Sistemas Dinâmicos não Lineares
Atividade 4 - 2020

Renata Biaggi Biazzi

24 de abril de 2020

1 Equações de Lorenz

Com base nas seções 9.1 e 9.2 do livro "CHAOS: An Introduction to Dynamical Systems, Alligood, Sauer and Yorke, Springer, 1996", vamos resolver numericamente o sistema de Lorenz de três equações diferenciáveis ordinárias:

$$\begin{aligned}\dot{x} &= \sigma(-x+y)\\ \dot{y} &= -xz + rx - y\\ \dot{z} &= xy - bz\end{aligned}$$

Com $\dot{x} = \frac{\partial x}{\partial t}$, e o mesmo para y e z e definindo a função $f(x,y,z) = (\dot{x},\dot{y},\dot{z})$. Resolveremos numericamente esses sistema calculando para cada variável $x(t), y(t), z(t)$ uma aproximação para sua evolução $(\dot{x},\dot{y},\dot{z})$ no tempo para diferentes valores de t. Ou seja, partindo de um um ponto no tempo t: $u_t = u(t) = (x(t), y(t), z(t))$, queremos calcular a aproximação $u_{t+1} = u(t+\Delta t)$. Faremos isso utilizando o método numérico de Runge-Kutta de 4ª Ordem (RK4), em que cada ponto u_{t+1} é calculado pela seguinte relação:

$$u_{t+1} = u_t + \frac{h}{6}(A + 2B + 2C + B)$$

sendo h a distância entre os pontos no tempo: $h = \Delta t$, e os parâmetros:

$$\begin{aligned}A &= f(u_t)\\ B &= f\left(u_t + \frac{hA}{2}\right)\\ C &= f\left(u_t + \frac{hB}{2}\right)\\ D &= f(u_t + hC)\end{aligned}$$

Iremos resolver esse sistema para os valores fixos: $\sigma = 10$ e $b = 8/3$, para os seguintes valores de r, em que queremos observar os comportamos descritos na Tabela 9.1 do livro CHAOS: An Introduction to Dynamical Systems:

r	Attractor
$[-\infty, 1.00]$	$(0,0,0)$ is an attracting equilibrium
$[1.00, 13.93]$	C_+ and C_- are attracting equilibria; the origin is unstable
$[13.93, 24.06]$	Transient chaos: There are chaotic orbits but no chaotic attractors
$[24.06, 24.74]$	A chaotic attractor coexists with attracting equilibria C_+ and C_-
$[24.74, ?]$	Chaos: Chaotic attractor exists but C_+ and C_- are no longer attracting

Table 9.1 Attractors for the Lorenz system (9.1).
For $\sigma = 10$, $b = 8/3$, a wide range of phenomena occur as r is varied.

Para o cálculo numérico, foi feito um código em Python 3:

```
from typing import Tuple

def lorenz (x:float,y:float,z:float,r: float) -> Tuple[float,float,float]:
    xt = -10*x + 10*y
    yt = -x*z + r*x - y
    zt = x*y - 8/3*z

    return (xt,yt,zt)

def rk4 (x: float, y: float, z: float, r: float, n: int,  dt: float) -> Tuple[float,float,float]:
    for i in range (n):
        (xa,ya,za) = lorenz(x,y,z,r)
        (xb,yb,zb) = lorenz(x+dt*0.5*xa,y+dt*0.5*ya,z+dt*0.5*za,r)
        (xc,yc,zc) = lorenz(x+dt*0.5*xb,y+dt*0.5*yb,z+dt*0.5*zb,r)
        (xd,yd,zd) = lorenz(x+dt*xc,y+dt*yc,z+dt*zc,r)
        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)
```

Então, para cada simulação com um parâmetro r diferente, foi necessário definir r, os valores iniciais x_0, y_0 e z_0, assim como o intervalo h entre cada tempo de integração e o número de pontos n representando o total de pontos no tempo em que a função foi calculada.

Para os plots, basta chamar as funções $plot2d(r, n, dt, N, s)$ para gerar o gráfico de xz com N pontos iniciais uniformemente aleatórios entre um intervalo para as variáveis, ou $plot3d(r, n, dt, N, s)$ para gerar um gráfico xyz com N pontos iniciais feitos da mesma maneira. O código dessas funções está disponível abaixo:

Plots 2d

```
r =
n =  #number of time points
dt = 0.005 #interval between points
N =  #number of initial points
s =  #random seed

def plot2d(r,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(-10,10)
        (X,Y,Z) = zip(*rk4(x0,y0,z0,r,n,dt))
        plt.plot(X, Z, ',')
    plt.xlim((-5,5))
    plt.ylim((-5,5))
    plt.xlabel('x')
    plt.ylabel('z')
    plt.title('Atrator de Lorenz')

    plt.show()
    return
```

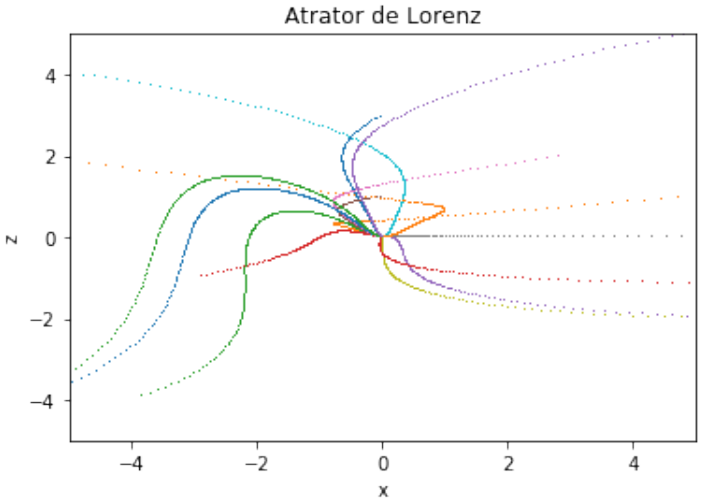
Plots 3d

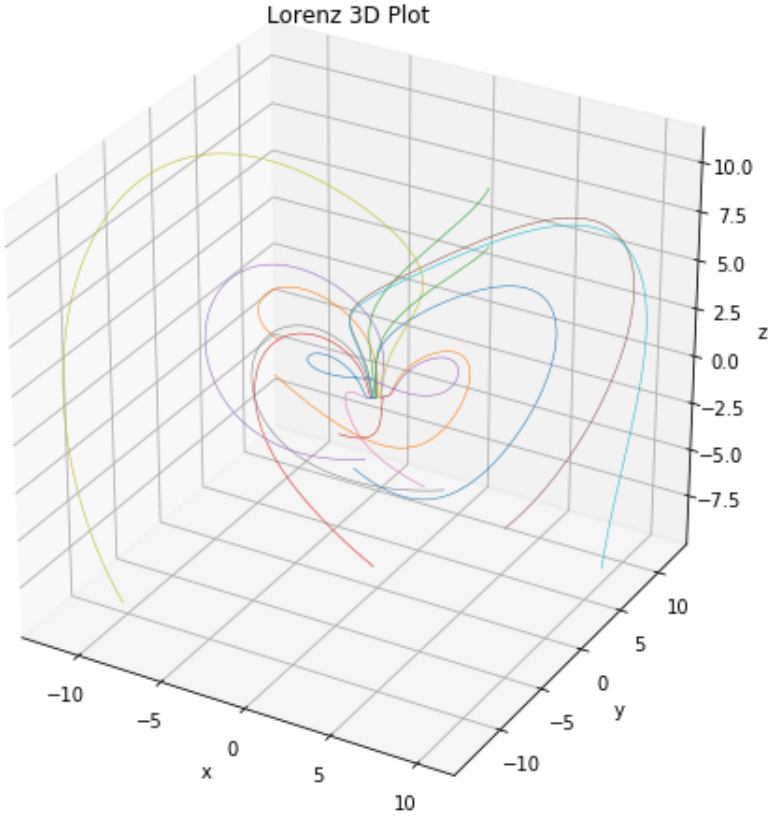
```
r =
n =  #number of time points
```

```
dt = 0.005 #interval between points
N =  #number of initial points
s=  #random seed

def plot3d (r, 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(-10,10)
    (X,Y,Z) = zip(*rk4(x0,y0,z0,r,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 ( 'Lorenz 3D Plot' )

plt.show ( )
return
```

1.a

Para o primeiro regime $-\infty \leq r \leq 1.00$, foi escolhido $r = 0$ e foram feitas simulações com valores iniciais aleatórios para $x, y, z \, \epsilon \, [-10, 10]$, $h = 0.005$, $n = 10^4$, $s = 1$ e $N = 15$.

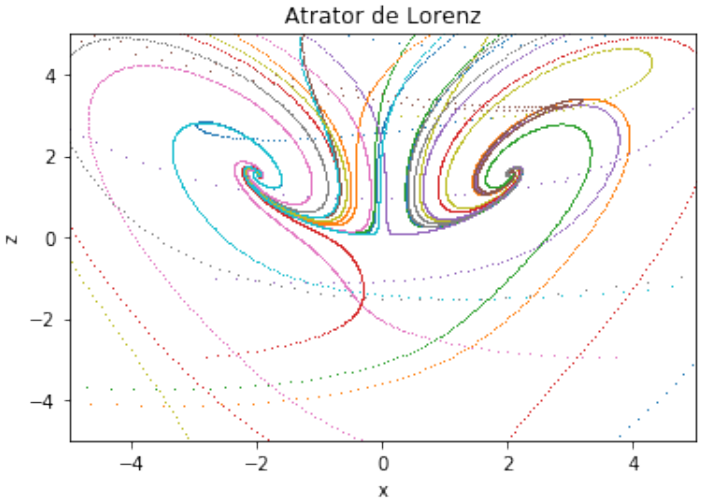


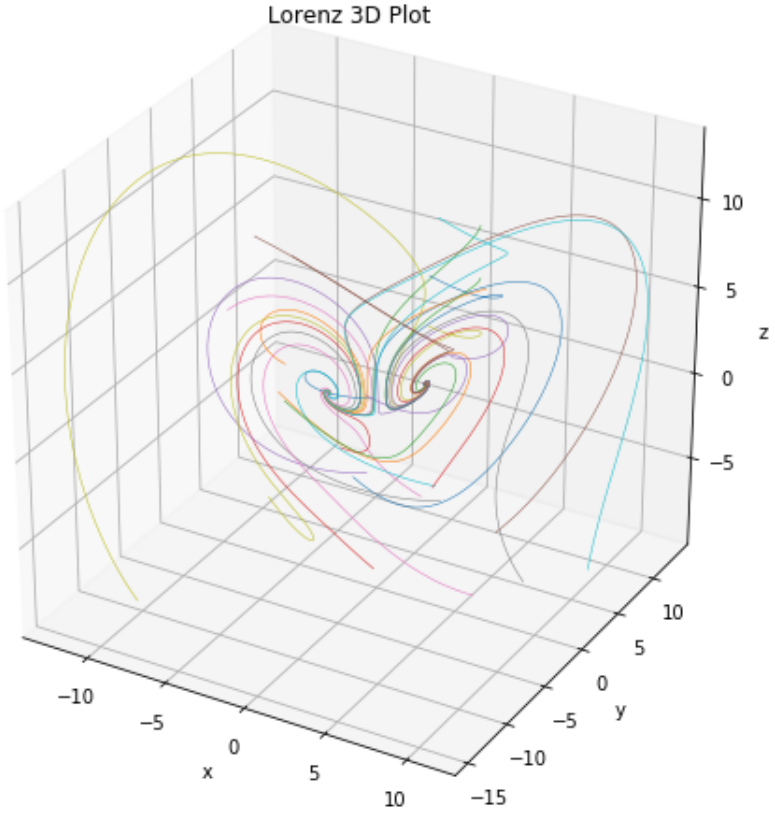


Podemos observar em ambos os gráficos que o ponto $(0, 0, 0)$ é o único **ponto estável atrativo**, pois todas as órbitas são atraídas para ele.

1.b

Para o regime $1 < r < 13.93$, foi escolhido $r = 2.5$ e foram feitas simulações com valores iniciais aleatórios para $x, y, z \, \epsilon \, [-10, 10]$, $h = 0.005$, $n = 10^4$, $s = 1$ e $N = 30$.





Como descrito no texto, nesse regime surgem dois novos pontos críticos estáveis:

$$C_+ = (\sqrt{b(r-1)}, \sqrt{b(r-1)}, r-1)$$
$$C_- = (-\sqrt{b(r-1)}, -\sqrt{b(r-1)}, r-1)$$

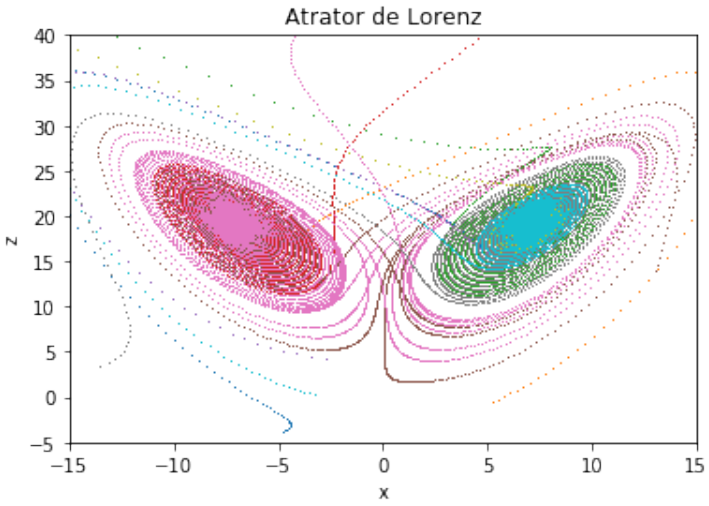
Para o nosso caso, $r = 2.5$ e $b = 8/3$, temos que:

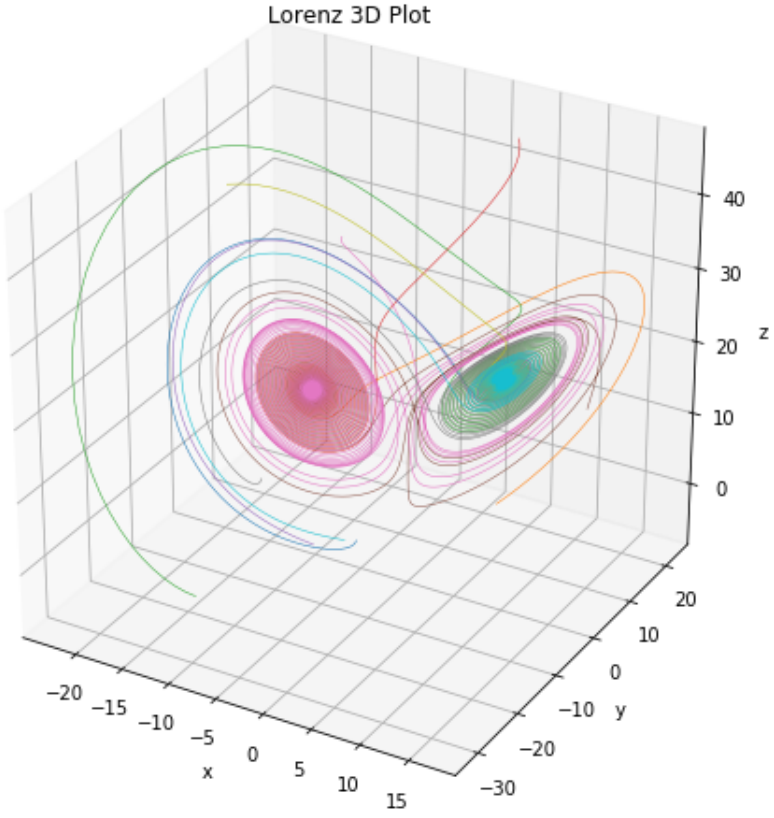
$$C_+ = (2, 2, 1.5)$$
$$C_- = (-2, -2, 1.5)$$

O ponto $(0, 0, 0)$ deixa de ser estável, e se torna um ponto fixo intável. Podemos observar em ambos os gráficos que o ponto $(0, 0, 0)$ é um ponto fixo instável (uma fonte), pois todas as órbitas que chegam perto dele em determinadas distâncias são afastadas, comportamento típico de "sources". Já os pontos $C_+ = (2, 2, 1.5)$ e $C_- = (-2, -2, 1.5)$ são pontos fixos estáveis, como previsto pelas equações, então são **pontos de equilíbrio atrativos**, atraindo todas as órbitas para si, durante o regime $r < 24\frac{14}{19} \approx 24.74$.

1.c

Para o regime $13.93 < r < 24.06$, foi escolhido $r = 20$ e foram feitas simulações com valores iniciais aleatórios para $x, y \, \epsilon \, [-20, 20]$ e $z \, \epsilon \, [-10, 50]$, $h = 0.005$, $n = 10^5$, $s = 4$ e $N = 10$.





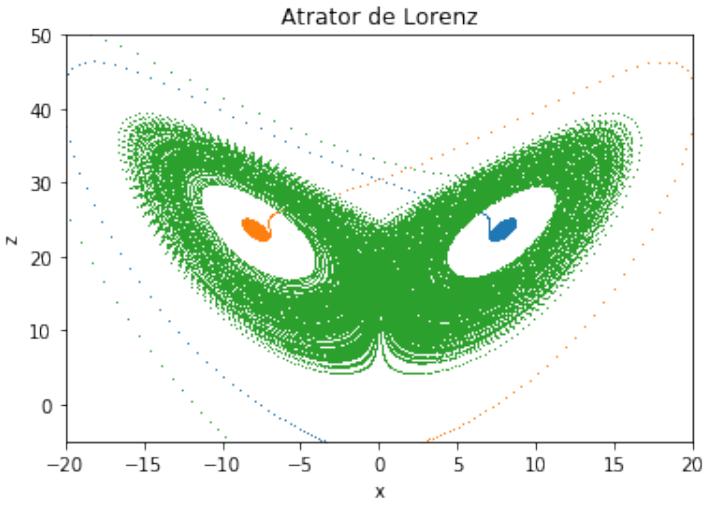
Os pontos críticos C_+ e C_- ainda são estáveis nesse regime: Para o nosso caso, $r = 20$ e $b = 8/3$, temos que:

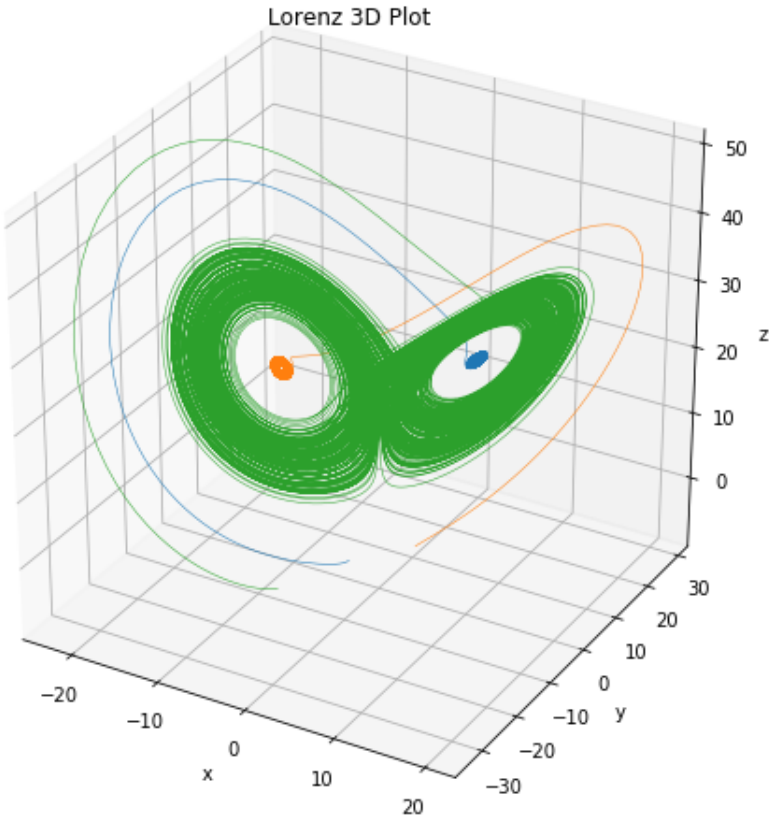
$$C_+ \approx (7.1, 7.1, 19)$$
$$C_- \approx (-7.1, -7.1, 19)$$

O ponto $(0, 0, 0)$ continua sendo instável e, como vemos nos gráficos há um período **transiente caótico** em que as órbitas oscilam ao redor dos pontos C_+ e C_-, se aproximando assintópticamente deles com o tempo. Esse é um período de caos transiente, até as órbitas tenderem para C_+ e C_-.

1.d

Para o regime $24.06 < r < 24.74$, foi escolhido $r = 24.5$ e foram feitas simulações com valores iniciais aleatórios para $x, y, z \, \epsilon \, [-10, 10]$, $h = 0.005$, $n = 5.10^4$, $s = 4$ e $N = 3$.





Nesse regime, como descrito no texto e mostrado nas figuras, um atrator caótico e os dois pontos estáveis C_+ e C_- coexistem. Para o nosso caso, $r = 24.5$ e $b = 8/3$, temos que:

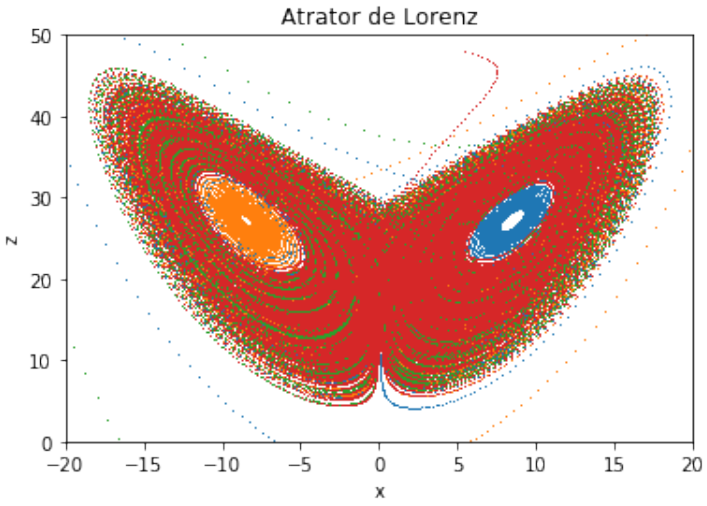
$$C_+ = (\sqrt{8(23.5)/3}, \sqrt{8(23.5)/3}, r - 1) \approx (7.9, 7.9, 23.5)$$
$$C_- = (\sqrt{8(23.5)/3}, \sqrt{8(23.5)/3}, r - 1) \approx (-7.9, -7.9, 23.5)$$

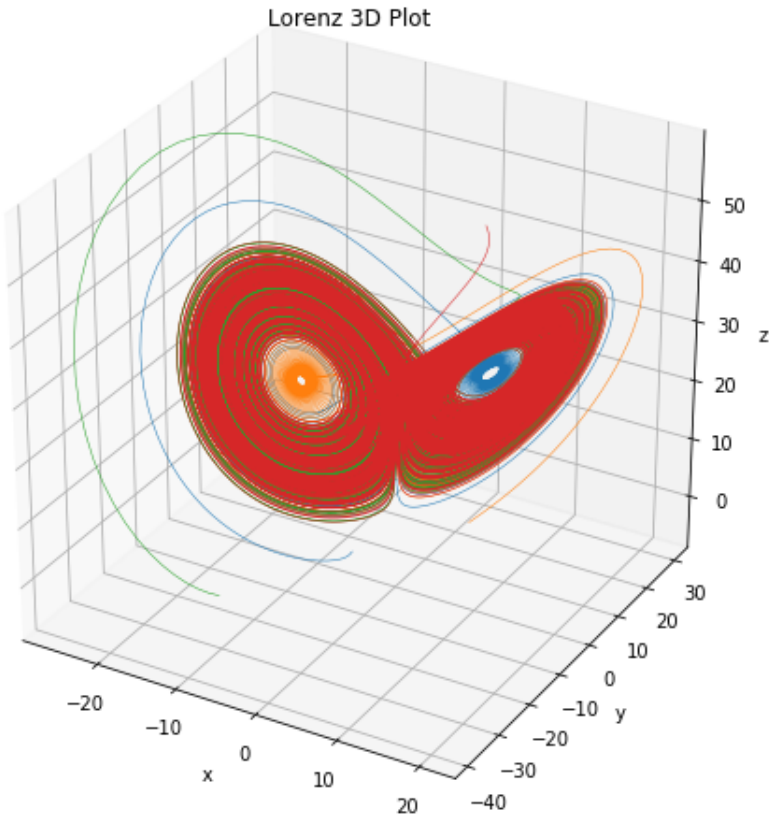
Podemos observar em ambos os gráficos que, a depender das condições iniciais, as trajetórias podem ser atraídas para os pontos $C_+ \approx (7.9, 7.9, 23.5$ e $C_- \approx (-7.9, -7.9, 23.5$, que são pontos fi-

xos estáveis, atraindo todas as órbitas para si, ou também podem ficar presas no atrator (trajetória de cor verde). De forma que **coexistem uma região atratora e dois pontos fixos estáveis**. Nesse caso, não foram plotados mais pontos iniciais do que $N = 3$ pois eles teriam um desses dois comportamentos coexistentes, de forma que mais pontos deixariam os gráficos mais densos sem acrecentar nenhuma informação e dificultando a vizualização desses dois comportamentos coexistentes.

1.e

Para o regime $r > 24.74$, foi escolhido $r = 28$ e foram feitas simulações com valores iniciais aleatórios para $x, y \, \epsilon \, [-20, 20]$ e $z \, \epsilon \, [-10, 50]$, $h = 0.005$, $n = 5.10^4$, $s = 4$ e $N = 4$.



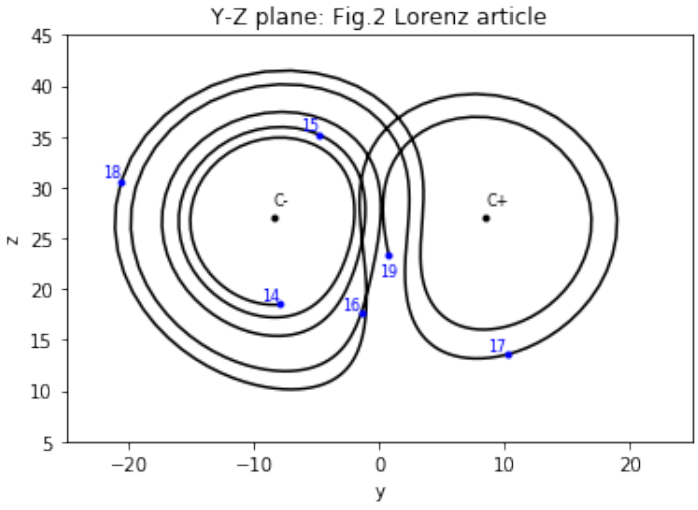


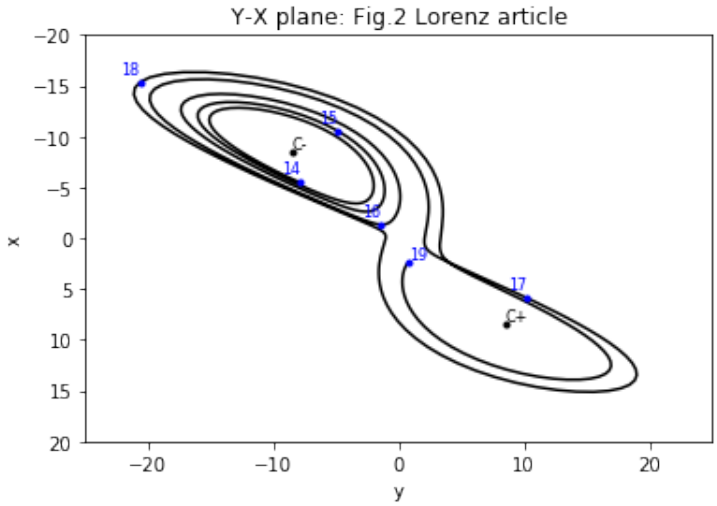
Podemos observar em ambos os gráficos que os pontos C_+ e C_- deixam de ser atratores, pois apesar de oscilar continuamente ao seu redor nenhuma órbita tende para esses valores. Ou seja, esses pontos passam a ser pontos instáveis. Nesse regime, temos um **atrator caótico**, ou seja, uma região caótica que independente das condições iniciais irá atrair todas as órbitas para uma certa região. Os valores das órbitas continuam tendo um comportamento caótico, altamente dependentes das condições iniciais, mas são limitadas a região do atrator.

2 Equações de Lorenz

Baseado no artigo "Deterministic Nonperiodic Flow", de Edward N. Lorenz,publicado no Journal of Atmospheric Sciences 20, 130-141 (1963), iremos reproduzir numericamente a resolução do seu sistema de equações, descritas na questão anterior.

a) Com base no que é descrito no artigo, utilizando $\sigma = 10$, $b = 8/3$ e $r = 28$, partindo dos pontos iniciais: $(x, y, z) = (0, 1, 0)$, em que começamos a plotar os pontos a partir de $n = 1400$ até $n = 1900$. Seguem as imagens geradas por meio de simulações numéricas reproduzindo a Figura 2 do artigo. Os pontos azuis nas figuras com um número x ao lado significam que aquele ponto corresponde ao ponto $n = 100 * x$ simulado. Os pontos C_+ e C_- são os pontos estáveis no regime $r < 24.74$. Como estamos no regime $r = 28$, esses pontos passam a ser instáveis e só há o atrator caótico, simulado nas figuras seguintes:





O código usado para gerar essas figuras é semelhante ao usado na questão anterior. A função lorenz(x,y,z,r) é a mesma, e a função rk4 foi modificada só no final:

```
    def q2_rk4 (x: float, y: float, z: float, r: float, n: int,  dt: float) -> Tuple[float,float,f
    for i in range (n):
        (xa,ya,za) = lorenz(x,y,z,r)
        (xb,yb,zb) = lorenz(x+dt*0.5*xa,y+dt*0.5*ya,z+dt*0.5*za,r)
        (xc,yc,zc) = lorenz(x+dt*0.5*xb,y+dt*0.5*yb,z+dt*0.5*zb,r)
        (xd,yd,zd) = lorenz(x+dt*xc,y+dt*yc,z+dt*zc,r)
        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)
    return (x,y,z)
```

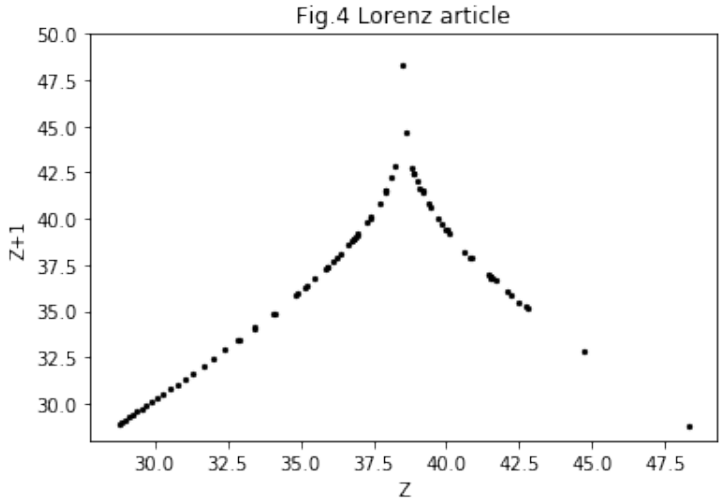
O Plot YZ foi feito com o seguinte código:

```
def YZ_q2_plot2d(x0,y0,z0,r,Ni,n,dt):
    x0,y0,z0 = q2_rk4(x0,y0,z0,r,Ni,dt)
    (X,Y,Z) = zip(*rk4(x0,y0,z0,r,n,dt))
    plt.plot(Y, Z, 'k-')
    text = ['14','15','16','17','18']
    for i in range (5):
        j = i
        i = 100*i
        plt.plot(Y[i], Z[i], 'b.')
        plt.text(Y[i], Z[i], text[j],verticalalignment='bottom', horizontalalignment='right', colo
    yc = (8*(r-1)/3)**(1/2)
    zc = r-1
    plt.plot(Y[499], Z[499], 'b.')
    plt.text(0, 21, '19',verticalalignment='bottom', color='blue',fontsize=8)
    plt.plot(yc, zc, 'k.')
    plt.text(yc, zc+1, 'C+',verticalalignment='bottom', color='black',fontsize=8)
    plt.plot(-yc, zc, 'k.')
    plt.text(-yc, zc+1, 'C-',verticalalignment='bottom', color='black',fontsize=8)
    plt.xlim((-25,25))
    plt.ylim((5,45))
    plt.xlabel('y')
    plt.ylabel('z')
    plt.title('Y-Z plane: Fig.2 Lorenz article')

    plt.savefig ( 'q2_fig21.png' )

    plt.show()

    return

x0 = 0 #initial points
y0 = 1
z0 = 0
r = 28 #parameter
Ni = 1400 #Iteration to start ploting
n = 500   #Nf - Ni = 1900 - 1400 = 500 -> number of points to plot
dt = 0.01 #time step

YZ_q2_plot2d(x0,y0,z0,r,Ni,n,dt)
```

O plot YX foi feito com uma função semelhante a anterior, substituindo Z por X e mudando as posições zc por xc.

b)

Essa figura foi feita selecionando em uma simulação de $N = 6000$ pontos, partindo das condições iniciais descritas acima, os pontos em que Z assume um valor máximo localmente. Assim, no eixo Z a posição dos pontos indica esses valores, e no eixo Z+1 os pontos indicam o valor do máximo seguinte ao máximo correspondente em Z.



O código usado para gerar essa figura está disponível abaixo:

```
x0 = 0 #initial points:
y0 = 1
z0 = 0
r = 28 #parameter
n = 6000
dt = 0.01 #time step

(X,Y,Z) = list(zip(*rk4(x0,y0,z0,r,n,dt)) ) #calcula todos os pontos até n=6000
plotx = []
ploty = []
j=0
k = 0
for i in range(n):
    if (i<n-2 and Z[i+1]>Z[i] and Z[i+1]>Z[i+2]): # salva os máximos no vetor  Z+1 (M+1)
        j = j+1
        ploty.append(Z[i+1])

plotx.append(38.5) #condição inicial para M0 usada no artigo

while (k<j-1):   #atribue ao vetor M os valores de M+1 menos uma posição
    plotx.append(ploty[k])
    k = k+1

plt.scatter(plotx,ploty, s=5,c='k', marker='o')
plt.ylim(28,50)
plt.xlabel('Z')
plt.ylabel('Z+1')
plt.title('Fig.4 Lorenz article')

plt.savefig ( 'q2_fig4.png' )

plt.show()
```

Essa figura é uma tentativa de tentar encontrar uma relação entre os pontos, no caso de um pes-

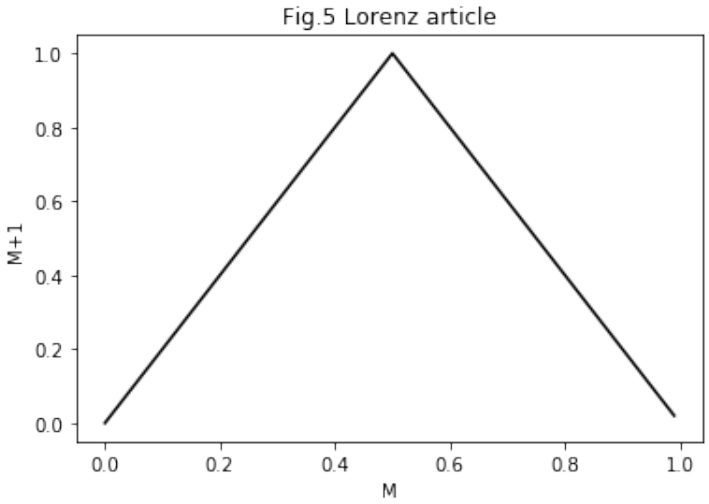
quisador que não conhecesse as equações que estão gerando o comportamento das variáveis X,Y e Z.

c)

Nessa figura, é feito um modelo na tentativa de descrever o comportamento observado para Z e Z+1 que foi simulado na questão anterior. O seguinte modelo é descrito, sendo M_n uma sequência de números entre 0 e 1.

$$\begin{aligned} M_{n+1} &= 2M_n & M_n &< 1/2 \\ M_{n+1} &= indefinido & M_n &= 1/2 \\ M_{n+1} &= 2 - 2M_n & M_n &> 1/2 \end{aligned}$$

Simulando o comportamento descrito pelo modelo, obtemos a Figura 5 do artigo:



O código usado para gerar essa figura foi:

```
import numpy as np
m = [0 + i/100 for i in range(100)] #cria vetor com os valores de M
m_1=[0 for i in range(100)] #cria vetor com as posições para M+1
for i in range (100):
    if (i<50):
        m_1[i] = 2*m[i] #atribui os valores de M+1 em função de M1
    if (i>50):
        m_1[i] = 2 -2*m[i]

plt.plot(m,m_1,'k-')
plt.xlabel('M')
plt.ylabel('M+1')
plt.title('Fig.5 Lorenz article')

plt.savefig ( 'q2_fig5.png' )

plt.show()
```