


Universidad Politécnica de Madrid

Ampliación de Matemáticas I

Informe Hito 3: Estimacion del error en Soluciones Analiticas.

Autora:
Inés Arauzo Andrés

Índice

Índice de figuras

1. Introducción

A lo largo del presente informe se pretenden comentar los resultados obtenidos al programar el Milestone 3. En este, se ha desarrollado una función para evaluar los errores cometidos en la integración numérica de diversos esquemas temporales (ver hito 2) por el método de la extrapolación de **Richardson** aplicada a un problema de Cauchy. Asimismo, se ha evaluado el **ratio de convergencia** de estos esquemas. De este modo, se procede a comentar el problema a resolver, el código utilizado y los resultados obtenidos.

2. Definición del problema

La extrapolación de Richardson es un método que permite evaluar el error cometido por un esquema numérico dado utilizando varias mallas. Para ello se procede de la siguiente forma:

Sea $\phi(\mathbf{h})$ el resultado numérico (ergo aproximado) de una simulación con paso de tiempo h de la que se conoce el valor exacto $\phi^{\mathbf{0}}$, considero que el valor numérico $\phi(\mathbf{h})$ admite expansión en serie de Taylor en base al resultado exacto $\phi^{\mathbf{0}}$ como función del paso de tiempo h,

$$\phi(\mathbf{h}) = \phi^{\mathbf{0}} + \mathbf{k_1}\mathbf{h^q} + \mathbf{O}(\mathbf{h^{q+1}}) \tag{2.0.1}$$

Donde q es el parámetro de convergencia, que corresponde al orden de integración del esquema numérico empleado

Consideramos 2 soluciones numéricas distintas ϕ^1, ϕ^2 con dos mallas t_1 y t_2 distintas

$$\phi^1 = \phi^0 + \mathbf{k_1}h_1^q + O(h_1^{q+1}) \tag{2.0.2}$$

$$\phi^2 = \phi^0 + \mathbf{k_1}h_2^q + O(h_2^{q+1}), \tag{2.0.3}$$

restando 2.0.3 a la 2.0.2,

$$\phi^1 - \phi^2 = \mathbf{k_1}(h_1^q - h_2^q) + O(h_{1,2}^{q+1}) \longrightarrow \mathbf{k_1} = \frac{\phi^1 - \phi^2}{h_1^q - h_2^q} \tag{2.0.4}$$

introduciendo este resultado en la ecuación 2.0.2, se tiene

$$\phi^1 = \phi^0 + \frac{\phi^1 - \phi^2}{h_1^q - h_2^q} h_1^q \tag{2.0.5}$$

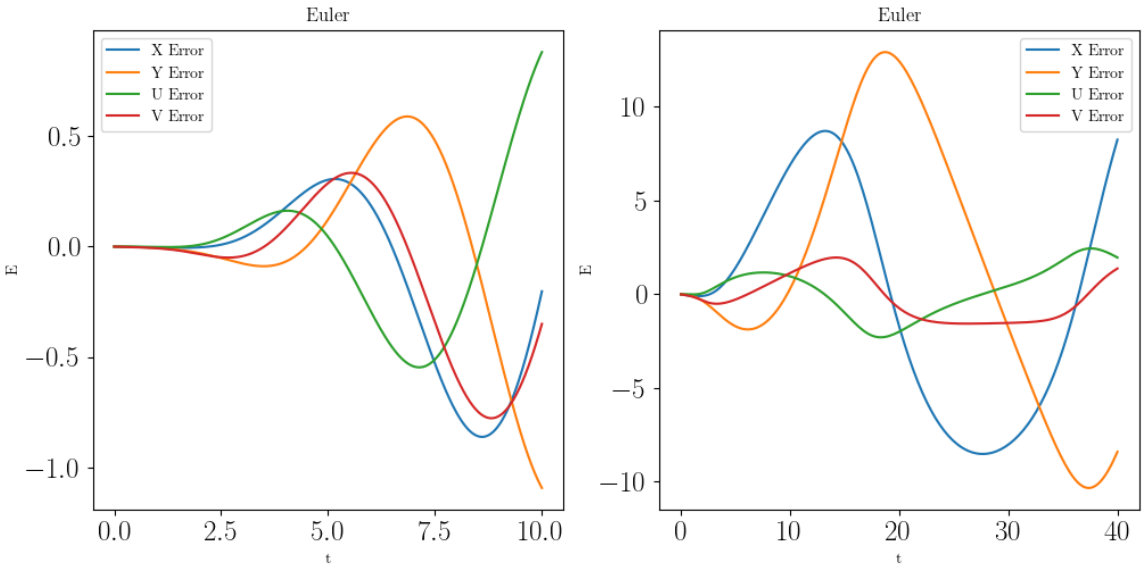
y definiendo finalmente el error cometido en una simulación numérica como la diferencia entre el resultado de la misma y el exacto, se obtiene

$$\mathbf{E^1} = \phi^0 - \phi^1 = \frac{\phi^2 - \phi^1}{h_1^q - h_2^q} h_1^q \tag{2.0.6}$$

Si se eligen las mallas de tal forma que $h_2 = \frac{h_1}{2}$, (es decir de tal forma que para recorrer el mismo T si la simulación 1 da N pasos, la 2 tenga que dar $2N$), la expresión 2.0.7 resulta,

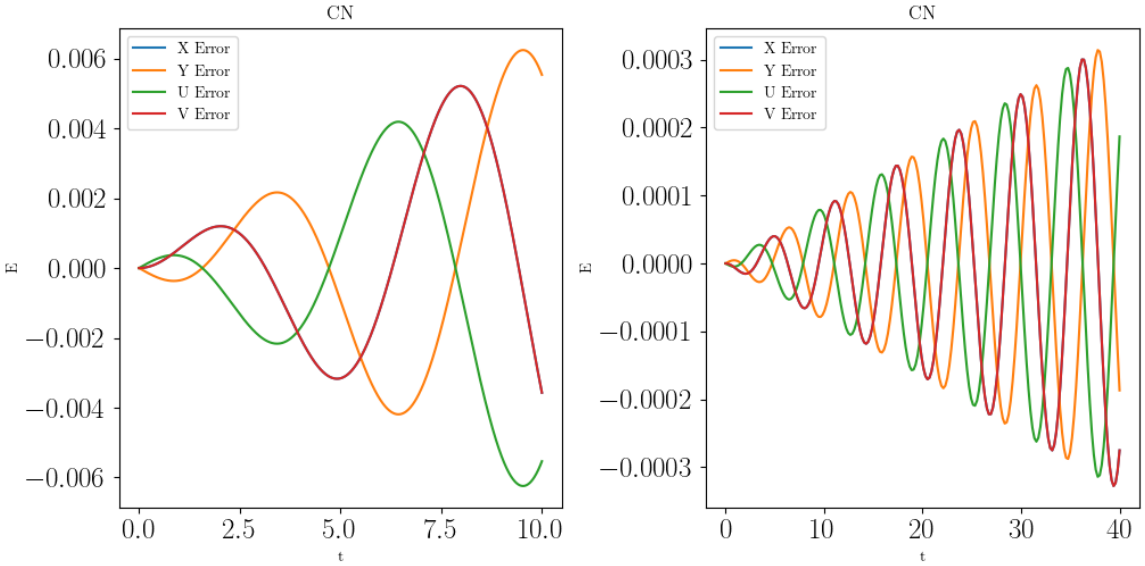
$$\mathbf{E^1} = \frac{\phi^2 - \phi^1}{1 - \frac{1}{2^q}} \tag{2.0.7}$$

Como se comentaba se ha aplicado este método a los diferentes esquemas temporales utilizados para integrar el problema de Cauchy planteado en el hito 2. Para ello se han realizado simulaciones de $10s$ con un numero de pasos $N = 1000$ ($dt = 0{,}01$). Los resultados se pueden ver a continuación, y como era de esperar, reiteran que se comentaba en el hito 2: El RK 4 y el CN tienen un error bastante bajo, del orden del paso del tiempo $\approx \frac{1}{100}$, mientras que el Euler tiene error mucho mayor, de orden unidad. Por su parte el Euler inverso diverge, ya que las soluciones del problema están fuera de la región de estabilidad, por lo que harían falta muchos más puntos.



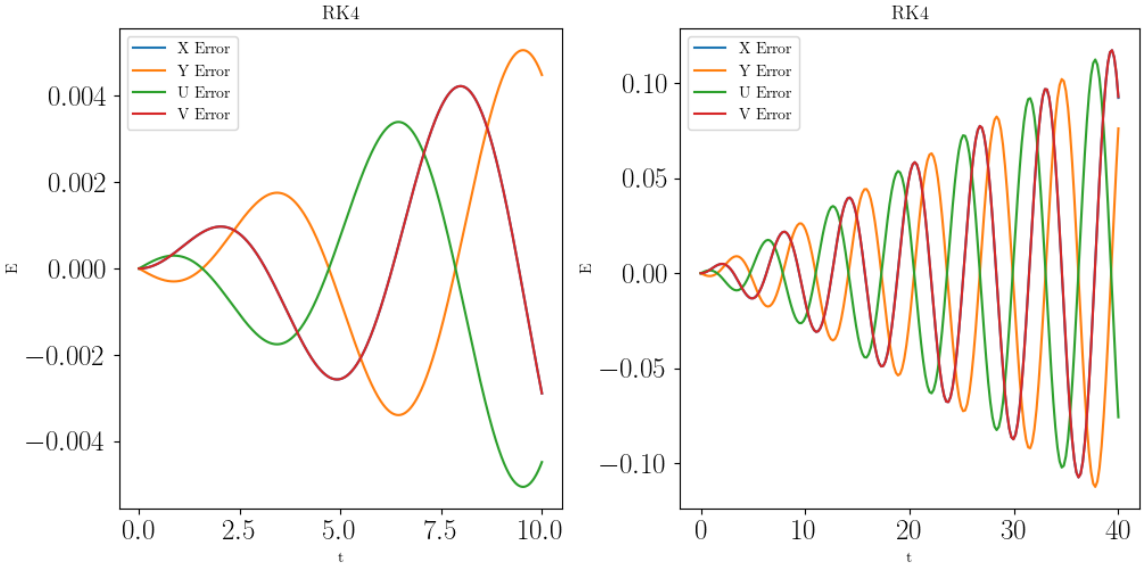
(a) Error para $T = 10$, $N = 1000$ (b) Error para $T = 40$, $N = 200$

Figura 2.1: Error del esquema de Euler



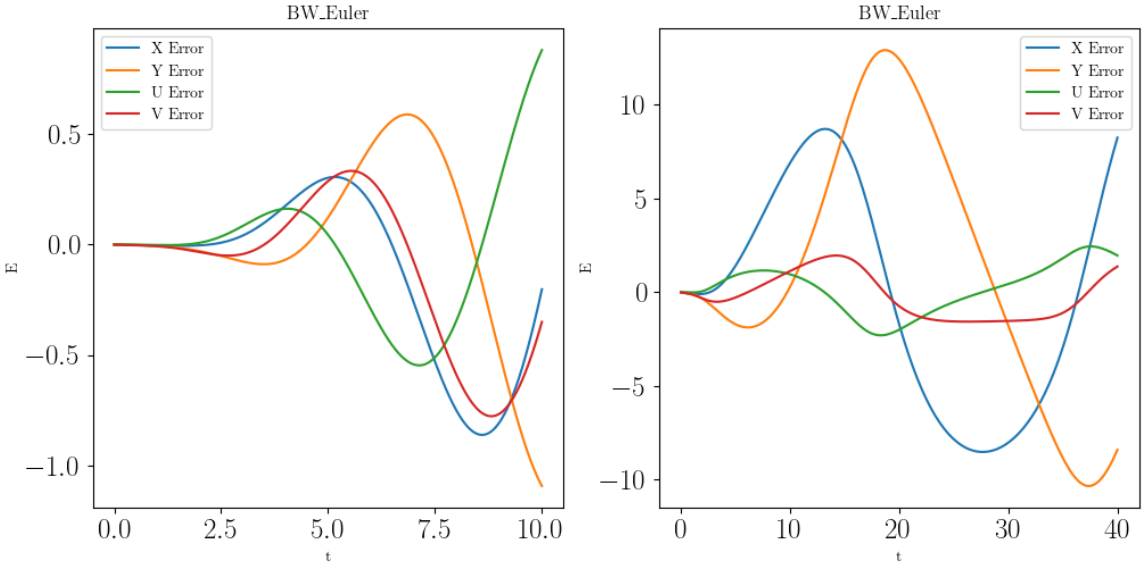
(a) Error para $T = 10$, $N = 1000$ (b) Error para $T = 40$, $N = 200$

Figura 2.2: Error del esquema de Crank-Nicolson



(a) Error para $T = 10$, $N = 1000$ (b) Error para $T = 40$, $N = 200$

Figura 2.3: Error del esquema de Runge-Kutta 4



(a) Error para $T = 10$, $N = 1000$ (b) Error para $T = 40$, $N = 200$

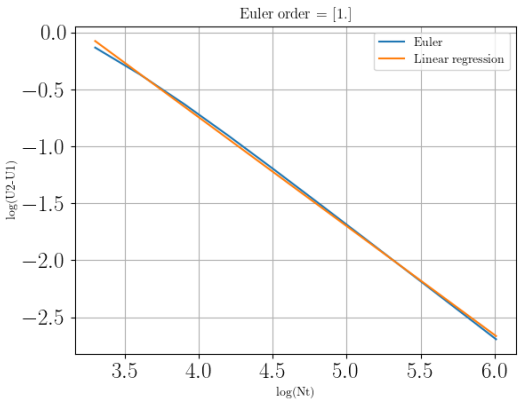
Figura 2.4: Error del esquema de Euler Inverso

3. Ratio de Convergencia

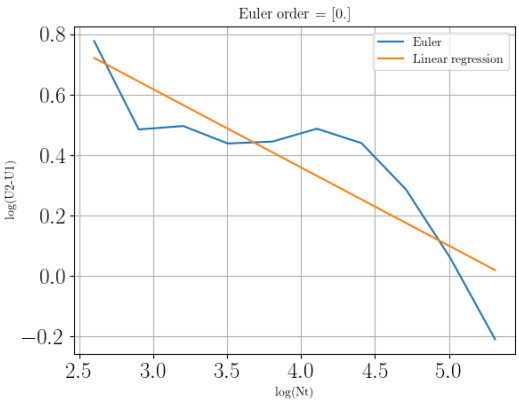
Al definir los parámetros de la extrapolación de Richardson se ha comentado que el parámetro de convergencia q de cada esquema temporal es el orden del esquema numérico. Mirando la expresión obtenida en 2.0.7 es fácil darse cuenta de que a mayor sea q, más bajo será el error, y por ende menos puntos harán falta para conseguir una buena integración.

Así, se ha representado el error $log(||U^{2N} - U^N||)$ para mallas que aumentan progresivamente su refinamiento ($log(N)$). La pendiente de la recta será por tanto el orden del esquema temporal. El orden obtenido se ha redondeado, pero en los casos de Euler tanto directo como inverso, la aproximación es un tanto tosca, (resulta $q = 0{,}6$ para el Euler).

Cuando el error está entorno a 10^{-12}, la precisión ya no puede subir más, porque nos topamos con el error propio de la computadora. Estos puntos no se han empleado en la regresión porque introduciríamos más ruido a la pendiente de la curva

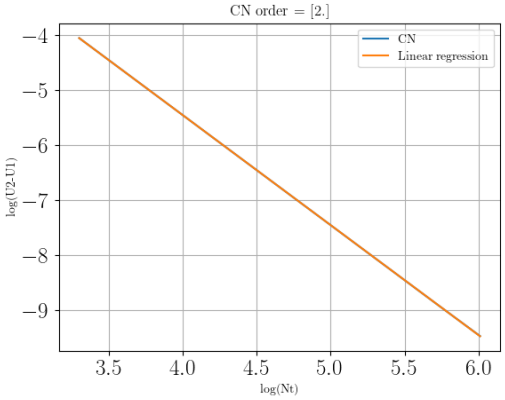


(a) $T = 10$, $N = 1000$

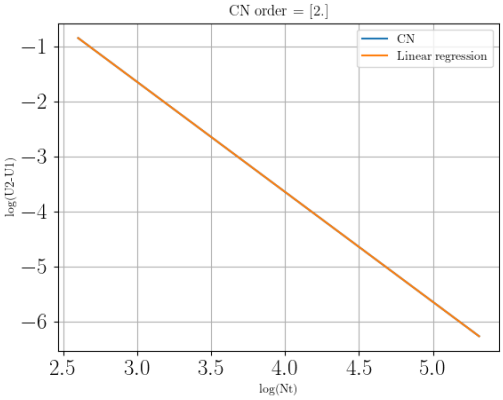


(b) $T = 40$, $N = 200$

Figura 3.1: Ratio de convergencia del esquema de Euler

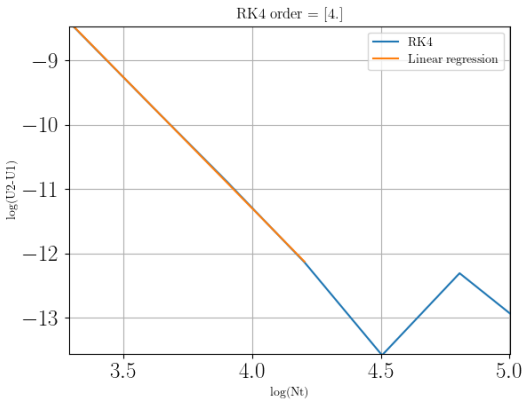


(a) $T = 10$, $N = 1000$

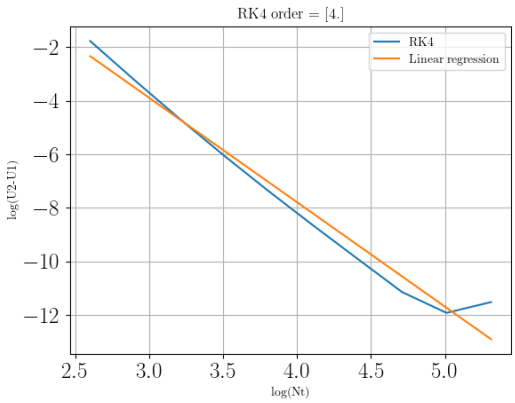


(b) $T = 40$, $N = 200$

Figura 3.2: Ratio de convergencia del esquema de Crank-Nicolson

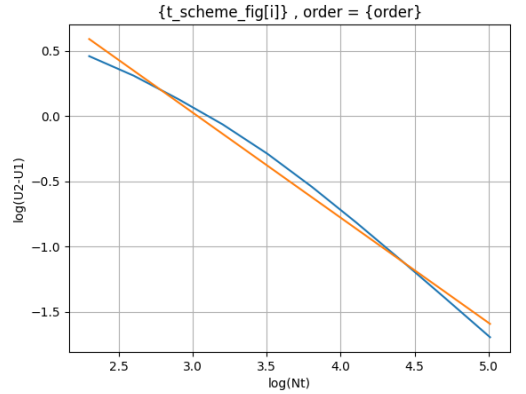


(a) $T = 10$, $N = 1000$

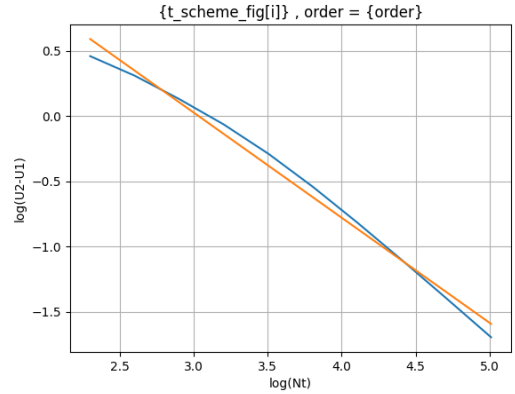


(b) $T = 40$, $N = 200$

Figura 3.3: Ratio de convergencia del esquema de Runge-Kutta 4



(a) $T = 10$, $N = 1000$



(b) $T = 40$, $N = 200$

Figura 3.4: Ratio de convergencia del esquema de Euler Inverso

4. Código

El código se ha desarrollado en un nuevo módulo encargado de la evaluación del error. Para ello llama a numpy, a sklearn y al módulo del problema de Cauchy:

```
from numpy import linspace, zeros, log10, round_
from numpy.linalg import norm
from sklearn.linear_model import LinearRegression
from resources.Cauchy_Problem import Cauchy
```

Algoritmo 4.1: Head of the Error$_{E}valuationModule$

La función que lleva a cabo la extrapolación de Richardson

```
 def Error_Cauchy_Problem (Time_Scheme, Scheme_Order, F, U0, t):


 # Computes the error of the obtained solution of a Cauchy problem
 # using the Richardson extrapolation:
 # For this, the program solves the cauchy Problem on two different temporal grids
 # (t1 & t2), with different dt but the same final T.

 Nt = len(t)
 NU = len(U0)
 E = zeros([Nt, NU])

 t_2 =linspace(0, t[Nt - 1], 2*Nt) #I already had a grid (t). This one (t_2) has twice the amount of elements (Nt_2 =2*Nt), thus dt_2 = dt/2

 U_t = Cauchy(F, Time_Scheme, U0, t)
 U_t_2 =  Cauchy(F, Time_Scheme, U0, t_2)

 for i in range (Nt):
     E[i,:] = (U_t_2[2*i, :] - U_t[i, :]) / (1 - 1 / (2** Scheme_Order))

 return E
```

Algoritmo 4.2: Función de la Extrapolación de Richardson

Y por último la que evalúa el ratio de convergencia:

```
 def Convergence_Rate(Time_Scheme, F, U0, t):
 #determines the error of the numerical solution as a function of the number of time
 #steps N. This subroutine internally integrates a sequence of refined dt_i and, by
 #means of the Richardson extrapolation, determines the error.

 k = 10 # Numero de ptos que cojo en la gráfica (N1, N2=2*N1, N3 =2*N2...) (numero de cauchy problems que te tienes que hacer)
 Nt = len(t)
 T = t[Nt-1]
 NU = len(U0)

 E = zeros([Nt, NU])
 log_diff_U21 = zeros(k)
 log_Nt = zeros(k)
 lin_log_diff_U21 = zeros(k)
 lin_log_Nt = zeros(k)
 order = 0

 U_t = Cauchy(F, Time_Scheme, U0, t)


```

```
    for i in range (k):

        Nt = 2*Nt
        t_2 =linspace(0, T, Nt)
        U_t_2 =  Cauchy(F, Time_Scheme, U0, t_2)


        log_diff_U21[i] = log10(norm(U_t_2[int(Nt - 1), :] - U_t[
    int(0.5*Nt - 1), :]))
        log_Nt[i] = log10(Nt)

        U_t = U_t_2

        for j in range(k):

         if (abs(log_diff_U21[j]) > 12):

             break

    l=min(j, k)


    reg= LinearRegression().fit(log_Nt[0:j+1].reshape((-1, 1)),
    log_diff_U21[0:j+1])
    order = round_(abs(reg.coef_),1)

    lin_log_Nt = log_Nt[0:j+1]
    lin_log_diff_U21 = reg.predict(log_Nt[0:j+1].reshape((-1, 1)))



    return [log_diff_U21, log_Nt, lin_log_diff_U21, lin_log_Nt,
    order]
```

Algoritmo 4.3: Ratio de Convergencia

Se ha tratado de hacer todo ello de la forma más limpia posible. Finalmente, se ha llamado iterativamente a estas funciones a traves del Milestone 3