


Universidad Politécnica de Madrid

Ampliación de Matemáticas I

Informe Hito 3

Autor:
Guillermo García del Río

Índice

Índice de figuras

1. Introducción

En este informe se presentan y comentan los resultados obtenidos en el hito 3 propuesto en la asignatura de Ampliación de Matemáticas I. En primer lugar se mostrará el enunciado que marca las pautas del proyecto, seguido de un análisis del problema a resolver, la exposición del código y los resultados obtenidos con él. Para terminar se comentarán y compararán dichos resultados.

2. Enunciado del hito

Los objetivos de este hito quedan establecidos por los siguientes puntos:

1 Escribir una función para evaluar errores de integración numérica mediante la extrapolación de Richardson. Esta función debe estar basada en la solución del problema de Cauchy implementada en el Hito 2.

2 Error numérico de distintos esquemas temporales: Euler, Euler inverso, Crank Nicolson y Runge Kutta de cuarto orden.

3 Escribir una función que evalúe el ratio de convergencia de diferntes esquemas numéricos.

4 Ratio de convergencia de los diferentes métodos para varios pasos de tiempo.

3. Problema a resolver

3.1. Error numérico por extrapolación de Richardson

El objetivo de este hito es calcular el error de distintos esquemas numéricos. Ya que el error de una solución numérica se define como la diferencia entre la solución exacta $u(t_n)$ menos la solución aproximada U^n en el mismo instante t_n

$$E^n = u(t_n) - U^n \tag{3.1.1}$$

la determinación del error requiere que se conozca la solución exacta, lo cual es poco común y hace que se tengan que buscar caminos alternativos, como la extrapolación de Richardson.

Si el error global se pudiera expandir en series de potencias de Δt como:

$$E^n = k(t_n)\Delta t^q + O(\Delta t^{q+1}) \tag{3.1.2}$$

con $k(t_n)$ independiente de Δt y sieno q el órden del esquema numérico utilizado, entonces se puede llevar una estimación basada en la extrapolación de Richardson. Suponiendo que se puede expresar el error como en 3.1.2 e integrando dos mallas temporales con paso de tiempo de Δt_n y $\Delta t_n/2$ se obtienen U_1 y U_2, que son las soluciones integradas con dichas mallas temporales. La ecuación 3.1.2 queda escrita como:

$$u(t_n) - U_1^n = k(t_n)\Delta t^q + O(\Delta t^{q+1}) \tag{3.1.3}$$

$$u(t_n) - U_2^n = k(t_n)\frac{\Delta t^q}{2} + O(\Delta t^{q+1}) \tag{3.1.4}$$

Restando ambas expresiones se llega a la estimación del error proporcionada por el método de Richardson:

$$E^n = \frac{U_1^n - U_2^n}{1 - \frac{1}{2^q}} \tag{3.1.5}$$

3.2. Ratio de convergencia de esquemas temporales

El segundo objetivo de este hito es estudiar el ratio de convergencia para los distintos esquemas temporales empleados. Se dice que un esquema numérico es de orden q si su error numérico es $O(\Delta t^q)$, lo cual significa que si Δt es lo suficientemente pequeño, el error tiende a cero con la misma velocidad que Δt^q. Tomando módulos y logaritmos en la ecuación 3.1.2 y teniendo en cuenta que $\Delta t \propto N^{-1}$:

$$log(|E^n|) = C - q \cdot log(N) \tag{3.2.1}$$

Cuando se representa gráficamente esta expresión en escala logarítmica, se muestra como una línea con pendiente negativa q, que es el orden del esquema numérico. Cuando se está lidiando con esquemas numéricos complejos o desarrollando nuevos métodos, es importante conocer el ratio de convergencia del esquema o su orden real. Para ello, es necesario conocer el error, el cual se puede determinar mediante la extrapolación de Richardson.

4. Exposición del código

De cara a poder entender lo que se está haciendo se va a proceder a exponer el código desarrollado para solucionar el problema propuesto. Ya que los códigos del problema de Cauchy y de los esquemas numéricos empleados se mostraron en el informe del Hito 2, en este informe tan solo se incluirán los códigos desarrollados para el cálculo del error, el ratio de convergencia y el archivo principal del Hito 3.

4.1. Módulo de error

En este módulo se han incluido tanto el cálculo del error mediante la extrapolación de Richardson como el ratio de convergencia del esquema temporal.

4.1.1. Extrapolación de Richardson

El código implementado para llevar a cabo el cálculo del error según la extrapolación de Richardson se muestra a continuación:

```
def Richardson(Problem,Scheme, t, U0):

    n  = size(t)          # Número de puntos del mallado temporal
    T  = t[n-1]           # Tiempo final de la integración
    t1 = t                # Primer mallado temporal

    t2 = linspace(0,T,2*n) # Segundo mallado temporal con el doble de puntos

    U1 = Cauchy_Problem(Problem, t1, U0, Scheme)
    U2 = Cauchy_Problem (Problem, t2, U0, Scheme )

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

    if Scheme == Euler:
        q = 1
    elif Scheme == RK4:
        q = 4
    elif Scheme == Crank_Nicolson:
        q = 2
    elif Scheme == Euler_inverso:
        q = 1

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

    return Error
```

Algoritmo 4.1: Extrapolación de Richardson

Como se puede observar es prácticamente idéntico a lo expresado en la explicación teórica. Los inputs de la función serán el problema que se quiere integrar (en este caso la órbita Kepler), el esquema temporal, un mallado temporal y las condiciones iniciales del problema. Como se preveía que fuera a ser utilizado para los esquemas utilizados empleados anteriormente, se estableció un orden variable dependiendo del esquema que se quisiera utilizar.

4.1.2. Ratio de convergencia de esquemas temporales

El código implementado para llevar a cabo el cálculo del ratio de convergencia se muestra a continuación:

```
def Temporal_convergence_rate(Problem, Scheme, t, U0):


    n  = size(t)
    T  = t[n-1]
    t1 = t

    U1 = Cauchy_Problem(Problem, t1, U0, Scheme)

    m = 8               #Número de puntos que se quiere plotear.
    log_E = zeros(m)
    log_N = zeros(m)
    Error = zeros(m)
    n = 2*n

    for i in range (0,m):

        t2 = linspace(0,T,(2**i)*n)
        U2 = Cauchy_Problem (Problem, t2, U0, Scheme)

        Error[i] = norm(U2[int((2**i)*n-1),:] - U1[int((2**i)*n/2-1),:])
```

```
        log_E[i] = log10(Error[i])
        log_N[i] = log10((2**i)*n)

        U1 = U2
        print(i)

        for j in range(0,m):

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

             break

    j = min(j, m)

    # Regresión lineal y cálculo de la pendiente

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

    log_N_lineal = log_N[0:j+1]
    log_E_lineal = reg.predict(log_N[0:j+1].reshape((-1, 1)))

    return [log_E, log_N, log_E_lineal, log_N_lineal, order]
```

Algoritmo 4.2: Ratio de convergencia

Usando como base la explicación teórica expuesta anteriormente esta función tiene como inputs los mismos elementos que la función del cálculo del error de Richardson, aunque los outputs serán varios. En primer lugar se calcula el ratio de convergencia en m puntos, valor que se ha ido cambiando dependiendo del esquema numérico introducido. Además, se calculaba la regresión lineal de estos puntos, en base a la cual se obtiene el orden del esquema numérico (la pendiente de la recta).

4.2. Módulo de matemáticas

Siguiendo en la línea de implementación propia de código y pensando en el código de Newton desarrollado para el Hito 2, se ha escrito el código para la factorización LU de cara a poder resolver el problema:

$$A \cdot x = b \tag{4.2.1}$$

donde $A \in M_{NxN}$ verifica $det(A) \neq 0$ y $b \in N$.

Este método se basa en el hecho de que para llegar a la matriz triangular superior del método de Gauss (a la que se hará referencia como U), se tienen que realizar unas cuantas operaciones de filas sobre la matriz A. Esto significa que existe una matriz invertible $L^{-1} \in M_{NxN}$ que contiene dichas operaciones de tal manera que cuando A sea multiplicada por ella se consigue U. En otras palabras, la matriz A se puede expresar como:

$$A = LU \tag{4.2.2}$$

donde L y U son las matrices triangulares inferior y superior respectivamente.

Los códigos de la factorización LU, solución de LU y resolución del problema 4.2.1 se han

incluido en el módulo de matemáticas junto al código del cálculo del jacobiano y del método de Newton. Dichos códigos se muestran a continuación.

```
1
2  ## LU FACTORIZATION ##
3  def factorization_LU(A):
4
5    n = size(A,1)
6    U = zeros([n,n])
7    L = zeros([n,n])
8
9    U[0,:] = A[0,:]
10   for i in range(0,n):
11     L[i,i] = 1
12
13   L[1:n,0] = A[1:n,0]/U[0,0]
14
15
16   for k in range(1,n):
17
18     for j in range(k,n):
19       U[k,j] = A[k,j] - dot(L[k,0:k], U[0:k,j])
20
21     for i in range(k+1,n):
22       L[i,k] =(A[i,k] - dot(U[0:k,k], L[i,0:k])) / (U[k,k])
23
24   return [L@U, L, U]
25
26 ## SOLVE LU ##
27
28 def solve_LU(M,b):
29
30   n=size(b)
31   y=zeros(n)
32   x=zeros(n)
33
34   [A,L,U] = factorization_LU(M)
35   y[0] = b[0]
36
37   for i in range(0,n):
38     y[i] = b[i] - dot(A[i,0:i], y[0:i])
39
40
41   x[n-1] = y[n-1]/A[n-1,n-1]
42
43   for i in range(n-2,-1,-1):
44     x[i] = (y[i] - dot(A[i, i+1:n+1], x[i+1:n+1])) / A[i,i]
45
46   return x
47
48 ## INVERSE ##
49
50 def Inverse(A):
51
52   n = size(A,1)
53   B = zeros([n,n])
54
55   for i in range(0,n):
56     one = zeros(n)
57     one[i] = 1
58
59     B[:,i] = solve_LU(A, one)
60
61   return B
```

Algoritmo 4.3: Funciones de factorización LU

4.3. Archivo principal para el Hito 3

De manera similar a hitos anteriores, de cara a automatizar el proceso se ha realizado un bucle en el que se iban recorriendo los distintos esquemas numéricos de los cuales se ha solicitado un análisis. En este archivo el usuario puede cambiar el valor del paso de integración dt, lo cual será conveniente para llevar a cabo el análisis requerido de los resultados.

```
## HITO 3 ##

from math import log10

import matplotlib.pyplot as plt
from numpy import array, linspace, zeros

from Numeric.Error import Richardson, Temporal_convergence_rate
from Numeric.Esquemas_numéricos import (RK4, Crank_Nicolson, Euler,
                                        Euler_inverso)
from Physics.Órbitas import Kepler

## Variables temporales ##

T = 20                              # Duración de la simulación en segundos
dt = 0.1                            # Paso de integración en segundos
n = int(T/dt)                       # Numero de pasos
t  = linspace(0,T,n)                # Vector de instantes separados dt



## Condiciones iniciales ##

U0 = array([1,0,0,1])

methods = [Euler,RK4, Crank_Nicolson, Euler_inverso]
lista = ['Euler','RK4','Crank Nicolson','Euler inverso']

for j in range(4):

    Error = Richardson(Kepler, methods[j], t, U0)
    Error_norm = (Error[:,0]**2 + Error[:,1]**2)**(1/2)

    plt.figure(1)
    plt.subplot(211)
    plt.title(f'Error de {lista[j]} con dt = {dt} s y T = {T} s')
    plt.plot(t, Error[:,0],"r", label = "X position")
    plt.plot(t, Error[:,1],"b", label = "Y position")
    plt.grid()
    plt.xlabel("t")
    plt.ylabel("Error")
    plt.legend(loc = "lower left")
    plt.subplot(212)
    plt.plot(t, Error_norm, 'c', label = 'Module')
    plt.xlabel("t")
    plt.ylabel("Error")
    plt.legend(loc ='lower right')
    plt.grid()
    plt.savefig('Plots/Hito 3/ Error ' + lista[j]+ ' ' + str(dt)+'.png')
    plt.show()

    [log_E, log_N, log_E_lineal, log_N_lineal, order] = Temporal_convergence_rate(
    Kepler, methods[j], t, U0)


    plt.plot(log_N, log_E, "b", label = lista[j])
    plt.plot(log_N_lineal, log_E_lineal, "r", label = 'Linear regression')
    plt.legend(loc ='lower left')
    plt.xlabel("log(N)")
```

```
    plt.ylabel("log(U2-U1)")
    plt.title(f'{lista[j]} , order = {order}')
    plt.plot()
    plt.grid()
    plt.savefig('Plots/Hito 3/ Conv '+lista[j]+ ' ' + str(dt)+'.png')
    plt.show()
```

Algoritmo 4.4: Archivo principal Hito 3

5. Resultados

Con el objetivo de llevar a cabo un buen análisis de los resultados obtenidos, y de forma parecida al hito anterior, se realizarán varias simulaciones variando el valor del paso de integración Δt y para un tiempo de 20 segundos.

Los valores elegidos para este hito son:

Tabla 5.1: Valores del paso de tiempo empleados

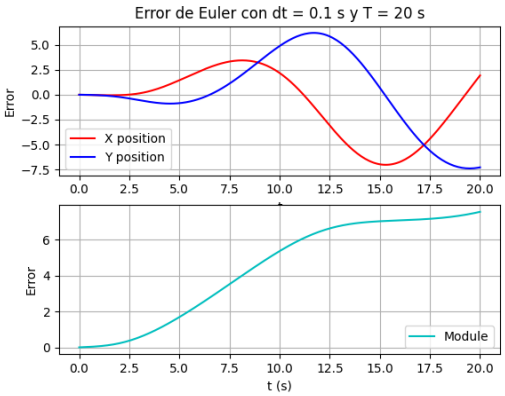
$\Delta t(s)$		
0.1	0.01	0.001

5.1. Error mediante extrapolación de Richardson

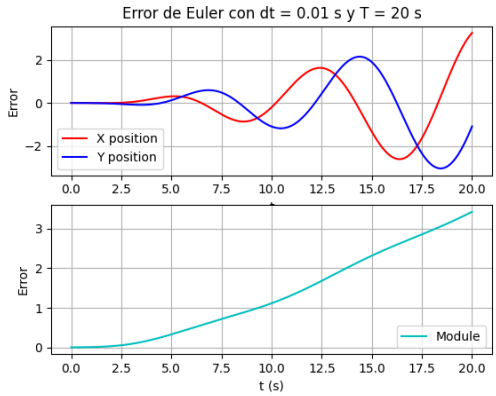
En este apartado se analizan los resultados obtenidos del error mediante la extrapolación de Richardson para los esquemas numéricos fijados por el enunciado del Hito. La manera en la que se van a exponer los resultados será mostrando el error en las componentes X e Y del movimiento acompañado del módulo del error en estas dos componentes.

5.1.1. Euler

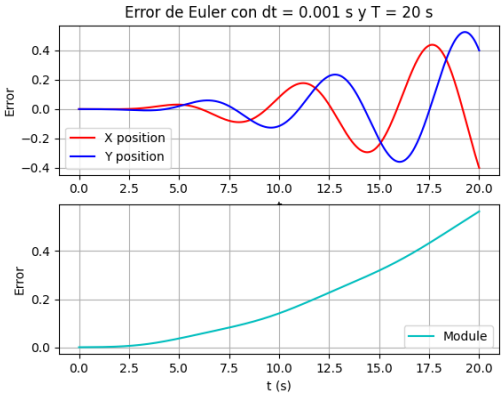
Los resultados obtenidos por el esquema numérico de Euler se presentan en la figura 5.1.



(a) Resultados obtenidos para un $dt = 0{,}1s$



(b) Resultados obtenidos para un $dt = 0{,}01s$



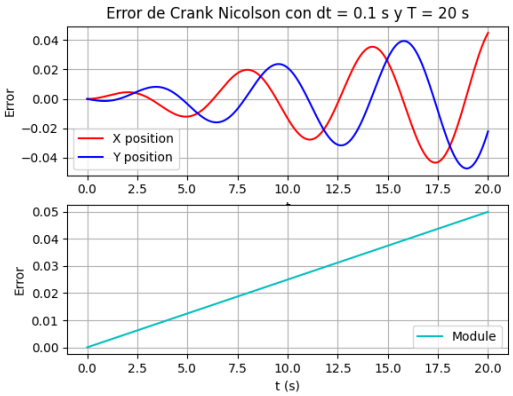
(c) Resultados obtenidos para un $dt = 0{,}001s$

Figura 5.1: Error obtenido con el esquema numérico de Euler

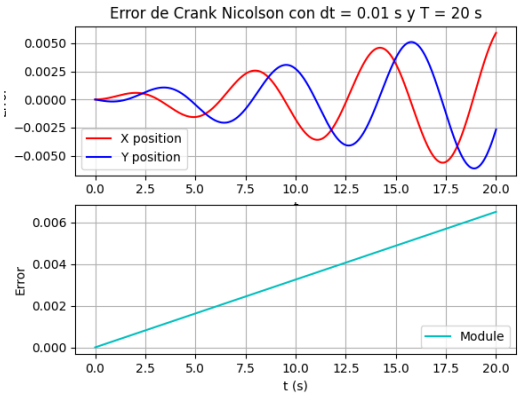
Como se puede observar en la figura 5.1, el error obtenido por el esquema numérico de Euler irá disminuyendo a medida que se va utilizando un paso de integración más pequeño. También cabe destacar la forma del módulo del error, que no es una recta sino una curva.

5.1.2. Crank Nicolson

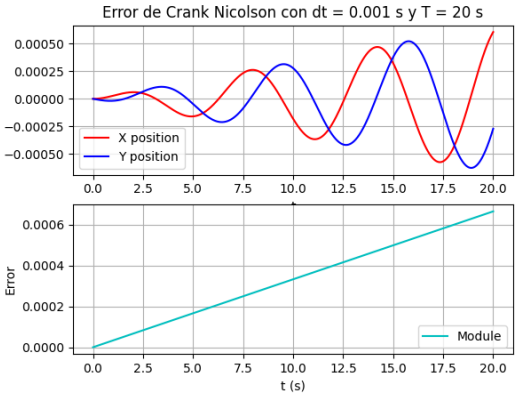
Los resultados obtenidos por el esquema numérico de Crank-Nicolson se presentan en la figura 5.2.



(a) Resultados obtenidos para un $dt = 0{,}1s$



(b) Resultados obtenidos para un $dt = 0{,}01s$



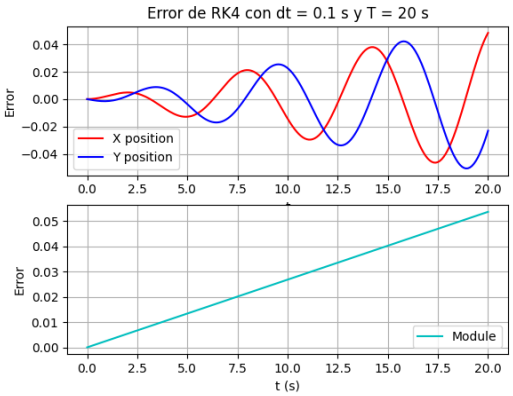
(c) Resultados obtenidos para un $dt = 0{,}001s$

Figura 5.2: Error obtenido con el esquema numérico de Crank Nicolson

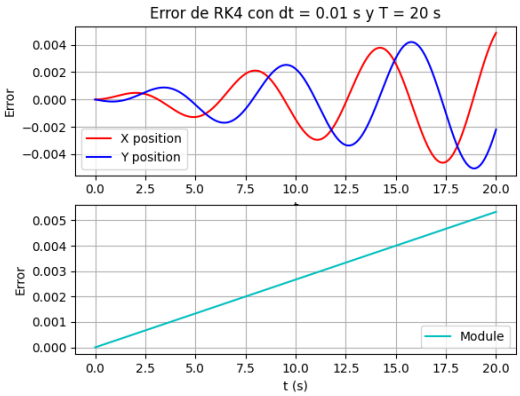
De la misma manera que ocurría con las órbitas integradas en el Hito 2, se puede apreciar que el resultado obtenido con este esquema numérico es bastante mejor que el obtenido con Euler, lo cual se manifiesta orden de magnitud del error más pequeño. Por otro lado, la variación del valor de paso de integración hace que el error vaya disminuyendo.

5.1.3. Runge Kutta de cuarto orden

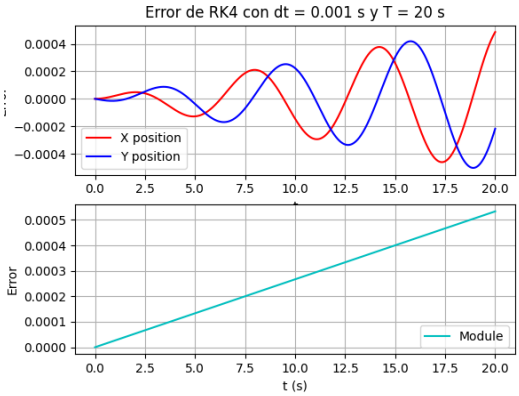
Los resultados obtenidos por el esquema numérico de Runge-Kutta de cuarto orden se presentan en la figura 5.3.



(a) Resultados obtenidos para un $dt = 0{,}1s$



(b) Resultados obtenidos para un $dt = 0{,}01s$



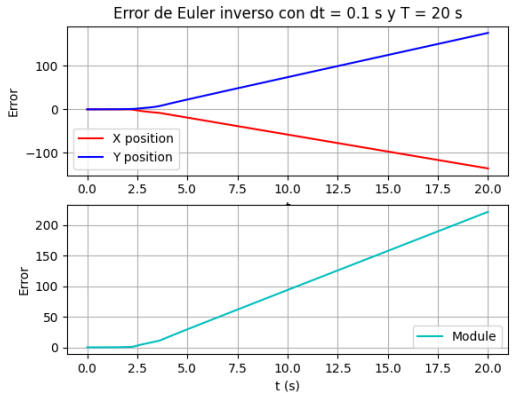
(c) Resultados obtenidos para un $dt = 0{,}001s$

Figura 5.3: Error obtenido con el esquema numérico de Runge Kutta de cuarto orden

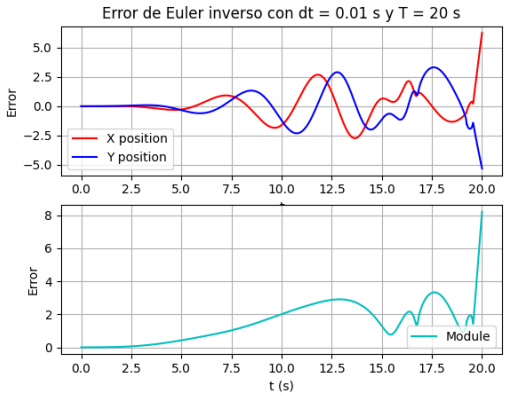
De la misma manera que con Crank-Nicolson, se puede observar que este esquema numérico es bastante mejor que el Euler, esto se debe a que es de cuatro etapas y de cuarto orden y por lo tanto mayor cuenta con precisión. La variación del valor de paso de integración produce una disminución del error al igual que en los casos anteriores.

5.1.4. Euler inverso

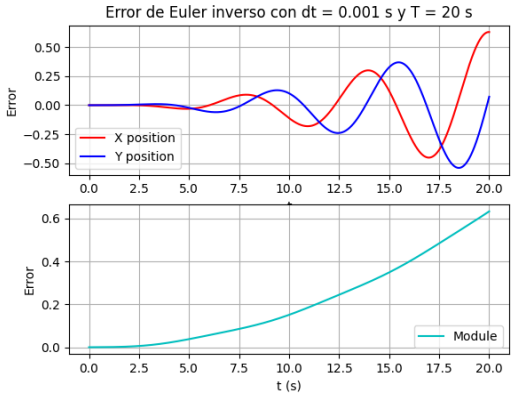
Los resultados obtenidos por el esquema numérico de Euler inverso se presentan en la figura 5.4.



(a) Resultados obtenidos para un $dt = 0{,}1s$



(b) Resultados obtenidos para un $dt = 0{,}01s$



(c) Resultados obtenidos para un $dt = 0{,}001s$

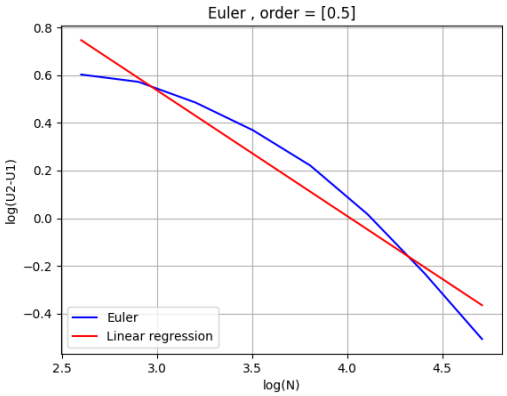
Figura 5.4: Error obtenido con el esquema numérico de Euler inverso

El esquema numérico del Euler inverso proporciona los resultados más peculiares de todos los métodos numéricos analizados. Esto podría ser achacado a la implementación propia del método de Newton para resolver esquemas implícitos junto al bajo orden del esquema numérico. Si bien es cierto que para valores del paso de integración altos ($dt = 0{,}1s$) el orden de magnitud del error es gigantesco, este se va asemejando más a los órdenes obtenidos con el Euler a medida que se va disminuyendo el paso de integración.

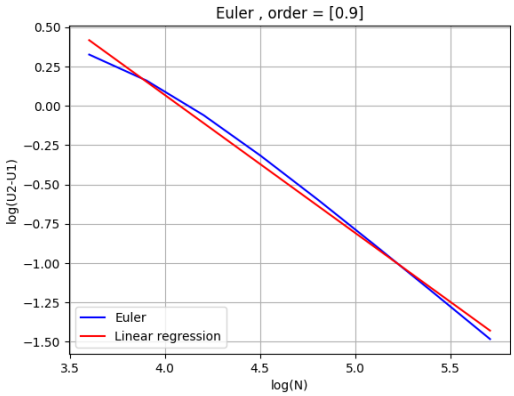
5.2. Ratio de convergencia de los distintos esquemas numéricos

Por último, se muestran las gráficas obtenidas del ratio de convergencia para los distintos esquemas numéricos. En ellas están incluidas dos líneas: una que está formada a partir de los puntos obtenidos con la impementación del código mostrado anteriormente y otra que será la regresión lineal de estos puntos, a partir de la cual se obtendrá el orden del esquema numérico, siendo este la pendiente de esta última recta.

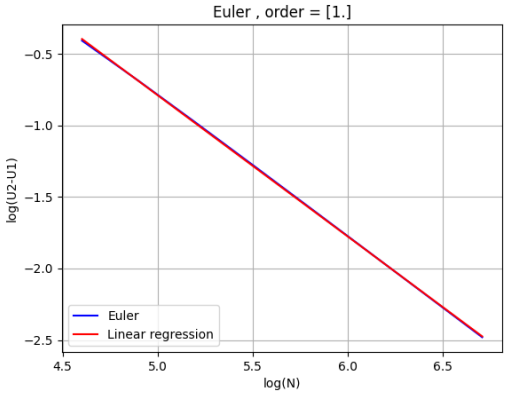
5.2.1. Euler



(a) Resultados obtenidos para un $dt = 0{,}1s$



(b) Resultados obtenidos para un $dt = 0{,}01s$

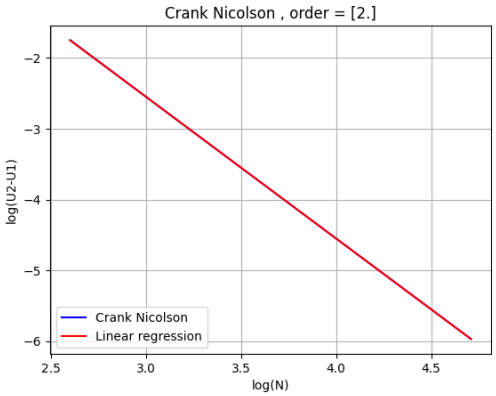


(c) Resultados obtenidos para un $dt = 0{,}001s$

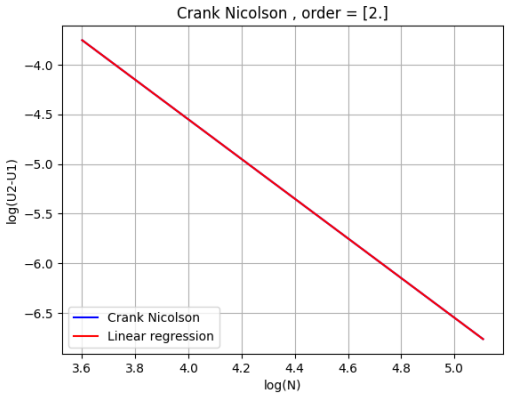
Figura 5.5: Ratio de convergencia obtenido para Euler

Como se puede comprobar en 5.5, a medida que se va disminuyendo el dt los puntos obtenidos forman más claramente una recta. El orden del esquema numérico se ha obtenido satisfactoriamente a partir de los valores de dt de $0{,}01s$ y $0{,}001s$.

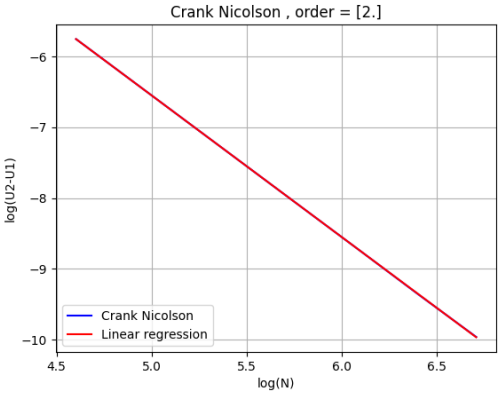
5.2.2. Crank Nicolson



(a) Resultados obtenidos para un $dt = 0{,}1s$



(b) Resultados obtenidos para un $dt = 0{,}01s$

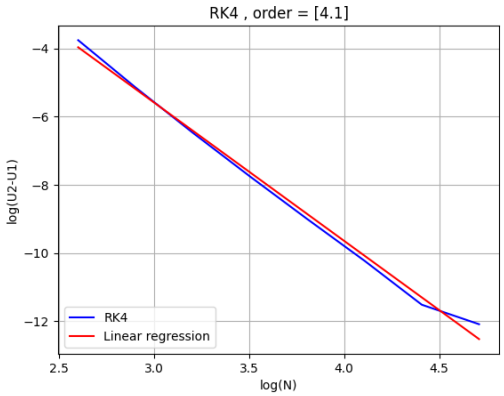


(c) Resultados obtenidos para un $dt = 0{,}001s$

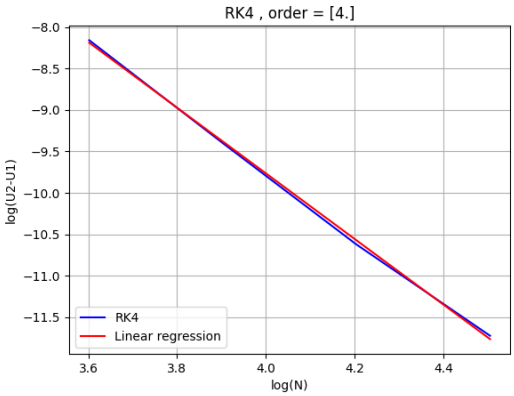
Figura 5.6: Ratio de convergencia obtenido para Crank Nicolson

En el caso de Crank Nicolson para todos los valores propuestos de dt los puntos obtenidos forman una recta que coincide con su regresión lineal, obteniéndose en todos los casos el valor del orden del esquema numérico con notable exactitud. Esto puede ser debido a las regiones de estabilidad absoluta de este método, las cuales se tratarán en el siguiente hito.

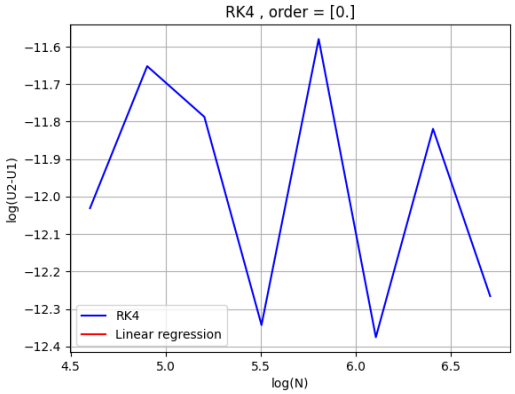
5.2.3. Runge Kutta de cuarto orden



(a) Resultados obtenidos para un $dt = 0{,}1s$



(b) Resultados obtenidos para un $dt = 0{,}01s$

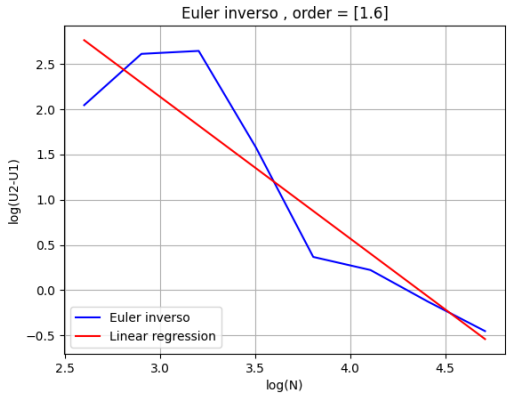


(c) Resultados obtenidos para un $dt = 0{,}001s$

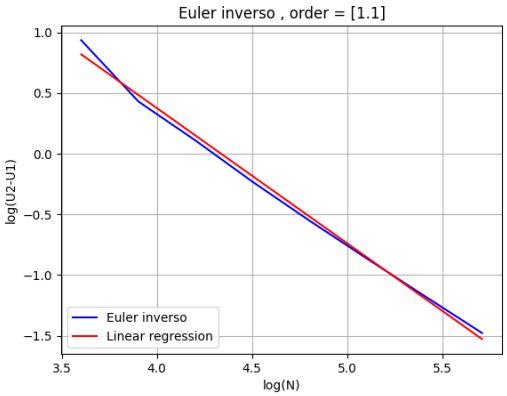
Figura 5.7: Ratio de convergencia obtenido para Runge Kutta de cuarto orden

Como se puede observar en 5.7, los resultados obtenidos para Runge Kutta de cuarto orden son satisfactorios, incluyendo el que a simple vista puede parecer una gráfica errónea. Este último caso de $dt = 0{,}001s$ está caracterizado por el hecho de que el error obtenido es tan pequeño que se acerca bastante al error de redondeo de la máquina.

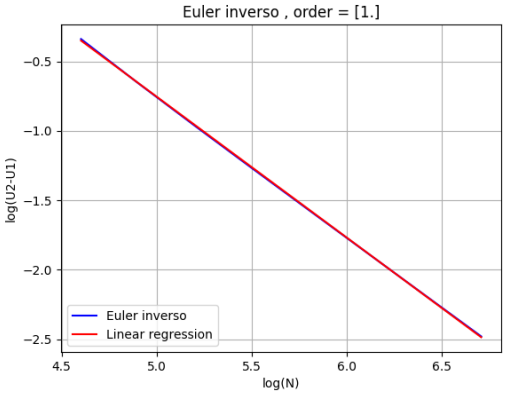
5.2.4. Euler inverso



(a) Resultados obtenidos para un $dt = 0{,}1s$



(b) Resultados obtenidos para un $dt = 0{,}01s$



(c) Resultados obtenidos para un $dt = 0{,}001s$

Figura 5.8: Ratio de convergencia obtenido para Euler inverso

Por último, para el caso del Euler inverso se puede comentar que a medida que se disminuye el dt la convergencia es mucho mejor.