


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Máster Universitario en Sistemas Espaciales

Milestone 1

Ampliación de Matemáticas I

24 de septiembre de 2022

Autor:

- Sergio Cavia Fraile

Índice

1. Introducción

El objetivo del trabajo será integrar órbitas de Kepler mediante distintos métodos numéricos utilizando el lenguaje Python.

2. Ecuaciones y Métodos

Las órbitas de Kepler se definen mediante el siguiente problema de Cauchy:

$\ddot{r} = \frac{\vec{r}}{|\vec{r}|^3} = F(U,t)$

$\vec{r}(0) = (1,0)$

$\dot{\vec{r}}(0) = (0,1)$

2.1. Método de Euler

El método de Euler se describe de la siguiente forma:

$U^1 = U^0 + \Delta t * F^0$

2.2. Método de Runge-Kutta de 4°orden

El método de Runge-Kutta de 4°orden se describe de la siguiente forma:

$U^1 = U^0 + \frac{\Delta t}{6} * (k_1 + k_2 + k_3 + k_4)$

Donde k_1, k_2, k_3, k_4 son:

$k_1 = f(x_1, y_1)$

$k_2 = f(x_1 + \frac{\Delta t}{2}, y_1 + \frac{k_1 * \Delta t}{2})$

$k_3 = f(x_1 + \frac{\Delta t}{2}, y_1 + \frac{k_2 * \Delta t}{2})$

$k_4 = f(x_1 + \Delta t, y_1 + k_3 * \Delta t)$

2.3. Método de Crank-Nicholson

El método de Crank-Nicholson se describe de la siguiente forma:

$U^1 = U^0 + \frac{\Delta t}{2} * (F^0 + F^1)$

3. Código

Lo primero es definir cuál es el intervalo temporal en el que se integrará y qué paso temporal se usará en los esquemas numéricos.

```
"Nº pasos"
n = 200
"Tiempo final"
tf = 20
dt = tf/n
"C.I."
U0 = array([1,0,0,1])
```

3.1. Método de Euler

El método de Euler se ha implementado mediante las siguientes líneas de código:

```
"EULER"
U = zeros((4,n))
U[:,0] = U0
F = zeros(4)
for i in range(1, n):
    F = array([U[2,i-1],
               U[3,i-1],
               -U[0,i-1]/(U[0,i-1]**2+U[1,i-1]**2)**(3/2),
               -U[1,i-1]/(U[0,i-1]**2+U[1,i-1]**2)**(3/2)])
    U[:,i] = U[:,i-1] + dt*F

plt.figure(1)
plt.plot(transpose(U[0,:]),transpose(U[1,:]))
plt.show()
```

3.2. Método de Runge-Kutta de 4°orden

El método de Runge-Kutta de 4°orden se ha implementado mediante las siguientes líneas de código:

```
"RK4"
U = zeros((4,n))
U[:,0] = U0
k1 = zeros(4)
k2 = zeros(4)
k3 = zeros(4)
k4 = zeros(4)
Unew1 = zeros(4)
Unew2 = zeros(4)
Unew3 = zeros(4)

for i in range(1, n):
    k1 = array([U[2,i-1],
                U[3,i-1],
                -U[0,i-1]/(U[0,i-1]**2+U[1,i-1]**2)**(3/2),
                -U[1,i-1]/(U[0,i-1]**2+U[1,i-1]**2)**(3/2)])
    Unew1 = U[:,i-1] + dt*k1[:]/2
    k2 = array([Unew1[2],
                Unew1[3],
                -Unew1[0]/(Unew1[0]**2+Unew1[1]**2)**(3/2),
                -Unew1[1]/(Unew1[0]**2+Unew1[1]**2)**(3/2)])
    Unew2 = U[:,i-1] + dt*k2[:]/2
    k3 = array([Unew2[2],
                Unew2[3],
                -Unew2[0]/(Unew2[0]**2+Unew2[1]**2)**(3/2),
                -Unew2[1]/(Unew2[0]**2+Unew2[1]**2)**(3/2)])
    Unew3 = U[:,i-1] + dt*k3[:]
    k4 = array([Unew3[2],
                Unew3[3],
                -Unew3[0]/(Unew3[0]**2+Unew3[1]**2)**(3/2),
                -Unew3[1]/(Unew3[0]**2+Unew3[1]**2)**(3/2)])
    U[:,i] = U[:,i-1] + dt*(k1+2*k2+2*k3+k4)/6


plt.figure(2)
plt.plot(transpose(U[0,:]),transpose(U[1,:]))
plt.show()
```

3.3. Método de Crank-Nicholson

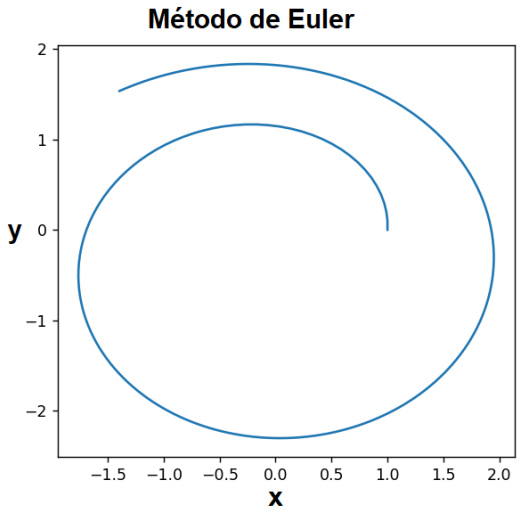
El método de Crank-Nicholson se ha implementado mediante las siguientes líneas de código:

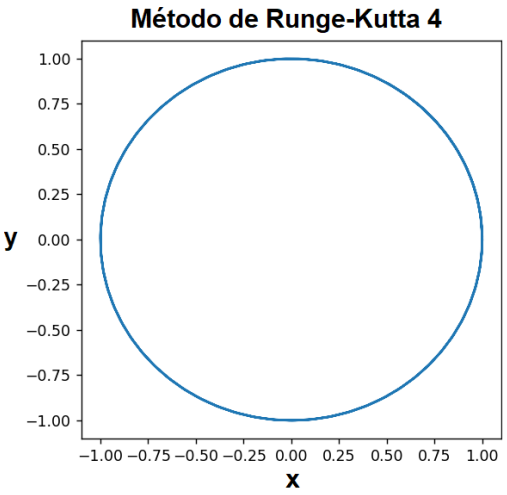
```
"CRANK-NICHOLSON"

U = zeros((4,n))
U[:,0] = U0
F = zeros(4)

for i in range(1,n):
    F = array([U[2,i-1],
               U[3,i-1],
               -U[0,i-1]/(U[0,i-1]**2+U[1,i-1]**2)**(3/2),
               -U[1,i-1]/(U[0,i-1]**2+U[1,i-1]**2)**(3/2)])
    def func_CN(x):
        return [x[0] - U[0,i-1] - (x[2] + F[0])*dt/2,
                x[1] - U[1,i-1] - (x[3] + F[1])*dt/2,
                x[2] - U[2,i-1] - (-x[0]/((x[0]**2+x[1]**2)**(3/2)) + F[2])*dt/2,
                x[3] - U[3,i-1] - (-x[1]/((x[0]**2+x[1]**2)**(3/2)) + F[3])*dt/2]
    U[:,i] = fsolve(func_CN, [U[:,i-1]])


plt.figure(3)
plt.plot(transpose(U[0,:]),transpose(U[1,:]))
plt.show()
```

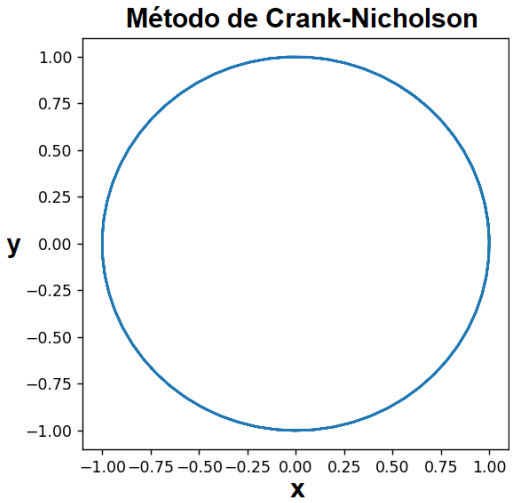
Como se puede observar en el código, en este caso se ha utilizado el comando fsolve para resolver Crank-Nicholson ya que el alumno no ha sido capaz de encontrar una forma de resolver el método implícito sin usar funciones.

4. Resultados

En las siguientes gráficas se pueden ver las órbitas que se obtienen integrando por cada uno de los métodos explicados durante 20 unidades de tiempo con un paso de 0.1.







Como se puede ver, la energía no se conserva en el método de Euler y el cuerpo va ganando energía con el tiempo dando como resultado una espira que se abre hacia fuera.

Utilizando los otros dos métodos no se ve diferencia a simple vista con una circunferencia de radio unidad, por lo que estos métodos tienen un error mucho inferior que el método de Euler.