


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone III Report

Ampliación de Matemáticas I

18 de octubre de 2022

Autora:

- Rosa Martínez Rubiella

1. Introducción

El objetivo del Hito III es la estimación del error de las soluciones numéricas obtenidas en hitos anteriores. Para esto, se implementa un método de extrapolación de **Richardson** para la evaluación del error en el problema de Cauchy, y se utiliza en los diferentes esquemas temporales usados. A continuación, se programa una función que evalúa el **ratio de convergencia** de dichos esquemas temporales. De esta forma se puede obtener el orden de integración de los mismos mediante una regresión lineal, como se verá a continuación.

Finalmente, se explican de manera breve las actualizaciones en la estructura y contenido del código, así como el razonamiento seguido en las nuevas implementaciones.

2. Extrapolación de Richardson

Este método permite evaluar el error que se produce al utilizar un determinado esquema numérico para la resolución de un problema de Cauchy. Para ello, se calcula la solución aproximada en dos mallas temporales diferentes (dentro del mismo intervalo), y se restan en sus puntos coincidentes. En este caso se eligen mallas con un N y 2N puntos de integración, que hace más fácil la evaluación de la solución en dichos puntos. La fórmula resultante queda:

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

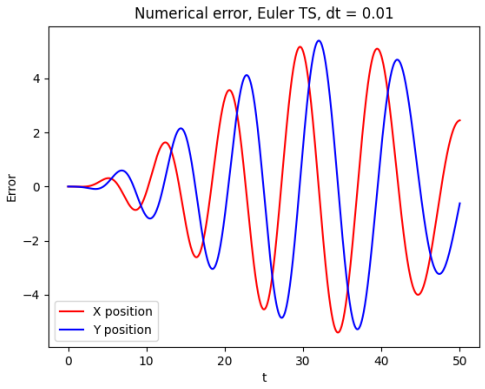
donde q corresponde al orden de integración del esquema utilizado. Dicho orden puede ser obtenido mediante la regresión lineal que posteriormente se muestra o de manera teórica ($q_{Euler} = 1$, $q_{RK-4} = 4$ ... etc). Se realiza una iteración para todos los puntos coincidentes de la malla (N puntos).

Para ejemplificar esto, se realiza una simulación de 50 segundos de duración, con un intervalo de integración de $\mathbf{dt = 0.01}$. Se representa en la **Figura 1** el error en ambas coordenadas de posición para cada esquema temporal. Como se esperaba, tanto el Runge-Kutta-4 como el Crank-Nicolson presentan un error relativamente pequeño, del orden de la centésima aproximadamente. Sin embargo, el Euler presenta errores de orden unidad, mientras que el Euler Inverso diverge totalmente ya que necesita más puntos de integración como se vio en el informe anterior.

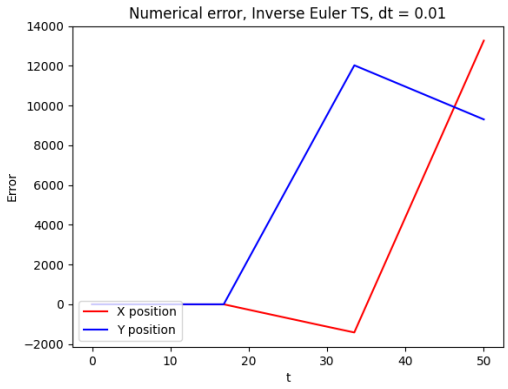
Por otro lado, si en vez de utilizar un método de Newton-Raphson para resolver la ecuación implícita del esquema Euler Inverso, se utiliza la función **fsolve** de Scipy, el error está más acotado. En la **Figura 2** se representa este fenómeno, para diferentes pasos de tiempo.

Por otro lado, se simula un tiempo final de 10 segundos con 10000 puntos de integración, es decir, se refina la malla con un dt = 0.001 para observar resultados más aproximados a la solución analítica. Además se ha añadido el módulo del error de posición para tener una mejor idea de sus dimensiones (**Figura 3**). En la **Figura 4** se representa la comparación del

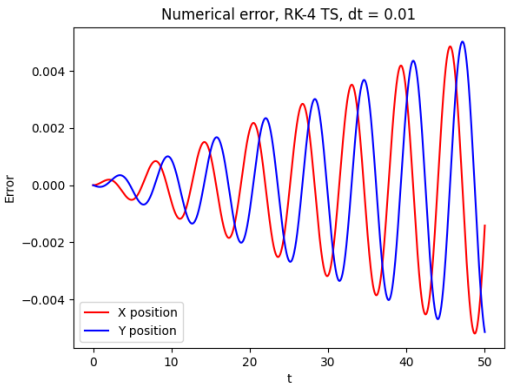
módulo del error de posición para los diferentes esquemas temporales con un dt de 0.001 s, observando así la gran diferencia entre los dos primeros esquemas .[Eu]lerz los de orden superior. Se ha reducido el tiempo final de simulación manteniendo el paso de integración para observar mejor la escala de los errores en el caso del Crank-Nicolson y el Runge-Kutta-4.



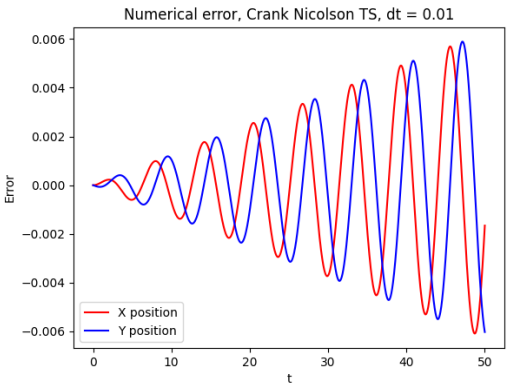
(a) Euler



(b) Inverse Euler

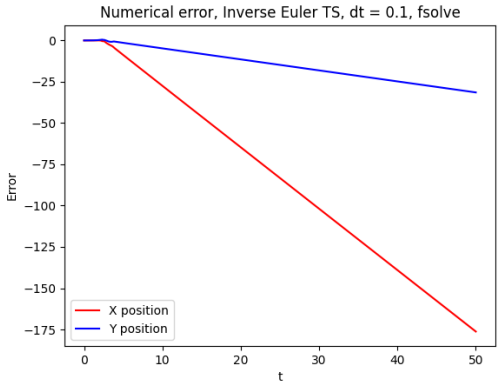


(c) Runge-Kutta-4

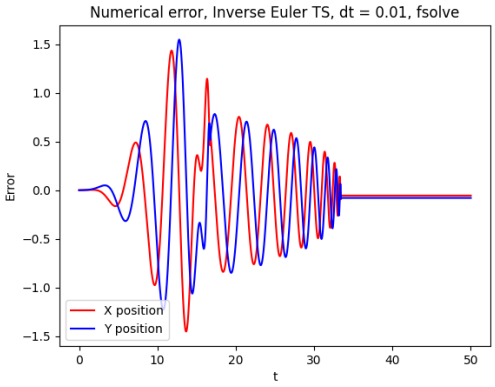


(d) Crank-Nicolson

Figura 1: Richardson extrapolation, Kepler problem, dt = 0.01

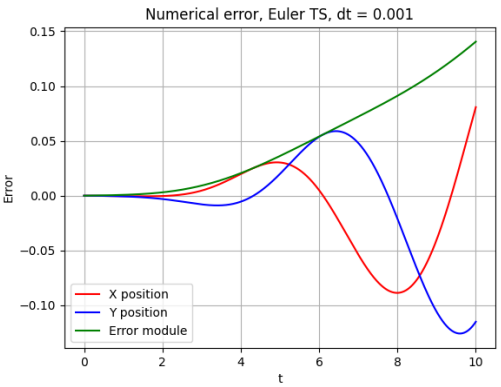


(a) dt = 0.1

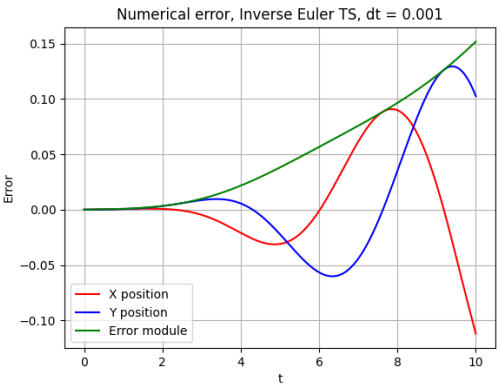


(b) dt = 0.01

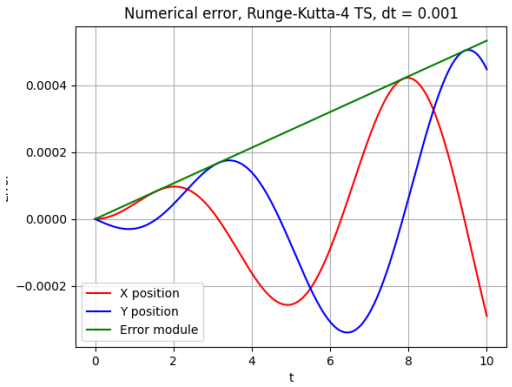
Figura 2: Richardson extrapolation, Inverse Euler, fsolve resolution



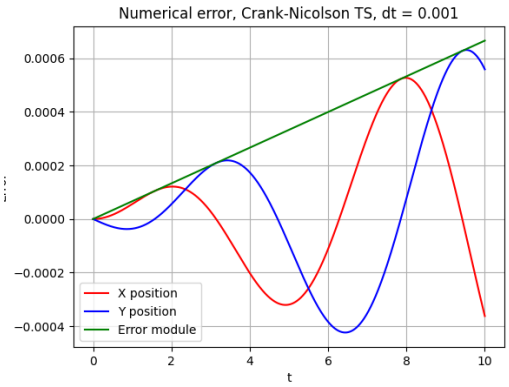
(a) Euler



(b) Inverse Euler



(c) Runge-Kutta-4



(d) Crank-Nicolson

Figura 3: Richardson extrapolation, Kepler problem, dt = 0.001

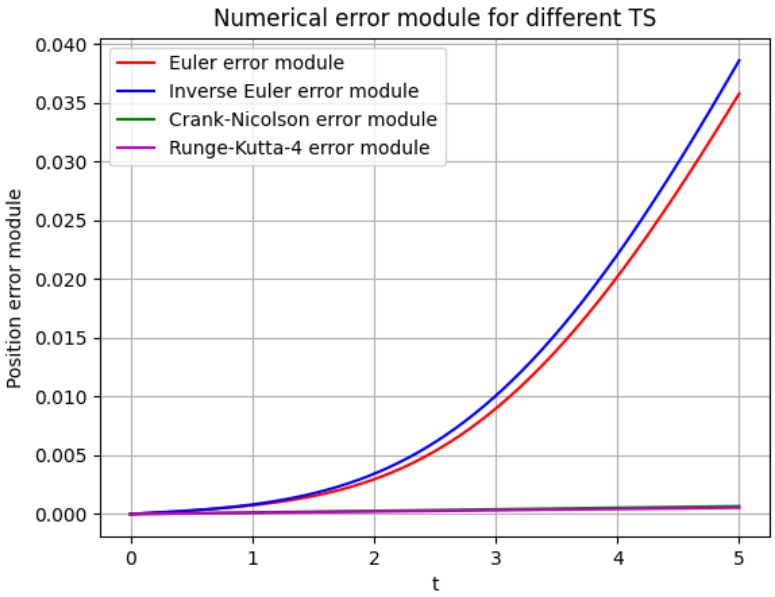
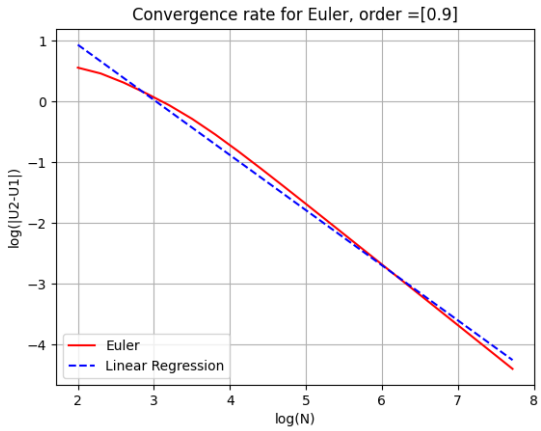


Figura 4: Richardson extrapolation, comparison between TS for dt = 0.001

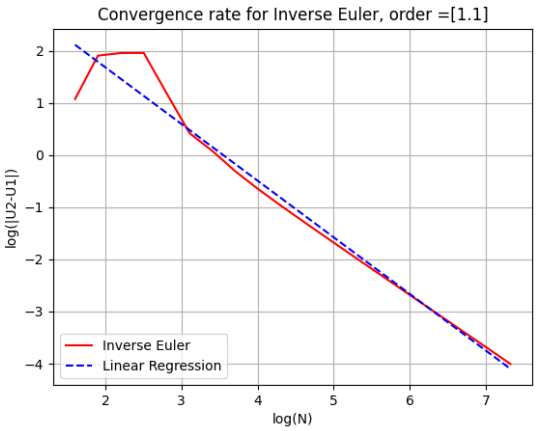
3. Ratio de convergencia

El ratio de convergencia nos permite obtener el orden de convergencia o integración de un esquema numérico. Esto es, a mayor orden, menor número de iteraciones serán necesarias para obtener una solución suficientemente precisa. Se realiza una regresión lineal de la gráfica que representa $\log(||U^{2N} - U^N||)$ (el error), para diferentes números de puntos de integración ($\log(N)$). La pendiente de dicha recta se corresponderá con el orden del esquema temporal. En la **Figura 5** se ha representado esto en los cuatro casos, así como su regresión lineal y el orden obtenido. Este orden se acerca mucho al real en el caso del Runge-Kutta-4 y el Crank-Nicolson para un número no muy grande de puntos. Sin embargo, si queremos que la precisión de aproximación del error sea buena para el Euler y el Euler Inverso, hay que aumentar mucho las iteraciones. El tiempo de compilación es bastante grande, ya que debe evaluar el problema para un número de puntos de integración de orden aproximado de 10^7 en los primeros casos.

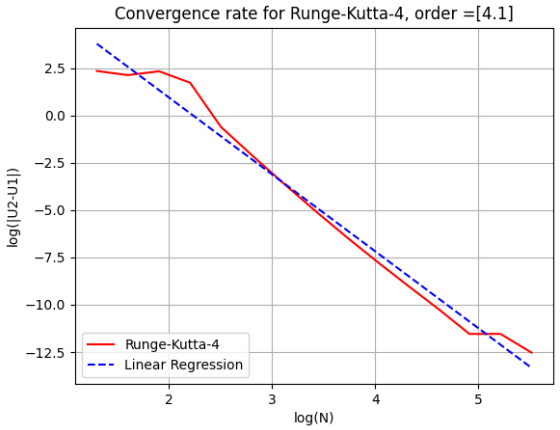
Cabe destacar que errores menores a un orden de 10^{-12} se corresponden a la precisión del propio ordenador, mientras que los errores para muy pocos puntos de integración derivan del propio método de integración. Estos puntos erráticos no son utilizados para la regresión lineal, ya que solo interesa la parte más lineal de la curva para hallar su pendiente.



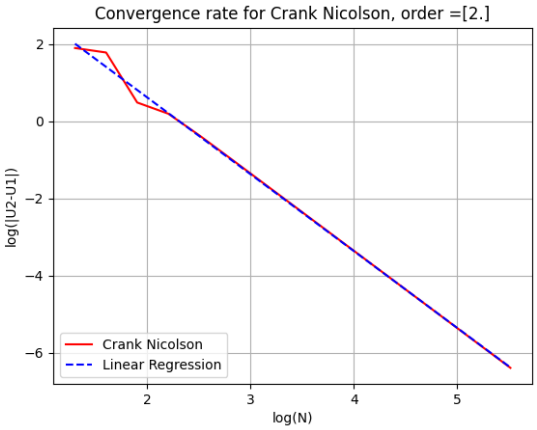
(a) Euler



(b) Inverse Euler



(c) Runge-Kutta-4



(d) Crank-Nicolson

Figura 5: Convergence rate, Kepler problem

4. Code Review

Primero, se ha implementado una factorización LU de matrices, mediante la cual se puede calcular su inversa. Esto se utiliza para realizar la inversa del Jacobiano en el método de Newton-Raphson (anteriormente implementado), así no se depende de otras bibliotecas.

```
def factorization_LU(A): #A es la matriz que queremos factorizar

  N = size(A,1)
  U = zeros([N,N])
  L = zeros([N,N])

  U[0,:] = A[0,:] #Se inicializa la U

  for k in range(0,N): #Se inicializa la L
    L[k,k] = 1
  L[1:N,0] = A[1:N,0]/U[0,0]

```

```
  for k in range(1,N): #Iteracion de las matrices diagonales, por
   eliminacion Gaussiana

    for j in range(k,N):
      U[k,j] = A[k,j] - dot(L[k,0:k], U[0:k,j]) #Matriz diagonal superior

    for i in range(k+1,N):
      L[i,k] =(A[i,k] - dot(U[0:k,k], L[i,0:k])) / (U[k,k]) #Matriz
   diagonal inferior

  return [L@U, L, U] #L*U es la matriz A factorizada


def solve_LU(M,b): #Resuelve el sistema de ecuaciones que resulta de la
   factorizacion LU, M es la matriz de coef. constantes y b el vector de
   terminos independientes

  N=size(b)
  y=zeros(N)
  x=zeros(N)

  [A,L,U] = factorization_LU(M) #Primero se factoriza la matriz
  y[0] = b[0]

  for i in range(0,N):
    y[i] = b[i] - dot(A[i,0:i], y[0:i])

  x[N-1] = y[N-1]/A[N-1,N-1]

  for i in range(N-2,-1,-1):
    x[i] = (y[i] - dot(A[i, i+1:N+1], x[i+1:N+1])) / A[i,i] #x es el
   vector solucion del sistema

  return x


def Inverse(A): #Realiza la inversa de la matriz

  N = size(A,1)
  B = zeros([N,N])

  for i in range(0,N):
    one = zeros(N)
    one[i] = 1

    B[:,i] = solve_LU(A, one) #Basicamente resuelve [A|I] por
   eliminacion Gaussiana como se hace a mano, en forma de sistema

  return B
```

Listing 1: LU factorization and Inverse

Por otro lado, se ha implementado dentro del módulo del problema de Cauchy tanto el método de estimación del error de Richardson como el ratio de convergencia.

```

```

```
def Error_Cauchy(F, t, U_0, Temporal_Scheme, order): #Calcula el error de Richardson, se introduce el problema de Cauchy y el orden del esquema temporal, donde se puede meter la solucion obtenida por la regresion lineal del ratio de convergencia

    N = size(t)
    E = zeros([N,size(U_0)])

    t1 = t
    t2 = linspace(0, t[N-1], N*2) #Malla refinada

    U2N = Cauchy_Problem(F, t2, U_0, Temporal_Scheme) #Solucion en la malla refinada
    U1N = Cauchy_Problem(F, t1, U_0, Temporal_Scheme) #Solucion en la malla inicial

    for i in range(0,N):
        E[i,:] = (U2N[2*i,:] - U1N[i,:]) / (1 - 1 / (2**order)) #Formula del error

    return E


def Convergence_rate(F, t, U_0, Temporal_Scheme): #Ratio de convergencia, se introduce el problema de Cauchy

    N = size(t)

    t1 = t
    tf = t1[N-1] #Tiempo final de la simulacion
    U1 = Cauchy_Problem(F, t1, U_0, Temporal_Scheme) #Solucion inicial del problema

    k = 20 #Numero de puntos donde se evalua el problema, en cada iteracion se duplican los puntos de integracion

    log_E = zeros(k)
    log_N = zeros(k)

    for i in range(0,k):

        N = 2*N #Se refina la malla
        t2 = linspace(0, tf, N) #Nueva division temporal

        U2 = Cauchy_Problem(F, t2, U_0, Temporal_Scheme) #Solucion en la malla refinada

        E = norm((U2[int(N-1),:] - U1[int(N/2-1),:])) #Norma de la diferencia entre las soluciones

        log_E[i] = log10(E)  #Escala logaritmica del error y los puntos de integracion usados
        log_N[i] = log10(N)

        t1 = t2  #Se inicializan las variables para la nueva iteracion
        U1 = U2
```

```
46
47      return [log_N, log_E]
48
49  def lineal_Convergence_rate(log_E, log_N): #Regresion lineal del ratio
        de convergencia
50
51      k = size(log_E)
52
53      for j in range(0,k): #Encuentra los valores para los que el error ya
        no sigue la progresion lineal, si no que entra en zonas donde se
        dispara, como por debajo de 10e-12
54          if (abs(log_E[j]) > 12): #Errores por precision del ordenador
55              break
56      j = min(j, k)
57      reg = LinearRegression().fit(log_N[0:j+1].reshape((-1,
        1)),log_E[0:j+1]) #Funcion de regresion lineal de sklearn
58      order = round_(abs(reg.coef_),1) #Calcula el orden, la pendiente de
        la recta
59
60      log_N_lineal = log_N[0:j+1]
61      log_E_lineal = reg.predict(log_N[0:j+1].reshape((-1, 1))) #Para
        poder graficar la regresion lineal
62
63      log_E_total = log_E - log10(1 - 1 / (2**order)) #Es el logaritmo del
        error de Richardson
64
65      return [log_N_lineal, log_E_lineal, order, log_E_total]
```

Listing 2: Richardson and Convergence Rate

Finalmente se ha creado un módulo desde donde se ejecuta el problema, de forma que solo es necesario llamar a la función del error o la convergencia para plotear las gráficas. Mediante esta estructura de módulos por funcionalidad se tiene el código mucho más escalonado, claro y compacto.