


Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

MÁSTER UNIVERSITARIO EN SISTEMAS ESPACIALES

MILESTONE 3

Ampliación de Matemáticas I

23 de octubre de 2022

Autor:
Alberto García Rincón

Índice

1. Introducción

Como continuación del ejercicio resuelto la semana pasada, en este trabajo se ha implementado el método de extrapolación de Richardson para evaluar el el error de los distintos métodos de cálculo numérico que ya que habían programado en el ejercicio anterior. Posteriormente se ha programado una función que evalúa el ratio de convergencia de dichos métodos de cálculo numérico.

2. Código de Python

La estructura de división de cada función en archivos distintos se mantiene para este trabajo, puesto que es la forma correcta de programar.

Se ha creado dos funciones para calcular el error numérico mediante la extrapolación de Richardson y otra para calcular el ratio de convergencia de los respectivos métodos de cálculo que ya se habían implementado en el trabajo anterior. Todas las funciones relativas a la creación de las gráficas están en otro archivo distinto denominado *plots.py*.

Solo se mostrarán y se realizará un breve análisis de las funciones nuevas creadas para este trabajo, así como las modificaciones o reestructura de código de algunos archivos.

2.1. *milestone_3.py*

Respecto al programa principal se ha reescrito el proceso de llamar al resto de funciones para calcular los distintos vectores de elementos que se piden para este trabajo. Ahora se ejecuta un bucle que recorre los distintos métodos implementados y en cada paso se calculan las distintas instancias (vector de estado, error y ratio de convergencia) que se piden.

```python
#Calculus
for i in range(len(mets)):
    if mets[i] == 'Euler':
        method = met.explicit_euler
        q = 1
    elif mets[i] == 'RK4':
        method = met.runge_kutta4
        q = 4
    elif mets[i] == 'Inverse Euler':
        method = met.inverse_euler
        q = 1
    else:
        method = met.crank_nicolson
        q = 2

    U[i,:] = cp.cauchy_problem(U0, t, orb.orbits, method)       #Cauchy solver orbits
    En[i,:] = orb.orbit_energy(U[i,:], t)                       #Energy orbit
    E[i,:] = re.richardson_error(U0, t, orb.orbits, method, q)  #Error Richardson
    CR[i,:] = cr.convergence_rate(U0, t, orb.orbits, method, m) #Convergence Rate

#Graphics
plo.plot_positions(U, mets, dt)
plo.plot_energy(En, mets, t, dt)
plo.plot_error(E, mets, t, dt)
plo.plot_met_error(E, mets, t, dt)
plo.plot_conv_rate(CR, mets, dt)
plo.plot_comparision_cr(CR, mets, dt)
```

Figura 1: *milestone3.py*.

2.2. *richardson_error.py*

Esta función tiene como argumentos el vector de estado inicial, el vector de tiempo donde se va a implementar la simulación, la función que define el comportamiento del objeto de estudio, el método que se va a usar para el cálculo de su comportamiento y el orden del método numérico.

La función nos devuelve una matriz que contiene todos los errores estimados para cada intervalo de tiempo para cada una de las variables de estado. El error asociado a cada variable esta definido en una fila.

```python
def richardson_error(U0, t, f, method, q): #(U0, time, function, method, order)
    N = len(t)
    E = np.zeros([len(U0), N])
    e_pos = np.array(np.zeros([1, N])) #error position module

    t1 = t
    t2 = np.linspace(0, t[N-1], 2*N)

    U1 = cp.cauchy_problem(U0, t1, f, method)
    U2 = cp.cauchy_problem(U0, t2, f, method)

    for i in range(N):
        E[:, i] = (U2[:,2*i] - U1[:,i]) / (1-1/(2**q))
        e_pos[0, i] = (E[0, i]**2 + E[1, i]**2)**0.5

    E = np.insert(E, 4, e_pos, axis=0)

    return E

```

Figura 2: *richardson_error.py*.

Se ha añadido una fila más a la matriz que devuelve la función: el módulo del error de la posición, es decir, el módulo entre las dos variables de estado que definen la posición para cada intervalo de tiempo, calculado en la misma función como: *e_pos*.

2.3. *convergence_rate.py*

Esta función necesita como argumentos el vector de estado inicial, el vector que define el tiempo de la simulación, la función que define el comportamiento del objeto de estudio, el método de cálculo numérico y el número de puntos que se van a utilizar para calcular la convergencia.

```python
def convergence_rate(U0, t, f, method, m): #(U0, time, function, method, nº points)
    log_E = np.zeros(m)
    log_N = np.zeros(m)
    N = len(t)-1
    tf = t[N-1]

    t1 = t
    U1 = cp.cauchy_problem(U0, t1, f, method)

    for i in range(m):
        N = 2*N
        t2 = np.linspace(0, tf, N+1)
        U2 = cp.cauchy_problem(U0, t2, f, method)

        E = np.linalg.norm(U2[:,N] - U1[:,int(N/2)])
        log_E[i] = np.log10(E)
        log_N[i] = np.log10(N)

        t1 = t2
        U1 = U2

    return [log_N, log_E]

```

Figura 3: *convergence_rate.py*.

3. Resultados de los métodos

Se ha fijado un tiempo total de simulación de 30 segundos se han realizado varias simulaciones con distintos Δt para comprobar que el código funciona correctamente. Aquí solo se muestran algunas de esas simulaciones.

3.1. Error por extrapolación de Richardson

Este método permite la estimación del error de cálculo que se obtiene utilizando un método de resolución. Para este trabajo se ha calculado el error utilizando los cuatro métodos de resolución siguientes: Euler explícito, Runge-Kutta de orden 4, Euler inverso y Crank-Nicholson. El método consiste en evaluar la solución en dos mallas temporales con diferente intervalo de tiempo, una de ellas con el mismo intervalo que la resolución con el método de cálculo y la otra con un intervalo la mitad del anterior. La ecuación que define la estimación del error por la extrapolación de Richardson es:

$$E = \frac{U^{2N} - U^{N}}{1 - \frac{1}{2^q}} \tag{1}$$

siendo q el orden de integración del método de cálculo utilizado.

3.1.1. Simulaciones con: $\Delta t = 0{,}1s$

Como se puede observar en la figura de abajo con un intervalo de integración alto, se obtienen errores considerables. La tendencia creciente del error en la posición en todos los métodos es notable puesto que según se va avanzando en el tiempo de simulación se va acumulando el error de la integración anterior.

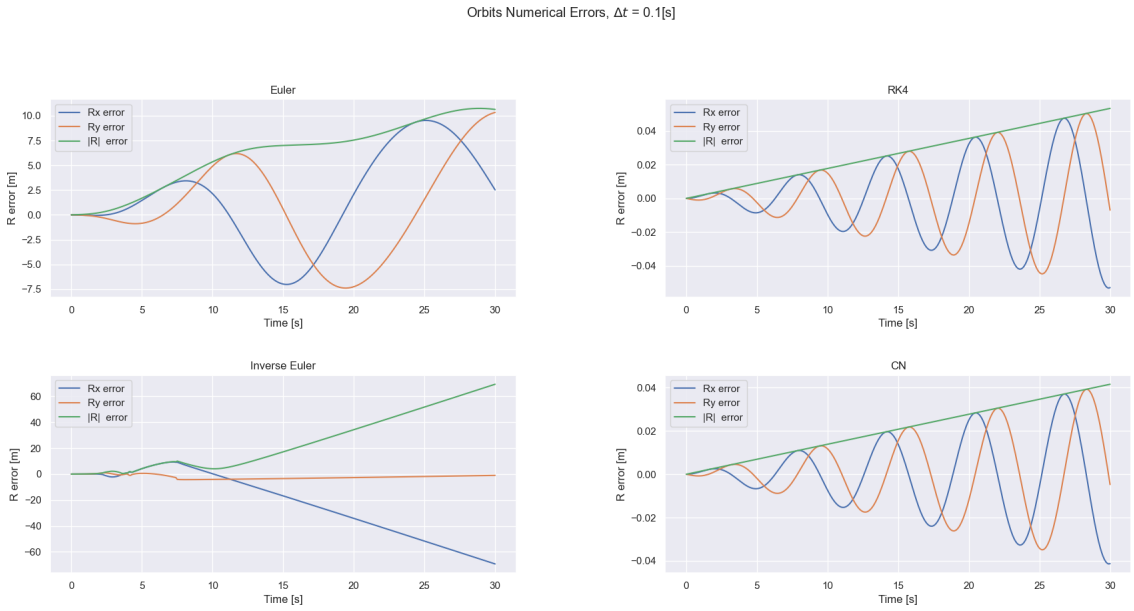

Figura 4: Errores de posición de los distintos métodos.

En la siguiente figura se observa una comparación entre los módulos de error de posición estimados para los distintos métodos de cálculo.

Cabe mencionar que con los métodos de orden superior (Runge-Kutta y Crank-Nicholson) se obtienen errores de posición menores que con el método de Euler. En el caso del método de Euler inverso, el error se dispara a los 10 segundos de la simulación debido a que en ese tiempo ha convergido su solución a cero y no estaría funcionando correctamente, para solucionarlo se debe disminuir el intervalo de tiempo con el que se realiza la simulación.

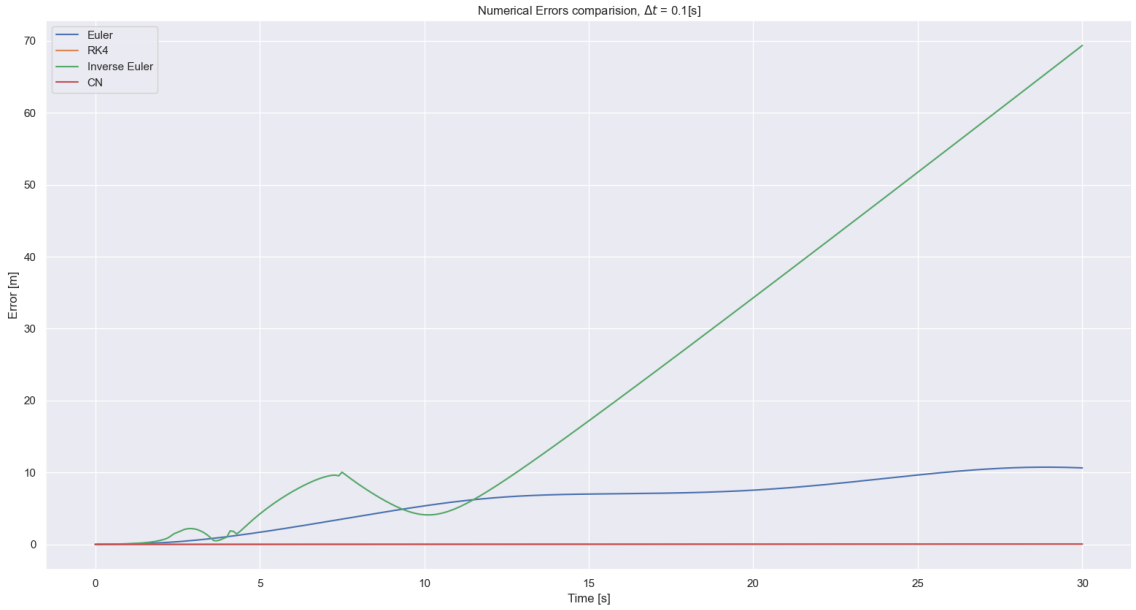

Figura 5: Comparación del módulo de error de la posición entre los distintos métodos.

3.1.2. Simulaciones con: $\Delta t = 0{,}001s$

En esta simulación se obtienen mejoras relativas de cada método respecto a la simulación anterior. En la figura siguiente se han representado los errores estimados de posición a lo largo del tiempo de simulación.

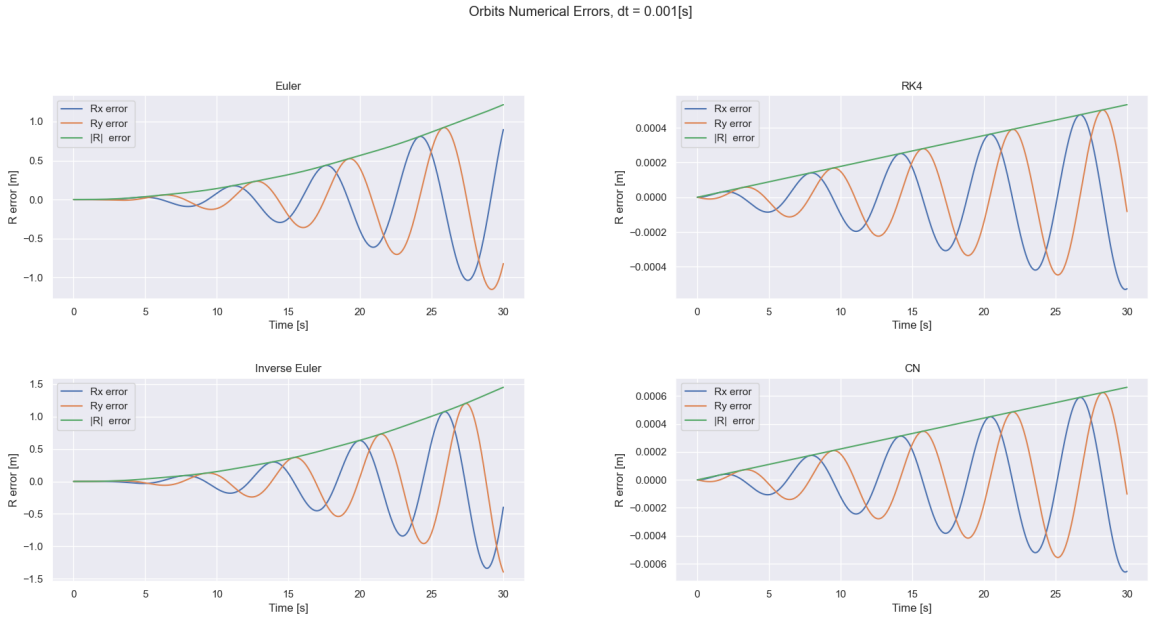

Figura 6: Errores de posición de los distintos métodos.

En esta otra figura se comparan los módulos de error de posición estimados. En este caso, al igual que en el anterior, los métodos de Crank-Nicholson y Runge-Kutta generan menos error que los métodos de Euler y de Euler inverso.

Para esta simulación, con el intervalo de tiempo utilizado ($\Delta t = 0{,}001s$), ha quedado solucionado el problema de la anterior simulación con el método de Euler inverso.

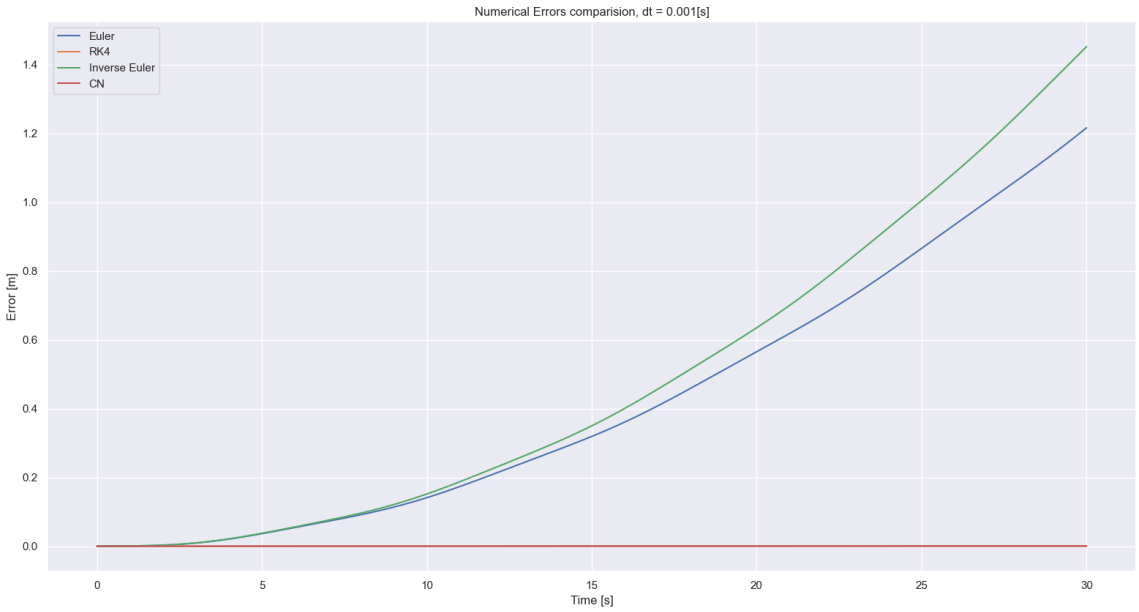

Figura 7: Comparación del módulo de error de la posición entre los distintos métodos.

3.2. Ratio de convergencia

Permite obtener el orden de integración de un método de cálculo numérico. Se representa gráficamente el logaritmo del error estimado con el método de la extrapolación de Richardson $log(|U^{2N} - U^N|)$ frente al logaritmo del número de puntos de integración $log(N)$. La pendiente de dicha gráfica es el orden de integración práctico.

3.2.1. Simulaciones con: $\Delta t = 0{,}1s$

Se observa que para simulaciones con incrementos de tiempo elevados el orden de integración se aleja del teórico, sobre todo en los métodos de Euler y de Euler inverso.

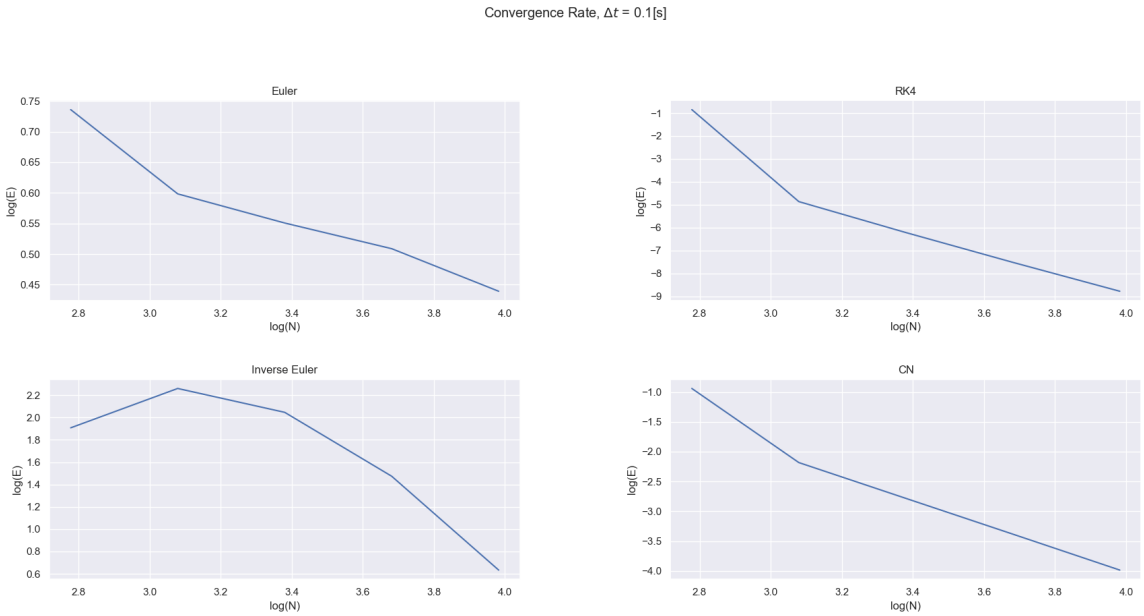

Figura 8: Ratio de convergencia de los distintos métodos.

3.2.2. Simulaciones con: $\Delta t = 0{,}001s$

Conforme se disminuye el Δt, los puntos obtenidos forman una recta más definida cuya pendiente corresponde con el orden del esquema numérico.

Para el método de Euler se puede calcular la pendiente de la recta de la figura de abajo, siendo aproximadamente -1, correspondiéndose en valor absoluto con el orden del método.

Para el resto de métodos se pueden obtener las pendientes de las rectas y se obtendrían conclusiones similares, pero con a su orden de integración.

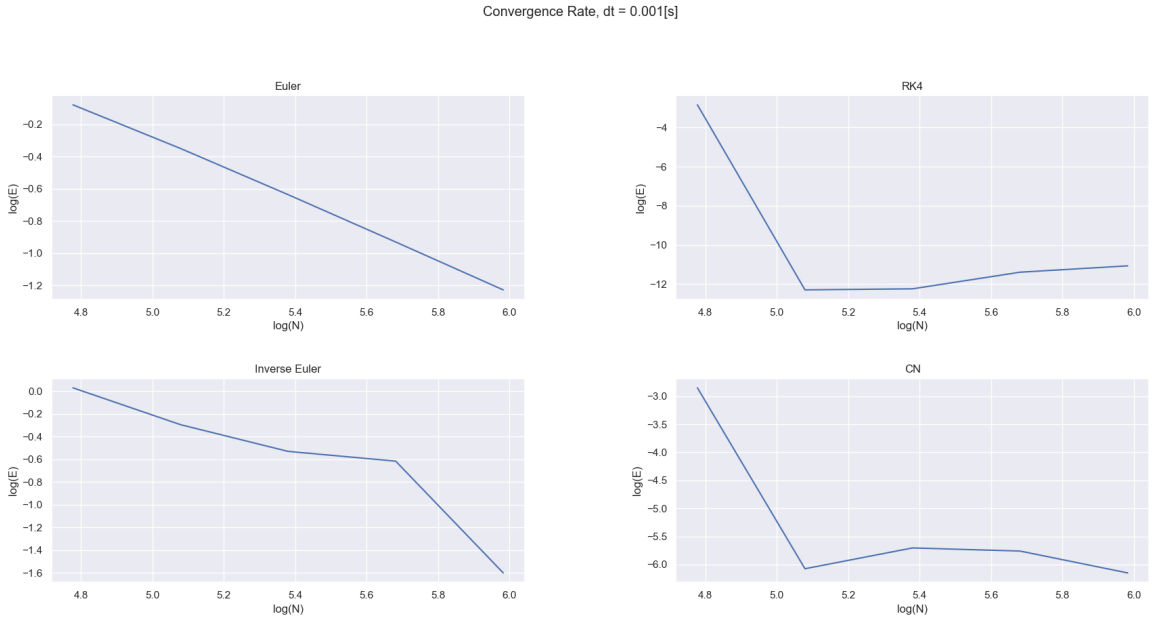

Figura 9: Ratio de convergencia de los distintos métodos.