Sistemas Dinâmicos não Lineares
Atividade 5 - 2020

Renata Biaggi Biazzi

16 de maio de 2020

1 Sistema de Rossler

Com base na seção 9.3 do livro "CHAOS: An Introduction to Dynamical Systems, Alligood, Sauer and Yorke, Springer, 1996", estudaremos o sistema de equações de Rössler:

$$\dot{x} = -y - z$$

$$\dot{y} = x + ay$$

$$\dot{z} = b + (x - c)z$$

a) Primeiro iremos simular numericamente as equações para diferentes parâmetros de c, mantendo os valores $a = b = 0.1$.

O código usado para isso foi feito em Python:

```
#Função que calcula o sistema de equações
    def ros (x:float,y:float,z:float,c: float) -> Tuple[float,float,float]:
    xt = -y -z
    yt = x + 0.1*y
    zt = 0.1 +(x-c)*z

    return (xt,yt,zt)

#Função que usa o método de Runge-Kutta 4 para o cálculo numérico com base na função ros
def rk4 (x: float, y: float, z: float, c: float, n: int,  dt: float) -> Tuple[float,float,float]:
    for i in range (n):
        (xa,ya,za) = ros(x,y,z,c)
        (xb,yb,zb) = ros(x+dt*0.5*xa,y+dt*0.5*ya,z+dt*0.5*za,c)
        (xc,yc,zc) = ros(x+dt*0.5*xb,y+dt*0.5*yb,z+dt*0.5*zb,c)
        (xd,yd,zd) = ros(x+dt*xc,y+dt*yc,z+dt*zc,c)
        x = x + dt/6*(xa + 2*xb + 2*xc + xd)
        y = y + dt/6*(ya + 2*yb + 2*yc + yd)
        z = z + dt/6*(za + 2*zb + 2*zc + zd)
        yield (x,y,z)

#dt = tamanho do passo de integração
#n = número de pontos (x,y,z) que são calculados
```

E os plots foram feitos com os seguintes códigos:

```
#Função que faz o plot 3d
    def plot3d (c, n, dt,N,s):
    rd.seed(s)
    fig = plt.figure ( figsize = (8.5,8.5))
    ax = fig.gca ( projection = '3d' )
    for i in range(N):
        x0 = rd.randint(-10,10)
```

```
        y0 = rd.randint(-10,10)
        z0 = rd.randint(-1,1)
        (X,Y,Z) = zip(*rk4(x0,y0,z0,c,n,dt))
        ax.plot ( X, Y, Z, linewidth = 0.5)
    ax.grid ( True )
    ax.set_xlabel ( 'x' )
    ax.set_ylabel ( 'y' )
    ax.set_zlabel ( 'z' )
    ax.set_title ( 'Atrator de Rossler 3D Plot' )
    plt.show ( )
    return

#Função que faz o plot 2d a partir de valores aleatórios para os pontos iniciais (x0,y0,z0)

 def plot2d_rand(c,n,dt,N,s):
    rd.seed(s)
    for i in range(N):
        x0 = rd.randint(-10,10)
        y0 = rd.randint(-10,10)
        z0 = rd.randint(-1,1)
        (X,Y,Z) = zip(*rk4(x0,y0,z0,c,n,dt))
        plt.plot(X, Y, ',')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Atrator de Rossler')
    plt.show()
    return

#Função que faz o plot 2d a partir de valores bem definidos para (x0,y0,z0)
#Essa função só plota os pontos a partir de 90000 pontos

def plot2d(x0,y0,z0,c,n,dt,N):
    for i in range(N):
        (X,Y,Z) = zip(*rk4(x0,y0,z0,c,n,dt))
        plt.plot(X[90000:], Y[90000:], 'k,')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Atrator de Rossler')
    plt.show()
    return
```

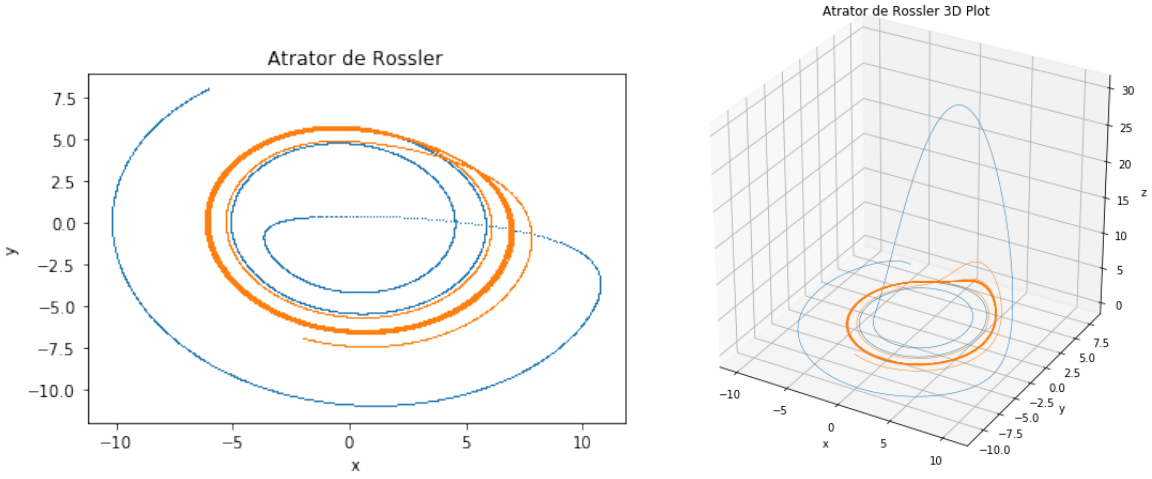
Então, para fazer os códigos funcionarem, precisamos fazer as seguintes definições das variáveis e chamar as funções da seguinte maneira:

```
c=4
s=1
n = 500000 #number of time points
dt = 0.005 #interval between points
N = 2 #number of initial points
x0 = 0
y0 = 5.5
z0 = 0
plot2d_rand(c,n,dt,N,s)
plot3d(c,n,dt,N,s)
plot2d(x0,y0,z0,c,100000,dt,1)
```

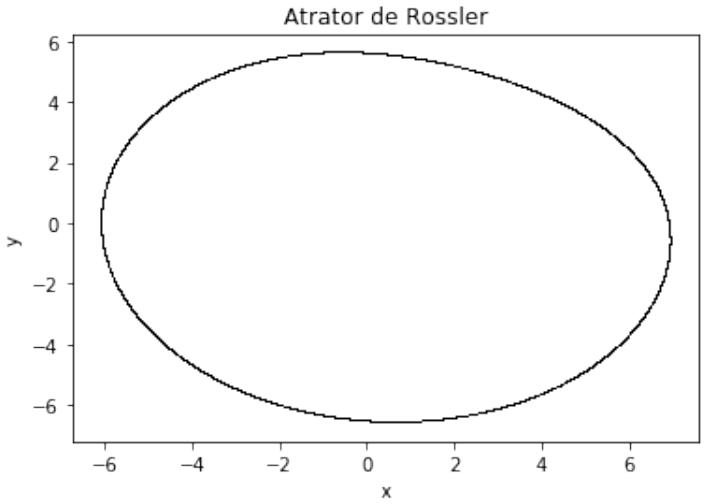
c=4

Primeiro foi simulada a função com valores iniciais aleatórios para identificar o atrator caótico. Na

figura abaixo, vemos que o ponto inicial e o ponto inicial laranja aos poucos convergem para o anel circular central na figura.



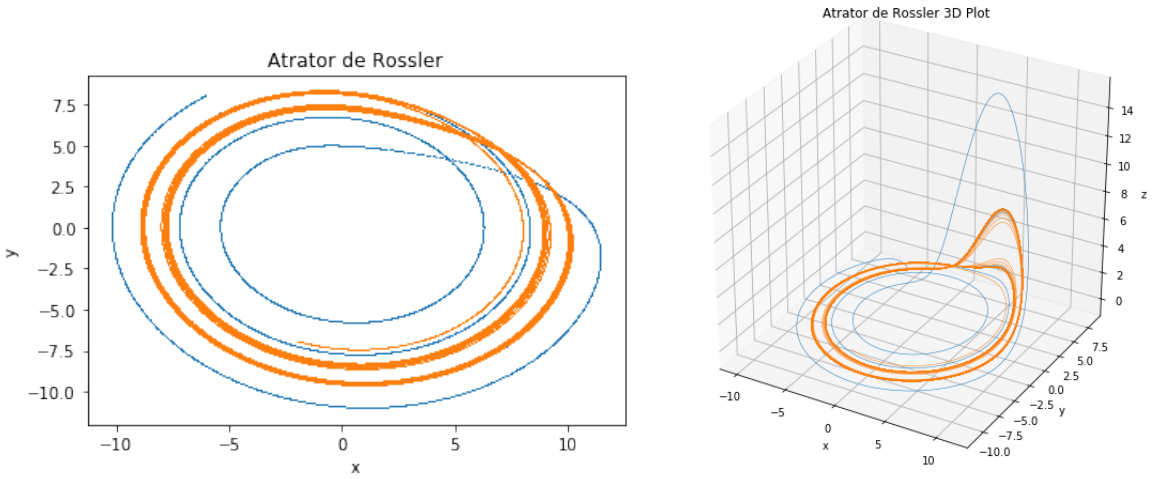
Então usando o ponto inicial: $(x_0, y_0, z_0) = (0.1, 5.5, 0.1)$, fazemos uma simulação de 10^6 pontos e descartamos os $n = 9.10^5$ primeiros pontos, ficando só com os pontos referentes ao atrator, e temos a seguinte figura:



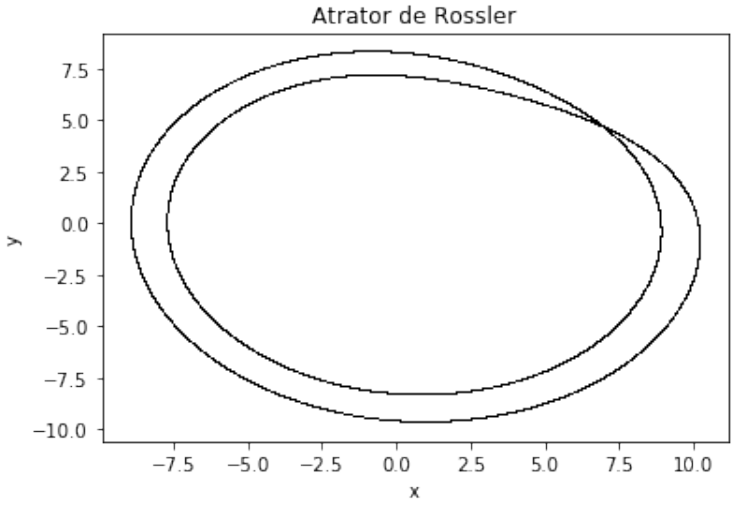
Essa figura é igual a figura 9.7.a (Figure 9.7 Attractors of the Rössler system as c is varied.) do livro citado anteriormente, corresponde a um atrator de período 1.

c=6

Primeiro foi simulada a função com valores iniciais aleatórios para identificar o atrator caótico. Na figura abaixo, vemos que o ponto inicial e o ponto inicial laranja aos poucos convergem para o anel circular central na figura.



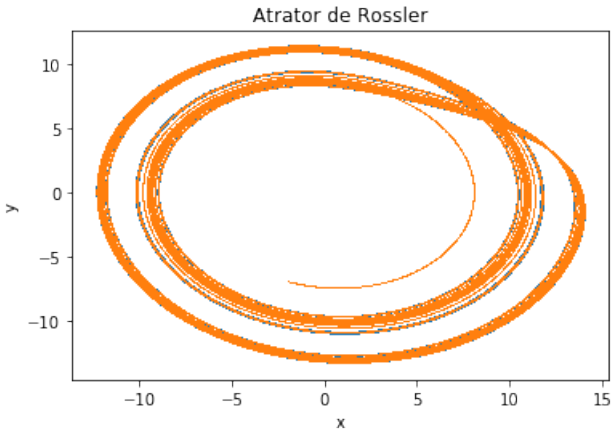
Então usando o ponto inicial: $(x_0, y_0, z_0) = (0.1, 7.5, 0.1)$, fazemos uma simulação de 10^6 pontos e descartamos os $n = 9.10^5$ primeiros pontos, ficando só com os pontos referentes ao atrator, e temos a seguinte figura:

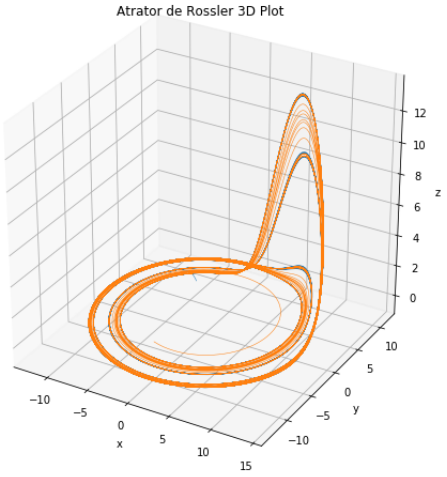


Essa figura é igual a figura 9.7.b (Figure 9.7 Attractors of the Rössler system as c is varied.) do livro citado anteriormente, corresponde a um atrator de período 2.

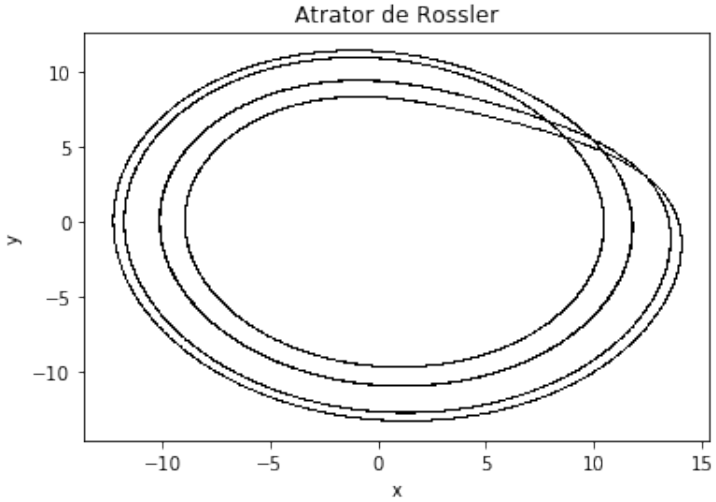
c=8.5

Primeiro foi simulada a função com valores iniciais aleatórios para identificar o atrator caótico. Na figura abaixo, vemos que o ponto inicial e o ponto inicial laranja aos poucos convergem para o anel circular central na figura.





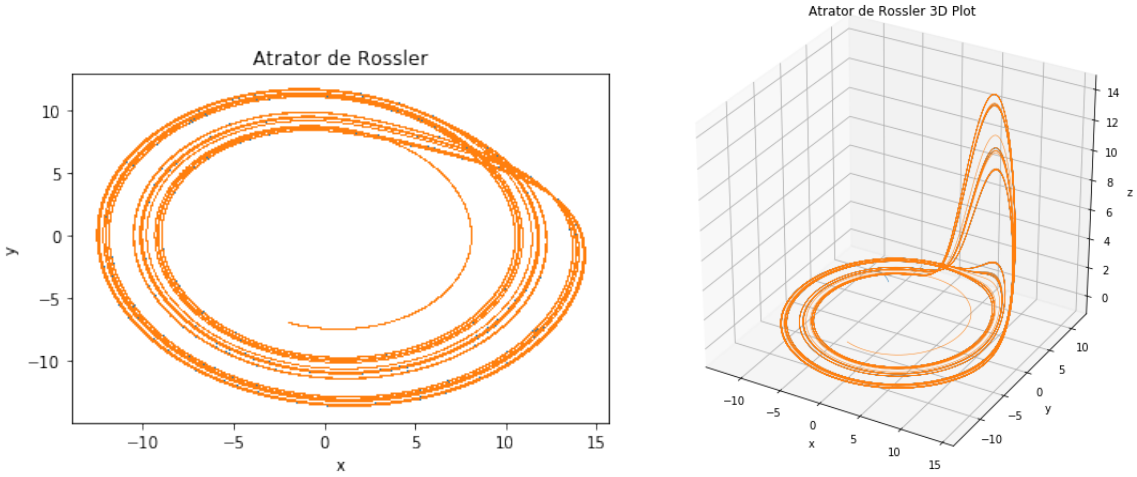
Então usando o ponto inicial: $(x_0, y_0, z_0) = (0.1, 7.5, 0.1)$, fazemos uma simulação de 10^6 pontos e descartamos os $n = 9.10^5$ primeiros pontos, ficando só com os pontos referentes ao atrator, e temos a seguinte figura:



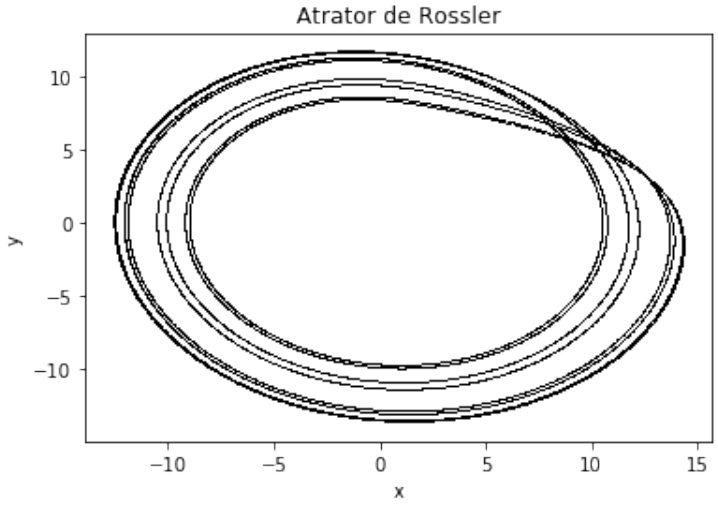
Essa figura é igual a figura 9.7.c (Figure 9.7 Attractors of the Rössler system as c is varied.) do livro citado anteriormente, corresponde a um atrator de período 4.

c=8.7

Primeiro foi simulada a função com valores iniciais aleatórios para identificar o atrator caótico. Na figura abaixo, vemos que o ponto inicial e o ponto inicial laranja aos poucos convergem para o anel circular central na figura.



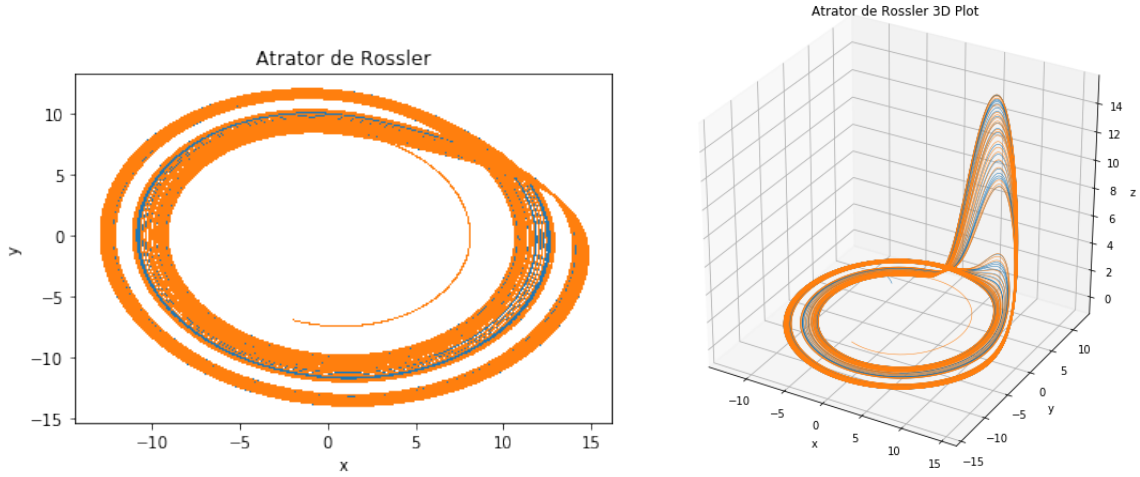
Então usando o ponto inicial: $(x_0, y_0, z_0) = (0.1, 7.5, 0.1)$, fazemos uma simulação de 10^6 pontos e descartamos os $n = 9.10^5$ primeiros pontos, ficando só com os pontos referentes ao atrator, e temos a seguinte figura:



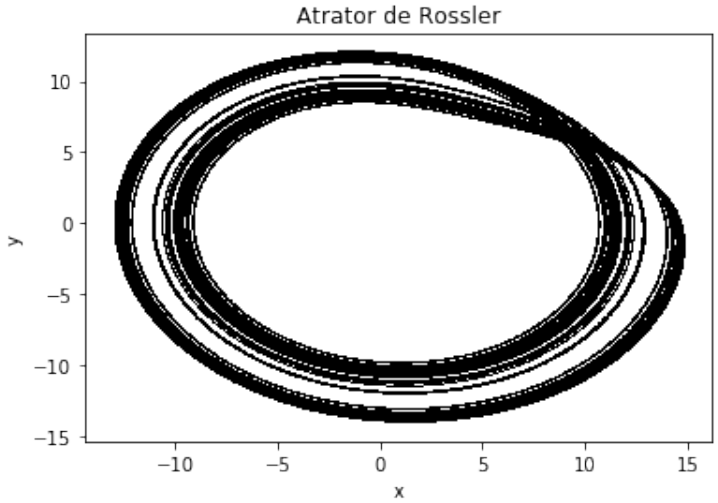
Essa figura é igual a figura 9.7.d (Figure 9.7 Attractors of the Rössler system as c is varied.) do livro citado anteriormente, corresponde a um atrator de período 8.

c=9

Primeiro foi simulada a função com valores iniciais aleatórios para identificar o atrator caótico. Na figura abaixo, vemos que o ponto inicial e o ponto inicial laranja aos poucos convergem para o anel circular central na figura.



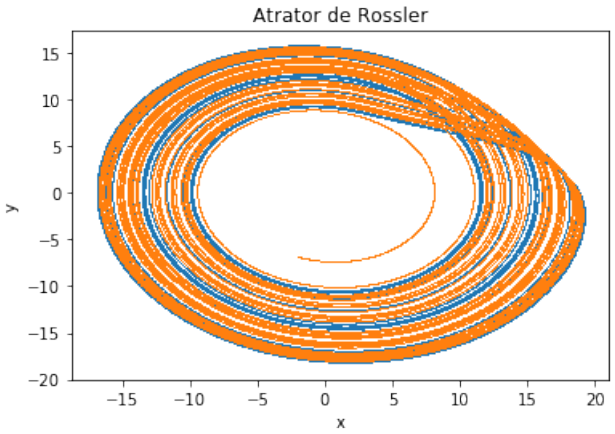
Então usando o ponto inicial: $(x_0, y_0, z_0) = (0.1, 7.5, 0.1)$, fazemos uma simulação de 20^6 pontos e descartamos os $n = 9.10^5$ primeiros pontos, ficando só com os pontos referentes ao atrator, e temos a seguinte figura:

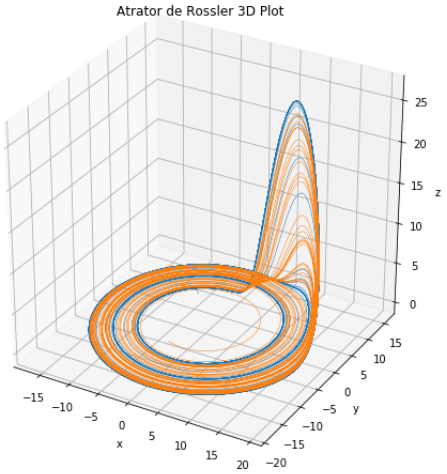


Essa figura é igual a figura 9.7.e (Figure 9.7 Attractors of the Rössler system as c is varied.) do livro citado anteriormente, corresponde a um atrator caótico, tanto que foi necessário simular 20^6 pontos no total.

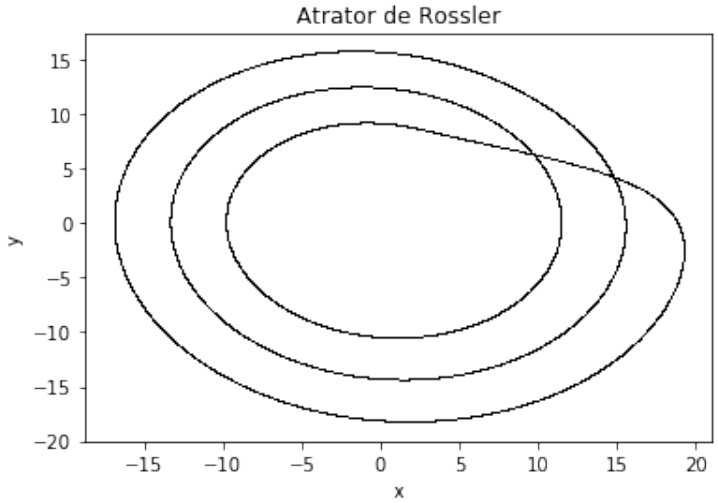
c=12

Primeiro foi simulada a função com valores iniciais aleatórios para identificar o atrator caótico. Na figura abaixo, vemos que o ponto inicial e o ponto inicial laranja aos poucos convergem para o anel circular central na figura.





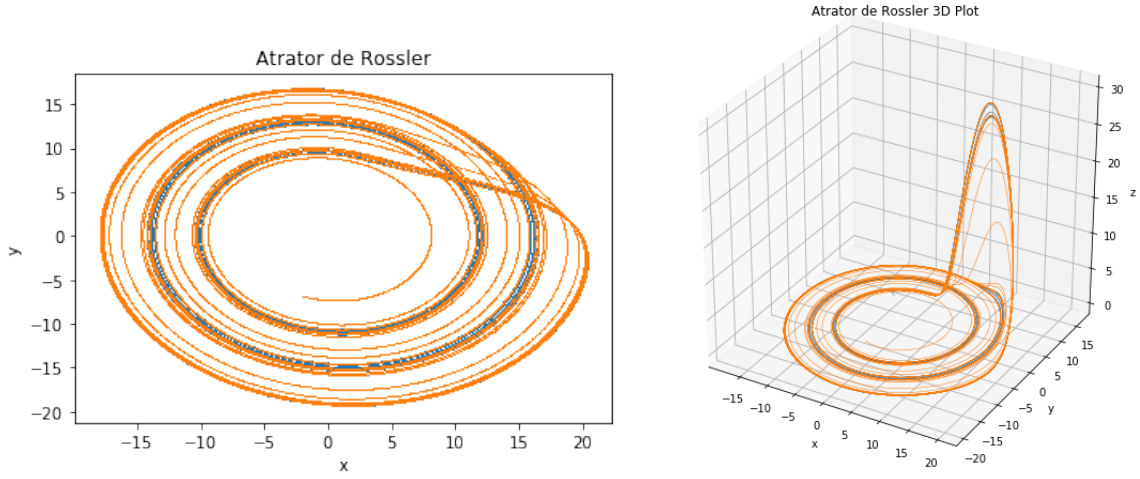
Então usando o ponto inicial: $(x_0, y_0, z_0) = (0.1, 7.5, 0.1)$, fazemos uma simulação de 10^6 pontos e descartamos os $n = 9.10^5$ primeiros pontos, ficando só com os pontos referentes ao atrator, e temos a seguinte figura:



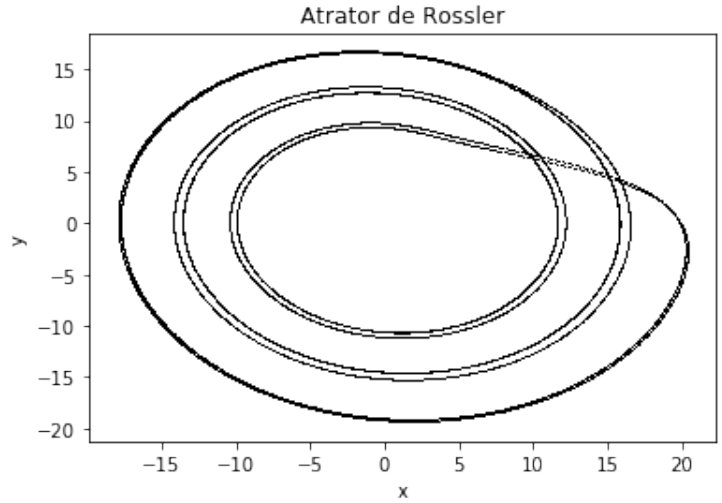
Essa figura é igual a figura 9.7.f (Figure 9.7 Attractors of the Rössler system as c is varied.) do livro citado anteriormente, corresponde a um atrator de período 3.

c=12.8

Primeiro foi simulada a função com valores iniciais aleatórios para identificar o atrator caótico. Na figura abaixo, vemos que o ponto inicial e o ponto inicial laranja aos poucos convergem para o anel circular central na figura.



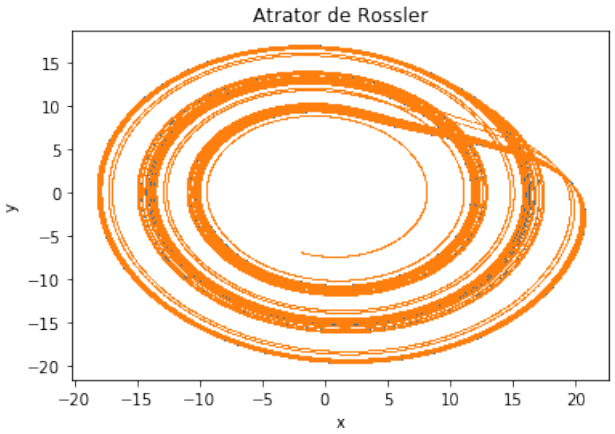
Então usando o ponto inicial: $(x_0, y_0, z_0) = (0.1, 7.5, 0.1)$, fazemos uma simulação de 10^6 pontos e descartamos os $n = 9.10^5$ primeiros pontos, ficando só com os pontos referentes ao atrator, e temos a seguinte figura:

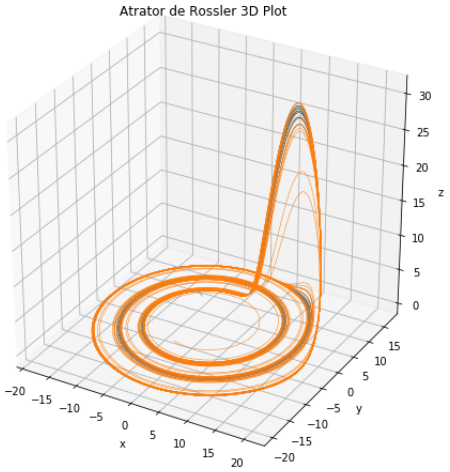


Essa figura é igual a figura 9.7.g (Figure 9.7 Attractors of the Rössler system as c is varied.) do livro citado anteriormente, corresponde a um atrator de período 6.

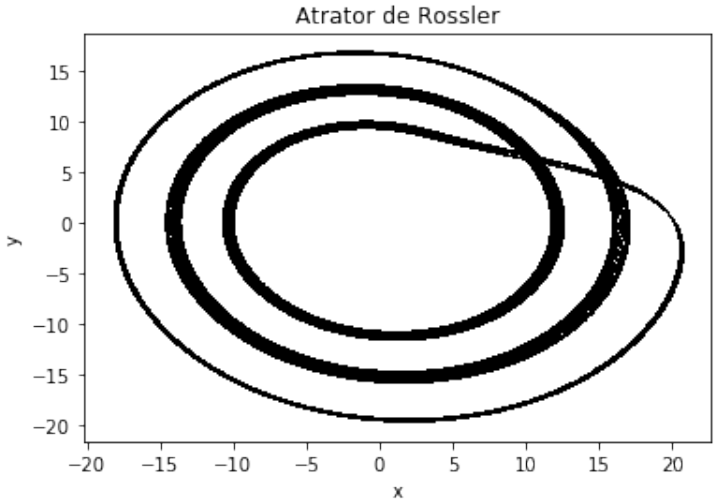
c=13

Primeiro foi simulada a função com valores iniciais aleatórios para identificar o atrator caótico. Na figura abaixo, vemos que o ponto inicial e o ponto inicial laranja aos poucos convergem para o anel circular central na figura.





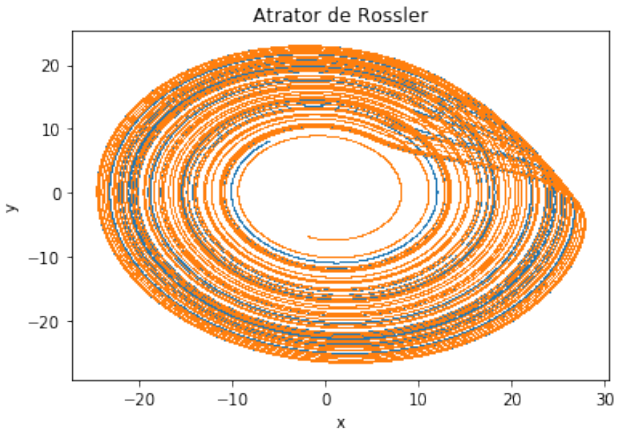
Então usando o ponto inicial: $(x_0, y_0, z_0) = (0.1, 7.5, 0.1)$, fazemos uma simulação de 10^6 pontos e descartamos os $n = 9.10^5$ primeiros pontos, ficando só com os pontos referentes ao atrator, e temos a seguinte figura:

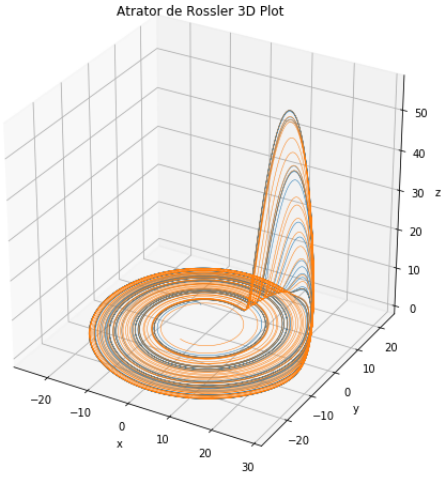


Essa figura é igual a figura 9.7.h (Figure 9.7 Attractors of the Rössler system as c is varied.) do livro citado anteriormente, corresponde a um atrator caótico, tanto que foi necessário simular 20^6 pontos no total.

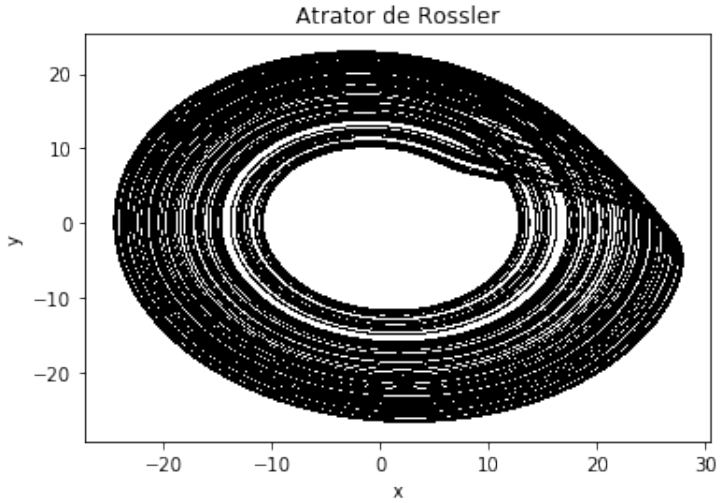
c=18

Primeiro foi simulada a função com valores iniciais aleatórios para identificar o atrator caótico. Na figura abaixo, vemos que o ponto inicial e o ponto inicial laranja aos poucos convergem para o anel circular central na figura.





Então usando o ponto inicial: $(x_0, y_0, z_0) = (0.1, 7.5, 0.1)$, fazemos uma simulação de 10^6 pontos e descartamos os $n = 9.10^5$ primeiros pontos, ficando só com os pontos referentes ao atrator, e temos a seguinte figura:



Essa figura é igual a figura 9.7.i (Figure 9.7 Attractors of the Rössler system as c is varied.) do livro citado anteriormente, corresponde a um atrator caótico, tanto que foi necessário simular 20^6 pontos no total.

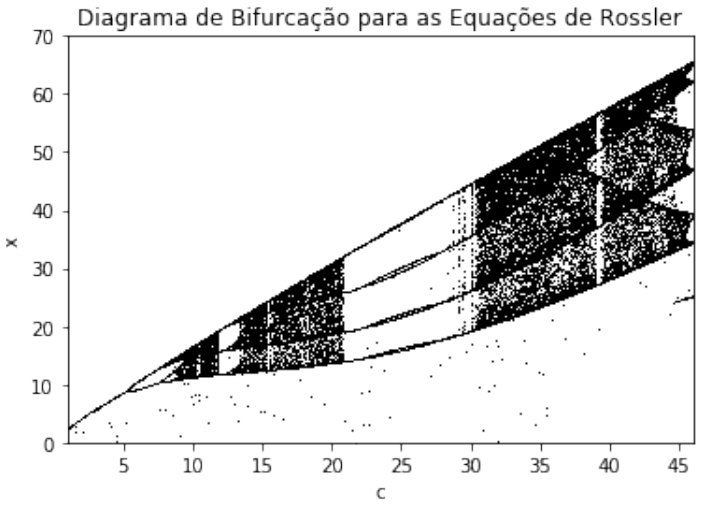
b) Para obter o diagrama de bifurcação da Fig. 9.8. do livro citado, usamos o seguinte código:

```
#Função ros() exatamente igual ao exercício anterior.

#Função que calcula todos os valores de x para um determinado n
def rk4_x (x: float, y: float, z: float, c: float, n: int,  dt: float):
    for i in range (n):
        (xa,ya,za) = ros(x,y,z,c)
        (xb,yb,zb) = ros(x+dt*0.5*xa,y+dt*0.5*ya,z+dt*0.5*za,c)
        (xc,yc,zc) = ros(x+dt*0.5*xb,y+dt*0.5*yb,z+dt*0.5*zb,c)
        (xd,yd,zd) = ros(x+dt*xc,y+dt*yc,z+dt*zc,c)
        x = x + dt/6*(xa + 2*xb + 2*xc + xd)
        y = y + dt/6*(ya + 2*yb + 2*yc + yd)
        z = z + dt/6*(za + 2*zb + 2*zc + zd)
        yield x
```

```
#Função que calcula os máximos locais:
def maiores_pontos (x0,y0,z0,c,n,dt,nint):
#Calcula os n pontos chamando a função rk4
    X = list(rk4_x(x0,y0,z0,c,n,dt))
#Seleciona só os últimos nint pontos (para descartar região transiente)
    X = X[-nint:]
    x_points =[]
#Compara para cada ponto do vetor os pontos vizinhos para saber se é um máximo local
    for i in range(0,nint-1):
        if(X[i]>X[i-1] and X[i]>X[i+1]):
            #se for, salva no vetor x_points
            x_points.append(X[i])
#Elimina pontos repetidos no caso de órbitas periódicas com períodos visíveis
    for i in range(len(x_points)-1):
        if (abs(x_points[0]-x_points[i+1])<0.04):
            x_points = x_points[:i+1]
            return x_points
    for i in range(len(x_points)-2):
        if (abs(x_points[1]-x_points[i+2])<0.04):
            x_points = x_points[:i+2]
            return x_points
#Reduz o tamanho do número de pontos que será plotado nos casos periódicos para 30 pontos
    x_points = x_points[:30]
    return x_points

#Parâmetros e simulação para 1<c<28.5
dt = 0.05
x_plot=[]
C=[]
n = 100000
nint = 6000
for i in range(550):
    x0 = rd.randint(-10,10)
    y0 = rd.randint(-10,10)
    z0 = rd.randint(-1,1)
    c = 1 + i*0.05
    xm = [*maiores_pontos(x0,y0,z0,c,n,dt,nint)]
    x_plot.extend(xm)
    for i in range(0,len(xm)):
        C.append(c)


#Parâmetros e simulação para 28.5<c<46
dt = 0.01
nint = 12000
n = 100000
for i in range(875):
    x0 = rd.randint(-10,10)
    y0 = rd.randint(-10,10)
    z0 = rd.randint(-1,1)
    c = 28.5 + i*0.02
    xm = [*maiores_pontos(x0,y0,z0,c,n,dt,nint)]
    x_plot.extend(xm)
    for i in range(0,len(xm)):
        C.append(c)
```

```
# Função que gera o gráfico
plt.xlim(1,46)
plt.ylim(0,70)
plt.xlabel('c')
plt.ylabel('x')
plt.plot(C, x_plot, 'k,')
plt.title('Diagrama de Bifurcação para as Equações de Rossler')
```

Então, com esse código geramos a seguinte figura:



É importante notar que a qualidade da figura foi altamente sensível ao tamanho do passo de integração dt, variável que por sua vez deve ser diferente para que os cálculos nas diferentes regiões de c não explodam. O código foi escrito de maneira a permitir essa regulação em diferentes regiões de c. Há alguns ruídos (pontos em regiões que não deveriam existir nenhum ponto), tentei reduzí-los desconsiderando os primeiros 40000 pontos da simulação (para garantir que a simulação convergiu para o atrator), e refinando os critérios de seleção de órbitas periódicas e os valoers dt.

2 Circuito de Chua

Com base na seção 9.4 do livro "CHAOS: An Introduction to Dynamical Systems, Alligood, Sauer and Yorke, Springer, 1996", estudaremos o circuito de Chua, que é utilizado em simulações numéricas e também em experimentos com circuitos reais. O sistema de equações de Chua é:

$$\dot{x} = c_1(y - x - g(x))$$

$$\dot{y} = c_2(x - y + z)$$

$$\dot{z} = -c_3 y$$

com:

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

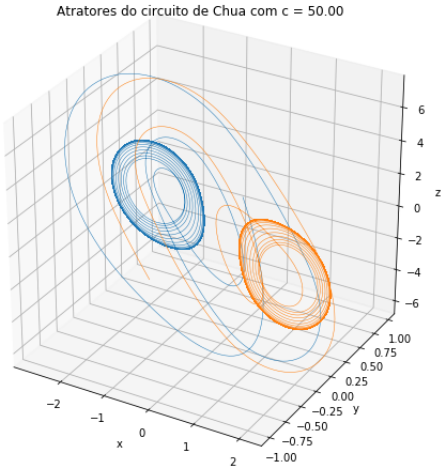
Iremos reproduzir numericamente os atratores da Fig.9.10 do livro, cujo os parâmetros são:$c_1 = 15.6$, $c_2 = 1$, $m_0 = -8/7$, $m_1 = -5/7$. Na figura, o parâmetro c_3 foi sendo modificado.

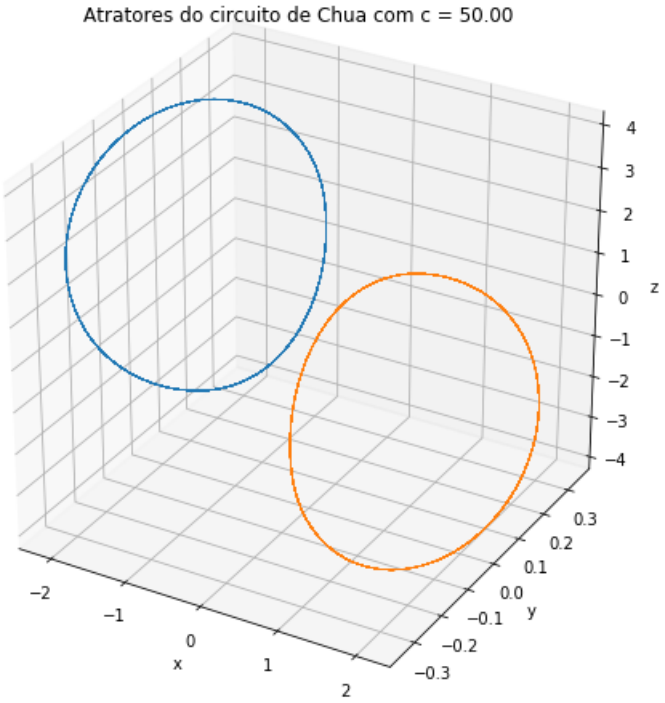
O código para essas simulações foi:

```
#Inicial conditions and parameters that call the simulation:

c3=50 #parameter c3 that is being changed
s=1 #seed for reproductiveness of the simulation
```

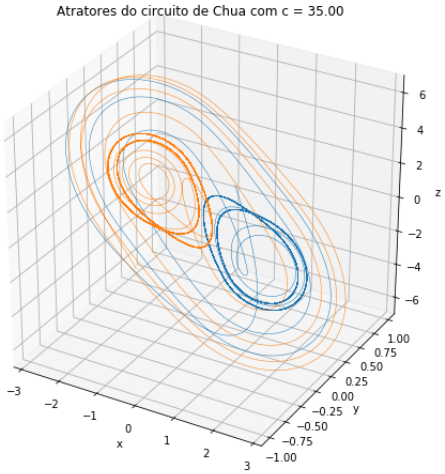
```
n = 50000 #number of time points for simulation
dt = 0.01 #interval between points for numerical calculus
N = 2 #number of initial points
n_inic = 0 # number of points that are disconsidered for the plot
plot3d(c3,n,n_inic,dt,N,s)

#Função que calcula os valores de x,y,z para cada ponto
def chua (x:float,y:float,z:float,c3: float) -> Tuple[float,float,float]:
    if (x<= -1):
        g = -5*x/7 -5/7 + 8/7
    elif (-1<x<=1):
        g = -8*x/7
    else:
        g = -5*x/7 -8/7 + 5/7

    xt = 15.6*(y-x-g)
    yt = 1*(x-y+z)
    zt = -c3*y

    return (xt,yt,zt)

#Função que calcula numericamente cada ponto através do método Runge-Kutta  4
def rk4 (x: float, y: float, z: float, c3: float, n: int,  dt: float) -> Tuple[float,float,float]:
    for i in range (n):
        (xa,ya,za) = chua(x,y,z,c3)
        (xb,yb,zb) = chua(x+dt*0.5*xa,y+dt*0.5*ya,z+dt*0.5*za,c3)
        (xc,yc,zc) = chua(x+dt*0.5*xb,y+dt*0.5*yb,z+dt*0.5*zb,c3)
        (xd,yd,zd) = chua(x+dt*xc,y+dt*yc,z+dt*zc,c3)
        x = x + dt/6*(xa + 2*xb + 2*xc + xd)
        y = y + dt/6*(ya + 2*yb + 2*yc + yd)
        z = z + dt/6*(za + 2*zb + 2*zc + zd)
        yield (x,y,z)

#Função que plota os pontos de n_inic até n
def plot3d (c3, n,n_inic, dt,N,s):
    rd.seed(s)
    fig = plt.figure ( figsize = (8.5,8.5))
    ax = fig.gca ( projection = '3d' )
    for i in range(N):
        x0 = rd.uniform(-1,1)
        y0 = rd.uniform(-1,1)
        z0 = rd.uniform(-1,1)
        (X,Y,Z) = zip(*rk4(x0,y0,z0,c3,n,dt))
        ax.plot ( X[n_inic:], Y[n_inic:], Z[n_inic:], linewidth = 0.5)
    ax.grid ( True )
    ax.set_xlabel ( 'x' )
    ax.set_ylabel ( 'y' )
    ax.set_zlabel ( 'z' )
    ax.set_title ( 'Atratores do circuito de Chua com c = %1.2f' % c3 )
    plt.show ( )
    return
```

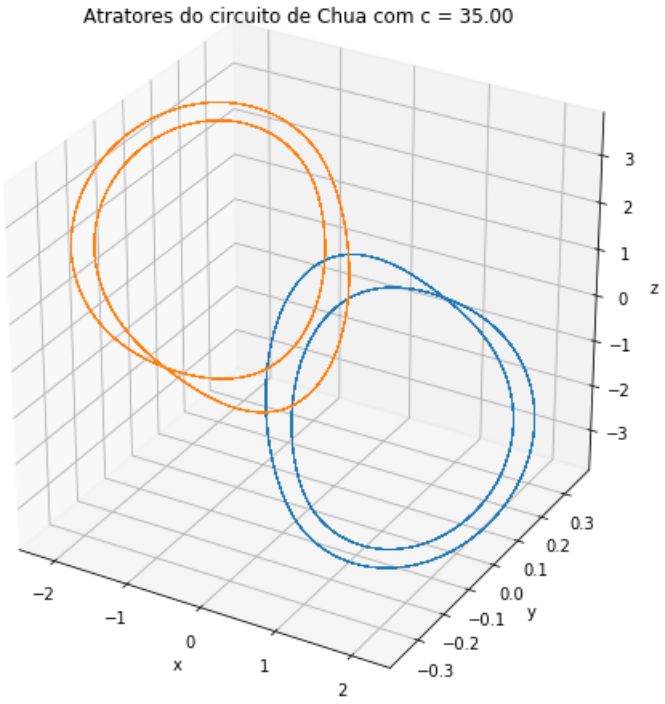
Então, para o primeiro valor de c, $c = 50$, primeiro simulamos $n = 50000$ pontos de duas condições iniciais distintas (com $s = 2$) e observamos que ambas tendem para um atrator de período 1 (Figura da esquerda). Então fazemos um segundo plot em que só plotamos os pontos de a partir de $n = 40000$ e descartamos a região transiente (Figura da direita). Simulamos mais condições iniciais, porém todas convergiam para um dos dois atratores. Então apresentamos aqui somente duas condições iniciais distintas.





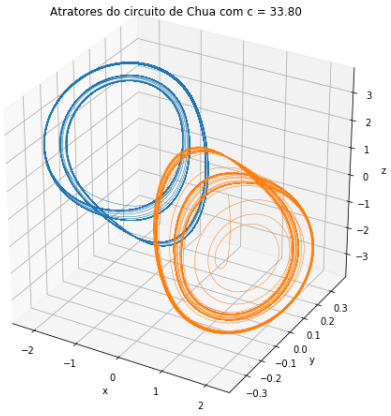
Para $c = 35$, e para todos os outros valores que iremos apresentar, fizemos o mesmo procedimento descrito para gerar a imagem da esquerda e da direita. Nesse caso usamos $n = 5.10^4$ e $n_{inin} = 4.10^4$, com $s = 3$. Simulamos mais condições iniciais, porém todas convergiam para um dos dois atratores de períodos 2. Então apresentamos aqui somente duas condições iniciais distintas.

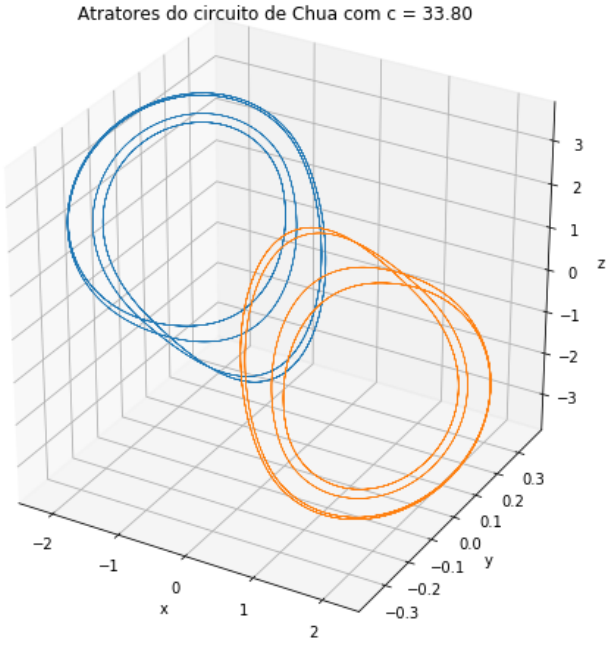




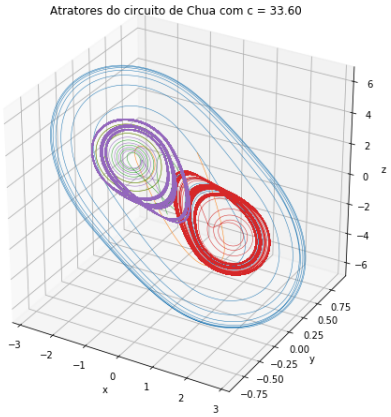
Para $c = 33.8$, usamos $n = 6.10^4$ e $n_{inin} = 4.10^4$, com $s = 3$. Simulamos mais condições

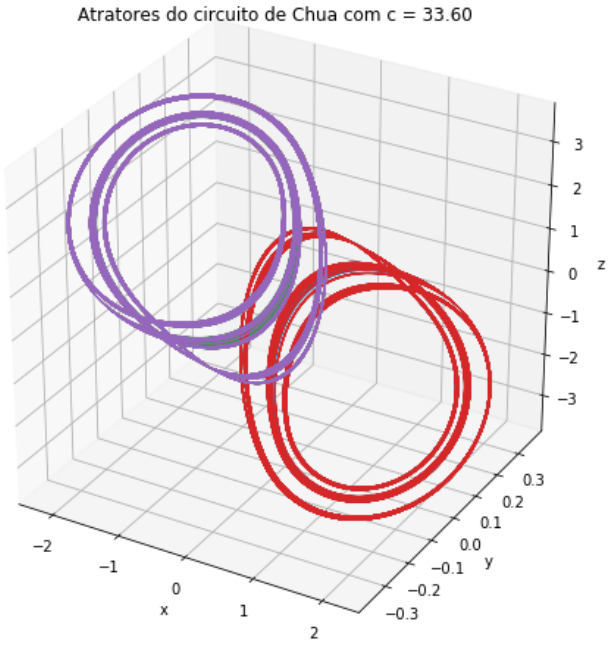
iniciais, porém todas convergiam para um dos dois atratores de período 4. Então apresentamos aqui somente duas condições iniciais distintas.





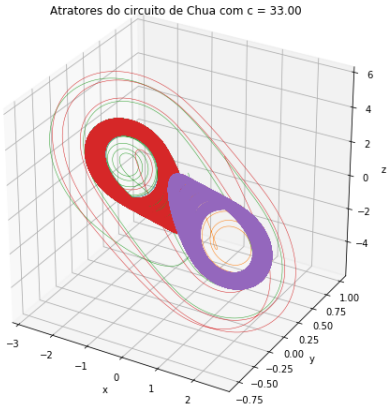
Para $c = 33.6$, usamos $n = 6.10^4$ e $n_{inin} = 4.10^4$, com $s = 2$. Simulamos 5 condições iniciais, todas convergiam para um dos dois atratores caóticos, e no caso em que descartamos a região transiente, só vemos as cores roxa e vinho porque são as cores das últimas simulações e que ficam por cima das outras. A Figura da esquerda deixa claro que há outras cores (outras condições iniciais) também atraídas pelo atrator caótico e que devem estar "por baixo"das trajetórias em cor roxa e vinho.

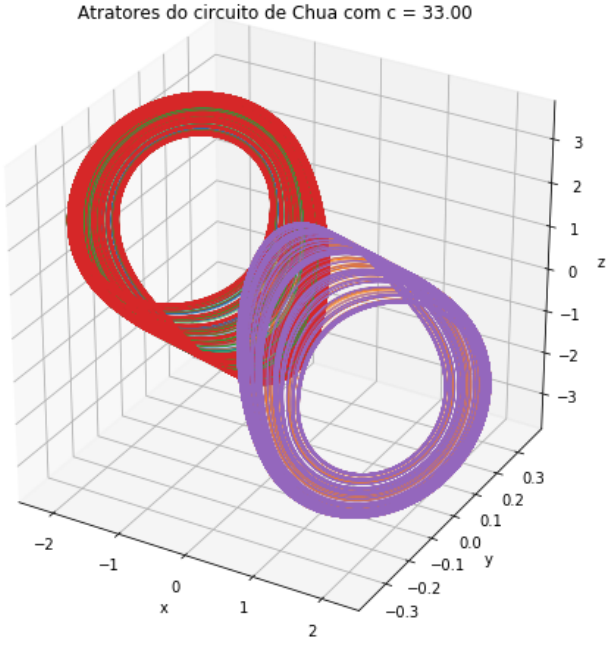




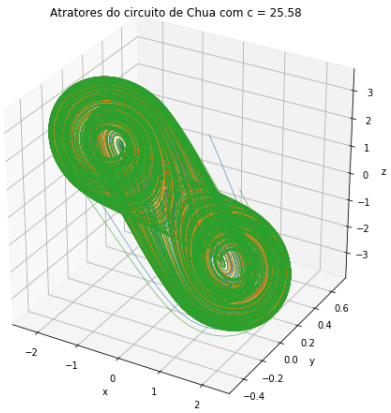
Para $c = 33$, usamos $n = 6.10^4$ e $n_{inin} = 4.10^4$, com $s = 3$. Simulamos 5 condições iniciais, todas

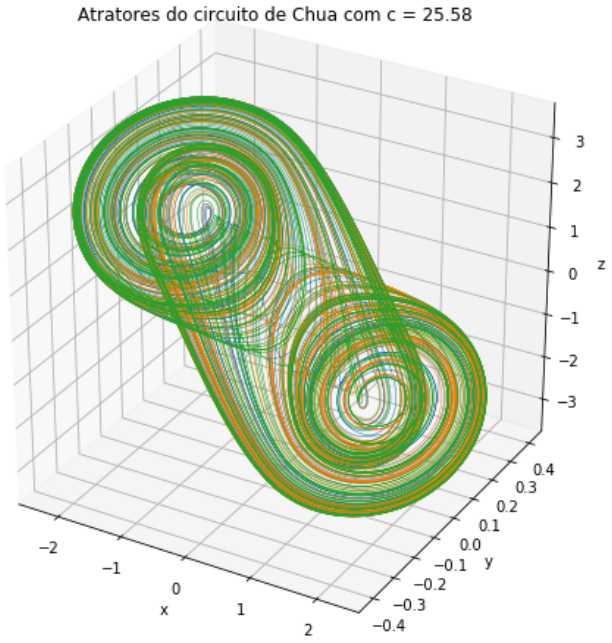
convergiam para um dos dois atratores caóticos, e no caso em que descartamos a região transiente, vemos mais cores do que nos atratores caóticos anteriores porque esses atratores caóticos são mais esparsos. Além disso, vemos que eles estão mais próximos do que no caso 33.8.





Para $c = 25.58$, usamos $n = 6.10^4$ e $n_{inin} = 4.10^4$, com $s = 1$. Simulamos 3 condições iniciais, pois com mais que esse número não era possível distinguir e ver as outras órbitas. Vemos que os dois atratores caóticos se fundem, gerando o que chamamos de "double scroll chaos".





3 Dragon-kings death in nonlinear wave interactions

Estudo de modelo para interações entre ondas, usado em ótica, física de plasma e geofísica, a partir do artigo: "Dragon-kings death in nonlinear wave interactions", Physica A 534, 12296 (2019).

Primeiramente, uma breve introdução aos conceitos. O termo "**Dragon King**"(DK) surge para nomear eventos que são extramemente grandes em seu efeito (daí o "king") e ao memos tempo únicos na sua origem (de onde vem o "dragão"). Esse conceito, criado por Didier Sornette, surge como uma hipótese de que muitas da crises são DK, *podendo ser previsíveis em algum grau*, e não **self-organized critically** [(SOC) is a property of dynamical systems that have a critical point as an attractor (...) is typically observed in slowly driven non-equilibrium systems with many degrees of freedom and strongly nonlinear dynamics.], ou chamados de outra maneira: SOC seriam "**black swans**", que é uma metáfora para eventos que não pode prever que vão existir (assim como os cisnes negros), que vêm de surpresa (cannot bem diagnosed, quantified, predictable). DK eventos: "are generated by / correspond to mechanisms such as positive feedback, tipping points, bifurcations, and phase transitions, that tend to occur in nonlinear and complex systems, and serve to amplify DK events to extreme levels". Para mais informações, ver vídeo: "https://www.youtube.com/watch?v=FlTSbzOvKZI"

O sistema que iremos simmular é dado pelas equações (1-4) do artigo, que com o parâmetro $r = 0$, $v_1 = 1$, $v_2 = v_3 = v$ passam a ser:

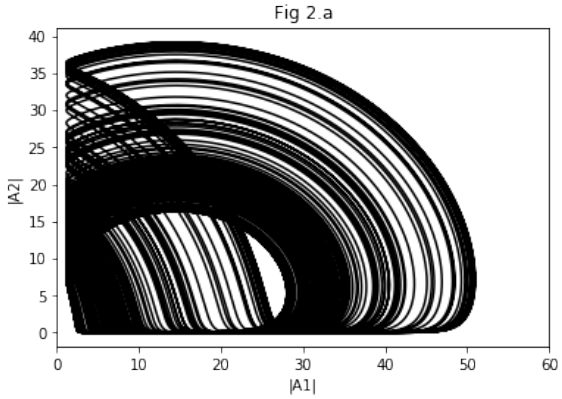
$$\dot{A}_1 = A_1 + A_2 A_3$$
$$\dot{A}_2 = vA_1 - A_1 A_3^*$$
$$\dot{A}_3 = vA_3 - A_1 A_2^* + i\delta_3 A_3$$

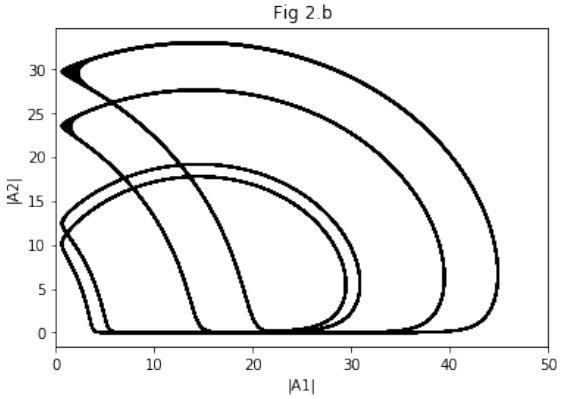
O código usado para fazer as simulações foi:

```
import random as rd
from typing import Tuple
from matplotlib import pyplot as plt
import numpy as np
import cmath

#Função que calcula o sistema de equações para cada ponto
def dragonk (a1:complex,a2:complex,a3:complex,v: float, s3:float) -> Tuple[complex,complex,
complex]:

    i = complex(0,1)
    a1t = a1 + a2*a3
    a2t = v*a2 - a1*np.conj(a3)
    a3t = v*a3 - a1*np.conj(a2) + i*s3*a3

    return (a1t,a2t,a3t)

#Cálculo numérico do ponto através do método RK4
def rk4 (x: complex, y: complex, z: complex, v: float,s3:float, n: int,  dt: float) -> Tuple[
complex,complex,complex]:

    for i in range (n):
        (xa,ya,za) = dragonk(x,y,z,v,s3)
        (xb,yb,zb) = dragonk(x+dt*0.5*xa,y+dt*0.5*ya,z+dt*0.5*za,v,s3)
        (xc,yc,zc) = dragonk(x+dt*0.5*xb,y+dt*0.5*yb,z+dt*0.5*zb,v,s3)
        (xd,yd,zd) = dragonk(x+dt*xc,y+dt*yc,z+dt*zc,v,s3)
        x = x + dt/6*(xa + 2*xb + 2*xc + xd)
        y = y + dt/6*(ya + 2*yb + 2*yc + yd)
        z = z + dt/6*(za + 2*zb + 2*zc + zd)
        yield (x,y,z)

#dt = tamanho do passo de integração
#n = número de pontos (x,y,z) que são calculados
```

```
#Função que plota a figura descartando a região transiente igual a n_init pontos.
def plot2(v,s3,n,n_inic,dt,N,s):
    rd.seed(s)
    for i in range(N):
        a1 = complex(rd.uniform(0,1),rd.uniform(0,1))
        a2 = complex(rd.uniform(0,1),rd.uniform(0,1))
        a3 = complex(rd.uniform(0,1),rd.uniform(0,1))
        (A1,A2,A3) = zip(*rk4(a1,a2,a3,v,s3,n,dt))
        plt.plot(np.absolute(A1[n_inic:]), np.absolute(A2[n_inic:]), 'k-')
    plt.xlabel('|A1|')
    plt.xlim(0,50)
    plt.ylabel('|A2|')
    plt.title('Fig 2.b')
    plt.show()
    return

#Definição dos parâmetros
v=-14.5 #parameter v
s3 = 2.2 #parameter delta_3
s=2 #seed for reproductiveness of the simulation
n = 500000 #number of time points for simulation
dt = 0.0005 #interval between points for numerical calculus
N = 1 #number of initial points
n_inic = 10000 # number of points that are disconsidered for the plot (transiente region)
plot2(v,s3,n,n_inic,dt,N,s)
```

Então, para a figura da esquerda (Figura 2.a do artigo), um atrator caótico, os parâmetros foram: $v = -14.5$, $\delta_3 = 2.2$, $s = 2$, $n = 5.10^5$, $dt = 0.0005$, $N = 1$, $n_inic = 10^4$. Para a figura da direita (Figura 2.b do artigo), um atrator de período 4, os parâmetros foram: $v = -14.5$, $\delta_3 = 1.1$, $s = 2$, $n = 5.10^4$, $dt = 0.005$, $N = 1$, $n_inic = 10^4$.





4 Neuronal Bursting

Esta questão diz respeito a resultados sobre o potencial de ação de neuronios, contidos no artigo A model of neuronal bursting using three coupled first order differential equations que foi publicado no Proc. R. Soc. Lond. B 221, 87-102 (1984).

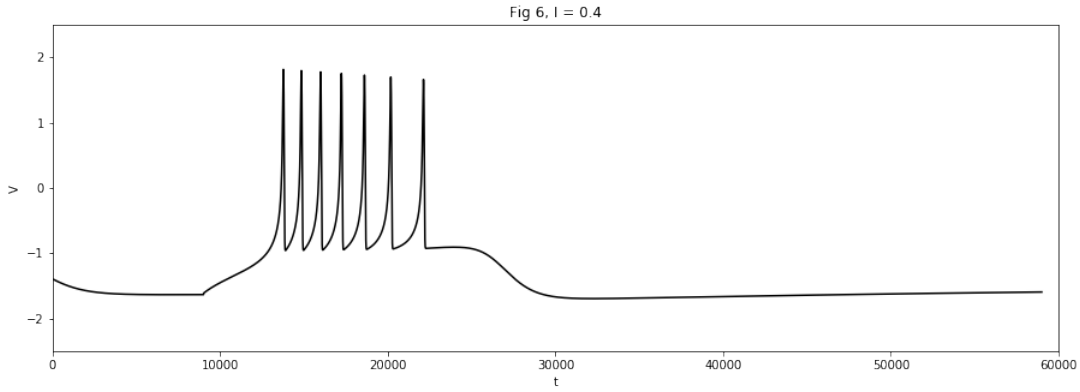
As equações que simulam o comportamento de membrana do neurônio são:

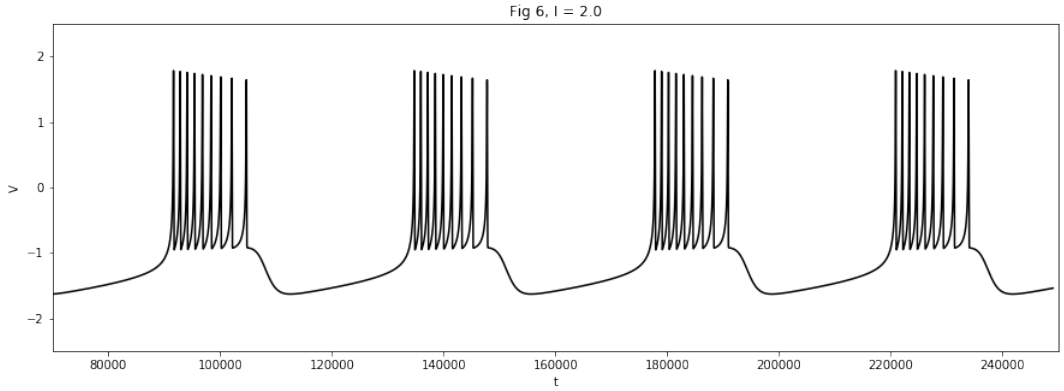
$$\dot{x} = y - ax^3 + bx^2 + I - z$$
$$\dot{y} = c - dx^2 - y$$
$$\dot{z} = r(s(x - x_1) - z)$$

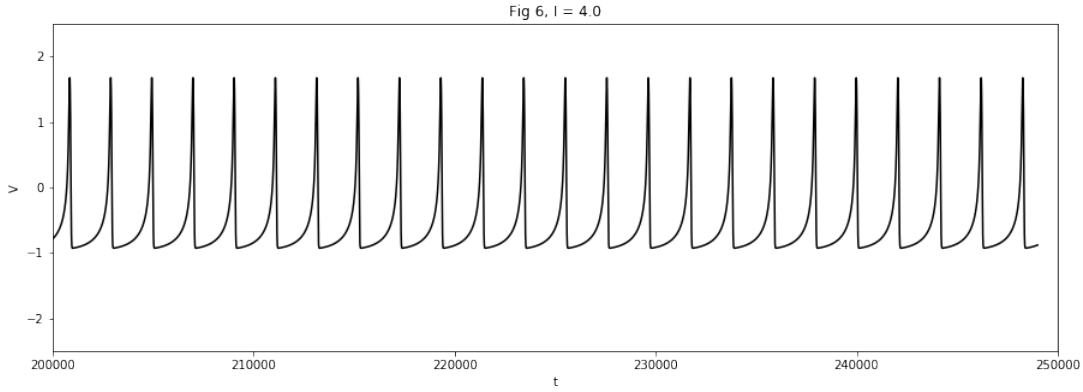
O código usado para simular essas equações, com os parâmetros e condições iniciais: $a = 1$, $b = 3$, $c = 1$, $d = 5$, $s = 4$, $r = 0.001$ e $(x_1, y_1, z_1) = (-1.6, -12, 0)$ foi:

```
from matplotlib import pyplot as plt

#Função que calcula o sistema dinâmico
def neuron (x:tuple,y:tuple,z:tuple,I: float):

    xt = y - 1*x**3 + 3*x**2 + I - z
    yt = 1 - 5*x**2 - y
    zt = 0.001*(4*(x-(-1.6))-z)

    return (xt,yt,zt)

#Função que faz o cálculo numérico para cada ponto, e coloca a corrente entre i_i e i_f
def rk4 (x:float, y: float, z: float, Ic: float, n: int,  dt: float,i_i,i_f):
    for i in range (n):
        if (i_i<i):
            if (i<i_f):
                I = Ic
        else:
            I = 0
        (xa,ya,za) = neuron(x,y,z,I)
        (xb,yb,zb) = neuron(x+dt*0.5*xa,y+dt*0.5*ya,z+dt*0.5*za,I)
        (xc,yc,zc) = neuron(x+dt*0.5*xb,y+dt*0.5*yb,z+dt*0.5*zb,I)
        (xd,yd,zd) = neuron(x+dt*xc,y+dt*yc,z+dt*zc,I)
        x = x + dt/6*(xa + 2*xb + 2*xc + xd)
        y = y + dt/6*(ya + 2*yb + 2*yc + yd)
        z = z + dt/6*(za + 2*zb + 2*zc + zd)
        yield (x,y,z)

#Função que plota a figura
def plot2(I,n,n_inic,dt,i_i,i_f,ni):

    plt.figure(figsize=(15, 5))

    x0 = -1.6
    y0 = -12
    z0 = 0
    (X,Y,Z) = zip(*rk4(x0,y0,z0,I,n,dt,i_i,i_f))
    t = [i for i in range(len(X[n_inic:]))]

    plt.plot(t, X[n_inic:], 'k-')

    plt.xlabel('t')
    plt.ylabel('V')
    plt.ylim(-2.5,2.5)
    plt.xlim(ni,n)
```

```
    plt.title('Fig 6, I = %.1f' %I)
    plt.show()

    return

#Parâmetros
I=0.4 #0.4, 2, 4
n = 60000 #número de pontos da simulação
dt = 0.01 #intervalo entre os pontos para cálculo numérico
n_inic = 1000 # número de pontos que são desconsiderados para o plot
ni = 0 # Ponto em que começamos a plotar a função
i_i = 10000 #Ponto que a corrente começa a ser injetada
i_f = 60000 #Ponto em que a corrente é desligada
plot2(I,n,n_inic,dt,i_i,i_f,ni)
```







5 Modelo SEIR - epidemiologia

Iremos estudar o modelo SEIR - Susceptible exposed infectious recovered, com base no artigo "Transmission dynamics of the COVID-19 outbreak and effectiveness of government interventions: A data-driven analysis", publicado no Journal of Medical Virology 92, 645-659 (2020).

O modelo SEIR faz parte dos modelos matemáticos para epideologia chamados de "warehouse", dentre os quais o modelo SEIR é um exemplo que leva em conta o período de incubação. Ele descreve os seguintes estados possíveis de uma pessoa e o fluxo entre eles: suceptível (S), exposto (E), infeccioso (I) e reabilitado (R). As equações que descrevem essa mudança são:

$$\begin{aligned}\dot{S} &= -\beta\frac{S}{N}I \\ \dot{E} &= \beta\frac{S}{N}I - \omega E \\ \dot{I} &= \omega E - \gamma I \\ \dot{R} &= \gamma I\end{aligned}$$

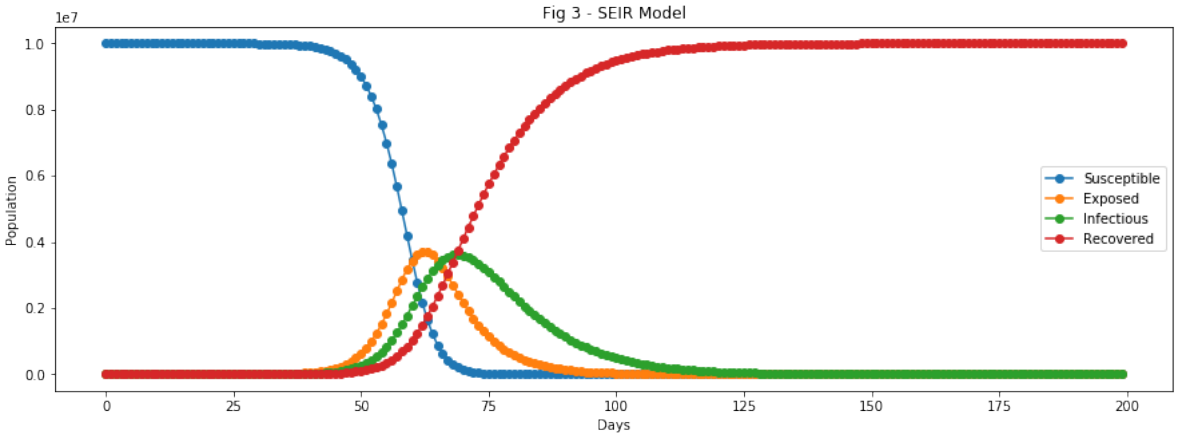
em que os coeficientes são:

$\beta = \beta_0 k$ **taxa de infecção**
β_0 probabilidade de infecção por exposição
k frequência de exposição
$\omega = 1/T_e$ **taxa de migração de latência**
T_e latência média
$\gamma = 1/T_i$ **taxa de migração**
T_i tempo de recupeção médio
N população total no ambiente

Os valores usados para os parâmetros foram: $N = 10^7$, $\beta_0 = 0.1$, $k = 10$, $T_e = 7$, $T_i = 10.25$ O código usado para simular a Figura 3 desse artigo foi feito em Python:

```
from matplotlib import pyplot as plt
import numpy as np

#Função que calcula o sistema dinâmico
def seir (S,E,I,R,prmt):

    (b,w,g,N) = prmt

    St = -b*S*I/N
    Et = b*S*I/N - w*E
    It = w*E - g*I
    Rt = g*I

    return (St,Et,It,Rt)


#Função que calcula numericamente para cada t o valor de cada ponto
def rk4 (S,E,I,R,n,dt,prmt):

    for i in range (n):

        (Sa,Ea,Ia,Ra) = seir(S,E,I,R,prmt)
        (Sb,Eb,Ib,Rb) = seir(S+dt*0.5*Sa,E+dt*0.5*Ea,I+dt*0.5*Ia,R+dt*0.5*Ra,prmt)
        (Sc,Ec,Ic,Rc) = seir(S+dt*0.5*Sb,E+dt*0.5*Eb,I+dt*0.5*Ib,R+dt*0.5*Rb,prmt)
        (Sd,Ed,Id,Rd) = seir(S+dt*Sc,E+dt*Ec,I+dt*Ic,R+dt*0.5*Rc,prmt)
        S = S + dt/6*(Sa + 2*Sb + 2*Sc + Sd)
        E = E + dt/6*(Ea + 2*Eb + 2*Ec + Ed)
```

```
        I = I + dt/6*(Ia + 2*Ib + 2*Ic + Id)
        R = R + dt/6*(Ra + 2*Rb + 2*Rc + Rd)

        yield (S,E,I,R)

#Função que plota a figura
def plot2(S,E,I,R,n,dt,prmt):

    (S,E,I,R) = zip(*rk4(S,E,I,R,n,dt,prmt))
    t = [i for i in range(len(S))]

    plt.figure(figsize=(15, 5))
    plt.plot(t, S, 'o-',label='Susceptible')
    plt.plot(t, E, 'o-',label='Exposed')
    plt.plot(t, I, 'o-',label='Infectious')
    plt.plot(t, R, 'o-',label='Recovered')

    plt.xlabel('Days')
    plt.ylabel('Population')
    plt.title('Fig 3 - SEIR Model')
    plt.legend()
    plt.show()

    return

#Definições dos parâmetros e condições:
b0 = 0.1; k = 10 ; te = 7 ; ti = 10.25; N = 10000000 #valores dos parâmetros iniciais
b = k*b0 ; w = 1/te ; g = 1/ti #Cálculo dos coeficientes

prmt = (b,w,g,N) #Coeficientes

n = 200 #número de pontos
dt = 1  #tamanho de cada passo

#Condições iniciais
S0 = N-1 # número de susceptíveis
E0 = 0   #número de expostos
I0 = 1   #número de contaminados
R0 = 0   #número de recuperados

plot2(S0,E0,I0,R0,n,dt,prmt)
```



6 Fractais

Resumo do vídeo "Fractais caçando a dimensão oculta".

Fractais estão presente na natureza toda, nos vegetais, paisagens e seres vivos. O que Mandelbrot dizia era para quando se olhar uma imagem complexa, não pensar naquilo que vemos, mas sim no que foi necessário para produzir o que vemos: é necessário uma repetição interminável. Autossimiliralidade do padrão dentro do padrão de um objeto. Até então, a matemática se baseava na regularidade, entretanto os padrões da natureza não parecem regulares dessa maneira, mesmo havendo uma ordem por trás do caos aparente, ordem essa captada pela geometria fractal. Por exemplo, a curva de Koch parece ser finita, mas se fazermos essa iteração continuamente, o tamanho dela é infinito. Não é possível medir o comprimento, mas pode-se medir a rugosidade. Para isso, era necessário repensar a noção de dimensionalidade. Quanto mais irregular, maior é a dimensão fractal.

Os computadores acompanharam o desenvolvimento dessa matemática, pois só a partir deles foi possível fazer muitas iterações seguidas.

As ideias da geometria e matemática fractal permitiram olhar para o mundo real, medí-lo e ajudar a fazer matemática sobre ele e entendê-lo melhor. Como caso da contrução de antenas, inovando com antenas fractais, trouxe inúmeros avanços tecnológicos.

No corpo humano as formas fractais aparecem constantemente, nos batimentos cardiácos, no olho e nas veias sanguíneas, podendo ser usados na medicina diagnóstica. Também foi utilizado para ajudar a entender a relação entre o consumo de energia e o tamanho dos corpos dos organismos de todos os seres vivos.

Também pode ser utilizada para compreender e fazer cálculos sobre os processos de troca de oxigêngio e gás carbônico em florestas, com consequêcias para o estudo das alterações climáticas globais.

Assim, a geometria fractal, com regras matemáticas simples, ajuda a compreender e fazer matemática sobre a natureza, que é muito mais semelhante a estruturas fractais e complexas do que a estruturas regulares da matemática anterior a introdução da geometria fractal.