LICENCIATURA EN FÍSICA

DEPARTAMENTO DE FÍSICA

FÍSICA COMPUTACIONAL 1

Evaluacion 2

Alumna:

Brambilla Zamorano Fátima Fernanda

Fecha:
26/04/18

1 Reporte de Actividad

1.1 Objetivo 1

Esta evaluación trato de recrear el *Atráctor de Lorenz*, el cual es un ejemplo de *caos dinámico*. Para esto, se nos proporciono un código para Python, el cual tomaríamos para recrearlo en nuestro notebook en Jupyter Lab, al tiempo que sería analizado para ver como funcionaba. Una vez hecho esto, debíamos cambiarle los parámetros *sigma, rho y beta*, para ver cual era el comportamiento esperado del atráctor con diferentes parámetros dentro de sus condiciones iniciales. A continuación se mostraran algunas partes del código original, y las gráficas que este mostraba una vez que era copilado.

```
%matplotlib inline
import numpy as np, matplotlib.pyplot as plt, matplotlib.font_manager as fm, os
from scipy.integrate import odeint
from mpl_toolkits.mplot3d.axes3d import Axes3D
```

En está primera celda que se muestra arriba, simplemente se han importado diferentes bibliotecas al programa, las cuales serán necesarias para resolver el sistema de Lorenz. En la siguiente celda que se muestra, se han definido parámetros con los que el sistema trabajara, siendo estos los mencionados al principio del documento. Además, se define un tiempo de inicio y final, así como un número de puntos en el lapso de tiempo definido, que son donde se resolverá el sistema.

```
# define the initial system state (aka x, y, z positions in space)
initial_state = [0.1, 0, 0]

# define the system parameters sigma, rho, and beta
sigma = 10.
rho   = 28.
beta  = 8./3.

# define the time points to solve for, evenly spaced between the start and end times
start_time = 0
end_time = 100
time_points = np.linspace(start_time, end_time, end_time*100)
```

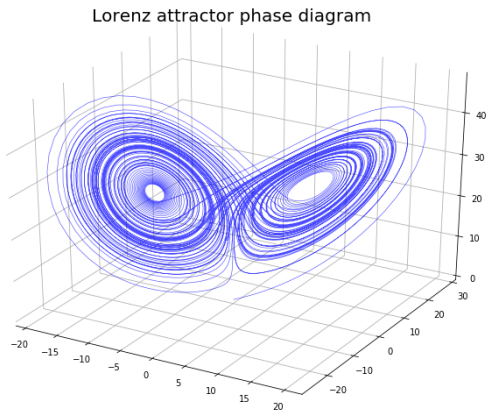
En las siguientes fracciones de código, se muestra como se definió el sistema de ecuaciones ordinarias que modelan el atráctor de Lorenz, también se muestra en la segunda parte, la función odeint(), la cual se usa para resolver sistemas de ecuaciones diferenciales ordinarias. En la segunda imagen se puede ver la manera en que se procedió para "armar" la gráfica que muestra el comportamiento del atráctor. Lo que se puede mencionar de estas dos partes del código, es que en el caso de la celda que se muestra en primer lugar, en ella se utilizan elementos que ya habíamos prácticado antes en el curso, mientras que en la celda que se muestra en la segunda imagen, es que se gráfica en tres dimensiones, algo que no habíamos hecho hasta el momento. Por ello es importante que analicemos bien la manera en que se gráfico este comportamiento, pues es posible que más adelante en nuestro paso por la carrera, necesitemos hacer gráficasd e ese mismo estilo.

```
# define the lorenz system
# x, y, and z make up the system state, t is time, and sigma, rho, beta are the system parameters
def lorenz_system(current_state, t):

    # positions of x, y, z in space at the current time point
    x, y, z = current_state

    # define the 3 ordinary differential equations known as the lorenz equations
    dx_dt = sigma * (y - x)
    dy_dt = x * (rho - z) - y
    dz_dt = x * y - beta * z

    # return a list of the equations that describe the system
    return [dx_dt, dy_dt, dz_dt]
```

```
# use odeint() to solve a system of ordinary differential equations
# the arguments are:
# 1, a function - computes the derivatives
# 2, a vector of initial system conditions (aka x, y, z positions in space)
# 3, a sequence of time points to solve for
# returns an array of x, y, and z value arrays for each time point, with the initial values in the first r
ow
xyz = odeint(lorenz_system, initial_state, time_points)

# extract the individual arrays of x, y, and z values from the array of arrays
x = xyz[:, 0]
y = xyz[:, 1]
z = xyz[:, 2]
```

```
# plot the lorenz attractor in three-dimensional phase space
fig = plt.figure(figsize=(12, 9))
ax = fig.gca(projection='3d')
ax.xaxis.set_pane_color((1,1,1,1))
ax.yaxis.set_pane_color((1,1,1,1))
ax.zaxis.set_pane_color((1,1,1,1))
ax.plot(x, y, z, color='g', alpha=0.7, linewidth=0.6)
ax.set_title('Lorenz attractor phase diagram', fontproperties=title_font)

fig.savefig('{}/lorenz-attractor-3d.png'.format(save_folder), dpi=180, bbox_inches='tight')
plt.show()
```

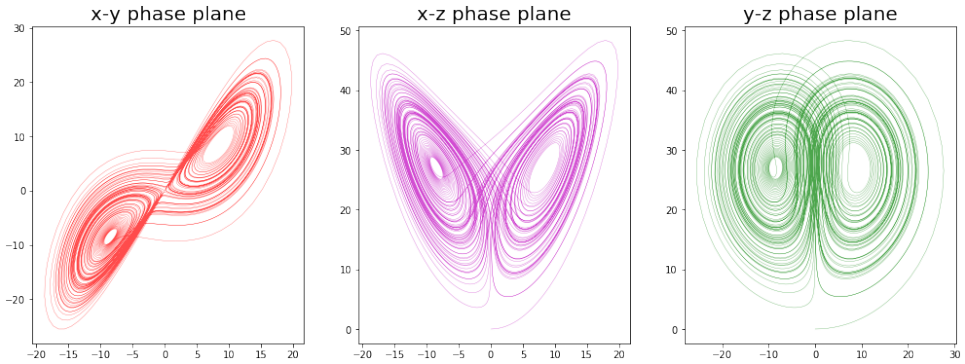
Y la imagen que nos gráfica el código en la última celda mostrada, es la siguiente:



Pero, ver su comportamiento en tres dimensiones no siempre nos deja en claro como es que se mueve, por ello es que a continuación se muestra como se gráficaron los comportamientos para los distintos planos del espacio

```
# now plot two-dimensional cuts of the three-dimensional phase space
fig, ax = plt.subplots(1, 3, sharex=False, sharey=False, figsize=(17, 6))

# plot the x values vs the y values
ax[0].plot(x, y, color='r', alpha=0.7, linewidth=0.3)
ax[0].set_title('x-y phase plane', fontproperties=title_font)

# plot the x values vs the z values
ax[1].plot(x, z, color='m', alpha=0.7, linewidth=0.3)
ax[1].set_title('x-z phase plane', fontproperties=title_font)

# plot the y values vs the z values
ax[2].plot(y, z, color='b', alpha=0.7, linewidth=0.3)
ax[2].set_title('y-z phase plane', fontproperties=title_font)

fig.savefig('{}/lorenz-attractor-phase-plane.png'.format(save_folder), dpi=180, bbox_inches='tight')
plt.show()
```

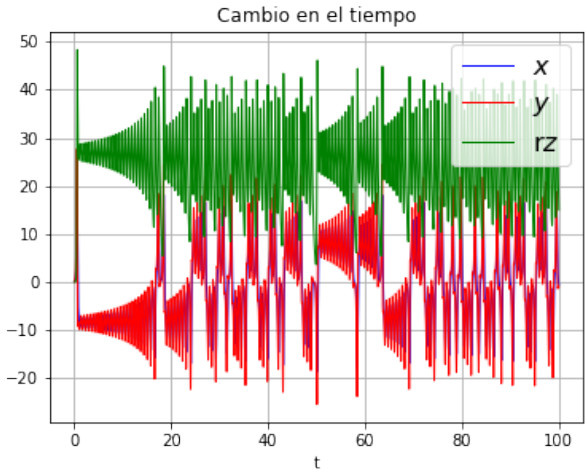
Y las gráficas arrojadas por este código son la siguientes:



Hacer estas gráficas fue la primera parte de la actividad, y de manera adicional, se nos pedía que se hiciera un gráfico que mostrara el comportamiento de las distintas variables involucradas (x, y, y z) en función del tiempo. Para esto, se uso el siguiente código:

```
from numpy import loadtxt
from pylab import figure, plot, xlabel, grid, hold, legend, title, savefig
from matplotlib.font_manager import FontProperties
%matplotlib inline
t= time_points
figure(1, figsize=(6, 4.5))

xlabel('t')
grid(True)
# hold(True)
lw = 1

plot(t, x, 'b', linewidth=lw)
plot(t, y, 'r', linewidth=lw)
plot(t, z, 'g', linewidth=lw)

legend((r'$x$', r'$y$','r$z$'), prop=FontProperties(size=16))
title('Cambio en el tiempo')
savefig('imagenbidimensional.png', dpi=100)
```

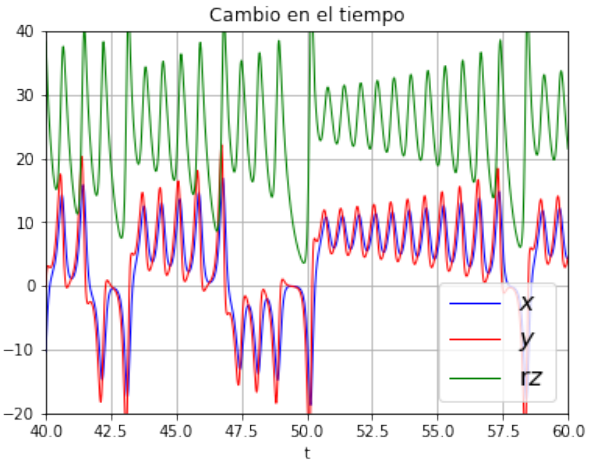
Las imagen creada es esta:



Sin embargo, en esta no es posible ver del todo bien el comportamiento de la variable x, el cual esta marcado por la linea azul, por ello es que se usaron limites en para poder ver mejor el comportamiento de esta. Mediante el siguiente fragmento:

```
#Como el movimiento de la variable x casi no se ve, delimitamos la gráfica a ciertos valoresfrom numpy imp
ort loadtxt
from numpy import loadtxt
from pylab import figure, plot, xlabel, grid, hold, legend, title, savefig
from matplotlib.font_manager import FontProperties
%matplotlib inline
t= time_points
figure(1, figsize=(6, 4.5))

xlabel('t')
grid(True)
# hold(True)
lw = 1

plot(t, x, 'b', linewidth=lw)
plot(t, y, 'r', linewidth=lw)
plot(t, z, 'g', linewidth=lw)
plt.xlim((40,60))
plt.ylim((-20,40))
legend((r'$x$', r'$y$','r$z$'), prop=FontProperties(size=16))
title('Cambio en el tiempo')
savefig('imagenbidimensionallimitado.png', dpi=100)
```

Obteniendo la siguiente imagen:



En esta imagen es fácil ver que el comportamiento de x y y es casi el mismo, de modo que sus lineas de movimientos están una sobre la otra.

En segundo lugar, para la primera parte del examen, que consitía en repetir el código de Geoff Beoing, teníamos que hacer una animación del atráctor de Lorenz, la cual también fue proporcionada, y se verá a continuación el código que crea dicha animación.

```
# return a list in iteratively larger chunks
def get_chunks(full_list, size):
    size = max(1, size)
    chunks = [full_list[0:i] for i in range(1, len(full_list) + 1, size)]
    return chunks
```

```
# get incrementally larger chunks of the time points, to reveal the attractor one frame at a time
chunks = get_chunks(time_points, size=20)
```

```
# get the points to plot, one chunk of time steps at a time, by integrating the system of equations
points = [odeint(lorenz_system, initial_state, chunk) for chunk in chunks]
```

```
# plot each set of points, one at a time, saving each plot
for n, point in enumerate(points):
    plot_lorenz(point, n)
```

```
# create a tuple of display durations, one for each frame
first_last = 10500 #show the first and last frames for 100 ms
standard_duration = 2000 #show all other frames for 5 ms
durations = tuple([first_last] + [standard_duration] * (len(points) - 2) + [first_last])
```

```
# load all the static images into a list
images = [Image.open(image) for image in glob.glob('{}/*.png'.format(save_folder))]
gif_filepath = 'images/animated-lorenz-attractorejemplo.gif'
```

```
# save as an animated gif
gif = images[0]
gif.info['duration'] = durations #ms per frame
gif.info['loop'] = 0 #how many times to loop (0=infinite)
gif.save(fp=gif_filepath, format='gif', save_all=True, append_images=images[1:])
```

```
# verify that the number of frames in the gif equals the number of image files and durations
Image.open(gif_filepath).n_frames == len(images) == len(durations)
```

```
True
```

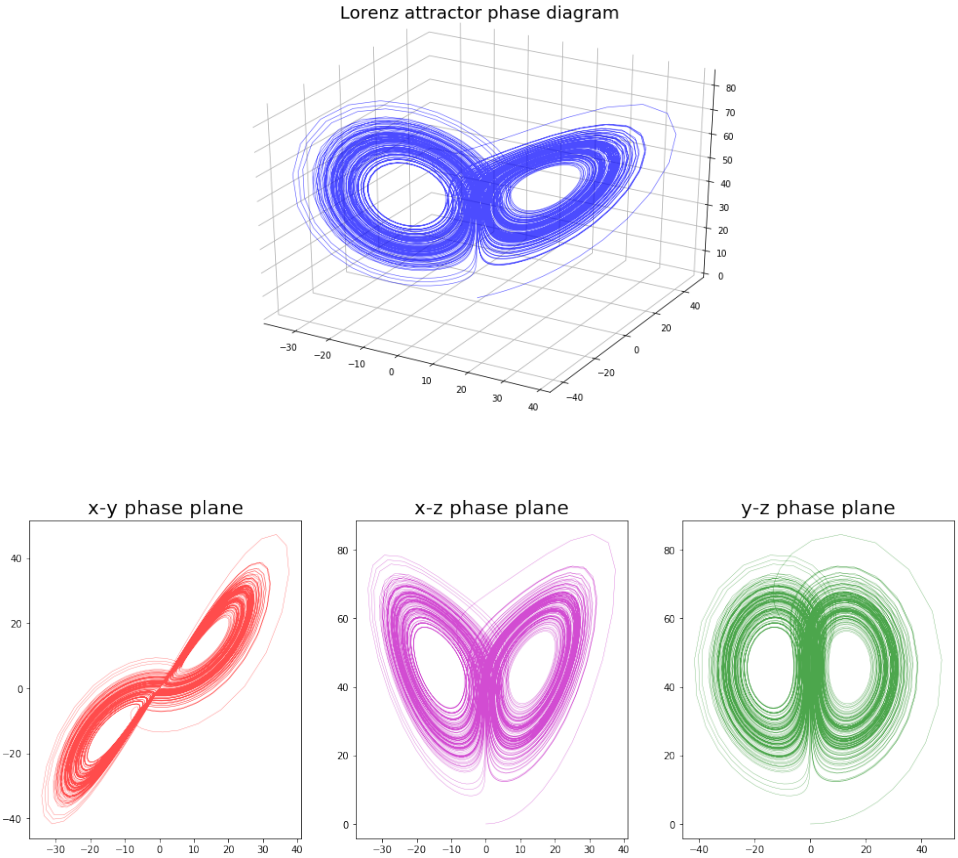
```
IPdisplay.Image(url=gif_filepath)
```

Sin embargo, la animación no puede ser mostrada ni en github ni aquí, sin embargo, al momento de correr el código en Python, se puede perfectamente.

1.2 Objetivo 2

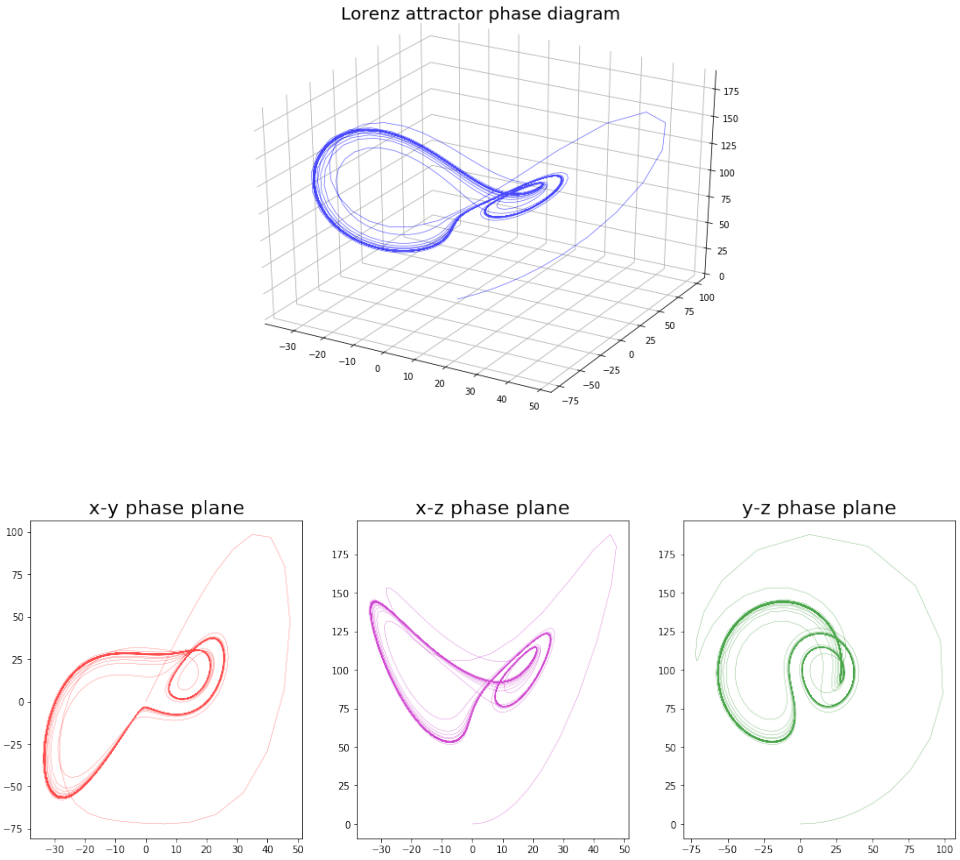
En segundo lugar, se nos pidió que observáramos el comportamiento del atráctor de Lorenz con distintos valores para los parámetros sigma, rho y beta. Los cuales se muestran a continuación en el primer caso, seguidos de sus gráficas en tres dimensiones, y las gráficas para los planos.

```
# Se definen las condiciones iniciales (aka x, y, z positions in space)
initial_state = [0.1, 0, 0]

# Definimos los parametros del sistema: sigma, rho y beta
sigma = 28.
rho   = 46.92
beta  = 4.

# Se definen los puntos de tiempo para resolver el sistema, posicionados entre los tiempos de inicio y fin
al
start_time = 0
end_time = 100
time_points = np.linspace(start_time, end_time, end_time*100)
```



Como en el objetivo uno, también debíamos crear una animación de este comportamiento, sin embargo, el código es exactamente el mismo, de modo que no se mostrará de nuevo en el documento. Por último, había que analizar el comportamiento del sistema en con otros parámetros, los cuales se muestran a continuación, seguido de las gráficas proporcionadas:

```
# Se definen las condiciones iniciales (aka x, y, z positions in space)
initial_state = [0.1, 0, 0]

# Definimos los parametros del sistema: sigma, rho y beta
sigma = 10.
rho   = 99.96
beta  = 8./3.

# Se definen los puntos de tiempo para resolver el sistema, posicionados entre los tiempos de inicio y fin
al
start_time = 0
end_time = 100
time_points = np.linspace(start_time, end_time, end_time*100)
```



Al igual que anteriormente, se pedía hacer una animación del comportamiento, sin embargo, esta no fue posible de hacer, ya que el sistema no podía leer los datos apropiadamente, lo que ocasionaba que el ordenador se congelará al poco tiempo de intentar leer los datos.