


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone III

Ampliación de Matemáticas

Autor:

Miguel del Fresno Zarza

Índice

1. Introducción

Este trabajo tiene como objetivo estudiar el error que se produce para los esquemas de Euler, Runge Kutta de cuarto orden, Crank Nicolson y Euler inverso. Para ello, se ha calculado el error mediante la extrapolación de

2. Marco teórico

2.1. Extrapolación de Richardson

Para poder calcular el error mediante el método de Richarson, se calcula la función U del problema en cuestión en dos intervalos de tiempo. El primer mallado tiene un intervalo de Δt, mientras que el segundo mallado contiene un intervalo de 2Δt. Así mismo, el error se calcula de la siguiente manera:

$$E = \frac{||U_1^n - U_2^n||}{1 - 2/2^q}$$

2.2. Ratio de convergencia

El ratio de convergencia para cada uno de los esquemas viene dado por (OΔ^q), donde q es el orden del esquema. Por lo que el error tenderá a 0 más rápido cuanto menor sea el Δt.

Para estudiar el ratio de convergencia en cuestión, se emplea la siguiente ecuación:

$$log(|E^n|) = C - q \cdot log(N)$$

Donde, para observar el ratio de convergencia, se obtiene log(|E^n|) en función de log(N).

3. Resultados

A continuación se muestran los resultados para el cálculo del error y de la convergencia para cada uno de los métodos. Para cada uno de los esquemas se mostrarán los resultados con un $\Delta t = [0{,}001, 0{,}01, 0{,}1]$ y un tiempo total de 20 segundos

3.1. Extrapolación de Richardson

Euler

A continuación se muestran los resultados para Euler.

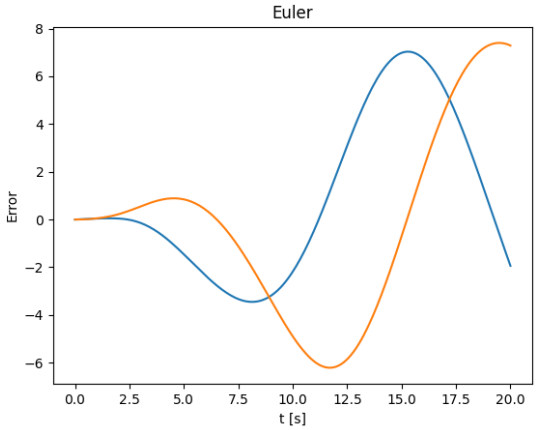


Figura 1: $\Delta t = 0{,}1$

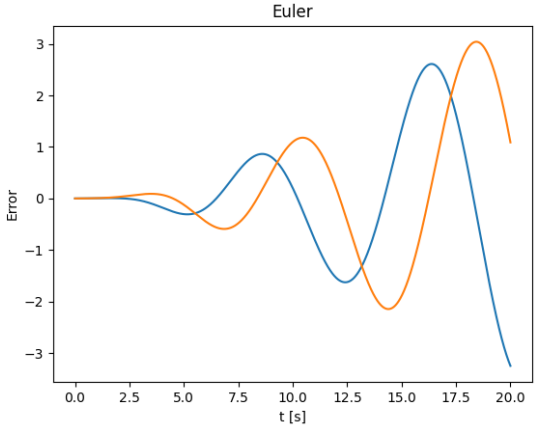


Figura 2: $\Delta t = 0{,}01$

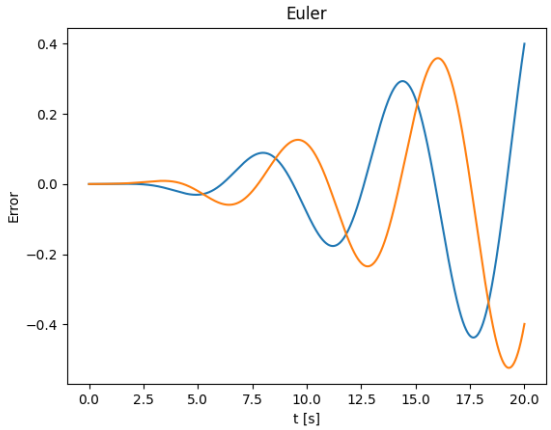


Figura 3: $\Delta t = 0{,}001$

Runge Kutta de orden 4

A continuación se muestran los resultados para Runge Kutta de orden 4.

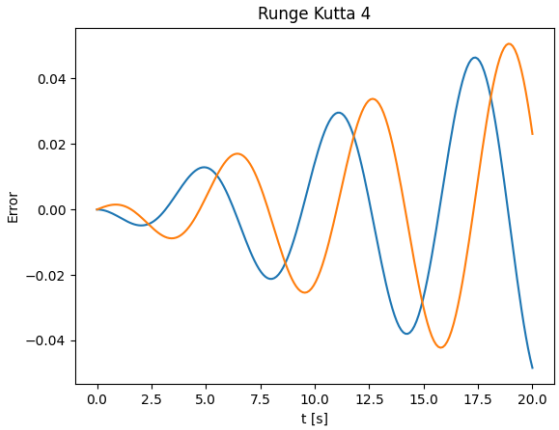


Figura 4: $\Delta t = 0{,}1$

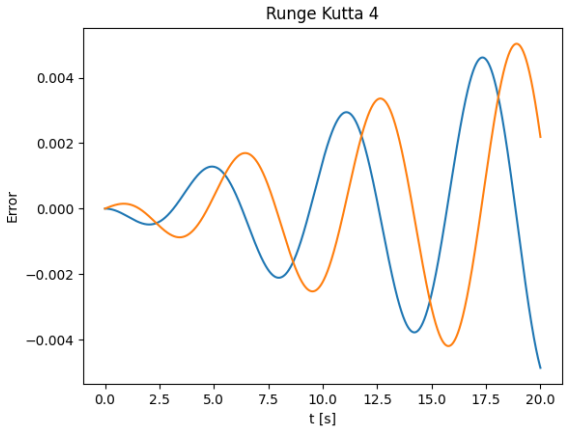


Figura 5: $\Delta t = 0{,}01$

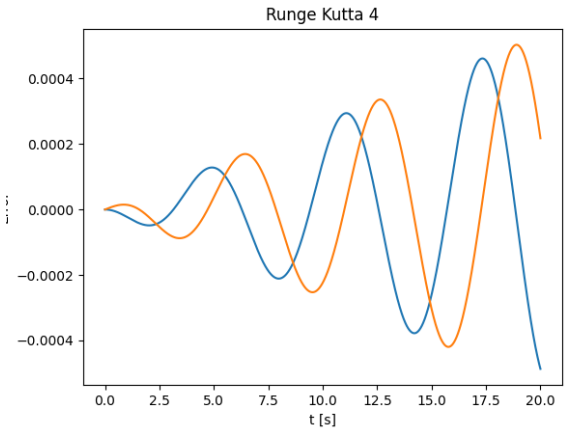


Figura 6: $\Delta t = 0{,}001$

Euler Inverso

A continuación se muestran los resultados para Euler Inverso.

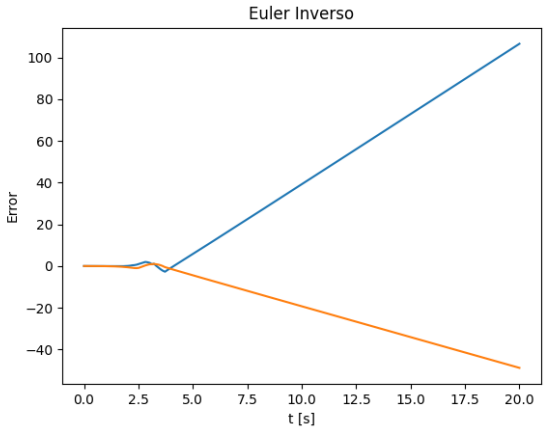


Figura 7: $\Delta t = 0{,}1$

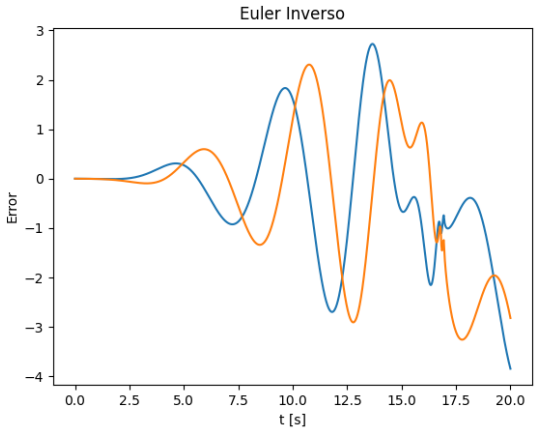


Figura 8: $\Delta t = 0{,}01$

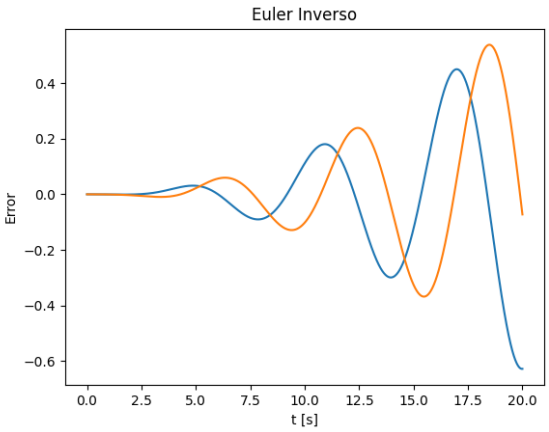


Figura 9: $\Delta t = 0{,}001$

Crank Nicolson

A continuación se muestran los resultados para Crank Nicolson.

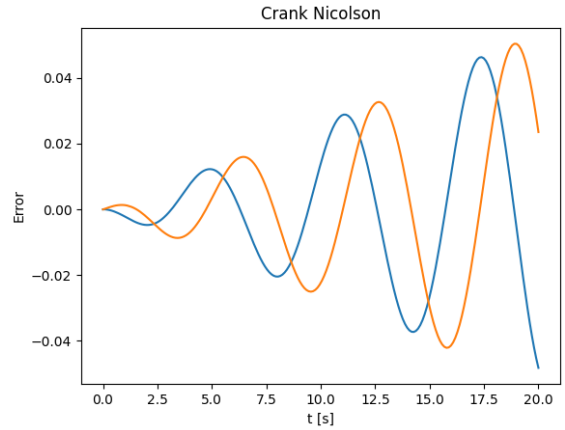


Figura 10: $\Delta t = 0{,}1$

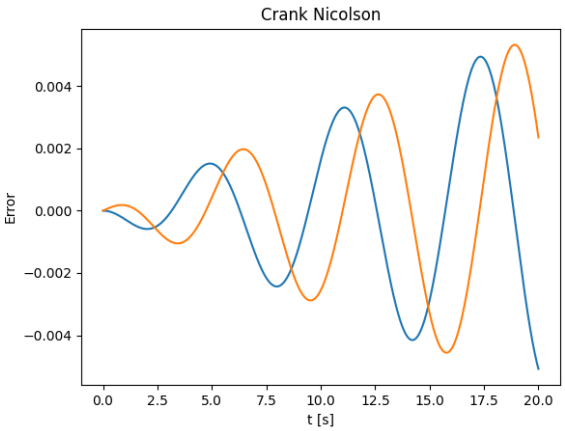


Figura 11: $\Delta t = 0{,}01$

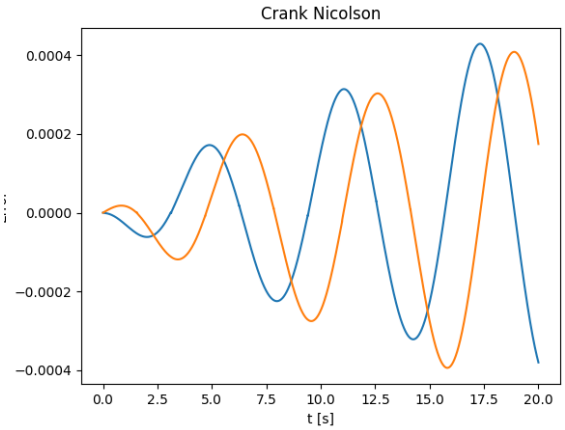
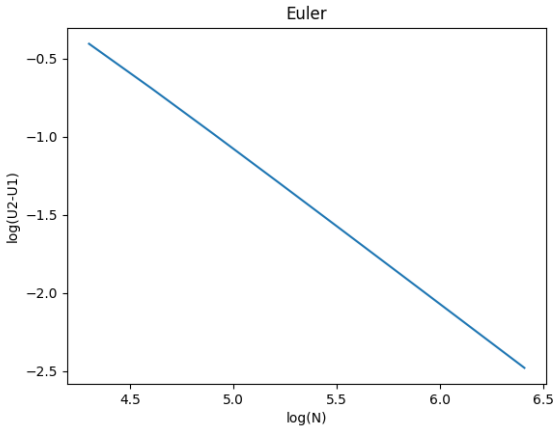


Figura 12: $\Delta t = 0{,}001$

3.2. Convergencia

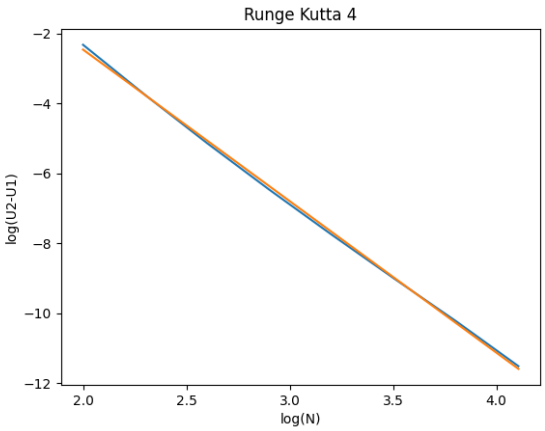
Euler

A continuación se muestran los resultados para Euler.



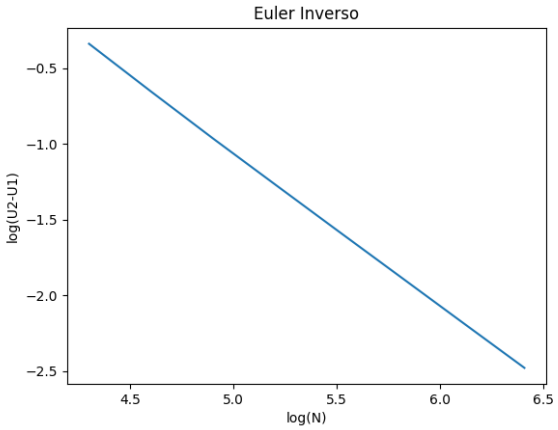
Runge Kutta de orden 4

A continuación se muestran los resultados para Euler.



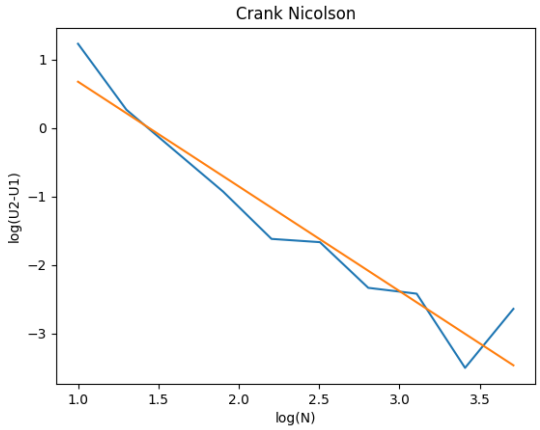
Euler Inverso

A continuación se muestran los resultados para Euler Inverso.



Crank Nicolson

A continuación se muestran los resultados para Crank Nicolson.



4. Código

Para el desarrollo del hito, se han reutilizado los archivos de los hitos anteriores, donde se calculan las órbitas con los cuatro esquemas diferentes.

Por otra parte, se tiene otro archivo llamado Error.py donde se hacen las operaciones correspondientes para el cálculo del error mediante la extrapolación de Richardson y la convergencia.

Finalmente, se encuentra el archivo llamado Milestone III.py, el cual se trata del main donde se llama a las funciones del archivo de Error.py para obtener los resultados.

4.1. Extrapolación de Richardson

A continuación se muestra el código correspondiente al cálculo del error mediante la extrapolación de Richardson.

```
def Richardson_error(F, Scheme, t, T, n, U0, q):

    t1 = t
    t2 = linspace(0,T,2*n)

    U1 = Cauchy(F, t1, U0, Scheme)
    U2 = Cauchy(F, t2, U0, Scheme)

    E = zeros((n, size(U0)))

    for i in range(n):
        E[i,:] = (U1[i,:] - U2[2*i,:])/(1 - 1/(2**q))

    return E
```

Como se muestra en la figura, hay unos datos de entrada en los que se tiene el tiempo de la órbita, el número de pasos en el que se divide el tiempo, las condiciones de contorno y el esquema con el que se calcula la órbita.

Dentro de la función Richardson_error, se hace un segundo mallado con el doble de puntos de mallado en el tiempo. Después, se hace el cálculo de U en cada mallado y finalmente se realiza el cálculo del error local mediante la diferencia de los resultados en cada mallado.

Finalmente, de la función se obitene el error para cada esquema.

4.2. Convergencia

A continuación, se muestra el código en el cual se encuentra el cálculo de la convergencia del error en cada uno de los esquemas.

```
def convergence(F, scheme, t, T, n, U0, q):

    t1 = t

    m = 8
    log_E = zeros(m)
    log_N = zeros(m)
    Error = zeros(m)

    for i in range(0, m):

        t1 = linspace(0, T, n)
        t2 = linspace(0, T, 2*n)
        U1 = Cauchy(F, t1, U0, scheme)
        U2 = Cauchy(F, t2, U0, scheme)

        Error[i] = norm(U1[n-1, :] - U2[2*n-1, :])
        log_E[i] = log10(Error[i])
        log_N[i] = log10(n)

        U1 = U2
        n = 2*n


    q = linregress(log_N, log_E)

    return log_N, log_E
```

Como se puede apreciar, sl igual que para el cálculo del error, se realiza el segundo mallado con el doble de puntos. Se destaca que se encuentra el valor m, el cual es el número de puntos que se quiere representar en las gráficas de la convergencia. Cuanto más grande es este número, mayor es la carga computacional ya que requiere hacer más cálculos, los cuales en este caso ya de por sí suponen una gran carga computacional, por lo que aumentar este número puede suponer estar corriendo el código durante más de media hora en total (tiempo que también se ve incrementado por el número de puntos que hay en el mallado del tiempo). Es por esto que en este caso se ha dejado en 8 puntos ya que supone mucho tiempo de cálculo.

Por otra parte, se muestra el cálculo de los logaritmos para el cálculo de la convergencia y finalmente la función devuelve estos logaritmos como resultado.

4.3. Main

A continuación, se muestra el código principal donde se llama a todas las funciones desarrolladas anteriormente y se generan las gráficas correspondientes.

```
U0 = array([1, 0, 0, 1])
T = 20
dt = 0.001
n = int(T/dt)
t = linspace(0, T, n)

schemes = [Euler, RK4, Euler_Inv, Crank_Nicolson]
q_schemes = [1, 4, 1, 2]
tit_scheme = ['Euler', 'Runge Kutta 4', 'Euler Inverso', 'Crank Nicolson']


i = 0

for scheme in schemes:
    q = q_schemes[i]

    Error = Richardson_error(Kepler, scheme, t, T, n, U0, q)

    plt.plot(t, Error[:,0])
    plt.plot(t, Error[:,1])
    plt.title(tit_scheme[i])
    plt.xlabel('t [s]')
    plt.ylabel('Error')
    plt.show()

    i = i + 1

s = 0
for scheme in schemes:
    q = q_schemes[s]

    log_N, log_E = convergence(Kepler, scheme, t, T, n, U0, q)

    plt.plot(log_N, log_E)
    plt.title(tit_scheme[s])
    plt.xlabel('log(N)')
    plt.ylabel('log(U2-U1)')
    plt.show()

    s = s + 1
```

En este código, como se puede apreciar, simplemente se establecen las condiciones de contorno, el mallado que se realiza, los esquemas con su orden y nombre y las funciones para obtener el error local y la convergencia.

5. Conclusiones

5.1. Extrapolación de Richardson

A continuación se comentarán las conclusiones acerca de la extrapolación de Richardson. Como principal observación acerca de este apartado, se destaca que el error calculado es el local en cada instante, no el global, por lo que en cada uno de los casos el error que se produce puede ir acumulándose y generar un gran error una vez pasada una cierta cantidad de tiempo.

Euler

En el cálculo del error para el esquema de Euler, como ya se pudo ver en anteriores hitos, cuando se disminuye el incremento del tiempo, el error que se produce es mucho menor y por lo tanto las soluciones son más exactas. De todas formas, el error aumenta con el paso del tiempo.

Runge Kutta de orden 4

Este esquema presenta un incremento de error menor con el paso del tiempo. A demás, como se aprecia, el error generado es mucho menor que para Euler con los mismos mallados. Esto se obtiene a cambio de una mayor carga computacional.

Euler Inverso

Este esquema presenta problemas para incrementos elevados de tiempo como se puede observar en las dos primeras gráficas. Cuando el incremento de tiempos e vuelve lo suficientemente pequeño, este esquema presenta un mejor comportamiento, el cual es del mismo orden que de Euler del primer apartado.

Cabe destacar que probablemente el comportamiento de este esquema no sea el adecuado para incrementos de tiempo elevados debido al propio código, ya que presenta problemas en el número de iteraciones en la función de Newton, las cuales se han tenido que aumentar para que pudiese finalizar los cálculos sin provocar un error en el programa.

Crank Nicolson

Como se observa en las gráficas, este esquema es el que menor error presenta y por lo tanto ofrece resultados más fiables. Al igual que para Runge Kutta, este esquema presenta una carga computacional mayor.

5.2. Extrapolación de Richardson

En este apartado se ha representado el ratio de convergencia de cada uno de los esquemas. Este apartado ha conllevado una gran carga computacional, la cual provoca que los tiempos de cálculo sean muy elevados.

Euler

Una vez localizado el número de puntos de mallado adecuado, la convergencia se ajusta a la pendiente de 1 que le corresponde con el orden del esquema.

Runge Kutta de orden 4

La pendiente que adopta el esquema es aproximadamente 4, que se ajusta al orden del esquema.

Euler Inverso

Al igual que para el esquema de Euler, este esquema termina por adoptar la misma pendiente.

Crank Nicolson

Este esquema presenta una peor adaptación a la convergencia correspondiente que debería de tener una pendiente de 2. Esto probablemente venga provocado también por algún tipo de error con la función de Newton de Python ya que ha presentado problemas con el número de iteraciones.

5.3. Comentarios

Finalmente, con este hito se ha aprendido acerca del error y ratio de convergencia de cada uno de los esquemas.

En cuanto al código, se ha comenzado a tener una mayor soltura a la hora de escribir el código y que no se produzcan tantos errores como anteriormente. Por la parte de la función de Newton, se han dado bastantes problemas, por lo que para próximos hitos se tratará de solucionar este problema creando una función propia.

Finalmente, los tiempos de computación de los resultados han resultado elevados, por lo que en estos casos puede resultar conveniente buscar alternativas para acelerar los procesos de cálculo y poder reducir estos tiempos y no perder demasiado tiempo en espera.