


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestones 3

Ampliación de Matemáticas 1

25 de octubre de 2022

Autor:

- Javier Zatón Miguel

Índice

1. Objetivo **1**

2. Desarrollo del código **1**

2.1. Función Error_R 1

2.2. Convergencer_R 2

3. Resultados **4**

3.1. Error de cada esquema 4

3.2. Convergencia del error 5

1. Objetivo

El objetivo de este hito es de crear dos nuevas funciones, una que calcule el error cometido por un esquema numérico por extrapolación de Richardson, y otro que represente el ratio de convergencia del error cometido,

2. Desarrollo del código

Para este ejercicio se ha desarrollado dos funciones, Error_R y Convergence_R.

2.1. Función Error_R

Esta función tiene cuatro inputs, los mismos inputs para el problema de cauchy:

Función (en este caso usaremos Kepler)

Condiciones iniciales (U0), este vector se define en el programa principal (Milestones_3.py) Vector espacial (puntos de t donde se resuelve)

Esquema numérico

El error se calcula como $E = ((U2n - Un)/(1 - 1/2^q))$

Esto se consigue creando dos *linspace* distintos, linspace_1, siendo este el original, que corresponde a uno de los inputs de la función. Después está el linspace_2, que se crea a partir del primer vector, duplicando el numero de términos.

La diferencia de Un y U2n se realiza en un bucle de 0 a N, en el caso de U2n, se utilizan los términos pares (ya que estos corresponden al instante del término Un para un paso del bucle).

El output de la función es la matriz E, para calcular los términos de esta matriz, a la diferencia de un y U2n se divide por un termino que depende del parámetro q, que depende a la vez del esquema analizado.

q = 1 para Euler y Euler inverso.

q = 2 para esquema de Crank Nicolson

q = 4 para Runge Kutta de orden 4

```
def Error_R(Function,U0,linspace_t,scheme):
    N = size(linspace_t)
    linspace_t2 = linspace(0,linspace_t[N-1],2*N-1)

    if scheme == RK4:
        q = 4
    elif scheme == Crank_Nicolson:
        q = 2
    else:
        q = 1

    E = zeros([size(U0),N])
    U2n = Cauchy(Function, U0, linspace_t2, scheme)
    U1n = Cauchy(Function, U0, linspace_t, scheme)

    for i in range (0,N):
        E[:,i]=((U2n[:,2*i]-U1n[:,i])/(1-1/(2**q)))

    return E
```

2.2. Convergencer_R

Para calcular la convergencia del error se crea una función distinta denominada Convergence_R , en esta función se tiene que introducir los mismos input que en la anterior.

Para la convergencia del error, se va a representar en una escala logarítmica, el error para distintos mallados de tamaño N duplicando su tamaño para cada iteración.

El numero de puntos de la gráfica es muy bajo debido a que el programa tiene que realizar una gran cantidad de operaciones, ya que cada proceso del bucle, aumenta el parámetro Nt2, que aumenta ambos mallados.

Primero la función solo calcula la diferencia de U2n y Un en el tiempo final del , para esto se especifica la posición [-1], si [0] corresponde a la columna primera, los números negativos retroceden al valor final. Esta operación se repite para 6 puntos.

Los resultados producen una recta decreciente (en escala logarítmica), con esto se calcula la pendiente, que equivale al orden del esquema q. el q debería de dar cerca del valor teórico de cada esquema detallado en la función Error_R.

Con este valor q se calcula la constante de la función lineal con forma: lg10(E) = lg10(N) - log10(1-1/2**q), reajustándose la recta en el eje y , dando como resultado la recta de convergencia del error de richardson.

La pendiente q se calcula tomando los valores en x e y de dos puntos, debido a que son pocos puntos y los valores iniciales como finales pueden no ser lineales, se coge la diferencia de los puntos centrales (punto 3 y 4).

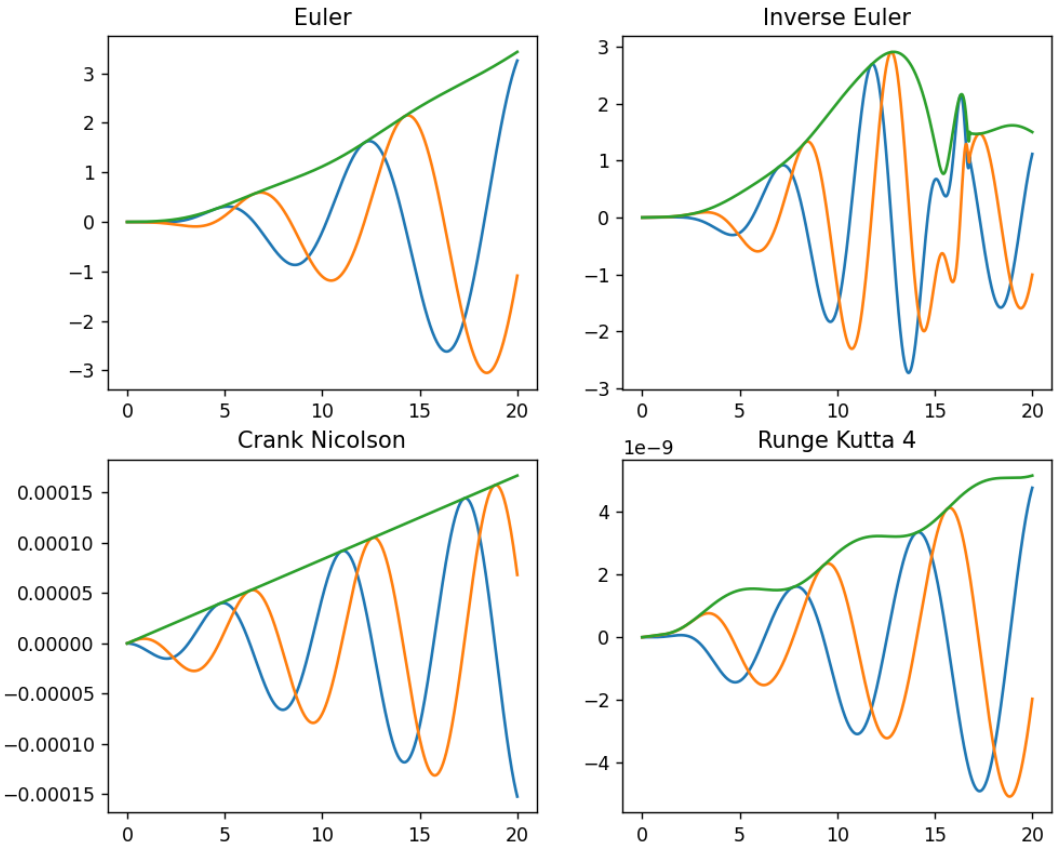
Para poder representar las gráficas ,la función extrae los valores del logaritmo en base diez del error (eje y), del tamaño de N utilizado (eje x) y el q calculado

```
def Convergence_R(Function,U0,linspace_t,scheme):

    p = 6
    Nt2 = size(linspace_t)
    U1= Cauchy(Function, U0, linspace_t, scheme)

    log_E = zeros(p)
    log_N = zeros(p)
    log_Eq = zeros(p)

    U2 = zeros( shape(U1) )

    for i in range(p):
        print(i)
        Nt2 = 2*Nt2
        linspace_t2 = linspace(0, linspace_t[-1],Nt2)
        U2 = Cauchy(Function, U0, linspace_t2, scheme)
        log_E[i] = log10(linalg.norm(  U2[:, -1]- U1[:,-1] ) )
        log_N[i] = log10(Nt2)
        linspace_t = linspace_t2
        U1 = U2

    q = -((log_E[3]-log_E[2])/(log_N[3]-log_N[2]))
    cq = log10(1-(1/(2**q)))

    cq_vector = array([cq,cq,cq,cq,cq,cq,cq,cq,cq,cq])
    for i in range(p):

        log_Eq[i] = log_E[i] - cq_vector[i]

    return log_Eq, log_N ,q
```

3. Resultados

3.1. Error de cada esquema

En el programa principal se ha escrito un código para obtener las cuatro gráficas del error, desde t=0 hasta t=tf=20 , se representa el error de posición x e y, y el error de radio orbital absoluto.



En la figura superior se puede observar para los cuatro esquemas, tres funciones distintas.

En azul , el error de la posición en X a lo largo del tiempo. De la misma forma, se obtiene el error de la posición en y.

Finalmente, en verde se puede observar el error de la distancia del radio orbital (en valor absoluto).

Como era de esperar, el error cometido por Crank Nicolson y RK4 es mucho menor que en el resto de esquemas, Euler inverso se incrementa muy deprisa para luego volver a reducirse, esto posiblemente sea porque el radio se reduce, ya que la orbita es mas pequeña, y a un cierto t el radio de la orbita sea igual a cero.

3.2. Convergencia del error

Para la convergencia del error, se llama a la función Convergence_R y con los output es posible expresar las cuatro funciones casi-lineales, además de los 4 valores de q.

Los resultados de q son los siguientes:

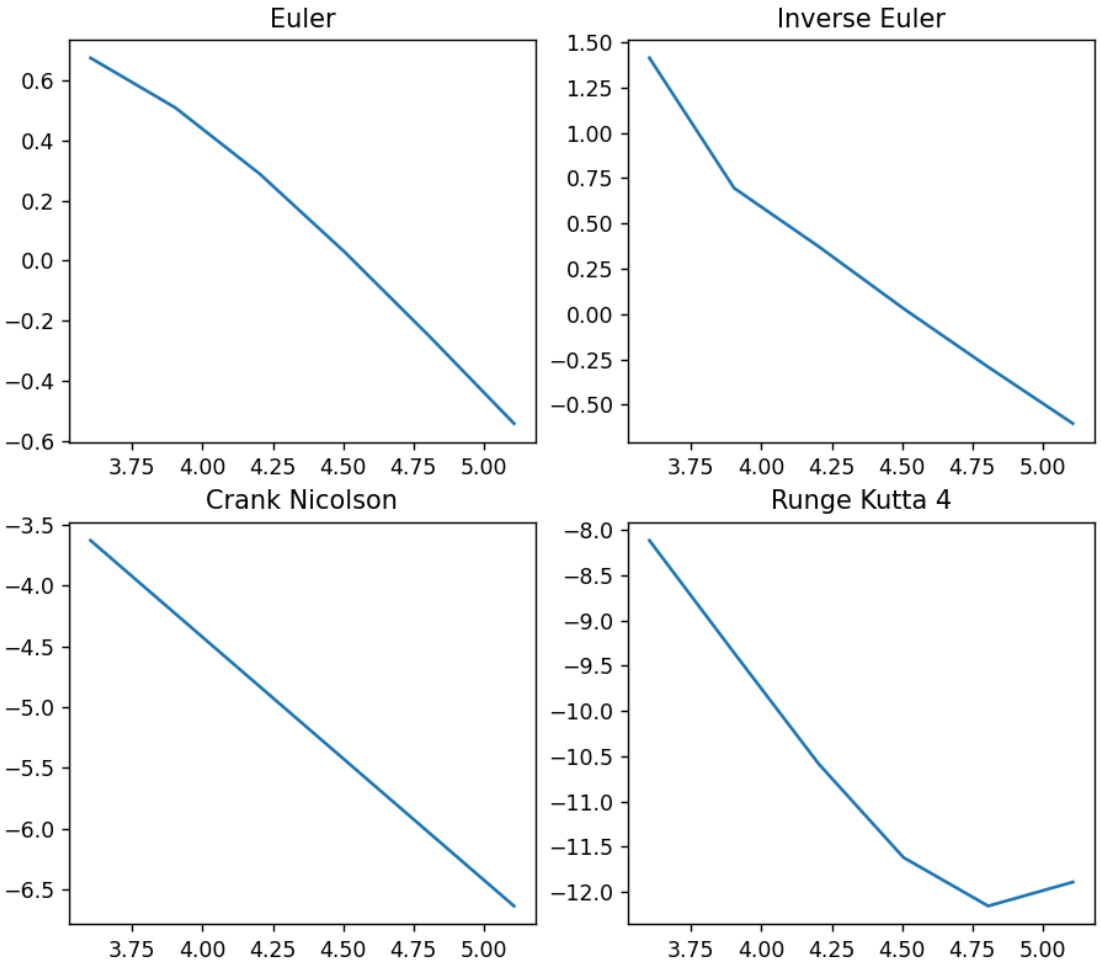
Euler: q = 0.861

Euler inverso: q = 1.124

Crank Nicolson: q = 2.001

RK4: q = 3.431

Estos resultados no son exactos, ya que el resultado se ha tomado como la distancia de dos puntos muy próximos, ya que los primeros y últimos puntos pueden añadir mas error a este cálculo.



En las gráficas se puede ver que el caso de RK4 es el primero en llegar al final de la zona lineal ya que al tener una q mayor, desciende mas rápido al error mínimo (cerca de 10^-15). Pasado este punto, la gráfica alcanza un mínimo.

En los dos esquemas de Euler , tiene un cambio de pendiente en la zona inicial, en cambio Crank Nicolson no, esto hace que el calculo de q previo sea tan preciso.