Taller de Aplicación de Modelos Numéricos

→|MODELO SIR

Recordemos ahora que el sistema de Lorenz está dado por el siguiente conjunto de ecuaciones:

$$\frac{dx_1}{dt} = -\sigma x_1 + \sigma x_2 \quad [1]$$
$$\frac{dx_2}{dt} = -x_1 x_3 + r x_1 - x_2 \quad [2]$$
$$\frac{dx_3}{dt} = x_1 x_2 - b x_3 \quad [3]$$

Aquí las variables **(X_1, X_2, X_3)** son proporcionales a **(ω, Ta Tb)**, respectivamente y cada celda de convección que se forma tiene una velocidad característica de rotación **ω** y una variación en temperatura horizontal y vertical **Ta** y **Tb**, respectivamente. Recordemos que el sistema de Lorenz es una simplificación de modelado de un fenómeno que llamamos convección, el cual es el responsable de muchas cosas en nuestra vida diaria como el clima y el agua hirviendo en el fogón de nuestra cocina y ahora en esta clase resolveremos estas ecuaciones usando RK4. El código que escribimos en la clase anterior para implementar un paso del algoritmo RK4 también sirve para resolver un sistema de EDOs, en ese caso tanto **y** como $\mathbf{y_0}$ serán estructuras tipo array() donde cada elemento corresponde a una de las variables **(X_1, X_2, X_3)**.

$$\frac{dx_1}{dt} = -\sigma x_1 + \sigma x_2 \quad [1]$$
$$\frac{dx_2}{dt} = -x_1 x_3 + r x_1 - x_2 \quad [2]$$
$$\frac{dx_3}{dt} = x_1 x_2 - b x_3 \quad [3]$$

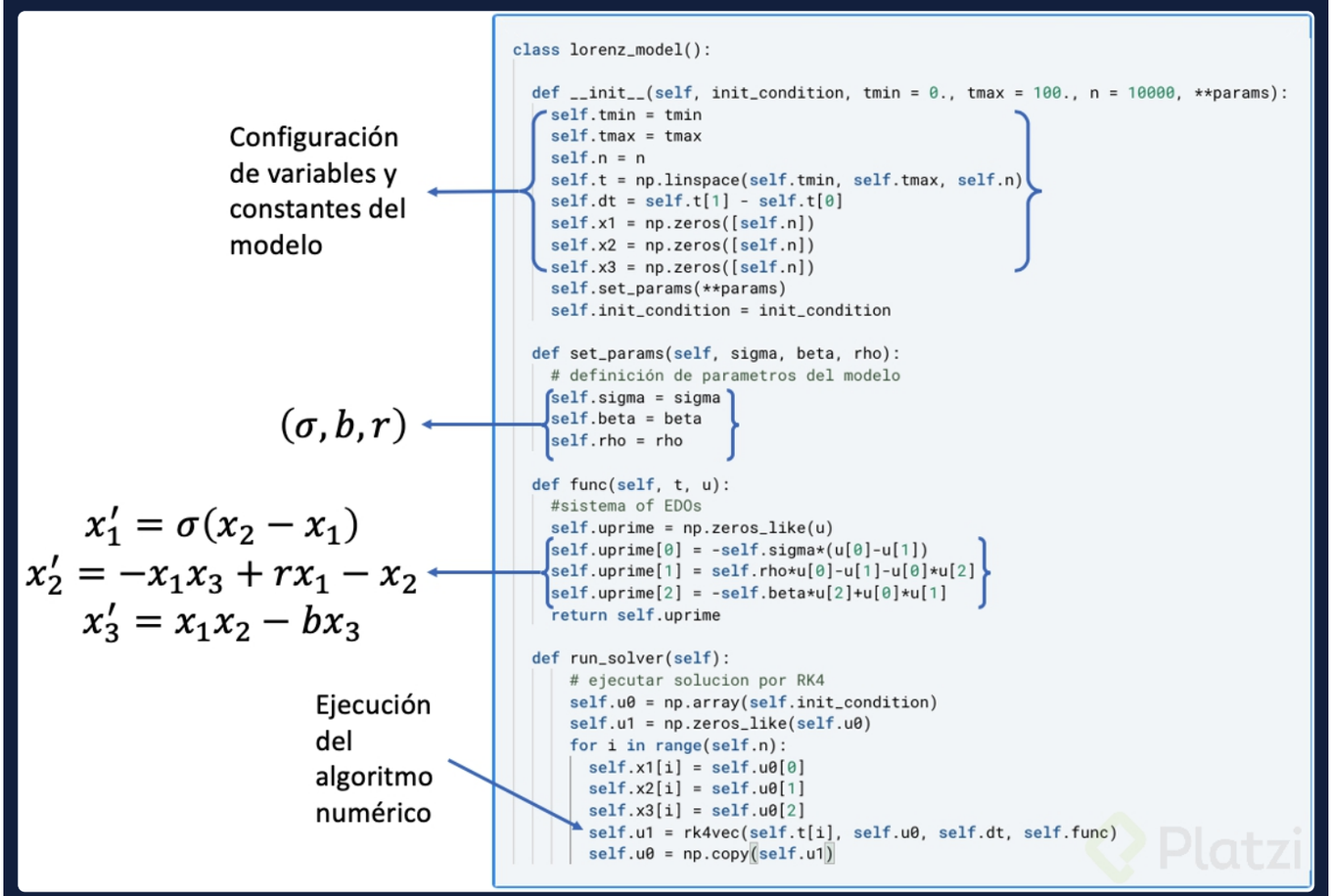
```
def rk4vec(t0, y0, dt, f):
    k1 = f(t0, y0)
    k2 = f(t0 + dt/2.0, y0 + dt * k1 / 2.0)
    k3 = f(t0 + dt/2.0, y0 + dt * k2 / 2.0)
    k4 = f(t0 + dt, y0 + dt * k3)
    y = y0 + (dt/6.0)*(k1 + 2.0*k2 + 2.0*k3 + k4)
    return y
```

Ahora, vamos a implementar una clase que contiene métodos para construir el sistema de EDOs de Lorenz y el uso de RK4 para obtener la solución numérica del mismo. Esta clase contiene lo siguiente:

1. Un constructor donde definimos el tiempo inicial y final de la simulación, la condición inicial y los parámetros de las ecuaciones de Lorenz, así mismo se define el número de pasos a ejecutar lo cual a su vez determina el intervalo de tiempo **Δ**t.
2. Un método ***set_params()*** que guarda los parámetros del modelo como atributos de la clase.
3. Un método ***func()*** que define las derivadas que determinan el sistema de EDOs, esto es la version en varias ecuaciones de la definición de **f(t, y)** que hicimos en clases previas

```
def f(t, y): return y
```

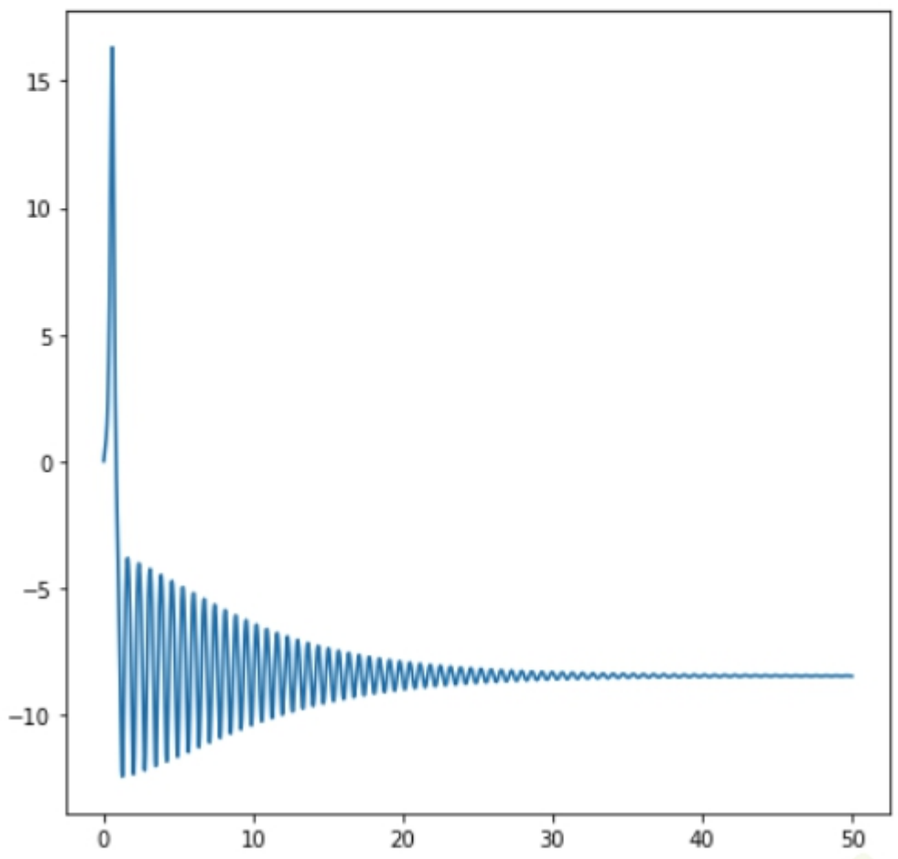
4. Finalmente, un método **run_solver()** que ejecuta el algoritmo RK4 paso a paso para calcular la solución numérica



Una vez definida esta clase es posible ejecutar la solución numérica unas pocas líneas, pero antes de hacerlo es importante notar que aquí el valor de los parámetros es crucial para determinar el tipo de comportamiento que el sistema tendrá, colocaremos dos ejemplos particulares:

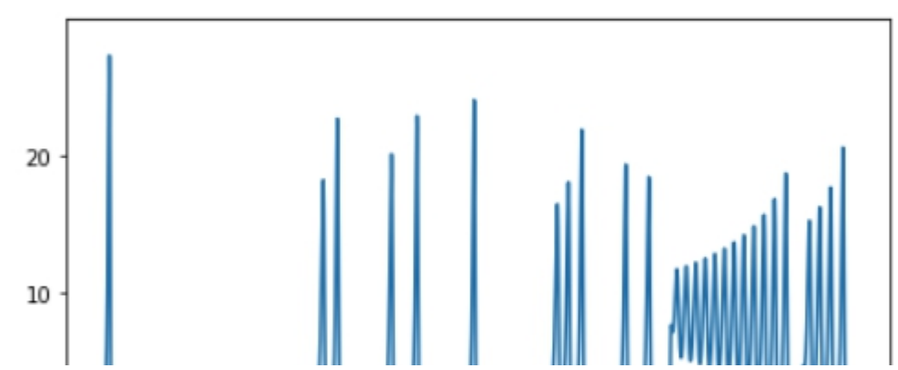
[18]

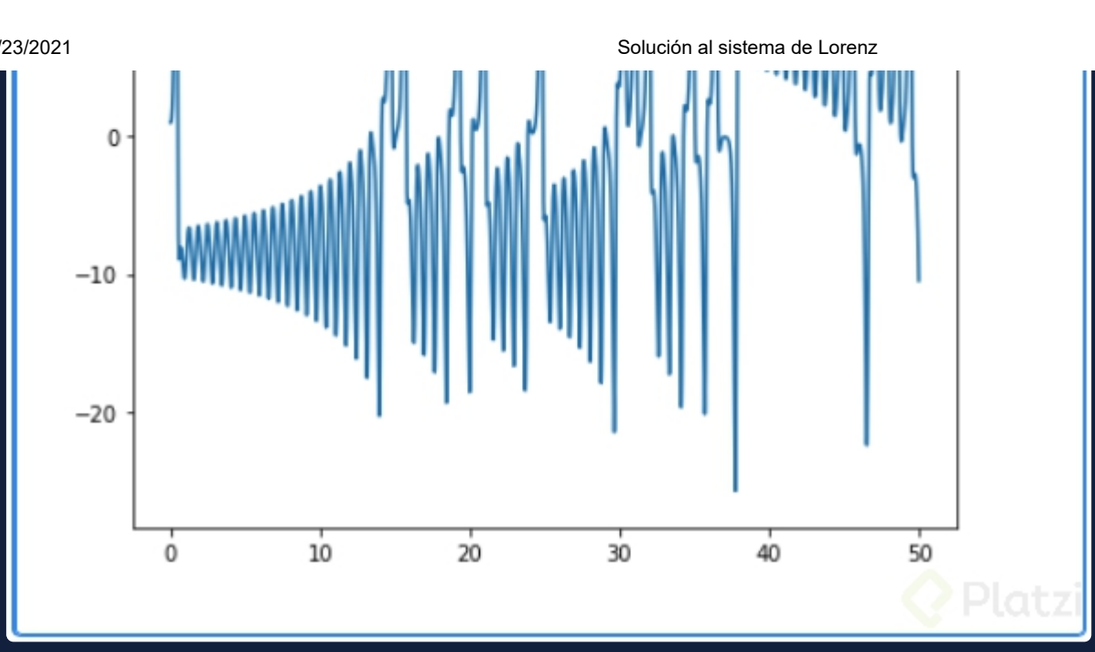
```
params = {'sigma': 5, 'beta': 2.667, 'rho': 28}
init_condition = [0, 1, 1.05]
model = lorenz_model(init_condition=init_condition, **params)
model.run_solver()

plt.figure(figsize=(7,7))
plt.plot(model.t, model.x1)
plt.show()
```



[19]

```
params = {'sigma': 10, 'beta': 2.667, 'rho': 28}
init_condition = [0, 1, 1.05]
model = lorenz_model(init_condition=init_condition, **params)
model.run_solver()

plt.figure(figsize=(7,7))
plt.plot(model.t, model.x2)
plt.show()
```

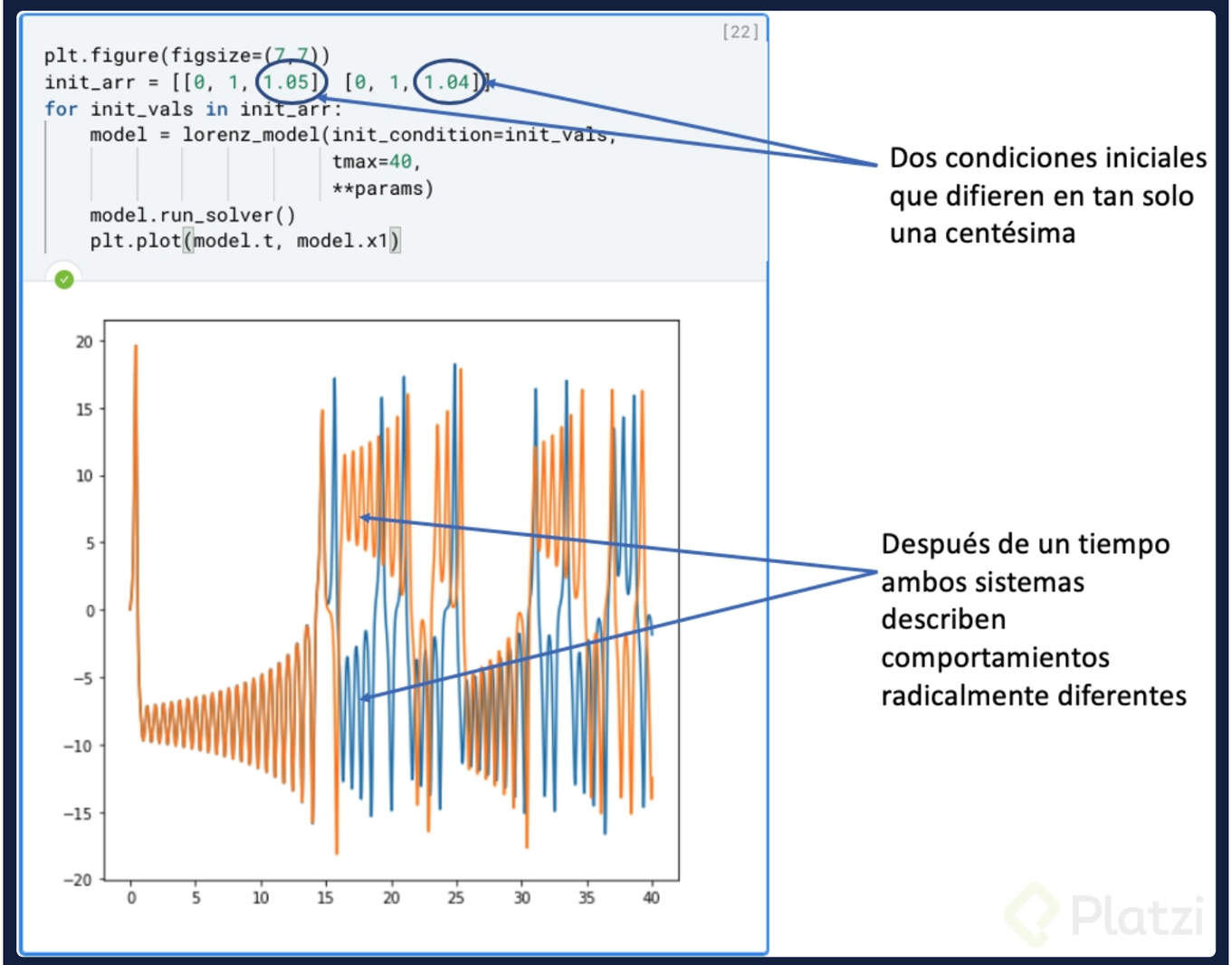




La única diferencia entre ambas soluciones es que el valor del parámetro **σ** cambia de 5 (a la izquierda) a 10 (a la derecha). En ambos casos graficamos la velocidad de rotación de la celda de convección, y a partir de eso vemos que:

1. En el caso de la izquierda, la velocidad de rotación de la celda convectiva se estabiliza en un valor constante después de un tiempo, lo que indica que la celda mantendrá un comportamiento estable durante tiempos largos.
2. En el caso de la derecha, la velocidad de rotación se vuelve errática y no logra estabilizarse, mostrando una evolución en el tiempo que no parece seguir un patrón bien definido y a este comportamiento lo denominamos convección turbulenta. En este caso las variables que describen el comportamiento de las celdas de convección muestran constantemente fluctuaciones erráticas.

Pero es importante resaltar que ambos casos consideran la misma condición inicial. Y la conclusión de esto es que el sistema puede mostrar un comportamiento regular para ciertas condiciones físicas particulares y para otras puede mostrar un comportamiento turbulento, que es lo que usualmente sucede en la vida real. Incluso, otro aspecto importante de este sistema dinámico es que presenta una propiedad interesante denominada Caos. ¿Qué es el Caos?, bueno para entender esto vamos a comparar dos simulaciones que tengan los mismos parámetros pero cuyas condiciones iniciales sean diferentes tan solo por una valor muy pequeño:



Del gráfico resultante vemos que una pequeña variación en la condición inicial conduce a comportamientos totalmente diferentes y a esta característica de un sistema dinámico la denominamos Caos. En general, el caos es un término que hace referencia a que una pequeña modificación de un sistema implica consecuencias muy grandes. A menudo encontramos referencias de esto en la cultura pop, como aquellas películas del Efecto Mariposa. Si no las has visto y aún no crees estar seguro de lo que significa el caos, recomiendo que las veas para que tengas más conciencia del significado de caos.

Así pues, cuando hay caos presente en un sistema dinámico es extremadamente difícil hacer predicciones de este, porque en la vida real los datos se obtienen a partir de instrumentos que tienen una precisión finita, así que medir con precisión absoluta una condición inicial para la formación de un huracán (por ejemplo) es imposible, y esto ya representa una limitación para poder hacer predicciones del clima en general.

En nuestras próximas clases entraremos a la sección final de nuestro curso donde veremos como podemos usar métodos numéricos para modelar de manera más acertada la propagación de epidemias como el SARS-COV-2, por ahora cerramos esta clase con el notebook definitivo de lo que acabamos de hacer en este link.