

# Practica 2

González Borja, Miguel  
Illescas Arizti, Rodrigo  
Meyer Mañón, Juan Carlos  
Rodríguez Orozco, Alejandro

28 / Abril / 2018

## Introducción

A continuación los ejercicios de la práctica. Todos fueron realizados en Python 3.6 utilizando los paquetes *numpy* y *scipy.optimize*. También se incluye un cuaderno de IPython para cada ejercicio dentro de la carpeta Ejercicios, junto con los scripts *.py* para obtener los resultados. Cada uno de estos utiliza los siguientes imports:

```
In [1]: import sys
        sys.path.append('../')
        import latexStrings as ls
        import numpy as np
        import matplotlib.pyplot as plt
        from math import *
        from IPython.display import Latex
        import odesolver
```

Se puede encontrar un repositorio del proyecto [aqui](#).

## Ejercicio 1

Tenemos el PVI:

$$y' = y + 2e^{-t} \quad y(0) = 1 \quad (1)$$

Dado que el lado derecho de la EDO,  $y + 2e^{-t}$  es de clase  $C^1$  para todo  $(t, y) \in \mathbb{R} \times \mathbb{R}$ , y por lo tanto Lipschitz Continua en dicha region, sabemos que existe una solucion unica a este problema, que en general es dada por:

$$y(t) = (y_0 e^{-t_0} + e^{-2t_0} - e^{-2t}) e^t \quad (2)$$

Ahora apliquemos el metodo de Euler Explicito con paso  $h = 0.1$ :

```
In [1]: f = lambda t, y : y+2*np.exp(-t)
        exact = lambda t, t0, y0 : np.exp(t)*(y0*np.exp(-t0)+np.exp(-2*t0)-np.exp(-2*t))
        I = (0, 1)
```

```

y0 = 1
y1 = exact(1,0,1)
T, W = odesolver.solve(f,y0,I,10,method="Explicit Euler")
globalError = abs(W[0,10]-y1)
localErrors = [abs(W[0,i+1] - exact(T[i+1], T[i], W[0,i])) for i in range(10)]
maxLocalError = max(localErrors)

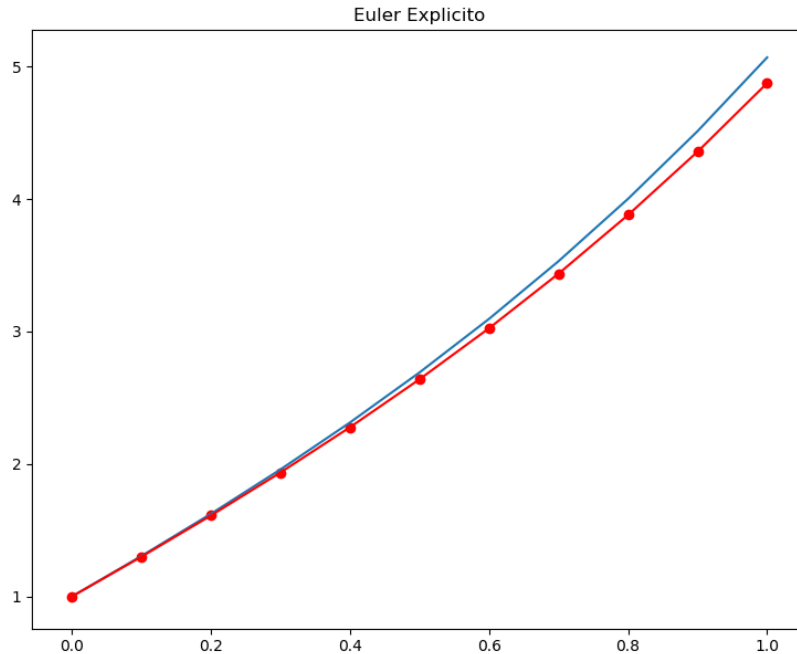
```

$w = \{1, 1.3, 1.6109675, 1.9358104, 2.2775551, 2.6393746,$   
 $3.0246182, 3.4368423, 3.8798436, 4.3576938, 4.8747771\}$

Error Global: 0.1939071469339151

Maximo Error Local: 0.022668868433444622

Graficamente, observemos como se comporta la solucion exacta contra la aproximada:



Vemos que la solucion aproximada, aunque sigue la direccion general, siempre va por debajo de la solucion verdadera. Esto se debe a que nuestra solucion general (2) es convexa en el intervalo donde estamos buscando la solucion, y por lo tanto el metodo subestima la pendiente en sus aproximaciones.

Ahora apliquemos el metodo de Euler Explicito con pasos  $h = 0.1 \times 2^{-k}$  para  $k \in \{0, 1, \dots, 5\}$  en el intervalo  $[0, 1]$ :

```

In [2]: res = []
        globalErrors=[]

```

```

for k in range(6):
    h = 0.1*(2**(-k))
    m = int((I[1]-I[0])/h)
    T, W = odesolver.solve(f, y0, I, m, method="Explicit Euler")
    g = abs(W[0,m]-y1)
    globalErrors.append(g)
    localErrors = [abs(W[0,i+1] - exact(T[i+1], T[i], W[0,i])) for i in range(m)]
    maxLocalError = max(localErrors)
    eoc = "NaN" if k==0 else log(g/prevg)/log(h/prevh)
    res.append([k, h, maxLocalError, eoc])
    prevh = h
    prevg = g

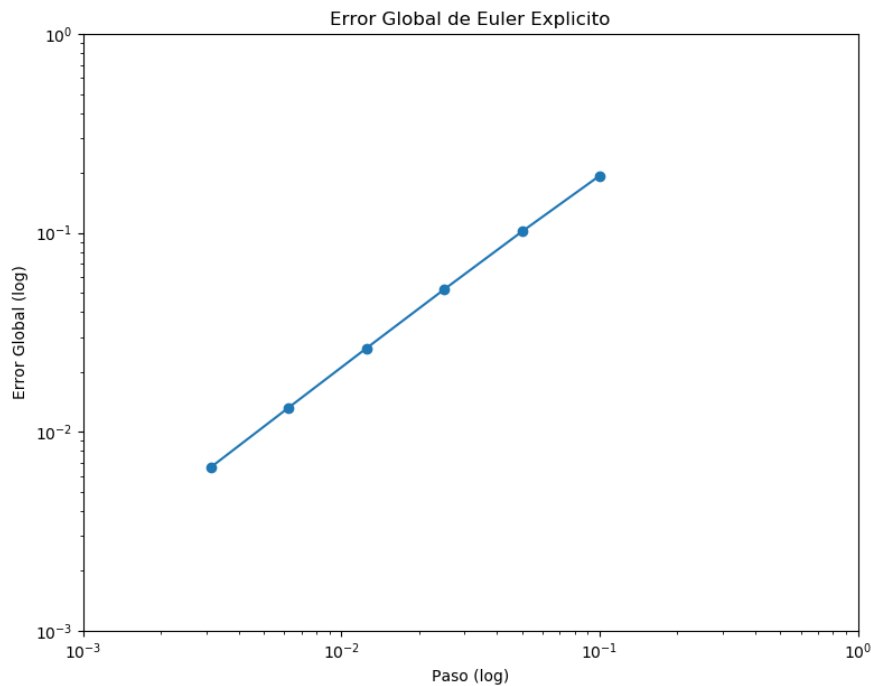
```

Observemos el comportamiento del error del metodo en relacion con el tamaño del paso:

k	Paso	Error Local Maximo	eoc
0	0.1	0.022668868433444622	NaN
1	0.05	0.005982387299206415	0.933587546963545
2	0.025	0.0015384740393056262	0.9656406382261543
3	0.0125	0.00039021932959482086	0.9825161515419355
4	0.00625	9.827082448321534e-05	0.9911799106800493
5	0.003125	2.4658225118656674e-05	0.9955701393945221

Sabemos que en teoria, los pasos de Euler Explicito tienen un error del orden de  $O(h^2)$ , mientras que el error global es de orden  $O(h)$ . En efecto, vemos que el error local maximo siempre cumple la cota deseada, pues en el peor de los casos, cuando  $h = 0.1$ , el error es aproximadamente  $0.022 = 2.2h^2$ , y conforme disminuye el tamaño de los pasos se sigue cumpliendo la cota.

Por otro lado, observamos que la razon de convergencia experimental (eoc) empieza siendo 0.93 pero rapidamente se aproxima a 1 conforme disminuye el tamaño de los pasos utilizados en su calculo. Esto nos lleva a la conclusion de que el error global en efecto es de orden 1, lo cual es aun mas evidente al observar el comportamiento del error en terminos del tamaño del paso, como se muestra en la siguiente grafica:



Podemos observar que, en escala logarítmica, el error global y el paso tienen una relación lineal, que es lo esperado de un método con error global de orden  $O(h)$ .

## Ejercicio 2

Tomando el PVI (1) y la solución general de la ecuación (2) del ejercicio anterior aplicaremos el método del Trapecio explícito con paso  $h_1 = 0.1$ . En el ejercicio 1 se argumentó la existencia y unicidad del PVI.

Calculemos el error global en el punto  $t = 1$ :

$$E_{\text{Trapezio}}(h_1) = |w_{10} - y(1)|$$

con  $w_{10}$  la aproximación del método en  $t=1$ .

```
In [1]: f = lambda t, y : y+2*np.exp(-t)
        exact = lambda t, t0, y0 : np.exp(t)*(y0*np.exp(-t0)+np.exp(-2*t0)-np.exp(-2*t))
        I = (0, 1)
        y0 = 1
        y1 = exact(1,0,1)
        T, W = odesolver.solve(f,y0,I,10,method="Explicit Trapezoid")
        globalError = abs(W[0,10]-y1)
```

$$E_{\text{Trapezio}}(h_1) = 0.0046374955401011775$$

Veamos lo que sucede cuando usamos el paso  $h_2 := \frac{h_1}{2} = 0.05$  y calculemos el correspondiente error global en  $t=1$ :

$$E_{\text{Trapezio}}(h_2) = |w_{20} - y(1)|$$

```
In [2]: T, W = odesolver.solve(f,y0,I,20,method="Explicit Trapezoid")
        globalError = abs(W[0,20]-y1)
```

$$E_{Trapezio}(h_2) = 0.0012209778161409446$$

Ahora calcuemos el error global en  $t=1$  usando el método Runge-Kutta 4 usando  $h_1 = 0.1$ :

$$E_{RK4}(h_1) = |w_{10} - y(1)|$$

```
In [3]: T, W = odesolver.solve(f,y0,I,10,method="RK4")
        globalError = abs(W[0,10]-y1)
```

$$E_{RK4}(h_1) = 2.334812125859287e - 06$$

	$E_{Trapezio}(h_1)$	$E_{Trapezio}(h_2)$	$E_{RK4}(h_1)$
Error (global en $t = 1$ )	$4.63749554010 \times 10^{-3}$	$1.22097781614 \times 10^{-3}$	$2.3348121258 \times 10^{-6}$
Número de estados	20	40	40

En la tabla anterior notamos lo siguiente:

$$E_{RK4}(h_1) < E_{Trapezio}(h_2) < E_{Trapezio}(h_1)$$

Podemos observar que mediante el método del Trapecio explícito con paso  $h_2$  obtenemos un error aproximadamente 4 veces menor que con paso  $h_1$  ya que:

$$\begin{aligned} E_{Trapezio}(h_1) &= O(h_1^2) \\ \Rightarrow E_{Trapezio}(h_2) &= O(h_2^2) = O\left(\frac{h_1}{2}\right)^2 \\ \Rightarrow E_{Trapezio}(h_2) &\approx \frac{1}{4}O(h_1^2) = \frac{1}{4}E_{Trapezio}(h_1) \end{aligned}$$

El problema es que el método con  $h_2$  necesita de 40 estados, el doble con respecto a usar  $h_1$ .

Por otro lado, el método Runge-Kutta 4 necesita de 40 estados al igual que el Trapecio explícito con  $h_2$  pero observando los resultados de la tabla notamos que el error global  $E_{RK4}$  es menor a  $E_{Trapezio}(h_2)$  por un factor de  $10^3$  aproximadamente.

Por lo tanto, para reducir el error, es preferible usar el método Runge-Kutta 4 ya que, por los mismos 40 estados requeridos, se obtiene un error menor al del Trapecio explícito con  $h_2$ .

### Ejercicio 3

Tomemos el siguiente PVI:

$$\begin{aligned} y_1' &= -y_1 + y_2 \\ y_2' &= -y_1 - y_2 \end{aligned} \quad \text{con} \quad \begin{aligned} y_1(0) &= 0 \\ y_2(0) &= 1 \end{aligned} \quad \text{para } t \in [0, 1] \quad (3)$$

Con solución exacta  $\vec{y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \end{pmatrix}$  dada por :

$$\begin{aligned} y_1(t) &= e^{-t} \sin(t) \\ y_2(t) &= e^{-t} \cos(t) \end{aligned} \quad (4)$$

Sea  $\vec{W}_j$  el vector de aproximación correspondiente a la aplicación del método del Punto Medio Implícito con paso  $h_j$ . La entrada  $w_i$  es la aproximación en  $t = 1$  de  $y_i$  para  $i, j \in \{1, 2\}$ . Definimos  $E_{PMImpl}(h_j) = \|\vec{G}_j\|_2$  el error global en  $t = 1$  para  $j \in \{1, 2\}$ , con  $\vec{G}_j := |\vec{W}_j - \vec{y}(1)|$  cuya entrada  $g_i = |w_i - y_i(1)|$ .

Aplicaremos el método para  $h_1 = 0.1$  y  $h_2 = 0.01$ :

```
In [1]: f = lambda t, y : np.array([-y[0]+y[1], -y[0]-y[1]])
        soln = lambda t: np.array([np.exp(-T)*np.sin(T), np.exp(-T)*np.cos(T)])
        y0 = np.array([0,1])
        T,W = odesolver.solve(f, y0, (0,1), 10, method = 'Implicit Midpoint')
        globalError1=linear.norm(W[:,10]-soln(T)[: ,10])
        dif=abs(W[:,10]-soln(T)[: ,10])
```

$$h_1 = 0.1$$

$$\vec{W}_1 = \begin{pmatrix} 0.310408 \\ 0.198583 \end{pmatrix}$$

$$\vec{G}_1 = \begin{pmatrix} 0.000848 \\ 0.000183 \end{pmatrix}$$

$$E_{PMImpl}(h_1) = 0.0008678198131427709$$

```
In [2]: T,W = odesolver.solve(f, y0, (0,1), 100, method = 'Implicit Midpoint')
        globalError2=linear.norm(W[:,100]-soln(1)[: ,100])
        dif=abs(W[:,100]-soln(T)[: ,100])
```

$$h_2 = 0.01$$

$$\vec{W}_2 = \begin{pmatrix} 0.309568 \\ 0.198764 \end{pmatrix}$$

$$\vec{G}_2 = \begin{pmatrix} 0.000008 \\ 0.000002 \end{pmatrix}$$

$$E_{PMImpl}(h_2) = 8.671073839286167e - 06$$

Sabemos que el método del Punto Medio Implícito con paso  $\hat{h}$  tiene un error global de de orden 2. Luego, con  $\hat{h} := h_1 = 0.1$  esperamos:

$$E_{PMImpl}(h_1) = O(h_1^2) = O(0.1^2)$$

Y con  $\hat{h} := h_2 = 0.01$  esperamos:

$$E_{PMImpl}(h_2) = O(h_2^2) = O(0.01^2)$$

Observamos que:

$$E_{PMImpl}(h_1) = O(h_1^2)$$

$$h_2 = \frac{h_1}{10} \Rightarrow E_{PMImpl}(h_2) = O\left(\frac{h_1}{10}\right)^2 \Rightarrow E_{PMImpl}(h_2) \approx \frac{1}{100}O(h_1^2) = \frac{1}{100}E_{PMImpl}(h_1)$$

Esto implica que el error global del método para  $h_2$  debe ser aproximadamente 100 veces menor al error global con el paso  $h_1$ . Comparemos esto con nuestros resultados:

$$E_{PMImpl}(h_2) = 8.671073839286167e - 06 \approx 8.67819813142771e - 06 = \frac{1}{100}E_{PMImpl}(h_1)$$

Y vemos que en efecto la reducción del error de  $h_1$  a  $h_2 = \frac{h_1}{10}$  es consistente con la teoría.

## **Ejercicio 4**

## **Ejercicio 5**

## **Ejercicio 6**