MAP2110 - Projeto 6 - Caos

Professor: Dr. Nelson Mugayar Kuhl

IME-USP 2021

Sumário

1 Introdução

A palavra caos geralmente remete a ideia de desordem, porém, na matemática, a teoria do caos descreve um sistema caótico como algo complexo, dinâmico, "deterministico"e sensível a condição inicial, sendo este penultimo fator, um tanto contradizente com as primeiras impressões da palavra caos.

Em primeira instância, um sistema caótico é deterministico e previsível, porém, como veremos no decorrer deste trabalho, a longo prazo, um sistema caótico torna-se imprevisível pois este é sensível a condição inicial.

De forma breve, a palavra caos descreve transformações na Matemática e certos fenômenos físicos que aparenta ter um comportamento totalmente aleatório mas que na verdade apresenta uma ordem bem determinado. Alguns exemplos citados no livro do Anton [1] são:

- A geração de números aleatórios.
- O ato de embaralhar cartas de um baralho.
- A arritimia cardíaca.
- A mudança das manchas na superficie de Jupiter.

Para fins desse trabalho, pretende-se estudar uma transformação caótica específica denominada **Transformação do gato de Arnold** para responder as tarefas especificadas.

2 Teoria

2.1 Transformação do gato de Arnold

2.1.1 Um pouco de aritmética modular

Suponha $a, b \in \mathbb{Z}$. Dizemos que a é congruente a b módulo m (com $m > 0$) se $m|(a - b)$. Denotamos isto por $a \equiv b (mod m)$.

Exemplo: $11 \equiv 3 \ (mod\ 2)$ pois $2|(11 - 3)$.

Se $x \in \mathbb{R}$,a notação $x \ mod\ 1$ denota o único número no intervalo $[0, 1)$ que difere de x por um número inteiro.

Exemplo: $2.3 \ mod\ 1 = 0.3$, $0.9 \ mod\ 1 = 0.9$, $-3.7 \ mod\ 1 = 0.3$, $2,0 \ mod\ 1 = 0$.

Ou seja, uma regra geral é:

1. Dê um valor para x.
2. Se o valor de x for positivo, pegue somente a parte fracionária.
3. Caso contrário, (se o valor for negativo), Peque $1-$ a parte fracionária. (Isto é, do terceiro exemplo acima, $-3.7 \ mod\ 1 = 1 - 0.7 = 0.3$)

Se (x, y) é um sistema de coordenadas de números reais, a notação $(x, y) \ mod\ 1$ retorna um par ordenado com valores de x e y no intervalo $[0, 1)$, ou seja, $(x, y) = (x \ mod\ 1; y \ mod\ 1)$.

Então podemos montar o seguinte conjunto S de par de pontos $(x, y) \ mod\ 1$:

$$S = \{(x, y) | 0 \leq x < 1, 0 \leq y < 1\}$$

Note que ao não incluir o valor 1 tanto na abscissa como na ordenada, as arestas do quadrado não estão incluídas em S.

2.1.2 A transformação do gato de Arnold

É a aplicação $\Gamma : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ definida por:

$$\Gamma : (x, y) \longrightarrow (x + y, x + 2y) \ mod\ 1$$

e a sua notação matricial é dado por:

$$\Gamma \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} mod\ 1$$

E é mais conveniente escrever a expressão acima como:

$$\Gamma \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} mod\ 1$$

"A matriz fatorada na forma acima permite entender que a transformação do gato de Arnold é uma composição de um cisalhamento da direção x de fator 1 seguido de um cisalhamento na direção y de fator 1 e como as contas são efetuados sob *mod* 1, a aplicação Γ transforma cada ponto de R^2 em um ponto do quadrado unitário *S*.

Em princípio, independente de quando aplicamos *mod* 1, (aplicar depois de cada cisalhamento ou depois de todas as contas) não faz diferença nenhuma. Mas considere ambos os métodos, primeiramente, considere o método que aplica a conta *mod* 1 somente no final.

1. Cisalhamento na direção x de fator 1.

$$(x, y) \rightarrow (x + y, y)$$

2. Cisalhamento na direção y de fator 1.

$$(x, y) \rightarrow (x, x + y)$$

3. Aplicar *mod* 1 (ou reagrupar no quadrado S).

$$(x, y) \rightarrow (x, y) mod\ 1$$

Veja abaixo o efeito da transformação do gato de Arnold no quadrado unitário *S*.

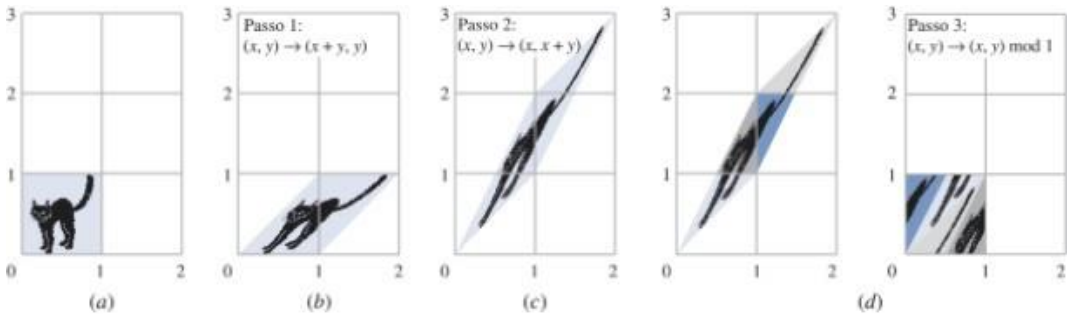


Figura 1: Processo quando aplicamos *mod* 1 após todos os cisalhamentos

Para aplicações computacionais, é mais conveniente aplicar *mod* 1 a cada cisalhamento. Desse modo, obtemos um reagrupamento a cada cisalhamento mesmo que o resultado final seja o mesmo.

1. Cisalhamento na direção x de fator 1 seguido da aplicação *mod* 1.

$$(x, y) \rightarrow (x + y, y) mod\ 1$$

2. Cisalhamento na direção y de fator 1 seguido da aplicação *mod* 1.

$$(x, y) \rightarrow (x, x + y) mod\ 1$$

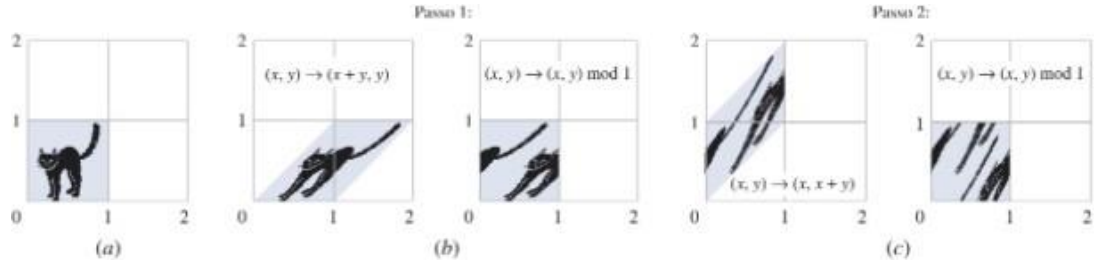


Figura 2: Processo quando aplicamos *mod* 1 a cada cisalhamento

Ao aplicar esse processo de forma repetida, ocorrem dois fenômenos interessantes:

- A imagem do gato retorna à original ao se fazer 25 iterações.
- Em algumas das iterações intermediárias, o gato está decomposto em faixas que parecem ter uma direção específica.

2.1.3 Pontos Periódicos

O nosso objetivo nesse tópico é explicar o por que o gato de Arnold retorna a figura original ao operar 25 vezes. Vale lembrar que computacionalmente as imagens são formadas de pixels, unidades estas que convém justapor sobre o eixo cartesiano xy para melhores explicações.

Se a imagem for dividido em 101 pixels por um lado, teremos um total de $101 \times 101 = 10.201$pixels e cada um, será preta ou branca.

Justapondo esses pixels sobre um plano cartesiano xy, podemos ver que cada pixel receberá um respectivo par de coordenadas da forma $(\frac{m}{101}, \frac{n}{101})$ com $m, n \in 0, 1, \ldots, 100$.

Para um caso geral, considere que p pixels formam o lado de uma imagem. Então, a imagem será composta por p^2 pixels espaçados a cada $\frac{1}{p}$ unidades nas direções x e y. Os pontos na região S vão ter coordenadas na forma $(\frac{m}{p}, \frac{n}{p})$ em que $m, n \in 0, 1, \ldots, p - 1$.

Na transformação do gato de Arnold, cada ponto de pixel S é transformado em um outro pixel de S. Veja a operação Γ aplicada ao vetor $(\frac{m}{p}, \frac{n}{p})$:

$$\Gamma \begin{pmatrix} \frac{m}{p} \\ \frac{n}{p} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} \frac{m}{p} \\ \frac{n}{p} \end{pmatrix} \mod 1 = \begin{pmatrix} \frac{m+n}{p} \\ \frac{m+2n}{p} \end{pmatrix} \mod 1$$

Como a transformação do gato de Arnold leva $S \to S$, ou seja, leva um pixel de S para outro pixel de S, e como existem p^2 pontos de pixel, segue que um ponto qualquer de S deve retornar a sua posição original depois de no máximo p^2 iterações.

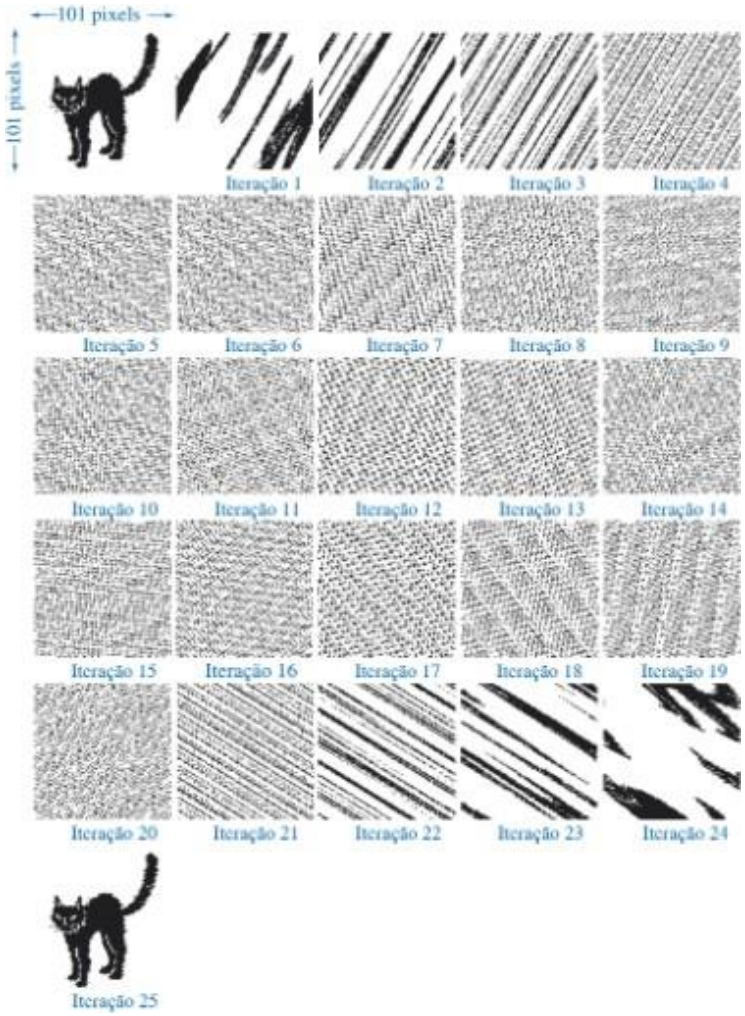


Figura 3: Processo aplicado à imagem do Gato. Note que ao se fazer 25 iterações, a posição do gato retorna ao original.

Quando um ponto retorna a sua posição original em n iterações, dizemos que o ponto têm periodo n. e que o conjunto de n iterações é denominado ciclo de periodo n.

No tópico de **Resultados numéricos**, é mostrado que o ponto (0,0) vai para (0,0) e este é o único ponto fixo da transformação.

Seja P_1 e P_2 dois pontos distintos de S com períodos q_1 e q_2 respectivamente.

Então, o ponto P_1 retorna para sua posição inicial em q_1 e o ponto P_2 retorna em q_2 e desse modo, os pontos P_1 e P_2 retornam a sua posição original para qualquer multiplo de q_1 e q_2 respectivamente de modo independente dos outros pontos.

Assim, desejamos encontrar o menor número inteiro que for multiplo comum de todos os períodos de todos os pontos (ou seja, o menor número de iterações que são necessárias para que a imagem retorne para a posição original) pois necessitamos que todos os pontos da imagem estejam na posição correta.Denotaremos a função que acha isto por $\Pi(p)$.

No tópico de **Resultados numéricos**, é mostrado que para $p = 101$, todos os pontos de pixel têm periodos 1, 5 ou 25 de modo que $\Pi(101) = 25$.

O mais curioso é que nas iterações intermediárias ao período, várias coisas inesperadas ocorrem como pode ser visto na figura abaixo:

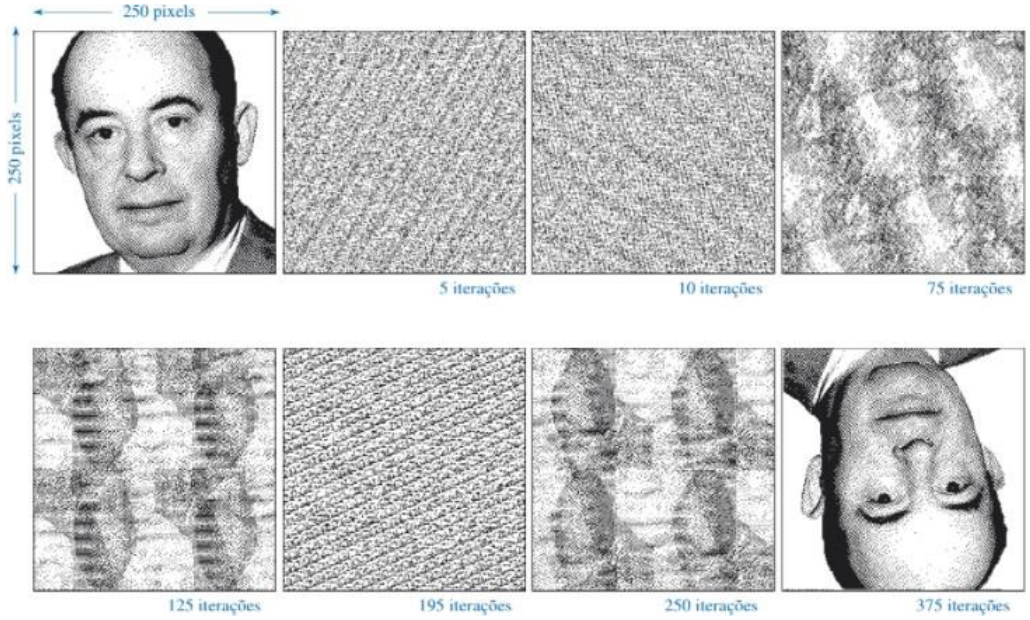


Figura 4: Processos aplicados ao foto do matemático John Von Neumann

2.1.4 O plano ladrilhado

Como visto na imagem 3, durante as iterações, aparecem algumas faixas retas. Para explicar esse fenômeno, convém ver a trasnformação do gato de Arnold de outra forma que não seja necessário o uso de aritmética modular.

Imagine que o quadrado unitário S seja um único ladrilho que têm a imagem de um gato impresso nele. Dizemos que o plano foi ladrilhado quando compusemos estes ladrilhos um do lado do outro. Se aplicarmos uma transformação matricial ao plano inteiro ladrilhado sem efetuar $mod\ 1$, pode ser visto que estaremos fazendo um processo semelhante ao quando utilizamos a aritmética $mod\ 1$ porém a nova forma é uma transformação linear.

Os dois métodos apresentam diferença na periodicidade. Se um ponto têm um periodo n, na aplicação de $mod\ 1$ este ponto retornaria a sua posição original após n iterações. No caso de ladrilhamento, os pontos não precisam voltar a sua posição original bastando que os pontos sejam substituídos por pontos da mesma cor ao final de n iterações.

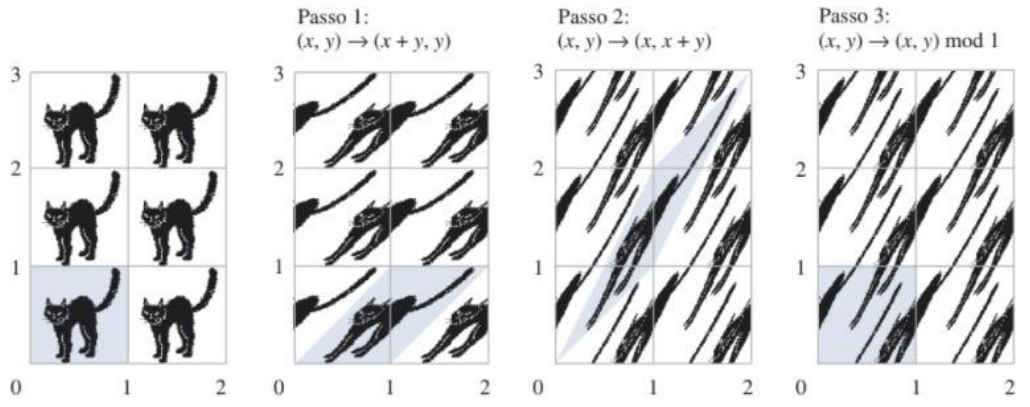


Figura 5: Pelo processo ladrilhanto

Quanto às faixas, observe a matriz abaixo:

$$C = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

A matriz C define a transformação do gato de Arnold e têm $det(C) = 1$. Devido ao seu determinante ser 1, o produto por essa matriz preserva áreas. (Observação: no primeiro método, o efeito do determinante era feito pelo $mod\ 1$ que era responsável por recortar e reagrupar os pedaços).

A simetria da matriz significa que seus autovalores são reais e os autovetores correspondentes são perpendiculares.

Os autovetores de C serão $\lambda_1 = \frac{3+\sqrt{5}}{2} = 2.618...$ e $\lambda_2 = \frac{3-\sqrt{5}}{2} = 0.3819.$

Os autovetores serão respectivamente: $v_1 = \begin{bmatrix} 1 \\ \frac{1+\sqrt{5}}{2} \end{bmatrix} = \begin{bmatrix} 1 \\ 1.6180 \end{bmatrix}$ e $v_2 = \begin{bmatrix} \frac{-1-\sqrt{5}}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} -1.6180 \\ 1 \end{bmatrix}$

Em cada aplicação da transformação, o autovalor causa uma dilatação na direção do seu respectivo autovetor num fator do valor do autovalor (a exemplo, o autovalor λ_1 causaria uma dilatação na direção do autovetor v_1 de fator 2.618..)

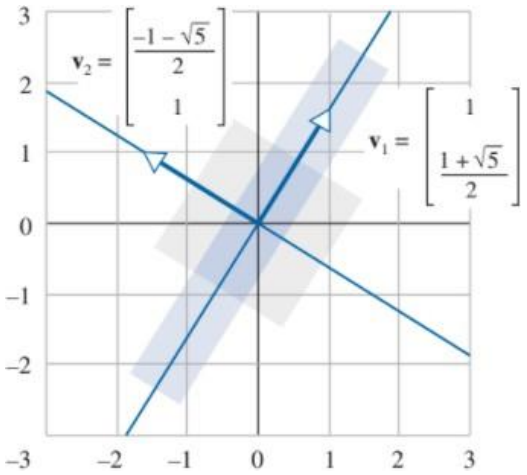


Figura 6: Sob ação da trasnformação, o quadrado é deformado no retângulo de lados paralelos às direções dos autovetores. As áreas do quadrado e do retângulo são iguais.

Considere, agora, S como uma parte do plano ladrilhado. Seja p um ponto de periodo n pertencente ao S. Por considerarmos o ladrilhamento, existe um ponto q com mesma cor que p e que com sucessivas iterações irá ocupar a posição inicial de p em uma determinada iterada. Esse ponto é $q = (C^{-1})^n p = C^{-1}p$, pois:

$$C^n q = C^n(C^{-n}p) = p$$

Desse modo, quando alguns pontos saem, outros pontos de mesma cor fluem para a direção destes pontos que sairam.

2.1.5 Pontos não periódicos

Quando passamos a considerar pontos $p \in S$ cuja uma das coordenadas (a, b) é um número irracional, estes pontos não são periódicos de modo que iterando sucessivamente, sempre obteremos pontos distintos em S.

Na figura abaixo, um ponto particular não parece se acumular em uma região específica e, em vez disso, parece se espalhar por todo espaço S.

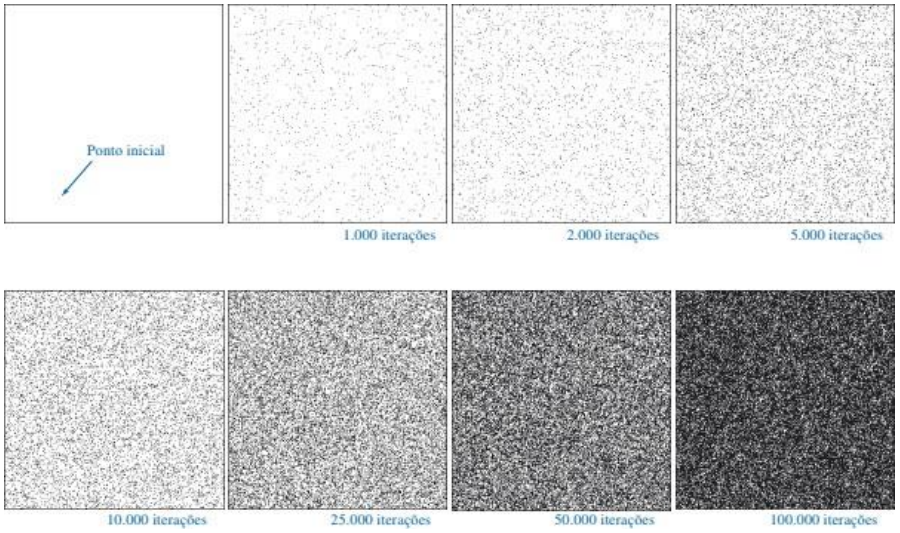
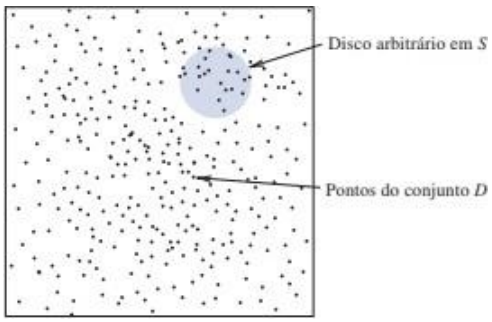


Figura 7: Note que quanto maior o número de iterações, mais as chances do ponto inicial passar por todos os pontos de S

Dizemos que um conjunto de pontos D de pontos S é denso em S se cada disco centrado em qualquer ponto de S contiver pontos de D por menor que seja o raio do disco.



2.1.6 Definição de caos

Definição: uma aplicação T de um conjunto S sobre sí mesmo é dita **caótica** se:

1. S contiver algum conjunto denso de pontos periódicos de T.

2. existir algum ponto em S cujas iteradas por T são densas em S

Note que a fusão de ordem e desordem caracteriza bem as aplicações caóticas. Neste trabalho, a transformação do gato de Arnold satisfaz a definição acima pois alguns pontos apresentam periodicidade enquanto outras não se movem de forma regular.

2.2 Aplicações

Uma aplicação clara da teoria do caos **consiste na previsão** do tempo. Na década de 1960, Edward Lorentz formulou a teria do caos analisando o clima e a sua relação com a condição inicial: Para cada condição inicial analisado, previa o clima. Mas notou-se que uma infima variação na condição inicial de uma das n variáveis que descrevem o clima, a longo prazo, levou em resultados completamente diferentes. Desse modo, torna-se difícil prever o clima para o próximo mês podendo ser previsível a um curto prazo desde que se tenha o valor **exato** de todas as condições iniciais de todos os parâmetros (o que é difícil de se obter pois podem existir parametros ainda desconhecidos, afetando o poder de previsão do sistema).

Como descrito no livro *Álgebra Linear com Aplicações* [1], uma aplicação caótica surge no estudo de **sistemas dinâmicos.** De forma breve, um sistéma dinâmico é um sistema da qual apresenta uma configuração específica em cada instante de tempo permitindo a compreensão de diversos tipos de sistemas como sistemas químicos, ecológicos, elétricos, biológicos, econômicos, etc.

Para um **Sistema dinâmico discreto**, o estado muda em cada pontos discretos do tempo em vez de mudar a cada instante e, por último, um **Sistema dinâmico discreto e caótico**, cada resultado gerado é resultado de uma aplicação caótica do estado anterior.

No estudo dos sistémas dinâmicos, um dos maiores problemas é prever configurações futuras do sistema a partir do conhecido e geralmente fazer o uso de um estado inicial exato é complexo devido a limitação dos instrumentos. Acreditava-se antigamente que computadores eliminariam esta limitação instrumental na medição, porém a descoberta de sistemas caóticos provou o contrário: Por menor que sejam o erro na medição do valor inicial, a cada iteração, o erro cresce exponencialmente impedindo uma predição dos estados futuros.

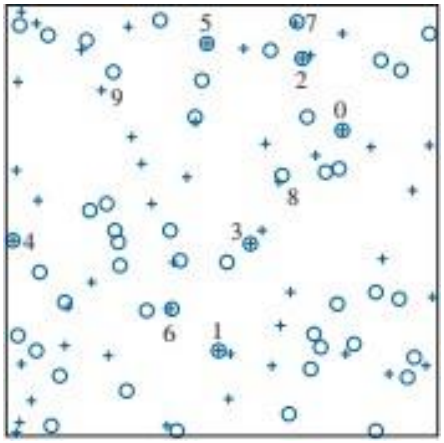


Figura 8: Na figura, apresentam 9 observações enumeradas. O simbolo + representa o ponto marcado computacionalmente enquanto o simbolo ⊗ representa o ponto real. Note que a cada iteração, os simbolos vão se distanciando comprovando que o erro compucional, por mínimo que seja, é acumulativo.

2.3 Resultados matemáticos

Este tópico serve unicamente para demonstrar resultados utilizados em tópicos anteriores.

1. Mostre que o único ponto que encontra-se inalterado nas transformações do gato de Arnold é $(0, 0)$.

   Demonstração. Considere o ponto $(0, 0)$.

   Aplicando $\Gamma : (x, y) \to (x + y, x + 2y)\ mod\ 1$ para o ponto $(0, 0)$:

   $$\Gamma(0, 0) = (0 + 0, 0 + 2 \cdot 0)\ mod\ 1 = (0, 0)$$

   Agora, para qualquer ponto (a, b) $(0, 0)$. Note que aplicando $\Gamma(a, b)$ ficaremos com $(a+b, a+2b)$ e como $a, b = 0$, o único ponto que permanece inalterado na transformação é a $(0, 0)$ como queriamos provar. □

2. Tomando $C = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$ e sabendo que $C^{25} = \begin{pmatrix} 7.778.742 & 12.586.269.025 \\ 12.586.269.025 & 20.365.011.074 \end{pmatrix}$

   sendo que $12.586.269.025$ é divisível por 101 e que o resto da divisão de $7.778.742$ e $20.365.011.074$ por 101 é 1, mostre que:

   (a) cada ponto em S da forma $(\frac{m}{101}, \frac{n}{101})$ retorna à sua posição inicial depois de 25 iterações da transformação do gato de Arnold

   Demonstração. Note que:

   $$C^{25} \begin{pmatrix} \frac{m}{101} \\ \frac{n}{101} \end{pmatrix} mod\ 1 = \begin{pmatrix} 7.778.742 & 12.586.269.025 \\ 12.586.269.025 & 20.365.011.074 \end{pmatrix} \begin{pmatrix} \frac{m}{101} \\ \frac{n}{101} \end{pmatrix} mod\ 1$$

   $$= \begin{pmatrix} \frac{7.778.742m + 12.586.269.025n}{101} \\ \frac{12.586.269.025m + 20.365.011.074n}{101} \end{pmatrix} mod\ 1$$

   $$= \begin{pmatrix} \frac{101 \cdot 77.017.248m}{101} + \frac{m}{101} + \frac{101 \cdot 124.616.525n}{101} \\ \frac{101 \cdot 124.616.525m}{101} + \frac{101 \cdot 201.633.473n}{101} + \frac{n}{101} \end{pmatrix} mod\ 1$$

   Aplicando $mod\ 1$:

   $$\begin{pmatrix} \frac{101 \cdot 77.017.248m}{101} mod\ 1 + \frac{m}{101} mod\ 1 + \frac{101 \cdot 124.616.525n}{101} mod\ 1 \\ \frac{101 \cdot 124.616.525m}{101} mod\ 1 + \frac{101 \cdot 201.633.773n}{101} mod\ 1 + \frac{n}{101} mod\ 1 \end{pmatrix}$$

   Note também que se $a \in \mathbb{Z}$, temos que $a\ mod\ 1 = 0$.

   Portanto, $77.017.248\ mod\ 1 = 124.616.525\ mod\ 1 = 201.633.713\ mod\ 1 = 0$ e desse modo:

   $$\begin{pmatrix} \frac{m}{101} \\ 0 + 0 + \frac{n}{101} mod\ 1 \end{pmatrix} = \begin{pmatrix} \frac{n}{101} \\ \frac{m}{101} \end{pmatrix} mod\ 1$$

Com isso:

$$C^{25} = \begin{pmatrix} \frac{m}{101} \\ \frac{n}{101} \end{pmatrix} \mod 1 = \begin{pmatrix} \frac{n}{101} \\ \frac{m}{101} \end{pmatrix} \mod 1$$

Ou seja, voltamos para o ponto inicial. □

(b) Mostre que cada ponto em S da forma $(\frac{m}{101}, \frac{n}{101})$ tem período 1, 5 ou 25.

Demonstração. Seja p o periodo de a. Se $C^n(a) = a$ então p divide n. Pelo algoritmo da divisão $n = qp + r$, $r < p$ ou $r = 0$. Suponha $r < p$, temos:

$$\begin{aligned} a = C^n(a) &= C^{ap+r}(a) = C^n(C^{ap}(a)) \\ &= C^r(C^{p+p+\ldots+p}(a)) \text{ Obs: } C^{p+p+\ldots+p} = C^{a\cdot p} \\ &= C^r(C^p(\ldots(C^p(a))\ldots)) \\ &= C^r(a) \end{aligned}$$

Como a definição de período é o menor inteiro positivo p tal que $C^p(a) = a$, como $r < p$, chegamos a uma contradição.

Logo $r = 0$.

Portanto $n = qp$.

No caso de $n = 25$, temos 1, 5, 25 dividem 25 e esses são os períodos possíveis. □

(c) Mostre que $\Pi(101) = 25$.

Demonstração. Por definição, a aplicação $\Pi(x)$ é o *mdc* entre todos os possíveis períodos de x^2 pontos. Como foi visto no ítem (b), os possíveis períodos para $p = 101$ é 1, 5 e 25.

$$mdc(1, 5, 25) = 25$$

□

O algoritmo formulado para responder as duas tarefas, segue abaixo:

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

def leia_imagem(diretorio):
    ''' (string) -> (array)
    Essa fun ção recebe um diretó rio de um arquivo txt onde as
    infomarções estão separadas por espaços e retorna uma array
    de floats com todos os dados presentes no txt

    '''
    arquivo = open(diretorio, "r" )
    arquivo_linhas = arquivo.readlines()


    for i in range(len(arquivo_linhas)):
        arquivo_linhas[i] = arquivo_linhas[i].split()

    arquivo.close()
    matriz = np.array(arquivo_linhas, dtype='float64')

    return matriz

def cria_imagem(R,G,B):
    ''' (array, array, array) -> imagem
    Essa função recebe matrizes de cada cor e retorna a imagem
    gerada por essas três matrizes.
    '''
    lin =len(R)
    col = len(R[0])

    M = np.zeros((lin,col,3), dtype = "float64")
    M[:,:,0]  = R
    M[:,:,1]  = G
    M[:,:,2]  = B
    im = plt.figure (figsize = (4,4))
    im = plt.imshow(M, aspect = 'auto')
    plt.axis("off")


    return im


def desembaralha_professor():


    c1 = input("Digite o diretório do arquivo txt que representa a
       cor vermelha:\n")
    R = leia_imagem(c1)
    c2 = input("Digite o diretório do arquivo txt que representa a
       cor verde:\n")
    G = leia_imagem(c2)
    c3 = input("Digite o diretório do arquivo txt que representa a
       cor azul:\n")
    B = leia_imagem(c3)

    print()

    n = int(input("Digite o n :  "))
```

```
57
    R1 = G1 = B1 = np.zeros((101,101))
59
    print("A imagem embaralhada é:\n")
61
    cria_imagem(R,G,B)
63
    plt.savefig('imagem_original.jpg', format='jpg')
65
    R1,G1,B1 = embaralha(n, R,G,B)
67
    print("A imagem após a transformação, fica:\n")
69
    cria_imagem(R1,G1,B1)
71
    i = str(n)
73
    path = input("Digite um diretorio")
75
    plt.savefig(diretorio + i + 'iterada.jpg', format='jpg')
77
    plt.show()
79
def embaralha_colega():
81
    path = input("Digite o diretório da imagem a ser embaralhada:\n
        ")
83
    n = int(input("Digite o número de iterações da transformações
       de arnold você deseja fazer: "))
85
    R,G,B = separa_imagem(path)
87
    R1,G1,B1 = embaralha(n, R,G,B)
89
    salva_cores_txt(R1,G1,B1)
91
    print("Geramos uma imagem embaralhada com as mesma função do
       gato de arnold com n = ",n)
93  print()
    print("A imagem a ser decodificada:")
95  cria_imagem(R1,G1,B1)
97  path = input('Digite um diretório:')          plt.
    savefig(path +'+embaralhado.jpg', format='jpg')
99
101 return
103
def desembaralha_colega():
105
    print("Digite o diretório do txt de cada cor conforme pedido
       abaixo:\n")
107 c1 = input("Digite o diretório do arquivo txt que representa a
       cor vermelha:\n")
    R = leia_imagem(c1)
109 c2 = input("Digite o diretório do arquivo txt que representa a
       cor verde:\n")
    G = leia_imagem(c2)
```

```
111     c3 = input("Digite o diretório do arquivo txt que representa a
            cor azul:\n")
        B = leia_imagem(c3)
113
        print("A imagem embaralhada:")
115
        cria_imagem(R,G,B)
117     plt.savefig('imagem_original_do_colega.jpg', format = 'jpg')
        plt.show()
119
        print("Aqui geraremos as imagens mas cabe aos participantes
            avaliarem qual é a certa :D")
121     diretorio = input("Digite o diretório que deseja salvar as
            imagens geradas:\n")

123     print("Testando todas as iterações com n = 1,...,25:\n")

125     for i in range(25):

127         R1,G1,B1 = embaralha(i, R,G,B)
            cria_imagem(R1,G1,B1)
129         n = str(i)
            plt.savefig('diretorio'+ n +'_iteradas_imagem_colega.jpg',
                format='jpg')
131
        print("Geramos as 25 imagens")
133
    def separa_imagem(diretorio):
135     ''' (str) -> (array, array, array)
        Esse função recebe um diretório onde se encontra a imagem e
137     retorna três matrizes que separam as cores de imagem, usando
        o sistema RGB.
139     Observação: o diretório tem que conter qual o arquivo e a
        extensão
141     '''
        # lendo a imagem
143     M = Image.open(diretorio)

145     # Transformando a imagem numa matriz
        M = np.asarray(M, dtype = "float64")/255
147


149     # visualizando

151     plt.figure(figsize = (4,4))
        plt.imshow(M, aspect = 'auto')
153     plt.axis('off')

155     # Separando as cores

157     R = M[:,:,0]
        G = M[:,:,1]
159     B = M[:,:,2]


161
        return R,G,B
163
    def salva_cores_txt(R,G,B):
165     ''' (array,array,array) -> None
        Função que cria que salva as matrizes de cores em txt
```

```
    '''

    np.savetxt('image_R.txt', R, newline='\n', fmt = '%.18f')
    np.savetxt('image_G.txt', G, newline='\n', fmt = '%.18f')
    np.savetxt('image_B.txt', B, newline='\n', fmt = '%.18f')

    return

def ciclo(i,j,n):
    x = i
    y = j
    for k in range(n):
        xn = (x + y)%101
        yn = (x + 2*y)%101
        x = xn
        y = yn

    return x,y

def embaralha(n, R,G,B):
    ''' (int, array, array, array) -> (array, array, array)
        Essa Função recebe as cores de uma imagem embaralhada
        pela transformação do gato de arnold e retorna desembaralha
        de acordo com n fornecido.
    '''
    lin = len (R)
    col = len(R[0])

    R1 = G1 = B1 = np.zeros((101,101))

    for i in range(lin):
        for j in range(col):
                x, y = ciclo(i,j,n)

                R1[x][y] = R[i][j]
                G1[x][y] = G[i][j]
                B1[x][y] = B[i][j]



    return R1,G1,B1

def main():

    print("Esse programa embaralha e desembaralhar uma imagem
       usando a transformação do gato de arnold com uma imagem de
       101x101.\n")

    print('Você deseja:\n')
    print('1 - Desembaralhar a imagem do professor?\n2 - Embaralhar
        a imagem de um colega?\n3 - Desembaralhar a imagem de um
       colega?\n')

    option = int(input("Digite a opção desejada:  "))

    if option == 1:
        desembaralha_professor()
    elif option == 2:
        embaralha_colega()
    elif option == 3:
```

```
223             desembaralha_colega()
        else:
225             print("Opção inválida\n")

227     print("\nBye!")

229 if __name__ == "__main__":
        main()
```

Listing 1: Algoritmo para decodificação.

3 Tarefa 1

A imagem que desejamos decodificar encontra-se abaixo:

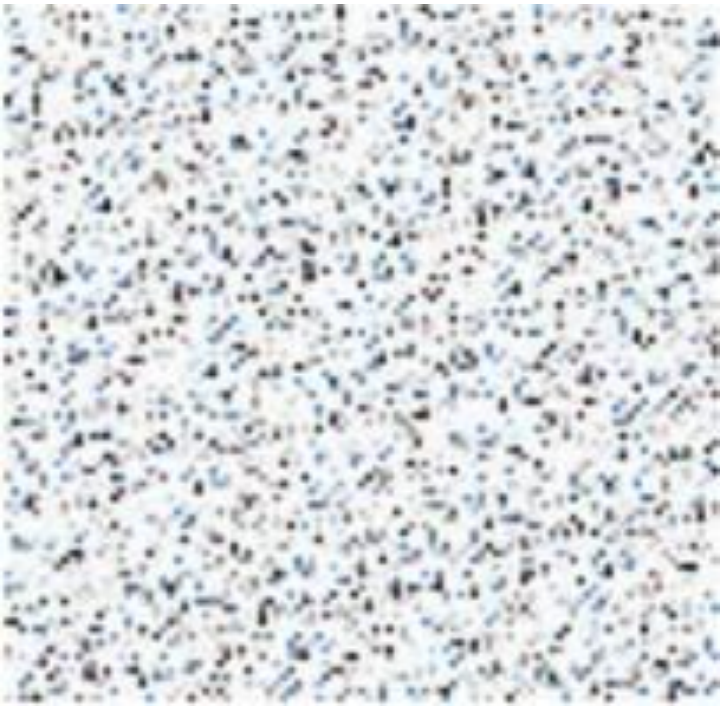


Figura 9: A imagem que desejamos decodificar

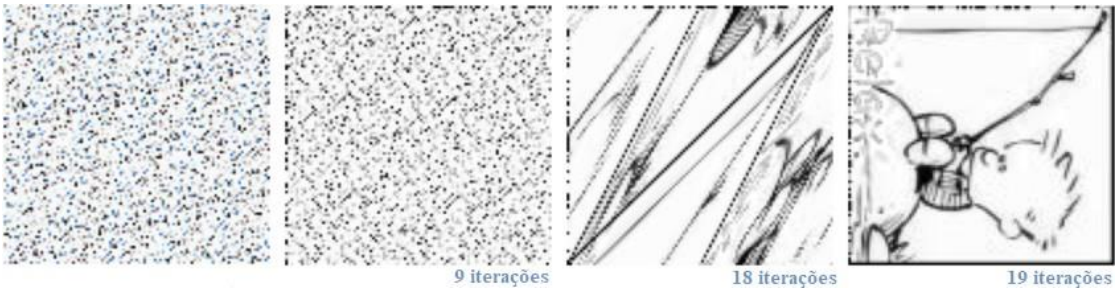
Se a imagem inicial não encontrasse embaralhado, ao reproduzir 25 iterações, supostamente a imagem iria embaralhar os pixels e retornar a sua posição original. Porém, como a imagem está embaralhado, foi-se necessário encontrar quantas iterações restam para retornar a posição original. (Ou seja, encontrar n com $n \leq 2, \ldots, 24$).

Neste caso, foi-se necessário 19 iterações para formar a figura abaixo:

Figura 10: A imagem decodificada

Desse modo, como o periodo para uma imagem de $p = 101$ pixels de lado é 25 iterações, a imagem embaralhada encontrava-se na 6° iteração.

Abaixo, encontra-se duas iterações intermediarias:



4 Tarefa 2

Para a tarefa dois como não encontramos grupos para realizar a decodificação, os participantes decidiram escolher uma imagem e embaralhar usando a a transformação do gato de Arnold.

Figura 11: A imagem escolhida pela Ivy

Figura 12: A imagem escolhida pela Karol

O embaralhamento se deu pelo mesmo algoritmo em ambos os casos apenas alterando o período da imagem. Trocamos os arquivos txt das imagens e como já sabemos que após 25 iterações nos voltamos a imagem original, geramos imagens para todas essas interações e avaliamos qual era a imagem a ser decodificada.

Figura 13: A imagem embaralhada que a Ivy recebeu

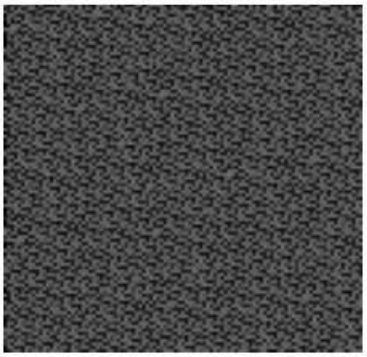


Figura 14: A imagem embaralhada que a Karol recebeu

A ivy usou o perídodo $p = 10$ e a Karol usou o período $p = 16$.
E após os uso do algoritmo voltaram as seguintes imagens:

Figura 15: A imagem decodificada pela Ivy

Figura 16: A imagem decodificada pela Karol

Perceba que por se tratar de um sistema caótico nos não obtemos a imagem original e ainda tem alguns pontos destoantes, esses pontos são os pontos irracionais do sistema.

Outro ponto é que a imagem ficou preto e branco devido a biblioteca usada na geração da imagem, como tivemos que normalizar as cores para um numero [0,1] não foi possível deixá-la colorida.

5 Nota posterior

(Dia 22 de julho) Foi recebido a imagem embaralhada abaixo:

Figura 17: Imagem enviado por outro grupo

Como sabemos que após 25 iterações voltamos para a imagem original, geramos

imagens para todas essas iterações e avaliamos qual era a imagem a ser decodificada. Porém:

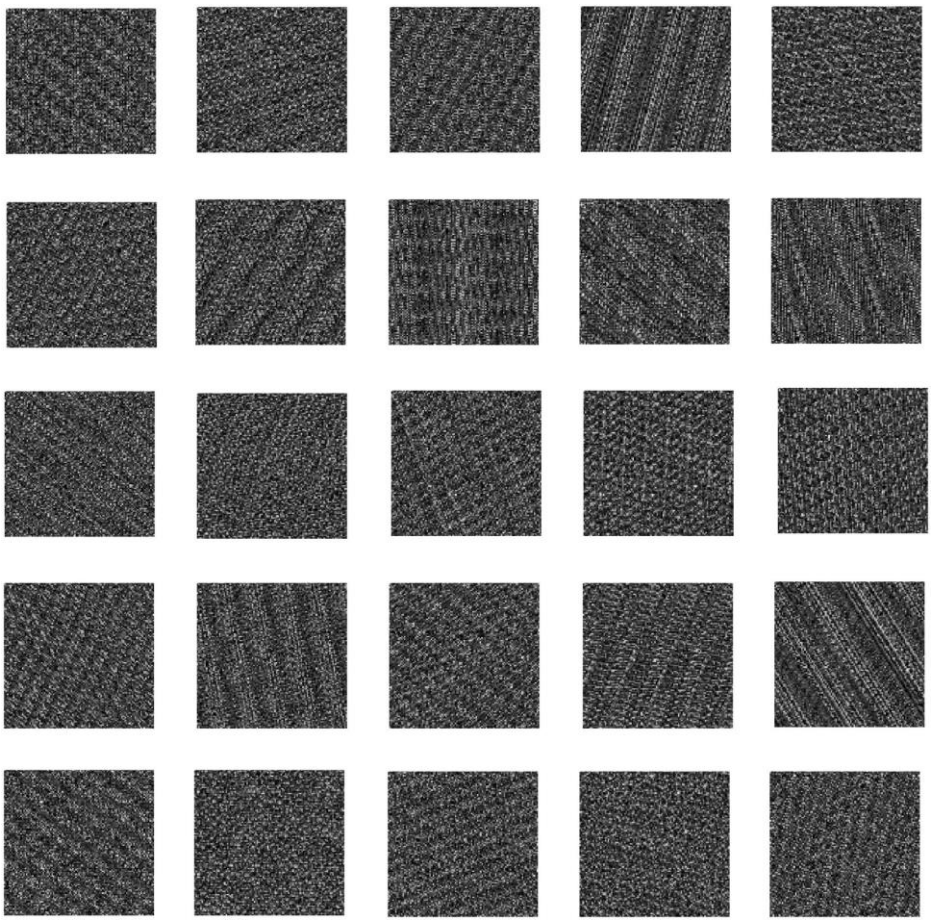


Figura 18: As 25 iterações para decodificar a imagem enviada pelo outro grupo

Nenhuma das 25 iterações parece ter produzido uma imagem.
Três possíveis causas para não decodificação da imagem:

- Limitação do algoritmo produzido
- Possibilidade de erro no arquivo enviado
- A mudança no algoritmo para emabbaralhar as imagens.

Em anexo ao trabalho, é colocado o arquivo *RGB* enviado por outro grupo.

6 Considerações Finais

Em suma, o projeto consistiu na compreensão da transformação caótica específica denominada transformação do gato de Arnold com objetivo primordial de formular um algoritmo para decodificar imagens. Quanto à pergunta destacada feita acima, é possível afirmar que o algoritmo apresentado foi capaz de solucionar o problema sugerido, retornando a imagem decodificada.

É interessante verificar que o inverso também pode ser feito: Gerar imagens para serem decodificados (como feito na **Tarefa 2**). Deve ficar claro que existem muitos outros métodos e processos caóticos que podem ser desenvolvidas de forma semelhante ao feito neste projeto.

Referências

[1] ANTON, H. e RORRES, C. (2012) *Álgebra Linear com Aplicações - 10ª edição* Disponível em: http://www.professores.uff.br/jcolombo/wp-content/uploads/sites/124/2018/08/Algebra_Linear_com_Aplica_10_-Edi_Anton_Rorres.pdf (Acessado em maio de 2021).