Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone 3: Estimación del error de soluciones numéricas

Ampliación de Matemáticas I:
Cálculo numérico

23 de octubre de 2022

Autor:

- Daniel Cerón Granda

ÍNDICE

PÁGINA

1.Introducción

El método de extrapolación de Richardson es una técnica numérica para hacer que una secuencia convergente se convierta en otra que converja más rápido. Se puede utilizar para mejorar los resultados numéricos obtenidos con una estimación previa. Para este informe, se estudiará cómo utilizar este método para obtener la estimación del error cometido utilizando un método numérico, además de utilizarlo para obtener el orden del esquema.

Se considera que, si el resultado obtenido al aplicar un método numérico no difiere demasiado del resultado real, se puede realizar un desarrollo en serie de Taylor en torno al resultado real:

$$\phi(h) = \phi^0 + K_1 * h^q + 0(h^{q+1})$$

Ecuación 1: Desarrollo en serie del resultado numérico

Siendo ϕ el resultado numérico obtenido, ϕ^0 el resultado real (a priori desconocido), h el paso temporal utilizado, K_1 una constante de integración a priori desconocida y q el orden de aproximación del sistema. Estudiando dos resultados numéricos con esta expresión con dos pasos de tiempo diferentes ($\phi_1 \rightarrow h_1$, $\phi_2 \rightarrow h_2$), y restando las expresiones, se puede llegar a la expresión de una estimación del error cometido despreciando los términos de orden superior a h^q:

$$E^i \cong \frac{\phi_2 - \phi_1}{h_1^q - h_2^q} * h_i^q$$

Ecuación 2: Estimación del error para una solución

Donde i puede ser 1 o 2 según se quiera obtener la solución 1 o la solución 2. Por tanto, utilizando el método de Richardson, con dos mallas se puede obtener una aproximación al error cometido en ambas para un paso de tiempo suficientemente pequeño. Hay que tener en cuenta que los tiempos en los que se evalúa ϕ en ambas soluciones debe ser el mismo. Como varía el paso de tiempo en las diferentes soluciones, también se debe adaptar el número de iteraciones N utilizado en ambas, ya que el tiempo de evaluación será el producto de N y h. Una solución común para tener en cuenta este problema es utilizar en la segunda malla un intervalo de tiempo que sea la mitad del utilizado en la primera. Llamando U^n a la solución obtenida en la iteración n, la estimación del error sería:

$$E \cong \frac{U^{2n} - U^n}{1 - \frac{1}{2^q}}$$

Ecuación 3: Estimación del error para el uso de una segunda malla con un paso de tiempo la mitad que la primera

Por otro lado, un esquema es de orden q si su error global tiende a E según la expresión:

$$E = K * h^q + 0(h^{q+1})$$

Ecuación 4: Convergencia del error

Siendo K una constante desconocida a priori. A partir de esta expresión, tomando logaritmos y teniendo en cuenta que h se puede expresar como el producto del tiempo total de integración T y el número de iteraciones que se haya realizado N, se llega a la expresión:

$$\log(E) = \log(K) + q * \log(T) - q * \log(N)$$

Ecuación 5: Expresión logarítmica del error

Que se puede expresar como la ecuación de una recta: y = n + m*x, siendo y = log(E), n = log(K) + q*log(T), m = -q, x = log(N). A priori n lo desconozco, pero para obtener la pendiente se puede estudiar como si fuera n = 0, ya que sólo se desplaza la recta arriba o abajo. Como puedo desconocer el valor de E si no conozco q (ya que es lo que quiero obtener), se puede tomar logaritmos en la ecuación 3:

$$\log(E) = \log(U^{2n} - U^{n}) - \log\left(1 - \frac{1}{2^{q}}\right)$$

Ecuación 6: Expresión logarítmica del error

Que a priori depende de q. Sin embargo, el término de q es una constante para un mismo método, por lo que, de forma análoga a n, se puede obviar en el problema de determinación de la pendiente. Por tanto, a partir de dos valores de U^{2n}-U^{n} para dos valores de N, se puede determinar el orden de convergencia de un esquema numérico. Hay que tener en cuenta que para valores de log(N) pequeños el método falla (ya que la truncación del desarrollo en serie pierde mucha precisión) y para valores grandes (que den log(E) del orden de -14 o -15) el error se estabiliza o su valor pasa a aumentar rápidamente, por lo que para calcular q hay que situar bien el tramo recto de la gráfica.

2.Función de evaluación de errores por el método de Richardson

En el programa utilizado, en primer lugar, se extraen de otros programas y librerías de Python una serie de funciones que se utilizarán en el programa. Posteriormente, se definen las constantes y condiciones iniciales que se utilizarán en el método:

```
1   # Importación de funciones de otros programas
2   from Funciones.Cauchy_Problem import Cauchy_problem
3   from Funciones.Esquemas_temporales import Euler, CN, RK4, Euler_inverso
4   from Funciones.Funciones_a_integrar import Kepler
5   # IMPORTACIÓN
6   from numpy import array, zeros, sqrt, log, abs, linspace, log10 #Importación de funciones numéricas
7   from scipy.optimize import newton #Importación del método de Newton
8   import matplotlib.pyplot as plt #Importación de gráficas
9
10  #Definición de condiciones iniciales
11  deltat = 0.1 #Intervalo de tiempo entre iteraciones
12  U0 = array([1,0,0,1]) #Condiciones iniciales
13  F0 = array([0,1,-1,0])#Valor de F en el instante inicial
14  T = 100 #Tiempo para el que se va a integrar el error
15  Error =array(zeros([int(T/deltat)]))
```

Figura 1: Declaración de variables y condiciones iniciales

La principal diferencia con otros milestones es que las funciones de métodos numéricos, ecuaciones de Kepler y los esquemas temporales se importan de otros programas creados específicamente con el propósito de albergarlas. Estas funciones son las mismas que se utilizaron en el Milestone 2, por lo que es innecesario volver a explicarlas en este informe.

Después, se continúa con la definición de funciones. En primer lugar, se encuentra la función para la evaluación de errores por el método de Richardson, que es la siguiente:

```
# Definición de funciones

#Función para calcular el error de un esquema temporal para un tiempo y delta de t dados
def Error_esquema (Cauchy_problem, Temporal_scheme, U0, deltat, F, F0, t, q):
    N = len(t)
    C = len(U0)
    E = array(zeros([N,C]))
    matriz_Un = array(Cauchy_problem(Temporal_scheme, U0, N, deltat, F, F0, len(U0)))
    matriz_U2n = array(Cauchy_problem(Temporal_scheme, U0, 2*N, deltat/2, F, F0, len(U0)))
    for i in range (N):
        E[i-1,:] = (matriz_U2n[2*i-1,:]-matriz_Un[i-1,:])/(1-0.5**q)
        continue
    return E
```

Figura 2: Función para calcular el error de un esquema numérico

Donde el error se evalúa a lo largo de un intervalo de tiempo que es un input. Puede ser un vector (lo que daría lugar a la evolución del error con el tiempo) o un escalar (que devuelve el error en un tiempo concreto). La variable E contiene el error de todas las coordenadas del vector U que se esté evaluando en el problema de Cauchy, por lo que esta función es válida para cualquier U y F que se quieran estudiar. La variable q es el orden del esquema utilizado, que es un input, por lo que se debe proporcionar. En el apartado 4 se verá cómo se calcula q para un esquema en el que a priori desconocido. También hay que recordar que las variables matriz_Un y matriz_U2n son matrices de tamaño NxC, que contiene la evolución con el tiempo del vector U, por lo que la matriz E que devuelve también lo será.

3.Error numérico de distintos esquemas

El código utilizado para hacer la representación del error cometido por los distintos esquemas es el siguiente:

```
Esquemas = [Euler, Euler_inverso, CN, RK4]
t = linspace(0, T, num=int(T/deltat))
for j in Esquemas:
    Orden = Orden_esquema(Cauchy_problem, j, U0, Kepler, F0, 1, 1000, 0.01, 0.001)
    matriz_Error = Error_esquema (Cauchy_problem, j, U0, deltat, Kepler, F0, t, Orden)
    norma_Error = array(zeros([len(t)]))
    for i in range(int(T/deltat)):
        norma_Error[i] = sqrt(matriz_Error[i,0]**2+matriz_Error[i,1]**2)
        continue

    plt.plot(t, norma_Error)
    plt.show()
    plt.xlabel("Tiempo de integración(s)")
    plt.ylabel("Error")
    continue
```

Figura 3: Código para la representación del error de distintos esquemas

Donde en primer lugar se define una lista en la que se introducen los distintos esquemas, que se introducen en un bucle para que con el mismo código se calcule el error de los 4 a la vez. El tiempo se define como un linspace entre 0 y el tiempo T que se haya dado al principio del programa. A continuación, se calcula el orden del esquema (ver apartado 4) y posteriormente se obtiene la matriz del error con la función del apartado 2, en la que el orden del esquema es un input. De esta forma, incluso desconociendo el orden del esquema se puede obtener el error.

A continuación, se define un vector para almacenar la norma del error con el tiempo. Como en el problema de Kepler se quiere conocer el error en la posición, la norma del error que nos interesa es la raíz cuadrada de las dos primeras componentes de la columna de la matriz del error. Haciendo esto para las N columnas, se obtiene un vector en el que se almacena la norma del error. Finalmente, se grafica esta norma con el tiempo definido previamente. Si se quiere obtener el error de otra función diferente, simplemente se tendrá que cambiar la parte del código donde se define la norma, pero ninguna otra.

Los resultados numéricos obtenidos por este método para los distintos esquemas son los siguientes:

a) Euler

Tanto para este método como para los siguientes, no se explicará el esquema en profundidad ni la función utilizada, ya que estos aspectos ya se explicaron en los informes de los Milestones anteriores. Para los cuatro métodos que se cubren en este informe, se evaluará la evaluación del error para un tiempo final T = 100s, y, si fuera necesario para algún método, se realizará la evaluación para un tiempo final menor o mayor.

Para el método de Euler, la evolución del error para T = 100s y un intervalo de tiempo entre iteraciones de 0,1s es la siguiente:

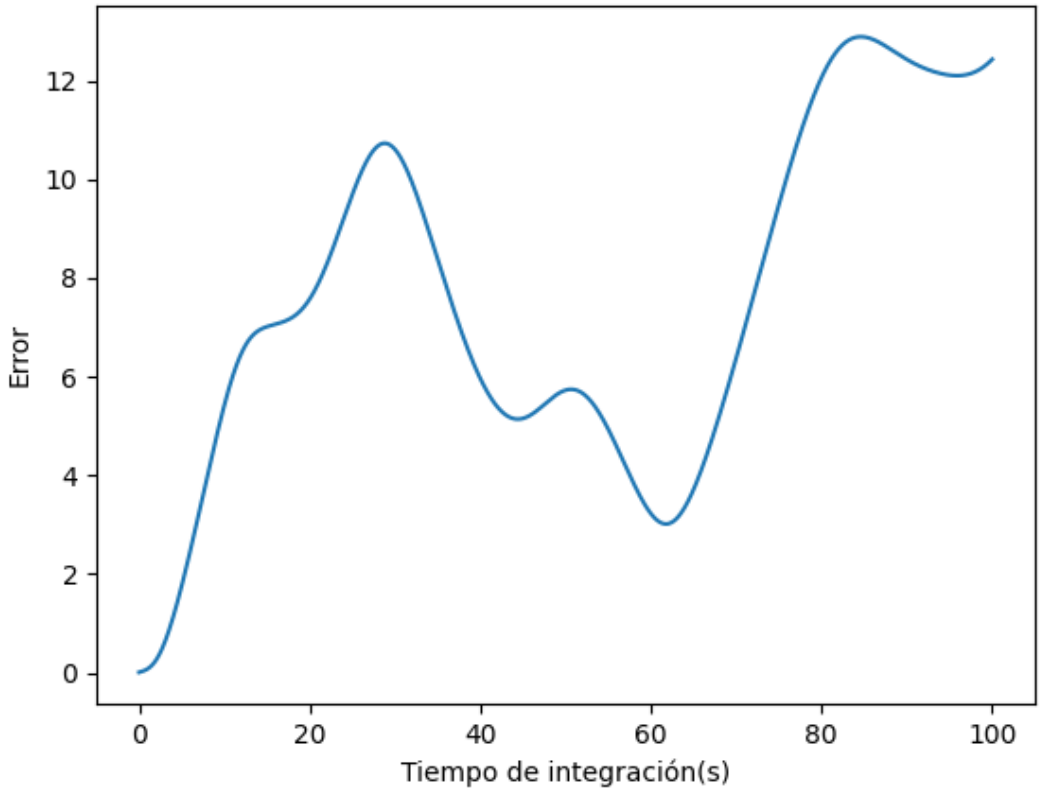

Figura 4: Evolución del error en el método de Euler para T = 100s y Δt = 0,1s

Se observa una evolución muy rápida del error creciente, dejando de ser un método preciso a los

pocos segundos. Se observa una disminución del error tras 30s, pero al haber divergido ya es muy probable que esto se deba a que la trayectoria a la que da lugar este método pasa de forma más cercana a la órbita, pero no a que el método converja a la trayectoria real. Al ser un esquema de orden 1, es lógico que el error cometido con el tiempo crezca rápidamente, más cuando el paso de tiempo no es demasiado elevado. En este método es más interesante reducir el tiempo de integración a 10s y ver la evolución del error en tiempos más cortos:

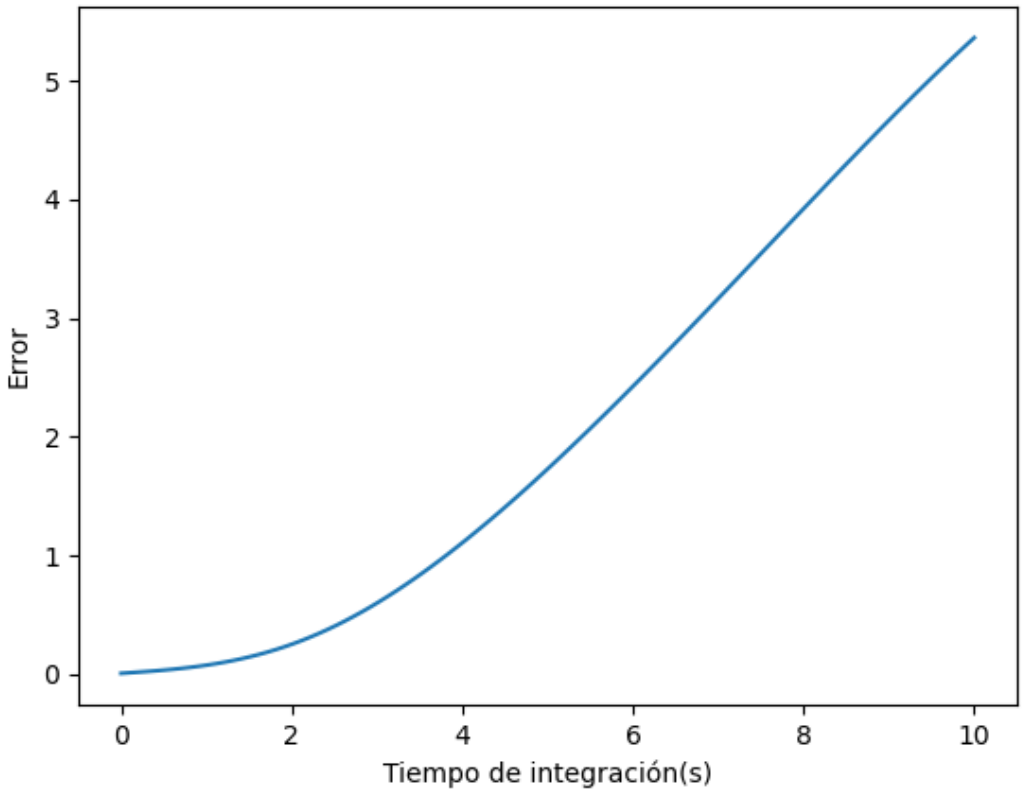

Figura 5: Evolución del error en el método de Euler para T = 10s y Δt = 0,1s

Donde se ve que en los primeros dos segundos el error se mantiene en márgenes aceptables, pero el error crece rápidamente. A continuación, se va a observar el efecto que tiene en estas gráficas el disminuir el Δt utilizado:

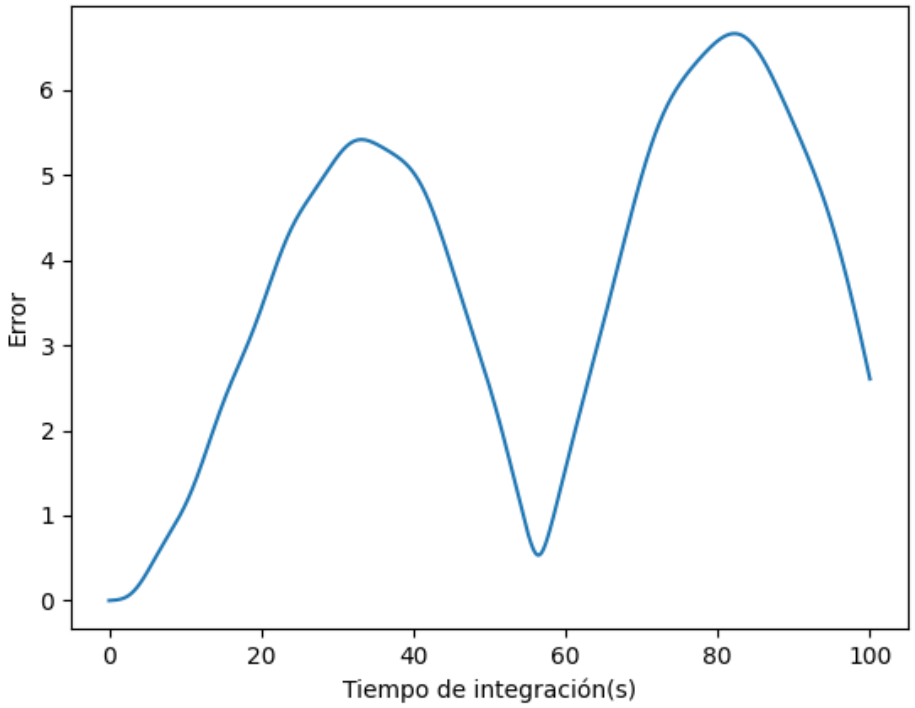

Figura 6: Evolución del error en el método de Euler para T = 100s y Δt = 0,01s

Donde se puede ver que la gráfica obtenida tiene una forma similar a la figura 4, pero con el error llegando a menores valores, lo que es un resultado lógico teniendo en cuenta que el error depende del paso de tiempo utilizado.

b) Euler inverso

Para el Euler inverso, los valores del error obtenido deberían ser similares a los obtenidos con Euler, ya que son esquemas del mismo orden, orden 1. Para T = 100s y un intervalo de tiempo entre iteraciones de 0,1s, la evolución del error en este esquema es:

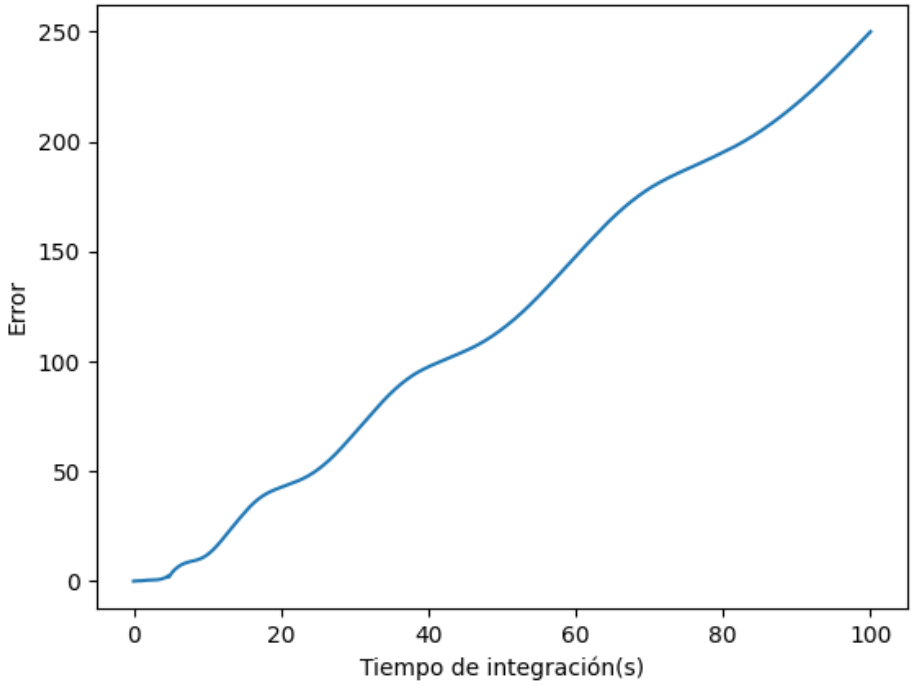

Figura 7: Evolución del error en el método de Euler inverso para T = 100s y un Δt=0,1 s

Donde se observa que la gráfica diverge para valores tempranos de t de forma brusca, más aún que en el caso de Euler. Observando la gráfica para tiempos menores:

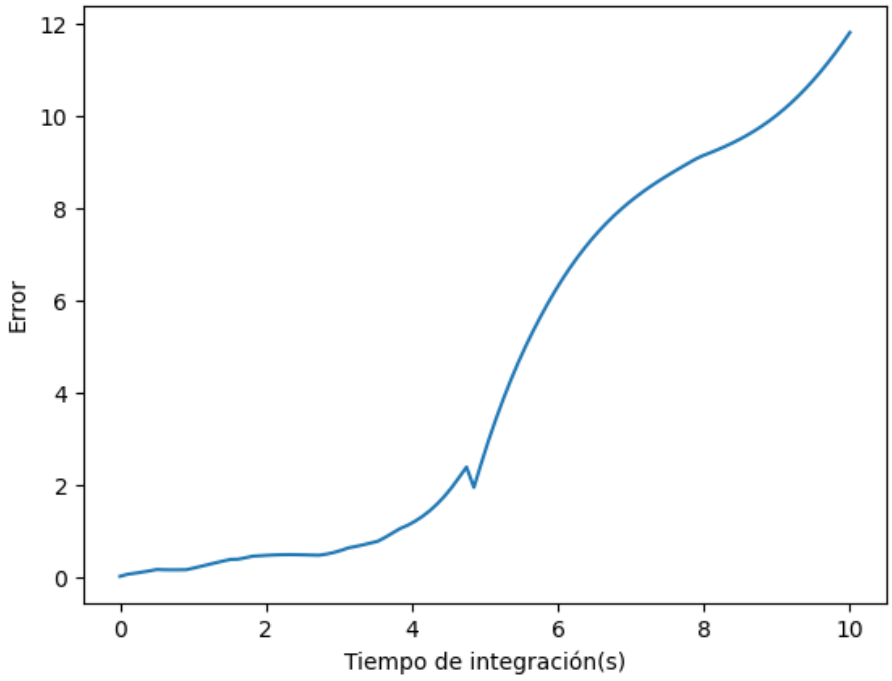

Figura 8: Evolución del error en el método de Euler inverso para T = 10s y un Δt=0,1 s

Donde se ve que la divergencia del error empieza a los 5 segundos, tras lo que empieza a crecer de forma muy rápida. Ya se había comentado en los Milestones anteriores, pero queda claro una vez más que ni el método de Euler ni el de Euler inverso son métodos apropiados para la modelización de las ecuaciones de Kepler.

c) Crank-Nicholson

En este esquema debería observarse una mejora apreciable respecto a los otros dos métodos, ya que es un método de orden 2. Para T = 100s y un intervalo de tiempo entre iteraciones de 0,1s, la evolución del error en este esquema es:

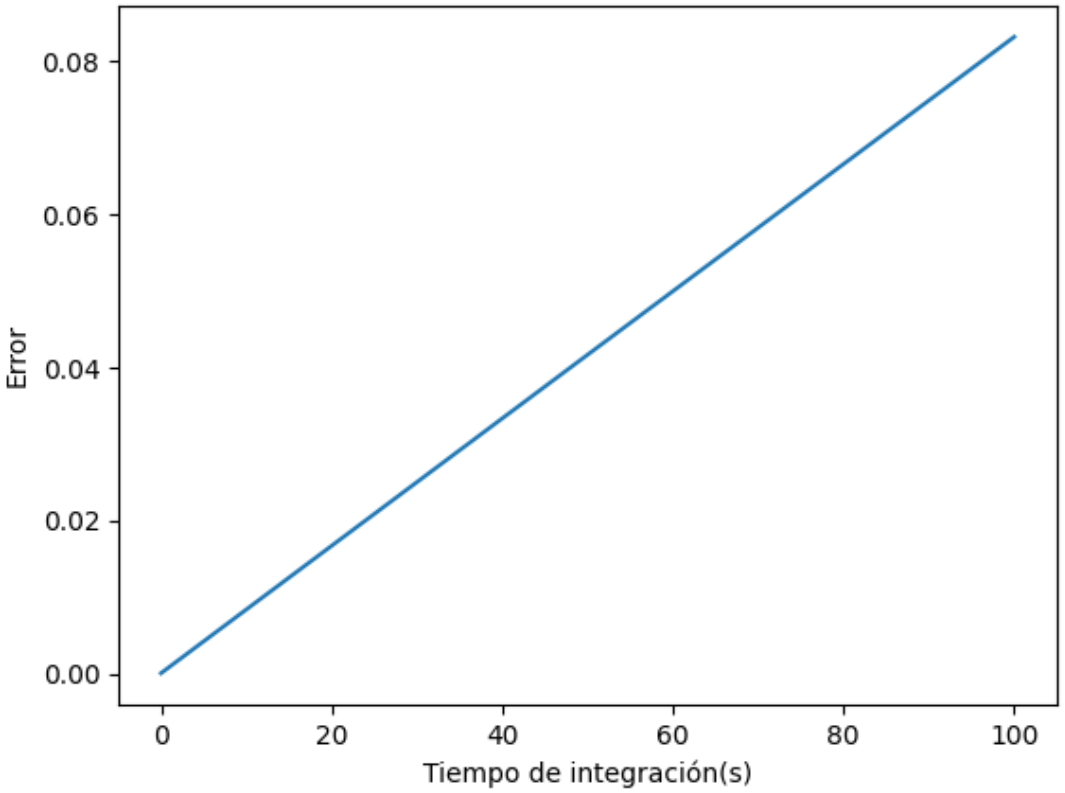

Figura 9: Evolución del error en el método de Crank-Nicholson para T = 100s y Δt = 0,1s

Se observa una evolución lineal del error con el tiempo, lo que es un resultado lógico teniendo en cuenta que el método no llega a divergir. Es un error mucho menor en comparación a los métodos de Euler y Euler inverso, y para este tiempo de integración da resultados suficientemente buenos como para poder plantearse la utilización de este método para la integración de las órbitas de Kepler. De este método es interesante aumentar el tiempo de integración:

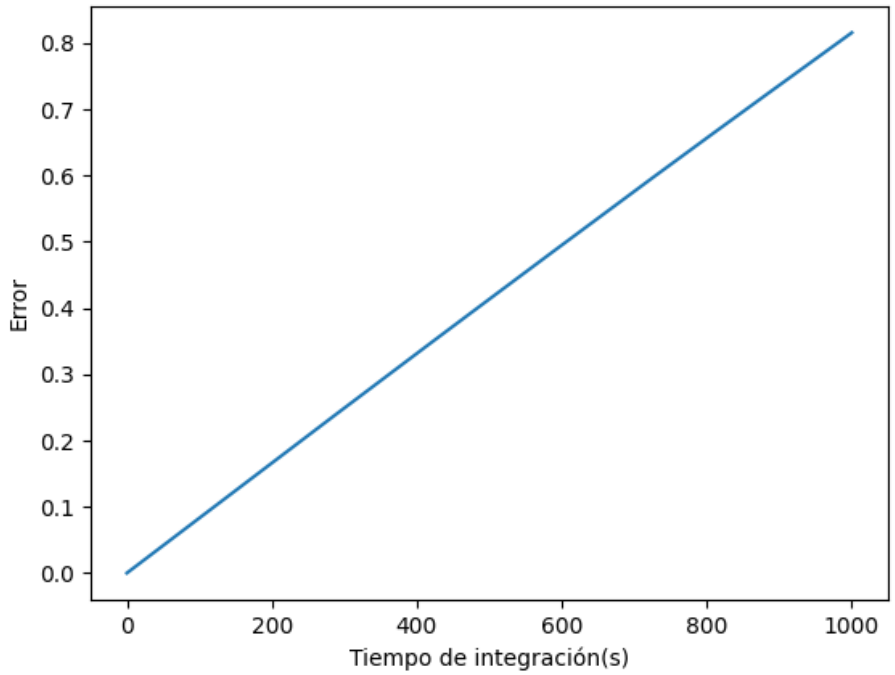

Figura 10: Evolución del error en el método de Crank-Nicholson para T = 1000s y Δt = 0,1s

El error no diverge, pero para tiempos cercanos a 1000 segundos llega a ser prácticamente del orden de la unidad, con lo que para estos tiempos deja de ser interesante su aplicación. Esto se puede intentar solucionar disminuyendo el Δt:

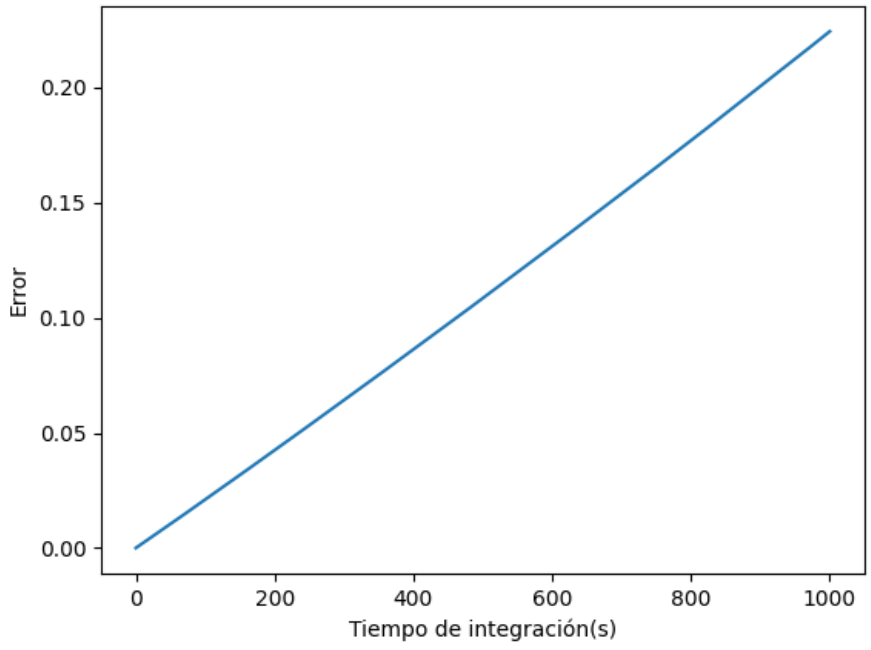

Figura 11: Evolución del error en el método de Crank-Nicholson para T = 1000s y Δt = 0,05s

Los resultados son prometedores, ya que se ha reducido la magnitud del error en una cuarta parte sólo disminuyendo el Δt a la mitad, por lo que, si el perjuicio de asumir un mayor coste computacional es asumible, es una técnica buena para poder analizar las ecuaciones de Kepler con este método para tiempos largos.

d) Range-Kutta de orden 4

Este método, el de mayor orden de los estudiados, debería ser el más efectivo para calcular el error. Para T = 100s y un intervalo de tiempo entre iteraciones de 0,1s, la evolución del error en este esquema es:

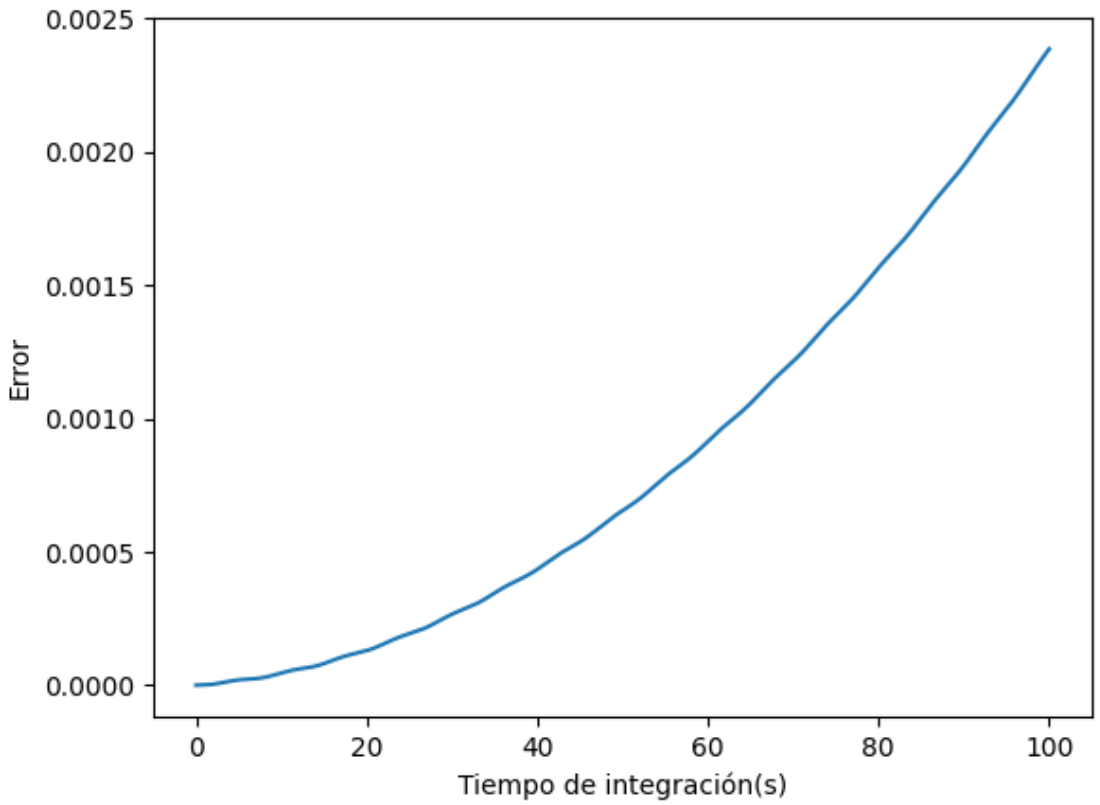

Figura 12: Evolución del error en el método RK4 para T = 100s y Δt = 0,1s

Donde se puede comprobar que el método tiene el menor error con mucha diferencia respecto al resto. No es una evolución exactamente lineal, parece más parabólica. Para comprobar si esta tendencia prosigue en el tiempo, se aumenta el tiempo de evaluación a 1000s:

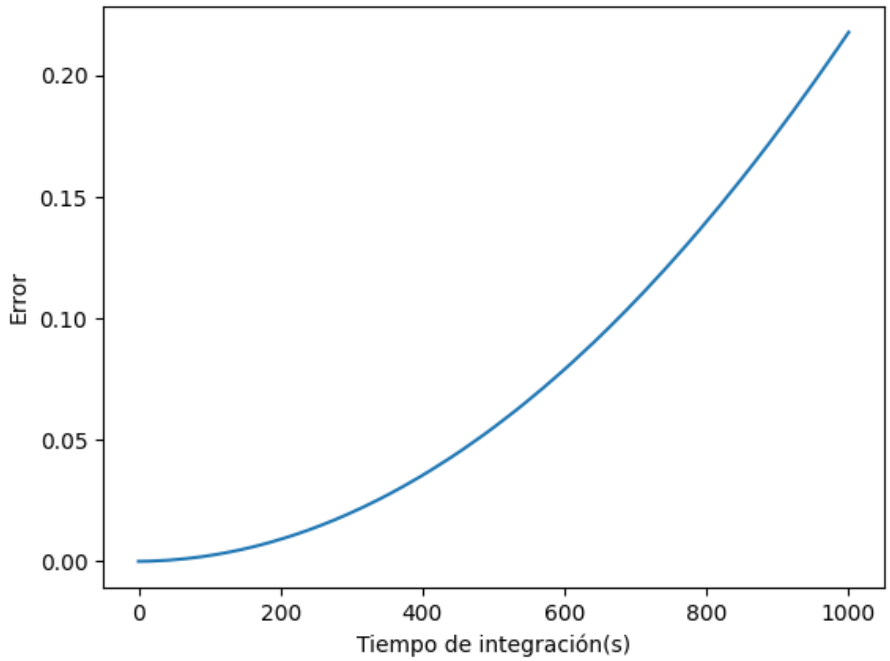

Figura 13: Evolución del error en el método RK4 para T = 1000s y Δt = 0,1s

Se observa que la tendencia exponencial continúa, pero sigue siendo una progresión del error menor a la que se observaba en los otros métodos. Por tanto, si no hay otros factores que lo desaconsejen, este método es el más propicio para el estudio de este problema de entre los estudiados.

4.Función de evaluación de ratio de convergencia

La función para la evaluación del ratio de convergencia utilizada en este programa es la siguiente:

```
#Función para calcular el orden de un esquema numérico para un tiempo T y dando un número de puntos
#de la gráfica N-E llamado K. deltatmin y deltatmax son los deltat máximo y mínimo para los que se
#calculará la gráfica N-E. Hay que darlos con cuidado, según el esquema que se esté usando. Darlos
#incorrectamente puede provocar que falle la función.
def Orden_esquema(Cauchy_problem, Temporal_scheme, U0, F, F0, T, K, deltatmin, deltatmax):
    h = linspace(deltatmax, deltatmin, K)
    n = array(zeros([K]))
    C = len(U0)
    U = array(zeros([K,C]))
    norma_U = array(zeros([K]))
    norma_cuad_U = array(zeros([K]))
    for k in range(K):
        deltat = h[k]
        N = int(T/deltat)
        matriz_Un = array(Cauchy_problem(Temporal_scheme, U0, N, deltat, F, F0, len(U0)))
        matriz_U2n = array(Cauchy_problem(Temporal_scheme, U0, 2*N, deltat/2, F, F0, len(U0)))
        U[k,:] = matriz_U2n[2*N-1,:]-matriz_Un[N-1,:]
        n[k] = N
        for j in range(len(U0)):
                norma_cuad_U[k] = norma_cuad_U[k]+U[k,j-1]**2
                continue
        norma_U[k]=sqrt(norma_cuad_U[k])
        continue
    logU = log10(norma_U)
    logN = log10(n)
    vector_q = array(zeros([int(K/2)]))
    for i in range (int(K/2)):
        vector_q[i] = (logU[i+int(K/4)]-logU[i-10+int(K/4)])/(logN[i+int(K/4)]-logN[i-10+int(K/4)])
        continue

    q = 0
    for i in range(int(K/2)-1):
        q= q + vector_q[i]
        continue
    media_q = q/(K/2)
    Orden = (-1)*round(media_q)
    print("El orden del esquema es", Orden)
    logE = logU/(1-0.5**Orden)
    plt.plot(logN, logE)
    plt.xlabel("log(N)")
    plt.ylabel("log(E)")
    plt.show()
    return Orden
```

Figuras 14 y 15: Función para calcular el orden de un esquema numérico

Aparte de los outputs normales del problema de Cauchy, se debe dar un número de puntos que se quiere calcular de la gráfica N-E, el tiempo para el que se van a calcular los errores, y los pasos de tiempo máximo y mínimo para los que se va a calcular el error. Para estos pasos de tiempo, se define un linspace de K puntos entre ellos. Para estos tiempos, se definen los Δt en cada iteración, que junto al T de estudio definen la N de cada iteración. Esto permite que, para unos Δt mínimo y máximo adecuados, la función permita construir la gráfica logE contra logN. En primer lugar, se calcula el logaritmo de U^{2n}-U^{n}, ya que a priori no se tiene por qué conocer el valor de q. q se calcula haciendo la media de los valores en el intervalo [K/4, 3K/4] en puntos separados por un valor de 10 intervalos de tiempo (ya que puntos muy próximos pueden dar errores). Se obtiene la media de q en ese intervalo y se redondea para obtener el orden. A partir de ese orden, con la ecuación 6, se obtiene el error, que se grafica respecto al logaritmo de N en cada punto. Finalmente, la función devuelve el orden del esquema, que, como se vio en el apartado 2, se utiliza para graficar el error del esquema con el tiempo.

5.Ratio de convergencia de diferentes esquemas

Para los esquemas ya vistos, se calculará el orden y se graficará el error respecto al número de iteraciones de forma logarítmica. Como ya se ha visto, se debe dar un tiempo de evaluación, que se ha escogido 1 segundo porque para tiempos mucho mayores los esquemas de Euler y Euler inverso divergen y su evaluación puede provocar errores. Los intervalos de tiempo escogidos son 0,01 y 0,001 (ya que intervalos mucho menores necesitan un tiempo de cálculo muy elevado, y es innecesario para lo que se pretende obtener) y se realizarán 100 iteraciones (se puede dar valores de K mayores, pero sólo suavizan más las irregularidades que se observan en la gráfica, la forma es la misma). La gráfica puede prolongarse o cortarse variando los intervalos de tiempo, pero hay que tener en cuenta que para log(N) demasiado pequeños o grandes el método falla.

a) Euler

Para el método de Euler, el error es:

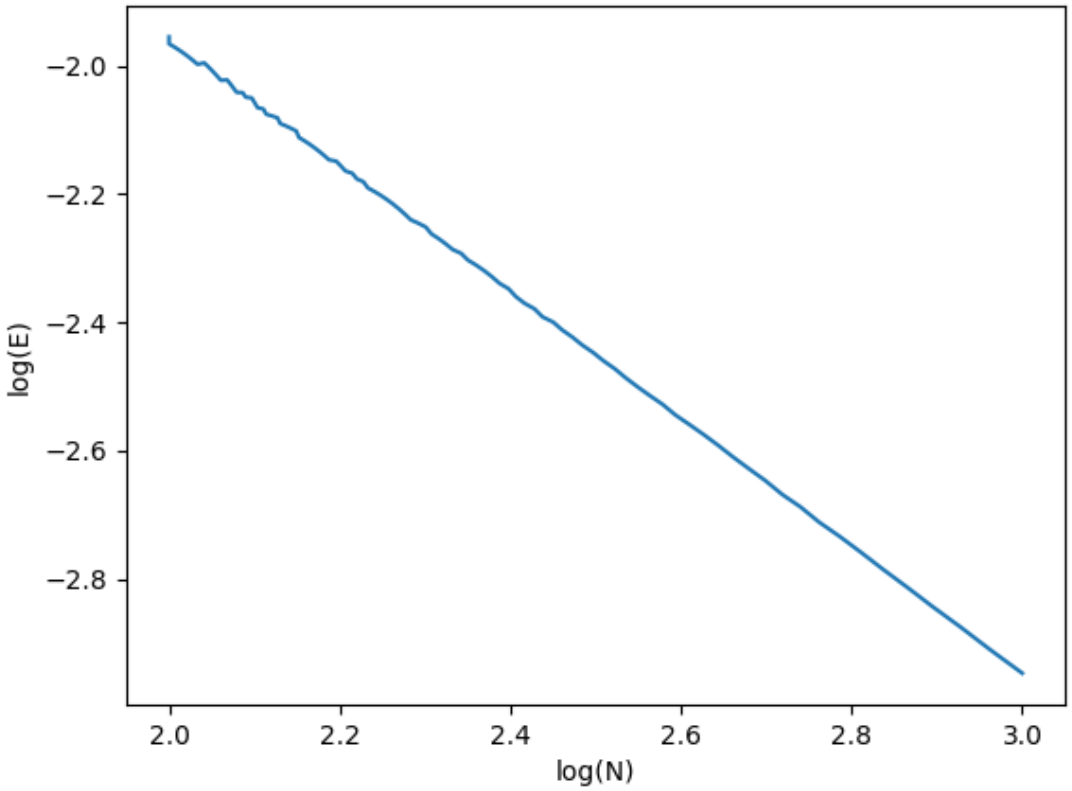

Figura 16: log E vs log N en el método de Euler para T = 1s

Al principio se observan algunas irregularidades, pero enseguida se estabilizan para formar una recta. La recta tiene pendiente -1, se puede ver en la gráfica, por lo que el esquema es de orden 1. El programa también devuelve este valor para el esquema de Euler. No se observan problemas apreciables para estos valores de log(N)

b) Euler inverso

Para el método de Euler, el error es:

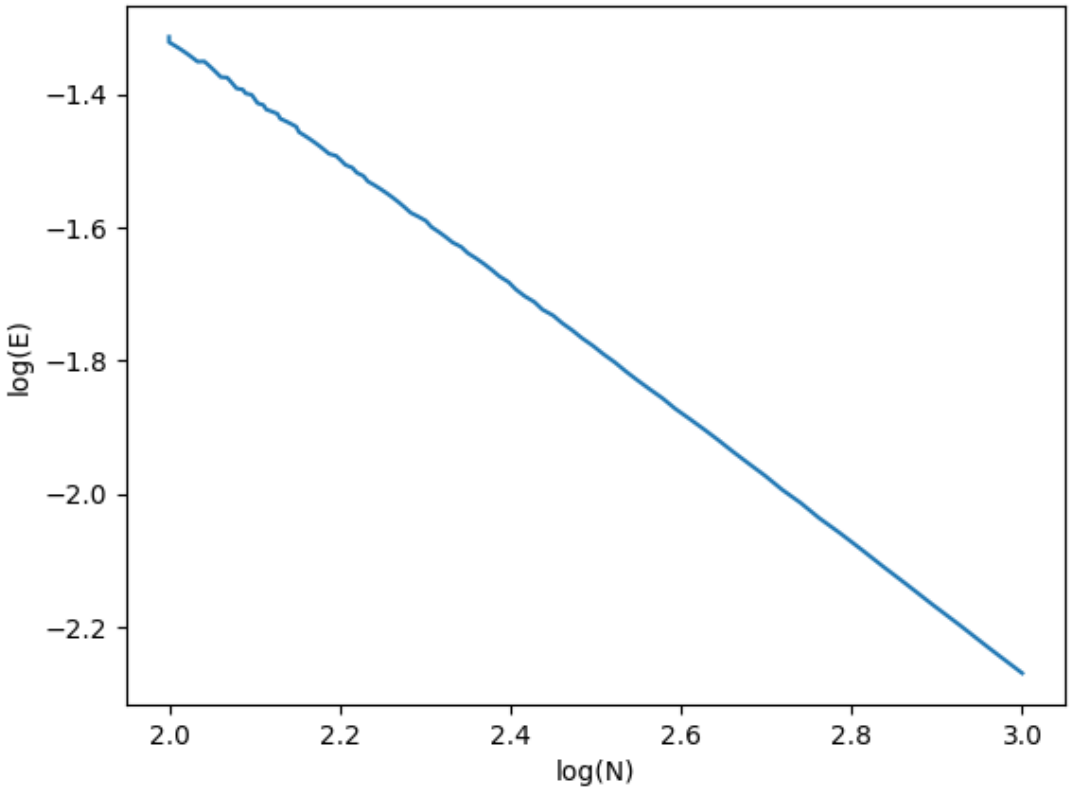

Figura 17: log E vs log N en el método de Euler inverso para T = 1s

Se observa una gráfica similar, pero para valores algo mayores (menos negativos) del error. Esto se corresponde con las gráficas del apartado 3, que demostraban que para las ecuaciones de Kepler el error del Euler inverso es superior al del Euler. También se observa una gráfica con pendiente -1, lo que corresponde a un método de orden 1, por lo que la gráfica es correcta. El programa también devuelve que el orden de este esquema es 1.

c) Crank-Nicholson

Para el método de Crank-Nicholson, el error es:

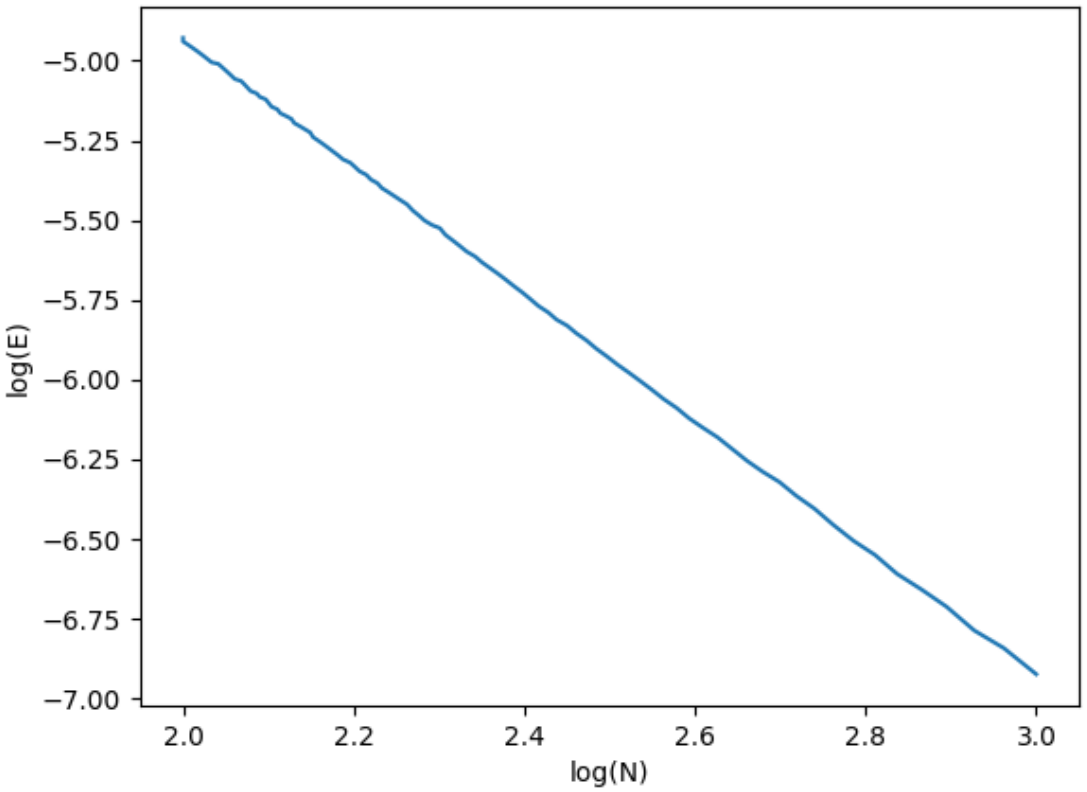

Figura 18: log E vs log N en el método de Crank-Nicholson para T = 1s

También es una gráfica recta prácticamente (otra vez, si se quieren disminuir esas irregularidades que se aprecian, basta con aumentar el número de iteraciones), pero se observa una pendiente superior, de -2, lo que corresponde a un método de orden 2 como este. El programa también devuelve que el orden de este esquema es 2.

d) Range-Kutta de orden 4

Para el método de Range-Kutta de orden 4, el error es:

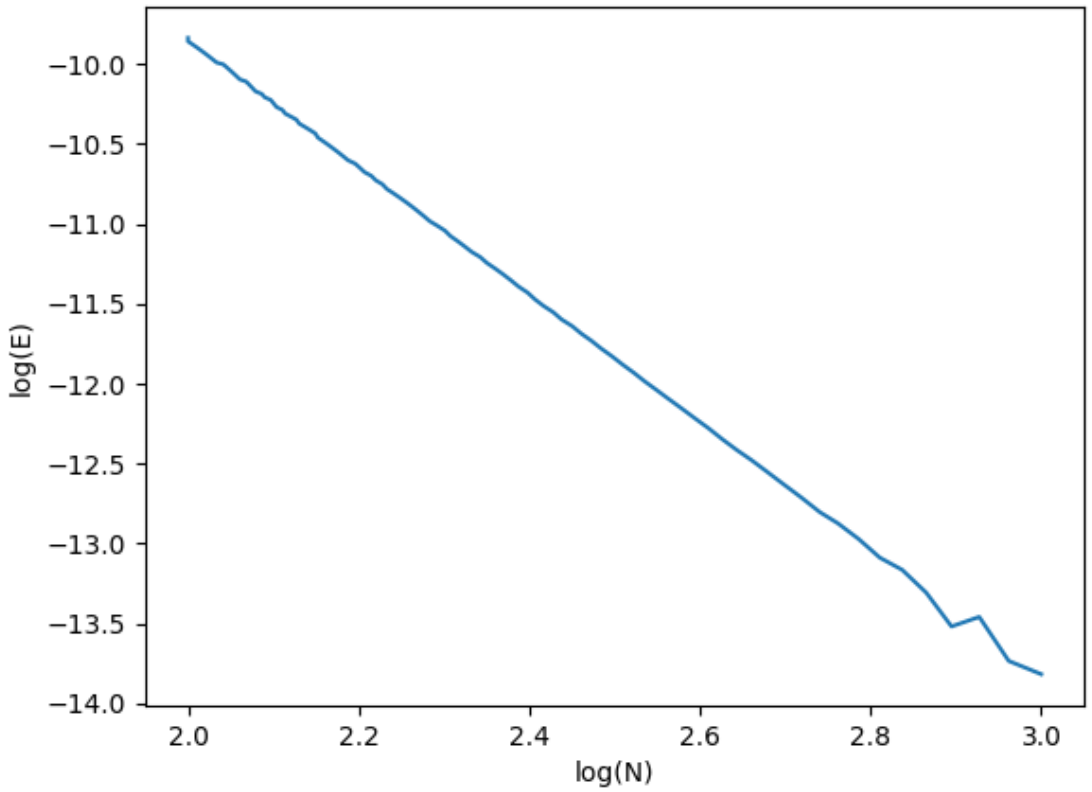

Figura 18: log E vs log N en el método RK4 para T = 1s

Se observa una gráfica casi recta, con pendiente -4, salvo en la parte final. Esto ocurre porque el método empieza a fallar para log(E) menores a -14 o -15, como se observa en esta gráfica. Esta es una de las limitaciones del método, y se observa en esta gráfica. Esta es una de las razones por las que en el programa se eligió calcular q en el intervalo intermedio de [K/4, 3K/4], para mitigar lo más posible si el intervalo de Δt no está bien escogido y en el extremo se producen problemas. De todas formas, el programa sigue dando que el orden del esquema es 4, por lo que este efecto para estos N se ha conseguido mitigar.