Los modelos cuya solución está basada en un algoritmo numérico los denominamos modelos numéricos. El algoritmo numérico más sencillo que existe es el método de Euler el cual discutiremos a continuación.

Método de Euler

Cuando tenemos una ecuación diferencial de primer orden para una función desconocida ***y(t)*** lo más general posible, escribimos:

$$\frac{dy}{dt} = f(t,y)$$

Donde ***f(y, t)*** representa una expresión que contiene todos los demás términos de la ecuación diferencial que no acompañen a la derivada. Ahora, como vimos con el problema de la propagación de una epidemia, un sistema dinámico debe tener un punto de inicio o condición inicial. En general siempre debemos definir cual es esta condición inicial para nuestro sistema así:

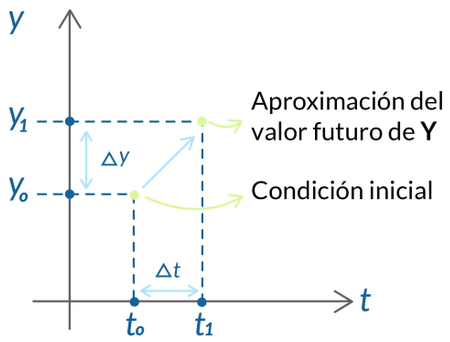
$$y(t_0) = y_0$$

Con $\mathbf{y_0}$ siendo el valor inicial de la variable que estamos buscando predecir. En este sentido, el método de Euler propone construir la solución de la ecuación diferencial paso a paso suponiendo que la derivada se aproxima por medio de diferencias finitas pero muy pequeñas:

$$\frac{dy}{dt} \sim \frac{\triangle y}{\triangle t} = \frac{y_1 - y_0}{t_1 - t_0}$$

$$\frac{y_1 - y_0}{\triangle t} \sim f(t_0, y_0)$$

$$y_1 \sim y_0 + \triangle t \; f(t_0, y_0)$$



Valor inicial

Intervalo de tiempo finito

EDO Evaluada en el tiempo inicial

Estas diferencias finitas representan intervalos de tiempo tan pequeños como uno los considere y paso a paso se puede ir calculando el siguiente valor de la variable ***y*** con base en el valor del tiempo anterior:

$$y_{n+1} = y_n + \Delta t \times f(t_n, y_n)$$

Esto representa un proceso iterativo donde vamos construyendo la solución paso a paso. El precio de esto es que el aproximar una derivada por intervalos finitos implica que en cada paso se genera un error que se acumula en los siguientes pasos. Matemáticamente hablando, la única manera de reducir este error es considerando un intervalo de tiempo $\Delta t \rightarrow 0$, pero computacionalmente no podemos considerar números infinitamente pequeños, entonces procuramos tomarlo tan pequeño como nuestros recursos de cómputo nos lo permitan. Esta es la gran diferencia entre el cálculo de infinitesimales y los métodos numéricos.

Dado que los métodos numéricos son muy sencillos conceptualmente hablando, pero pueden representar procesos que deban repetirse muchas veces, el uso de un computador para su ejecución es lo ideal y por lo tanto hoy en día todos los métodos numéricos se implementan en un lenguaje de programación para poder realizar múltiples pasos en tiempos razonablemente cortos y dejándole todo el trabajo a una máquina. Como dato curioso estos algoritmos eran calculados a mano antes de que aparecieran las grandes computadoras, pero ya te imaginarás lo engorroso que eso era en aquella época.

Solución de Euler en Python

Tomaremos como ejemplo una ecuación diferencial sencilla junto con una condición inicial que usaremos como base para aplicar el método de Euler:

$$\frac{dy}{dt} = y$$
$$y(0) = 1$$

Es decir esto representa un problema donde la solución será una función que es igual a su derivada y que su valor para ***t = 0*** sea 1. El algoritmo de Euler en este caso quedaría de la siguiente manera:

$$y_{n+1} = y_n + \Delta t \times y_n = (1 + \Delta t) y_n$$
$$y_0 = 1$$

Para inicial el algoritmo debemos definir el intervalo de tiempo, digamos que tomamos ***Δt = 0.01*** y calculamos los pasos así:

$$y_1 = (1 + 0.01) \times 1 = 1.01 \text{ para } t = 0.01$$
$$y_2 = (1 + 0.01) \times 1.01 = 1.0201 \text{ para } t = 0.02$$
$$y_3 = (1 + 0.01) \times 1.0201 = 1.030301 \text{ para } t = 0.03$$

Y así sucesivamente hasta el número de pasos que sea necesario. Por ejemplo si necesitamos construir la solución hasta un tiempo ***t = 10*** , requerimos calcular al menos 1000 pasos con un ***Δt = 0.01***. Esto es mejor hacerlo en un computador, que es justamente lo que haremos a continuación en las siguientes líneas de código:

[23]

```
import numpy as np

ys = [1] # creamos un arreglo inicial de valores en y
ts = [0] # creamos un arreglo inicial de valores en t
dt = 0.01
num_steps = 1000
for i in range(num_steps):
    ts.append(ts[-1]+dt) # calculamos el proximo t y lo agregamos al arreglo
    ys.append((1+dt)*ys[-1]) # calculamos el proximo y y lo agregamos al arreglo

print(ts)
print(ys)
```

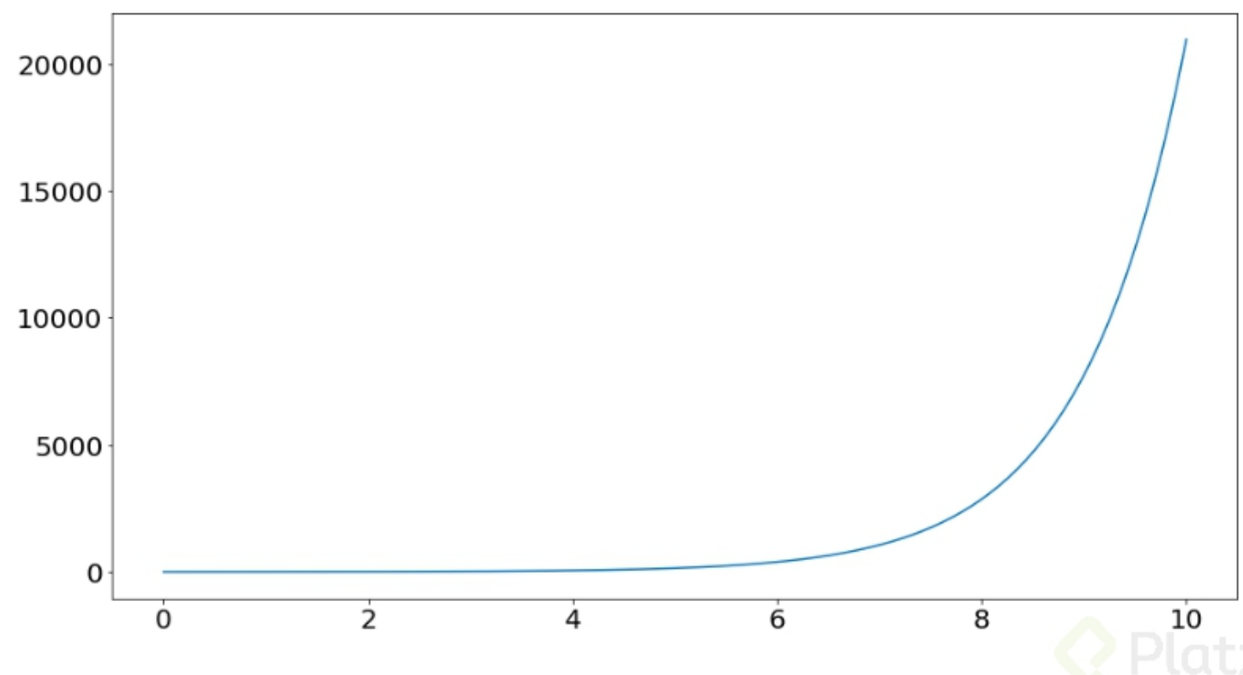
```
[0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.060000000000000005, 0.07, 0.08, 0.09, 0.099999999999
[1, 1.01, 1.0201, 1.030301, 1.04060401, 1.0510100501, 1.061520150601, 1.0721353521070098
```

De aquí, la solución numérica correspondiente se vé así:

[25]

```
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 8))
plt.plot(ts, ys)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```



Y para los que ya saben que la solución exacta (la que obtenemos de los métodos del cálculo) de esta ecuación ecuación diferencial es la función exponencial:

$$f(x) = e^x$$

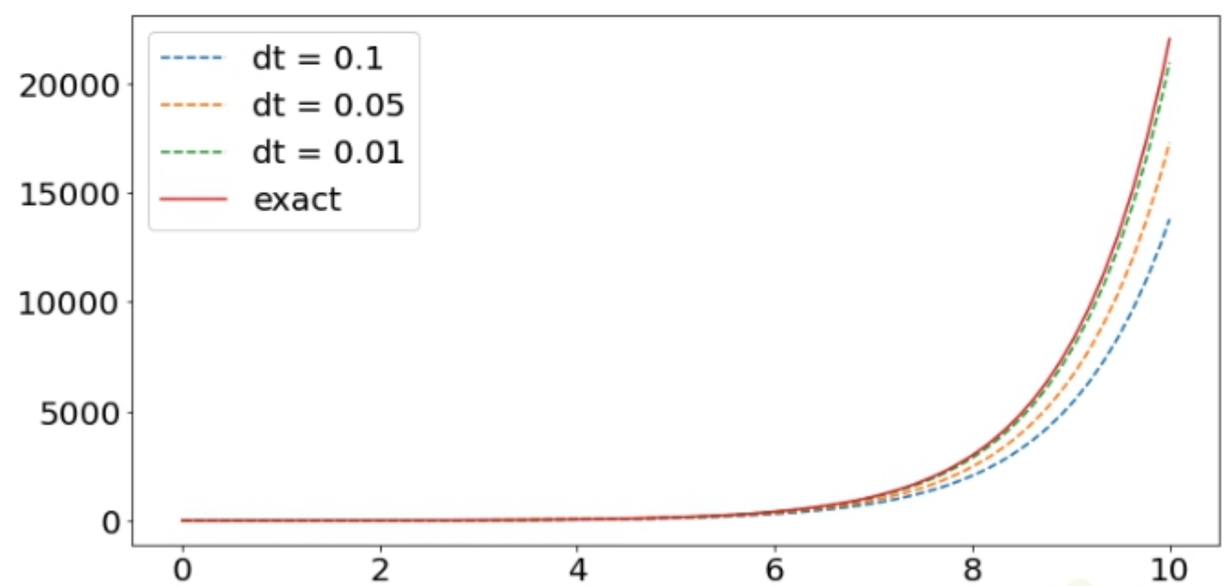
Se darán cuenta que al comparar ambos tipos de soluciones, se observa una pequeña diferencia que va creciendo conforme avanzamos en el tiempo y que es mayor si escogemos intervalos de tiempo más largos:

[10]

```
def exact_sol(ts): return np.exp(ts)

def num_sol(ts, dt, tf=10, y0 = 1):
    ys = [y0]
    ts = [0]
    num_steps = int(tf/dt)
    for i in range(num_steps):
        ts.append(ts[-1]+dt)
        ys.append((1+dt)*ys[-1])
    return ts, ys

plt.figure(figsize=(12, 6))
for dt in [0.1, 0.05, 0.01]:
    ts, ys = num_sol(ts, dt)
    plt.plot(ts, ys, '--', label = 'dt = {}'.format(dt))
plt.plot(ts, exact_sol(ts), label = 'exact')
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(fontsize = 20)
plt.show()
```



Y si quieres seguir jugando con este código te darás cuenta de que esta diferencia desaparece a medida que el intervalo de tiempo es cada vez más pequeño, solo que cada vez tendrás que realizar más pasos.

Y es así como vemos una alternativa para obtener soluciones a modelos de sistemas dinámicos a partir del paradigma numérico, con la advertencia que los métodos numéricos tienen el problema que el error es directamente proporcional al intervalo de tiempo que se usa y esto, a su vez, es inversamente proporcional al número de operaciones que deben realizarse. Es decir, que para obtener mejores soluciones es preciso realizar más pasos y por lo tanto consumir más recursos computacionales.

El notebook de esta clase lo encuentras en este link de Google Colab. En nuestra próxima clase discutiremos en detalle las diferencias entre las soluciones exactas que obtenemos con álgebra y cálculo contra las soluciones numéricas.