Se a opção `[0] Input File` é escolhida, então deve-se informar qual o nome do arquivo a ser processado. Em seguida, deve-se escolher quais testes serão executados, os quais podem ser escolhidos individualmente, em grupo ou todos (digitando `1`). Se nesta etapa foi digitado `0`, então deve-se escolher quais testes serão executados, os quais são ativados marcando-se `1`, caso não se deseje executar um teste, marca-se `0`. Passada esta etapa pode-se ajustar os parâmetros para os testes. Finalmente, deve-se informar como se apresenta o fluxo de bits a serem analisados, se estão no formato ASCII ou no formato binário. Abaixo, um exemplo de execução da suíte.

```
C:\~\Documents\Pacotes\Random\STS\sts-2.1.2\OrbitsTest>assess 681622656
          G E N E R A T O R    S E L E C T I O N
          ---------------------------------------

    [0] Input File                 [1] Linear Congruential
    [2] Quadratic Congruential I   [3] Quadratic Congruential II
    [4] Cubic Congruential         [5] XOR
    [6] Modular Exponentiation     [7] Blum-Blum-Shub
    [8] Micali-Schnorr             [9] G Using SHA-1

   Enter Choice: 0

               User Prescribed Input File: orbits

               S T A T I S T I C A L   T E S T S
               ---------------------------------

    [01] Frequency                       [02] Block Frequency
    [03] Cumulative Sums                 [04] Runs
    [05] Longest Run of Ones             [06] Rank
    [07] Discrete Fourier Transform      [08] Nonperiodic Template Matchings
    [09] Overlapping Template Matchings  [10] Universal Statistical
    [11] Approximate Entropy             [12] Random Excursions
    [13] Random Excursions Variant       [14] Serial
    [15] Linear Complexity
```

```
      INSTRUCTIONS
         Enter 0 if you DO NOT want to apply all of the
         statistical tests to each sequence and 1 if you DO.

Enter Choice: 0

      INSTRUCTIONS
         Enter a 0 or 1 to indicate whether or not the numbered statistical
         test should be applied to each sequence.

   123456789111111
            012345
   111111111111111


     P a r a m e t e r   A d j u s t m e n t s
     ----------------------------------------
 [1] Block Frequency Test - block length(M):         128
 [2] NonOverlapping Template Test - block length(m): 9
 [3] Overlapping Template Test - block length(m):    9
 [4] Approximate Entropy Test - block length(m):     10
 [5] Serial Test - block length(m):                  16
 [6] Linear Complexity Test - block length(M):       500

Select Test (0 to continue): 0

How many bitstreams? 1

Input File Format:
 [0] ASCII - A sequence of ASCII 0's and 1's
 [1] Binary - Each byte in data file contains 8 bits of data

Select input mode:  1

  Statistical Testing In Progress.........
```

Os testes executados pela suíte NIST STS e seus objetivos são:

- **Frequency (Monobits) Test:** O foco do teste é a proporção de zeros e uns na sequência inteira. O propósito deste teste é determinar se aquele número de zeros e uns na sequência são aproximadamente iguais como deveria ser esperado para uma sequência verdadeiramente aleatória. O teste avalia a proximidade da fração de uns com $\frac{1}{2}$, isto é, o número de uns e zeros na sequência deveria ser aproximadamente o mesmo.

- **Test For Frequency Within A Block:** O foco do teste é a proporção de zeros e uns dentro de blocos com M-bits. O propósito deste teste é determinar se a frequência de uns em um bloco com M-bits é aproximadamente $\frac{M}{2}$.

- **Runs Test:** O foco deste teste é o número total de runs de zeros e uns na sequência inteira, onde um run é uma sequência ininterrupta de bits idênticos. Um run de comprimento k significa que este run consiste de exatamente k bits idênticos e é limitado antes e após com um bit de valor oposto. O propósito do teste runs é determinar ser o número de runs de uns e zeros de vários comprimentos é como esperado para uma sequência aleatória. Em particular, este teste determina se a oscilação entre tais substrings é muito rápida ou muito devagar.

- **Test For The Longest Run Of Ones In A Block:** O foco deste teste é o mais longo dos runs de uns dentro de blocos de M-bits. O propósito deste teste é determinar se o comprimento do maior run de uns dentro da sequência testada é consistente com o comprimento do mais longo run de uns que seria esperado em uma sequência aleatória. Note que uma irregularidade no comprimento esperado do mais longo run de uns inplica que existe também uma irregularidade no comprimento esperado do mais longo run para zeros. Runs para zeros não foram avaliados separadamente devido a preocupação sobre independência entre os testes.

- **Random Binary Matrix Rank Test:** O foco do teste é o rank de submatrizes disjuntas da sequência inteira. O propósito deste teste é verificar dependência linear entre substrings de comprimento fixo da sequência original.

- **Discrete Fourier Transform (Spectral) Test:** O foco deste teste é a altura dos picos na FFT (Fast Fourier Transform), com o objetivo de detectar características periódicas (i. é, padrões repetitivos que estão próximos um do outro) na sequência testada, o que poderia indicar um desvio da assunção de aleatoriedade.

- **Non-Overlapping (Aperiodic) Template Matching Test:** O foco do teste é o número de ocorrências de substrings alvo pré-definidas. O propósito deste teste é rejeitar sequências que exibem muitas ocorrências de um dado padrão não periódico (aperiódico). Para este teste e para o teste Overlapping Template Matching, uma janela com m-bits é usada para procurar por um específico padrão com m-bits. Se o padrão não é encontrado, a janela é deslocada em 1 bit. Para este teste, quando o padrão é encontrado, a janela é redefinida para o bit após o padrão encontrado, e a procura é retomada.

- **Overlapping (Periodic) Template Matching Test:** O foco deste teste é o número de substrings alvo pré-definidas. O propósito deste teste é rejeitar sequências que mostram desvios do número esperado de runs de uns de um dado comprimento. Note que quanto existe um desvio do número esperado de uns de um dado comprimento, existe também um desvio nos runs de zeros. Runs de zeros

não foram avaliados separadamente devido a preocupação sobre independência estatística entre os testes. Para este teste e para o teste Non-overlapping Template Matching, uma janela com m-bits é usada para procurar por um específico padrão com m-bits. Se o padrão não é encontrado, a janela é deslocada 1 bit. Para este teste, quando o padrão é encontrado, a janela novamente é deslocada 1 bit, e a procura é retomada.

- **Maurer's Universal Statistical Test:** O foco deste teste é o número de bits entre casamento de padrões. O propósito deste teste é detectar se, ou não, a sequência pode ser significantemente comprimida sem perda de informação. Uma sequência excessivamente comprimível não é considerada aleatória.

- **Linear Complexity Test:** O foco deste teste é o comprimento de um registrador de feedback gerador. O propósito deste teste é determinar se, ou não, a sequência é complexa o suficiente para ser considerada aleatória. Sequências aleatórias são caracterizadas por um longo registrador de feedback. Um registrador de feedback curto implica em não aleatoriedade.

- **Serial Test:** O foco deste teste é a frequência de cada e todo padrão sobreposto de m-bits ao longo da sequência inteira. O propósito deste teste é determinar se o número de ocorrências de padrões sobrepostos $2m$ de m-bits é aproximadamente o mesmo que seria esperado para uma sequência aleatória. O padrão pode se sobrepor.

- **Approximate Entropy Test:** O foco deste teste é a frequência de cada e todo padrão sobreposto de m-bits. O propósito do teste é comparar a frequência de blocos sobrepostos de comprimentos consecutivos/adjacentes (m e $m+1$) contra o resultado esperado para uma sequência aleatória.

- **Cumulative Sum (Cusum) Test:** O foco deste teste é a excursão maximal (a partir de zero) de uma caminhada aleatória pela soma cumulativa de dígitos ajustados (-1, +1) na sequência. O propósito do teste é determinar se a soma cumulativa das sequências parciais ocorrendo na sequência testada é muito grande ou muito pequena relativo ao comportamento esperado da soma cumulativa para sequências aleatórias. Esta soma cumulativa pode ser considerada como uma caminhada aleatória. Para uma sequência aleatória, uma caminhada aleatória deveria ser próxima de zero. Para sequências não aleatórias, as excursões desta caminhada aleatória a partir de zero serão bem grandes.

- **Random Excursions Test:** O foco deste teste é o número de ciclos tendo exatamente K visitas em uma soma cumulativa de uma caminhada aleatória. A soma cumulativa da caminhada aleatória é encontrada se somas parciais da sequência

(0, 1) são ajustadas para (-1, +1). Uma excursão aleatória de uma caminhada aleatória consiste em uma sequência de n passos de comprimento tomadas aleatoriamente que começa e retorna para a origem. O propósito deste teste é determinar se o número de visitas a um estado dentro de uma caminhada aleatória excede aquele que alguém espera para uma sequência aleatória.

- **Random Excursions Variant Test:** O foco deste teste é o número de vezes que um estado particular ocorre em uma soma cumulativa de uma caminhada aleatória. O propósito deste teste é detectar desvios a partir do número de ocorrências dos vários estados em uma caminhada aleatória.

Referências Bibliográficas

Alia, M. A. e Samsudin, A. B. (2007). Generalized scheme for fractal based digital signature (GFDS). *International Journal of Computer Science and Network Security (IJCSNS)*, 7(7):99–104.

Alligood, K. T., Sauer, T. e Yorke, J. (1996). *Chaos - An Introduction to Dynamical Systems*. Textbooks in Mathematical Sciences. Springer-Verlag New York.

Alvarez, E., Fernández, A., García, P., Jiménez, J. e Marcano, A. (1999). New approach to chaotic encryption. *Physics Letters A*, 263(4-6):373–375.

Alvarez, G. (2006). Some basic cryptographic requeriments for chaos-based cryptosystems. *International Journal of Bifurcation and Chaos*, 16(8):2129–2151.

Banks, J., Brooks, J., Cairns, G., Davis, G. e Stacey, P. (1992). On devaney's definition of chaos. *The American Mathematical Monthly*, 99(4):332–334.

Baptista, M. (1998). Cryptography with chaos. *Physics Letters A*, 240(1):50–54.

Barnsley, M. F. (2006). *Superfractals*. Cambridge University Press.

Barnsley, M. F. e Demko, S. (1985). Iterated function systems and the global construction of fractals. *Proceedings Mathematical Physical & Engineering Sciences*, 399(1817):243–275.

Bellovin, S. M. (2011). Frank miller: Inventor of the one-time pad. *Cryptologia*, 35(3):203–222.

Bendixson, I. (1901). Sur les courbes définies par des équations différentielles. *Acta Math.*, 24:1–88.

Bennett, C. H., Brassard, G. e Breidbart, S. (2014). Quantum cryptography ii: How to re-use a one-time pad safely even if p=np. *Natural Computing*, 13(4):453–458.

Blum, L., Blum, M. e Shub, M. (1986). A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383.

Boeing, G. (2016). Visual analysis of nonlinear dynamical systems: Chaos, fractals, self-similarity and the limits of prediction. *Systems*, 4(37).

Bright, H. S. e Enison, R. L. (1979). Quasi-random number sequences from a long-period tlp generator with remarks on application to cryptography. *ACM Computing Surveys (CSUR)*, 11(4):357–370.

Brown, R. e Chua, L. O. (1996). Clarifying chaos: Examples and counterexamples. *International Journal of Bifurcation and Chaos*, 6(2):219–249.

Bucci, M. e Luzzi, R. (2016). A fully-digital chaos-based random bit generator. Em Ryan, P. Y., Naccache, D. e Quisquater, J.-J., editores, *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, volume 9100 de *Lecture Notes in Computer Science*, páginas 396–414. Springer.

Cantor, G. (1883). Über unendliche, lineare punktmannigfaltigkeiten. *V. Math. Ann.*, páginas 545–591.

Chaitin, G. J. (1966). On the length of programs for computing finite binary sequences. *Journal of ACM*, 11(4):547–569.

Chaitin, G. J. (1975). Randomness and mathematical proof. *Scientific American*, 232(5):47–52.

Chua, L. O. (1992). The genesis of chua's circuit. *Archiv für Elektronik und Ubertragungstechnik*, 46(4):250–257.

Church, A. (1940). On the concept of a random sequence. *Bull. Amer. Math. Soc.*, 46(2):130–135.

Cocks, C. (1973). A note on 'non-secret encryption'. Publicado intenamento no Government Communications Headquarters (GCHQ).

Crounse, K. R., Yang, T. e Chua, L. O. (1996). Pseudo-random sequence generation using the cnn universal machine with applications to cryptography. Em *1996 Fourth IEEE International Workshop on Cellular Neural Networks and their Applications Proceedings (CNNA-96)*, páginas 433–438, Seville, Spain. IEEE.

Cuomo, K. M., Oppenheim, A. V. e Strogatz, S. H. (1993). Synchronization of lorenz-b ased chaotic circuits with applications to communications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 40(10):626 – 633.

Cvitanović, P., Artuso, R., Mainieri, R., Tanner, G. e Vattay, G., editores (2013). *Chaos: Classical and Quantum*. ChaosBook.org, Niels Bohr Institute, Copenhagen.

Danforth, C. M. (2013). Chaos in an atmosphere hanging on a wall. *http://mpe2013.org/2013/03/17/chaos-in-an-atmosphere-hanging-on-a-wall/*. Acessado em: 15/01/2018.

Delfs, H. e Knebl, H. (2007). *Introduction to Cryptography - Principles and Applications*. Information Security and Cryptography - Texts and Monographs. Springer, 2nd edition.

Devaney, R. L. (2003). *An Introduction to Chaotic Dynamical Systems*. Westview Press, 2nd edition.

Diffie, W. e Hellman, M. E. (1976). New directions on cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

ElGamal, T. (1985a). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472.

ElGamal, T. (1985b). A public key cryptosystem and a signature scheme based on discrete logarithms. Em Blakley, G. R. e Chaum, D., editores, *Advances in Cryptology: Proceedings of CRYPT0 84*, volume 196 de *Lecture Notes in Computer Science*, páginas 10–17. Springer-Verlag.

Ellis, J. H. (1970). The possibility of non-secret digital encryption. CESG Research Report 3006, Government Communications Headquarters (GCHQ).

Feigenbaum, M. J. (1978). Quantitative universality for a class of nonlinear transformations. *Journal of Statistical Physics*, 19(1):25–52.

Fridrich, J. (1998). Symmetric ciphers based on two-dimensional chaotic maps. *International Journal of Bifurcation and Chaos*, 08(06):1259–1284.

Fu, C., chuan Zhang, Z., Chen, Y. e wei Wang, X. (2007). An improved chaos-based image encryption scheme. Em Shi, Y., van Albada, G. D., Dongarra, J. e Sloot, P. M. A., editores, *Computational Science – ICCS 2007: 7th International Conference, Beijing, China, May 27 - 30, 2007, Proceedings, Part I*, Lecture Notes in Computer Science 4487, páginas 575–582, Berlin, Heidelberg. Springer.

Glansdorff, P. e Prigogine, I. (1971). *Thermodynamic Theory of Structure, Stability and Fluctuations*. Wiley-Interscience.

Gleick, J. (1987). *Chaos: Making a new science*. Viking Penguin.

Grebogi, C., Ott, E., Pelikan, S. e Yorke, J. A. (1984). Strange attractors thar are not chaotic. *Phisica D*, 13:261–268.

Guyeux, C., Wang, Q. e Bahi, J. (2010). A pseudo random numbers generator based on chaotic iterations. application to watermarking. Em *Web Information Systems and Mining, WISM 2010*, número 6318 em Lecture Notes in Computer Science, páginas 202–211. Springer-Verlag.

Habutsu, T., Nishio, Y., lwao Sasase e Mori, S. (1991). A secret key cryptosystem by iterating a chaotic map. Em Davies, D. W., editor, *Advances in Cryptology - EUROCRYPT '91*, Lecture Notes in Computer Science 547, páginas 127–134. Springer-Verlag.

Hadamard, J. (1898). Les surfaces à courbures opposées et leurs lignes géodésiques. *Journal de mathématiques pures et appliquées 5^e série*, tome 4:27–74.

Hastings, A., Hom, C. L., Ellner, S., Turchin, P. e Godfray, H. C. (1993). Chaos in ecology: Is mother nature a strange attractor. *Annual Review of Ecology and Systematics*, 24:1–33.

Hénon, M. (1976). A two-dimensional mapping with a strange attractor. *Communications in Mathematical Physics*, 50(1):69–77.

Hirsch, M. W., Smale, S. e Devaney, R. L. (2004). *Differential Equations, Dynamical Systems, and An Introduction to Chaos*. Elsivier, 2nd edition.

Hopf, E. (1934). On causality, statistics and probability. *Studies in Applied Mathematics*, 13(1-4):51–102.

Hu, H., LingFengLiu e Ding, N. (2013). Pseudorandom sequence generator based on the chen chaotic system. *Computer Physics Communications*, 184(3):765–768.

Ikeda, K. (1979). Multiple-valued stationary state and its instability of the transmitted light by a ring cavity system. *Optics Communications*, 30(2):257–261.

Ikeda, K., Daido, H. e Akimoto, O. (1980). Optical turbulence: Chaotic behavior of transmitted light from a ring cavity. *Phys. Rev. Lett.*, 45(9):709–712.

Jakimoski, G. e Kocarev, L. (2001). Analysis of some recently proposed chaos-based encryption algorithms. *Physics Letters A*, 291(6):381–384.

Jakobson, M. V. (1981). Absolutely continuous invariant measures for one-parameter families of one-dimensional maps. *Commun. Math. Phys.*, 81(1):38–88.

jun Huang, F. e qian Zhao, Y. (2013). New key-stream generation scheme based on hénon chaotic system. *Journal of Central South University*, 20(7):1904–1908.

Kaijser, T. (1981). On a new contraction condition for random systems with complete connections. *Rev. Roumaine Math. Pures Appl.*, 26(8):1075–1117.

Kerckhoffs, A. (1883). La cryptographie militaire. *Journal des sciences militaires*, IX:5–38.

Knuth, D. E. (1998). *The Art of Computer Programming - Vol. 2, Seminumerical Algorithms*. Addison Wesley, 3rd edition.

Kocarev, L. (2001). Chaos-based cryptography: A brief overview. *IEEE Circuits and Systems Magazine*, 1(3):6–21.

Kocarev, L. e Tasev, Z. (2003). Public-key encryption based on chebyshev maps. Em *2003 IEEE International Symposium on Circuits and Systems, ISCAS 2003*, páginas III–28–III–31.

Kolmogorov, A. N. (1965). Three approaches to the definition of the concept “quantity of information". *Probl. Peredachi Inf.*, 1(1):3–11.

Kumar, S. (2006). Public-key cryptographic system using mandelbrot sets. Em *MILCOM 2006 - 2006 IEEE Military Communications conference*, páginas 1–5.

Lardjane, S. (2006). On some stochastic properties in devaney's chaos. *Chaos, Solitons & Fractals*, 28:668–672.

L'Ecuyer, P. (1988). Efficient and portable combined random number generators. *Communications of the ACM*, 31(6):742–751.

Li, T. Y. e Yorke, J. A. (1978). Ergodic transformations from an interval into itself. *Trans. Amer. Math. Soc.*, 235:183–192.

Lian, S. (2009). *Multimedia Content Encryption - Techniques and Applications*. CRC Press.

Liapounoff, M. A. (1907). Problème général de la stabilité du mouvement. *Annales de la Faculté des sciences de Toulouse: Mathématiques*, 9:203–474.

Löf, P. M. (1966). The definition of random sequences. *Information and Control*, 9(6):602–619.

Löf, P. M. (1969). The literature of von mises kollektivs revisited. *Theoria*, 35(1):12–37.

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141.

Lorenz, E. N. (1993). *The Essence of Chaos*. University of Washington Press.

Luizi, P. C. S., Cruz, F. R. B. e van de Graaf, J. (2010). Assessing the quality of pseudo-random number generators. *Computational Economics*, 36(1):57–67.

Lyapunov, A. M. (1992). *The General Problem of the Stability of Motion*. Taylor & Francis.

Mandelbrot, B. B. (1983). *The Fractal Geometry of Nature*. W. H. Freeman and Company, San Francisco.

Matsumoto, M. e Nishimura, T. (1998a). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30.

Matsumoto, M. e Nishimura, T. (1998b). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30.

Matsumoto, T. (1984). A chaotic attractor from chua's circuit. *IEEE Transactions on Circuits and Systems*, 31(12):1055–1058.

Matsumoto, T., Chua, L. O. e Komuro, M. (1985). The double scroll. *IEEE Transactions on Circuits and Systems*, 32(8):797–818.

Matthews, R. (1989). On the derivation of a "chaotic" encryption algorithm. *Criptologia*, 13(1):29–42.

May, R. M. (1976). Simple mathematical models with very complicated dynamics. *Nature*, 261:459–467.

Meier, W. e Stdelbach, O. (1991). Analysis of pseudo random sequences generated by cellular automata. Em Davies, D. W., editor, *Advances in Cryptology - EUROCRYPT '91*, volume 547 de *Lecture Notes in Computer Science*, páginas 186–199. Springer-Verlag.

Menezes, A. J., van Oorschot, P. C. e Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications. CRC Press.

Merkle, R. C. (1978). Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299.

Miller, F. (1882). *Telegraphic code to Insure Privacy and Secrecy in the Transmission of Telegrams*. Charles M. Cornwell.

Mises, R. V. (1964). Grundlagen der wahrscheinlichkeitsrechnung. Em Frank, P., Goldstein, S., Kac, M., Prager, W., Szego, G. e Birkhoff, G., editores, *Selected Papers of*

Richard von Mises, volume 2 - Probability and Statistics, General, capítulo: 4, páginas 57–108. American Mathematical Society.

Mishkovski, I. e Kocarev, L. (2011). Chaos-based public-key cryptography. Em Kocarev, L. e Lian, S., editores, *Chaos-Based Cryptography*, volume 354 de *Studies in Computational Intelligence*, capítulo: 2, páginas 27–65. Springer.

Mollin, R. A. (2007). *An Introduction to Cryptography*. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, 2nd edition.

Mooney, A., Keating, J. G. e Pitas, I. (2008). A comparative study of chaotic and white noise signals in digital watermarking. *Chaos, Solitons and Fractals*, 35(5):913–921.

Nicolis, G. (1971). *Stability and Dissipative Structures in Open Systems far from Equilibrium*, volume 19 de *Advances in Chemical Physics*. John Wiley & Sons, Hoboken, NJ.

Niu, X., Wang, Y. e Wu, D. (2014). A method to generate random number for cryptographic application. Em *2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, páginas 235–238, Kitakyushu, Japan. IEEE.

Nolte, D. D. (2010). The tangled tale of phase space. *Physics Today*, 63(4):33–38.

Örjan Stenflo (2002). Uniqueness of invariantmeasures for place-dependent random iterations of functions. Em Barnsley, M. F., Saupe, D. e Vrscay, E. R., editores, *Fractals in Multimedia*, The IMA Volumes in Mathematics and its Application 132, páginas 13–32. Springer-Verlag.

P., J. R. e S., D. B. (2011). Symmetric encryption using sierpinski fractal geometry. Em Venugopal, K. R. e Patnaik, L. M., editores, *Computer Networks and Intelligent Computing: 5th International Conference on Information Processing, ICIP 2011, Bangalore, India, August 5-7, 2011. Proceedings*, páginas 240–245. Springer Berlin Heidelberg.

Palis, J. (1999). Sistemas caóticos e sistemas complexos. Em *Complexidade & Caos*, capítulo: I, páginas 27–38. Editora URFJ/COPEA.

Pareschi, F. (2006). *Chaos-based Random Number Generators: Monolithic Implementation, Testing and Applications*. Tese de doutorado, Università di Bologna.

Parlitz, U., Chua, L. O., Kocarev, L., Halle, K. S. e Shang, A. (1992). Transmission of digital signals by chaotic synchronization. *International Journal of Bifurcation and Chaos*, 2(4):973–977.

Pecora, L. M. e Carroll, T. L. (1990). Synchronization in chaotic systems. *Phys. Rev. Lett.*, 64(8):821–824.

Poincaré, H. (1881). Mémoire sur les courbes définies par une équation différentielle. *Journal de mathématiques pures et appliquées*, 7:375–422.

Poincaré, H. (1890). Sur le problème des trois corps et les équations de la dynamique. *Acta mathematica*, 13:1–270. Oeuvres, tome VII, pp. 262-479.

Prigogine, I. (2000). *As leis do caos*. Editora UNESP.

Rani, P. J., Rao, M. S. e Bhavani, S. D. (2011). Design of secure chaotic hash function based on logistic and tent maps. Em C.Wyld, D., Wozniak, M., Chaki, N., Meghanathan, N. e Nagamalai, D., editores, *Advances in Network Security and Applications - 4th International Conference, CNSA 2011*, Communications in Computer and Information Science 196, páginas 43–52.

Raza, S. F. e Satpute, V. R. (2018). Practo: Pseudo random bit generator for cryptographic application. *KSII Transactions on Internet and Information Systems*, 12(12):6161–6176.

Rivest, R. L., Shamir, A. e Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Röck, A. (2005). Pseudorandom number generators for cryptographic applications. Dissertação de Mestrado, Univercity of Salzburg.

Röck, A. (2009). *Quantifying Studies of (Pseudo) Random Number Generation for Cryptography*. Tese de doutorado, L'école Polytechnique.

Rössler, O. E. (1976). An equation for continuous chaos. *Physics Letters A*, 57(5):397–398.

Rubin, F. (1996). One-time pad cryptography. *Cryptologia*, 20(4):359–364.

Ruelle, D. (1993). *Acaso e Caos*. Editora UNESP.

Ruelle, D. (2006). What is ... a strange attractor? *Notices of the AMS*, 53(7):764–765.

Ruelle, D. e Takens, F. (1971). On the nature of turbulence. *Commun. Math. Phys.*, 20(3):167–192.

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J. e Vo, S. (2010). A statistical test suite

for random and pseudorandom number generators for cryptographic applications. Relatório Técnico NIST Special Publication 800-22, Revision 1a, National Institute of Standards and Technology (NIST).

Sathishkumar, G. A., bagan, D. K. B. e Sriraam, D. N. (2011). Image encryption based on diffusion and multiple chaotic maps. *International Journal of Network Security & Its Applications (IJNSA)*, 3(2):181–194.

Schuster, P. e Sigmund, K. (1983). Replicator dynamics. *Journal of Theoretical Biology*, 100(3):533–538.

Shannon, C. E. (1949). Communication theory of secrecy systems*. *Bell System Technical Journal*, 28(4):656–715.

Shujuna, L., Xuanqinb, M. e Yuanlongc, C. (2001). Pseudo-random bit generator based on couple chaotic systems and its applications in stream-cipher cryptography. Em *Progress in Cryptology - INDOCRYPT 2001*, número 2247 em Lecture Notes in Computer Science. Springer-Verlag.

Singh, S. (2001). *The Code Book - How to make it, break it, hack it, crack it.* Delacorte Press.

Smith, L. A. (2007). *Chaos - A Very Short Introduction.* Very Short Introductions. Oxford University Press.

Smith, P. (1998). *Explaining Chaos.* Cambridge University Press.

Solomonoff, R. J. (1964). A formal theory of inductive inference. part i. *Information and Control*, 7(1):1–22.

Soto, J. (1999). Statistical testing of random number generators. Em *Proceedings of the 22nd National Information Systems Security Conference*, Arlington, Virginia, United States.

Stewart, I. (1997). *Does God Play Dice?: The New Mathematics of Chaos.* Penguin Books, 2nd edition.

Stewart, I. (2000). The lorenz attractor exists. *Nature*, 406(6799):948–949.

Stinson, D. (2001). *Cryptographie - Théorie et pratique.* Vuibert Informatique. Tradudor: Serge Vaudenay.

Taylor, P. D. e Jonker, L. B. (1978). Evolutionarily stable strategies and game dynamics. *Mathematical Biosciences*, 40(1-2):145–156.

Tefas, A., Nikolaidis, A., Nikolaidis, N., Solachidis, V., Tsekeridou, S. e Pitas, I. (2003). Markov chaotic sequences for correlation based watermarking schemes. *Chaos, Solitons and Fractals*, 17(2-3):567–573.

Tenny, R. e Tsimring, L. S. (2005). Additive mixing modulation for public key encryption based on distributed dynamics. *IEEE Transactions on Circuits and Systems - I: Regular Papers*, 52(3):672–679.

Tenny, R., Tsimring, L. S., Larson, L. e Abarbanel, H. D. I. (2003). Using distributed nonlinear dynamics for public key encryption. *Physical Review Letters*, 90(4).

Tomassini, M., Zolla, M. S. M. e Perrenoud, M. (1999). Generating high-quality random numbers in parallel by cellular automata. *Future Generation Computer Systems*, 16(2-3):291–305.

Tucker, W. (1999). The lorenz attractor exists. *C. R. Acad. Sci. Paris*, 328 - Série 1:1197–1202.

Tucker, W. (2009). Fundamentals of chaos. Em Kocarev, L., Galias, Z. e Lian, S., editores, *Intelligent Computing Based on Chaos*, volume 184 de *Studies in Computational Intelligence*, páginas 1–23. Springer.

Verhulst, P. F. (1838). Notice sur la loi population dans son accroissement. Em Quételet, A., editor, *Correspondance Mathématique et Physique*, volume X, páginas 113–121. Société Belge de Librairie.

Vernam, G. S. (1919). Us patent #1,310,719: Secret signaling system, july 1919.

Viana, M. (2000). What's new on lorenz strange attractors? *The Mathematical Intelligencer*, 22(3):6–19.

Ville, J. (1939). *Étude critique de la notion de collectif*. Tese de doutorado, Faculté des Sciences de Paris. Editor: Gauthier-Villars.

von Mises, R. (1919). Grundlagen der Wahrscheinlichkeitsrechnung. *Math Zeitschrift*, 5(1-2):52–99.

von Neumann, J. (1951). Various techniques used in connection with random digits. Em Householder, A., Forsythe, G. e Germond, H., editores, *Monte Carlo Method*, páginas 36–38. National Bureau of Standards Applied Mathematics Series, Washington, D.C.: U.S. Government Printing Office.

Wald, A. (1936). Sur la notion de collectif dans le calcul des probabilités. Em *Comptes Rendus des Séances de L'Académie des Sciences*, número 202, páginas 180–183.

Wolfram, S. (1986). Random sequence generation by cellular automata. *Advances in Applied Mathematics*, 7(2):123–169.

Wu, Q. (2015). A chaos-based hash function. Em *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery - CyberC 2015*, páginas 1–4. IEEE.

Yi, X. (2005). Hash function based on chaotic tent maps. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 52(6):354–357.

Zuchowski, L. C. (2017). *A Philosophical Analysis of Chaos Theory*. New Directions in the Philosophy of Science. Palgrave Macmillan.